Introducción a Machine Learning Supervisado

Ejercicio práctico: Predicción de precios de casas

En este notebook, exploraremos un problema de regresión utilizando el conjunto de datos "House Prices: Advanced Regression Techniques". Vamos a realizar un análisis exploratorio, preprocesar los datos y entrenar un modelo de regresión lineal simple.

Pasos:

1. Carga de datos
2. Análisis exploratorio de datos (EDA)
3. Preprocesamiento de los datos
4. Entrenamiento del modelo
5. Evaluación del modelo
6. Conclusiones

---

1. Carga de datos

Primero, vamos a cargar los datos y ver una vista general del dataset. Usa la librería pandas para leer el archivo House_Price_dataset.csv y crear un df.

```
# prompt: carga los datos en un dataframe de pandas

import pandas as pd

# Reemplaza 'House_Price_dataset.csv' con la ruta correcta a tu archivo CSV
df = pd.read_csv('House_Price_dataset.csv')

# Muestra las primeras filas del DataFrame para verificar que se cargó correctamente
print(df.head())
```

```
   property_id  location_id  \
0       237062         3325
1       346905         3236
2       386513          764
3       656161          340
4       841645         3226

                                            page_url property_type  \
0  https://www.zameen.com/Property/g_10_g_10_2_gr...          Flat
1  https://www.zameen.com/Property/e_11_2_service...          Flat
2  https://www.zameen.com/Property/islamabad_g_15...         House
3  https://www.zameen.com/Property/islamabad_bani...         House
4  https://www.zameen.com/Property/dha_valley_dha...         House

         price     location       city      province_name   latitude  \
0  10000000.0         G-10  Islamabad  Islamabad Capital  33.679890
1   6900000.0         E-11  Islamabad  Islamabad Capital  33.700993
2  16500000.0         G-15  Islamabad  Islamabad Capital  33.631486
3  43500000.0    Bani Gala  Islamabad  Islamabad Capital  33.707573
4   7000000.0  DHA Defence  Islamabad  Islamabad Capital  33.492591

   longitude  baths       area   purpose  bedrooms  date_added         agency  \
0  73.012640    2.0    4 Marla  For Sale       2.0  02-04-2019            NaN
1  72.971492    3.0  5.6 Marla  For Sale       3.0  05-04-2019            NaN
2  72.926559    6.0    8 Marla  For Sale       5.0  07-17-2019            NaN
3  73.151199    4.0    2 Kanal  For Sale       4.0  04-05-2019            NaN
4  73.301339    3.0    8 Marla  For Sale       3.0  07-10-2019  Easy Property
```

```
                                        agent Area Type  Area Size  \
0                                         NaN     Marla        4.0
1                                         NaN     Marla        5.6
2                                         NaN     Marla        8.0
3                                         NaN     Kanal        2.0
4  Muhammad Junaid Ceo Muhammad Shahid Director     Marla        8.0

  Area Category
0     0-5 Marla
1    5-10 Marla
2    5-10 Marla
3     1-5 Kanal
4    5-10 Marla
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7716 entries, 0 to 7715
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   property_id    7716 non-null   int64
 1   location_id    7716 non-null   int64
 2   page_url       7715 non-null   object
 3   property_type  7715 non-null   object
 4   price          7715 non-null   float64
 5   location       7715 non-null   object
 6   city           7715 non-null   object
 7   province_name  7715 non-null   object
 8   latitude       7715 non-null   float64
 9   longitude      7715 non-null   float64
 10  baths          7715 non-null   float64
 11  area           7715 non-null   object
 12  purpose        7715 non-null   object
 13  bedrooms       7715 non-null   float64
 14  date_added     7715 non-null   object
 15  agency         1940 non-null   object
 16  agent          1940 non-null   object
 17  Area Type      7715 non-null   object
 18  Area Size      7715 non-null   float64
 19  Area Category  7715 non-null   object
dtypes: float64(6), int64(2), object(12)
memory usage: 1.2+ MB
```

2. Análisis exploratorio de datos (EDA)

Antes de entrenar un modelo, es importante explorar los datos para entender mejor las características, la distribución de la variable objetivo y las correlaciones entre las variables.

a. Información del dataset

Revisemos la estructura de los datos y veamos si hay valores faltantes. Usa la función de pandas.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7716 entries, 0 to 7715
Columns: 9732 entries, property_id to Area Category_90-100 Kanal
dtypes: bool(9724), float64(8)
memory usage: 72.0 MB
```

```
valores_nulos = df.isnull().sum()
print(valores_nulos[valores_nulos > 0])
```

```
Series([], dtype: int64)
```

b. Estadísticas descriptivas

Veamos algunas estadísticas descriptivas para entender mejor la distribución de los datos. Usa la función de pandas.

