U02 PRÁCTICA 3

ALGORITMOS DE REGRESIÓN Y MÉTODOS DE TRABAJO

SISTEMAS DE APRENDIZAJE AUTOMÁTICO

IES SERRA PERENXISA
TORRENT (VALENCIA)

ALGORITMOS DE REGRESIÓN

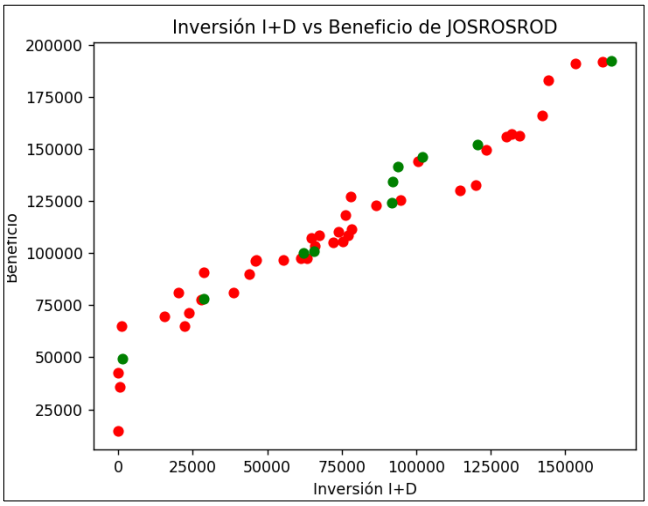
📌ACTIVIDAD 1: REPASAR ALGORITMOS

REGRESIÓN LINEAL SIMPLE

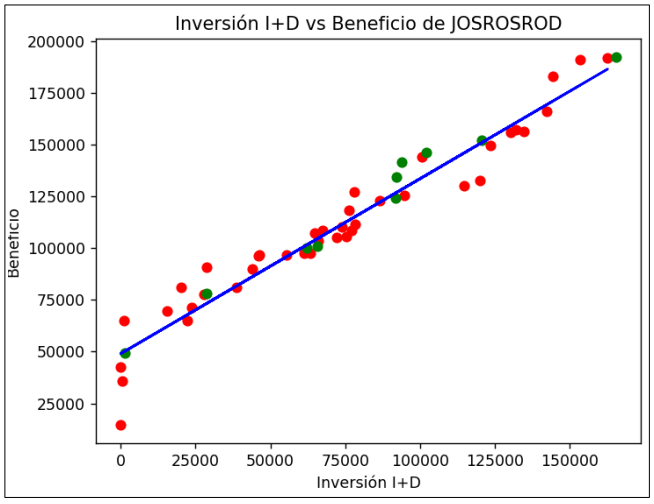
Crea el fichero ***u02_p03_a1_reglin_simple_<tus_iniciales>.py*** o bien ***.ipynb.*** Utiliza pandas y carga los datos del fichero ***"50_startups.csv"***. Utilizaremos como predictora la columna ***"R&D Spend"*** (que significa gasto en I+D) y como target usaremos ***"Profit"*** (beneficios).

Como todo es un proceso aleatorio, para poder obtener resultados comparables vamos a utilizar la semilla "123" en todos los procesos. Divide los datos en train y test dejando el 80% para entrenamiento.

Haz un gráfico ***scatterplot()*** de los datos, con los datos de entrenamiento como círculos de color rojo y los de test círculos de color verde. Pon leyenda a cada variable y título ***"Inversión I+D vs Beneficio de <tus iniciales>".***



Antes del código que hace el gráfico crea un objeto ***LinearRegresion()*** y lo entrenas. Añade al gráfico la línea recta de predicción a todos los datos de test en color azul.



Calcula su ***score*** (coeficiente de determinación R^2) para datos de *train* y *test*. ¿Es un buen modelo?

```
Train R2: 0.9387391481701634
Test R2:  0.9728849912273875
```

ENTREGA 1: Muestra:

a) Capturas de ejecución y el código.

b) ¿Qué significa en este caso que el R^2 sea 0.97?

REGRESIÓN LINEAL MÚLTIPLE

Crea el fichero ***u02_p03_a1_reglin_multi_<tus_iniciales>.py.*** Utiliza pandas y carga los datos del fichero ***"50_startups.csv"***. Utilizaremos como predictoras las columnas ***"R&D Spend"***, ***"Marqueting Spend"*** y ***"State"*** y como target usaremos ***"Profit"*** (beneficios). Imprime los 5 primeros ejemplos de las predictoras:

```
Primeros 5 ejemplos:
 [[165349.2 471784.1 'New York']
 [162597.7 443898.53 'California']
 [153441.51 407934.54 'Florida']
 [144372.41 383199.62 'New York']
 [142107.34 366168.42 'Florida']]
```

Para realizar regresión lineal no podemos usar una columna categórica como ***"State"***. Codifica sus valores usando el método *one-hot-encoder* de manera que aparecerán 3 nuevas columnas que corresponden a cada uno de los valores que puede tener la columna original (*'New York'*, *'California'*, *'Florida'*) y sus valores estarán a 1 cuando sea ese valor y a 0 cuando no. En vez de hacerlo a mano, vamos a utilizar los objetos ***sklearn.compose.ColumnTransformer*** y ***sklearn.preprocessing.OneHotEncoder***. Busca información de como hacerlo. Tras realizarlo imprime las primeras 5 filas de los datos transformados.

```
Después de aplicar OneHotEncoder a la característica 'State':
 [[0.0 0.0 1.0 165349.2 471784.1]
 [1.0 0.0 0.0 162597.7 443898.53]
 [0.0 1.0 0.0 153441.51 407934.54]
 [0.0 0.0 1.0 144372.41 383199.62]
 [0.0 1.0 0.0 142107.34 366168.42]]
```

Como todo es un proceso aleatorio, para poder obtener resultados comparables utilizamos la semilla "123" en todos los procesos. Divide los datos en train y test dejando el 80% para entrenamiento. Imprime los 5 primeros ejemplos de ***X_train***, ***y_train***, ***X_test*** e ***y_test***:

```
X_train 5 primeras filas
 [[0.0 0.0 1.0 78389.47 299737.29]
 [1.0 0.0 0.0 0.0 0.0]
 [1.0 0.0 0.0 100671.96 249744.55]
 [0.0 1.0 0.0 27892.92 164470.71]
 [0.0 0.0 1.0 131876.9 362861.36]]
```

```
y_train 5 primeras filas
 [[111313.02]
 [ 42559.73]
 [144259.4 ]
 [ 77798.83]
 [156991.12]]
```

```
X_test 5 primeras filas
 [[0.0 0.0 1.0 78389.47 299737.29]
 [1.0 0.0 0.0 0.0 0.0]
 [1.0 0.0 0.0 100671.96 249744.55]
 [0.0 1.0 0.0 27892.92 164470.71]
 [0.0 0.0 1.0 131876.9 362861.36]]
```

```
y_test 5 primeras filas
 [[111313.02]
 [ 42559.73]
 [144259.4 ]
 [ 77798.83]
 [156991.12]]
```

Crea un objeto ***sklearn.linear_model.LinearRegression*** y lo entrenas. Calcula su ***score*** para datos de *train* y test.

```
Score del modelo en train 0.944872235399549 y en test 0.9657541989322397
```

ENTREGA 1: Muestra:

a) Capturas de ejecución y el código.

b) ¿Qué significa en este caso que el *score* sea aproximadamente del `0.966` en los datos de test?

