

```
In [371]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import timeit
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import f_regression, mutual_info_regression
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor

pd.set_option('display.max_rows', None) # Muestra todas las filas
```

```
In [453]: datos1=pd.read_excel('datos1.xlsx')
datos2=pd.read_excel('datos2.xlsx')
datos3=pd.read_excel('datos3.xlsx')
datos4=pd.read_excel('datos4.xlsx')
datos5=pd.read_excel('datos5.xlsx')
datosVal=pd.read_excel('datosVal.xlsx')

datospre=pd.concat([datos1,
                    datos2,
                    datos3,
                    datos4,
                    datos5]).drop_duplicates('Matricula', keep = 'last').reset_index(['Versión','Distintivo Ambiental','Historial de revisiones','Estándar de calidad'])

datos = pd.concat([datospre,
                   datosVal]).drop_duplicates('Matricula', keep = 'first').reset_index(['Versión','Distintivo Ambiental','Historial de revisiones','Estándar de calidad'])

# datosVal = datos[datos['Validacion'] == 1]
# datos = datos[datos['Validacion'] != 1].drop('Validacion', axis = 1)
```

Las marcas Land Rover y Alfa Romeo no están bien codificadas ya que les falta la segunda palabra, que está incluida en el modelo. Se les añade a la marca y se eliminan del modelo

```
In [454]: mapeo_marcas = {'Alfa': 'Alfa Romeo', 'Land': 'Land Rover'}

def transformar_marca(marca):
    return mapeo_marcas.get(marca, marca)

datos['Marca'] = datos['Marca'].apply(transformar_marca)

def transformar_modelo(marca, modelo):
    if marca == 'Alfa Romeo':
        return modelo.split(' ', 1)[1] if modelo.startswith('Romeo') else modelo
    elif marca == 'Land Rover':
        return modelo.split(' ', 1)[1] if modelo.startswith('Rover') else modelo
    else:
        return modelo

datos['Modelo'] = datos.apply(lambda row: transformar_modelo(row['Marca'], row['Mode
```

Tras algunos análisis, debido a que algunas marcas tienen modelos de coches con nombre iguales a los de otras marcas (Seat León y Cupra León), el modelo se va a construir como Marca concatenado con Modelo

```
In [455]: datos['Modelo'] = datos['Marca'] + ' ' + datos['Modelo']
```

```
In [456]: datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3052 entries, 0 to 3051
Data columns (total 29 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Marca            3052 non-null    object  
 1   Modelo           3052 non-null    object  
 2   Precio           3052 non-null    int64  
 3   Primera matriculación 3052 non-null    object  
 4   Kilometraje     3052 non-null    object  
 5   Carburante      3052 non-null    object  
 6   Transmisión     3052 non-null    object  
 7   Potencia         3052 non-null    object  
 8   Tracción         3016 non-null    object  
 9   Tipo de vehículo 3052 non-null    object  
 10  Puertas          3050 non-null    float64 
 11  Número de asientos 3052 non-null    int64  
 12  Color            3051 non-null    object  
 13  Tapicería       3049 non-null    object  
 14  Tipo de ruedas  2794 non-null    object  
 15  Motor original   3052 non-null    object  
 16  Cilindrada       3052 non-null    object  
 17  Consumo          2759 non-null    object  
 18  Clase de eficiencia CO2 3052 non-null    object  
 19  Emisiones de CO2  2794 non-null    object  
 20  País de origen   3052 non-null    object  
 21  Número de llaves 3052 non-null    int64  
 22  Coche accidentado y reparado 2794 non-null    object  
 23  La última revisión se realizó el 2672 non-null    object  
 24  Tipo de IVA      3052 non-null    object  
 25  ITV válida hasta 2794 non-null    object  
 26  Matricula        3051 non-null    object  
 27  Número de inventario 3052 non-null    object  
 28  Validacion      3052 non-null    int64  
dtypes: float64(1), int64(4), object(24)
memory usage: 691.6+ KB
```

```
In [457]: datos.isnull().sum()
```

```
Out[457]: Marca          0
           Modelo         0
           Precio          0
           Primera matriculación 0
           Kilometraje      0
           Carburante        0
           Transmisión       0
           Potencia          0
           Tracción          36
           Tipo de vehículo 0
           Puertas           2
           Número de asientos 0
           Color              1
           Tapicería          3
           Tipo de ruedas     258
           Motor original      0
           Cilindrada          0
           Consumo            293
           Clase de eficiencia CO2 0
           Emisiones de CO2    258
           País de origen       0
           Número de llaves      0
           Coche accidentado y reparado 258
           La última revisión se realizó el 380
           Tipo de IVA          0
           ITV válida hasta     258
           Matrícula            1
           Número de inventario 0
           Validacion           0
           dtype: int64
```

```
In [458]: datos = datos.dropna()
           datos.reset_index(drop=True, inplace=True)
```

In [459]: `datos.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2430 entries, 0 to 2429
Data columns (total 29 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Marca            2430 non-null    object  
 1   Modelo           2430 non-null    object  
 2   Precio           2430 non-null    int64  
 3   Primera matriculación  2430 non-null    object  
 4   Kilometraje      2430 non-null    object  
 5   Carburante       2430 non-null    object  
 6   Transmisión      2430 non-null    object  
 7   Potencia          2430 non-null    object  
 8   Tracción          2430 non-null    object  
 9   Tipo de vehículo 2430 non-null    object  
 10  Puertas          2430 non-null    float64 
 11  Número de asientos 2430 non-null    int64  
 12  Color             2430 non-null    object  
 13  Tapicería        2430 non-null    object  
 14  Tipo de ruedas   2430 non-null    object  
 15  Motor original    2430 non-null    object  
 16  Cilindrada        2430 non-null    object  
 17  Consumo           2430 non-null    object  
 18  Clase de eficiencia CO2 2430 non-null    object  
 19  Emisiones de CO2  2430 non-null    object  
 20  País de origen    2430 non-null    object  
 21  Número de llaves   2430 non-null    int64  
 22  Coche accidentado y reparado 2430 non-null    object  
 23  La última revisión se realizó el 2430 non-null    object  
 24  Tipo de IVA       2430 non-null    object  
 25  ITV válida hasta  2430 non-null    object  
 26  Matricula          2430 non-null    object  
 27  Número de inventario 2430 non-null    object  
 28  Validacion         2430 non-null    int64  
dtypes: float64(1), int64(4), object(24)
memory usage: 550.7+ KB
```

In [460]:

```
#primer paso: modificar variables para poder tratarlas
#las fechas se van a convertir en fechas de pandas y luego en días hasta hoy, para q
datos['Primera matriculación'] = pd.to_datetime(datos['Primera matriculación'], format = '%d.%m.%Y')
datos['Días desde matriculación'] = (pd.Timestamp.now() - datos['Primera matriculación']).dt.days
datos['La última revisión se realizó el'] = pd.to_datetime(datos['La última revisión se realizó el'], format = '%d.%m.%Y')
datos['Días desde revisión'] = (pd.Timestamp.now() - datos['La última revisión se realizó el']).dt.days
datos['ITV válida hasta'] = pd.to_datetime(datos['ITV válida hasta'], format = '%d.%m.%Y')
datos['Días hasta ITV'] = (datos['ITV válida hasta'] - pd.Timestamp.now()).dt.days
```

```
In [461]: #las variables numéricas
datos['Kilometraje'] = datos['Kilometraje'].str.replace(' km', '').str.replace('.','')
datos['Potencia CV'] = datos['Potencia'].str.extract(r'(\d+) CV \/ (\d+) kW').iloc[:,0]
datos['Cilindrada'] = datos['Cilindrada'].str.replace(' ccm', '').astype(float)
datos['Emisiones de CO2'] = datos['Emisiones de CO2'].str.replace(' g/km', '').astype(float)
datos['Puertas'] = datos['Puertas'].astype(int)
datos['Precio'] = datos['Precio'].astype(float)

#El consumo hay que partirllo en 3
partes = datos['Consumo'].str.split(')', expand=True)
datos['Consumo ciudad'] = partes.iloc[:,0].str.extract(r'(\d+(:\.\d+)) 1\100 km')
datos['Consumo combinado'] = partes.iloc[:,1].str.extract(r'(\d+(:\.\d+)) 1\100 km')

#arreglo de combinados que aparecen en la primera columna
incluir = partes.iloc[:,0].str.extract(r'(\d+(:\.\d+)) 1\100 km \(\Combinado)').astype(bool)
datos['Consumo combinado'] = datos['Consumo combinado'].combine_first(incluir)
datos['Consumo fuera'] = partes.iloc[:,2].str.extract(r'(\d+(:\.\d+)) 1\100 km \(\Frente)')
```

```
In [462]: datos.info()
```

#	Column	Non-Null Count	Dtype
0	Marca	2430 non-null	object
1	Modelo	2430 non-null	object
2	Precio	2430 non-null	float64
3	Primera matriculación	2430 non-null	datetime64[ns]
4	Kilometraje	2430 non-null	float64
5	Carburante	2430 non-null	object
6	Transmisión	2430 non-null	object
7	Potencia	2430 non-null	object
8	Tracción	2430 non-null	object
9	Tipo de vehículo	2430 non-null	object
10	Puertas	2430 non-null	int32
11	Número de asientos	2430 non-null	int64
12	Color	2430 non-null	object
13	Tapicería	2430 non-null	object
14	Tipo de ruedas	2430 non-null	object
15	Motor original	2430 non-null	object
16	Cilindrada	2430 non-null	float64
17	Consumo	2430 non-null	object
18	Clase de eficiencia CO2	2430 non-null	object
19	Emisiones de CO2	2430 non-null	float64
20	País de origen	2430 non-null	object
21	Número de llaves	2430 non-null	int64
22	Coche accidentado y reparado	2430 non-null	object
23	La última revisión se realizó el	2430 non-null	datetime64[ns]
24	Tipo de IVA	2430 non-null	object
25	ITV válida hasta	2430 non-null	datetime64[ns]
26	Matricula	2430 non-null	object
27	Número de inventario	2430 non-null	object
28	Validacion	2430 non-null	int64
29	Días desde matriculación	2430 non-null	int64
30	Días desde revisión	2430 non-null	int64
31	Días hasta ITV	2430 non-null	int64
32	Potencia CV	2430 non-null	float64
33	Consumo ciudad	2404 non-null	float64
34	Consumo combinado	2430 non-null	float64
35	Consumo fuera	2404 non-null	float64

dtypes: datetime64[ns](3), float64(8), int32(1), int64(6), object(18)  
memory usage: 674.1+ KB

```
In [463]: datos.isnull().sum()
```

```
Out[463]: Marca          0  
Modelo         0  
Precio         0  
Primera matriculación 0  
Kilometraje    0  
Carburante     0  
Transmisión    0  
Potencia       0  
Tracción        0  
Tipo de vehículo 0  
Puertas        0  
Número de asientos 0  
Color          0  
Tapicería      0  
Tipo de ruedas   0  
Motor original    0  
Cilindrada      0  
Consumo         0  
Clase de eficiencia CO2 0  
Emisiones de CO2 0  
País de origen    0  
Número de llaves   0  
Coche accidentado y reparado 0  
La última revisión se realizó el 0  
Tipo de IVA       0  
ITV válida hasta 0  
Matricula        0  
Número de inventario 0  
Validacion       0  
Días desde matriculación 0  
Días desde revisión 0  
Días hasta ITV    0  
Potencia CV       0  
Consumo ciudad    26  
Consumo combinado 0  
Consumo fuera     26  
dtype: int64
```

Eliminamos los campos de Consumo y nos quedamos solo con el combinado

```
In [464]: datos = datos.drop(['Consumo ciudad', 'Consumo fuera', 'Primera matriculación',  
                           'La última revisión se realizó el', 'ITV válida hasta', 'Número de llaves'])
```

In [465]: `datos.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2430 entries, 0 to 2429
Data columns (total 30 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Marca            2430 non-null    object  
 1   Modelo           2430 non-null    object  
 2   Precio           2430 non-null    float64 
 3   Kilometraje     2430 non-null    float64 
 4   Carburante       2430 non-null    object  
 5   Transmisión      2430 non-null    object  
 6   Potencia         2430 non-null    object  
 7   Tracción          2430 non-null    object  
 8   Tipo de vehículo 2430 non-null    object  
 9   Puertas          2430 non-null    int32   
 10  Número de asientos 2430 non-null    int64   
 11  Color             2430 non-null    object  
 12  Tapicería        2430 non-null    object  
 13  Tipo de ruedas   2430 non-null    object  
 14  Motor original    2430 non-null    object  
 15  Cilindrada        2430 non-null    float64 
 16  Consumo           2430 non-null    object  
 17  Clase de eficiencia CO2 2430 non-null    object  
 18  Emisiones de CO2   2430 non-null    float64 
 19  País de origen     2430 non-null    object  
 20  Número de llaves    2430 non-null    int64   
 21  Coche accidentado y reparado 2430 non-null    object  
 22  Tipo de IVA        2430 non-null    object  
 23  Matricula          2430 non-null    object  
 24  Validacion         2430 non-null    int64   
 25  Días desde matriculación 2430 non-null    int64   
 26  Días desde revisión 2430 non-null    int64   
 27  Días hasta ITV      2430 non-null    int64   
 28  Potencia CV         2430 non-null    float64 
 29  Consumo combinado   2430 non-null    float64 

dtypes: float64(6), int32(1), int64(6), object(17)
memory usage: 560.2+ KB
```

In [466]: `datos.describe().T`

Out[466]:

	count	mean	std	min	25%	50%	75%	max
Precio	2430.0	15719.411523	6008.143492	5799.0	11199.0	14449.0	18499.0	43899.0
Kilometraje	2430.0	74134.246091	38527.369171	202.0	43580.5	69870.0	102161.0	159645.0
Puertas	2430.0	4.816461	0.569624	2.0	5.0	5.0	5.0	6.0
Número de asientos	2430.0	5.008642	0.557452	2.0	5.0	5.0	5.0	9.0
Cilindrada	2430.0	1483.693416	370.776519	875.0	1199.0	1498.0	1598.0	2998.0
Emisiones de CO2	2430.0	118.088477	21.265345	1.0	106.0	115.0	129.0	226.0
Número de llaves	2430.0	1.821399	0.386307	1.0	2.0	2.0	2.0	4.0
Validacion	2430.0	0.208230	0.406126	0.0	0.0	0.0	0.0	1.0
Días desde matriculación	2430.0	2302.264198	900.501946	217.0	1662.0	2205.0	2929.0	4540.0
Días desde revisión	2430.0	142.609877	296.194099	3.0	42.0	76.0	125.0	3484.0
Días hasta ITV	2430.0	317.460082	239.209099	-703.0	130.0	307.0	501.0	1231.0
Potencia CV	2430.0	124.637037	38.456714	60.0	100.0	120.0	140.0	340.0
Consumo combinado	2430.0	4.827901	0.961569	0.6	4.2	4.8	5.3	9.2

In [467]: `datos.groupby('Puertas').agg(Frecuencia=('Puertas', "count"))`

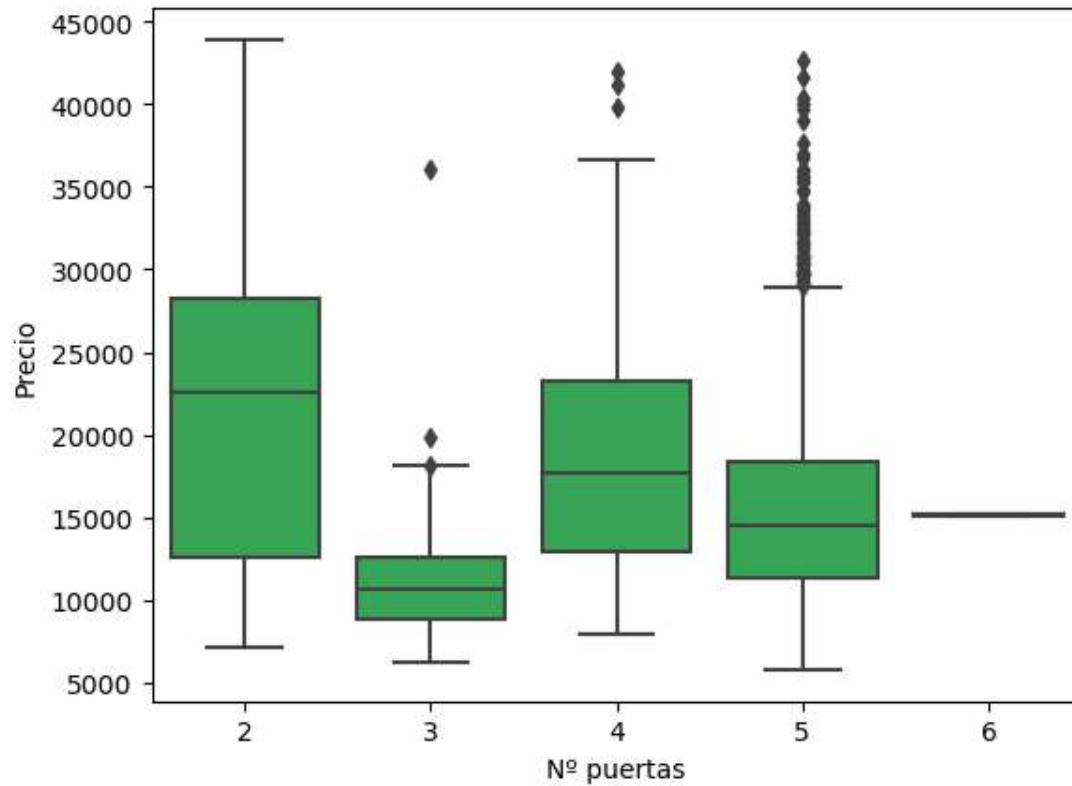
Out[467]:

Frecuencia

Puertas	Frecuencia
2	34
3	108
4	130
5	2156
6	2

```
In [468]: plot = sns.boxplot(x = 'Puertas', y = 'Precio', data = datos, color = '#28b84f')
plot.set(xlabel = 'Nº puertas')
```

```
Out[468]: [Text(0.5, 0, 'Nº puertas')]
```



```
In [469]: datos.groupby('Número de asientos').agg(Frecuencia=('Número de asientos', "count"))
```