```
resumen_estadistico = df.describe()
print(resumen_estadistico)
```

```
         property_id   location_id         price      latitude     longitude  \
count  7.716000e+03  7.716000e+03  7.716000e+03  7.716000e+03  7.716000e+03
mean  -2.357425e-16  2.210086e-17 -7.366954e-18 -2.210086e-16 -2.685255e-15
std    1.000065e+00  1.000065e+00  1.000065e+00  1.000065e+00  1.000065e+00
min   -5.154320e+00 -1.165849e+00 -4.355849e-01 -1.426392e+00 -1.046948e+01
25%   -2.596140e-01 -8.795351e-01 -4.292875e-01 -1.372787e+00 -1.409592e+00
50%    2.332416e-01 -2.668288e-01 -2.529045e-01  3.249277e-01  5.121184e-01
75%    6.797940e-01  7.997798e-01  3.242918e-02  8.668385e-01  8.725325e-01
max    1.039365e+00  2.428859e+00  2.742892e+01  1.101351e+01  2.812203e+00

              baths      bedrooms     Area Size
count  7.716000e+03  7.716000e+03  7.716000e+03
mean   4.051825e-17  6.630259e-17  1.095834e-16
std    1.000065e+00  1.000065e+00  1.000065e+00
min   -1.407616e+00 -1.777676e+00 -9.372982e-01
25%   -5.576371e-01 -7.565858e-01 -4.843671e-01
50%   -1.326475e-01 -2.460409e-01 -1.608449e-01
75%    7.173319e-01  7.750488e-01  5.347280e-01
max    2.842280e+00  5.369953e+00  6.632297e+01
```

c. Distribución de la variable objetivo

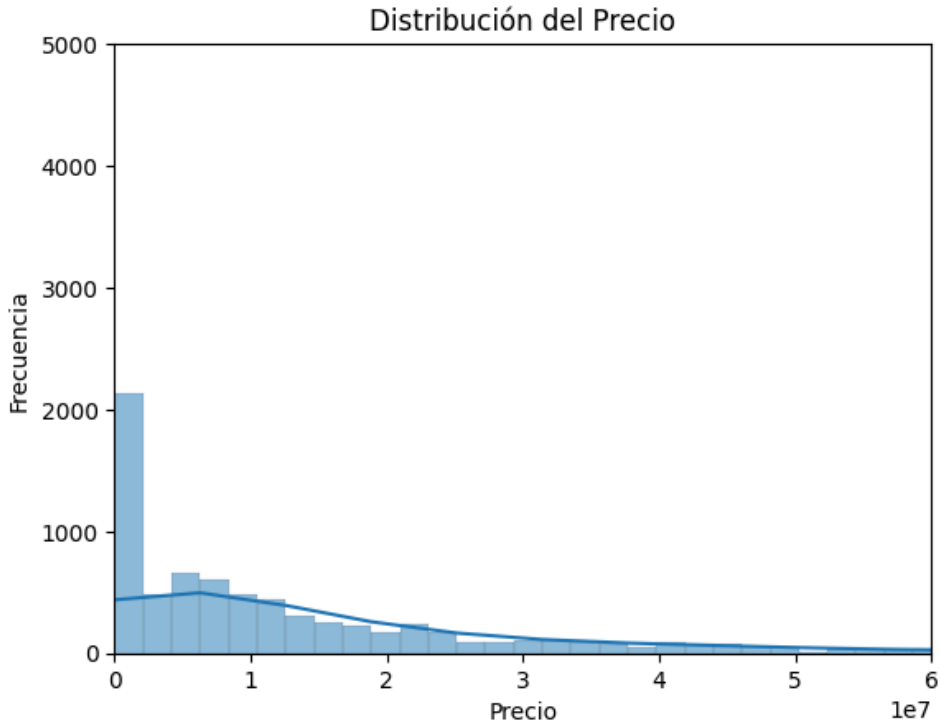
La variable que queremos predecir es `price`. Visualicemos su distribución con MatplotLib.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Visualizar la distribución de una variable (por ejemplo, 'variable_objetivo')
sns.histplot(df['price'], kde=True)
plt.title('Distribución del Precio')
plt.xlabel('Precio')
plt.ylabel('Frecuencia')

plt.xlim(0, 60000000)
plt.ylim(0, 5000)

plt.show()
```