Ahora mira los apuntes de la unidad y como se calcula el ***SSE***, ***SST*** y ***SSR***, R^2 y R^2 ***ajustado***. Define 3 funciones de Python que calcule el el ***SSE***, ***SST*** y ***SSR***. Calcula e imprime sus valores para los datos de train y test. Debes obtener:

```
SST train=63300608353.770935 SST test=15071605761.115088
SSR train=3489621036.392024 SSR test=516139212.6668584
SSE train=59810987317.36765 SSE test=13307210659.677673
R2 de train:0.944872235399549 R2 de test:0.9657541989322397
R2 Ajustado de train:0.8816374465931494 R2 ajustado de test:0.9264722506486323
```

Ahora queremos saber si añadiendo la característica ***"Administration"*** (gastos de administración) podríamos mejorar el modelo. Para ello volvemos a generar un array de *numpy* ***X*** con los datos de las características ***"R&D Spend"***, ***"Administration"***, ***"Marqueting Spend"*** y ***"State"*** y en **y** los valores de ***"profit"***. volvemos a codificar la columna ***"State"***. Volvemos a particionar, crear y entrenar el modelo

de la misma manera que antes (para que use los mismos datos, inicializamos el proceso al mismo valor aleatorio). Volvemos a imprimir el ***score***:

```
Score del modelo en train 0.9449965926433526 y en test 0.9667998486974319
```

Calcula y muestra el ***SSE***, ***SST***, ***SSR***, ***R^2*** y ***R^2 ajustado***.

```
SST train=63300608353.770935 SST test=15071605761.115088
SSR train=3481749147.2060595 SSR test=500379591.6416779
SSE train=59818859206.56135 SSE test=13329973691.58707
R2 de train:0.9449965926433526 R2 de test:0.9667998486974319
R2 Ajustado de train:0.8799925657673147 R2 ajustado de test:0.9275633062489422
```

ENTREGA 2: Muestra:

a) Capturas de ejecución y el código.
b) Haz una marca donde aumente el R^2: [_]Train [_]Test
c) Haz una marca donde aumente del R^2 ajustado: [_]Train [_]Test
d) ¿Dónde aparece mayor mejora? En el R^2 o en el R^2 ajustado.
e) ¿Marca lo que calcula ***score()*** en *scikit-learn*? ***(_)SSE (_)SST (_)SSR (_)R^2 (_)R^2 ajustado***

K-NN

Como tenemos pocos datos vamos a ver como se comportará un ***K-NN***, por ejemplo un ***3-NN***. A continuación del fichero, añade las sentencias para crear uno y muestra su *score* para los datos de test tras entrenarlo. Usaremos todas las columnas de predictoras:

```
Score de 3-NN: 0.7108530068221302
```

ENTREGA 3: Muestra:

a) Capturas de ejecución y el código.
b) Valor del hiperparámetro K que mejore el score alcanzado con K=3.

📌ACTIVIDAD 2: UN CASO MÁS REAL.

Crea el fichero ***u02_p03_a2_reglin_<tus_iniciales>.py.*** Vamos a trabajar con los datos del fichero ***"seguros_de_coches.csv"*** que utiliza una compañía aseguradora**.**

COMPRENDER LOS DATOS DEL DATASET

Hay 3 tipos de características:

(a) La especificación de un automóvil en términos de diferentes características.

(b) Su calificación de riesgo de seguro asignada. Es un indicador del grado en que el automóvil es más problemático de lo que indica su precio. Inicialmente, a los automóviles se les asigna un símbolo de factor de riesgo asociado con su precio. Luego, si es más arriesgado (o menos), este símbolo se ajusta moviéndolo hacia arriba (o hacia abajo) en la escala. Los profesionales llaman a este proceso ***"symboling"***. Un valor de 3 indica que el automóvil es riesgoso y -3 que probablemente sea bastante seguro.

(c) El pago medio relativo por pérdida por año de vehículo asegurado. Este valor está normalizado para todos los automóviles dentro de una clasificación de tamaño particular (*two-door small, station wagons, sports/speciality*, etc…) y representa la pérdida promedio por automóvil por año.

Hay que tener en cuenta que los valores ausentes están definidos con un ***"?"***. Completa los siguientes pasos en el fichero Python comenzando por:

a) Cargar los datos en un ***Dataframe***, (adapta la ruta al código de la siguiente figura).
b) Mostrar los 5 primeros ejemplos por pantalla (completa y obtén mismos resultados).
c) Muestra un resumen de las columnas del dataset (completa y obtén mismos resultados).

```python
#_*_coding: utf-8 _*_
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

columnas = ['symboling',        'perdidas_normal', 'marca',    'combustible',       'inyección',
            'puertas',          'chasis',          'traccion', 'lugar_de_motor',    'potencia_base',
            'longitud',         'anchura',         'altura',   'peso',              'tipo_motor',
            'cilindros',        'tamaño_motor',    'sistema',  'calibre',           'ataque',
            'ratio_compresion', 'potencia_cv',     'max_rpm',  'consumo_carretera', 'consumo_ciudad',
            'precio']
autos = pd.read_csv("seguros_de_coches.csv", header=None, names=columnas, na_values='?', delimiter=",")
```

```
Primeros 4 ejemplos:
    symboling  perdidas_normal       marca combustible inyección  ... potencia_cv max_rpm consumo_carretera consumo_ciudad  precio
0          3              NaN  alfa-romeo         gas       std  ...       111.0  5000.0                21             27   13495
1          3              NaN  alfa-romeo         gas       std  ...       111.0  5000.0                21             27   16500
2          1              NaN  alfa-romeo         gas       std  ...       154.0  5000.0                19             26   16500
3          2            164.0        audi         gas       std  ...       102.0  5500.0                24             30   13950

[4 rows x 26 columns]
```

```
Tamaño del dataset: (201, 26)
```

```
========== Resumen:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          201 non-null    int64
 1   perdidas_normal    164 non-null    float64
 2   marca              201 non-null    object
 3   combustible        201 non-null    object
 4   inyección          201 non-null    object
 5   puertas            199 non-null    object
 6   chasis             201 non-null    object
 7   traccion           201 non-null    object
 8   lugar_de_motor     201 non-null    object
 9   potencia_base      201 non-null    float64
 10  longitud           201 non-null    float64
 11  anchura            201 non-null    float64
 12  altura             201 non-null    float64
 13  peso               201 non-null    int64
 14  tipo_motor         201 non-null    object
 15  cilindros          201 non-null    object
 16  tamaño_motor       201 non-null    int64
 17  sistema            201 non-null    object
 18  calibre            197 non-null    float64
 19  ataque             197 non-null    float64
 20  ratio_compresion   201 non-null    float64
 21  potencia_cv        199 non-null    float64
 22  max_rpm            199 non-null    float64
 23  consumo_carretera  201 non-null    int64
 24  consumo_ciudad     201 non-null    int64
 25  precio             201 non-null    int64
dtypes: float64(10), int64(6), object(10)
memory usage: 41.0+ KB
```