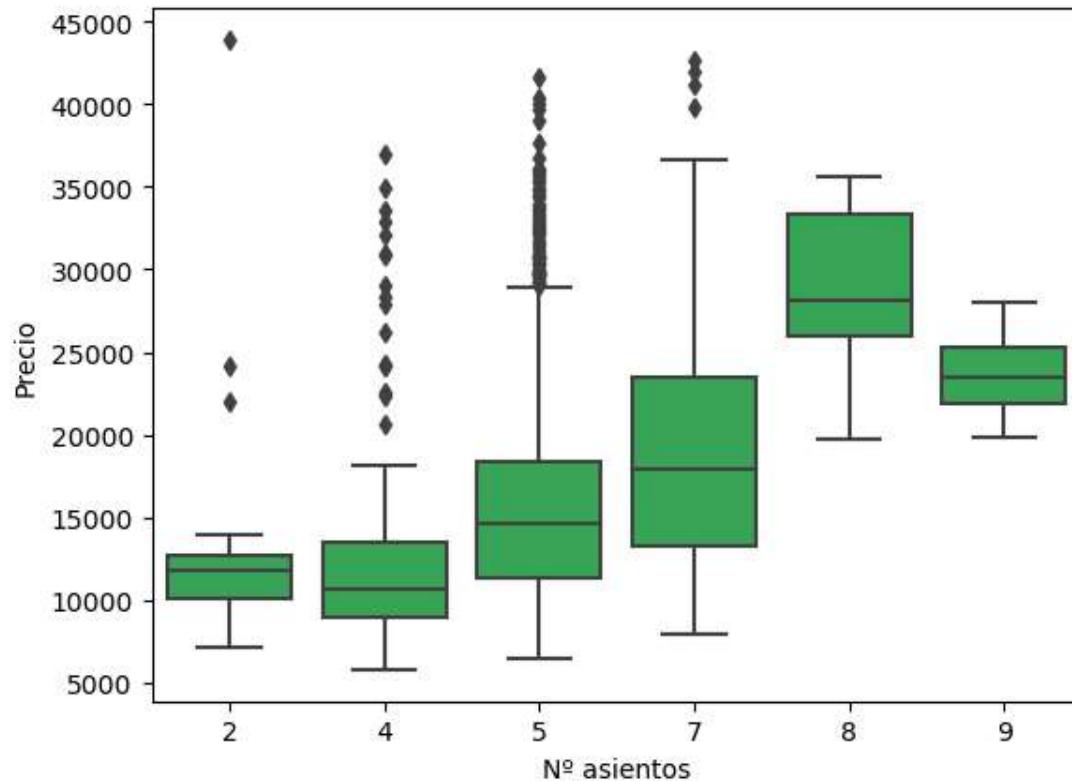
```
Out[469]:
```

Frecuencia

Número de asientos	Frecuencia
2	16
4	141
5	2175
7	88
8	6
9	4

```
In [470]: plot = sns.boxplot(x = 'Número de asientos', y = 'Precio', data = datos, color = '#2ca02c')
plot.set(xlabel = 'Nº asientos')
```

Out[470]: [Text(0.5, 0, 'Nº asientos')]



El caso extremo de dos asientos es un BWM deportivo. ¿Deberíamos quitarlo?

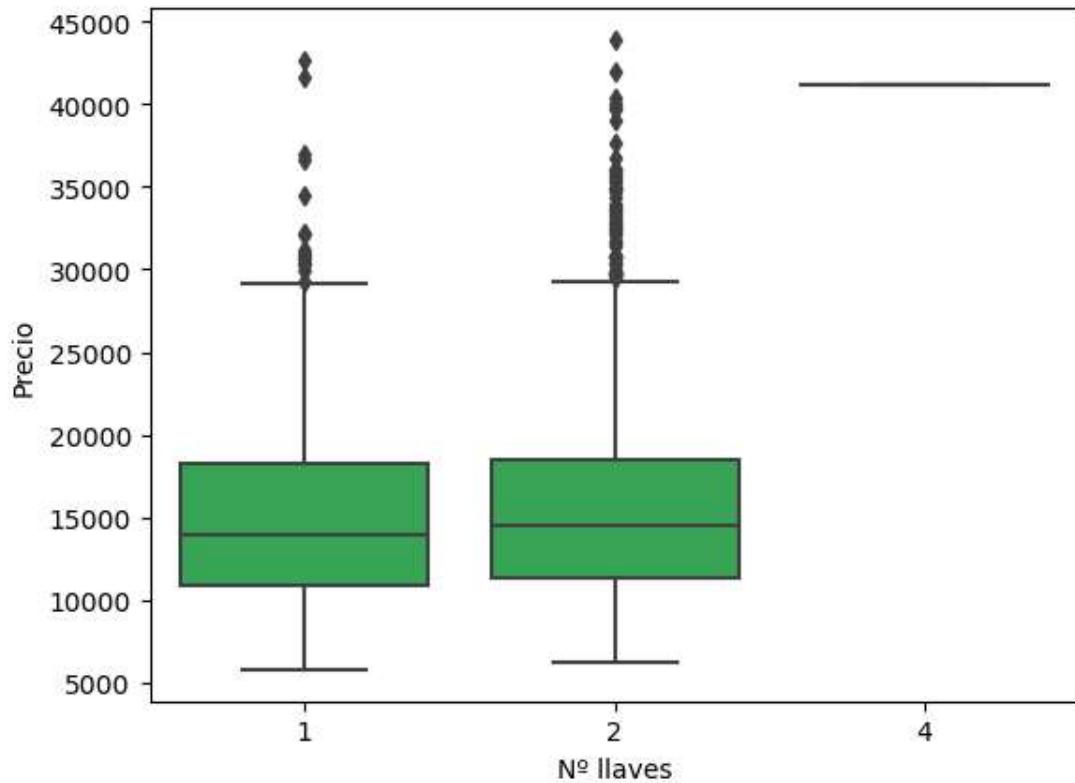
```
In [471]: datos.groupby('Número de llaves').agg(Frecuencia=('Número de llaves', "count"))
```

Out[471]:

Frecuencia	
Número de llaves	
1	436
2	1993
4	1

```
In [472]: plot = sns.boxplot(x = 'Número de llaves', y = 'Precio', data = datos, color = '#28b')
plot.set(xlabel = 'Nº llaves')
```

Out[472]: [Text(0.5, 0, 'Nº llaves')]



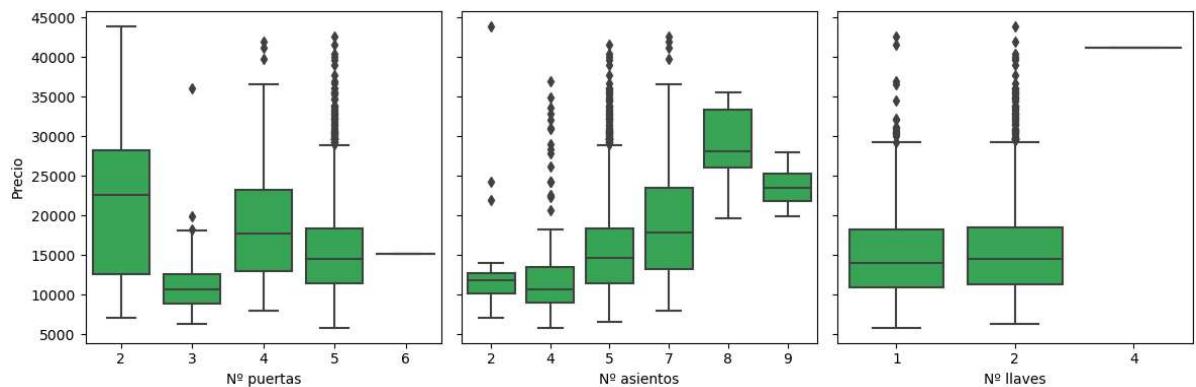
```
In [473]: fig, axes = plt.subplots(1, 3, figsize=(12,4))

plot1 = sns.boxplot(x='Puertas', y='Precio', data=datos, color='#28b84f', ax=axes[0])
plot1.set(xlabel='Nº puertas')

plot2 = sns.boxplot(x='Número de asientos', y='Precio', data=datos, color='#28b84f',
plot2.set(xlabel='Nº asientos')
plot2.set_ylabel('')
plot2.set_yticklabels([])

plot3 = sns.boxplot(x='Número de llaves', y='Precio', data=datos, color='#28b84f', a
plot3.set(xlabel='Nº llaves')
plot3.set_ylabel('')
plot3.set_yticklabels([])

plt.tight_layout()
plt.show()
```



Eliminamos los valores extremos de 4 llaves (el de 2 asientos lo dejamos para más tarde)

```
In [474]: datos = datos[~((datos['Número de llaves'] == 4) & (datos['Precio'] > 40000))]
#datos = datos[~((datos['Puertas'] == 2) & (datos['Precio'] > 20000))]
```

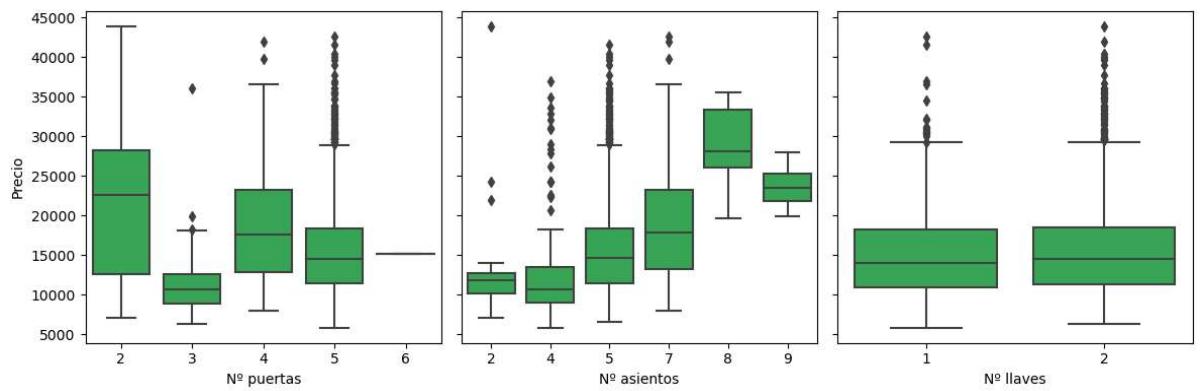
```
In [475]: fig, axes = plt.subplots(1, 3, figsize=(12,4))

plot1 = sns.boxplot(x='Puertas', y='Precio', data=datos, color='#28b84f', ax=axes[0])
plot1.set(xlabel='Nº puertas')

plot2 = sns.boxplot(x='Número de asientos', y='Precio', data=datos, color='#28b84f', ax=axes[1])
plot2.set(xlabel='Nº asientos')
plot2.set_ylabel('')
plot2.set_yticklabels([])

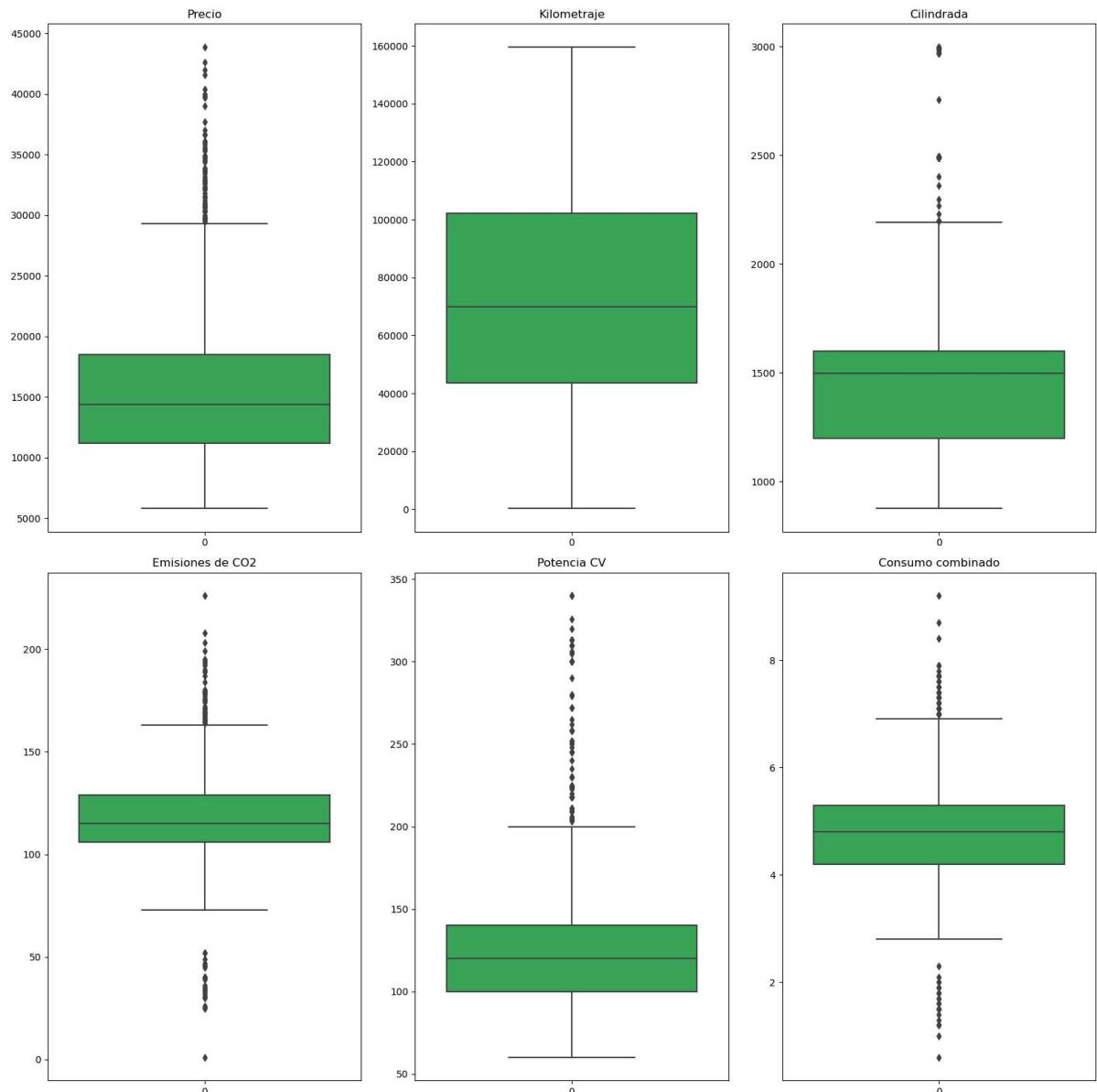
plot3 = sns.boxplot(x='Número de llaves', y='Precio', data=datos, color='#28b84f', ax=axes[2])
plot3.set(xlabel='Nº llaves')
plot3.set_ylabel('')
plot3.set_yticklabels([])

plt.tight_layout()
plt.show()
```



```
In [476]: fig, axes = plt.subplots(2, 3, figsize=(16,16))
k = 0
for i in datos.select_dtypes(exclude=['object', 'datetime64', 'int']).columns:
    sns.boxplot(datos[i],ax = axes.flatten()[k], color = '#28b84f').set_title(i)
    k = k+1

plt.tight_layout()
plt.show()
```