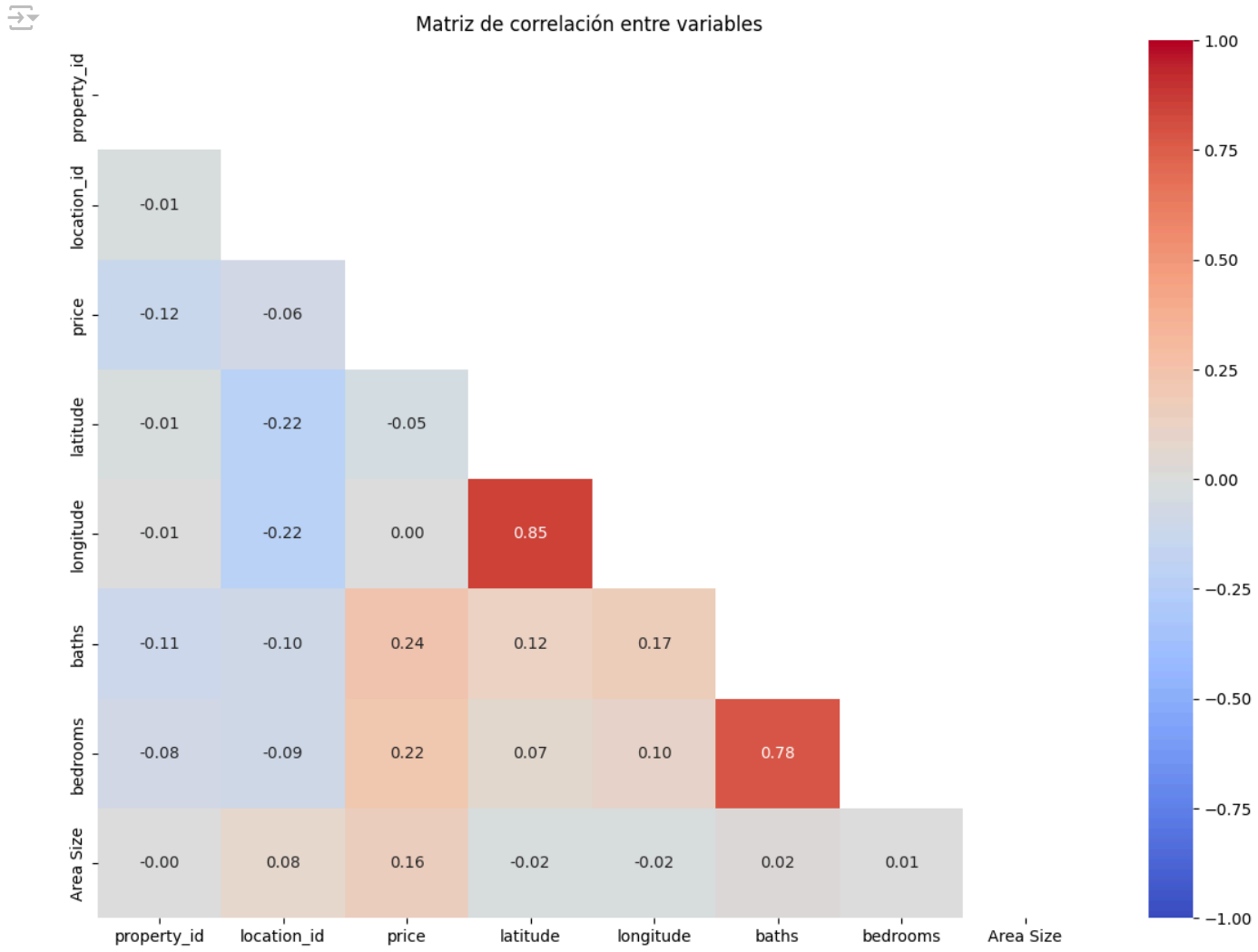


d. Matriz de correlación

Vamos a visualizar una matriz de correlación para identificar qué variables están más relacionadas con `price`. Para ello usad seaborn y pandas.

```
# Usaremos solo las columnas numéricas para poder analizar la matriz de correlacion
numerical_df = df.select_dtypes(include=['float64', 'int64'])
corr_matrix = numerical_df.corr()
```

```
import numpy as np
plt.figure(figsize=(14, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm",
        vmin=-1, vmax=1, mask=np.triu(corr_matrix),
        annot_kws={"size": 10})
plt.title("Matriz de correlación entre variables ")
plt.show()
```