DATOS AUSENTES Y CATEGÓRICOS

Los datos se han cargado remplazando los símbolos ***"?"*** como valores ausentes ***NaN***. como estaba definido en la información del dataset. Primero sumarizamos los datos ausentes por característica. Luego nos podemos fijar en el listado que nos ha devuelto el método ***autos.info()*** y todas las que no tengan un tipo numérico (***intX*** y ***floatX***) serán categóricas de una u otra forma.

```python
# Valores ausentes y columnas categóricas
print("========== Valores ausentes:\n", autos.isna().sum())
cols_categoricas = ["marca", "sistema"]                    # Completa el array!!!
autos[cols_categoricas] = autos[cols_categoricas].astype("category")
autos["puertas"] = pd.Categorical(autos["puertas"], categories=["two","four"], ordered=True)
autos["cilindros"] = pd.Categorical(autos["cilindros"],
                                    categories=["two", "three", "four", "five", "six", "eight","twelve"],
                                    ordered=True)
print("========== Cambiamos a tipo category:\n",autos.info())
```

Para corregirlas podemos crearnos un array con sus nombres. Completa el array ***cols_categoricas*** de la figura con el resto de características categóricas (ahora solamente aparecen la primera y la última). Usamos este array para cambiarles el tipo de dato. Y a dos características ordinales le indicamos que lo son. Por último mostramos información del dataframe. Deberías obtener algo similar a esto como salida.

```
========== Cambiamos a tipo category:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          201 non-null    int64
 1   perdidas_normal    164 non-null    float64
 2   marca              201 non-null    category
 3   combustible        201 non-null    category
 4   inyección          201 non-null    category
 5   puertas            199 non-null    category
 6   chasis             201 non-null    category
 7   traccion           201 non-null    category
 8   lugar_de_motor     201 non-null    category
 9   potencia_base      201 non-null    float64
 10  longitud           201 non-null    float64
 11  anchura            201 non-null    float64
 12  altura             201 non-null    float64
 13  peso               201 non-null    int64
 14  tipo_motor         201 non-null    category
 15  cilindros          201 non-null    category
 16  tamaño_motor       201 non-null    int64
 17  sistema            201 non-null    category
 18  calibre            197 non-null    float64
 19  ataque             197 non-null    float64
 20  ratio_compresion   201 non-null    float64
 21  potencia_cv        199 non-null    float64
 22  max_rpm            199 non-null    float64
 23  consumo_carretera  201 non-null    int64
 24  consumo_ciudad     201 non-null    int64
 25  precio             201 non-null    int64
dtypes: category(10), float64(10), int64(6)
memory usage: 29.7 KB
```

Descripción estadística

Se debe hacer un análisis de cada una de las variables y describir sus propiedades. Realizar el análisis univariado es muy importante para entender el comportamiento de cada una de las variables y poder detectar posibles problemas en los datos. Nunca hay que saltarse este paso. Por ejemplo vas a contar cuantas veces aparece cada valor de las columnas categóricas. Aquellas columnas que tengan valores que solamente aparezcan una vez, las añades al array ***cols_categoricas_escasas***.

```python
cols_categoricas_escasas = []                          # completa el array
for col in cols_categoricas:
    print(autos[col].value_counts())
    print("")
```

Algunos valores categóricos solo aparecen en un único ejemplo, es el caso de la característica ***"marca"*** donde ***"mercury"*** solamente aparece una vez y eso puede dar problemas a los encoders. Igual ocurre con otras características. Una solución es ocuparse de estos casos en la fase de ingeniería de características (es lo que haremos en este ejemplo). Pero se puede perder conocimiento.

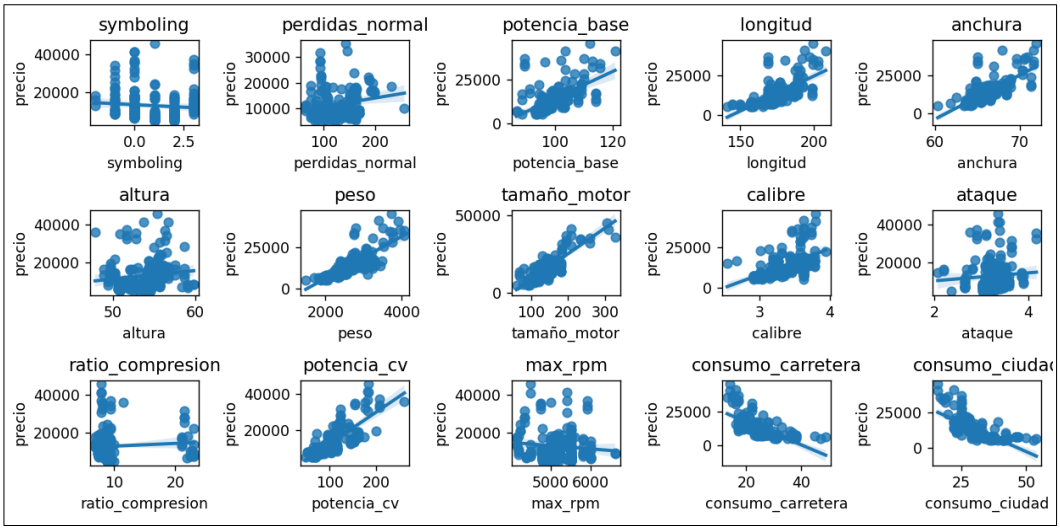
Otra solución es añadir más datos (posibilidad que no tenemos) o encargarse de garantizar que siempre estén estos ejemplos en el conjunto de entrenamiento. Este procedimiento necesita una implementación más compleja. Podemos crearnos un dataset temporal y añadirlo a los datos de train los ejemplos con valores escasos de manera artificial. Para ello creamos una condición para cada característica que tenga estos valores escasos y podemos generar sus datos. Completa las dos condiciones que faltan del siguiente código:

```python
cond_1 = "sistema in ['mfi', 'spfi']"
cond_2 = 
cond_3 = 
unir_a_train = autos.query(f"{cond_1} | {cond_2} | {cond_3}")
print("Datos que no debemos perder:\n", unir_a_train)
```