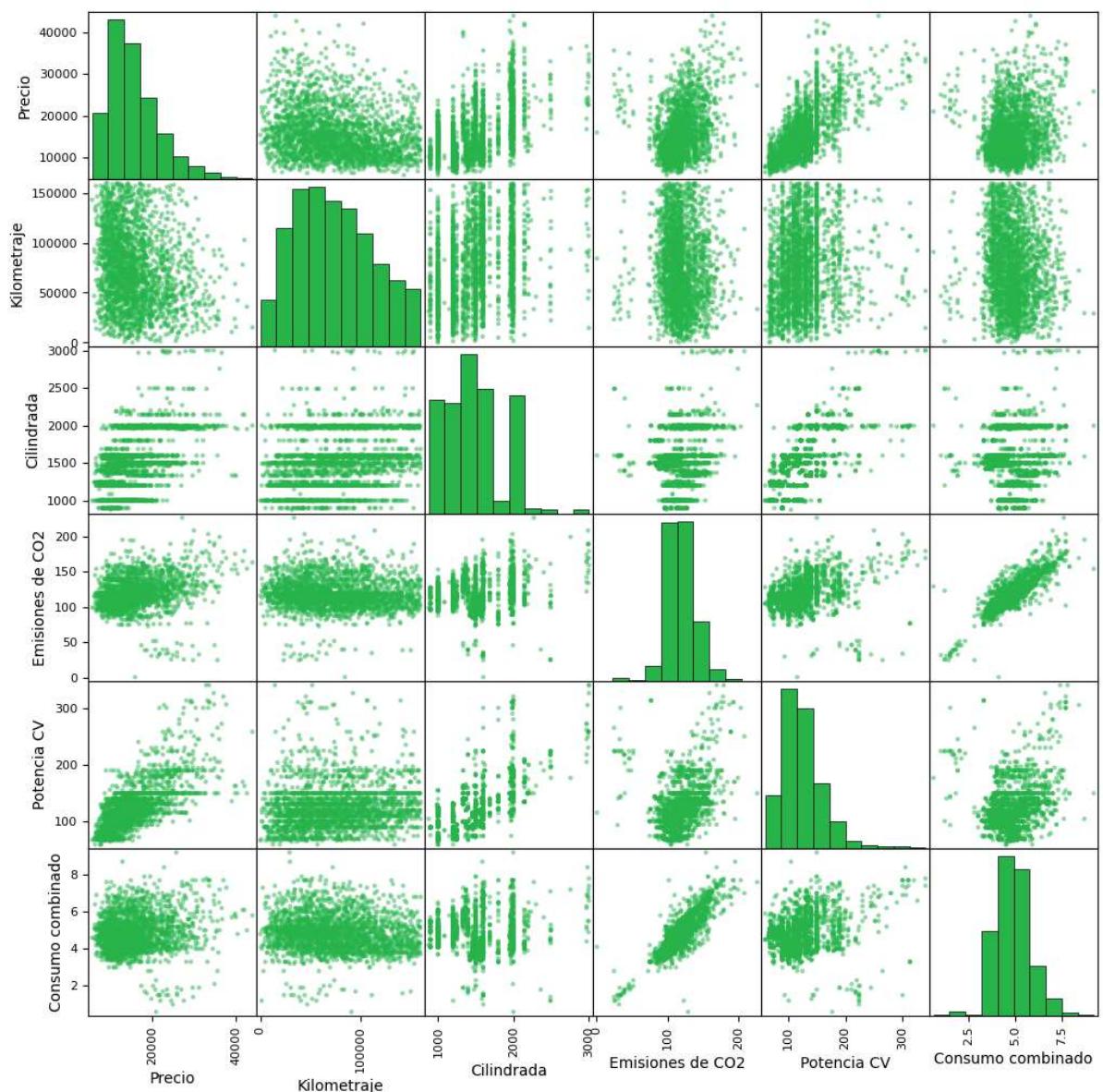
Tenemos algunos valores muy altos y muy bajos pero nada tan extremo

```
In [477]: from pandas.plotting import scatter_matrix

# Crear matriz de diagramas de dispersión
scatter_matrix(datos.select_dtypes(exclude=['object','int']), figsize=(12, 12),
               color='#28b84f') # Esto no cambia el color de las diagonales

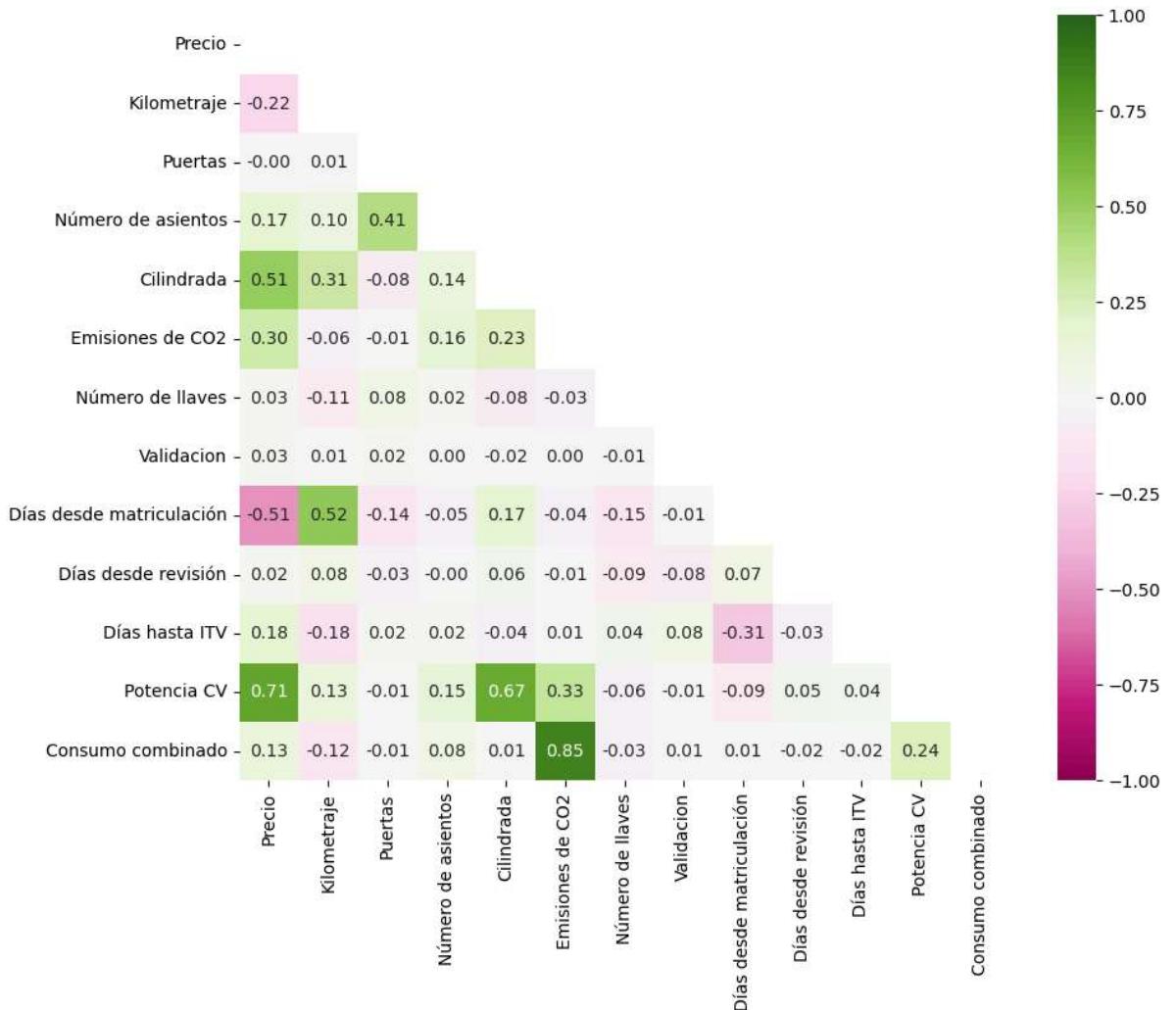
# Cambiar color de las barras en los gráficos de la diagonal
for ax in plt.gcf().get_axes():
    if hasattr(ax, 'hist'):
        for _, spine in ax.spines.items():
            spine.set_visible(True) # Mostrar los bordes del gráfico
    for bar in ax.patches:
        bar.set_facecolor('#28b84f') # Color de las barras del histograma
        bar.set_edgecolor('black') # Borde de las barras
        bar.set linewidth(0.5) # Grosor del borden de las barras

plt.show()
```



```
In [478]: correlacion = datos.select_dtypes(exclude=['object']).corr()
mask = np.zeros_like(correlacion)

plt.figure(figsize = (10,8))
mask[np.triu_indices_from(mask)] = True
sns.heatmap(correlacion,mask=mask,vmin = -1, vmax=1, center=0, cmap="PiYG",annot=True)
plt.show()
```

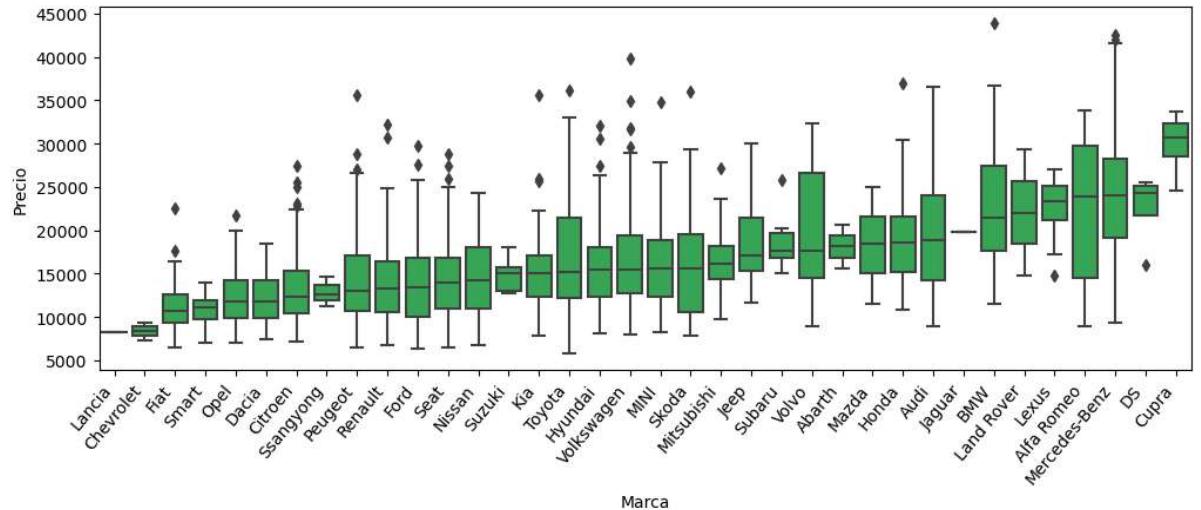


```
In [479]: abs(correlacion['Precio']).sort_values(ascending=False).head(10)
```

```
Out[479]: Precio          1.000000
Potencia CV        0.707649
Días desde matriculación  0.514635
Cilindrada         0.507657
Emisiones de CO2    0.297897
Kilometraje        0.220013
Días hasta ITV     0.183038
Número de asientos   0.172469
Consumo combinado    0.134391
Validacion         0.029973
Name: Precio, dtype: float64
```

```
In [480]: plt.subplots(figsize=(12,4))
mediana_precios_por_marca = datos.groupby('Marca')['Precio'].median().sort_values()
marca_order = mediana_precios_por_marca.index

plot = sns.boxplot(x='Marca', y='Precio', data=datos, color='#28b84f', order=marca_order)
plot.set(xlabel='Marca')
locs, labels = plt.xticks(rotation=50, ha='right')
```



```
In [481]: datos.groupby('Marca').agg(Frecuencia=('Marca', "count")).sort_values('Frecuencia',
```

```
Out[481]:
```

Frecuencia	
Marca	
Peugeot	219
Seat	218
Citroen	207
Volkswagen	173
Renault	152
Opel	151
Ford	147
Toyota	115
Audi	111
BMW	105
Kia	99
Mercedes-Benz	95
Hyundai	93
Fiat	92
Dacia	76
Nissan	72
Skoda	54
Mazda	41
Jeep	34
MINI	33
Volvo	29
Smart	21
Honda	15
Mitsubishi	13
Alfa Romeo	12
Cupra	11
Lexus	10
Suzuki	9
Subaru	6
DS	4
Ssangyong	3
Abarth	3
Land Rover	2
Chevrolet	2
Lancia	1
Jaguar	1

Eliminamos las marcas que tienen menos de 4 coches

```
In [482]: datos = datos[~((datos['Marca'] == 'Chevrolet') |  
                           (datos['Marca'] == 'Jaguar') |  
                           (datos['Marca'] == 'Lancia') |  
                           (datos['Marca'] == 'Abarth') |  
                           (datos['Marca'] == 'Ssangyong') |  
                           (datos['Marca'] == 'Land Rover'))]
```

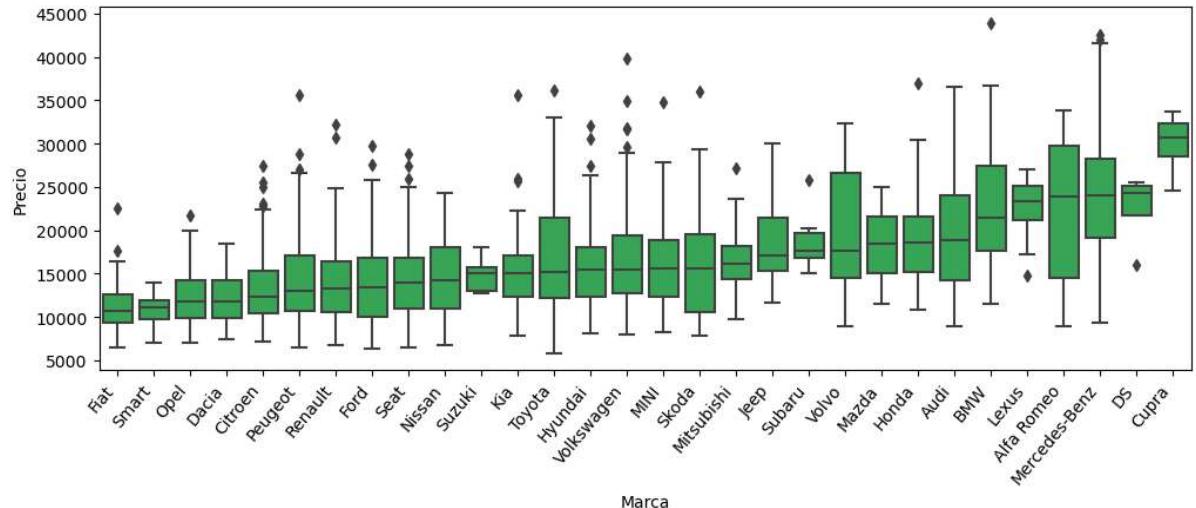
```
In [483]: datos.groupby('Marca').agg(Frecuencia=('Marca', "count")).sort_values('Frecuencia',
```

Out[483]:

Marca	Frecuencia
Peugeot	219
Seat	218
Citroen	207
Volkswagen	173
Renault	152
Opel	151
Ford	147
Toyota	115
Audi	111
BMW	105
Kia	99
Mercedes-Benz	95
Hyundai	93
Fiat	92
Dacia	76
Nissan	72
Skoda	54
Mazda	41
Jeep	34
MINI	33
Volvo	29
Smart	21
Honda	15
Mitsubishi	13
Alfa Romeo	12
Cupra	11
Lexus	10
Suzuki	9
Subaru	6
DS	4

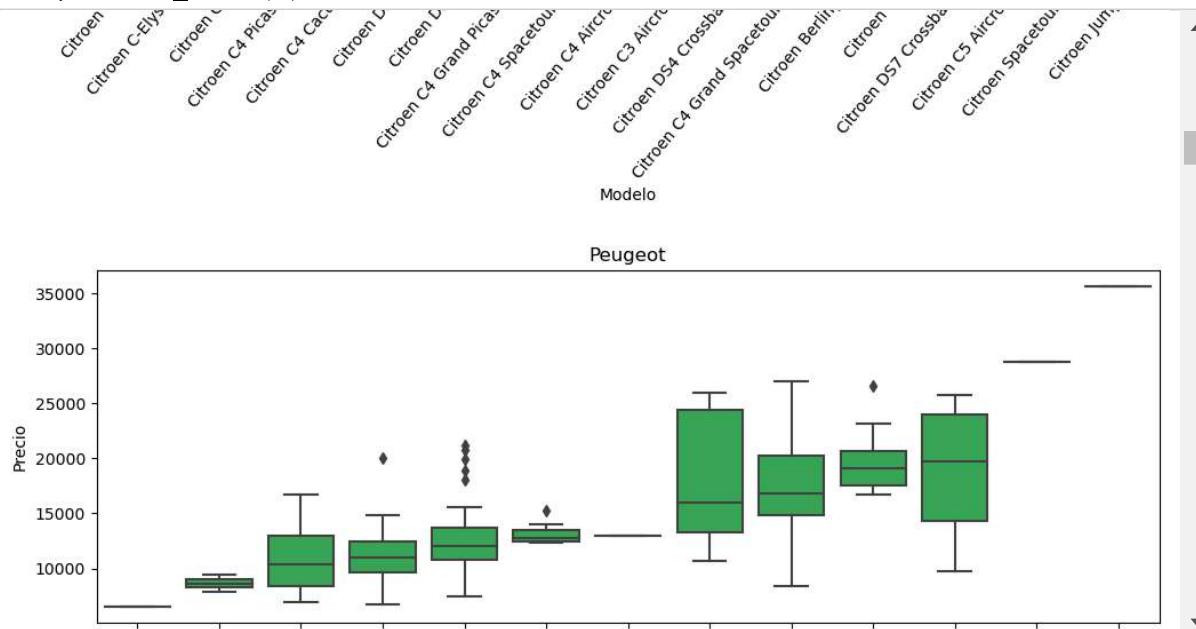
```
In [484]: plt.subplots(figsize=(12,4))
mediana_precios_por_marca = datos.groupby('Marca')['Precio'].median().sort_values()
marca_order = mediana_precios_por_marca.index

plot = sns.boxplot(x='Marca', y='Precio', data=datos, color='#28b84f', order=marca_order)
plot.set(xlabel='Marca')
locs, labels = plt.xticks(rotation=50, ha='right')
```



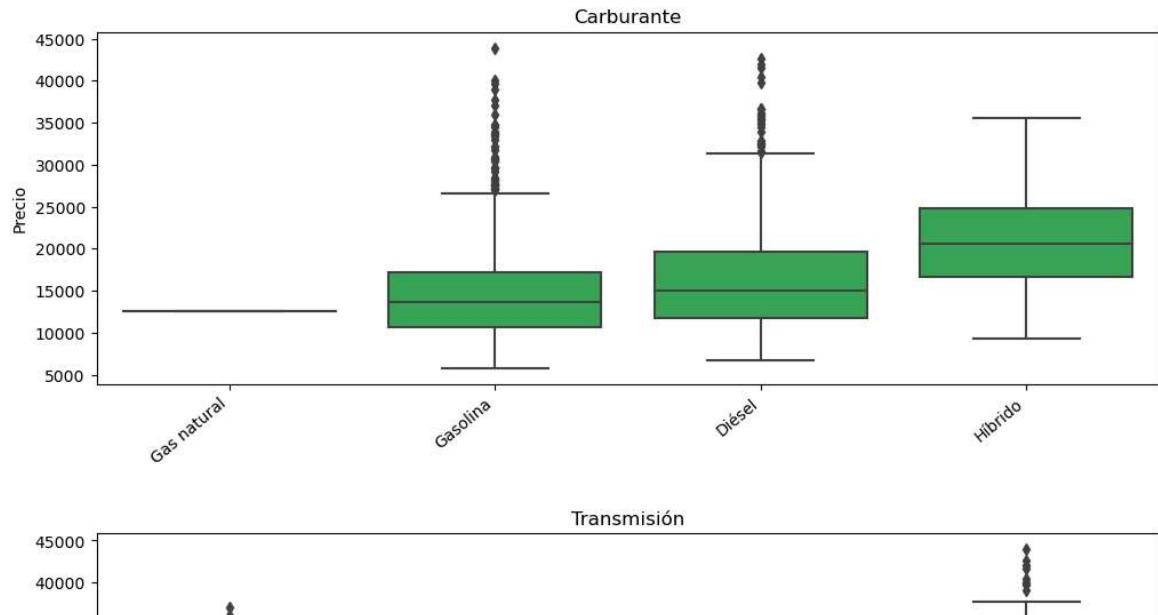
```
In [485]: for i in marca_order:
    plt.subplots(figsize=(12,4))
    mediana_precios_por_modelo = datos[datos['Marca'] == i].groupby('Modelo')['Precio'].median()
    modelo_order = mediana_precios_por_modelo.index

    plot = sns.boxplot(x='Modelo', y='Precio', data=datos[datos['Marca'] == i], color='green')
    plot.set(xlabel='Modelo')
    locs, labels = plt.xticks(rotation=50, ha='right')
    plot.set_title(i)
```



```
In [486]: for i in ['Carburante', 'Transmisión', 'Tracción', 'Tipo de vehículo',
   'Color', 'Tapicería', 'Tipo de ruedas', 'Clase de eficiencia CO2',
   'País de origen', 'Motor original', 'Coche accidentado y reparado',
   'Tipo de IVA']:
    plt.subplots(figsize=(12,4))
    mediana_precios_por_carac = datos.groupby(i)['Precio'].median().sort_values()
    carac_order = mediana_precios_por_carac.index

    plot = sns.boxplot(x=i, y='Precio', data=datos, color='#28b84f', order=carac_order)
    plot.set_xlabel('')
    plot.set_title(i)
    locs, labels = plt.xticks(rotation=40, ha='right')
```



Para algunas variables se pueden hacer agrupaciones para reducir los diferentes valores de cada una, por ejemplo, en el tipo de cambio para eliminar el semi- automático y el doble embrague. En la variable tapicería es necesario eliminar el texto entre paréntesis. Eliminamos el pais, ya que solo tiene un valor.