3. Preprocesamiento de los datos

Ahora vamos a preparar los datos para el modelo:

1. **Manejo de valores faltantes**: Imputamos los valores faltantes con la media.
2. **Codificación de variables categóricas**: Convertimos las variables categóricas en variables dummy.
3. **Escalado de las características**: Escalamos los valores para que tengan media cero y varianza uno.

Para estos procesos usamos funciones de pandas y de sklearn.

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
# Imputación con la media para columnas numéricas
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
imputer = SimpleImputer(strategy='mean')
df[numerical_columns] = imputer.fit_transform(df[numerical_columns])
```

```
# Codificación de variables categóricas (one-hot encoding)
categorical_columns = df.select_dtypes(include=['object']).columns
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
```

```
# Escalado de características (media cero y varianza uno)
scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

4. Dividir el conjunto de datos en entrenamiento y prueba

Vamos a dividir los datos en un conjunto de entrenamiento (70%) y un conjunto de prueba (30%).

```
from sklearn.model_selection import train_test_split

# Asumiendo que 'price' es la variable objetivo
X = df.drop('price', axis=1)  # Variables predictoras
y = df['price']  # Variable objetivo

# División en conjuntos de entrenamiento (70%) y prueba (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# X_train, y_train contienen el 70% de los datos (entrenamiento)
# X_test, y_test contienen el 30% de los datos (prueba)


print("Datos de entrenamiento (X_train):")
print(X_train.head())
print("\nVariable objetivo de entrenamiento (y_train):")
print(y_train.head())
```

```
3095                                                False
2117                                                False
4128                                                False
5860                                                False
6520                                                False

      page_url_https://www.zameen.com/Property/adiala_road_green_villas_5_marla_double_storey_house_at_
3095                                                False
2117                                                False
4128                                                False
5860                                                False
6520                                                False

      ...  Area Type_Marla  Area Category_1-5 Kanal  \
3095  ...             True                    False
2117  ...             True                    False
4128  ...             True                    False
5860  ...             True                    False
6520  ...             True                    False
```

```
3095                    True                    False
2117                    True                    False
4128                   False                    False
5860                    True                    False
6520                   False                    False

[5 rows x 9731 columns]

Variable objetivo de entrenamiento (y_train):
3095   -0.034446
2117   -0.434738
4128   -0.379967
5860   -0.435050
6520   -0.246217
Name: price, dtype: float64
```

5. Entrenar el modelo

Ahora vamos a entrenar un modelo de **Regresión Lineal** usando el conjunto de entrenamiento.

```
from sklearn.linear_model import LinearRegression

# Crear una instancia del modelo de Regresión Lineal
modelo = LinearRegression()

# Entrenar el modelo usando el conjunto de entrenamiento
modelo.fit(X_train, y_train)

# Ahora el modelo está entrenado, puedes visualizar el coeficiente e intercepto
print("Coeficientes del modelo:", modelo.coef_)
print("Intercepto del modelo:", modelo.intercept_)
```

```
Coeficientes del modelo: [ 0.01913609 -0.0159323  -0.01628281 ...  2.64352656 -0.30259099
 -0.17021919]
Intercepto del modelo: -0.03435724715477852
```

6. Hacer predicciones y evaluar el modelo

Después de entrenar el modelo, predecimos los valores en el conjunto de prueba y evaluamos su rendimiento usando métricas como el error cuadrático medio (MSE) y el coeficiente de determinación (R^2).

```
from sklearn.metrics import mean_squared_error, r2_score

# Hacer predicciones en el conjunto de prueba
y_pred = modelo.predict(X_test)

# Calcular el error cuadrático medio (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Error cuadrático medio (MSE):", mse)

# Calcular el coeficiente de determinación (R²)
r2 = r2_score(y_test, y_pred)
print("Coeficiente de determinación (R²):", r2)
```

```
Error cuadrático medio (MSE): 0.5791649054738045
Coeficiente de determinación (R²): 0.574074219216345
```

```
# El MSE de aproximadamente 0.579 indica que las predicciones están, en promedio,
# a 0.579 unidades al cuadrado del valor real.
# Este error puede considerarse bajo o alto en función de la escala de los datos.
# Al nosotros contar con un dataset con bastantes datos, es un desempeño aceptable.
```