ESTUDIAR ESTADÍSTICAS DE PAREJAS DE VARIABLES

En primer lugar vamos a estudiar si todas las características numéricas influyen de manera lineal en el target. Podemos hacerlo de manera visual creando ***scatterplot()*** de cada una con el target.

```python
#===== Análisis bivariable
# generar listado de variables numéricas excepto target
cols_numericas = (autos.drop(columns=["precio"]).select_dtypes(include=np.number).columns.tolist())
n_cols_numericas = len(cols_numericas)          # Gráficos de regresión simple con target
import seaborn as sns
import math
fig, ejes = plt.subplots( math.ceil(n_cols_numericas / 5), 5, figsize=(10, 5))  # son 15, plot de 5 columnas
ejes = ejes.flatten()
for i, col in enumerate(cols_numericas):
    sns.regplot(data=autos, x=col, y="precio", ax=ejes[i])
    ejes[i].set_title(col)
plt.tight_layout()
plt.show()
```



Estudiar la correlación entre variables

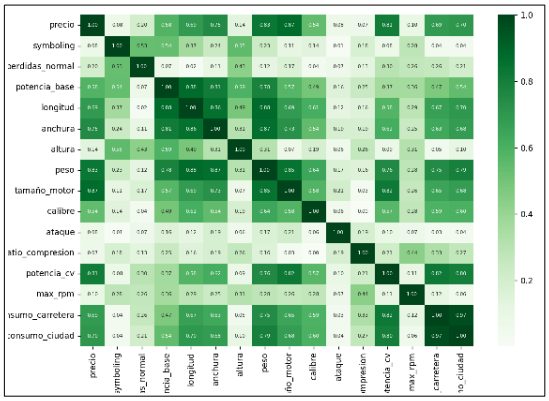
La correlación lineal entre las variables predictoras que sean numéricas. Visualmente se puede ver en las graficas como las anteriores pero realizadas entre cada par de predictoras o podemos hacer el estudio de manera numérica creando una matriz de correlaciones. Vamos a usar esta segunda aproximación, pero antes:

- La variable ***"tamaño_motor"*** tiene una relación lineal positiva con el target.
- Pero ***"symboling"*** no tiene relación lineal y por tanto el valor de la correlación no es válido.

Como estamos interesados en la cantidad de correlación y no en si esta es positiva o negativa, lo que vamos a hacer es generar la matriz de correlaciones y transformarla en sus valores absolutos para visualizar mejor los posibles casos a contemplar. Primero ponemos la característica target la primer apara que la primera fila y la primera columna se vean claramente. Debería aparecer lo más oscura posible. Si es inferior a 0.1 (color claro cercano al blanco) es candidata a eliminarla como predictora.

Además, si encontramos correlaciones importantes con otras predictoras es que hay columnas con colinealidad y eso tampoco es bueno, podemos eliminar la que tenga menor correlación con target.

```
58  from matplotlib.colors import Colormap as cm
59  cols = autos.columns.tolist()    # mover la columna precio al inicio del dataframe
60  cols.insert(0, cols.pop(cols.index("precio")))
61  autos = autos.reindex(columns= cols)
62  autos_matriz_correlaciones = autos.corr(numeric_only=True)
63  fig, ejes = plt.subplots(figsize=(12, 10))
64  absoluta = autos_matriz_correlaciones.abs()
65  sns.heatmap(absoluta, annot=True, annot_kws={'size':6}, fmt=".2f", cmap="Greens")
66  plt.show()
```



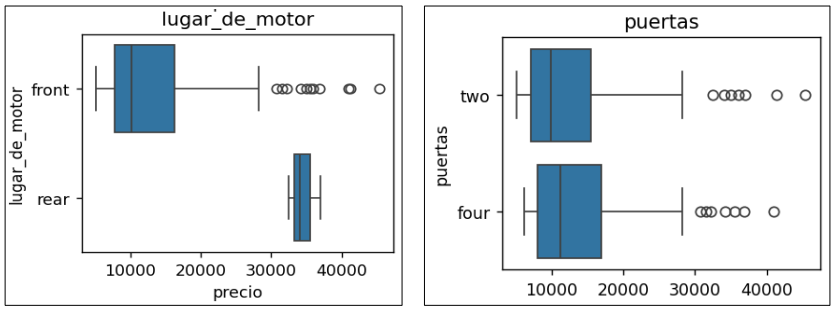
Es el caso de las características longitud, anchura, peso y tamaño del motor. Todas ellas tienen una alta correlación. Serían candidatas a examinarlas en profundidad para eliminar algunas.

A las características que tienen valores categóricos no podemos calcularles el coeficiente de correlación pero sí podemos hacerles otros test estadísticos como ***ANOVA*** (Análisis de la Varianza). Pero vamos simplemente a dibujar sus ***boxplot***.

```
# Gráficos boxplot de categóricas con el target
n_categoricas = len(cols_categoricas)
fig, ejes = plt.subplots(n_categoricas // 4 + 1, 4, figsize=(12, 7))
ejes = ejes.flatten()
for i, col in enumerate(cols_categoricas):
    sns.boxplot(data=autos, x="precio",  y=col, ax=ejes[i])
    ejes[i].set_title(col)
plt.tight_layout()
plt.show()
```

Hay variables categóricas que permiten distinguir entre grupos de valores de la variable objetivo, por ejemplo la variable ***"lugar_de_motor"*** los dos *boxplots* tienen diferencias significativas para cada valor, por lo que es una variable que sí aporta información al modelo.

La variable ***"puertas"*** no permite distinguir claramente grupos de valores del target, por lo que no es una variable que aporte información al modelo. Visualmente puedes identificar que la distribución de los ***boxplots*** son similares para ambos valores: ***two*** ó ***four***.



Aunque se detectan características a eliminar, en este ejemplo, continuamos trabajando con ellas.

FASE DE INGENIERÍA DE CARACTERÍSTICAS

Aplicamos una imputación simple sin hacer un análisis mas profundo de los datos. Las variables numéricas se imputan con la media y las categóricas con la moda. Usamos pipelines de transformación porque nos dan flexibilidad y nos van a ahorrar mucho trabajo a la larga:

- ***OneHotEncoder*** para las variables categóricas nominales.
- ***OrdinalEncoder*** para las variables categóricas ordinales.

```
#========== Ingeniería de Características
from sklearn.pipeline import Pipeline              # librerias de preprocesamiento de datos
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

y = autos['precio']                        # quitamos el precio de las predictoras
X = autos.drop('precio', axis='columns')
# Distribuir columnas predictoras en 1 de estas categorías
cols_categoricas = ["marca", "combustible", "inyección", "chasis", "tracción", "lugar_de_motor", "tipo_motor",  "sistema"]
cols_categoricas_ordinales = ["puertas", "cilindros"]
cols_todas = X.columns.tolist()
cols_numericas = [x for x in cols_todas if x not in cols_categoricas and x not in cols_categoricas_ordinales]
print("Predictoras ordinales:", cols_categoricas_ordinales)
print("Predictoras categóricas", cols_categoricas)
print("Predictoras numéricas", cols_numericas)

pipe_numericas = Pipeline(steps=[('imputer', SimpleImputer(strategy='median')), ('scaler', StandardScaler())])
pipe_categoricas = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))])
pipe_categoricas_ordinales = Pipeline(steps=[
            ('ordenc', OrdinalEncoder(handle_unknown='use_encoded_value',unknown_value=np.nan)),
            ('imputer', SimpleImputer(strategy='most_frequent'))])
preprocesa_columnas = ColumnTransformer( transformers=[
        ('numericas', pipe_numericas, cols_numericas),
        ('categoricas', pipe_categoricas, cols_categoricas),
        ('categoricas ordinales', pipe_categoricas_ordinales, cols_categoricas_ordinales)
        ])
```