El coche con ruedas de emergencia no tiene sentido y se va a eliminar

Se va a eliminar el coche de gas natural

Se va a eliminar el de tipo Pickup

Se va a eliminar la clase de eficiencia EURO 6c

Todo esto va a hacer que tengamos menos variables dummies, cosa que beneficiará los procesos de búsqueda de los modelos

```
In [487]: datos['Tapicería'] = [texto.split('(')[0].strip() for texto in datos['Tapicería']]

#¿el mapeo de transmisión se justificará en el word con referencia a alguna página q
#¿es necesario?
mapeo_transmision = {'Semi-automático': 'Automático', 'Doble embrague': 'Automático'
                     'Cambio tipo manual': 'Manual', 'Cambio tipo automático': 'Automático'}
def transformar_transmision(transmision):
    return mapeo_transmision.get(transmision, transmision)

datos['Transmisión'] = datos['Transmisión'].apply(transformar_transmision)

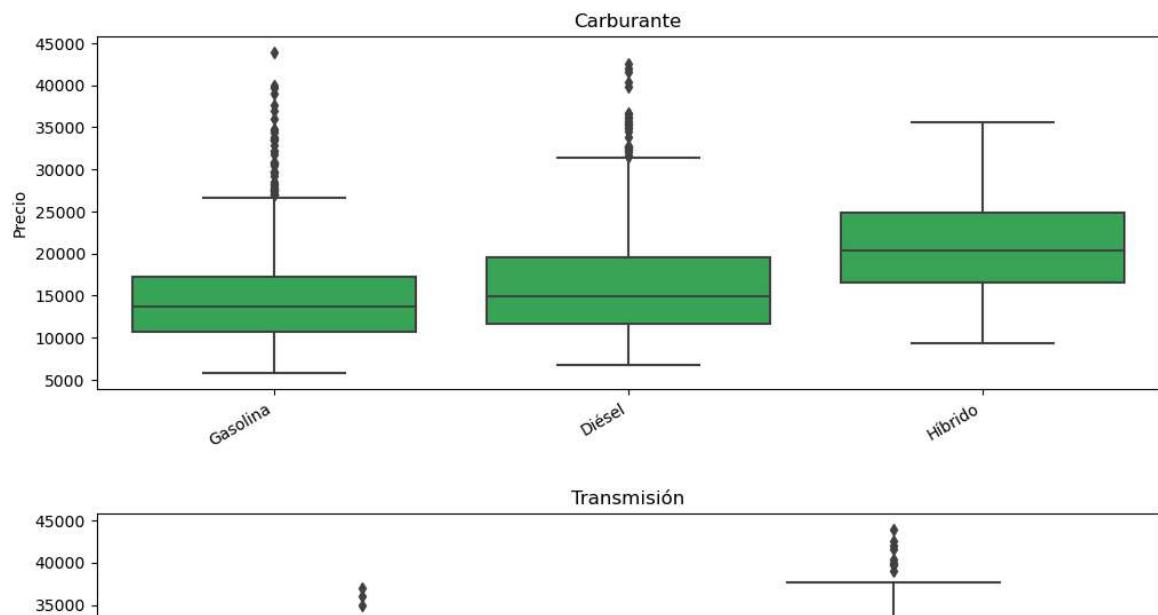
datos = datos[~((datos['Tipo de ruedas'] == 'Emergencia') |
                 (datos['Tipo de vehículo'] == 'Pickup') |
                 (datos['Carburante'] == 'Gas natural') |
                 (datos['Clase de eficiencia CO2'] == 'EURO 6c'))]

datos = datos.drop('País de origen', axis=1)
```

```
In [488]: for i in ['Carburante', 'Transmisión', 'Tracción', 'Tipo de vehículo',
                  'Color', 'Tapicería', 'Tipo de ruedas', 'Clase de eficiencia CO2',
                  'Motor original', 'Coche accidentado y reparado',
                  'Tipo de IVA']:

    plt.subplots(figsize=(12,4))
    mediana_precios_por_carac = datos.groupby(i)['Precio'].median().sort_values()
    carac_order = mediana_precios_por_carac.index

    plot = sns.boxplot(x=i, y='Precio', data=datos, color='#28b84f', order=carac_order)
    plot.set_xlabel('')
    plot.set_title(i)
    locs, labels = plt.xticks(rotation=30, ha='right')
```



Ahora haremos dummies de las variables categóricas que son interesantes en el estudio y convertimos en 0 y 1 las variables binarias:

- Marca
- Modelo
- Carburante
- Transmisión
- Tracción
- Tipo de vehículo
- Color
- Tapicería

- Tipo de ruedas
- **Motor original**
- Eficiencia
- País de origen
- **Coche accidentado y reparado**
- **Tipo de IVA**

In [489]: datos

Out[489]:

	Marca	Modelo	Precio	Kilometraje	Carburante	Transmisión	Potencia	Tracción
0	Opel	Opel Adam	8799.0	17256.0	Gasolina	Manual	87 CV / 64 kW	Tracción delantera
1	Ford	Ford Puma	19899.0	28670.0	Híbrido	Manual	125 CV / 92 kW	Tracción delantera
2	Honda	Honda CR-V	14999.0	130594.0	Gasolina	Automático	155 CV / 114 kW	Tracción total (4x4)
3	Renault	Renault Clio	10499.0	86083.0	Diésel	Manual	90 CV / 66 kW	Tracción delantera
4	Audi	Audi A3	23699.0	54925.0	Diésel	Automático	150 CV / 110 kW	Tracción delantera
5	Renault	Renault Clio	8799.0	26382.0	Gasolina	Manual	90 CV / 66 kW	Tracción delantera

In [490]: #Binarias

```
datos['Motor original'] = datos['Motor original'].map({'Sí': 1, 'No': 0})
datos['Coche accidentado y reparado'] = datos['Coche accidentado y reparado'].map({'Sí': 1, 'No': 0})
datos['Tipo de IVA'] = datos['Tipo de IVA'].map({'IVA no deducible': 0, 'IVA deducible': 1})
```

In [491]: #Categóricas

```
datos = pd.get_dummies(datos, columns=['Marca', 'Modelo',
                                         'Carburante', 'Transmisión', 'Tracción', 'Tipo de carrocería',
                                         'Color', 'Tapicería', 'Tipo de ruedas', 'Clase'])
```

In [492]: datos

Out[492]:

	Precio	Kilometraje	Potencia	Puertas	Número de asientos	Motor original	Cilindrada	Consumo	Emisiones de CO2
0	8799.0	17256.0	87 CV / 64 kW	3	4	1	1398.0	6.6 l/100 km (En la ciudad) 5 l/100 km (Combina...)	119.0
1	19899.0	28670.0	125 CV / 92 kW	5	5	1	999.0	5.6 l/100 km (Combinado)	122.0
2	14999.0	130594.0	155 CV / 114 kW	5	5	1	1997.0	10.1 l/100 km (En la ciudad) 7.7 l/100 km (Combina...) 4 l/100 km (En la ciud...)	179.0

```
In [493]: datos = datos.select_dtypes(exclude=['object'])
```

```
In [494]: datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2412 entries, 0 to 2429
Columns: 348 entries, Precio to Clase de eficiencia CO2_EURO 6d
dtypes: float64(6), int32(333), int64(9)
memory usage: 3.4 MB
```

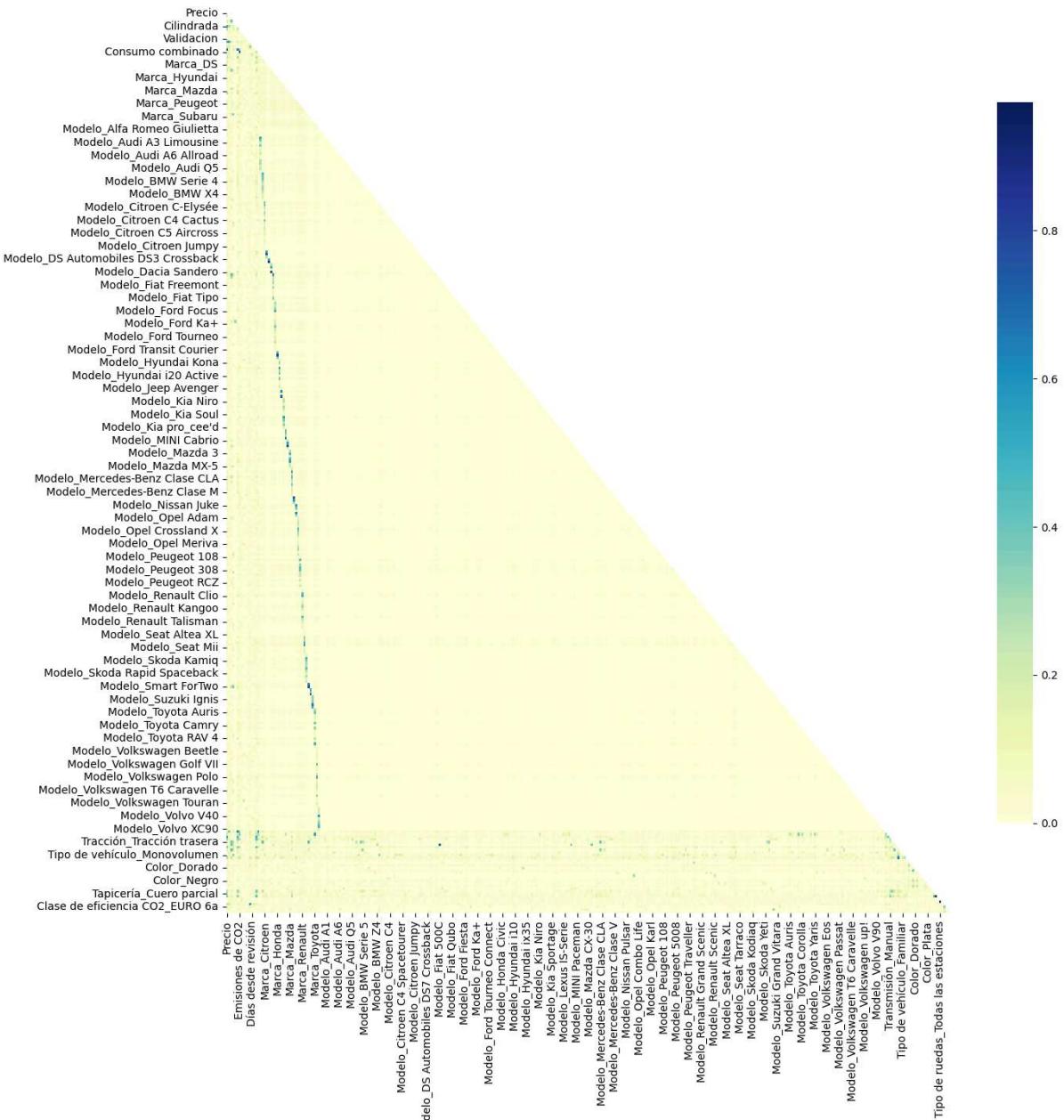
Lo que más columnas mete al crear dummies es el modelo. ¿Es lo suficientemente importante como para dejarlo?

```
In [495]: datos.isnull().sum()
```

```
Out[495]: Precio          0
Kilometraje      0
Puertas          0
Número de asientos 0
Motor original    0
Cilindrada        0
Emisiones de CO2 0
Número de llaves   0
Coche accidentado y reparado 0
Tipo de IVA        0
Validacion        0
Días desde matriculación 0
Días desde revisión 0
Días hasta ITV     0
Potencia CV        0
Consumo combinado 0
Marca_Audi         0
Marca_BMW          0
Marca_Citroen     0
Marca_Guardia     0
```

```
In [496]: correlacion = np.abs(datos.corr())
mask = np.zeros_like(correlacion)

mask[np.triu_indices_from(mask)] = True
plt.subplots(figsize=(15,15))
sns.heatmap(correlacion,mask=mask, cmap="YlGnBu", cbar_kws={"shrink": .8})
plt.show()
```



In [497]: `pd.DataFrame(abs(correlacion['Precio']).sort_values(ascending=False).head(20))`

Out[497]:

	Precio
Precio	1.000000
Potencia CV	0.708135
Transmisión_Manual	0.558699
Días desde matriculación	0.513368
Cilindrada	0.506581
Tapicería_Tejido	0.395157
Tipo de vehículo_Coche pequeño	0.361924
Tracción_Tracción total (4x4)	0.317346
Marca_Mercedes-Benz	0.308920
Emisiones de CO2	0.295965
Tipo de vehículo_SUV	0.294115
Tapicería_Cuero	0.292107
Clase de eficiencia CO2_EURO 6	0.257252
Marca_BMW	0.235137
Tapicería_Cuero parcial	0.229061
Carburante_Híbrido	0.220244
Kilometraje	0.220025
Carburante_Gasolina	0.194977
Días hasta ITV	0.183467
Número de asientos	0.175681

Viendo que no hay ningún color ni modelo que tengan una gran correlación con el precio, vamos a optar por quitarlos y facilitar así los modelos. Cuantas menos características, menos dummies y la búsqueda será más rápida

In [498]: `columnas = datos.filter(regex='^(Color_|Modelo_)').columns`  
*# Eliminar las columnas seleccionadas*  
`datos = datos.drop(columns=columnas)`

```
In [499]: columnasVal
```

```
Out[499]: Index(['Precio', 'Kilometraje', 'Puertas', 'Número de asientos',  
'Motor original', 'Cilindrada', 'Emisiones de CO2', 'Número de llaves',  
'Coche accidentado y reparado', 'Tipo de IVA',  
'Días desde matriculación', 'Días desde revisión', 'Días hasta ITV',  
'Potencia CV', 'Consumo combinado', 'Marca_Audi', 'Marca_BMW',  
'Marca_Citroen', 'Marca_Cupra', 'Marca_DS', 'Marca_Dacia', 'Marca_Fiat',  
'Marca_Ford', 'Marca_Honda', 'Marca_Hyundai', 'Marca_Jeep', 'Marca_Kia',  
'Marca_Lexus', 'Marca_MINI', 'Marca_Mazda', 'Marca_Mercedes-Benz',  
'Marca_Mitsubishi', 'Marca_Nissan', 'Marca_Opel', 'Marca_Peugeot',  
'Marca_Renault', 'Marca_Seat', 'Marca_Skoda', 'Marca_Smart',  
'Marca_Subaru', 'Marca_Suzuki', 'Marca_Toyota', 'Marca_Volkswagen',  
'Marca_Volvo', 'Carburante_Gasolina', 'Carburante_Híbrido',  
'Transmisión_Manual', 'Tracción_Tracción total (4x4)',  
'Tracción_Tracción trasera', 'Tipo de vehículo_Cabrio',  
'Tipo de vehículo_Coche pequeño', 'Tipo de vehículo_Coupé',  
'Tipo de vehículo_Familiar', 'Tipo de vehículo_Monovolumen',  
'Tipo de vehículo_SUV', 'Tapicería_Cuero', 'Tapicería_Cuero parcial',  
'Tapicería_Tejido', 'Tipo de ruedas_Todas las estaciones',  
'Tipo de ruedas_Verano', 'Clase de eficiencia CO2_EURO 6',  
'Clase de eficiencia CO2_EURO 6a', 'Clase de eficiencia CO2_EURO 6b',  
'Clase de eficiencia CO2_EURO 6d'],  
dtype='object')
```

```
In [500]: datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2412 entries, 0 to 2429
Data columns (total 65 columns):
 #   Column           Non-Null Count Dtype
 ---  -----
 0   Precio          2412 non-null   float64
 1   Kilometraje     2412 non-null   float64
 2   Puertas          2412 non-null   int32
 3   Número de asientos  2412 non-null   int64
 4   Motor original    2412 non-null   int64
 5   Cilindrada        2412 non-null   float64
 6   Emisiones de CO2  2412 non-null   float64
 7   Número de llaves   2412 non-null   int64
 8   Coche accidentado y reparado 2412 non-null   int64
 9   Tipo de IVA       2412 non-null   int64
 10  Validacion       2412 non-null   int64
 11  Días desde matriculación 2412 non-null   int64
 12  Días desde revisión   2412 non-null   int64
 13  Días hasta ITV      2412 non-null   int64
 14  Potencia CV         2412 non-null   float64
 15  Consumo combinado   2412 non-null   float64
 16  Marca_Audi          2412 non-null   int32
 17  Marca_BMW           2412 non-null   int32
 18  Marca_Citroen      2412 non-null   int32
 19  Marca_Cupra         2412 non-null   int32
 20  Marca_DS            2412 non-null   int32
 21  Marca_Dacia         2412 non-null   int32
 22  Marca_Fiat          2412 non-null   int32
 23  Marca_Ford          2412 non-null   int32
 24  Marca_Honda         2412 non-null   int32
 25  Marca_Hyundai       2412 non-null   int32
 26  Marca_Jeep           2412 non-null   int32
 27  Marca_Kia            2412 non-null   int32
 28  Marca_Lexus          2412 non-null   int32
 29  Marca_MINI          2412 non-null   int32
 30  Marca_Mazda          2412 non-null   int32
 31  Marca_Mercedes-Benz 2412 non-null   int32
 32  Marca_Mitsubishi     2412 non-null   int32
 33  Marca_Nissan         2412 non-null   int32
 34  Marca_Opel            2412 non-null   int32
 35  Marca_Peugeot        2412 non-null   int32
 36  Marca_Renault        2412 non-null   int32
 37  Marca_Seat            2412 non-null   int32
 38  Marca_Skoda           2412 non-null   int32
 39  Marca_Smart           2412 non-null   int32
 40  Marca_Subaru          2412 non-null   int32
 41  Marca_Suzuki          2412 non-null   int32
 42  Marca_Toyota          2412 non-null   int32
 43  Marca_Volkswagen     2412 non-null   int32
 44  Marca_Volvo           2412 non-null   int32
 45  Carburante_Gasolina   2412 non-null   int32
 46  Carburante_Híbrido     2412 non-null   int32
 47  Transmisión_Manual    2412 non-null   int32
 48  Tracción_Tracción total (4x4) 2412 non-null   int32
 49  Tracción_Tracción trasera 2412 non-null   int32
 50  Tipo de vehículo_Cabrio 2412 non-null   int32
 51  Tipo de vehículo_Coche pequeño 2412 non-null   int32
 52  Tipo de vehículo_Coupé    2412 non-null   int32
 53  Tipo de vehículo_Familiar 2412 non-null   int32
 54  Tipo de vehículo_Monovolumen 2412 non-null   int32
 55  Tipo de vehículo_SUV      2412 non-null   int32
 56  Tapicería_Cuero         2412 non-null   int32
 57  Tapicería_Cuero parcial 2412 non-null   int32
 58  Tapicería_Tejido        2412 non-null   int32
 59  Tipo de ruedas_Todas las estaciones 2412 non-null   int32
```

```

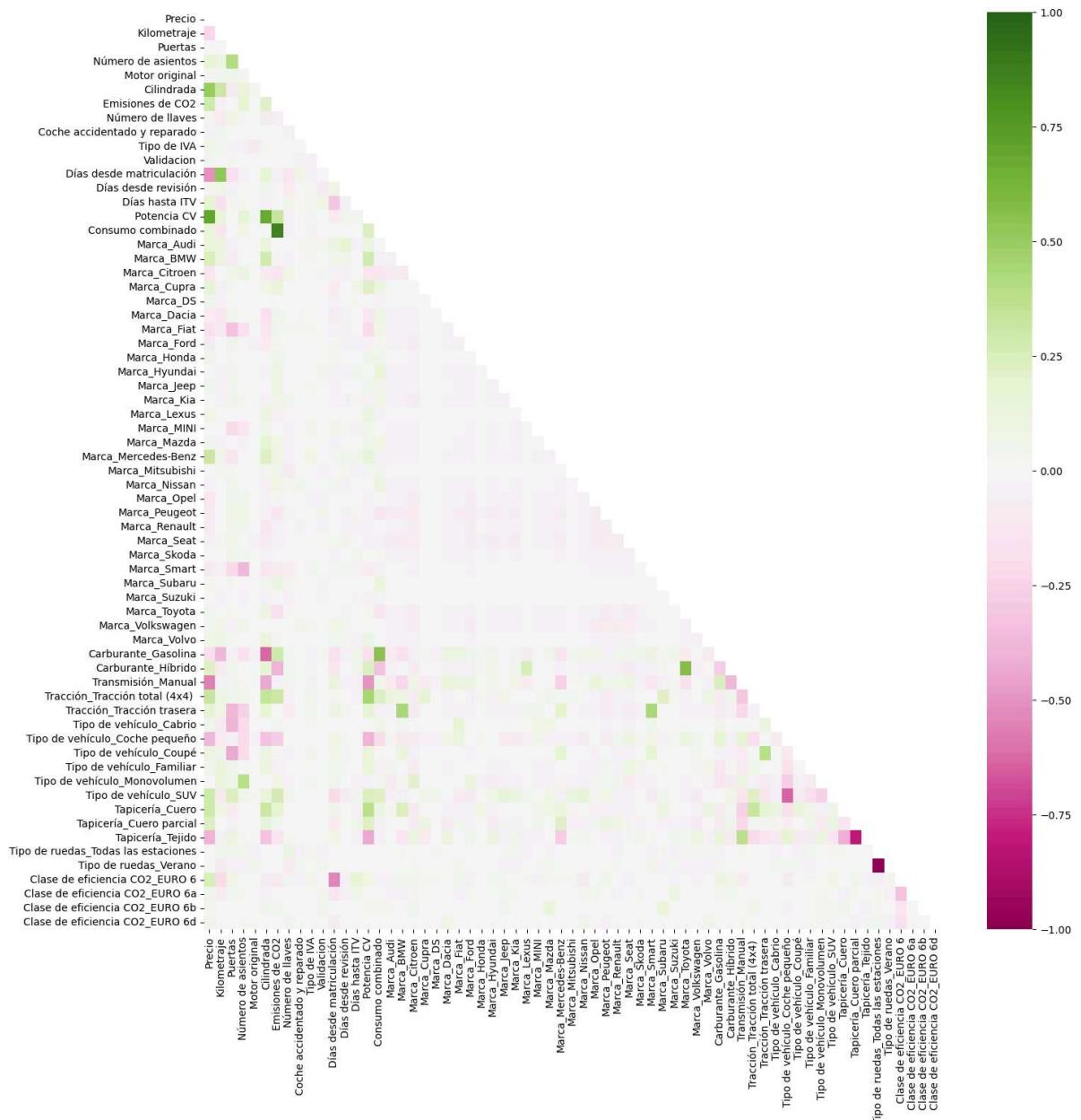
60  Tipo de ruedas_Verano           2412 non-null  int32
61  Clase de eficiencia CO2_EURO 6   2412 non-null  int32
62  Clase de eficiencia CO2_EURO 6a    2412 non-null  int32
63  Clase de eficiencia CO2_EURO 6b    2412 non-null  int32
64  Clase de eficiencia CO2_EURO 6d    2412 non-null  int32
dtypes: float64(6), int32(50), int64(9)
memory usage: 772.6 KB