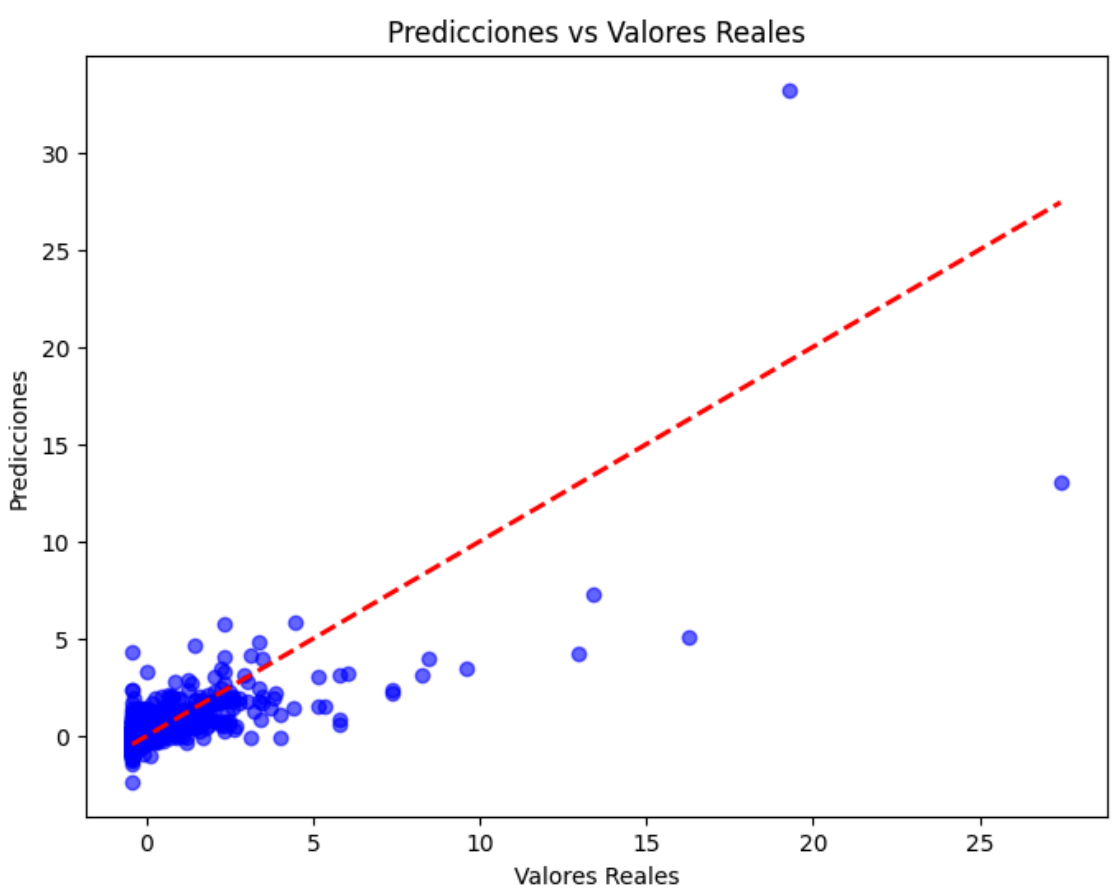
```
# El coeficiente de determinación (R²) de 0.574 muestra que el modelo explica el 57.4% de la variabilidad en
# los datos de prueba.
# Esto sugiere que el modelo captura parcialmente la relación entre las variables, pero también existe una
# variación significativa no explicada.
```

Conclusión General: El modelo de regresión lineal ofrece un desempeño razonable, aunque no ideal.

7. Visualizar predicciones vs valores reales

Para tener una mejor comprensión del rendimiento del modelo, graficamos los valores reales vs las predicciones.

```
# Crear la gráfica de dispersión
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color='b')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
plt.xlabel('Valores Reales')
plt.ylabel('Predicciones')
plt.title('Predicciones vs Valores Reales')
plt.show()
```



Podemos observar que el modelo tiene un ajuste aceptable, no se ve sobre ajuste, y los datos pasan cerca de la línea.