DIVIDIR EL DATASET EN TRAIN Y TEST

En el exterior, dejamos el 80% de los datos para entrenar y el 20% para test. No tenemos restricciones de balanceo, etc. Si acaso, vendría bien escalar características numéricas según los algoritmos que usemos.

```
108  from sklearn.model_selection import train_test_split
109  x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
110  print("Dimensiones de train:", x_train.shape)
111  print("Dimensiones de test: ", x_test.shape)
```

BUSCAR UN BUEN MODELO DE REGRESIÓN

Si se tienen **muchos modelos,** realizar validación cruzada con todos es costoso, así que se deben ir descartando los modelos con menos desempeño hasta llegar al modelo final. Una forma de hacerlo:

- Inicialmente se dividen los datos en dos partes: Una para realizar la selección del modelo (datos de selección del modelo) y otra para realizar la prueba de desempeño (datos de test). Esta parte de los datos se debe usar solo en el final de todo el proceso.
- Luego se hace una evaluación de todos los modelos con la división anterior y se seleccionan los mejores (es preferible usar modelos con principios de funcionamiento diferentes entre ellos).
- Con lo mejores modelos (la cantidad depende de los resultados) se realiza la validación cruzada (detectar si hay *overfitting*) para obtener los que tengan mejor resultado. Se saca el mejor o los mejores modelos (mejor desempeño y poca varianza) y se realiza optimización de hiperparámetros. Este proceso es costoso, por eso se debe realizar con muy pocos modelos.
- Luego se selecciona el mejor modelo (mejor desempeño y menor varianza) y se obtienen los hiperparámetros que dieron el mejor resultado.
- Finalmente, se entrena el modelo seleccionado con los hiperparámetros encontrados con los datos de selección de modelos y se hace la prueba con los datos de test**.**

Creamos una función que se encargue de entrenar a un modelo. Esta función dividirá los datos que le pasemos de nuevo en *train* y *test*. Aunque normalmente le pasaremos un *x_train* e *y_train* externo:

```
113  from sklearn.dummy import DummyRegressor
114  from sklearn.linear_model import LinearRegression
115  from sklearn.linear_model import Lasso
116  from sklearn.linear_model import Ridge
117  from sklearn.linear_model import ElasticNet
118  from sklearn.preprocessing import PolynomialFeatures   # Generar datos de potencias de los originales
119  from sklearn.neighbors import KNeighborsRegressor
120  import warnings
121  warnings.filterwarnings("ignore")
122
123  dic_resultados = {}            # diccionario con resultados entrenamiento: {clave:{train_score,test_score},...}
124
125  def entrenar_modelo(modelo, procesador, X: pd.DataFrame, y: pd.Series, pct_test:float=0.2):
126      x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=123, test_size=pct_test)
127      pipe = Pipeline(steps=[("preprocessor", procesador), ("model", modelo)])
128      modelo_entrenado = pipe.fit(x_train, y_train)         # entrenar
129      train_score = modelo_entrenado.score(X=x_train, y=y_train)
130      test_score = modelo_entrenado.score(X=x_test, y=y_test)
131      return {'train_score': train_score, 'test_score': test_score}
```

Hacemos otra función para mostrar los resultados del entrenamiento:

```
133  def compara_resultados():      # función para comparar los resultados de los modelos
134      for key in dic_resultados:
135          print('Regresion: ', key)
136          print(f'  train_score: {dic_resultados[key]["train_score"]:10.6f}')
137          print(f'  test_score:  {dic_resultados[key]["test_score"]:10.6f}')
138          print()
```

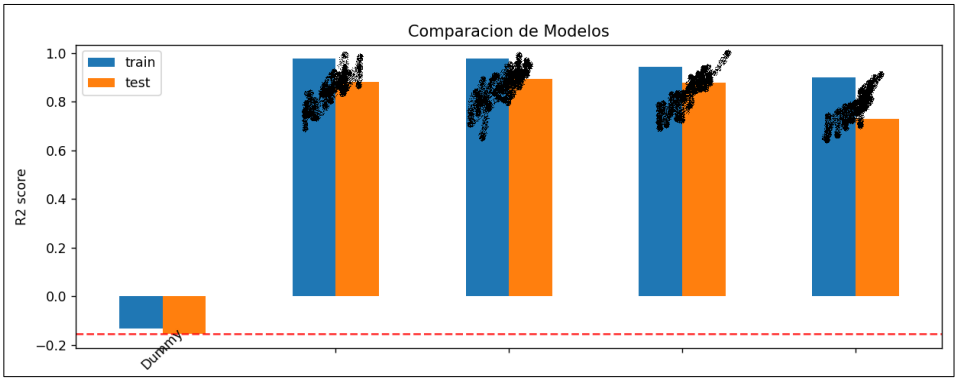
Ahora creamos y entrenamos varios modelos de regresión lineal con la función que hemos definido y registramos los ***score*** de cada uno en el diccionario creado a tal fin. Para el caso del modelo de regresión polinomial, vamos a generar características nuevas así que para automatizar el proceso, creamos un pipeline que automatice este nuevo paso, de esta manera no tendremos que hacerlo nosotros manualmente:

```
dic_resultados['Dummy'] = entrenar_modelo(DummyRegressor(strategy='median'), preprocesa_columnas, x_train, y_train)
#dic_resultados['Regresión Lineal'] = entrenar_modelo(LinearRegression(), preprocesa_columnas, x_train, y_train)
dic_resultados['Lasso'] = entrenar_modelo(Lasso(alpha=0.1), preprocesa_columnas, x_train, y_train)
dic_resultados['Ridge'] = entrenar_modelo(Ridge(alpha=0.1), preprocesa_columnas, x_train, y_train)
dic_resultados['Elasticnet'] = entrenar_modelo(ElasticNet(alpha=0.1, l1_ratio=0.5, max_iter= 10000, warm_start= True),
                                               preprocesa_columnas, x_train, y_train)
pipe_poli = Pipeline([('polynomial_features', PolynomialFeatures(degree=3, include_bias=False)),
                      ('linear_regressor', LinearRegression())])
dic_resultados['Polinomial'] = entrenar_modelo( pipe_poli, preprocesa_columnas, x_train, y_train)
dic_resultados['KNN'] = entrenar_modelo(KNeighborsRegressor(n_neighbors=5), preprocesa_columnas, x_train, y_train)
compara_resultados()
```