```

Pasamos a tener 64 columnas y va a ser más fácil trabajar

```
In [501]: correlacion = datos.corr()
mask = np.zeros_like(correlacion)

plt.figure(figsize = (16,16))
mask[np.triu_indices_from(mask)] = True
sns.heatmap(correlacion,mask=mask,vmin = -1, vmax=1, center=0, cmap="PiYG")
plt.show()
```



¿Incluyo el gráfico anterior en el word o es demasiado? Puede que solo interese la tabla con la correlación con el precio

```
In [502]: pd.DataFrame(abs(correlacion['Precio']).sort_values(ascending=False).head(20))
```

Out[502]:

	Precio
Precio	1.000000
Potencia CV	0.708135
Transmisión_Manual	0.558699
Días desde matriculación	0.513368
Cilindrada	0.506581
Tapicería_Tejido	0.395157
Tipo de vehículo_Coche pequeño	0.361924
Tracción_Tracción total (4x4)	0.317346
Marca_Mercedes-Benz	0.308920
Emisiones de CO2	0.295965
Tipo de vehículo_SUV	0.294115
Tapicería_Cuero	0.292107
Clase de eficiencia CO2_EURO 6	0.257252
Marca_BMW	0.235137
Tapicería_Cuero parcial	0.229061
Carburante_Híbrido	0.220244
Kilometraje	0.220025
Carburante_Gasolina	0.194977
Días hasta ITV	0.183467
Número de asientos	0.175681

In [503]: `datos.describe().T #Este no se va a incluir`

Out[503]:

		count	mean	std	min	25%	50%	75%	mi
	Precio	2412.0	15699.373134	5983.022769	5799.0	11199.00	14399.0	18499.0	43899
	Kilometraje	2412.0	74205.473051	38501.086704	202.0	43631.75	69991.5	102276.0	159645
	Puertas	2412.0	4.820066	0.562888	2.0	5.00	5.0	5.0	6
Número de asientos		2412.0	5.009536	0.556542	2.0	5.00	5.0	5.0	9
	Motor original	2412.0	0.999171	0.028790	0.0	1.00	1.0	1.0	1
	Cilindrada	2412.0	1480.996683	368.799280	875.0	1199.00	1498.0	1598.0	2998
	Emisiones de CO2	2412.0	117.842869	20.992498	1.0	106.00	115.0	129.0	208
	Número de llaves	2412.0	1.820481	0.383866	1.0	2.00	2.0	2.0	2
Coche accidentado y reparado		2412.0	0.009536	0.097204	0.0	0.00	0.0	0.0	1
	Tipo de IVA	2412.0	0.030680	0.172485	0.0	0.00	0.0	0.0	1
	Validacion	2412.0	0.208955	0.406647	0.0	0.00	0.0	0.0	1
	Días desde matriculación	2412.0	2302.738806	900.106605	217.0	1662.00	2206.5	2931.5	4540
	Días desde revisión	2412.0	142.856551	297.232165	3.0	42.00	76.0	125.0	3484
	Días hasta ITV	2412.0	317.818823	239.281482	-703.0	130.00	306.5	501.0	1231
	Potencia CV	2412.0	124.452322	38.373276	60.0	100.00	120.0	140.0	340
Consumo combinado		2412.0	4.819900	0.956582	0.6	4.10	4.8	5.3	9
	Marca_Audi	2412.0	0.046020	0.209572	0.0	0.00	0.0	0.0	1
	Marca_BMW	2412.0	0.043532	0.204094	0.0	0.00	0.0	0.0	1
	Marca_Citroen	2412.0	0.085821	0.280157	0.0	0.00	0.0	0.0	1
	Marca_Cupra	2412.0	0.004561	0.067392	0.0	0.00	0.0	0.0	1
	Marca_DS	2412.0	0.001658	0.040698	0.0	0.00	0.0	0.0	1
	Marca_Dacia	2412.0	0.031509	0.174725	0.0	0.00	0.0	0.0	1
	Marca_Fiat	2412.0	0.037728	0.190577	0.0	0.00	0.0	0.0	1
	Marca_Ford	2412.0	0.060945	0.239280	0.0	0.00	0.0	0.0	1
	Marca_Honda	2412.0	0.006219	0.078631	0.0	0.00	0.0	0.0	1
	Marca_Hyundai	2412.0	0.038557	0.192577	0.0	0.00	0.0	0.0	1
	Marca_Jeep	2412.0	0.014096	0.117912	0.0	0.00	0.0	0.0	1
	Marca_Kia	2412.0	0.041045	0.198435	0.0	0.00	0.0	0.0	1
	Marca_Lexus	2412.0	0.004146	0.064269	0.0	0.00	0.0	0.0	1
	Marca_MINI	2412.0	0.013682	0.116190	0.0	0.00	0.0	0.0	1
	Marca_Mazda	2412.0	0.016584	0.127732	0.0	0.00	0.0	0.0	1
Marca_Mercedes-Benz		2412.0	0.039386	0.194553	0.0	0.00	0.0	0.0	1
	Marca_Mitsubishi	2412.0	0.004975	0.070373	0.0	0.00	0.0	0.0	1
	Marca_Nissan	2412.0	0.029436	0.169061	0.0	0.00	0.0	0.0	1
	Marca_Opel	2412.0	0.062604	0.242299	0.0	0.00	0.0	0.0	1
	Marca_Peugeot	2412.0	0.090796	0.287378	0.0	0.00	0.0	0.0	1
	Marca_Renault	2412.0	0.063018	0.243046	0.0	0.00	0.0	0.0	1
	Marca_Seat	2412.0	0.090381	0.286787	0.0	0.00	0.0	0.0	1
	Marca_Skoda	2412.0	0.022388	0.147973	0.0	0.00	0.0	0.0	1
	Marca_Smart	2412.0	0.008706	0.092921	0.0	0.00	0.0	0.0	1

	count	mean	std	min	25%	50%	75%	max
<b>Marca_Subaru</b>	2412.0	0.002488	0.049824	0.0	0.00	0.0	0.0	1
<b>Marca_Suzuki</b>	2412.0	0.003731	0.060983	0.0	0.00	0.0	0.0	1
<b>Marca_Toyota</b>	2412.0	0.047264	0.212246	0.0	0.00	0.0	0.0	1
<b>Marca_Volkswagen</b>	2412.0	0.071725	0.258085	0.0	0.00	0.0	0.0	1
<b>Marca_Volvo</b>	2412.0	0.012023	0.109012	0.0	0.00	0.0	0.0	1
<b>Carburante_Gasolina</b>	2412.0	0.538557	0.498614	0.0	0.00	1.0	1.0	1
<b>Carburante_Híbrido</b>	2412.0	0.059287	0.236210	0.0	0.00	0.0	0.0	1
<b>Transmisión_Manual</b>	2412.0	0.699420	0.458606	0.0	0.00	1.0	1.0	1
<b>Tracción_Tracción total (4x4)</b>	2412.0	0.050995	0.220033	0.0	0.00	0.0	0.0	1
<b>Tracción_Tracción trasera</b>	2412.0	0.043947	0.205020	0.0	0.00	0.0	0.0	1
<b>Tipo de vehículo_Cabrio</b>	2412.0	0.005804	0.075980	0.0	0.00	0.0	0.0	1
<b>Tipo de vehículo_Coche pequeño</b>	2412.0	0.412106	0.492316	0.0	0.00	0.0	1.0	1
<b>Tipo de vehículo_Coupé</b>	2412.0	0.009950	0.099274	0.0	0.00	0.0	0.0	1
<b>Tipo de vehículo_Familiar</b>	2412.0	0.048922	0.215750	0.0	0.00	0.0	0.0	1
<b>Tipo de vehículo_Monovolumen</b>	2412.0	0.103234	0.304327	0.0	0.00	0.0	0.0	1
<b>Tipo de vehículo_SUV</b>	2412.0	0.364428	0.481369	0.0	0.00	0.0	1.0	1
<b>Tapicería_Cuero</b>	2412.0	0.067579	0.251074	0.0	0.00	0.0	0.0	1
<b>Tapicería_Cuero parcial</b>	2412.0	0.220564	0.414713	0.0	0.00	0.0	0.0	1
<b>Tapicería_Tejido</b>	2412.0	0.699420	0.458606	0.0	0.00	1.0	1.0	1
<b>Tipo de ruedas_Todas las estaciones</b>	2412.0	0.269900	0.444000	0.0	0.00	0.0	1.0	1
<b>Tipo de ruedas_Verano</b>	2412.0	0.719320	0.449425	0.0	0.00	1.0	1.0	1
<b>Clase de eficiencia CO2_EURO 6</b>	2412.0	0.833333	0.372755	0.0	1.00	1.0	1.0	1
<b>Clase de eficiencia CO2_EURO 6a</b>	2412.0	0.020730	0.142507	0.0	0.00	0.0	0.0	1
<b>Clase de eficiencia CO2_EURO 6b</b>	2412.0	0.004146	0.064269	0.0	0.00	0.0	0.0	1
<b>Clase de eficiencia CO2_EURO 6d</b>	2412.0	0.006219	0.078631	0.0	0.00	0.0	0.0	1

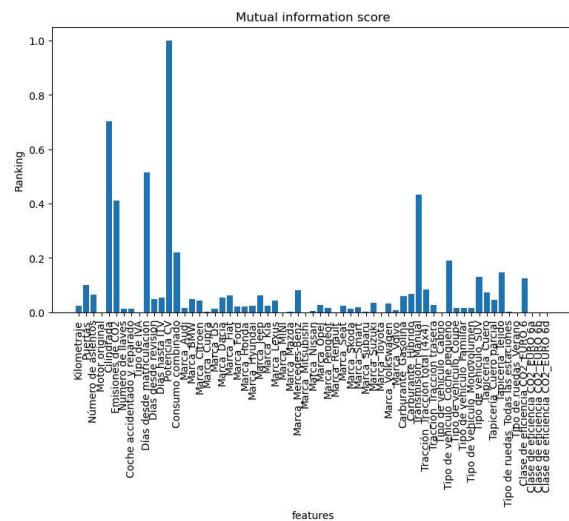
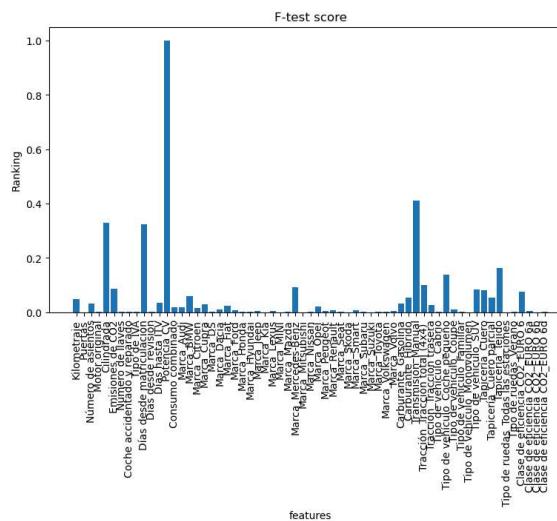
Veamos la influencia de variables

```
In [504]: datosVal = datos[datos['Validacion'] == 1].drop('Validacion', axis = 1)
datos = datos[datos['Validacion'] == 0].drop('Validacion', axis = 1)
X=datos.drop(['Precio'],axis=1)
y=datos['Precio']

f_test, _ = f_regression(X, y)
f_test /= np.max(f_test)
mi = mutual_info_regression(X, y)
mi /= np.max(mi)

feature_names = X.columns

plt.figure(figsize=(20, 5))
plt.subplot(1,2,1)
plt.bar(range(X.shape[1]), f_test, align="center")
plt.xticks(range(X.shape[1]), feature_names, rotation = 90)
plt.xlabel('features')
plt.ylabel('Ranking')
plt.title('F-test score')
plt.subplot(1,2,2)
plt.bar(range(X.shape[1]), mi, align="center")
plt.xticks(range(X.shape[1]), feature_names, rotation = 90)
plt.xlabel('features')
plt.ylabel('Ranking')
plt.title('Mutual information score')
plt.show()
```



podemos intentar seleccionar variables que tengan una puntuación alta en estos gráficos pero quizás nos queden pocas variables (solo la potencia pasa del 0.8)

## Modelos

En primer lugar separaremos la muestra en entrenamiento (70%) y test (30%). También separamos la validación

```
In [505]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

## Regresión lineal

```
In [506]: lm = LinearRegression()
lm.fit(X_train, y_train)
y_pred = lm.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = lm.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))

El ECM es 3975171.541549
El R^2 es 0.878886
```

In [507]: # Escalamos todas las variables al intervalo [0,1]

```
X=MinMaxScaler(feature_range=(0, 1)).fit_transform(datos.drop(['Precio'],axis=1))
X = pd.DataFrame(X, columns=datos.drop(['Precio'],axis=1).columns)
y=MinMaxScaler(feature_range=(0, 1)).fit_transform(datos['Precio'].array.reshape(-1,
y = pd.DataFrame(y, columns=['Precio']))
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0

lm = LinearRegression()
lm.fit(X_train, y_train)
y_pred = lm.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = lm.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))

El ECM es 0.002738
El R^2 es 0.878886
```

Este modelo es bueno con buen  $R^2$

In [508]: # Probamos a quitar Los coches caros de 2 puertas

```
datos = datos[~((datos['Puertas'] == 2) & (datos['Precio'] > 20000))]
X=MinMaxScaler(feature_range=(0, 1)).fit_transform(datos.drop(['Precio'],axis=1))
X = pd.DataFrame(X, columns=datos.drop(['Precio'],axis=1).columns)
y=MinMaxScaler(feature_range=(0, 1)).fit_transform(datos['Precio'].array.reshape(-1,
y = pd.DataFrame(y, columns=['Precio']))
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()
lm = LinearRegression()
lm.fit(X_train, y_train)
y_pred = lm.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = lm.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))