8. Optimización del modelo: Validación cruzada y búsqueda de hiperparámetros

Para mejorar el rendimiento del modelo, vamos a utilizar un proceso de **validación cruzada** y una búsqueda exhaustiva de los **mejores hiperparámetros** con **GridSearchCV**. En este ejemplo, utilizaremos un modelo de **Regresión Ridge**.

¿Por qué Ridge?

La regresión Ridge es una variante de la regresión lineal que añade una penalización al tamaño de los coeficientes, lo que puede ayudar a evitar el sobreajuste.

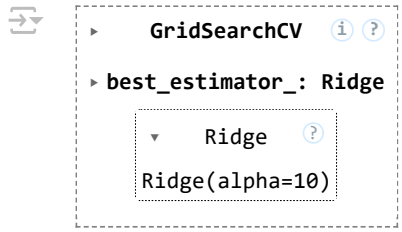
a. Búsqueda de hiperparámetros con GridSearchCV

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Definir el modelo de regresión Ridge
ridge_model = Ridge()


# Definimos los hiperparámetros a probar (valores de alpha)
param_grid = {
    'alpha': [0.01, 0.1, 1, 10, 100, 1000]
}
# Configuramos GridSearchCV con validación cruzada de 5 folds
grid_search = GridSearchCV(estimator=ridge_model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_erro


# Entrenaremos el modelo con GridSearchCV en el conjunto de entrenamiento
grid_search.fit(X_train, y_train)
```




```
# Con el siguiente codigo obtendremos los mejores hiperparámetros y el rendimiento asociado
best_alpha = grid_search.best_params_['alpha']
best_score = -grid_search.best_score_  # Eliminaremos el signo negativo para obtener MSE positivo

print("Mejor valor de alpha:", best_alpha)
print("Mejor puntuación (MSE) en validación cruzada:", best_score)
```

```
Mejor valor de alpha: 10
Mejor puntuación (MSE) en validación cruzada: 0.4445030288069199
```

b. Evaluación del mejor modelo

Una vez que encontramos los mejores hiperparámetros, usamos el modelo optimizado para predecir en los datos de prueba y evaluamos su rendimiento.

```
from sklearn.metrics import mean_squared_error, r2_score

# Obtener el mejor modelo del GridSearch
mejor_modelo = grid_search.best_estimator_

# Hacer predicciones en el conjunto de prueba con el modelo optimizado
y_pred_optimizado = mejor_modelo.predict(X_test)

# Evaluar el rendimiento con MSE y R²
mse_optimizado = mean_squared_error(y_test, y_pred_optimizado)
r2_optimizado = r2_score(y_test, y_pred_optimizado)

print("Error cuadrático medio (MSE) del modelo optimizado:", mse_optimizado)
print("Coeficiente de determinación (R²) del modelo optimizado:", r2_optimizado)
```

```
Error cuadrático medio (MSE) del modelo optimizado: 0.6615311447689861
Coeficiente de determinación (R²) del modelo optimizado: 0.5135009620136943
```

Podemos observar que se ha optimizado el MSE , pero el Coeficiente de determinacion ha bajado un poco.

Haz doble clic (o pulsa Intro) para editar

c. Comparación de rendimiento

Ahora, comparemos el rendimiento del modelo optimizado con GridSearchCV y el modelo original de regresión lineal.

```
# Evaluación del modelo de regresión lineal original
y_pred_lineal = modelo.predict(X_test)
mse_lineal = mean_squared_error(y_test, y_pred_lineal)
r2_lineal = r2_score(y_test, y_pred_lineal)

print("Modelo de Regresión Lineal:")
print("MSE:", mse_lineal)
print("R²:", r2_lineal)
```

```
Modelo de Regresión Lineal:
MSE: 0.5791649054738045
R²: 0.574074219216345
```

```
# Evaluación del modelo optimizado de Ridge
y_pred_ridge = mejor_modelo.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print("\nModelo de Regresión Ridge Optimizado:")
print("MSE:", mse_ridge)
print("R²:", r2_ridge)
```

```

Modelo de Regresión Ridge Optimizado:
MSE: 0.6615311447689861
R²: 0.5135009620136943
```

```
# Comparación
print("\nComparación de Modelos:")
print(f"Diferencia en MSE (Lineal - Ridge): {mse_lineal - mse_ridge}")
print(f"Diferencia en R² (Ridge - Lineal): {r2_ridge - r2_lineal}")
```