Ahora podemos dibujar. Pero antes una aclaración, el modelo etiquetado como "Dummy" es uno que directamente devuelve siempre un valor fijo independientemente de la entrada, así que solamente acierta si se le aparece la virgen en un patinete. Como eso es improbable, es muy malo, en regresión similar a uno aleatorio. De hecho, uno que sea peor que él, tiene mérito. Y sin embargo, uno de los que tenemos tiene tanto overfitting que es mucho peor, porque al menos el dummy da la mediana y no se aleja mucho en sus predicciones, tendrá un ***score*** negativo (coeficiente de determinación malo) pero no muy alto. Así que cuando ejecutes el código, el modelo que de un score horrible es candidato a descartarlo (aunque si antes hubiésemos realizado limpieza de predictoras, selección de características formalmente hablando, el resultado podría cambiar).

```
# Crear un diccionario solo con los resultados de prueba de cada modelo
nombre_modelos = dic_resultados.keys()
res_train = {}   # crear diccionario vacio
res_test = {}    # crear diccionario vacio
for nombre in nombre_modelos:
    res_train[nombre] = dic_resultados[nombre]['train_score']
    res_test[nombre] = dic_resultados[nombre]['test_score']
df_comparacion = pd.DataFrame([res_train, res_test], index=['train', 'test'])
# Dibujar un gráfico de barras
fig, ax = plt.subplots(figsize=(10, 4))
df_comparacion.T.plot(kind='bar', ax=ax)
ax.set_ylabel('R2 score')
ax.set_title('Comparación de Modelos')
ax.set_xticks(range(len(df_comparacion.columns)))
ax.set_xticklabels([])
# Dibujar etiquetas dentro de las barras rotadas 90 grados
for i, label in enumerate(df_comparacion.columns):
    bar_center = (df_comparacion.loc['train', label] + df_comparacion.loc['test', label]) / 2
    ax.text(i, bar_center, label, ha='center', va='center_baseline', rotation=45)
# Dibujar una línea del resultado del DUmmerRegressor
ax.axhline(df_comparacion['Dummy']['test'], color='red', linestyle='--', alpha=0.8)
plt.tight_layout()
plt.show()
```

Vuelve a ejecutar el código, y si vuelve a aparecer uno que destaque por lo malo que es, lo volvemos a nominar y comentamos su código.



Al final el gráfico que te quedará parecido al de arriba, verás que son todos bastante competentes (salvo el dummy, claro y alguno que tenga overfitting). He tachado los nombres para que tengas que averiguar tu mismo cuales tienen problemas si mantenemos todas las predictoras. Nos hemos quedado con 4 que parecen mejores.

VALIDACIÓN CRUZADA

Analizar la varianza de los modelos más prometedores para obtener los que tengan mejor desempeño. Aunque tu debes usar los 4 que hayas considerado mejores, yo te voy a mostrar el código de todos:

```
#========= Validación cruzada
# lista para almacenar cada uno los modelos seleccionados para CV
modelos = []
# Almacenando los modelos como una tupla (nombre, modelo)
modelos.append(('Regresión Lineal', LinearRegression()))
modelos.append(('Lasso', Lasso(alpha=0.1)))
modelos.append(('Ridge', Ridge(alpha=0.1)))
modelos.append(('Elastic_net', ElasticNet(alpha=0.1, l1_ratio=0.5, max_iter= 10000, warm_start= True)))
modelos.append(('Polinomial', pipe_poli))
modelos.append(('KN', KNeighborsRegressor(n_neighbors=5)))
```

Ahora vamos a definir estructuras que nos permitan realizar la CV con 10-Fold para cada modelo. Yo voy a usar todos y eso hará que mis gráficos no sean muy buenos al tener dos modelos muy malos.

```
# Grabar los resultados de cada modelo
from sklearn import model_selection
seed = 123              # Semilla para obtener los mismos resultados de pruebas
resultados = []
nombres = []
for n, m in modelos:
    kfold = model_selection.KFold(n_splits=10)    #  Kfold cross validation
    modelo_pipe = Pipeline(steps=[("preprocessor", preprocesa_columnas), ("model", m)])
    cv_resultados = model_selection.cross_val_score(estimator=modelo_pipe, X=x_train, y=y_train, cv=kfold, scoring="r2")
    resultados.append(cv_resultados)
    nombres.append(n)
    msg = f"(10-Fold CV de {n}, media: {cv_resultados.mean()}, desviación: {cv_resultados.std()}"
    print(msg)

plt.figure(figsize = (8,4))
result_df = pd.DataFrame(resultados, index=nombres).T
sns.boxplot(data=result_df)
plt.title("Resultados de Cross Validation")
plt.show()

plt.figure(figsize = (8,4))
sns.lineplot(data=result_df)
plt.title("Resultados de cada Kfold")
plt.show()
```

Comparación Estadística de Modelos

Ahora tras la validación cruzada vamos a usar los resultados para comprobar si hay o no una diferencia significativa entre los modelos considerados. Usaremos un ***test ANOVA de un factor***:

```
# Comparación estadística de los modelos
from scipy.stats import f_oneway

model1 = result_df['Regresión Lineal']
model2 = result_df['Lasso']
model2 = result_df['Ridge']
model3 = result_df['Elastic_net']
model4 = result_df['Polinomial']
model5 = result_df['KN']
estadistico, p_value = f_oneway(model1, model2, model3, model4, model5)  # ANOVA de un factor
print(f'Estadístico: {estadistico}')
print(f'p_value: {p_value}')
# Un test de contraste de hipótesis con los datos obtenidos
alfa = 0.05  # nivel de significancia
if p_value < alfa:
    print("Existe una diferencia estadísticamente significativa en los resultados de"
          " cross-validation de los modelos.")
else:
    print("No Existe una diferencia estadísticamente significativa en los resultados de "
          "cross-validation de los modelos.")
```

En mi caso el test no es adecuado porque una de las condiciones que asume es que las desviaciones de todos los datos que participan en el el test son similares (*homoscedasticidad*) y en mi ejecución no se cumple al mantener los dos modelos que son rematadamente malos con una varianza excesiva respecto a los demás.

TUNING DE HIPERPARÁMETROS

Una vez tenemos dos modelos de muy diferente funcionamiento les ajustamos los hiperparámetros. Para automatizar el proceso utilizo ***Grid search***. En mi caso voy a hacerlos con ***Ridge*** y ***KNN*** para que veáis como hacerlo. En el diccionario de parámetros podéis incluir una entrada para cada hiperparámetro diferente que tenga el algoritmo de aprendizaje y una lista de valores que queréis probar. Lógicamente ***grid search*** probará las combinaciones de todos y eso lleva tiempo.

```
233  #=========== Tuning de parámetros
234  from sklearn.model_selection import GridSearchCV
235  #=== Para Lasso
236  print("====== Tuning de Parámetros\n---- Para Lasso ----")
237  lasso_pipe = Pipeline(steps=[("preprocessor", preprocesa_columnas), ("model", Lasso())])
238  parametros = {'model__alpha': [0.01, 0.1, 0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 1.0]}
239  gs= GridSearchCV(estimator=lasso_pipe, param_grid=parametros, cv=5, return_train_score=True, scoring='r2')
240  gs.fit(x_train, y_train)
241  print(f"Mejor resultado = {gs.best_score_}")
242  print(f"Mejor std = {gs.cv_results_['std_test_score'][gs.best_index_]}")
243  print(f"Mejores parámetros = {gs.best_params_}")
244  #=== Para KNN
245  print("----- KNN -----")
246  knn_pipe = Pipeline(steps=[("preprocessor", preprocesa_columnas), ("model", KNeighborsRegressor())])
247  parametros = {'model__n_neighbors': [5, 10, 12, 14, 18, 20, 25, 30, 35, 50]}
248  gs1= GridSearchCV(knn_pipe, parametros, cv=5, return_train_score=True, scoring='r2')
249  gs1.fit(x_train, y_train)
250  print(f"Mejor resultado = {gs1.best_score_}")
251  print(f"Mejor std = {gs1.cv_results_['std_test_score'][gs1.best_index_]}")
252  print(f"Mejores parámetros = {gs1.best_params_}")
```