El ECM es 0.002905
El R^2 es 0.893594
```

```
In [509]: Xval = MinMaxScaler(feature_range=(0, 1)).fit(X).transform(datosVal.drop(['Precio'], axis=1))
Xval = pd.DataFrame(Xval, columns=X.columns)
yval = MinMaxScaler(feature_range=(0, 1)).fit(
    datos['Precio'].array.reshape(-1, 1)).transform(datosVal['Precio'].array.reshape(-1, 1))
yval = pd.DataFrame(yval, columns=['Precio'])
yval = yval.values.ravel()
```

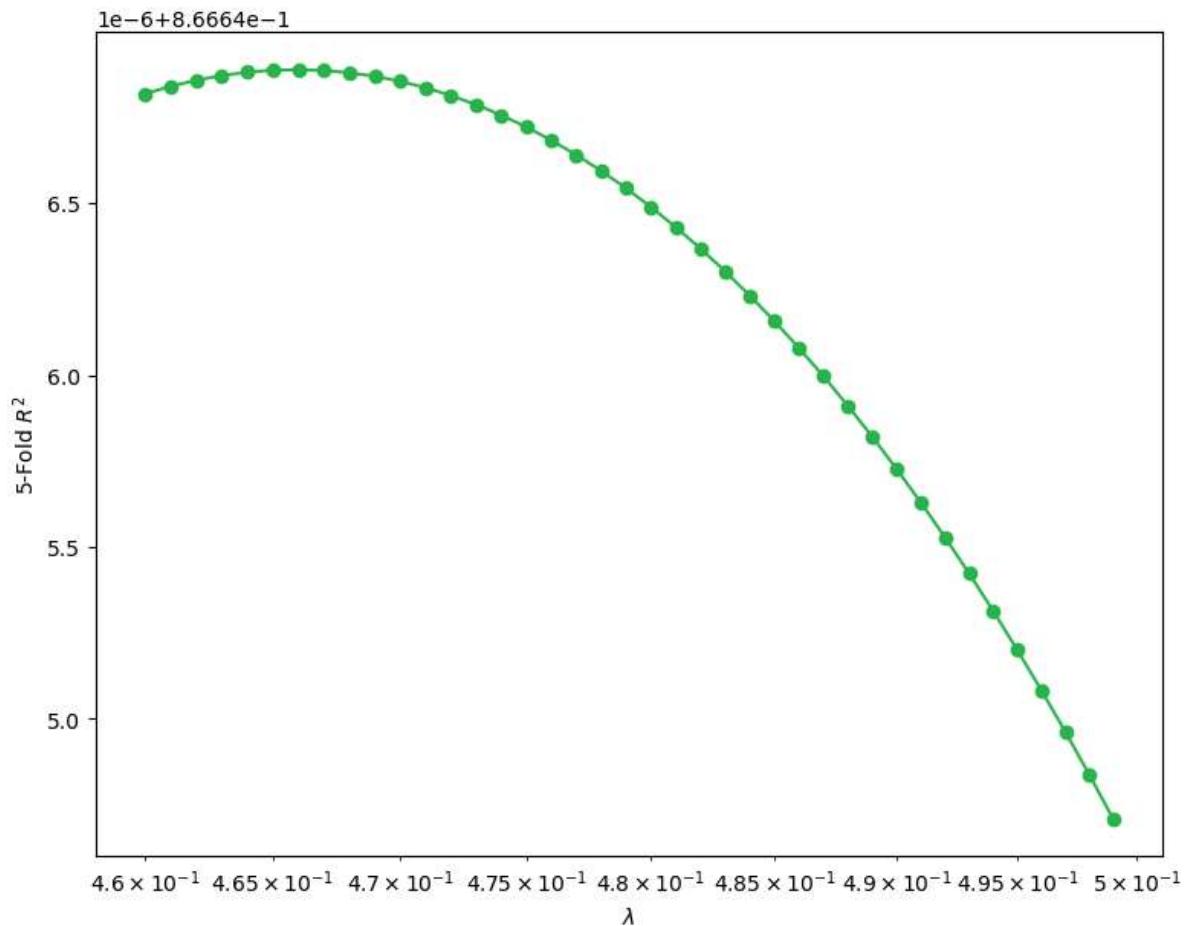
Nos quedamos mejor con el modelo sin los coches de 2 puertas que eran más caros

## Ridge regression

Intentamos optimizar el valor de  $\alpha$  que penaliza el error de estimación cometido

```
In [147]: lambda_vector = np.arange(0.46, 0.5, 0.001)
param_grid = {'alpha': lambda_vector}
grid = GridSearchCV(Ridge(random_state=0), param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.6f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))
scores = np.array(grid.cv_results_['mean_test_score'])
plt.figure(figsize=(9, 7))
plt.semilogx(lambda_vector, scores, '-o', color='#28b84f')
plt.xlabel('$\lambda$', fontsize=10)
plt.ylabel('5-Fold $R^2$')
plt.show()
```

best mean cross-validation score: 0.866647  
best parameters: {'alpha': 0.466}



```
In [146]: alpha_optimo = grid.best_params_['alpha']
ridge = Ridge(random_state=0,**grid.best_params_).fit(X_train, y_train)

y_pred = ridge.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = ridge.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))
```

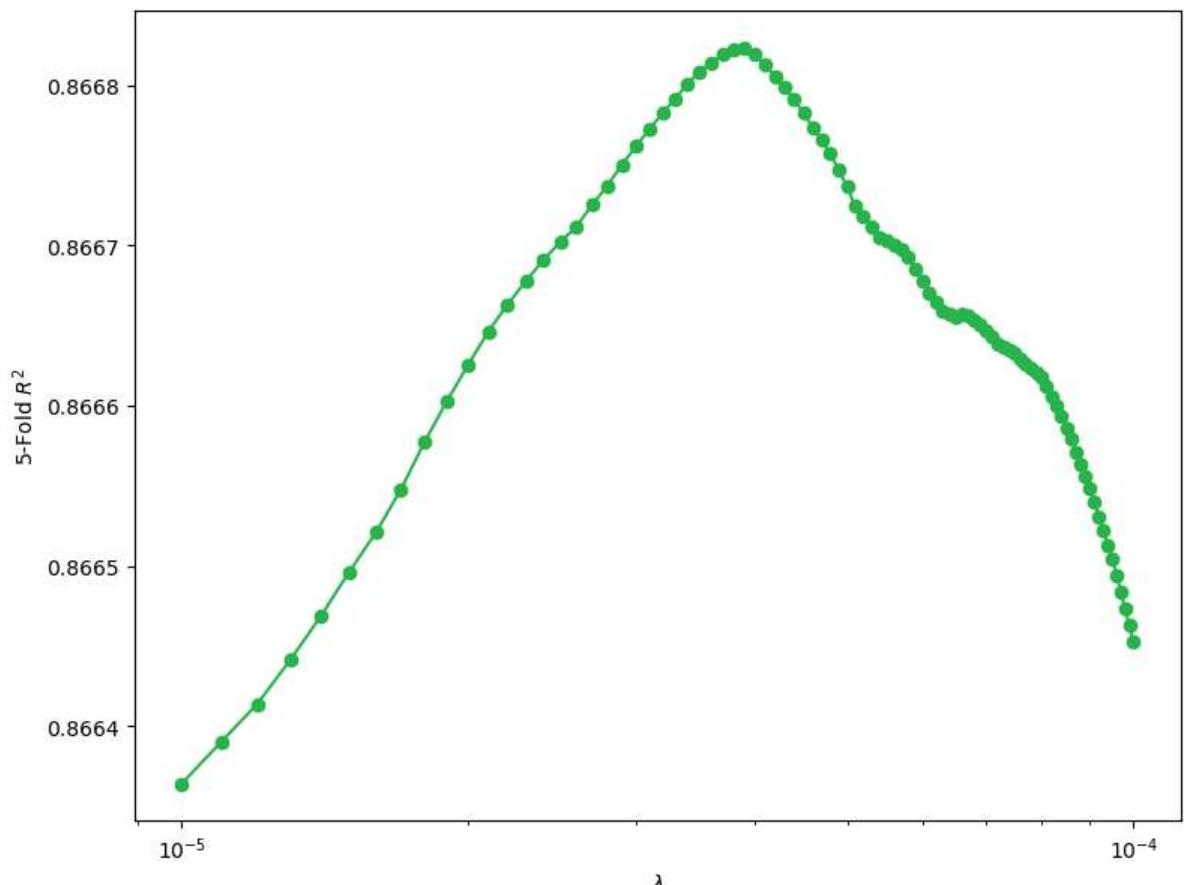
El ECM es 0.002887  
 El R<sup>2</sup> es 0.894233

Modelo con un  $R^2$  bastante bueno

## Regresión lasso

```
In [151]: # alpha_vector = np.logspace(0, 1, 100)
alpha_vector = np.arange(0.00001, 0.0001, 0.000001)
param_grid = {'alpha': alpha_vector}
grid = GridSearchCV(Lasso(random_state=0), param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.6f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))
scores = np.array(grid.cv_results_['mean_test_score'])
plt.figure(figsize=(9, 7))
plt.semilogx(alpha_vector, scores, '-o', color="#28b84f")
plt.xlabel('$\lambda$', fontsize=10)
plt.ylabel('5-Fold $R^2$')
plt.show()
```

best mean cross-validation score: 0.866824  
 best parameters: {'alpha': 3.9000000000002e-05}



```
In [152]: alpha_optimo = grid.best_params_['alpha']
lasso = Lasso(random_state=0,**grid.best_params_).fit(X_train, y_train)

y_pred = lasso.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = lasso.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))

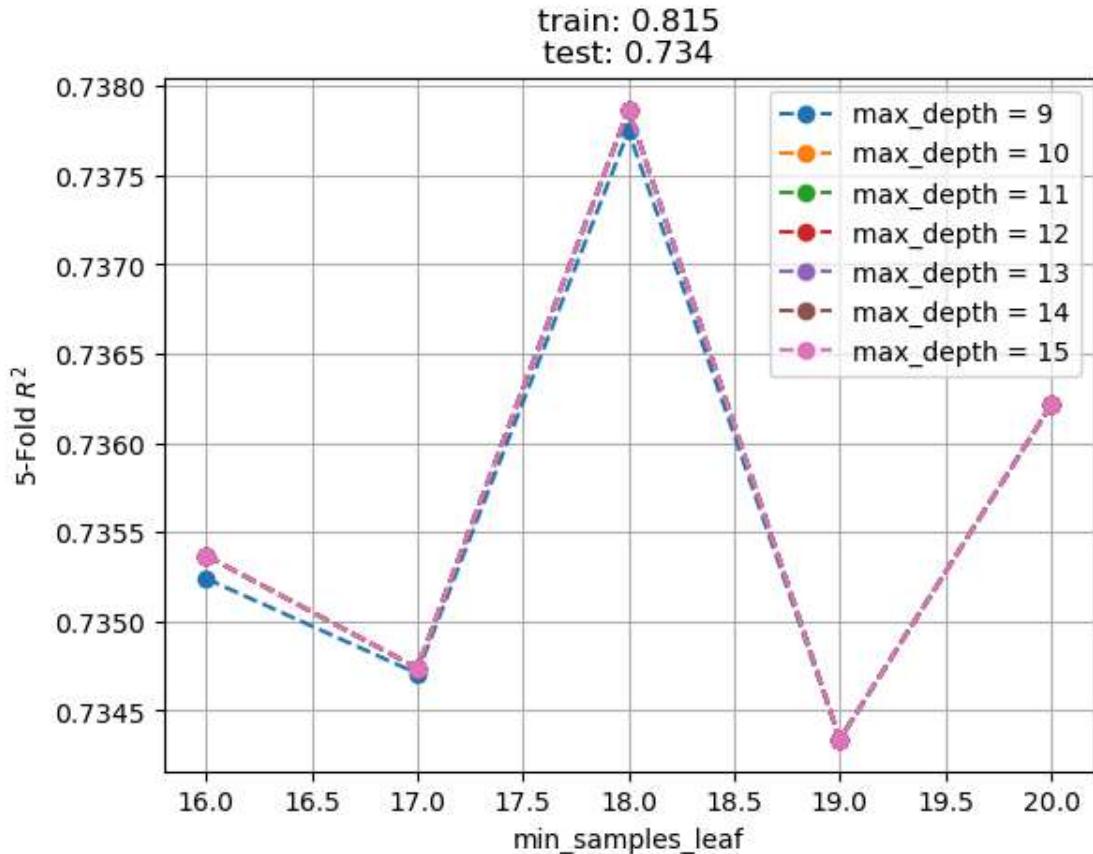
El ECM es 0.002860
El R^2 es 0.895222
```

## Árbol de decisión

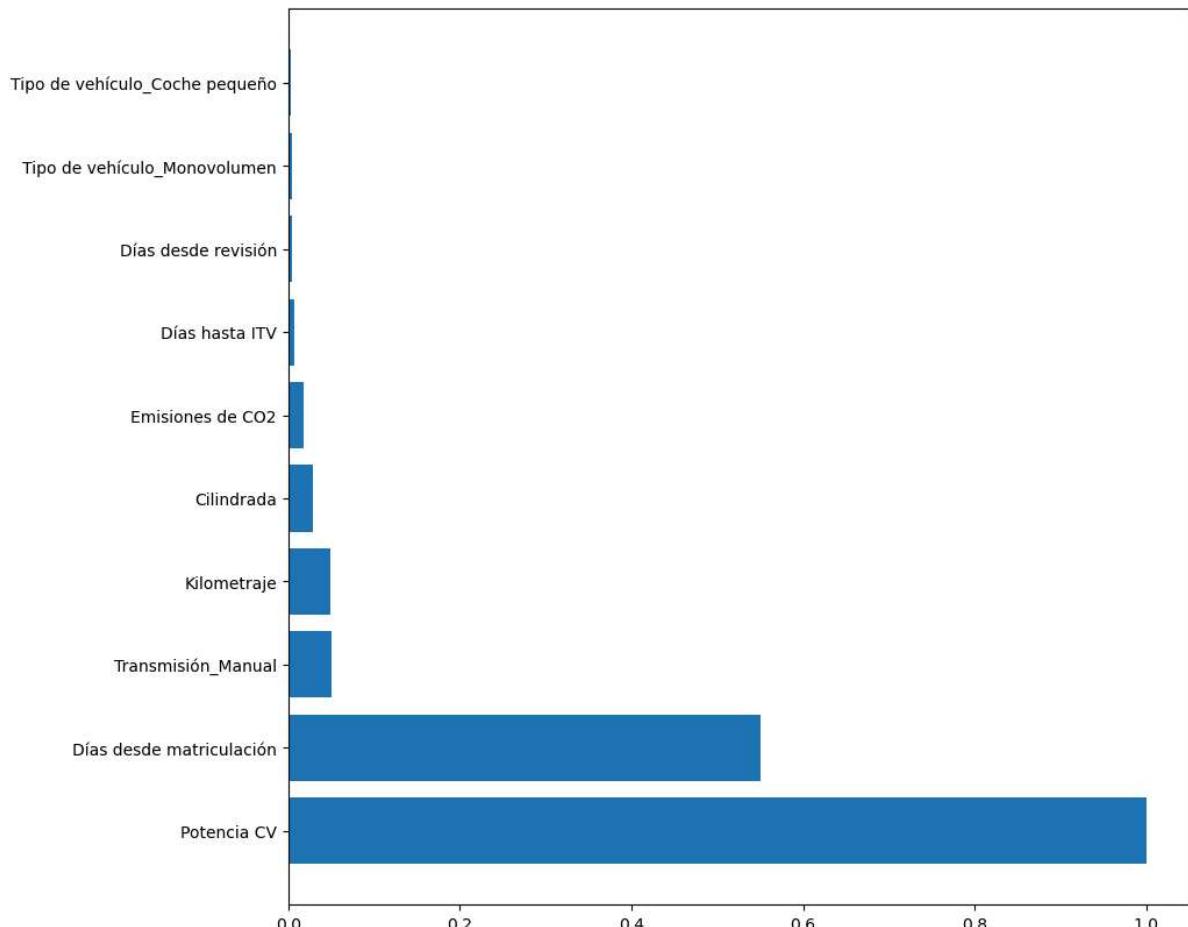
```
In [154]: # SELECCIÓN DEL MODELO
max_depth = np.arange(9, 16, 1)
min_samples_leaf = np.arange(16, 21, 1)
# rejilla de valores posibles que queremos explorar
param_grid = {'max_depth': max_depth, 'min_samples_leaf': min_samples_leaf}
# Validaremos el modelo mediante validación cruzada en n_folds
n_folds = 5
grid = GridSearchCV(DecisionTreeRegressor(random_state=0),
                     param_grid=param_grid, cv=5, return_train_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.6f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

best mean cross-validation score: 0.737868
best parameters: {'max_depth': 10, 'min_samples_leaf': 18}
```

```
In [156]: arboll = DecisionTreeRegressor(random_state=0,**grid.best_params_).fit(X_train, y_tr  
error = grid.cv_results_['mean_test_score'].reshape(len(max_depth),len(min_samples_1  
for i, lr in enumerate(list(max_depth)):  
    plt.plot(min_samples_leaf, error[i], '--o', label='max_depth = %g'%lr)  
plt.legend()  
plt.xlabel('min_samples_leaf')  
plt.ylabel('5-Fold $R^2$')  
plt.title('train: %0.3f\ntest: %0.3f'%(arboll.score(X_train, y_train), arboll.score(
```



```
In [179]: importances = arbol1.feature_importances_
importances = importances / np.max(importances)
indices = np.argsort(importances)[::-1]
indices = indices[:10]
plt.figure(figsize=(10,10))
plt.barh(range(X_train[X_train.columns[indices]].shape[1]),importances[indices])
plt.yticks(range(X_train[X_train.columns[indices]].shape[1]),X.columns[indices])
plt.show()
```



```
In [157]: y_pred = arbol1.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = arbol1.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))
```

El ECM es 0.007263  
 El R<sup>2</sup> es 0.733949

## Random forest

```
In [57]: # y_train = y_train.values.ravel()
# y_test = y_test.values.ravel()
```

```
In [233]: start = timeit.default_timer()

# SELECCIÓN DEL MODELO (model selection)
max_depth = np.arange(27, 50, 1)
min_samples_leaf = np.arange(2, 30, 1) SALE 2
n_estimators = [850,851,852]
#Tarda bastante
# rejilla de valores posibles que queremos explorar
# param_grid = {'max_depth': max_depth, 'min_samples_leaf': min_samples_leaf, 'n_estimators': n_estimators}
param_grid = {'max_depth': max_depth, 'n_estimators': n_estimators}
# Validaremos el modelo mediante validación cruzada en n_folds (este tarda bastante)
n_folds = 5
grid = GridSearchCV(RandomForestRegressor(random_state=0),
                     param_grid=param_grid, cv=5, return_train_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.4f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

stop = timeit.default_timer()
print('Time: ', stop - start)
# best mean cross-validation score: 0.8323
# best parameters: {'max_depth': 25, 'n_estimators': 400}
# Time: 7074.2864482

best mean cross-validation score: 0.8332
best parameters: {'max_depth': 28, 'n_estimators': 852}
Time: 4688.971457600128
```

Se podría intentar mejorar este modelo buscando otros parámetros. Vamos a fijar el max\_depth a 28 y buscamos min\_samples\_leaf y n\_estimators

```
In [227]: start = timeit.default_timer()

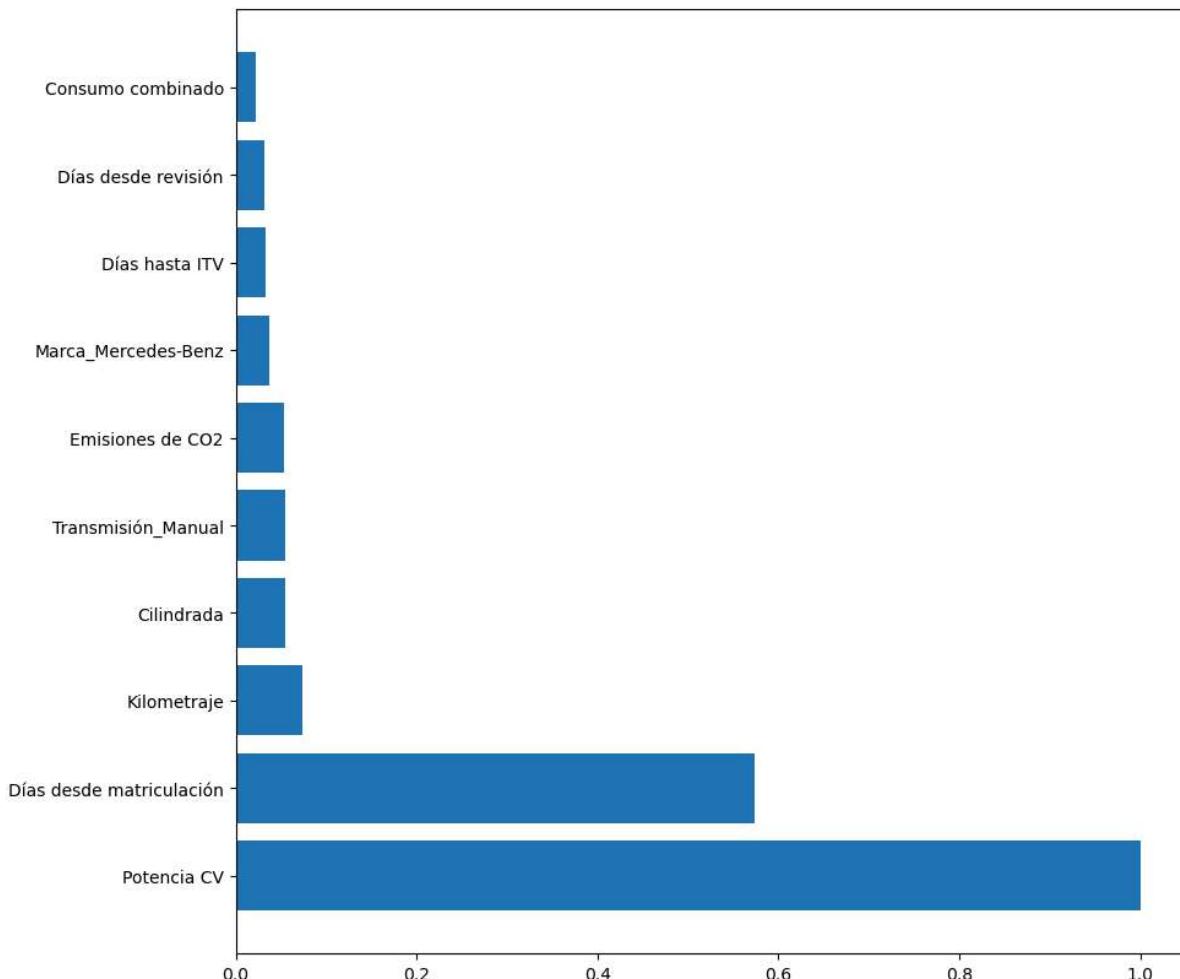
# SELECCIÓN DEL MODELO (model selection)
# max_depth = np.arange(25, 70, 1)
min_samples_leaf = np.arange(2, 30, 1)
n_estimators = [852,853,854]
# rejilla de valores posibles que queremos explorar
# param_grid = {'max_depth': max_depth, 'min_samples_leaf': min_samples_leaf, 'n_estimators': n_estimators}
param_grid = {'min_samples_leaf': min_samples_leaf, 'n_estimators': n_estimators}
# Validaremos el modelo mediante validación cruzada en n_folds (este tarda bastante)
n_folds = 5
grid = GridSearchCV(RandomForestRegressor(random_state=0,max_depth=28),
                     param_grid=param_grid, cv=5, return_train_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.4f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

stop = timeit.default_timer()
print('Time: ', stop - start)

best mean cross-validation score: 0.8267
best parameters: {'min_samples_leaf': 2, 'n_estimators': 600}
Time: 1458.6616035001352
```

```
In [159]: arbol2 = RandomForestRegressor(random_state=0, max_depth= 28,
                                         n_estimators = 852).fit(X_train, y_train)
```

```
In [180]: importances = arbol2.feature_importances_
importances = importances / np.max(importances)
indices = np.argsort(importances)[::-1]
indices = indices[:10]
plt.figure(figsize=(10,10))
plt.barh(range(X_train[X_train.columns[indices]].shape[1]),importances[indices])
plt.yticks(range(X_train[X_train.columns[indices]].shape[1]),X.columns[indices])
plt.show()
```



```
In [160]: y_pred = arbol2.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = arbol2.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))
```

El ECM es 0.004106  
El R<sup>2</sup> es 0.849605

## Boosted trees

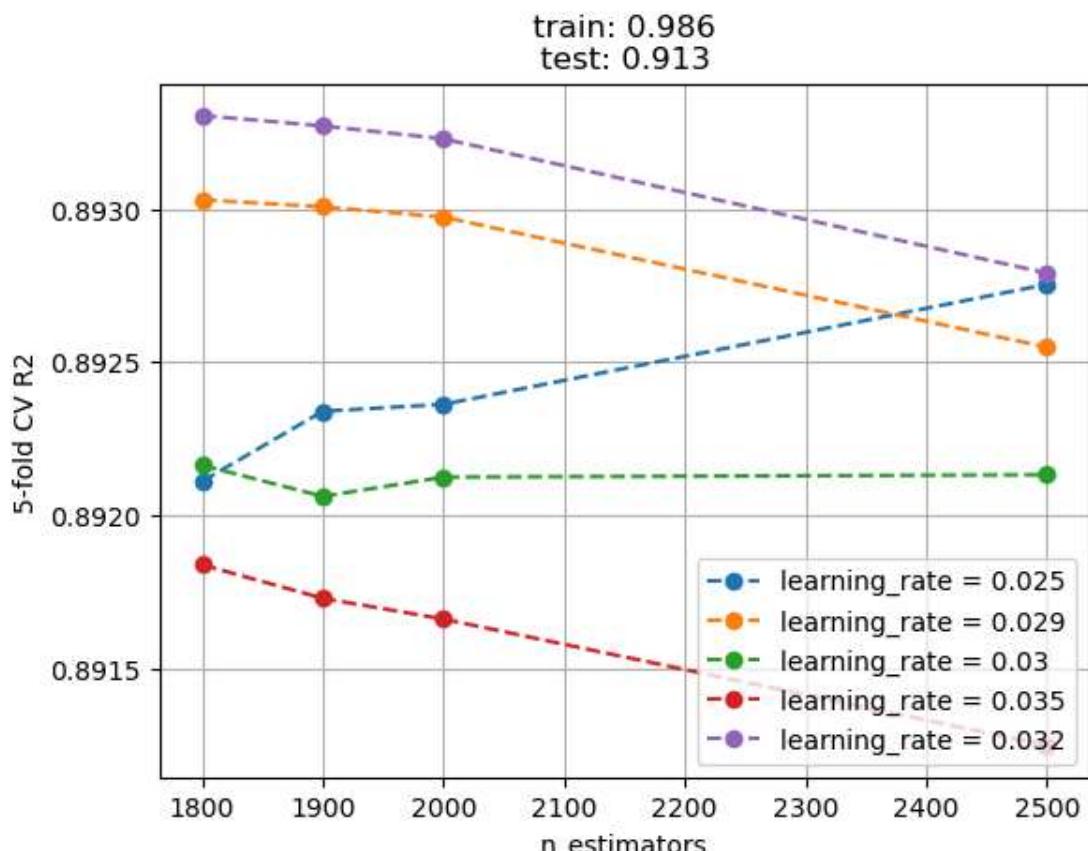
```
In [58]: start = timeit.default_timer()
# SELECCIÓN DEL MODELO (model selection)
n_estimators = [1800, 1900, 2000, 2500]
learning_rate = [0.025, 0.029, 0.03, 0.035, 0.032]
# rejilla de valores posibles que queremos explorar
param_grid = {'n_estimators': n_estimators, 'learning_rate': learning_rate}

# Validaremos el modelo mediante validación cruzada en n_folds (este tarda bastante)
n_folds = 5
grid = GridSearchCV(GradientBoostingRegressor(random_state=0),
                     param_grid=param_grid, cv=5, return_train_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.4f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

stop = timeit.default_timer()
print('Time: ', stop - start)

best mean cross-validation score: 0.8933
best parameters: {'learning_rate': 0.032, 'n_estimators': 1800}
Time: 748.1010269999997
```

```
In [59]: arbol3 = GradientBoostingRegressor(random_state=0, **grid.best_params_).fit(X_train,
error = grid.cv_results_[‘mean_test_score’].reshape(len(learning_rate), len(n_estimators))
for i, lr in enumerate(list(learning_rate)):
    plt.plot(n_estimators, error[i], ‘--o’, label=‘learning_rate = %g’%lr)
plt.legend()
plt.xlabel(‘n_estimators’)
plt.ylabel(‘{}-fold CV R2’.format(n_folds))
plt.title(‘train: {:.3f}\ntest: {:.3f}’%(arbol3.score(X_train, y_train), arbol3.score(
plt.grid()
plt.show()
```



Vamos a fijar n\_estimators en 1800, ya que parece que a partir de ahí no mejoramos mucho el resultado

```
In [55]: start = timeit.default_timer()
# SELECCIÓN DEL MODELO (model selection)
# n_estimators = [1800, 1900, 2000, 2500]
learning_rate = np.arange(0.03, 0.04, 0.001)
max_depth = np.arange(2, 10, 1)
# rejilla de valores posibles que queremos explorar
param_grid = {'max_depth': max_depth, 'learning_rate': learning_rate}

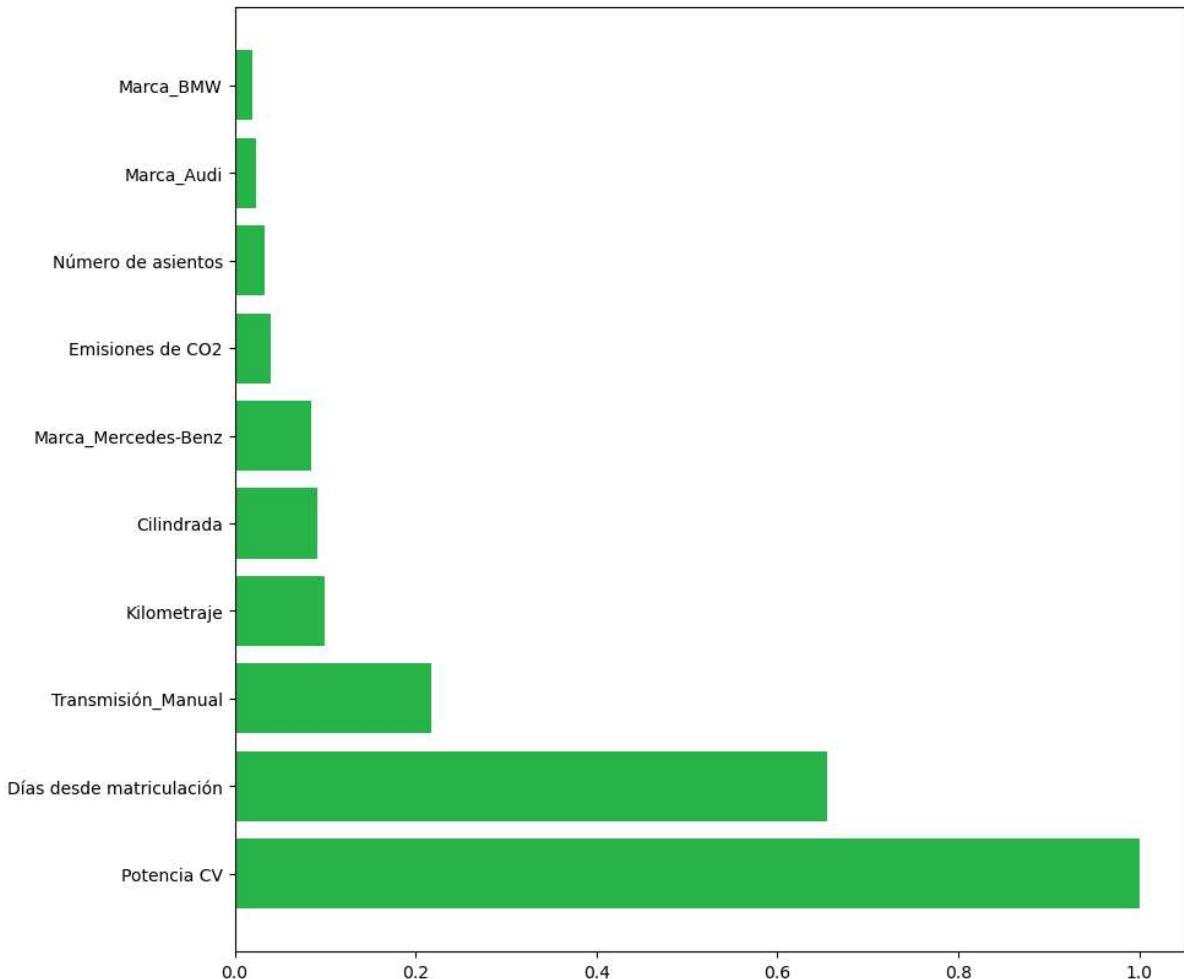
# Validaremos el modelo mediante validación cruzada en n_folds (este tarda bastante)
n_folds = 5
grid = GridSearchCV(GradientBoostingRegressor(random_state=0, n_estimators = 1800),
                     param_grid=param_grid, cv=5, return_train_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.4f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

stop = timeit.default_timer()
print('Time: ', stop - start)

best mean cross-validation score: 0.9002
best parameters: {'learning_rate': 0.037000000000000005, 'max_depth': 2}
Time:  5406.2887801998295
```

```
In [510]: arbol3 = GradientBoostingRegressor(random_state=0, learning_rate = 0.037000000000000005,
                                             max_depth=2,n_estimators = 1800).fit(X_train, y_t
```

```
In [511]: importances = arbol3.feature_importances_
importances = importances / np.max(importances)
# pos = np.where(importances>0.01)
# importances = importances[pos]
indices = np.argsort(importances)[::-1]
indices = indices[:10]
plt.figure(figsize=(10,10))
plt.barh(range(X_train[X_train.columns[indices]].shape[1]),importances[indices],color='green')
plt.yticks(range(X_train[X_train.columns[indices]].shape[1]),X.columns[indices])
plt.show()
```



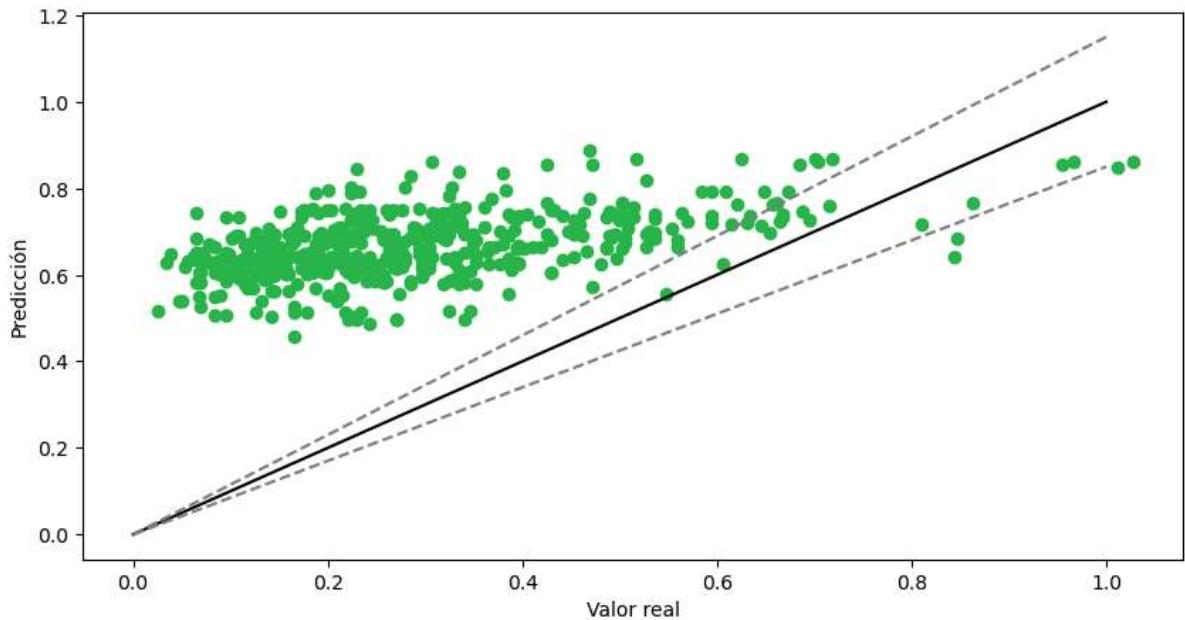
```
In [512]: y_pred = arbol3.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = arbol3.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))
```

El ECM es 0.002315  
 El R<sup>2</sup> es 0.915194

Este último modelo es el mejor por ahora