ENTRENAR EL MEJOR MODELO CON LOS MEJORES HIPERPARÁMETROS

Una vez tengamos los mejores scores de nuestros mejores modelos, podemos comparar con qué hiperparámetros de uno de ellos conseguimos el mejor puntuación. A igualdad de scores o valores similares nos quedaremos con el que sea más eficiente, escalable, etc. Pero si hay diferencias nos quedaremos con el mejor y sabiendo la mejor de sus configuraciones. Ahora solamente nos queda entrenarlo con todos los datos:

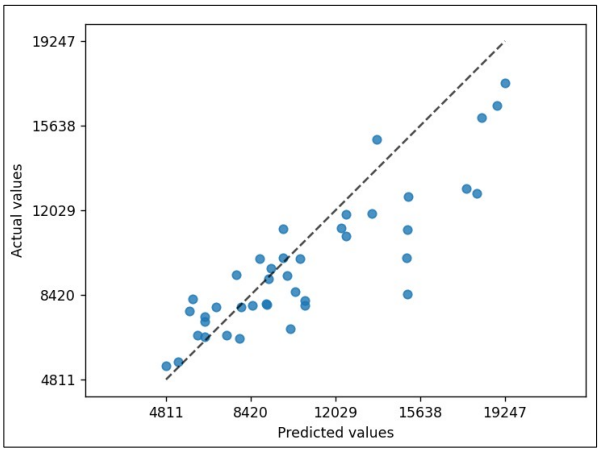
```
265  #======= Entrenar, medir y Guardar el modelo ganador
266  #--- Suponiendo que haya sido el KNN
267  from sklearn import metrics
268  knn = KNeighborsRegressor(n_neighbors=gs.best_params_['n_neighbors'])
269  knn.fit(x_train, y_train)
270  y_pred = knn.predict(x_test)
271  print("R2:", metrics.r2_score(y_true=y_test, y_pred=y_pred))
272
273  from joblib import dump                    # libreria de serializacion
274  dump(knn, 'modelo_ganador.joblib')
275  # ... para usarlo más tarde en alguna aplicación.
276  from joblib import load
277  modelo = load('modelo_ganador.joblib')
278  datos_prueba = x_test[:1]
279  print(f"Predicción {modelo.predict(datos_prueba)}")  # resultados de predicion con el modelo
```

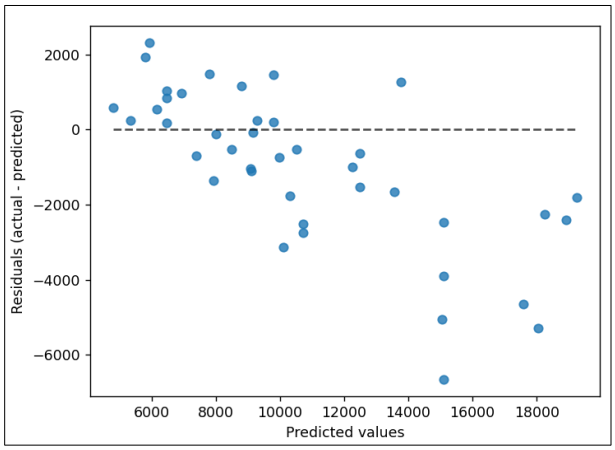
Y validarlo con los datos de test, además de mostrar gráficamente como son los errores que comete e incluso hacerle alguna medida de importancia de características para poder descartar más

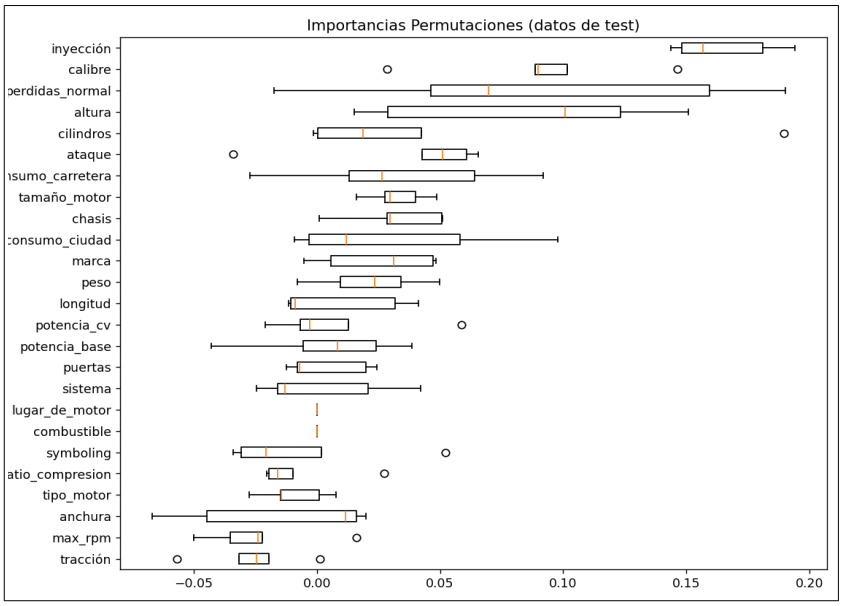
características predictoras sin que afecte mucho a su desempeño. En esta ocasión voy a utilizar un test de permutaciones para situar la importancia de cada característica en un gráfico. Ten en cuenta que la importancia de cada característica depende del modelo, es decir, en dos modelos distintos aunque usen los mismos datos, la importancia puede cambiar:

```
#======= Entrenar, medir y Guardar el modelo ganador
#--- Suponiendo que haya sido el KNN
from sklearn import metrics
knn_pipe = Pipeline(steps=[("preprocessor", preprocesa_columnas),
                           ("model",KNeighborsRegressor(n_neighbors=gs1.best_params_['model__n_neighbors']))])
knn_pipe.fit(x_train, y_train)
y_pred = knn_pipe.predict(x_test)
print("R2:", metrics.r2_score(y_true=y_test, y_pred=y_pred))

from sklearn.metrics import PredictionErrorDisplay
PredictionErrorDisplay.from_predictions(y_true=y_test, y_pred=y_pred, kind="actual_vs_predicted")
plt.show()
PredictionErrorDisplay.from_predictions(y_true=y_test, y_pred=y_pred, kind="residual_vs_predicted")
plt.show()
from sklearn.inspection import permutation_importance
imps = permutation_importance(knn_pipe, x_test, y_test, n_repeats = 5, scoring= "r2", n_jobs=-1, random_state=123)
fig= plt.figure(figsize=(10,8))
perm_sorted_idx = imps.importances_mean.argsort()
plt.boxplot(imps.importances[perm_sorted_idx].T, vert=False, labels=x_test.columns[perm_sorted_idx])
plt.title("Importancias Permutaciones (datos de test)")
plt.show()
```







Y una vez entrenado como necesitemos, lo guardaríamos para usarlo en alguna aplicación:

```
276  from joblib import dump                    # libreria de serializacion
277  dump(knn_pipe, 'modelo_ganador.joblib')
278  # ... para usarlo más tarde en alguna aplicación.
279  from joblib import load
280  modelo = load('modelo_ganador.joblib')
281  datos_prueba = x_test[:1]
282  print(f"Predicción {modelo.predict(datos_prueba)}")  # resultados de predicion con el modelo
```

APLICAR LO ANTERIOR

Haz una copia al fichero y lo renombras añadiéndole el sufijo ***"_v2.py" o "_v2.ipybn***. Debes modificarlo ejecutando el original de manera que:

ENTREGA 4: Entrega los ficheros `u02_p03_a2_<alumno>.*` y *`u02_p03_a2_<alumno>_v2`* de manera que el de la segunda versión:

a) Cuando se genere el primer gráfico, busca una de las características que no tengan una relación lineal con el target. La eliminas del *DataFrame* y de los arrays de nombres de columnas en el fichero ***_v2***. Indica qué característica eliminas:_____

b) Cuando aparezca la matriz de correlaciones, identifica las características que tengan menos del 0.2% de correlación lineal con el target y las eliminas del *DataFrame* en el nuevo fichero. ¿Cuáles son las variables que eliminas? _____

c) Además, para eliminar colinealidad localiza todas las parejas (puedes ir bajando desde la primera a la última fila y buscar y eliminar correlaciones mayores de 0.8 que no sea con el target). ¿Qué columnas has eliminado? _____

d) En los diagramas *boxplot* de las columnas categóricas, elimina una o dos que no genere diferencias importantes entre sus valores (las identificas porque los *boxplot* de cada valor serán parecidos). ¿Cuáles eliminas?_____

e) En el paso de "*Buscar un buen modelo de regresión*" descarta los 3 peores (aquellos que tengan peor R^2 en los datos de test). Muestra captura de *scores* y el gráfico e indica los modelos que descartas. _____.

f) En la validación cruzada, utiliza solamente los 3 mejores modelos del paso anterior. Muestra captura de resultados y escoge aquel que tenga menor varianza (es más estable).

g) En el tunning de hiperparámetros, busca los parámetros que se puedan usar con el modelo con el que te has quedado (si das uno erróneo al grid search, el mensaje de error te dice los que puedes usar). Indica captura de ejecución de esta fase y del entrenamiento del modelo elegido con los datos de train y test.

h) Por último, analiza los errores que comete con el test mediante gráficos de predicciones vs valores reales, gráfico de residuos, y gráfico de test de permutaciones para estudiar la importancia de las predictoras en el modelo. Guarda el modelo en un fichero.

📌ACTIVIDAD 3: CREAR UN SISTEMA TASADOR DE VIVIENDAS.

DEFINIR Y RECOPILAR DATOS

En primer lugar debemos ponernos de acuerdo de manera que cada uno de nosotros piense una característica que:

- Pueda influir en el precio de una vivienda.
- Sea una información que tengamos la capacidad de recolectar.

ENTREGA 5: Añade al documento enlace enlace:

a) En la hoja de cálculo ***"datos_viviendas.xlsl"*** en la hoja datos, añade una columna cuyo prefijo tenga tus iniciales. Ya tiene estas nueve: ***josrosrod_autor, josrosrod_origen, josrosrod_municipio, josrosrod_dia, josrosrod_mes, josrosrod_año, josrosrod_lat, josrosrod_lon, josrosrod_precio***
b) En la hoja ***"descripción"*** añade una descripción del significado de esa columna y de cómo se puede conseguir. Ya tendrás anotaciones de las características que están añadidas.

Cuando ya tengamos definidas las características que vamos a utilizar para esta actividad, vamos a recolectar datos, al menos 10 cada uno. Intentando completar la información que no dispongamos (intentaremos no inventar nada, de manera que cuando algo no lo sepamos, tendremos que indicar de donde lo hemos conseguido). No puedes dejar más de un valor ausente en tus datos.

ENTREGA:

a0) Añade a la hoja de cálculo 10 ejemplos y aporta una columna.
a) Análisis y preprocesamiento de datos:
- Análisis de datos: histogramas, boxplots, cantidad de datos ausentes, presencia de outliers.
- Codificación de categóricas: label-encoding o one-hot-encoding.
- Detección e imputación de ausentes.
- Detección e imputación de anomalías.
- Escalado de numéricas.
- Debes ir creando un pipeline de manera que se le facilite uno de los ejemplos y lo preprocese para dárselo al modelo.

a) Selección de características:
- Matriz de correlaciones y mapa de calor: más fácil de interpretar si dejas el target como la primera característica.
- Estudio estadístico univariado (eliminar características con poca influencia en target).
- Estudio de colinealidad por parejas (eliminar alta correlación entre predictoras).

c) Selección de modelos de regresión: debes considerar al menos 5 diferentes y uno de ellos el método de mínimos cuadrados. Recuerda que tienes LinearRegressor, SGDRegressor, Ridge, Lasso, ElasticNet, K-NN, polinómico...
d) Estudio de varianza de modelos con CV.
e) Seleccionar los dos que mejor desempeño muestren justificando su elección.
f) Configuración de hiperparámetros con grid-search en los mejores 2 modelos seleccionados.
f) Estudio final de errores en las predicciones y gráfico de importancia de características.
g) Selección justificada del mejor y su descripción:
- Qué predictoras utiliza.
- Una descripción de su utilización y procesos que realiza (preprocesamiento, ...).
- Qué desempeño alcanza en train y test (indica también la métrica empleada)
- Una valoración final de tu modelo según los resultados obtenidos.

h) Guarda el modelo en un fichero con el nombre ***<iniciales>_modelo_u02_p03_a3.joblib*** donde ***<iniciales>*** son las 3 primeras letras del nombre y apellidos, en mi caso sería ***"josrosrod_modelo_u02_p03_a3.joblib".***

La entrega de esta última actividad es el 70% de la nota de la práctica y se valora:
- 50% puntos por la entrega: la realización de las actividades propuestas.
- Al modelo que entregues se le pedirá que realice 10 predicciones con datos probablemente no vistos. Según el error promedio que cometa se le asignará posición en un ranking. En función de la posición que ocupe (lo bien que lo haga), conseguirá más o menos puntuación adicional hasta completar el 20% restante.