```
In [513]: y_pred = arbol3.predict(Xval)
mse_test = np.mean((yval-y_pred)**2)
score = arbol3.score(Xval, yval)
```

```
In [514]: plt.figure(figsize=(10,5))
plt.plot(yval, y_pred, marker='o', linestyle = '', color = '#28b84f')
plt.plot((0, 1), (0, 1), linestyle='-', color = 'black')
plt.plot((0, 1), (0, 0.85), linestyle='--', color = 'grey')
plt.plot((0, 1), (0, 1.15), linestyle='--', color = 'grey')
plt.xlabel('Valor real')
plt.ylabel('Predicción')
plt.show()
```



```
In [250]: error = y_test-y_pred
acerto = abs(error/y_test)<= 0.15
```

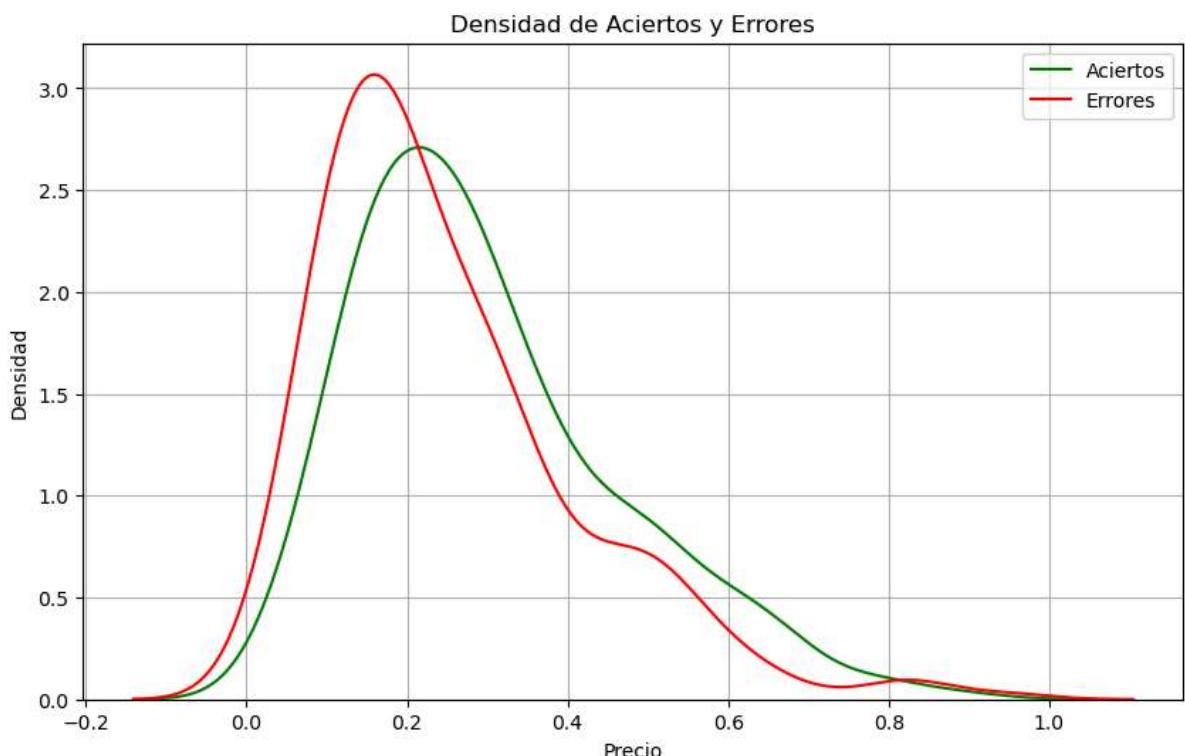
```
In [244]: plt.figure(figsize=(10, 6))

# Densidad de los aciertos
sns.kdeplot(y_test[acerto], color='green', label='Aciertos', fill=False)

# Densidad de los errores
sns.kdeplot(y_test[~acerto], color='red', label='Errores', fill=False)

# Añade etiquetas y título
plt.xlabel('Precio')
plt.ylabel('Densidad')
plt.title('Densidad de Aciertos y Errores')
plt.legend()
plt.grid(True)

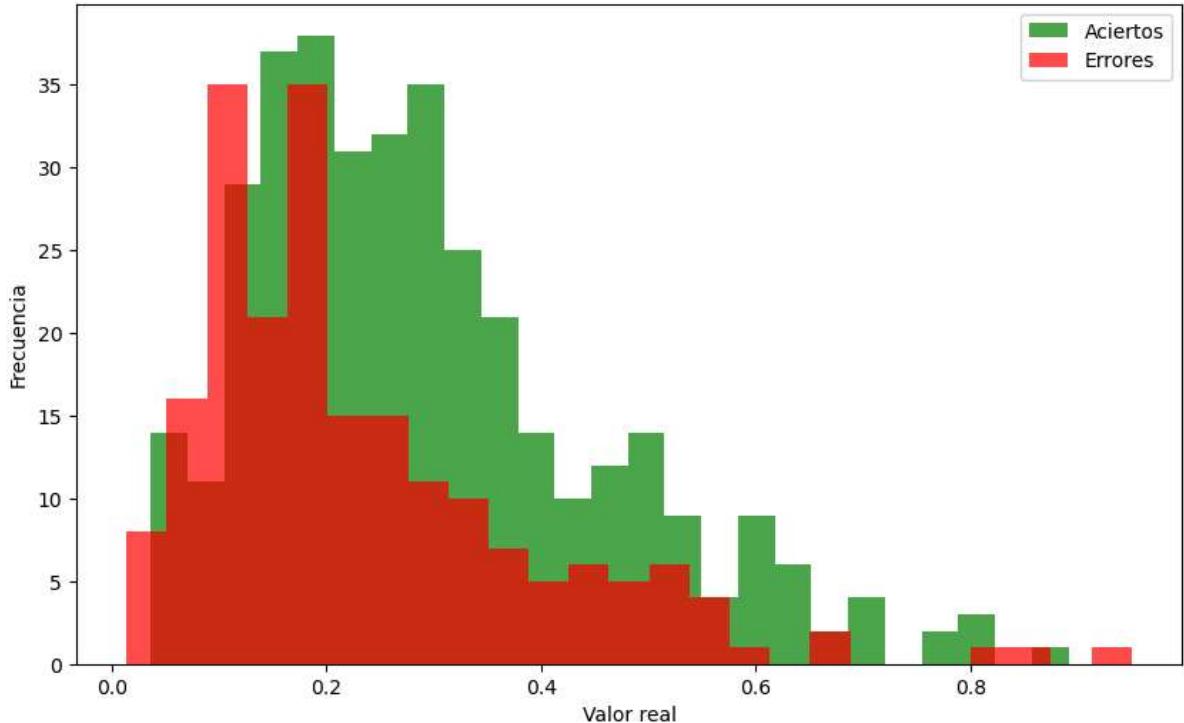
# Muestra el gráfico
plt.show()
```



```
In [258]: plt.figure(figsize=(10, 6))
plt.hist(y_test[acerto], bins=25, color='green', alpha=0.7, label='Aciertos')
plt.hist(y_test[~acerto], bins=25, color='red', alpha=0.7, label='Errores')

plt.xlabel('Valor real')
plt.ylabel('Frecuencia')
plt.legend()

plt.show()
```



## Máquinas de vectores soporte (SVM)

```
In [81]: start = timeit.default_timer()

# C = np.logspace(5, 6, 5)
# gamma = np.logspace(-5, -3, 5)

C = np.arange(197499, 197509, 1)
gamma = np.arange(0.00002, 0.000032, 0.000002)

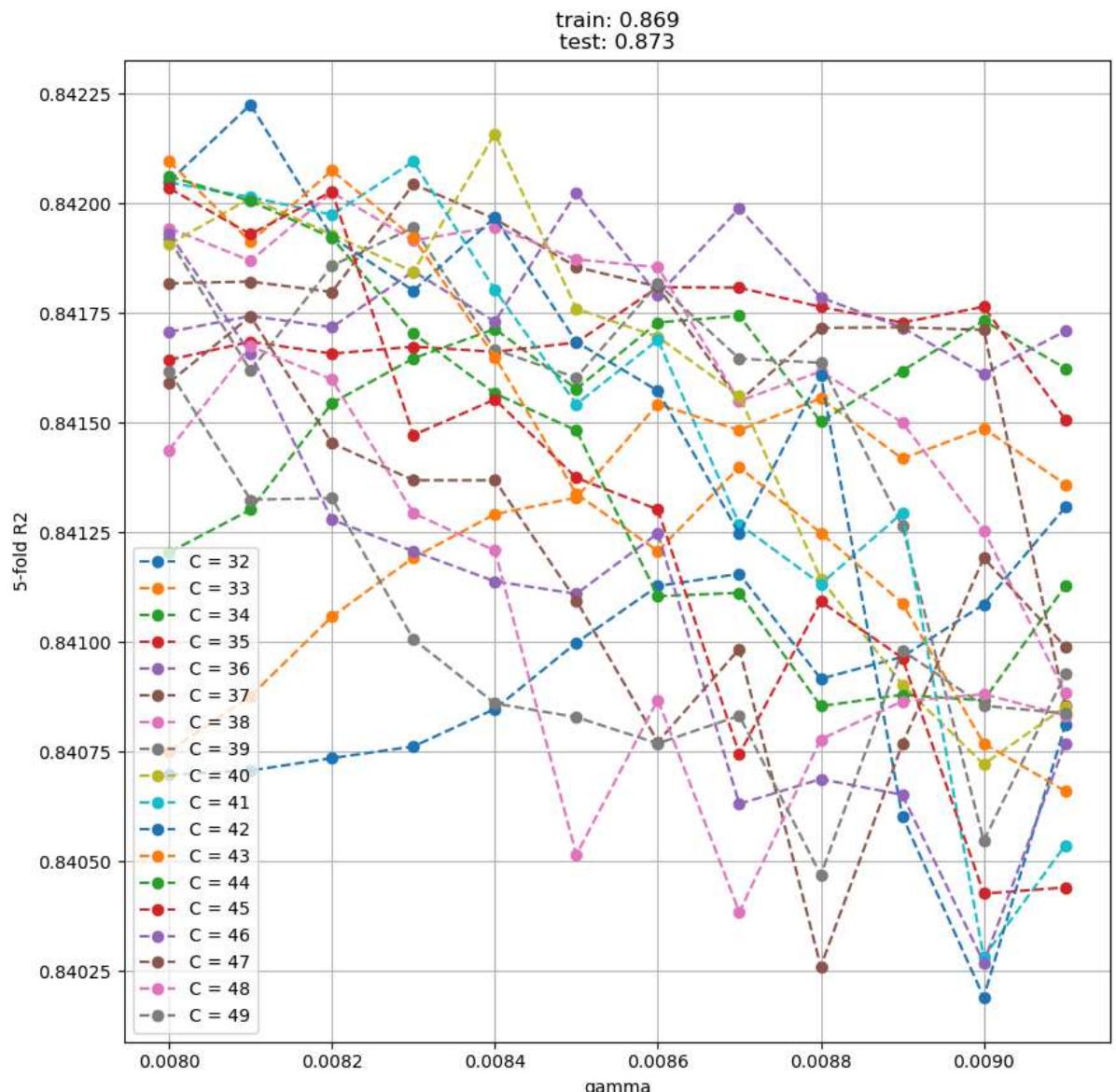
grid = GridSearchCV(SVR(kernel = 'rbf'),
                     param_grid={'C': C, 'gamma': gamma},
                     cv = 5, return_train_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.6f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

stop = timeit.default_timer()
print('Time: ', stop - start)
# best mean cross-validation score: 0.871267
# best parameters: {'C': 197500, 'gamma': 3.000000000000004e-05}
# Time: 113.28513589990325

best mean cross-validation score: 0.871267
best parameters: {'C': 197500, 'gamma': 3.000000000000008e-05}
Time: 265.0443410000298
```

```
In [77]: svm = SVR(kernel = 'rbf', **grid.best_params_).fit(X_train, y_train)
```

```
In [124]: error = grid.cv_results_['mean_test_score'].reshape(len(C),len(gamma))
plt.figure(figsize=(10,10))
for i, lr in enumerate(list(C)):
    plt.plot(gamma, error[i], '--o', label='C = %g'%lr)
plt.legend()
plt.xlabel('gamma')
plt.ylabel('{}-fold R2'.format(5))
plt.title('train: {:.3f}\ntest: {:.3f}'.format(svm.score(X_train, y_train), svm.score(X_test,
plt.grid()
plt.show()
```



```
In [80]: y_pred = svm.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print('El ECM es {:.6f}'.format(mse_test))
score = svm.score(X_test, y_test)
print('El R^2 es {:.6f}'.format(score))
```

El ECM es 0.003434  
El R<sup>2</sup> es 0.874199

```
In [121]: start = timeit.default_timer()

# C = np.Logspace(5,6, 5)
# gamma = np.Logspace(-5,-3, 5)
# C = np.arange(1068, 1075, 1)
C = np.arange(32, 50, 1)
gamma = np.arange(0.0080, 0.0091, 0.0001)

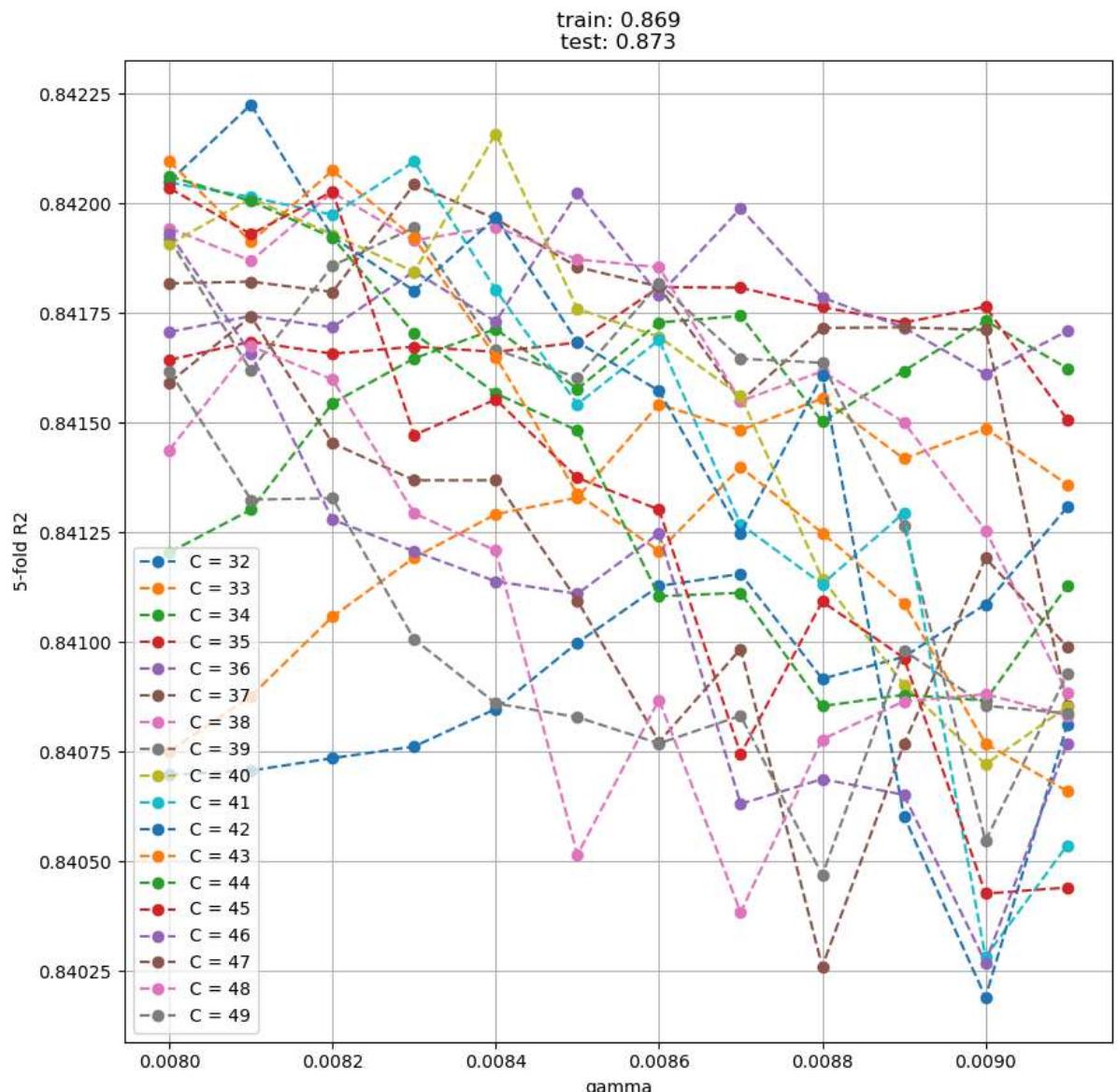
# gamma = [0.008,0.009,0.01,0.011]

grid = GridSearchCV(SVR(kernel = 'sigmoid'),
                     param_grid={'C': C, 'gamma': gamma},
                     cv = 5, return_train_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.6f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

stop = timeit.default_timer()
print('Time: ', stop - start)
# best mean cross-validation score: 0.841872
# best parameters: {'C': 38, 'gamma': 0.0085}
# Time:  23.2498943998944
```

```
best mean cross-validation score: 0.842225
best parameters: {'C': 42, 'gamma': 0.0081}
Time:  36.174323400016874
```

```
In [126]: svm = SVR(kernel = 'sigmoid', **grid.best_params_).fit(X_train, y_train)
error = grid.cv_results_[‘mean_test_score’].reshape(len(C),len(gamma))
plt.figure(figsize=(10,10))
for i, lr in enumerate(list(C)):
    plt.plot(gamma, error[i], ‘--o’, label=‘C = %g’%lr)
plt.legend()
plt.xlabel(‘gamma’)
plt.ylabel(‘{}-fold R2’.format(5))
plt.title(‘train: {:.3f}\ntest: {:.3f}’%(svm.score(X_train, y_train), svm.score(X_test,
plt.grid()
plt.show()
```



```
In [123]: y_pred = svm.predict(X_test)
mse_test = np.mean((y_test-y_pred)**2)
print(‘El ECM es {:.6f}’ .format(mse_test))
score = svm.score(X_test, y_test)
print(‘El R^2 es {:.6f}’ .format(score))
```

El ECM es 0.003461  
El R<sup>2</sup> es 0.873229

In [ ]:

