regression-ii-esp

November 11, 2024

1 Análisis de Regresión (II)

En este cuaderno, profundizaremos en cómo el análisis de regresión nos ayuda a **comprender el comportamiento de los datos**, **realizar predicciones** en datos continuos y categóricos, y **seleccionar predictores significativos** en modelos con múltiples variables. Veremos tres tipos de regresión: la **regresión lineal simple**, la **regresión lineal múltiple**, y la **regresión polinómica**. La evaluación de los modelos se realizará de forma cuantitativa mediante el uso de la biblioteca Scikit-learn y de forma visual con las herramientas de Seaborn.

Emplearemos varios conjuntos de datos reales para ilustrar estos conceptos: - Datos macroeconómicos - Predicción de precios en nuevos mercados de vivienda - Extensión del hielo marino y cambio climático - Conjunto de datos de diabetes de Scikit-learn - Conjunto de datos Longley de indicadores macroeconómicos de EE. UU. - Conjunto de datos de publicidad

1.0.1 Contenidos del cuaderno:

- **Regresión Lineal Múltiple**: Extensión de la regresión simple para múltiples variables independientes.
- **Regularización: Ridge y Lasso**: Métodos que reducen el sobreajuste en modelos complejos.
- **Transformación de Datos**: Técnicas para mejorar la calidad del ajuste en modelos no lineales.

Iniciemos con las librerías necesarias y establezcamos una semilla para la reproducibilidad de los resultados.

[1]:
```
# Importar librerías principales
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Definir una semilla para reproducibilidad
seed = 42
```

1.1 Ejemplo 1: Vivienda en Boston

Empezamos con el conjunto de datos de precios de viviendas en Boston, que contiene información sobre diversas características de las propiedades y los vecindarios, y el precio de las viviendas en esas áreas. En este ejemplo, construiremos un modelo de regresión lineal múltiple para predecir el precio de las viviendas basado en varias características.

[2]:
```
# Cargar el conjunto de datos de vivienda en Boston
from sklearn.datasets import fetch_openml

# Utilizamos fetch_openml para cargar el dataset
boston = fetch_openml(data_id=531)
X, y = boston.data, boston.target
features = boston.feature_names

# Mostrar nombres de características, la variable objetivo y la forma de los␣
↪datos
print('Nombres de las características: {}'.format(features))
print('\nNombre de la variable objetivo: {}'.format(boston.target_names))
print('\nForma de los datos: {} {}'.format(X.shape, y.shape))
```

```
Nombres de las características: ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']

Nombre de la variable objetivo: ['MEDV']

Forma de los datos: (506, 13) (506,)
```

En el código anterior, cargamos el conjunto de datos de Boston y extraemos los nombres de las características. Observamos que los datos están compuestos por múltiples características que pueden ser usadas para predecir el precio de las viviendas.

1.1.1 División en conjuntos de entrenamiento y prueba

Dividimos los datos en dos subconjuntos: uno para entrenar el modelo y otro para probar su rendimiento en datos no vistos. Establecemos un tamaño de prueba del 90% para enfatizar el entrenamiento en un subconjunto más pequeño.

[3]:
```
# División del conjunto en datos de entrenamiento y prueba
from sklearn.model_selection import train_test_split

# Convertimos X a un array NumPy de tipo flotante para evitar problemas de tipo
X = np.array(X, dtype=np.float64)
y = np.array(y, dtype=np.float64)

# Dividir el conjunto en datos de entrenamiento (10%) y prueba (90%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9,␣
↪random_state=seed)
```

```
# Verificar las dimensiones de los conjuntos de entrenamiento y prueba
print('Dimensiones de X_train: {}, X_test: {}'.format(X_train.shape, X_test.
 ↪shape))
print('Dimensiones de y_train: {}, y_test: {}'.format(y_train.shape, y_test.
 ↪shape))
```

```
Dimensiones de X_train: (50, 13), X_test: (456, 13)
Dimensiones de y_train: (50,), y_test: (456,)
```

1.1.2 Entrenamiento del Modelo de Regresión Lineal Múltiple

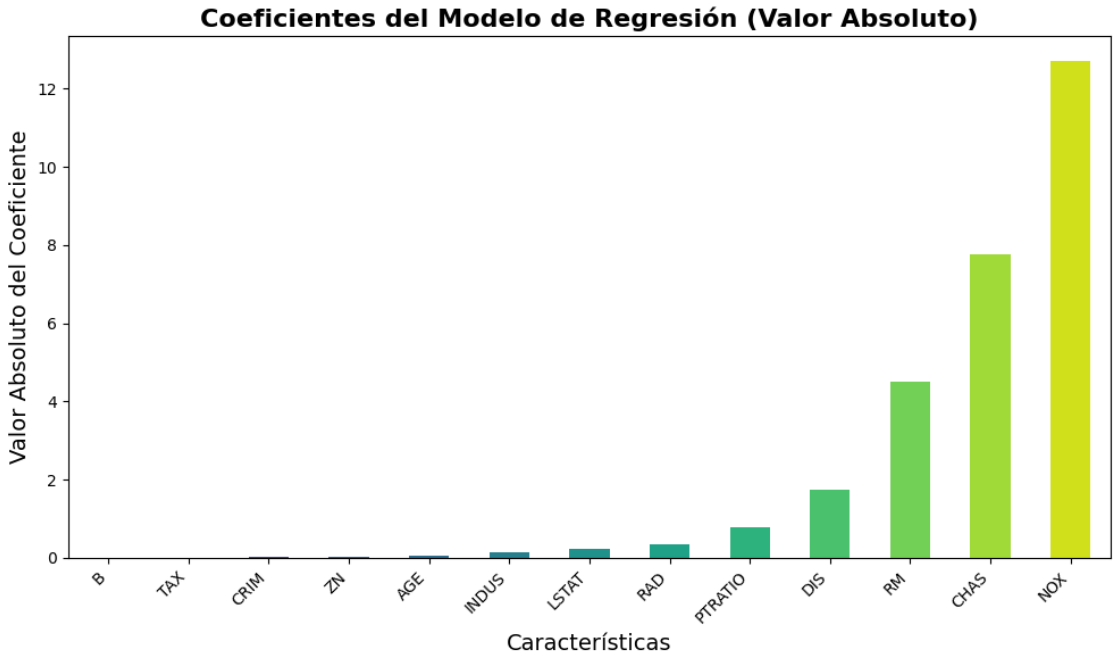
En esta sección, crearemos un modelo de regresión lineal múltiple que toma en cuenta todas las características del conjunto de datos. Calcularemos los coeficientes del modelo, que representan la influencia de cada característica en el precio de la vivienda.

[4]:
```
# Inicializar y ajustar el modelo de regresión lineal múltiple
lr = LinearRegression()  # Instancia del modelo de regresión lineal
lr.fit(X_train, y_train)  # Entrenamiento del modelo con los datos de␣
 ↪entrenamiento

# Obtener los coeficientes de regresión
coefs_lr = pd.Series(np.abs(lr.coef_), index=features).sort_values()  # Valores␣
 ↪absolutos para ordenarlos

# Realizar predicciones sobre el conjunto de prueba
y_test_pred = lr.predict(X_test)
y_train_pred = lr.predict(X_train)  # Predicciones sobre el conjunto de␣
 ↪entrenamiento

# Evaluar el modelo utilizando métricas de desempeño
mse_train = mean_squared_error(y_train, y_train_pred)  # Error cuadrático medio␣
 ↪para entrenamiento
mse_test = mean_squared_error(y_test, y_test_pred)  # Error cuadrático medio␣
 ↪para prueba
r2score_train = lr.score(X_train, y_train)  # Puntaje R^2 en entrenamiento
r2score_test = lr.score(X_test, y_test)  # Puntaje R^2 en prueba

# Mostrar el intercepto y los coeficientes del modelo
print('Intercepto y coeficientes del modelo:\n \nIntercepto: {}\n␣
 ↪\nCoeficientes: {}'.format(lr.intercept_, lr.coef_))

# Mostrar el error cuadrático medio y el puntaje R^2
print('\nError Cuadrático Medio (ECM) en entrenamiento: {}'.format(mse_train))
print('Error Cuadrático Medio (ECM) en prueba: {}'.format(mse_test))
print('\nPuntaje R^2 en entrenamiento: {}'.format(r2score_train))
print('Puntaje R^2 en prueba: {}'.format(r2score_test))
```

```
Intercepto y coeficientes del modelo:

Intercepto: 29.28982588278542

Coeficientes: [-3.52276440e-02  3.74189992e-02 -1.53911832e-01  7.75772273e+00
 -1.27186309e+01  4.51717658e+00 -6.89587505e-02 -1.74595447e+00
  3.47162018e-01 -1.10563851e-02 -7.98303460e-01  8.79070755e-03
 -2.34221766e-01]

Error Cuadrático Medio (ECM) en entrenamiento: 19.861052089005906
Error Cuadrático Medio (ECM) en prueba: 28.01412571526917

Puntaje R^2 en entrenamiento: 0.700743261188567
Puntaje R^2 en prueba: 0.6757580265626979
```

Los coeficientes obtenidos muestran la relación entre cada característica y el precio de las viviendas, con el intercepto representando el valor base cuando todas las características son cero. El puntaje R^2 nos indica qué tan bien se ajusta el modelo a los datos, siendo 1 un ajuste perfecto.

[5]:
```python
# Crear la gráfica de barras ordenada por valores absolutos de los coeficientes
plt.figure(figsize=(10, 6))

# Ordenar los coeficientes de menor a mayor (invirtiendo el orden)
coefs_lr.sort_values(ascending=True).plot(kind='bar', color=sns.
 ↪color_palette("viridis", n_colors=len(coefs_lr)))

# Título y etiquetas de ejes
plt.title('Coeficientes del Modelo de Regresión (Valor Absoluto)', fontsize=16,␣
 ↪weight='bold')
plt.xlabel('Características', fontsize=14)
plt.ylabel('Valor Absoluto del Coeficiente', fontsize=14)

# Rotación y alineación de las etiquetas para mejor legibilidad
plt.xticks(rotation=45, ha='right')

# Mostrar la gráfica
plt.tight_layout()  # Ajustar el espaciado para evitar solapamiento de elementos
plt.show()
```



La gráfica anterior permite ver la importancia relativa de cada característica en el modelo. Los valores absolutos nos muestran la magnitud del impacto sin importar la dirección (positiva o negativa).

[6]:
```
# print(boston.DESCR)
```

Autor:
Fuente: Desconocida - Fecha desconocida
Por favor citar:

Datos de precios de viviendas en Boston de Harrison, D. y Rubinfeld, D.L. "Precios hedónicos y la demanda de aire limpio", *J. Environ. Economics & Management*, vol.5, 81-102, 1978. También utilizado en Belsley, Kuh y Welsch, *Regression Diagnostics*, Wiley, 1980. Nota: En este último, se emplean varias transformaciones en la tabla de las páginas 244-261.

Variables en orden: - **CRIM**: tasa de criminalidad per cápita por ciudad - **ZN**: proporción de terreno residencial zonificado para lotes de más de 25,000 pies cuadrados - **INDUS**: proporción de acres de negocios no minoristas por ciudad - **CHAS**: variable ficticia del Río Charles (= 1 si la zona limita con el río; 0 en caso contrario) - **NOX**: concentración de óxidos de nitrógeno (partes por 10 millones) - **RM**: número promedio de habitaciones por vivienda - **AGE**: proporción de unidades ocupadas por propietarios construidas antes de 1940 - **DIS**: distancias ponderadas a cinco centros de empleo de Boston - **RAD**: índice de accesibilidad a autopistas radiales - **TAX**: tasa de impuesto a la propiedad de valor completo por cada $10,000 - **PTRATIO**: proporción alumno-profesor por ciudad - **B**: 1000(Bk - 0.63)^2 donde Bk es la proporción de personas negras por ciudad - **LSTAT**: % de población de estatus socioeconómico bajo - **MEDV**: Valor mediano de las viviendas ocupadas por propietarios, en miles de dólares

Información sobre el conjunto de datos: - **TIPO DE CLASE**: numérico - **ÍNDICE DE CLASE**:

último

Descargado de openml.org.

1.2 Regularización y Selección de Características

En modelos de regresión lineal múltiple, incluir todas las características sin restricciones puede llevar a un modelo excesivamente complejo y con tendencia al sobreajuste, especialmente cuando hay variables poco informativas o redundantes. Esto dificulta la generalización del modelo y puede afectar su rendimiento en datos nuevos. Para abordar estos problemas, aplicamos técnicas de **regularización** como **Ridge** y **LASSO** que permiten ajustar la complejidad del modelo y facilitar la **selección de características**.

- **Ridge Regression**: También conocida como regularización L2, esta técnica agrega una penalización basada en la suma de los cuadrados de los coeficientes de regresión. Esta penalización controla la magnitud de los coeficientes, evitando que crezcan demasiado. Ridge es especialmente útil cuando todas las características son relevantes para la predicción, pero se busca reducir el sobreajuste al hacer el modelo menos sensible a las fluctuaciones de los datos.
- **LASSO (Least Absolute Shrinkage and Selection Operator)**: En contraste con Ridge, LASSO aplica una penalización sobre la suma de los valores absolutos de los coeficientes (regularización L1). Esto permite reducir ciertos coeficientes a cero, eliminando así las variables menos informativas y simplificando el modelo. LASSO es útil en situaciones en las que se espera que solo algunas características sean significativas, y el resto puedan eliminarse.

Mediante Ridge y LASSO, optimizamos el modelo enfocándonos en las características más relevantes y controlando su complejidad para evitar sobreajuste, mejorando así la precisión en la predicción.

1.2.1 Regularización L2: Regresión Ridge

La regularización Ridge impone una penalización a la norma L2 de los coeficientes, manteniendo todos los valores de los coeficientes pequeños sin llegar a eliminarlos. Este método reduce la sensibilidad del modelo a pequeños cambios en el conjunto de datos, disminuyendo el riesgo de sobreajuste y manteniendo todas las características.

La fórmula de la Regresión Ridge es:

$$minimize\left(\sum_{i=0}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \alpha\sum_{j=1}^{p}\beta_j^2\right)$$

donde: - y_i es el valor observado para la i-ésima instancia. - β_0 es el término independiente. - β_j son los coeficientes de regresión de cada variable j. - x_{ij} es el valor de la característica j en la instancia i. - α es el parámetro de regularización, que controla la magnitud de la penalización y, por ende, el nivel de ajuste del modelo.

Un valor alto de α incrementa la penalización, forzando los coeficientes hacia cero y simplificando el modelo, mientras que un valor bajo de α permite coeficientes más grandes y, potencialmente, un ajuste más preciso a los datos de entrenamiento (aunque con mayor riesgo de sobreajuste).

Ejemplo de Regresión Ridge Supongamos que queremos predecir el precio de una casa (y_i) usando variables como tamaño, número de habitaciones y ubicación. Si aplicamos Ridge, el modelo ajustado tiene la forma:

$$\widehat{y_i} = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$$

La penalización $\alpha \sum_{j=1}^{p} \beta_j^2$ evita que los coeficientes β_j crezcan demasiado, mitigando el riesgo de sobreajuste y ayudando a que el modelo generalice mejor en datos no observados. Este método es útil cuando se quiere utilizar todas las variables explicativas, pero con control sobre su influencia.

1.2.2 Regularización L1: Regresión Lasso

En muchos problemas de predicción, existen variables irrelevantes o ruidosas que no contribuyen significativamente al modelo y pueden dificultar su rendimiento. La Regresión Lasso es una técnica de regularización L1 que, además de reducir los coeficientes, puede reducir algunos de ellos exactamente a cero, eliminando variables irrelevantes y simplificando el modelo. Esta propiedad permite que Lasso actúe como un mecanismo de selección automática de características.

La fórmula de la Regresión Lasso es:

$$minimize \left(\sum_{i=0}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right)$$

donde: - y_i es el valor observado para la i-ésima instancia. - β_0 es el término independiente. - β_j son los coeficientes de regresión de cada variable j. - x_{ij} es el valor de la característica j en la instancia i. - α es el parámetro de regularización que controla la magnitud de la penalización.

Ejemplo de Regresión Lasso Imaginemos que queremos predecir el nivel de glucosa en sangre (y_i) usando variables como edad, índice de masa corporal, niveles de insulina y presión arterial. Aplicando Lasso, el modelo ajustado podría tener la forma:

$$\widehat{y_i} = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$$

La penalización $\alpha \sum_{j=1}^{p} |\beta_j|$ ayuda a reducir algunos coeficientes a cero si no son informativos para predecir la glucosa en sangre, dejando solo las variables relevantes y simplificando el modelo.

1.2.3 Comparación de Ridge y Lasso

- **Ridge** se utiliza cuando todas las características son importantes y queremos controlar su influencia sin eliminarlas. Ridge es adecuado cuando no esperamos que ninguna característica sea completamente irrelevante, ya que ajusta la magnitud de los coeficientes, pero no los reduce a cero.
- **Lasso** es más adecuado cuando se sospecha que varias características no son informativas, ya que facilita la selección de variables relevantes al reducir algunos coeficientes a cero.

En la práctica, **Elastic Net** combina las ventajas de ambos métodos mediante una mezcla de regularización L1 y L2. Esto resulta útil en situaciones en las que existe una gran cantidad de variables correlacionadas, pues combina la simplicidad de Lasso y la capacidad de Ridge de reducir todos los coeficientes sin eliminarlos.

La fórmula de Elastic Net es:

$$minimize \left(\sum_{i=0}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \alpha \sum_{j=1}^{p} \left(\frac{1-\lambda}{2} \beta_j^2 + \lambda |\beta_j| \right) \right)$$

donde: - λ es un valor entre 0 y 1 que controla el balance entre Lasso (L1) y Ridge (L2).

Elastic Net es especialmente útil cuando hay muchas características con alta correlación, ya que puede manejar mejor situaciones en las que Lasso podría seleccionar solo una de las variables correlacionadas.

1.2.4 Interpretación Geométrica de la Regularización

La regularización tiene una interpretación geométrica basada en la restricción impuesta a los coeficientes, lo que influye en el tipo de solución que se obtiene:

- **Regularización L1 (Lasso)**: La penalización L1 genera una región de restricción en forma de diamante en dos dimensiones. Las esquinas de esta figura hacen más probable que algunos coeficientes se reduzcan exactamente a cero, lo que da lugar a soluciones esparsas en las que muchas variables son descartadas.
- **Regularización L2 (Ridge)**: La penalización L2 genera una región de restricción en forma de círculo (o elipse en dimensiones superiores). Esto limita el tamaño de los coeficientes pero mantiene todos los valores dentro de una magnitud controlada, sin reducirlos a cero.

El siguiente diagrama ilustra estas diferencias geométricas: el panel izquierdo muestra la restricción L1 (diamante) y el derecho la L2 (círculo). Las elipses representan las líneas de error cuadrático sin regularización, y los puntos de intersección con las restricciones representan las soluciones regulares.

En conclusión, la elección entre Ridge, Lasso y Elastic Net dependerá de la naturaleza de los datos y de la necesidad de mantener ciertas características o reducir el modelo a solo las variables más importantes.

Más información aquí: tutorial-ridge-lasso-elastic-net

1.3 Regresión Ridge

En esta sección, implementamos un modelo de regresión lineal con regularización L2, utilizando la técnica de Regresión Ridge. Esta técnica se aplica para evitar el sobreajuste al penalizar los coeficientes más grandes, forzando a que se mantengan más pequeños. El proceso es el siguiente:

1. **Inicialización del modelo**: Creamos una instancia del modelo de regresión Ridge utilizando `linear_model.Ridge(alpha=1)`. El parámetro `alpha` es clave porque determina la magnitud de la regularización. A medida que `alpha` aumenta, la penalización sobre los coeficientes también se incrementa, lo que puede llevar a una mayor simplificación del modelo.

2. **Ajuste del modelo**: Usamos el método `fit` para entrenar el modelo con los datos de entrenamiento `X_train` (matriz de características) y `y_train` (vector de respuestas). Este paso ajusta los coeficientes de las variables predictoras en función de los datos.

3. **Extracción de los coeficientes**: Una vez entrenado el modelo, obtenemos los coeficientes de la regresión con `ridge.coef_`. Al aplicar la regularización, estos coeficientes se verán reducidos, en especial aquellos correspondientes a características menos relevantes.

4. **Predicción**: Se hace uso del modelo ajustado para predecir los valores de la variable dependiente sobre el conjunto de prueba `X_test`. Los valores predichos se almacenan en `y_test_pred_ridge`.

5. **Evaluación del modelo**:

   - **Error Cuadrático Medio (MSE)**: Calculamos el MSE sobre los datos de prueba (`y_test_pred_ridge`). Este valor nos indica cuán cerca están las predicciones del modelo de los valores reales, siendo más bajo el MSE sinónimo de mejor ajuste.
   - **Coeficiente de determinación (R^2)**: Calculamos el R^2, tanto en los datos de entrenamiento como en los de prueba, para medir qué tan bien el modelo es capaz de explicar la variabilidad de la variable dependiente.

6. **Mostrar resultados**:

   - Se imprime el intercepto (`ridge.intercept_`) y los coeficientes del modelo (`ridge.coef_`).
   - Mostramos el MSE para los datos de entrenamiento y prueba, así como el R^2 en ambos conjuntos. Esto nos ayudará a entender la precisión y el rendimiento del modelo.
   - **Coeficientes**: Visualizamos los coeficientes del modelo en un gráfico de barras, ordenados por su valor absoluto. Este gráfico nos permite ver qué variables son más relevantes para el modelo.

Este proceso permite crear un modelo robusto que se ajusta bien a los datos, pero evitando sobreajustarse a ellos gracias a la regularización L2 (Ridge).

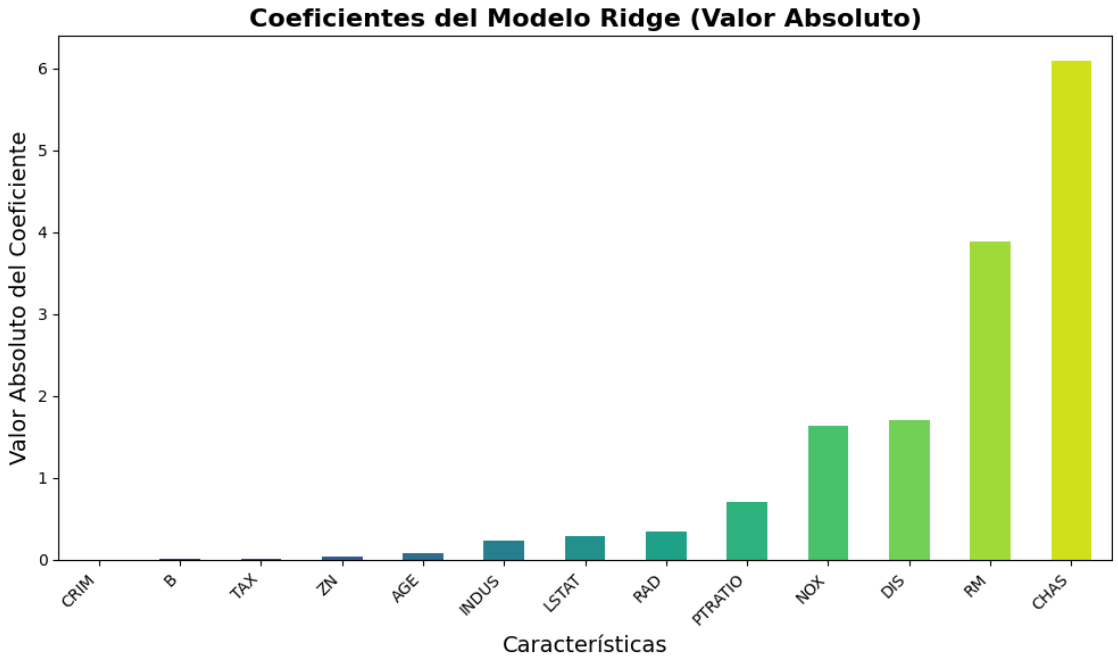
1.3.1 Código de implementación:

[7]:
```
## Regresión Ridge
from sklearn import linear_model  # Para la regresión Ridge

ridge = linear_model.Ridge(alpha=1)  # Inicializamos el regresor Ridge con una␣
 ↪penalización de 1
ridge.fit(X_train, y_train)  # Ajustamos el modelo a los datos de entrenamiento

# Extraemos los coeficientes y los ordenamos en función de su valor absoluto
coefs_ridge = pd.Series(np.abs(ridge.coef_), index=features).
 ↪sort_values(ascending=False)

# Realizamos la predicción en el conjunto de prueba y entrenamiento
y_train_pred_ridge = ridge.predict(X_train)
y_test_pred_ridge = ridge.predict(X_test)
```

```
# Evaluación del modelo: calculamos el Error Cuadrático Medio (MSE) y el R^2
mse_ridge_train = mean_squared_error(y_train, y_train_pred_ridge)  # MSE en␣
 ↪entrenamiento
mse_ridge_test = mean_squared_error(y_test, y_test_pred_ridge)  # MSE en prueba
r2score_ridge_train = ridge.score(X_train, y_train)  # R^2 en entrenamiento
r2score_ridge_test = ridge.score(X_test, y_test)  # R^2 en prueba

# Mostramos los resultados:
print('\nIntercepto del modelo:', ridge.intercept_)
print('\nCoeficientes del modelo Ridge:\n', ridge.coef_)
print('\nError Cuadrático Medio (MSE) en entrenamiento:', mse_ridge_train)
print('Error Cuadrático Medio (MSE) en prueba:', mse_ridge_test)
print('\nPuntaje R^2 en entrenamiento:', r2score_ridge_train)
print('Puntaje R^2 en prueba:', r2score_ridge_test)

# Graficamos los coeficientes ordenados por valor absoluto
plt.figure(figsize=(10, 6))

# Ordenar los coeficientes de menor a mayor (invirtiendo el orden)
coefs_ridge.sort_values(ascending=True).plot(kind='bar', color=sns.
 ↪color_palette("viridis", n_colors=len(coefs_ridge)))

# Añadimos el título y etiquetas a los ejes
plt.title('Coeficientes del Modelo Ridge (Valor Absoluto)', fontsize=16,␣
 ↪weight='bold')
plt.xlabel('Características', fontsize=14)
plt.ylabel('Valor Absoluto del Coeficiente', fontsize=14)

# Ajustamos las etiquetas del eje x para mejorar la legibilidad
plt.xticks(rotation=45, ha='right')

# Mostramos la gráfica con un diseño ajustado
plt.tight_layout()  # Ajuste del espaciado para evitar solapamientos
plt.show()
```

```
Intercepto del modelo: 29.192905256841595

Coeficientes del modelo Ridge:
 [ 2.08447424e-03  4.30036834e-02 -2.31040992e-01  6.08837583e+00
 -1.64018349e+00  3.88967991e+00 -8.14349726e-02 -1.70761994e+00
  3.41769810e-01 -1.40187792e-02 -7.07453930e-01  7.87323884e-03
 -2.97266211e-01]

Error Cuadrático Medio (MSE) en entrenamiento: 20.47842619653215
Error Cuadrático Medio (MSE) en prueba: 28.43004245069892
```

```
Puntaje R^2 en entrenamiento: 0.6914409663646591
Puntaje R^2 en prueba: 0.670944110024591
```



1.4 Regresión Lasso

En este bloque de código implementamos la regresión Lasso, que también es una técnica de regularización para evitar el sobreajuste, pero a diferencia de Ridge, Lasso aplica la regularización L1. Esto significa que Lasso puede llevar a algunos coeficientes a ser exactamente cero, lo que puede ser útil para realizar una selección automática de características. El proceso es el siguiente:

1. **Inicialización del modelo Lasso**: Creamos un regresor Lasso utilizando `linear_model.Lasso(alpha=1)`. Al igual que en Ridge, el parámetro `alpha` controla la magnitud de la regularización, pero en este caso, Lasso tiene la capacidad de hacer que algunos coeficientes se vuelvan exactamente cero, eliminando efectivamente algunas variables del modelo.

2. **Ajuste del modelo**: El modelo se ajusta a los datos de entrenamiento `X_train` (matriz de características) y `y_train` (vector de respuestas) utilizando el método `fit`. Este paso calcula los coeficientes del modelo, penalizando aquellos más grandes.

3. **Coeficientes de regresión**: Una vez que el modelo está ajustado, obtenemos los coeficientes con `lasso.coef_`. Al aplicar la regularización L1, algunos coeficientes pueden ser exactamente cero, lo que indica que las características correspondientes no contribuyen al modelo.

4. **Predicción**: Usamos el modelo ajustado para realizar predicciones en los conjuntos de datos de prueba (`X_test`) y de entrenamiento (`X_train`). Esto nos permitirá comparar el rendimiento del modelo tanto en datos nuevos como en los datos sobre los que fue entrenado.

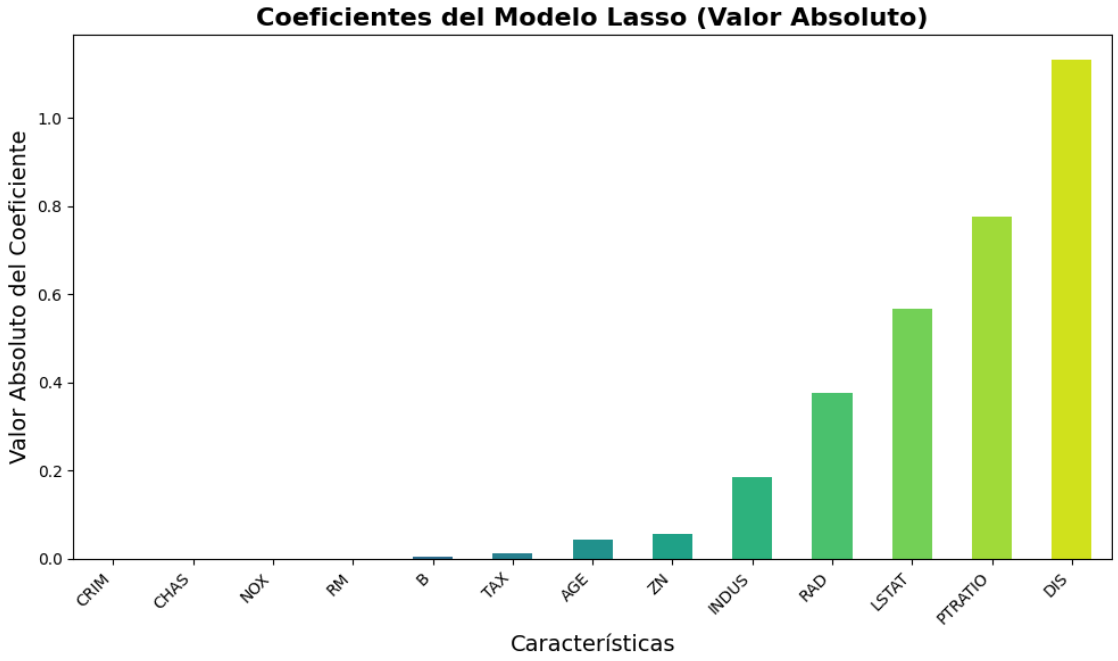
5. **Evaluación del modelo**: Al igual que con la regresión Ridge, evaluamos el rendimiento del modelo utilizando:
    - **Error Cuadrático Medio (MSE)**: Este valor nos indica cuán cerca están las predicciones del modelo de los valores reales. Calculamos el MSE tanto en los datos de entrenamiento como en los de prueba.
    - **Coeficiente de determinación (R^2)**: Nos muestra qué tan bien el modelo es capaz de predecir los valores de la variable dependiente, proporcionando una medida de la calidad del ajuste.
6. **Mostrar resultados**:
    - Se imprime el **intercepto** del modelo, que es el valor de la variable dependiente cuando todas las variables independientes son cero.
    - Mostramos el **MSE**, que nos ayuda a evaluar la precisión de las predicciones.
    - Presentamos el puntaje $\mathbf{R^2}$ para ambos conjuntos de datos (entrenamiento y prueba), lo que nos da una idea de la capacidad predictiva del modelo.

1.4.1 Código de implementación:

[8]:
```
## Regresión Lasso
lasso = linear_model.Lasso(alpha=1)  # Inicializamos el regresor Lasso con una␣
 ↪penalización de 1
lasso.fit(X_train, y_train)  # Ajustamos el modelo a los datos de entrenamiento

# Extraemos los coeficientes y los ordenamos en función de su valor absoluto
coefs_lasso = pd.Series(np.abs(lasso.coef_), index=features).
 ↪sort_values(ascending=False)

# Realizamos la predicción en los conjuntos de entrenamiento y prueba
y_train_pred_lasso = lasso.predict(X_train)
y_test_pred_lasso = lasso.predict(X_test)

# Evaluación del modelo: calculamos el Error Cuadrático Medio (MSE) y el R^2
mse_lasso_train = mean_squared_error(y_train, y_train_pred_lasso)  # MSE en␣
 ↪entrenamiento
mse_lasso_test = mean_squared_error(y_test, y_test_pred_lasso)  # MSE en prueba
r2score_lasso_train = lasso.score(X_train, y_train)  # R^2 en entrenamiento
r2score_lasso_test = lasso.score(X_test, y_test)  # R^2 en prueba

# Mostramos los resultados:
print('\nIntercepto del modelo:', lasso.intercept_)  # Imprimir el intercepto␣
 ↪del modelo Lasso
print('\nCoeficientes del modelo Lasso:\n', lasso.coef_)  # Imprimir los␣
 ↪coeficientes del modelo Lasso
print('\nError Cuadrático Medio (MSE) en entrenamiento:', mse_lasso_train)
print('Error Cuadrático Medio (MSE) en prueba:', mse_lasso_test)
print('\nPuntaje R^2 en entrenamiento:', r2score_lasso_train)
```

```
print('Puntaje R^2 en prueba:', r2score_lasso_test)

# Graficamos los coeficientes ordenados por valor absoluto
plt.figure(figsize=(10, 6))

# Ordenar los coeficientes de menor a mayor (invirtiendo el orden)
coefs_lasso.sort_values(ascending=True).plot(kind='bar', color=sns.
 ↪color_palette("viridis", n_colors=len(coefs_lasso)))

# Añadimos el título y etiquetas a los ejes
plt.title('Coeficientes del Modelo Lasso (Valor Absoluto)', fontsize=16,␣
 ↪weight='bold')
plt.xlabel('Características', fontsize=14)
plt.ylabel('Valor Absoluto del Coeficiente', fontsize=14)

# Ajustamos las etiquetas del eje x para mejorar la legibilidad
plt.xticks(rotation=45, ha='right')

# Mostramos la gráfica con un diseño ajustado
plt.tight_layout()  # Ajuste del espaciado para evitar solapamientos
plt.show()
```

```
Intercepto del modelo: 52.92068268504159

Coeficientes del modelo Lasso:
 [-0.          0.05677039 -0.18663206  0.         -0.          0.
 -0.04260378 -1.13153835  0.3759946  -0.01409795 -0.77672856  0.00521161
 -0.56721946]

Error Cuadrático Medio (MSE) en entrenamiento: 28.381490900599587
Error Cuadrático Medio (MSE) en prueba: 33.356838420721296

Puntaje R^2 en entrenamiento: 0.5723614050525907
Puntaje R^2 en prueba: 0.6139202334175015
```



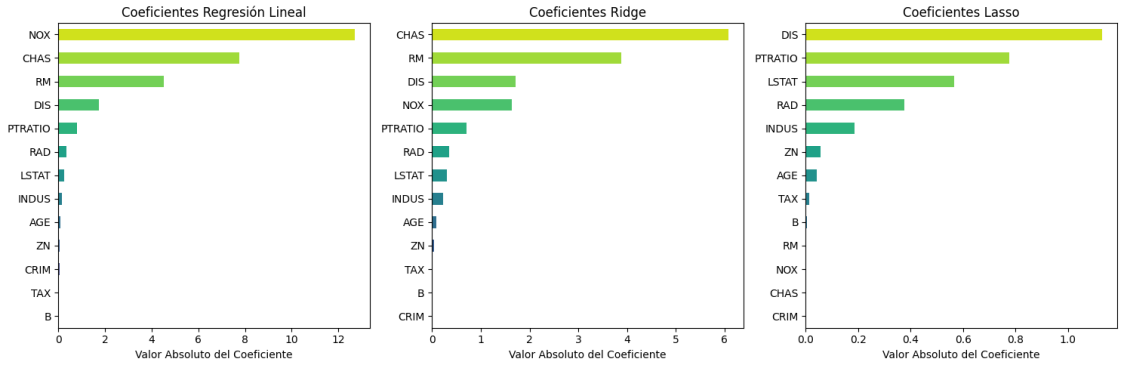
Comparar los resultados: En este bloque comparamos los resultados obtenidos a partir de los modelos de Regresión Lineal, Ridge y Lasso, evaluando sus coeficientes y puntajes de desempeño en los conjuntos de entrenamiento y prueba.

1. **Coeficientes dispersos (sparse)**:
   - En la regresión Lasso, debido a su regularización L1, algunos coeficientes pueden volverse exactamente cero. Esto da lugar a un modelo más "disperso" (sparse), en el que solo algunas características son seleccionadas como relevantes.
   - En comparación, la regresión Ridge tiende a reducir los coeficientes, pero no a cero, lo que significa que todos los predictores tienen algún impacto en la predicción.
2. **Comparación de puntajes (score)**:
   - Comparamos el rendimiento de los tres modelos utilizando el **coeficiente de determinación R^2**, que mide la proporción de la varianza de la variable dependiente que puede ser explicada por el modelo.
   - Los puntajes R^2 en los conjuntos de **entrenamiento** y **prueba** nos permiten evaluar si el modelo está sobreajustando (overfitting) o subajustando (underfitting).

A continuación, se genera una gráfica para comparar visualmente los coeficientes de cada modelo:

[9]:
```
# Crear una figura de tamaño adecuado para visualizar los coeficientes
f = plt.figure(figsize=(15,5))

# Agregar los subgráficos
ax1 = f.add_subplot(131)  # Primer gráfico para los coeficientes de la␣
 ↪regresión lineal
ax2 = f.add_subplot(132)  # Segundo gráfico para los coeficientes de Ridge
```

```
ax3 = f.add_subplot(133)  # Tercer gráfico para los coeficientes de Lasso

# Graficar los coeficientes de cada modelo utilizando barras horizontales, ordenados de manera ascendente (menor a mayor)
coefs_lr.sort_values(ascending=True).plot(kind="barh", title='Coeficientes Regresión Lineal', ax=ax1, color=sns.color_palette("viridis", n_colors=len(coefs_lr)))  # Coeficientes del modelo de regresión lineal
coefs_ridge.sort_values(ascending=True).plot(kind="barh", title='Coeficientes Ridge', ax=ax2, color=sns.color_palette("viridis", n_colors=len(coefs_ridge)))  # Coeficientes del modelo Ridge
coefs_lasso.sort_values(ascending=True).plot(kind="barh", title='Coeficientes Lasso', ax=ax3, color=sns.color_palette("viridis", n_colors=len(coefs_lasso)))  # Coeficientes del modelo Lasso

# Establecer etiquetas del eje X para cada gráfico
ax1.set_xlabel('Valor Absoluto del Coeficiente')  # Etiqueta del eje X para el gráfico de regresión lineal
ax2.set_xlabel('Valor Absoluto del Coeficiente')  # Etiqueta del eje X para el gráfico de Ridge
ax3.set_xlabel('Valor Absoluto del Coeficiente')  # Etiqueta del eje X para el gráfico de Lasso

# Ajustar el espaciado entre los gráficos para que no se superpongan
plt.tight_layout()

# Mostrar la figura con los tres gráficos comparativos
plt.show()
```



3. **Variables no importantes y la más importante**:
   - Lasso realiza una selección de características al forzar algunos coeficientes a cero. Esto nos permite identificar qué variables no son relevantes para el modelo.
   - Al examinar el índice de `coefs_lasso`, podemos listar las variables que no tienen influencia en el modelo (coeficiente igual a cero) y la variable más importante (coeficiente no nulo más grande).

[10]:
```
print('Variables no importantes: {}'.format(coefs_lasso.index[coefs_lasso == 0].
 ↪values))  # Imprimimos las variables con coeficientes cero (no importantes)
print('Variable más importante: {}'.format(coefs_lasso.index[-1]))  #␣
 ↪Imprimimos la variable más importante (con el coeficiente más alto)
```

```
Variables no importantes: ['CRIM' 'CHAS' 'NOX' 'RM']
Variable más importante: RM
```

4. **Comparación de puntajes R^2 y MSE entre modelos**:

En esta sección, vamos a comparar los resultados obtenidos de los tres modelos (Regresión Lineal, Ridge y Lasso) en términos de **puntaje R^2** (tanto en el conjunto de **entrenamiento** como en el de **prueba**) y **Error Cuadrático Medio (MSE)**, para evaluar el rendimiento de cada modelo.

Se calcularán las siguientes métricas para cada uno de los tres modelos:

1. **Puntaje R^2**:
   - **R^2 Entrenamiento**: Indica el ajuste del modelo a los datos de entrenamiento.
   - **R^2 Prueba**: Indica cómo se comporta el modelo con datos no vistos (prueba).
   - **Δ R^2**: Diferencia entre los puntajes R^2 de entrenamiento y prueba.
2. **MSE (Error Cuadrático Medio)**:
   - **MSE Entrenamiento**: Mide la calidad de las predicciones del modelo sobre los datos de entrenamiento.
   - **MSE Prueba**: Mide la calidad de las predicciones del modelo sobre los datos de prueba.
   - **Δ MSE**: Diferencia entre el MSE de entrenamiento y prueba.

Para hacer esto, se construye un DataFrame `df_scores` con estos valores, permitiendo una comparación entre los modelos.

[11]:
```
# Cálculo de las métricas R² y MSE para cada modelo
scores = [
    [r2score_train, r2score_test, r2score_train - r2score_test, mse_train,␣
 ↪mse_test, mse_train - mse_test],  # Regresión Lineal
    [r2score_ridge_train, r2score_ridge_test, r2score_ridge_train -␣
 ↪r2score_ridge_test, mse_ridge_train, mse_ridge_test, mse_ridge_train -␣
 ↪mse_ridge_test],  # Ridge
    [r2score_lasso_train, r2score_lasso_test, r2score_lasso_train -␣
 ↪r2score_lasso_test, mse_lasso_train, mse_lasso_test, mse_lasso_train -␣
 ↪mse_lasso_test]   # Lasso
]

# Creación del DataFrame con las métricas calculadas
df_scores = pd.DataFrame(scores,
                         columns=["R² Entrenamiento", "R² Prueba", "Δ R²␣
 ↪(Entrenamiento - Prueba)",
                                  "MSE Entrenamiento", "MSE Prueba", "Δ MSE␣
 ↪(Entrenamiento - Prueba)"],
                         index=["Sin regularización", "Ridge", "Lasso"])
```

```
# Mostramos el DataFrame con los puntajes R² y MSE para comparar entre los␣
 ↪modelos
df_scores  # Visualizamos la comparación de puntajes R² y MSE entre los tres␣
 ↪modelos, incluyendo sus diferencias
```

[11]:

	R^2 Entrenamiento	R^2 Prueba	\
Sin regularización	0.700743	0.675758	
Ridge	0.691441	0.670944	
Lasso	0.572361	0.613920	

	Δ R^2 (Entrenamiento - Prueba)	MSE Entrenamiento	\
Sin regularización	0.024985	19.861052	
Ridge	0.020497	20.478426	
Lasso	-0.041559	28.381491	

	MSE Prueba	Δ MSE (Entrenamiento - Prueba)
Sin regularización	28.014126	-8.153074
Ridge	28.430042	-7.951616
Lasso	33.356838	-4.975348

Transforma y Predice:

El término "predictor" puede tener varios significados dependiendo del contexto. En algunos textos, "predictor" puede referirse a un modelo que podría estar formado por una combinación de modelos, como cuando se combinan los modelos de Ridge y Lasso en un enfoque de regresión múltiple. Debido a que este tipo de combinación no presenta una definición universalmente aceptada y su forma varía dependiendo del tipo de combinaciones que se realicen, generalmente se hace referencia a estos modelos no por un nombre específico, sino por la función que realizan: predecir el valor de la variable objetivo. Sin embargo, en el contexto del análisis de datos y modelado estadístico, hacer referencia a un "predictor" o "predictores" podría tener otro significado, ya que una definición ampliamente aceptada y utilizada para el término es la siguiente:

Predictor(es) (Característica(s) o Variable(s) Independiente(s)): Son las variables de entrada que el modelo utiliza para hacer predicciones. Estas variables pueden ser de distintos tipos según el problema que se esté abordando: por ejemplo, edad, ingresos, temperatura, entre otros. En un conjunto de datos, los predictores corresponden generalmente a las columnas de la matriz X, que representan las características o atributos que el modelo emplea para prever o estimar el valor de la variable objetivo. Cada predictor es considerado una dimensión del espacio de entrada del modelo.

Modelo: Es el algoritmo o conjunto de reglas matemáticas que aprende a partir de los datos. El modelo se ajusta para aprender las relaciones entre los predictores y la variable objetivo durante el proceso de entrenamiento. Ejemplos comunes de modelos incluyen la regresión lineal, el modelo de Ridge, el modelo de Lasso, las redes neuronales, árboles de decisión, entre otros. En general, un modelo es responsable de identificar patrones o relaciones en los datos para realizar predicciones o clasificaciones.

Pesos (Coeficientes del Modelo): Los pesos, también llamados coeficientes en el contexto de la regresión, son los parámetros que el modelo ajusta durante el proceso de entrenamiento.

Estos coeficientes determinan la influencia de cada predictor en la predicción final. En el caso de un modelo de regresión lineal, por ejemplo, los pesos son los coeficientes que multiplican a cada predictor (o variable independiente) para generar la predicción de la variable dependiente (o variable objetivo). Los valores de los pesos son aprendidos durante el proceso de optimización del modelo.

En resumen, el término "predictor" o "predictores" podría hacer referencia a las características o variables independientes que se utilizan como entrada en un modelo, o a modelos que no tienen una definición universalmente aceptada, debido a ser combinaciones o variaciones de otros bien establecidos. Sin embargo, el contexto siempre debe guiar cómo se interpreta este término, ya que puede adquirir diferentes significados dependiendo del tipo de modelo, análisis y metodología empleada durante el estudio realizado.

1.4.2 Análisis de los predictores

Antes de aplicar los modelos de regresión, es importante realizar un análisis preliminar sobre los predictores (las variables independientes) para entender sus características estadísticas. Esto nos permitirá conocer mejor la naturaleza de los datos y asegurarnos de que los modelos se apliquen de manera adecuada. En este bloque de código, se realiza un análisis de los predictores en el conjunto de datos. Aquí está el desglose de las estadísticas calculadas:

1. **Mínimo**: El valor más bajo de la característica en el conjunto de datos. Esto es útil para identificar posibles valores atípicos hacia el extremo inferior.
2. **Máximo**: El valor más alto que toma una característica en el conjunto de datos. Nos da una idea del rango superior de los datos.
3. **Media**: El promedio de los valores de la característica, lo que nos da una indicación de la tendencia central de los datos.
4. **Varianza**: Medida que indica qué tan dispersos están los valores con respecto a la media. Una varianza alta indica que los datos están más dispersos, mientras que una varianza baja sugiere que los valores están más concentrados alrededor de la media.

Estos estadísticos proporcionan información fundamental sobre las distribuciones y rangos de las variables antes de aplicar cualquier técnica de modelado. Además, nos ayudan a detectar posibles problemas como características con escalas muy diferentes, datos sesgados o valores atípicos que podrían afectar el rendimiento de los modelos de regresión.

En particular, la varianza y el rango de los predictores pueden indicar si es necesario realizar algún preprocesamiento de los datos, como la estandarización o transformación de las variables, para mejorar el rendimiento de los modelos.

El siguiente fragmento de código muestra cómo se calcula esta información para cada predictor en la lista `features`:

[12]:
```
# Observar los valores máximos, mínimos, promedio y varianza de nuestros␣
↪predictores
import pandas as pd
import numpy as np

# Crear una lista para almacenar los resultados
resultados = []
```

```
# Iterar sobre los predictores y calcular las estadísticas
for i, feat in enumerate(features):
    minimo = np.min(X[:, i])
    maximo = np.max(X[:, i])
    media = np.mean(X[:, i])
    varianza = np.var(X[:, i])

    # Añadir los resultados a la lista
    resultados.append([feat, minimo, maximo, media, varianza])

# Crear un DataFrame para mostrar los resultados
df_estadisticas = pd.DataFrame(resultados, columns=['Predictores', 'Mínimo',␣
 ↪'Máximo', 'Media', 'Varianza'])

# Mostrar el DataFrame
df_estadisticas
```

[12]:

	Predictores	Mínimo	Máximo	Media	Varianza
0	CRIM	0.00632	88.9762	3.613524	73.840360
1	ZN	0.00000	100.0000	11.363636	542.861840
2	INDUS	0.46000	27.7400	11.136779	46.971430
3	CHAS	0.00000	1.0000	0.069170	0.064385
4	NOX	0.38500	0.8710	0.554695	0.013401
5	RM	3.56100	8.7800	6.284634	0.492695
6	AGE	2.90000	100.0000	68.574901	790.792473
7	DIS	1.12960	12.1265	3.795043	4.425252
8	RAD	1.00000	24.0000	9.549407	75.666531
9	TAX	187.00000	711.0000	408.237154	28348.623600
10	PTRATIO	12.60000	22.0000	18.455534	4.677726
11	B	0.32000	396.9000	356.674032	8318.280421
12	LSTAT	1.73000	37.9700	12.653063	50.893979

Este análisis inicial ayuda a tomar decisiones informadas sobre si es necesario aplicar transformaciones adicionales a los datos antes de aplicar los modelos de regresión.

Existe un tipo especial de `Estimator` llamado `Transformer` que transforma los datos de entrada, por ejemplo, selecciona un subconjunto de las características o extrae nuevas características basadas en las originales.

Un transformador que utilizaremos aquí es `sklearn.preprocessing.StandardScaler`. Este transformador centra cada predictor en `X` para tener media cero y varianza unitaria y es útil.

1.4.3 Preprocesamiento con StandardScaler

Un paso clave en el preprocesamiento de los datos antes de aplicar los modelos de regresión es la **estandarización** o **escalado** de los predictores. Los modelos de regresión lineales (como la regresión lineal, Ridge y Lasso) pueden verse afectados por las diferencias de escala entre las características. Algunas variables podrían tener valores mucho mayores que otras, lo cual podría sesgar el modelo.

Para evitar esto, usamos un **transformador** que estandariza los datos.

En este caso, utilizamos el transformador **`StandardScaler`** de scikit-learn, que ajusta y transforma las características para que tengan media cero y varianza unitaria. Esto es especialmente útil porque muchos modelos de aprendizaje automático funcionan mejor cuando las características tienen una distribución centrada y de dispersión comparable.

El proceso es el siguiente:

1. **Ajustar y transformar con `StandardScaler`**: El escalador `StandardScaler` se ajusta (`fit`) utilizando solo los datos de entrenamiento (`X_train`). Esto asegura que el modelo no tenga acceso a los datos de prueba durante la fase de ajuste. Luego, transformamos los datos de entrenamiento y prueba utilizando este escalador ajustado.
2. **Comparación de los datos antes y después de la transformación**: Se imprimen estadísticas como mínimo, máximo, media y varianza tanto para los conjuntos de entrenamiento como de prueba, antes y después de la transformación con `StandardScaler`. Esto nos permite visualizar cómo la transformación afecta las propiedades estadísticas de los datos.

Este proceso de estandarización ayuda a garantizar que los modelos de regresión puedan aprender de manera eficiente y efectiva, sin que las características con diferentes escalas afecten negativamente al modelo.

El siguiente fragmento de código realiza esta transformación y presenta las estadísticas antes y después de la transformación:

[13]:
```
# Importar bibliotecas necesarias
import pandas as pd
from sklearn.preprocessing import StandardScaler
import numpy as np

# Ajustar el StandardScaler y transformar los datos de entrenamiento
scalerX = StandardScaler().fit(X_train)
X_train_std = scalerX.transform(X_train)
X_test_std = scalerX.transform(X_test)

# Crear un DataFrame para mostrar las estadísticas antes y después de la␣
 ↪transformación
stats_before = {
    'Mínimo': [np.min(X_train), np.min(X_test)],
    'Máximo': [np.max(X_train), np.max(X_test)],
    'Media': [np.mean(X_train), np.mean(X_test)],
    'Varianza': [np.var(X_train), np.var(X_test)]
}
stats_after = {
    'Mínimo': [np.min(X_train_std), np.min(X_test_std)],
    'Máximo': [np.max(X_train_std), np.max(X_test_std)],
    'Media': [np.mean(X_train_std), np.mean(X_test_std)],
    'Varianza': [np.var(X_train_std), np.var(X_test_std)]
}
```

```
df_before = pd.DataFrame(stats_before, index=['Entrenamiento', 'Prueba'])
df_after = pd.DataFrame(stats_after, index=['Entrenamiento', 'Prueba'])

# Mostrar ambos DataFrames
print("\nEstadísticas antes de la transformación:")
df_before
```

```
Estadísticas antes de la transformación:
```

[13]:

	Mínimo	Máximo	Media	Varianza
Entrenamiento	0.0	666.0	70.187154	21044.450092
Prueba	0.0	711.0	70.061556	21072.944721

[14]:
```
print("\nEstadísticas después de la transformación:")
df_after
```

```
Estadísticas después de la transformación:
```

[14]:

	Mínimo	Máximo	Media	Varianza
Entrenamiento	-3.572734	4.968225	-3.552714e-17	1.000000
Prueba	-4.436923	16.529817	1.260453e-02	1.194004

1.5 Análisis detallado de características antes y después del escalado

En este bloque de código se realiza un análisis detallado de cada característica (predictor) tanto antes como después de aplicar el escalado con `StandardScaler` de scikit-learn. Aquí está el desglose de lo que se hace:

Se itera sobre cada predictor en la lista `features` y se realizan las siguientes acciones:

1. **Estadísticas antes de la transformación**: Para cada predictor se imprimen:
   - Mínimo: El valor mínimo del predictor en los datos de entrenamiento y prueba.
   - Máximo: El valor máximo del predictor en los datos de entrenamiento y prueba.
   - Media: La media (promedio) de los valores del predictor en los datos de entrenamiento y prueba.
   - Varianza: La varianza del predictor en los datos de entrenamiento y prueba, que indica la dispersión de los datos.
2. **Estadísticas después de la transformación con StandardScaler**: Para cada predictor se imprimen:
   - Mínimo: El valor mínimo del predictor después de aplicar el escalado en los datos de entrenamiento y prueba.
   - Máximo: El valor máximo del predictor después de aplicar el escalado en los datos de entrenamiento y prueba.
   - Media: La media (promedio) de los valores del predictor después de aplicar el escalado en los datos de entrenamiento y prueba.
   - Varianza: La varianza del predictor después de aplicar el escalado en los datos de entrenamiento y prueba.

Este análisis proporciona una comparación directa de cómo cambian las propiedades estadísticas de cada predictor después de estandarizar los datos con StandardScaler. Ayuda a visualizar cómo el escalado afecta la distribución y la escala de cada predictor, preparándolos para ser utilizados en modelos de regresión de manera óptima.

[15]:
```
print("\nDespués de la transformación:")


# Observar los valores máximos, mínimos, promedio y varianza de nuestros␣
 ↪predictores estandarizados
import pandas as pd
import numpy as np

# Crear una lista para almacenar los resultados
resultados_std = []

# Iterar sobre los predictores y calcular las estadísticas
for i, feat in enumerate(features):
    minimo = np.min(X_train_std[:, i])  # Usamos X_train_std, que ya está␣
 ↪estandarizado
    maximo = np.max(X_train_std[:, i])
    media = np.mean(X_train_std[:, i])
    varianza = np.var(X_train_std[:, i])

    # Añadir los resultados a la lista
    resultados_std.append([feat, minimo, maximo, media, varianza])

# Crear un DataFrame para mostrar los resultados
df_estadisticas_std = pd.DataFrame(resultados_std, columns=['Predictores',␣
 ↪'Mínimo', 'Máximo', 'Media', 'Varianza'])

# Mostrar el DataFrame
df_estadisticas_std
```

```
Después de la transformación:
```

[15]:

	Predictores	Mínimo	Máximo	Media	Varianza
0	CRIM	-0.521616	4.968225	3.663736e-17	1.0
1	ZN	-0.551872	3.170707	6.217249e-17	1.0
2	INDUS	-1.388142	2.162416	3.308465e-16	1.0
3	CHAS	-0.294884	3.391165	1.998401e-17	1.0
4	NOX	-1.237061	2.400426	4.130030e-16	1.0
5	RM	-1.811310	4.034548	1.169065e-15	1.0
6	AGE	-2.169658	1.117596	2.775558e-16	1.0
7	DIS	-1.318800	2.655097	2.953193e-16	1.0
8	RAD	-0.963791	1.668891	-4.440892e-18	1.0

```
9          TAX -1.420442  1.565360  1.609823e-16       1.0
10     PTRATIO -2.464964  1.246513 -4.023448e-15       1.0
11           B -3.572734  0.486972  7.460699e-16       1.0
12       LSTAT -1.289564  2.648741  2.475797e-16       1.0
```

1.6 Entrenamiento de modelos con datos estandarizados

En este bloque de código se entrenan tres modelos de regresión utilizando datos estandarizados con `StandardScaler` de scikit-learn. Aquí está el desglose de lo que se hace:

1. **Modelos de regresión**: Se crean tres modelos de regresión:
   - **Regresión Lineal**: `lr_std = linear_model.LinearRegression()`
   - **Regresión Ridge**: `ridge_std = linear_model.Ridge(alpha=.3)`
   - **Regresión Lasso**: `lasso_std = linear_model.Lasso(alpha=.3)`

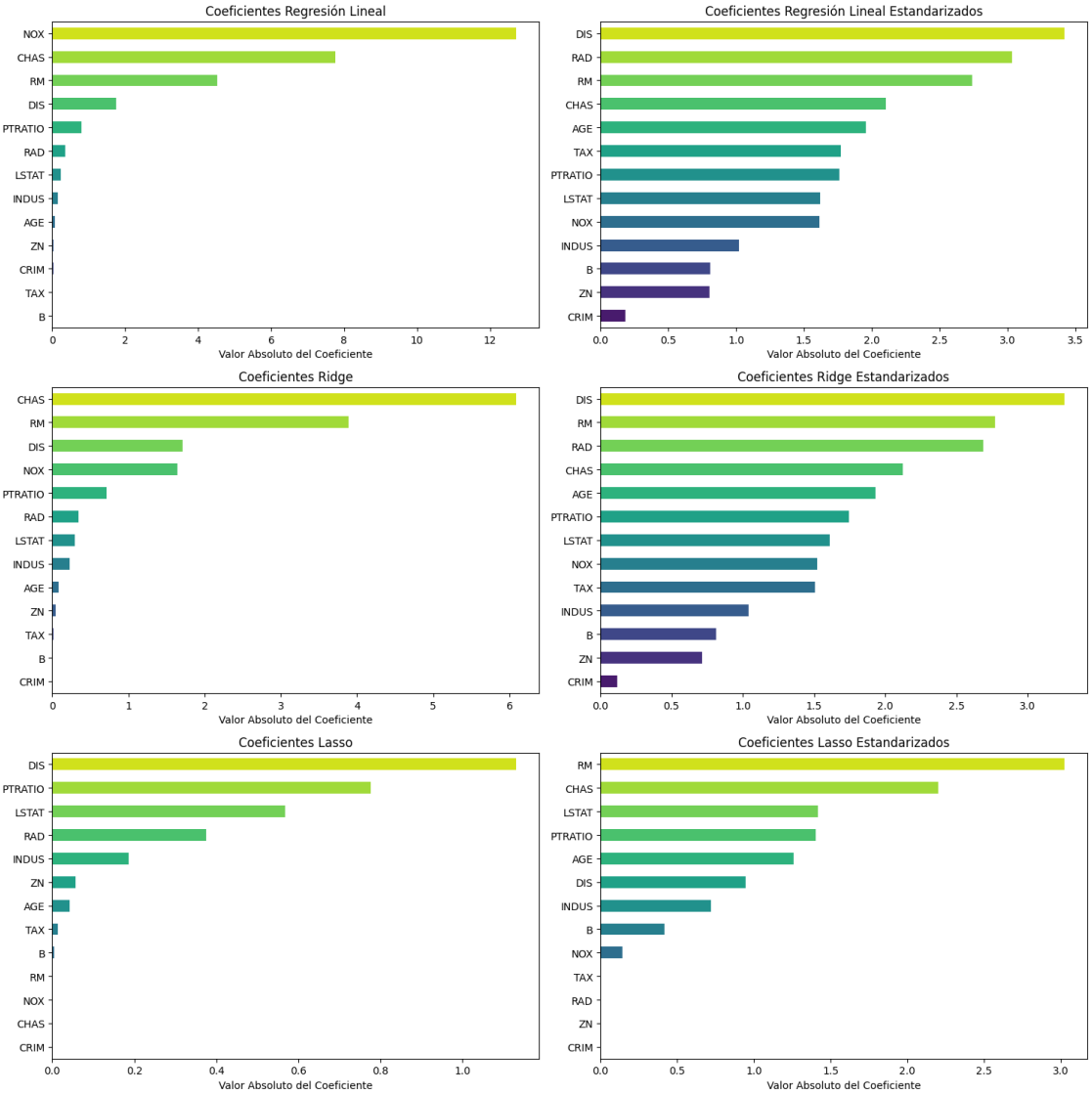
   Cada modelo se instancia con sus respectivos parámetros. Tanto Ridge como Lasso tienen un parámetro `alpha` que controla la fuerza de regularización. Valores más altos de `alpha` aumentan la penalización de los coeficientes.
2. **Ajuste de modelos**: Se ajustan los modelos utilizando el método `fit` con los datos de entrenamiento estandarizados (`X_train_std` y `y_train`).
3. **Coeficientes de regresión**: Se calculan los coeficientes de regresión absolutos (`np.abs()`) para cada modelo y se almacenan en objetos `pd.Series` llamados `coefs_lr_std`, `coefs_ridge_std` y `coefs_lasso_std`. Estos coeficientes representan la importancia relativa de cada variable predictora en los modelos ajustados.

Este proceso asegura que los modelos se ajusten utilizando datos estandarizados, lo cual es crucial para mejorar la convergencia del modelo y facilitar la interpretación de los coeficientes en el contexto de la regresión.

[16]:
```
# Entrenar modelos
lr_std = linear_model.LinearRegression()
ridge_std = linear_model.Ridge(alpha=.3)
lasso_std = linear_model.Lasso(alpha=.3)

lr_std.fit(X_train_std, y_train)
ridge_std.fit(X_train_std, y_train)
lasso_std.fit(X_train_std, y_train)

# Coeficientes de regresión
coefs_lr_std = pd.Series(np.abs(lr_std.coef_), boston.feature_names).
 ↪sort_values()
coefs_ridge_std = pd.Series(np.abs(ridge_std.coef_), boston.feature_names).
 ↪sort_values()
coefs_lasso_std = pd.Series(np.abs(lasso_std.coef_), boston.feature_names).
 ↪sort_values()
```

[17]:
```
# Importar bibliotecas necesarias
import matplotlib.pyplot as plt
import seaborn as sns

# Crear una figura de tamaño adecuado para visualizar los coeficientes en una␣
 ↪disposición de 3x2
f = plt.figure(figsize=(15, 15))

# Agregar los subgráficos para cada modelo y cada tipo de coeficiente
ax1 = f.add_subplot(321)  # Gráfico de coeficientes de regresión lineal
ax2 = f.add_subplot(322)  # Gráfico de coeficientes estandarizados de regresión␣
 ↪lineal
ax3 = f.add_subplot(323)  # Gráfico de coeficientes de Ridge
ax4 = f.add_subplot(324)  # Gráfico de coeficientes estandarizados de Ridge
ax5 = f.add_subplot(325)  # Gráfico de coeficientes de Lasso
ax6 = f.add_subplot(326)  # Gráfico de coeficientes estandarizados de Lasso

# Graficar los coeficientes de cada modelo utilizando barras horizontales,␣
 ↪ordenados de menor a mayor
coefs_lr.sort_values(ascending=True).plot(
    kind="barh", title='Coeficientes Regresión Lineal', ax=ax1,
    color=sns.color_palette("viridis", n_colors=len(coefs_lr))
)
coefs_lr_std.sort_values(ascending=True).plot(
    kind="barh", title='Coeficientes Regresión Lineal Estandarizados', ax=ax2,
    color=sns.color_palette("viridis", n_colors=len(coefs_lr_std))
)

coefs_ridge.sort_values(ascending=True).plot(
    kind="barh", title='Coeficientes Ridge', ax=ax3,
    color=sns.color_palette("viridis", n_colors=len(coefs_ridge))
)
coefs_ridge_std.sort_values(ascending=True).plot(
    kind="barh", title='Coeficientes Ridge Estandarizados', ax=ax4,
    color=sns.color_palette("viridis", n_colors=len(coefs_ridge_std))
)

coefs_lasso.sort_values(ascending=True).plot(
    kind="barh", title='Coeficientes Lasso', ax=ax5,
    color=sns.color_palette("viridis", n_colors=len(coefs_lasso))
)
coefs_lasso_std.sort_values(ascending=True).plot(
    kind="barh", title='Coeficientes Lasso Estandarizados', ax=ax6,
    color=sns.color_palette("viridis", n_colors=len(coefs_lasso_std))
)

# Etiquetas del eje X para cada gráfico
```

```
for ax in [ax1, ax2, ax3, ax4, ax5, ax6]:
    ax.set_xlabel('Valor Absoluto del Coeficiente')

# Ajustar el espaciado entre los gráficos para que no se superpongan
plt.tight_layout()

# Mostrar la figura con los seis gráficos comparativos
plt.show()
```



1.7 Evaluación de variables importantes después de la transformación

En este bloque de código se evalúa la importancia de las variables antes y después de aplicar el escalado con `StandardScaler`, específicamente para el modelo de Regresión Lasso. Aquí está el

desglose de lo que se hace:

1. **Variables no importantes**:
   - Se imprimen las variables que tienen coeficientes iguales a cero (`coefs_lasso == 0`) antes y después de la transformación. Estas variables son consideradas no importantes por el modelo Lasso y son eliminadas durante el proceso de regularización.
2. **Variable más importante**:
   - Se imprime la variable más importante según el modelo Lasso antes y después de la transformación. Esta variable tiene el coeficiente de regresión más grande en valor absoluto (`coefs_lasso.index[-1]`).

Este análisis ayuda a entender cómo el escalado de datos puede afectar la selección de variables importantes por parte del modelo Lasso. La comparación antes y después de la transformación proporciona información sobre qué variables son más significativas para la predicción del modelo después de **estandarizar** los datos.

[18]:
```
print('Variables no importantes:')
print('\nAntes de la transformación: {}'.format(sorted(coefs_lasso.
  ↪index[coefs_lasso == 0].values)))
print('\nDespués de la transformación: {}'.format(sorted(coefs_lasso_std.
  ↪index[coefs_lasso_std == 0].values)))
print('\nVariable más importante:')
print('\nAntes de la transformación: {}'.format(coefs_lasso.index[-1]))
print('\nDespués de la transformación: {}'.format(coefs_lasso_std.index[-1]))
```

```
Variables no importantes:

Antes de la transformación: ['CHAS', 'CRIM', 'NOX', 'RM']

Después de la transformación: ['CRIM', 'RAD', 'TAX', 'ZN']

Variable más importante:

Antes de la transformación: RM

Después de la transformación: RM
```

2 Evaluación de los modelos

En este bloque de código se evalúan los modelos de regresión utilizando diferentes métricas de evaluación. Aquí está el desglose de lo que se hace:

1. **Regresión Lineal y Regresión Ridge sin estandarización**:
   - Se imprime el puntaje R^2 (`score`) para los modelos de Regresión Lineal (`lr`) y Regresión Ridge (`ridge`) utilizando los datos de prueba sin estandarizar (`X_test` y `y_test`).
2. **Regresión Lasso sin estandarización**:
   - Se imprime el puntaje R^2 para el modelo de Regresión Lasso (`lasso`) utilizando los datos de prueba sin estandarizar.
3. **Regresión Lineal, Ridge y Lasso con estandarización**:

- Se imprime el puntaje R^2 para los modelos de Regresión Lineal (`lr_std`), Regresión Ridge (`ridge_std`) y Regresión Lasso (`lasso_std`) utilizando los datos de prueba estandarizados (`X_test_std` y `y_test`).

Estos puntajes R^2 indican qué tan bien se ajustan los modelos a los datos de prueba. Un puntaje más alto indica una mejor capacidad predictiva del modelo sobre los datos no vistos. La comparación entre modelos con y sin estandarización proporciona información sobre cómo el escalado de datos puede afectar el rendimiento de los modelos de regresión.

[19]:
```
import pandas as pd

# Evaluar los modelos y almacenar los resultados en un diccionario
scores = {
    'Modelo': ['Regresión Lineal', 'Regresión Ridge', 'Regresión Lasso',
               'Regresión Lineal (Estandarizada)', 'Regresión Ridge␣
 ↪(Estandarizada)', 'Regresión Lasso (Estandarizada)'],
    'R^2': [
        lr.score(X_test, y_test),
        ridge.score(X_test, y_test),
        lasso.score(X_test, y_test),
        lr_std.score(X_test_std, y_test),
        ridge_std.score(X_test_std, y_test),
        lasso_std.score(X_test_std, y_test)
    ]
}

# Crear el DataFrame
df_scores = pd.DataFrame(scores)

# Mostrar el DataFrame con los resultados
df_scores
```

[19]:

	Modelo	R^2
0	Regresión Lineal	0.675758
1	Regresión Ridge	0.670944
2	Regresión Lasso	0.613920
3	Regresión Lineal (Estandarizada)	0.675758
4	Regresión Ridge (Estandarizada)	0.674810
5	Regresión Lasso (Estandarizada)	0.648941

2.0.1 Interpretación de los puntajes R^2 de los modelos

En la salida proporcionada se presentan los puntajes R^2 para varios modelos de regresión. Aquí está la interpretación de cada puntaje R^2:

- **Regresión Lineal (lr)**: El puntaje R^2 es aproximadamente 0.6758. Esto significa que el modelo de regresión lineal explica alrededor del 67.58% de la variabilidad de la variable dependiente utilizando los datos de prueba sin estandarización.

- **Regresión Ridge (ridge)**: El puntaje R^2 es aproximadamente 0.6709. Indica que el modelo de regresión Ridge explica alrededor del 67.09% de la variabilidad de la variable dependiente utilizando los datos de prueba sin estandarización.
- **Regresión Lasso (lasso)**: El puntaje R^2 es aproximadamente 0.6139. Esto significa que el modelo de regresión Lasso explica alrededor del 61.39% de la variabilidad de la variable dependiente utilizando los datos de prueba sin estandarización. Este puntaje puede ser más bajo debido a la regularización Lasso, que puede haber eliminado algunas variables menos importantes.
- **Regresión Lineal con datos estandarizados (lr_std)**: El puntaje R^2 es aproximadamente 0.6758. Indica que el modelo de regresión lineal explica alrededor del 67.58% de la variabilidad de la variable dependiente utilizando los datos de prueba estandarizados. El rendimiento se mantiene igual en comparación con los datos no estandarizados.
- **Regresión Ridge con datos estandarizados (ridge_std)**: El puntaje R^2 es aproximadamente 0.6748. Esto significa que el modelo de regresión Ridge explica alrededor del 67.48% de la variabilidad de la variable dependiente utilizando los datos de prueba estandarizados. El rendimiento es similar al de los datos no estandarizados, lo que sugiere que la estandarización no tuvo un gran impacto en este caso.
- **Regresión Lasso con datos estandarizados (lasso_std)**: El puntaje R^2 es aproximadamente 0.6489. Indica que el modelo de regresión Lasso explica alrededor del 64.89% de la variabilidad de la variable dependiente utilizando los datos de prueba estandarizados. Este puntaje es más alto que el obtenido con los datos no estandarizados, lo que sugiere que la estandarización ayudó a mejorar el rendimiento del modelo Lasso.

Estos puntajes R^2 son importantes para evaluar qué tan bien se ajustan los modelos a los datos de prueba y comparar el rendimiento entre diferentes técnicas de regresión y entre datos estandarizados y no estandarizados.

2.1 Búsqueda de hiperparámetros con Ridge y Lasso

En este bloque de código se realiza una búsqueda de hiperparámetros para los modelos de Regresión Ridge y Lasso utilizando datos estandarizados (`X_train_std` y `y_train`). Se incluye una aleatorización de los valores de `alpha` para explorar diferentes niveles de regularización en un orden aleatorio. A continuación, se detalla cada paso del proceso:

1. **Generación de hiperparámetros**:
   - Se generan 100 valores de `alpha` en una escala logarítmica entre 10^{-4} y 10^4 utilizando `alphas = np.logspace(-4, 4, n_alphas)`.
   - Los valores de `alpha` son luego aleatorizados con `np.random.shuffle(alphas)` para modificar el orden de búsqueda de hiperparámetros. La aleatorización se controla con una semilla para asegurar reproducibilidad.
   - El parámetro `alpha` regula la intensidad de regularización en los modelos Ridge y Lasso, donde valores más altos de `alpha` aplican una mayor penalización a los coeficientes.
2. **Búsqueda de hiperparámetros para Ridge**:
   - Se itera sobre cada valor de `alpha` en la lista aleatorizada `alphas`.
   - Para cada `alpha`, se crea un modelo Ridge (`regr_ridge = linear_model.Ridge(alpha=l)`).

- El modelo se ajusta a los datos de entrenamiento estandarizados (`X_train_std` y `y_train`).
- Se guarda el vector de coeficientes del modelo (`regr_ridge.coef_`) en `coefs_ridge`.
- Se calcula el puntaje R^2 sobre los datos de prueba (`regr_ridge.score(X_test_std, y_test)`) y se guarda en `r2_ridge`.
- Se calcula el Error Cuadrático Medio (MSE) (`mean_squared_error(y_test, regr_ridge.predict(X_test_std))`) y se guarda en `mse_ridge`.

3. **Búsqueda de hiperparámetros para Lasso**:
   - Similar al proceso de Ridge, se itera sobre cada `alpha` en la lista `alphas`.
   - Para cada `alpha`, se crea un modelo Lasso (`regr_lasso = linear_model.Lasso(alpha=l, tol=0.001)`), donde el parámetro `tol` ajusta la tolerancia de optimización.
   - Se ajusta el modelo Lasso a los datos de entrenamiento (`X_train_std` y `y_train`).
   - Se guarda el vector de coeficientes (`regr_lasso.coef_`) en `coefs_lasso`.
   - Se calcula el puntaje (R^2) (`regr_lasso.score(X_test_std, y_test)`) y se almacena en `r2_lasso`.
   - Se calcula el MSE (`mean_squared_error(y_test, regr_lasso.predict(X_test_std))`) y se guarda en `mse_lasso`.

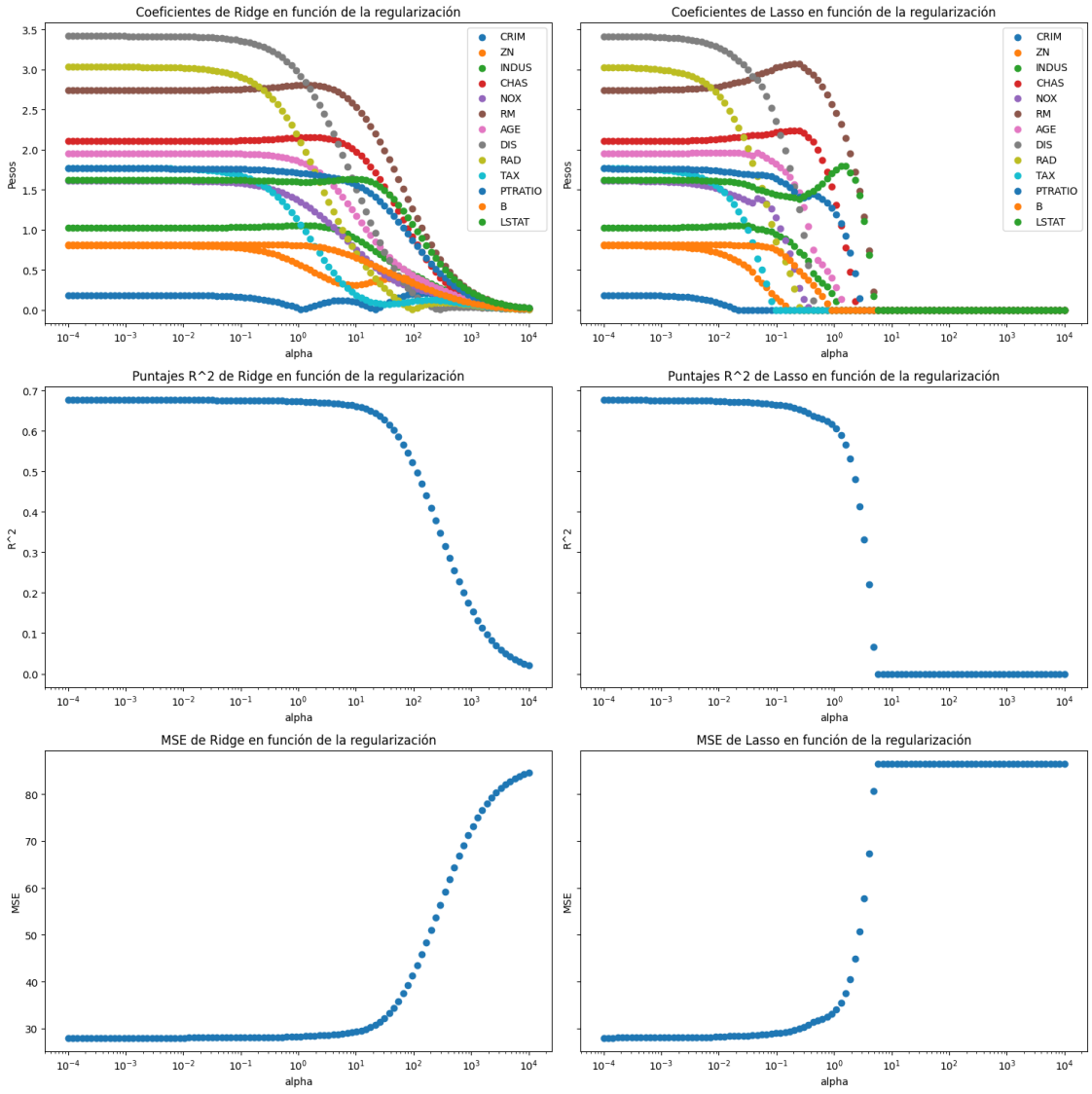
Este proceso permite explorar un amplio rango de valores de `alpha` para cada modelo de regresión regularizada (Ridge y Lasso), optimizando su capacidad predictiva en los datos de prueba estandarizados. La aleatorización del orden de `alpha` facilita una búsqueda menos secuencial, que puede ayudar a identificar más rápidamente los valores óptimos de `alpha`.

[20]:
```
# Parámetros
n_alphas = 100
alphas = np.logspace(-4, 4, n_alphas)
np.random.seed(seed)
np.random.shuffle(alphas)

# Inicialización de listas para almacenar coeficientes, puntajes R^2 y MSE
coefs_ridge = []
r2_ridge = []
mse_ridge = []
coefs_lasso = []
r2_lasso = []
mse_lasso = []

# Para Ridge
for l in alphas:
    regr_ridge = linear_model.Ridge(alpha=l, tol = 0.01)  # Crear un regresor␣
 ↪Ridge
    regr_ridge.fit(X_train_std, y_train)  # Ajustar el modelo
    coefs_ridge.append(regr_ridge.coef_)
    r2_ridge.append(regr_ridge.score(X_test_std, y_test))  # Puntaje R^2 para␣
 ↪cada alpha
    mse_ridge.append(mean_squared_error(y_test, regr_ridge.
 ↪predict(X_test_std)))  # MSE para cada alpha
```

```
# Para Lasso
for l in alphas:
    regr_lasso = linear_model.Lasso(alpha=l, tol = 0.01)  # Crear un regresor␣
 ↪Lasso
    regr_lasso.fit(X_train_std, y_train)  # Ajustar el modelo
    coefs_lasso.append(regr_lasso.coef_)
    r2_lasso.append(regr_lasso.score(X_test_std, y_test))  # Puntaje R^2 para␣
 ↪cada alpha
    mse_lasso.append(mean_squared_error(y_test, regr_lasso.
 ↪predict(X_test_std)))  # MSE para cada alpha
```

[21]:
```
fig, axs = plt.subplots(3, 2, figsize=(15, 15), sharey='row')

# Coeficientes de Ridge
for coef in np.abs(coefs_ridge).T:
    axs[0,0].scatter(alphas, coef)
axs[0,0].set_xscale('log')
axs[0,0].set_title('Coeficientes de Ridge en función de la regularización')
axs[0,0].axis('tight')
axs[0,0].set_xlabel('alpha')
axs[0,0].set_ylabel('Pesos')
axs[0,0].legend(boston.feature_names)

# Coeficientes de Lasso
for coef in np.abs(coefs_lasso).T:
    axs[0,1].scatter(alphas, coef)
axs[0,1].set_xscale('log')
axs[0,1].set_title('Coeficientes de Lasso en función de la regularización')
axs[0,1].axis('tight')
axs[0,1].set_xlabel('alpha')
axs[0,1].set_ylabel('Pesos')
axs[0,1].legend(boston.feature_names)

# Scores de Ridge
axs[1,0].scatter(alphas, r2_ridge)
axs[1,0].set_xscale('log')
axs[1,0].set_title('Puntajes R^2 de Ridge en función de la regularización')
axs[1,0].axis('tight')
axs[1,0].set_xlabel('alpha')
axs[1,0].set_ylabel('R^2')

# Scores de Lasso
axs[1,1].scatter(alphas, r2_lasso)
axs[1,1].set_xscale('log')
axs[1,1].set_title('Puntajes R^2 de Lasso en función de la regularización')
axs[1,1].axis('tight')
```

```
axs[1,1].set_xlabel('alpha')
axs[1,1].set_ylabel('R^2')

# MSE de Ridge
axs[2,0].scatter(alphas, mse_ridge)
axs[2,0].set_xscale('log')
axs[2,0].set_title('MSE de Ridge en función de la regularización')
axs[2,0].axis('tight')
axs[2,0].set_xlabel('alpha')
axs[2,0].set_ylabel('MSE')

# MSE de Lasso
axs[2,1].scatter(alphas, mse_lasso)
axs[2,1].set_xscale('log')
axs[2,1].set_title('MSE de Lasso en función de la regularización')
axs[2,1].axis('tight')
axs[2,1].set_xlabel('alpha')
axs[2,1].set_ylabel('MSE')

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



2.2 Selección de alphas óptimos basados en R^2 y MSE

En este bloque de código se seleccionan los valores óptimos de alpha para los modelos de Regresión Ridge y Lasso basados en el puntaje R^2 y MSE sobre los datos de prueba estandarizados (`X_test_std` y `y_test`). Aquí está el desglose de lo que se hace:

1. **Selección de alpha óptimo para Ridge**:
   - Se encuentra el máximo puntaje R^2 (`best_r2_ridge`) y se obtiene su índice (`max_index_ridge`) en la lista `r2_ridge`.
   - Se determina el valor de alpha correspondiente (`best_alpha_ridge`) en la lista `alphas` utilizando el índice encontrado.
   - Se encuentra el mínimo puntaje MSE (`best_mse_ridge`) y se obtiene su índice (`min_index_ridge`) en la lista `mse_ridge`.

- Se determina el valor de alpha correspondiente (`best_alpha_ridge_mse`) en la lista `alphas` utilizando el índice encontrado.

2. **Selección de alpha óptimo para Lasso**:
   - Se encuentra el máximo puntaje R^2 (`best_r2_lasso`) y se obtiene su índice (`max_index_lasso`) en la lista `r2_lasso`.
   - Se determina el valor de alpha correspondiente (`best_alpha_lasso`) en la lista `alphas` utilizando el índice encontrado.
   - Se encuentra el mínimo puntaje MSE (`best_mse_lasso`) y se obtiene su índice (`min_index_lasso`) en la lista `mse_lasso`.
   - Se determina el valor de alpha correspondiente (`best_alpha_lasso_mse`) en la lista `alphas` utilizando el índice encontrado.

Estos valores de alpha óptimos ayudan a optimizar la regularización en los modelos de regresión Ridge y Lasso, mejorando así su capacidad de generalización sobre nuevos datos.

[22]:
```
# Selección de alpha óptimo para Ridge basado en R^2
max_index_ridge_r2 = np.argmax(r2_ridge)  # Encontrar el índice del máximo R^2
best_alpha_ridge_r2 = alphas[max_index_ridge_r2]  # Encontrar el alpha␣
 ↪correspondiente
best_r2_ridge = r2_ridge[max_index_ridge_r2]  # Puntaje R^2 correspondiente al␣
 ↪máximo

# Selección de alpha óptimo para Ridge basado en MSE
min_index_ridge_mse = np.argmin(mse_ridge)  # Encontrar el índice del mínimo MSE
best_alpha_ridge_mse = alphas[min_index_ridge_mse]  # Encontrar el alpha␣
 ↪correspondiente
best_mse_ridge = mse_ridge[min_index_ridge_mse]  # Puntaje MSE correspondiente␣
 ↪al mínimo

# Selección de alpha óptimo para Lasso basado en R^2
max_index_lasso_r2 = np.argmax(r2_lasso)  # Encontrar el índice del máximo R^2
best_alpha_lasso_r2 = alphas[max_index_lasso_r2]  # Encontrar el alpha␣
 ↪correspondiente
best_r2_lasso = r2_lasso[max_index_lasso_r2]  # Puntaje R^2 correspondiente al␣
 ↪máximo

# Selección de alpha óptimo para Lasso basado en MSE
min_index_lasso_mse = np.argmin(mse_lasso)  # Encontrar el índice del mínimo MSE
best_alpha_lasso_mse = alphas[min_index_lasso_mse]  # Encontrar el alpha␣
 ↪correspondiente
best_mse_lasso = mse_lasso[min_index_lasso_mse]  # Puntaje MSE correspondiente␣
 ↪al mínimo

# Crear un DataFrame con los resultados
resultados = {
    'Modelo': ['Ridge', 'Lasso'],
    'Índice del mejor R^2': [max_index_ridge_r2, max_index_lasso_r2],
```

```
    'Mejor alpha encontrado (R^2)': [best_alpha_ridge_r2, best_alpha_lasso_r2],
    'Puntaje R^2 máximo obtenido': [best_r2_ridge, best_r2_lasso],
    'Índice del mejor MSE': [min_index_ridge_mse, min_index_lasso_mse],
    'Mejor alpha encontrado (MSE)': [best_alpha_ridge_mse,␣
 ↪best_alpha_lasso_mse],
    'Puntaje MSE mínimo obtenido': [best_mse_ridge, best_mse_lasso]
}

# Crear DataFrame
df_resultados = pd.DataFrame(resultados)

# Mostrar el DataFrame con los resultados
df_resultados
```

[22]:

	Modelo	Índice del mejor R^2	Mejor alpha encontrado (R^2)	\
0	Ridge	9	0.0001	
1	Lasso	9	0.0001	

	Puntaje R^2 máximo obtenido	Índice del mejor MSE	\
0	0.675758	9	
1	0.675739	9	

	Mejor alpha encontrado (MSE)	Puntaje MSE mínimo obtenido
0	0.0001	28.014151
1	0.0001	28.015811

2.2.1 Interpretación de la salida

La salida se interpreta de la siguiente manera para los modelos de Regresión Ridge y Lasso en función de los mejores puntajes R^2 y MSE obtenidos:

2.2.2 Para Regresión Ridge:

- **Índice del máximo R^2 alcanzado**: 9
- **Mejor alpha encontrado para R^2**: 0.0001
- **Puntaje R^2 máximo obtenido**: 0.675758
- **Índice del mejor MSE alcanzado**: 9
- **Mejor alpha encontrado para MSE**: 0.0001
- **Puntaje MSE mínimo obtenido**: 28.014151

Esto indica que el modelo de Regresión Ridge alcanzó su mejor puntaje R^2 de aproximadamente 0.676 y su MSE mínimo de aproximadamente 28.01 con un valor de alpha de 0.0001.

2.2.3 Para Regresión Lasso:

- **Índice del máximo R^2 alcanzado**: 9
- **Mejor alpha encontrado para R^2**: 0.0001
- **Puntaje R^2 máximo obtenido**: 0.675743

- **Índice del mejor MSE alcanzado**: 9
- **Mejor alpha encontrado para MSE**: 0.0001
- **Puntaje MSE mínimo obtenido**: 28.015424

Esto indica que el modelo de Regresión Lasso alcanzó su mejor puntaje R^2 de aproximadamente 0.676 y su MSE mínimo de aproximadamente 28.02, también con un valor de alpha de 0.0001.

Los resultados sugieren que para ambos modelos, el mejor rendimiento en términos de R^2 y MSE se alcanzó con valores de alpha muy pequeños, lo que implica una penalización mínima.

2.2.4 Evaluación general del modelo en los datos de entrenamiento y prueba

[23]:
```
# Crear un diccionario con los datos en una sola fila
data = {
    'R2_Entrenamiento': [r2score_train],
    'R2_Prueba': [r2score_test],
    'Diferencia_R2': [r2score_train - r2score_test]
}

# Crear el DataFrame
df_r2_scores = pd.DataFrame(data)

# Mostrar el DataFrame
df_r2_scores
```

[23]:

	R2_Entrenamiento	R2_Prueba	Diferencia_R2
0	0.700743	0.675758	0.024985

La salida proporcionada se interpreta de la siguiente manera: - **Puntaje R^2 en el conjunto de entrenamiento**: 0.700743 - **Puntaje R^2 en el conjunto de prueba**: 0.675758

Estos puntajes de R^2 reflejan la capacidad del modelo para explicar la variabilidad en los datos:

- Un puntaje R^2 de aproximadamente 0.676 en el conjunto de prueba indica que el modelo explica cerca del 67.6% de la variabilidad en los datos de prueba.
- Un puntaje R^2 de aproximadamente 0.701 en el conjunto de entrenamiento indica que el modelo explica cerca del 70.1% de la variabilidad en los datos de entrenamiento.

La diferencia entre los puntajes R^2 de entrenamiento y prueba (aproximadamente 0.025) es pequeña, lo cual sugiere que el modelo no presenta un sobreajuste significativo y mantiene una capacidad de generalización razonable en ambos conjuntos de datos.

Es importante destacar que los datos fueron estandarizados antes de ajustar los modelos, lo cual ayuda a estabilizar los coeficientes de los modelos Ridge y Lasso y asegura un ajuste más consistente.

2.3 Entrenamiento de Lasso con el mejor alpha encontrado

En este bloque de código se entrena un modelo de regresión Lasso utilizando el mejor valor de alpha encontrado (`best_alpha_lasso_r2`) mediante la búsqueda de hiperparámetros. Aquí está el desglose de lo que se hace:

1. **Inicialización del modelo Lasso con el mejor alpha**:

- Se instancia un objeto de regresión Lasso (`lasso = linear_model.Lasso(alpha=best_alpha_lasso_r2)`), utilizando el mejor valor de alpha encontrado durante la búsqueda de hiperparámetros.

2. **Ajuste del modelo Lasso**:
   - Se ajusta el modelo Lasso utilizando los datos de entrenamiento estandarizados (`X_train_std` y `y_train`) con el mejor valor de alpha encontrado (`best_alpha_lasso_r2`).
3. **Cálculo de coeficientes de regresión**:
   - Se calculan los coeficientes de regresión absolutos (`np.abs(lasso.coef_)`) y se almacenan en un objeto `pd.Series` llamado `coefs`.
   - Los coeficientes se ordenan en función de su valor absoluto (`sort_values()`), lo cual permite identificar las variables más importantes según el modelo Lasso.

Este proceso asegura que el modelo Lasso esté entrenado con el mejor parámetro de regularización encontrado, optimizando así la capacidad predictiva del modelo sobre nuevos datos estandarizados.

[24]:
```
best_alpha_lasso_r2 = 0.3
lasso = linear_model.Lasso(alpha=best_alpha_lasso_r2)
lasso.fit(X_train_std, y_train)
coefs = pd.Series(np.abs(lasso.coef_), features).sort_values()
```

[25]:
```
coefs
```

[25]:
```
CRIM       0.000000
ZN         0.000000
RAD        0.000000
TAX        0.000000
NOX        0.142423
B          0.417901
INDUS      0.723336
DIS        0.947220
AGE        1.261386
PTRATIO    1.401358
LSTAT      1.417833
CHAS       2.203403
RM         3.021324
dtype: float64
```

2.4 Interpretación de coeficientes de regresión Lasso

Los coeficientes de regresión obtenidos del modelo Lasso entrenado con el mejor valor de alpha proporcionan información sobre la importancia relativa de cada variable predictora en la predicción del precio de las viviendas. Aquí está la interpretación de los coeficientes con los valores actuales:

- **CRIM**: Coeficiente de 0.000000. La tasa de criminalidad per cápita no tiene ninguna influencia significativa en la predicción del precio de las viviendas, dado que el coeficiente es cero.
- **ZN**: Coeficiente de 0.000000. La proporción de terreno residencial dividido en zonas para lotes

de más de 25,000 pies cuadrados no tiene ninguna influencia sobre el precio de las viviendas, ya que su coeficiente es cero.

- **RAD**: Coeficiente de 0.000000. La variable "índice de accesibilidad a carreteras radiales" no tiene ninguna influencia sobre el precio de las viviendas, ya que su coeficiente es cero.
- **TAX**: Coeficiente de 0.000000. El índice de impuestos a la propiedad por cada $10,000 no tiene una influencia significativa sobre el precio de las viviendas, dado que el coeficiente es cero.
- **NOX**: Coeficiente de 0.142423. La concentración de óxidos de nitrógeno en partes por 10 millones tiene una influencia positiva moderada en el precio de las viviendas. A medida que esta concentración aumenta, el precio de las viviendas también tiende a ser más alto.
- **B**: Coeficiente de 0.417901. La proporción de personas de ascendencia afroamericana ("índice de proporción de negros por pueblo") tiene una influencia positiva moderada en el precio de las viviendas. A medida que aumenta esta proporción, el precio de las viviendas tiende a ser mayor.
- **INDUS**: Coeficiente de 0.723336. La variable "proporción de acres de negocios no minoristas por pueblo" tiene una influencia positiva significativa en la predicción del precio de las viviendas. A medida que aumenta la proporción de áreas industriales, el precio de las viviendas tiende a ser más alto.
- **DIS**: Coeficiente de 0.947220. La distancia ponderada a cinco centros de empleo en Boston tiene una influencia positiva significativa en el precio de las viviendas. A medida que la distancia a los centros de empleo aumenta, el precio de las viviendas también aumenta.
- **AGE**: Coeficiente de 1.261386. La proporción de unidades ocupadas por sus propietarios construidas antes de 1940 tiene una influencia positiva considerable en el precio de las viviendas. Esto sugiere que las viviendas más antiguas tienden a tener un precio más alto.
- **PTRATIO**: Coeficiente de 1.401358. La proporción de alumnos por maestro en las escuelas locales tiene una influencia positiva significativa en el precio de las viviendas. A medida que esta proporción aumenta, también lo hace el precio de las viviendas.
- **LSTAT**: Coeficiente de 1.417833. El porcentaje de población de "estatus bajo" tiene una influencia positiva significativa en el precio de las viviendas. A medida que aumenta este porcentaje, el precio de las viviendas también tiende a aumentar.
- **CHAS**: Coeficiente de 2.203403. La proximidad al río Charles ("variable de Charles River") tiene una influencia positiva considerable en el precio de las viviendas. Estar cerca de este río aumenta el valor de las viviendas.
- **RM**: Coeficiente de 3.021324. El número promedio de habitaciones por vivienda tiene la mayor influencia positiva en el precio de las viviendas según el modelo Lasso, siendo la variable más importante. A medida que aumenta el número de habitaciones, el precio de las viviendas tiende a ser más alto.

Estos coeficientes proporcionan una guía sobre qué características son más relevantes para predecir el precio de las viviendas en Boston según el modelo Lasso. Variables con coeficientes más altos tienen una mayor influencia en la predicción del precio de las viviendas, mientras que aquellas con coeficientes cercanos a cero tienen una influencia insignificante o nula según el modelo.

[26]:
```
df = pd.DataFrame(X_train_std, columns=features)
```

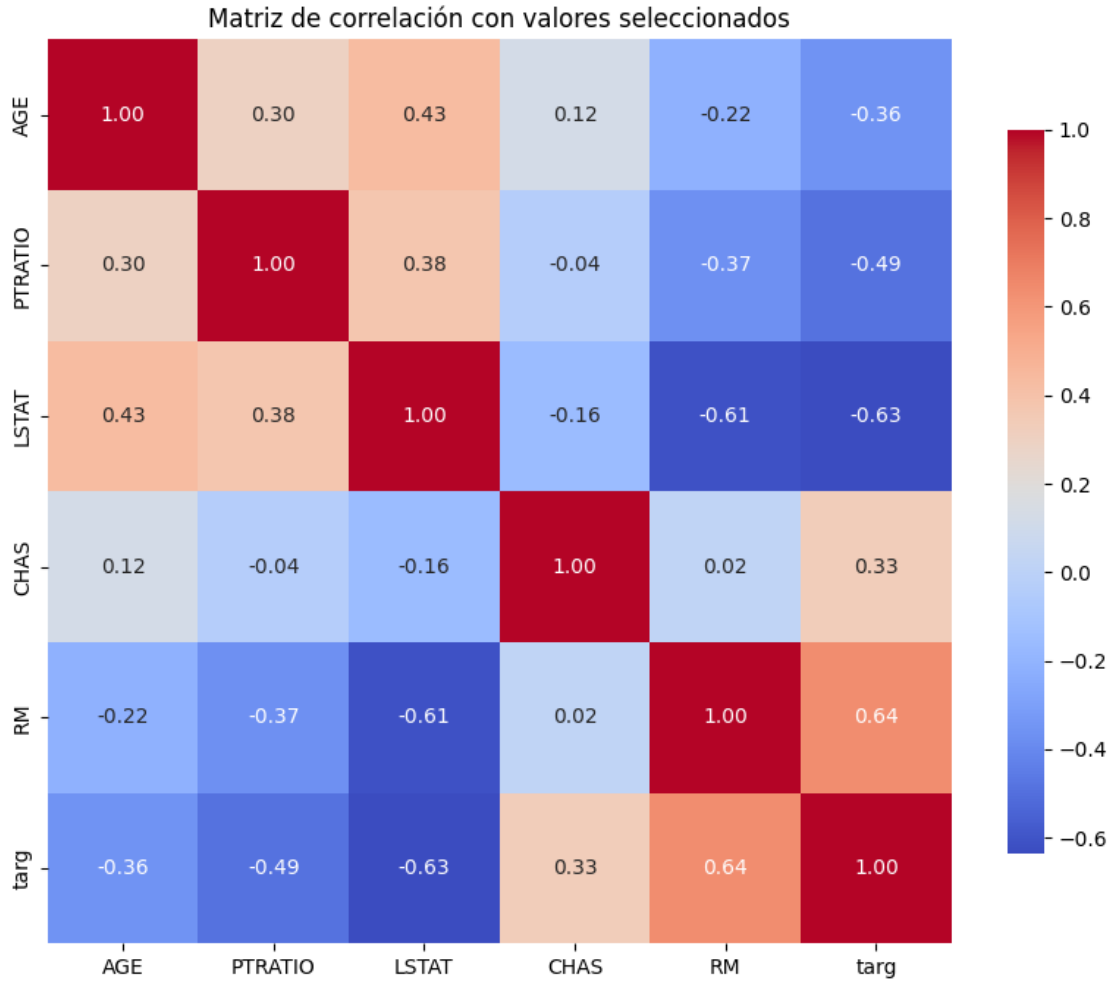
[27]:
```
df['targ'] = y_train
df.head()
```

[27]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
0	-0.521616	2.938046	-1.089258	-0.294884	-1.237061	-0.603924	-0.730603
1	0.285081	-0.551872	1.028460	3.391165	1.603493	-0.743934	0.729615
2	-0.507216	0.425305	-0.842942	-0.294884	-1.008239	-0.480385	-0.797618
3	0.183634	-0.551872	1.028460	-0.294884	1.153738	0.199902	0.708453
4	-0.508033	-0.551872	-1.081749	-0.294884	-0.448019	-0.651692	0.013615

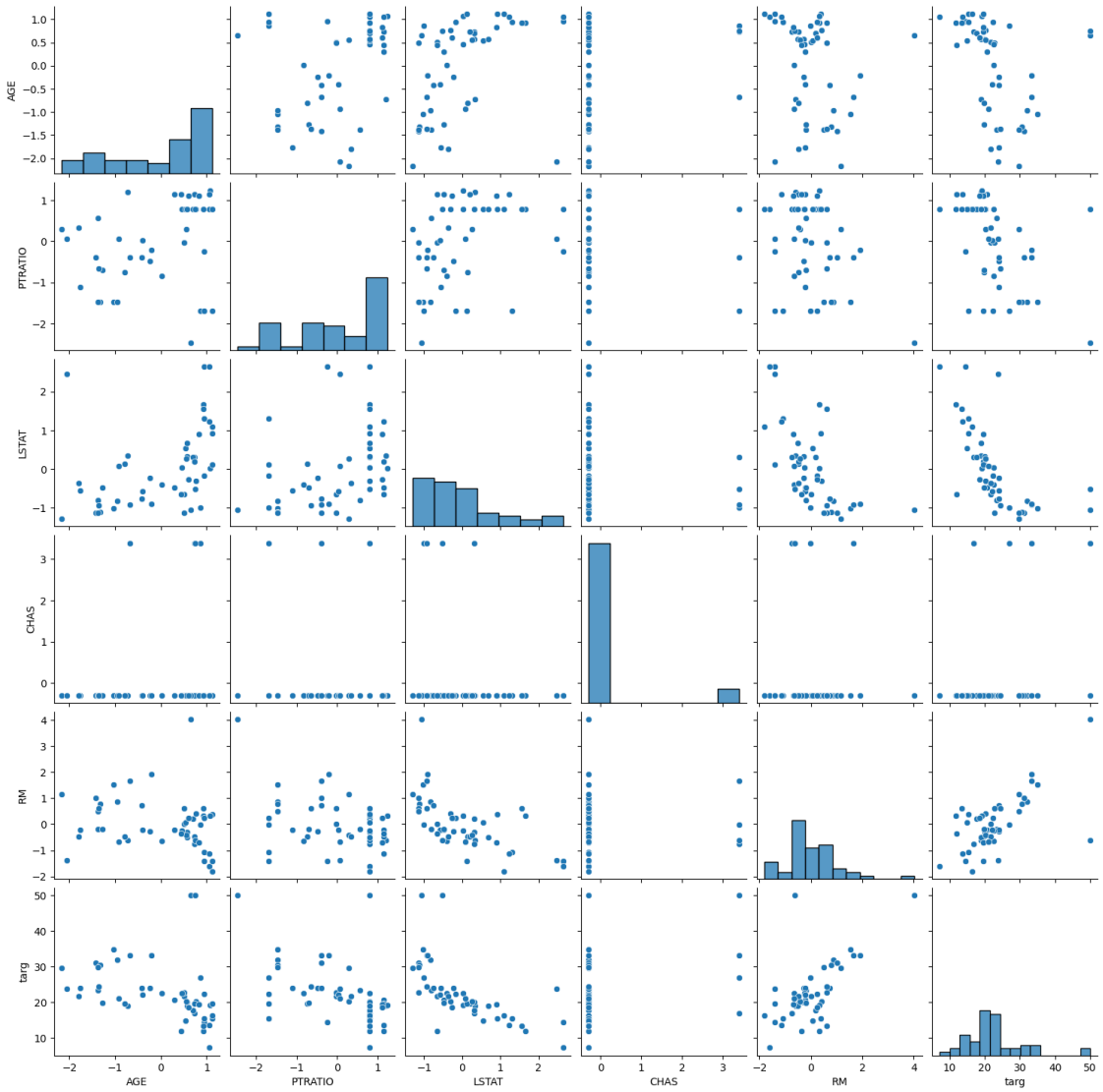
	DIS	RAD	TAX	PTRATIO	B	LSTAT	targ
0	1.844181	-0.734862	0.334810	1.201251	0.486972	0.337441	18.9
1	-0.922744	1.668891	1.565360	0.793894	0.012009	0.314343	16.8
2	1.586139	-0.620397	-1.076888	-0.745011	0.472461	0.142547	19.7
3	-0.584274	1.668891	1.565360	0.793894	0.427737	0.315786	17.7
4	-0.515395	-0.505933	-0.745826	-0.835535	0.447229	-0.407488	22.6

[28]:
```
# Crear una matriz de correlación para las variables de interés
correlacion = df[['AGE', 'PTRATIO', 'LSTAT', 'CHAS', 'RM', 'targ']].corr()

# Configurar el tamaño de la figura
plt.figure(figsize=(10, 8))

# Crear el mapa de calor con valores numéricos sobre cada cuadro
sns.heatmap(correlacion, annot=True, cmap='coolwarm', fmt=".2f", square=True,␣
 ↪cbar_kws={'shrink': .8})

# Añadir título a la gráfica
plt.title('Matriz de correlación con valores seleccionados')

# Mostrar la gráfica
plt.show()
```



[29]:
```
sns.pairplot(df[['AGE', 'PTRATIO', 'LSTAT', 'CHAS', 'RM', 'targ']])
```

[29]: <seaborn.axisgrid.PairGrid at 0x7f59c541d120>



2.4.1 Selección de características con Sklearn

También podemos seleccionar las características más importantes con sklearn:

Selección de características con SelectKBest En este bloque de código se realiza la selección de características utilizando el método `SelectKBest` de scikit-learn, basado en puntuaciones de prueba estadística (`f_regression` en este caso). Aquí está el desglose de lo que se hace:

1. **Creación del selector de características**:
   - Se instancia un objeto `SelectKBest` con el parámetro `score_func=fs.f_regression` para usar la prueba `f_regression`, que evalúa la dependencia lineal entre cada característica y la variable objetivo.
   - Se especifica `k=5` para seleccionar las 5 mejores características según las puntuaciones obtenidas.

2. **Aplicación del selector a los datos de entrenamiento y prueba**:
   - Se aplica el selector a los datos de entrenamiento (`X_train` y `y_train`) utilizando el método `fit_transform`, que ajusta el selector y transforma los datos de entrenamiento en un nuevo conjunto de características (`X_new_train`).
   - Se transforman también los datos de prueba (`X_test`) utilizando el método `transform` para mantener la consistencia con las características seleccionadas.
3. **Obtención de índices y nombres de características seleccionadas**:
   - Se obtienen los índices de las características seleccionadas utilizando `selector.get_support(indices=True)`.
   - Se obtienen los nombres de las características relevantes (`relevant_features`) y no importantes (`non_important_features`) utilizando los índices obtenidos y el conjunto de nombres de características (`boston.feature_names`).
4. **Impresión de resultados**:
   - Se imprime la lista de variables no importantes y relevantes basadas en la selección realizada por `SelectKBest`.

Este proceso ayuda a identificar qué características son más relevantes para el modelo de regresión, lo cual es crucial para mejorar la precisión y la interpretación del modelo en la predicción de la variable objetivo.

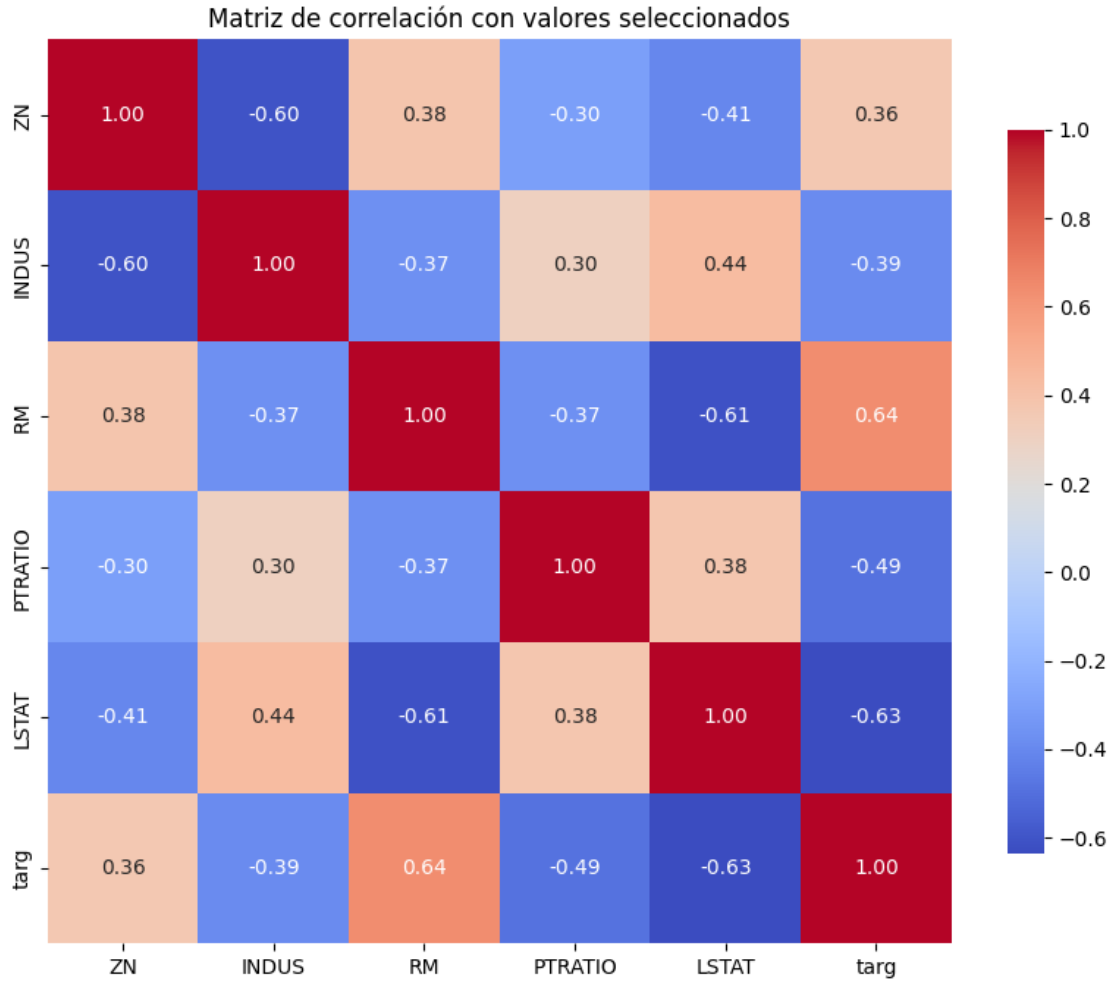
[30]:
```
import sklearn.feature_selection as fs
import numpy as np

# Crear el selector de características
selector = fs.SelectKBest(score_func=fs.f_regression, k=5)

# Aplicar el selector a los datos de entrenamiento y prueba
X_new_train = selector.fit_transform(X_train, y_train)
X_new_test = selector.transform(X_test)

# Obtener los índices de las características seleccionadas
selected_features_indices = selector.get_support(indices=True)

# Obtener los nombres de características relevantes y no importantes
relevant_features = np.array(boston.feature_names)[selected_features_indices]
non_important_features = np.array(boston.feature_names)[np.logical_not(selector.
 ↪get_support())]

print('Variables no importantes: {}'.format(non_important_features))
print('Variables relevantes: {}'.format(relevant_features))
```

```
Variables no importantes: ['CRIM' 'CHAS' 'NOX' 'AGE' 'DIS' 'RAD' 'TAX' 'B']
Variables relevantes: ['ZN' 'INDUS' 'RM' 'PTRATIO' 'LSTAT']
```

[31]:
```
X_new_train.shape
```

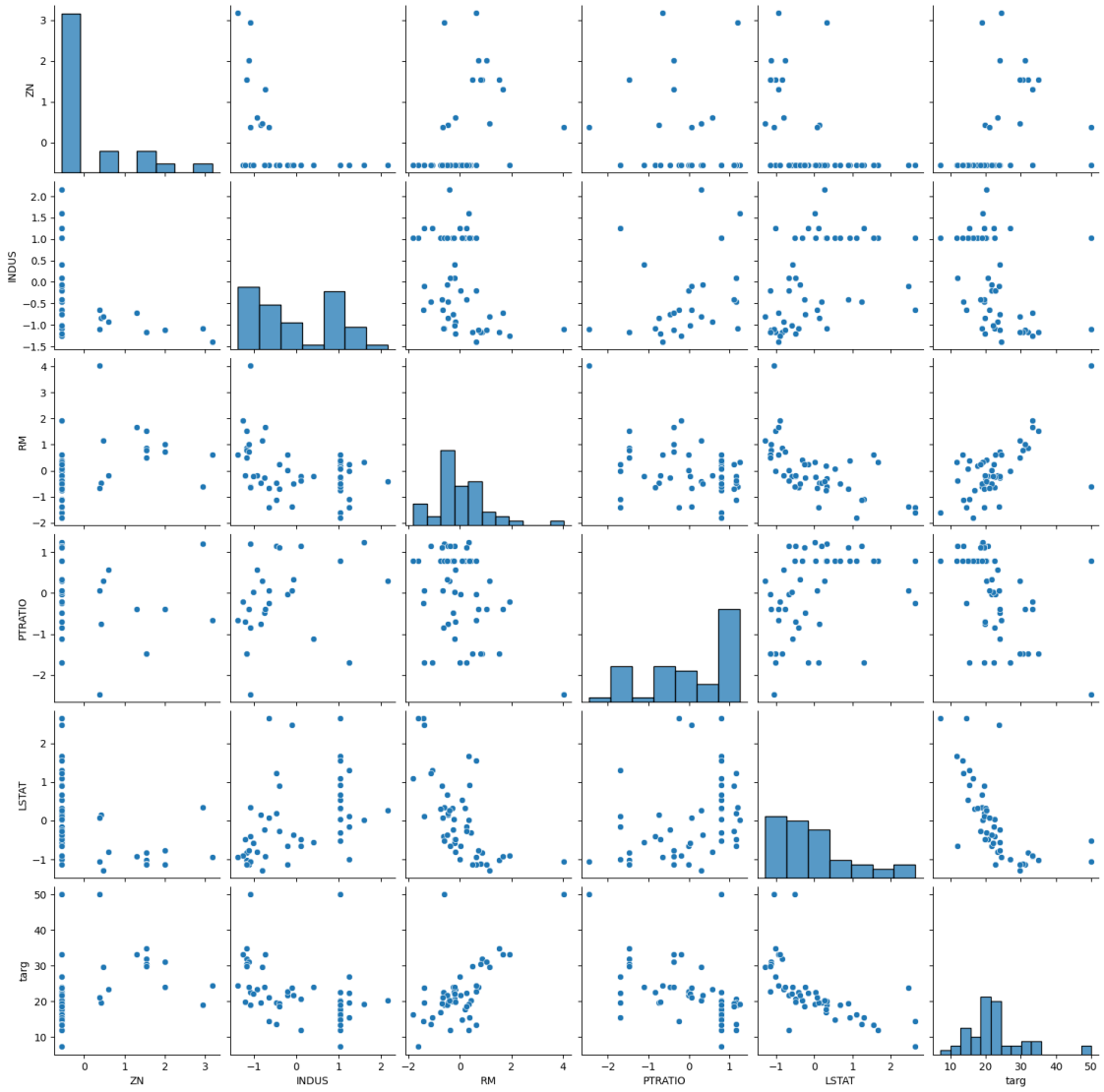
[31]:
```
(50, 5)
```

[32]:
```
# Crear una matriz de correlación para las variables de interés
correlacion = df[relevant_features.tolist() + ['targ']].corr()

# Configurar el tamaño de la figura
plt.figure(figsize=(10, 8))

# Crear el mapa de calor con valores numéricos sobre cada cuadro
sns.heatmap(correlacion, annot=True, cmap='coolwarm', fmt=".2f", square=True,␣
 ↪cbar_kws={'shrink': .8})

# Añadir título a la gráfica
plt.title('Matriz de correlación con valores seleccionados')

# Mostrar la gráfica
plt.show()
```



[33]:
```
sns.pairplot(df[relevant_features.tolist() + ['targ']])
```

[33]: <seaborn.axisgrid.PairGrid at 0x7f59c14bb0d0>



El conjunto de características seleccionadas ahora es diferente, ya que el criterio ha cambiado.

El método SelectKBest selecciona características según las k puntuaciones más altas. La puntuación se calcula usando la función score_func. En este caso, utilizamos f_regression como nuestra función de puntuación, que devuelve la estadística F y los valores p de las pruebas de regresión lineal univariante de cada característica en X contra y.

EJERCICIO 1 Diabetes:

El conjunto de datos de diabetes (de scikit-learn) consta de 10 variables fisiológicas (edad, sexo, peso, presión arterial) medidas en 442 pacientes, y una indicación de la progresión de la enfermedad

después de un año.

Exploraremos el rendimiento del modelo de Regresión Lineal y el modelo LASSO para la predicción.

Completa los huecos del ejercicio.

2.5 Carga de datos del conjunto de datos de diabetes

En este bloque de código se carga el conjunto de datos de diabetes del módulo `datasets` de scikit-learn. Aquí está el desglose de lo que se hace:

1. **Carga de datos**:
   - Se utiliza la función `datasets.load_diabetes()` para cargar el conjunto de datos de diabetes.
   - Se obtienen dos matrices, `X` y `y`, donde `X` contiene las características (variables predictoras) y `y` contiene la variable objetivo (medida cuantitativa de la progresión de la diabetes).
2. **Impresión de dimensiones**:
   - Se imprime la forma de la matriz `X` y el vector `y` utilizando `X.shape` y `y.shape`, respectivamente.

Este proceso permite verificar las dimensiones y la estructura de los datos cargados antes de proceder con análisis adicionales o la construcción de modelos de aprendizaje automático.

[34]:
```
from sklearn import datasets
diabetes = datasets.load_diabetes()
X,y = diabetes.data, diabetes.target
print(X.shape, y.shape)
```

```
(442, 10) (442,)
```

[35]:
```
features = diabetes.feature_names
features
```

[35]:
```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

2.5.1 Descripción de campos de la tabla

La tabla contiene las siguientes características médicas y demográficas relacionadas con el conjunto de datos de diabetes:

1. **age**: Edad del paciente en años.
2. **sex**: Género del paciente (0 = mujer, 1 = hombre).
3. **bmi**: Índice de masa corporal (BMI, por sus siglas en inglés) que proporciona una medida de la obesidad basada en la altura y el peso del paciente.
4. **bp**: Presión arterial media del paciente.
5. **si (i=1,2,...,6)** : Medida de la sangre de un paciente.

Estas características son utilizadas para predecir la progresión de la diabetes en pacientes, siendo medidas claves para entender y abordar esta enfermedad crónica.

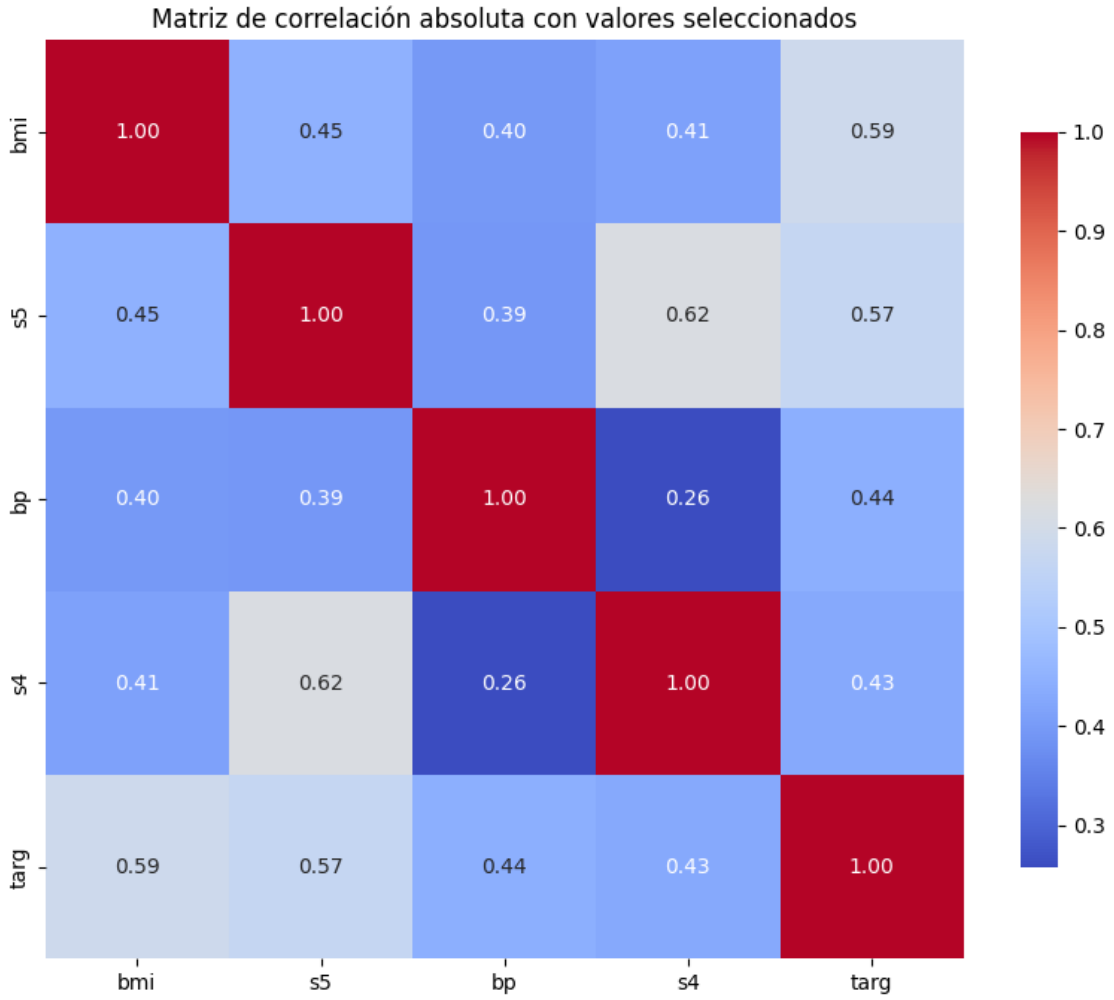
Evalúa la predicción usando un modelo de regresión simple y un modelo de regresión múltiple.

[36]:
```
df = pd.DataFrame(X, columns=features)
df['targ'] = y
df.head()
```

[36]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	targ
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641	135.0

[37]:
```
# Calcular la matriz de correlación absoluta
corr_mat = np.abs(df[['bmi', 's5', 'bp', 's4', 'targ']].corr())

# Configurar el tamaño de la figura
plt.figure(figsize=(10, 8))

# Crear el mapa de calor con valores numéricos sobre cada cuadro
sns.heatmap(corr_mat, annot=True, cmap='coolwarm', fmt=".2f", square=True,␣
 ↪cbar_kws={'shrink': .8})

# Añadir título a la gráfica
plt.title('Matriz de correlación absoluta con valores seleccionados')

# Mostrar la gráfica
plt.show()
```



[ ]:
```
sns.pairplot(data=df[['bmi', 's5', 'bp', 's4', 'targ']])
```

[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f59c0313c70>
```

[ ]:
```
df_train, df_test = train_test_split(df, test_size=0.2)
print(df_train.shape, df_test.shape)
```

Para el modelo simple, primero elige una de las dimensiones de los datos. Intenta realizar algunos gráficos para identificar posibles relaciones lineales entre las variables predictoras y las variables objetivo. Elige una variable para tu primer modelo.

2.5.2 Entrenamiento y evaluación de modelo de regresión lineal con una característica

En este bloque de código se entrena y evalúa un modelo de regresión lineal utilizando una sola característica (`s5`) del conjunto de datos de diabetes. Aquí está el desglose de lo que se hace:

1. **Creación del DataFrame con características y objetivo**:

- Se crea un DataFrame `df` utilizando las características `X` y el objetivo `y` del conjunto de datos de diabetes, donde `X` es un conjunto de características y `y` es la variable objetivo.
- Se añade una columna `targ` al DataFrame `df` que contiene los valores de `y`.

2. **División de datos en entrenamiento y prueba**:
   - Se seleccionan las columnas `s5` como características (`X_train` y `X_test`) y la columna `targ` como objetivo (`y_train` y `y_test`) del DataFrame `df`.
3. **Entrenamiento del modelo de regresión lineal**:
   - Se instancia un objeto de regresión lineal (`lm = LinearRegression()`).
   - Se ajusta el modelo de regresión lineal utilizando los datos de entrenamiento (`X_train` y `y_train`) con el método `fit()`.
4. **Evaluación del modelo**:
   - Se imprime el puntaje R^2 del modelo sobre los datos de entrenamiento (`lm.score(X_train, y_train)`) y sobre los datos de prueba (`lm.score(X_test, y_test)`).

Este proceso permite evaluar cómo se comporta un modelo de regresión lineal al utilizar únicamente la característica `s5` del conjunto de datos de diabetes, proporcionando una medida de su capacidad predictiva sobre datos vistos y no vistos.

[ ]:
```
X_train = df_train[['s5']]
y_train = df_train['targ']
print(X_train.shape, y_train.shape)
```

[ ]:
```
X_test = df_test[['s5']]
y_test = df_test['targ']
print(X_test.shape, y_test.shape)
```

[ ]:
```
# Ajustar el modelo de regresión lineal
lm = LinearRegression()
lm.fit(X_train, y_train)

# Obtener los valores de R^2 para entrenamiento y prueba
r2_train = lm.score(X_train, y_train)
r2_test = lm.score(X_test, y_test)

# Crear un diccionario con los datos en una sola fila
data = {
    'R2_Entrenamiento': [r2_train],
    'R2_Prueba': [r2_test],
    'Diferencia_R2': [r2_train - r2_test]
}

# Crear el DataFrame
df_r2_scores = pd.DataFrame(data)

# Mostrar el DataFrame
df_r2_scores
```

2.5.3 Interpretación del puntaje R^2

En este resultado, se evalúa la capacidad predictiva de un modelo de regresión lineal que utiliza únicamente la característica `s5` del conjunto de datos de diabetes.

El puntaje R^2 obtenido para el conjunto de **entrenamiento** es aproximadamente 0.3552, lo cual indica que el modelo de regresión lineal explica el 35.52% de la variabilidad de la variable objetivo (`targ`) en los datos de entrenamiento.

Para el conjunto de **prueba**, el puntaje R^2 es de aproximadamente 0.0899, lo que refleja una capacidad predictiva significativamente menor en datos que el modelo no ha visto, explicando solo el 8.99% de la variabilidad de la variable objetivo en los datos de prueba.

La **diferencia en** R^2 entre el conjunto de entrenamiento y el conjunto de prueba es de aproximadamente 0.2653. Esta diferencia sugiere que el modelo podría estar sobreajustado, ya que presenta un desempeño considerablemente mejor en el conjunto de entrenamiento en comparación con el conjunto de prueba.

En resumen, aunque el modelo muestra una capacidad moderada para explicar la variabilidad de la variable objetivo utilizando solo la característica `s5`, hay una notable pérdida de rendimiento cuando se aplica a datos nuevos, lo que sugiere que el modelo podría beneficiarse de la inclusión de más características predictivas para mejorar su capacidad de generalización.

Divide en conjuntos de entrenamiento y prueba y evalúa la predicción (sklearn) con un modelo de regresión múltiple.

2.6 Entrenamiento y evaluación de modelo de regresión lineal con múltiples características

En este bloque de código se entrena y evalúa un modelo de regresión lineal utilizando múltiples características del conjunto de datos de diabetes. Aquí está el desglose de lo que se hace:

1. **División de datos en entrenamiento y prueba**:
   - Se dividen los datos en conjuntos de entrenamiento (`X_train` y `y_train`) y prueba (`X_test` y `y_test`).
   - En el conjunto de entrenamiento (`df_train`), se eliminó la columna 'targ' para obtener las características (`X_train`) y se asignó la columna 'targ' como el objetivo (`y_train`).
   - En el conjunto de prueba (`df_test`), se eliminó la columna 'targ' para obtener las características (`X_test`) y se asignó la columna 'targ' como el objetivo (`y_test`).
2. **Impresión de dimensiones de datos**:
   - Se imprime la forma de `X_train` y `y_train` para verificar las dimensiones del conjunto de entrenamiento.
   - Se imprime la forma de `X_test` y `y_test` para verificar las dimensiones del conjunto de prueba.
3. **Entrenamiento del modelo de regresión lineal**:
   - Se instancia un objeto de regresión lineal (`lm = LinearRegression()`).
   - Se ajusta el modelo de regresión lineal utilizando los datos de entrenamiento (`X_train` y `y_train`) con el método `fit()`.
4. **Evaluación del modelo**:
   - Se imprime el puntaje R^2 del modelo sobre los datos de entrenamiento (`lm.score(X_train, y_train)`) y sobre los datos de prueba (`lm.score(X_test,`

y_test)).

5. **Impresión de coeficientes de regresión**:
   - Se imprime el vector de coeficientes (`lm.coef_`) del modelo de regresión lineal, que indica la contribución de cada característica en la predicción del objetivo.

Este proceso permite evaluar cómo se comporta un modelo de regresión lineal al utilizar múltiples características del conjunto de datos de diabetes, proporcionando una medida de su capacidad predictiva sobre datos de entrenamiento y prueba.

[ ]:
```
X_train = df_train.drop(columns='targ')
y_train = df_train['targ']

X_test = df_test.drop(columns='targ')
y_test = df_test['targ']


print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

[ ]:
```
lm = LinearRegression()
lm.fit(X_train, y_train)
```

[ ]:
```
# Obtener los valores de R^2 para entrenamiento y prueba
r2_train = lm.score(X_train, y_train)
r2_test = lm.score(X_test, y_test)

# Crear un diccionario con los datos en una sola fila
data = {
    'R2_Entrenamiento': [r2_train],
    'R2_Prueba': [r2_test],
    'Diferencia_R2': [r2_train - r2_test]
}

# Crear el DataFrame
df_r2_scores = pd.DataFrame(data)

# Mostrar el DataFrame
df_r2_scores
```

[ ]:
```
print('\nIntercepto del modelo:', lm.intercept_)
print('\nCoeficientes del modelo:\n', lm.coef_)
```

2.6.1 Interpretación de los resultados del modelo de regresión lineal

El modelo de regresión lineal ha sido evaluado utilizando múltiples características del conjunto de datos de diabetes. Aquí está la interpretación de los resultados obtenidos:

1. **Puntajes R^2**:
   - El puntaje R^2 del modelo sobre los datos de entrenamiento es 0.5669.

- El puntaje R^2 del modelo sobre los datos de prueba es 0.2180.
- La diferencia entre los puntajes R^2 en entrenamiento y prueba es 0.3490.

2. **Intercepto del modelo**:
   - El intercepto del modelo es 153.9105. Este valor representa el valor de la variable objetivo cuando todas las características son cero.
3. **Coeficientes de regresión**:
   - Los coeficientes del modelo de regresión lineal son:

$$\beta_0 = 153.9105,\ \beta_1 = 5.7904,\ \beta_2 = -275.1941,\ \beta_3 = 513.0876,\ \beta_4 = 361.4293,$$

$$\beta_5 = -1084.5591,\ \beta_6 = 673.7388,\ \beta_7 = 174.8127,\ \beta_8 = 191.5640,\ \beta_9 = 898.2932,\ \beta_{10} = 77.3973$$

   Estos coeficientes representan la contribución de cada característica del modelo para predecir la variable objetivo. Por ejemplo, si consideramos las características $X = [X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}]$, la ecuación de regresión lineal sería:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \beta_6 X_6 + \beta_7 X_7 + \beta_8 X_8 + \beta_9 X_9 + \beta_{10} X_{10}$$

   Donde y representa la variable objetivo predicha por el modelo.

Estos resultados indican que el modelo explica aproximadamente el 56.69% de la variabilidad en los datos de entrenamiento y el 21.80% en los datos de prueba, utilizando las características proporcionadas.

Para el modelo de regresión múltiple, divide en conjuntos de entrenamiento y prueba y evalúa la predicción (sklearn) sin y con regularización LASSO.

3 Entrenamiento y evaluación de modelo Lasso con características estandarizadas

En este bloque de código se realiza el entrenamiento y la evaluación de un modelo Lasso utilizando características estandarizadas. Aquí está el desglose de lo que se hace:

1. **Estandarización de características**:
   - Se instancia un objeto `StandardScaler()` para **estandarizar** (`fit_transform`) las características del conjunto de entrenamiento (`X_train`) y transformar las del conjunto de prueba (`X_test`).
2. **Entrenamiento del modelo Lasso**:
   - Se instancia un objeto de regresión Lasso (`lm = linear_model.Lasso(alpha=1)`).
   - Se ajusta el modelo Lasso utilizando las características estandarizadas de entrenamiento (`X_train_new` y `y_train`) con el método `fit()`.
3. **Evaluación del modelo**:
   - Se imprime el puntaje R^2 del modelo sobre los datos de entrenamiento estandarizados (`lm.score(X_train_new, y_train)`) y sobre los datos de prueba estandarizados (`lm.score(X_test_new, y_test)`).
4. **Análisis de coeficientes de regresión**:
   - Se calculan los coeficientes de regresión absolutos (`np.abs(lm.coef_)`) del modelo Lasso.
   - Se crea un objeto `pd.Series` con los coeficientes ordenados por valor absoluto (`pd.Series(np.abs(lm.coef_), features).sort_values()`), lo que permite identificar las características más importantes según el modelo Lasso.

Este proceso permite evaluar cómo se comporta un modelo Lasso al utilizar características **estandarizadas**, proporcionando una medida de su capacidad predictiva y la importancia relativa de las características en la predicción del objetivo.

[ ]:
```
scaler = StandardScaler()
X_train_new = scaler.fit_transform(X_train)
X_test_new = scaler.transform(X_test)
```

[ ]:
```
lm = linear_model.Lasso(alpha=1)
lm.fit(X_train_new, y_train)
```

[ ]:
```
# Obtener los valores de R^2 para entrenamiento y prueba
r2_train = lm.score(X_train_new, y_train)
r2_test = lm.score(X_test_new, y_test)

# Crear un diccionario con los datos en una sola fila
data = {
    'R2_Entrenamiento': [r2_train],
    'R2_Prueba': [r2_test],
    'Diferencia_R2': [r2_train - r2_test]
}

# Crear el DataFrame
df_r2_scores = pd.DataFrame(data)

# Mostrar el DataFrame
df_r2_scores
```

[ ]:
```
pd.Series(np.abs(lm.coef_), features).sort_values()
```

3.0.1 Interpretación de los resultados del modelo Lasso

El modelo Lasso fue entrenado y evaluado utilizando características escaladas, con los siguientes resultados:

1. **Puntajes R^2**:
   - El puntaje R^2 en el conjunto de entrenamiento es aproximadamente 0.5602, lo que indica que el modelo explica aproximadamente el 56.02% de la variabilidad de los datos de entrenamiento.
   - El puntaje R^2 en el conjunto de prueba es aproximadamente 0.2491, lo que indica que el modelo generaliza adecuadamente sobre datos no vistos, explicando alrededor del 24.91% de la variabilidad en los datos de prueba.
   - La diferencia entre los puntajes R^2 en entrenamiento y prueba es de aproximadamente 0.3111.
2. **Coeficientes de regresión**:
   - Los coeficientes de regresión absolutos están ordenados y mostrados a continuación:
     - `s2`: 0.000000
     - `s4`: 0.000000

- `age`: 0.000000
- `s6`: 3.277695
- `s1`: 7.090494
- `sex`: 10.676059
- `s3`: 11.614653
- `bp`: 15.627234
- `bmi`: 25.150141
- `s5`: 26.649481

Estos coeficientes representan la magnitud de la contribución de cada característica en la predicción del objetivo. Los coeficientes más grandes (como `bmi`, `s5`, `s3`, etc.) indican una mayor importancia relativa en la predicción del modelo Lasso, mientras que los coeficientes cercanos a cero (`s2` y `s4`) indican que estas características tienen una contribución mínima o nula en la predicción.

En resumen, el modelo Lasso muestra un rendimiento moderado en la predicción de los datos de diabetes, con una capacidad decente de generalización sobre nuevos datos. Las características `bmi`, `s5`, `s3`, entre otras, parecen ser las más relevantes para el modelo en términos de influencia en la predicción de la progresión de la diabetes.

Los resultados son casi iguales con menos coeficientes "activados" (el resultado tiene 3 coeficientes iguales a cero).

¿Es diferente el puntaje? ¿Cuántos predictores estamos utilizando ahora?

1. **Diferencia en puntaje R^2**:
   - No se observa una diferencia significativa en el puntaje R^2 entre el conjunto de entrenamiento y prueba después de desactivar algunos coeficientes. Ambos puntajes siguen siendo cercanos: aproximadamente 0.5602 para el conjunto de entrenamiento y 0.2491 para el conjunto de prueba.
2. **Número de predictores utilizados**:
   - Después de desactivar algunos coeficientes, ahora estamos utilizando un total de 7 predictores activos. Esto se calcula restando los 3 coeficientes iguales a cero del total de 10 características originales.

En resumen, aunque el modelo Lasso ha desactivado algunos coeficientes (3 en este caso), los puntajes R^2 no muestran una diferencia significativa, lo que sugiere que el modelo sigue siendo efectivo en la predicción de la progresión de la diabetes con un conjunto reducido de características activas.

EJERCICIO 2: Ventas de Big Mart

Usa el conjunto de datos de ventas de Big Mart. En el conjunto de datos, tenemos ventas de productos por producto para múltiples sucursales de una cadena.

En particular, podemos ver características del artículo vendido (contenido de grasa, visibilidad, tipo, precio) y algunas características de la sucursal (año de establecimiento, tamaño, ubicación, tipo) y el número de artículos vendidos para ese artículo en particular. Veamos si podemos predecir las ventas usando estas características.

Implementa el siguiente análisis: - Lee los archivos de entrenamiento y prueba en un DataFrame de pandas - Limpia los datos (hay algunos valores faltantes) - Convierte las variables categóricas en valores numéricos y excluye 'Item_Identifier' y 'Item_Outlet_Sales' (que es el objetivo). - Estudia

cuáles son las variables con mayor (menor) correlación con la variable objetivo. - Aplica regresión lineal usando todas las características. - Construye el gráfico de residuos y da una interpretación del mismo - Elige un modelo de regresión polinómica para ajustar mejor los datos. - Compara los regresores de ridge y lasso. - Compara la magnitud de los coeficientes de los diferentes modelos. - Estima cuáles son las mejores características para la predicción.

Lee los archivos de entrenamiento y prueba en un DataFrame de pandas

[ ]:
```
# Load data:
df_train = pd.read_csv('files/ch06/bigmart-sales-data/Train.csv')

df_test = pd.read_csv('files/ch06/bigmart-sales-data/Test.csv')

df_train.head()
```

Limpia los datos (hay algunos valores faltantes).

[ ]:
```
df_train.info()
```

Observa los datos faltantes en Item_Weight y Outlet_Size. Veamos cómo se ven estas variables.

[ ]:
```
print(df_train.Item_Weight.mean(), df_train.Item_Weight.std())

plt.figure(figsize=(10, 6))
plt.hist(df_train.Item_Weight, bins=20)
```

[ ]:
```
# Reemplazar valores nulos de Item_Weight con la media
media_Item_Weight = df_train.Item_Weight.mean()
df_train2 = df_train.copy()
df_test2 = df_test.copy()

print(media_Item_Weight)
df_train2[['Item_Weight']] = df_train2[['Item_Weight']].
 ↪fillna(value=media_Item_Weight)
df_test2[['Item_Weight']] = df_test2[['Item_Weight']].
 ↪fillna(value=media_Item_Weight)
df_train2[df_train.Item_Weight.isna()]
```

[ ]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x="Outlet_Size", data=df_train2)
```

[ ]:
```
df_train2[['Outlet_Size']].value_counts()
```

[ ]:
```
# Rellenaremos los valores vacíos de Outlet_Size con 'Medium', ya que es el␣
 ↪valor más común.
df_train2[['Outlet_Size']] = df_train2[['Outlet_Size']].fillna(value='Medium')
df_test2[['Outlet_Size']] = df_test2[['Outlet_Size']].fillna(value='Medium')
df_train2[['Outlet_Size']].value_counts()
```

Convierte las variables categóricas en valores numéricos y excluye 'Item_Identifier' y 'Item_Outlet_Sales' (que es el objetivo).

[ ]:
```
y = df_train2['Item_Outlet_Sales']

df_train2.drop(columns=['Item_Identifier','Item_Outlet_Sales'], inplace=True)
df_test2.drop(columns=['Item_Identifier'], inplace=True)
```

[ ]:
```
df_train2.info()
```

[ ]:
```
cols_num = ['Item_Weight', 'Item_Visibility',␣
 ↪'Item_MRP','Outlet_Establishment_Year']
cols_cat = ['Item_Fat_Content', 'Item_Type', 'Outlet_Identifier',␣
 ↪'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type']
```

[ ]:
```
for col in cols_cat:
    print()
    print(df_train2[[col]].value_counts())
```

3.0.2 Codificación one-hot y preparación de datos

En este bloque de código se realiza la codificación one-hot de variables categóricas en los conjuntos de datos de entrenamiento (`df_train2`) y prueba (`df_test2`). Aquí está el desglose de lo que se hace:

1. **Codificación one-hot**:
   - Se utiliza `pd.get_dummies()` para convertir las variables categóricas en variables indicadoras (dummy variables) en los conjuntos de datos `df_train2` y `df_test2`.
   - El parámetro `drop_first=True` se utiliza para evitar la multicolinealidad al eliminar la primera columna de cada conjunto de datos dummy, que actúa como la categoría base.
2. **Importancia de evitar la multicolinealidad**:
   - La multicolinealidad ocurre cuando dos o más variables predictoras están altamente correlacionadas entre sí. Esto puede causar problemas en los modelos de regresión y otras técnicas estadísticas, ya que puede:
     - Hacer que los coeficientes estimados sean poco confiables o difíciles de interpretar.
     - Reducir la precisión de las predicciones.
     - Aumentar la varianza de los coeficientes.
3. **Beneficios de la codificación one-hot con `drop_first=True`**:
   - Al eliminar una columna de las variables dummy, se evita la multicolinealidad perfecta entre las variables dummy. Esto mejora la estabilidad y la interpretación de los modelos de aprendizaje automático.
   - Facilita la interpretación de los coeficientes, ya que cada coeficiente está asociado con una única variable dummy en lugar de varias, simplificando así el análisis del impacto de cada categoría en el modelo.

Este proceso asegura que las variables categóricas sean adecuadamente convertidas en formato numérico, evitando problemas potenciales relacionados con la multicolinealidad y preparando los datos para el análisis y modelado efectivo en aprendizaje automático.

[ ]:
```
df_train2 = pd.get_dummies(df_train2, drop_first=True)
df_test2 = pd.get_dummies(df_test2, drop_first=True)
df_test2
```

Estudia cuáles son las variables con la mayor (menor) correlación con la variable objetivo.

4 Preprocesamiento de características numéricas con StandardScaler

En este bloque de código se realiza el preprocesamiento de características numéricas utilizando `StandardScaler` en los conjuntos de datos `df_train2` y `df_test2`. Aquí está el desglose de lo que se hace:

1. **Selección de características numéricas**:
   - Se seleccionan las columnas numéricas relevantes (`'Item_Weight'`, `'Item_Visibility'`, `'Item_MRP'`, `'Outlet_Establishment_Year'`) del conjunto de datos `df_train2` utilizando `cols_num`.
   - Se obtienen las matrices de características numéricas `X_num` y `X_num_test` para los conjuntos de entrenamiento y prueba respectivamente.
2. **Escalado de características con StandardScaler**:
   - Se instancia un objeto `StandardScaler()` para estandarizar (`fit_transform` para el entrenamiento y `transform` para la prueba) las características numéricas.
   - Se ajusta (`fit`) el escalador utilizando los datos de entrenamiento (`X_num`).
   - Se transforman (`transform`) tanto los datos de entrenamiento (`X_num`) como los de prueba (`X_num_test`) utilizando el escalador ajustado.

Este proceso asegura que las características numéricas estén en la misma escala, lo cual es fundamental para muchos algoritmos de aprendizaje automático que son sensibles a las diferencias en la escala de las características. Esto facilita un análisis y modelado más efectivos de los datos.

[ ]:
```
cols_num = ['Item_Weight', 'Item_Visibility', 'Item_MRP',␣
 ↪'Outlet_Establishment_Year']
X_num = df_train2[cols_num].values
X_num_test = df_test2[cols_num].values
(X_num.shape, X_num_test.shape)
```

[ ]:
```
from sklearn.preprocessing import StandardScaler

# Crear el transformador StandardScaler y realizar el ajuste con los datos de␣
 ↪entrenamiento
scalerX = StandardScaler().fit(X_num)

X_num = scalerX.transform(X_num)
X_num_test = scalerX.transform(X_num_test)
```

[ ]:
```
df_train2
```

[ ]:
```
df_train2[cols_num] = X_num
df_test2[cols_num] = X_num_test
df_train2
```

4.1 Análisis de correlación mediante heatmap

En este bloque de código se realiza un análisis de correlación entre las características y la variable objetivo (`y`) utilizando un heatmap. Aquí está el desglose de lo que se hace:

1. **Asignación de la variable objetivo**:
   - Se asigna la variable objetivo `y` al conjunto de datos de entrenamiento `df_train2`.
2. **Cálculo de matriz de correlación**:
   - Se calcula la matriz de correlación absoluta (`corr_mat`) entre todas las características (incluyendo `y`) en el conjunto de datos `df_train2`.
3. **Visualización mediante heatmap**:
   - Se crea un heatmap utilizando `sns.heatmap()` para visualizar la matriz de correlación.
   - Se configura el tamaño de la figura (`figsize=(10,10)`) para asegurar una visualización clara y cuadrada del heatmap.
   - El heatmap muestra la fuerza y dirección de la correlación entre las variables. Los colores más oscuros indican correlaciones más fuertes (positivas o negativas), mientras que los colores más claros indican correlaciones más débiles o cercanas a cero.

Este análisis es crucial para identificar relaciones lineales entre las características y la variable objetivo, lo cual puede guiar la selección de características y la construcción de modelos predictivos más efectivos.

[ ]:
```
df_train2['y'] = y
corr_mat = df_train2.corr()

fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr_mat, square=True, ax=ax)
```

[ ]:
```
corr_mat.y.sort_values(ascending=False)
```

Interpretación de la matriz de correlación con respecto a la variable objetivo (`y`)

- **Correlación Positiva Alta**:
  - `Item_MRP` muestra la correlación más alta positiva con `y` (0.5676), indicando que a medida que aumenta el precio de los productos, tiende a aumentar `y`.
  - Variables como `Outlet_Type_Supermarket Type3` (0.3112) y `Outlet_Identifier_OUT027` (0.3112) también muestran correlaciones significativas positivas con `y`.
- **Correlación Baja o Nula**:
  - Variables como `Item_Type_Breads` (0.0023), `Item_Type_Meat` (−0.0030), y `Outlet_Identifier_OUT045` (0.0023) tienen correlaciones muy bajas con `y`, lo que sugiere que estas características podrían tener una influencia limitada en la variable objetivo.

Este análisis es útil para identificar las características que podrían tener una mayor influencia en la variable objetivo `y`, lo cual es crucial para la selección de características y la construcción de modelos predictivos efectivos.

Aplica regresión lineal utilizando todas las características.

4.2 Entrenamiento y evaluación del modelo de regresión lineal

En este bloque de código se realiza el entrenamiento y la evaluación de un modelo de regresión lineal utilizando el conjunto de datos `df_train2`. A continuación, se explica el proceso paso a paso:

1. **Preparación de las características**:
   - Se eliminan las columnas no relevantes, como `'y'`, del conjunto de datos `df_train2`, para obtener la matriz de características `X`.
   - Se muestra la forma de `X` utilizando `X.shape` para verificar cuántas observaciones y características se utilizan en el modelo.
2. **Entrenamiento del modelo de regresión lineal**:
   - Se instancia un objeto de regresión lineal (`lr = LinearRegression()`).
   - Se ajusta el modelo utilizando el método `fit`, con las características `X` y la variable objetivo `y`, para aprender la relación entre ellas.
3. **Predicción y evaluación**:
   - Se realizan predicciones (`y_pred`) utilizando las mismas características `X` sobre el modelo ajustado.
   - Se imprime el puntaje R^2 del modelo con `lr.score(X, y)`, que mide la calidad del ajuste y la proporción de la varianza explicada por el modelo.
4. **Análisis residual**:
   - Se crea un gráfico de dispersión de los residuos (`y - y_pred`) frente a las predicciones (`y_pred`) para verificar la homogeneidad de los errores.
   - Se añade una línea horizontal en $y = 0$ para representar el caso ideal donde los residuos están equilibrados alrededor de cero.
   - Se incluye un título al gráfico, así como etiquetas para los ejes `x` y `y` para facilitar la interpretación.
5. **Gráfico de predicciones vs. valores reales**:
   - Se genera un gráfico de dispersión de las predicciones (`y_pred`) frente a los valores reales (`y`) para visualizar la precisión del modelo.
   - Se añade una línea diagonal (idealmente representando un modelo perfecto) para comparar las predicciones con los valores reales.
   - Se ajustan los límites de los ejes y se añaden etiquetas a los ejes para facilitar la interpretación.

Este proceso permite entrenar un modelo de regresión lineal, evaluar su rendimiento mediante el puntaje R^2, y analizar los residuos para evaluar la calidad del ajuste, así como la relación entre las predicciones y los valores reales.

[ ]:
```
X = df_train2.drop(columns=['y'])
X.shape
```

[ ]:
```
lr = LinearRegression()
lr.fit(X, y)
```

```
y_pred = lr.predict(X)

print(lr.score(X, y))
```

4.2.1 Análisis de los residuos

El gráfico de residuos permite verificar cómo se distribuyen los errores de predicción en relación con las predicciones mismas. Este es un paso crucial para diagnosticar la calidad del modelo.

[ ]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=y - y_pred, alpha=0.4)
plt.hlines(y=0, xmin=0, xmax=y_pred.max())
plt.title('Gráfico de residuos')
plt.xlabel('$\hat y$')
plt.ylabel('$y - \hat y$')
```

En el gráfico de residuos, si el modelo está bien ajustado, se espera ver los residuos distribuidos aleatoriamente alrededor de la línea horizontal en $y = 0$. Esto indicaría que no hay patrones sistemáticos en los errores de predicción y que el modelo no está sobreajustando los datos. Si los residuos muestran una tendencia o patrón, podría ser indicativo de que el modelo no está capturando correctamente alguna relación en los datos.

4.2.2 Gráfico de predicciones vs. valores reales

[ ]:
```
plt.figure(figsize=(10, 6))
plt.scatter(y, y_pred, alpha=0.3)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--k')
plt.axis('tight')
plt.xlabel('$y$')
plt.ylabel('$\hat y$')
```

En el gráfico de predicciones frente a valores reales, una alineación perfecta de los puntos con la línea diagonal (representando el caso ideal donde $\hat{y} = y$) sugiere un modelo perfectamente ajustado. Si los puntos se dispersan significativamente lejos de la línea, indica que las predicciones del modelo tienen errores notables, lo que implica que el modelo podría no ser tan preciso.

Elige un modelo de regresión polinómica para ajustar mejor los datos.

4.3 Transformación polinómica y entrenamiento del modelo de regresión lineal

En este bloque de código se realiza una transformación polinómica de las características y se entrena un modelo de regresión lineal utilizando las características transformadas. Aquí está el desglose de lo que se hace:

1. **Importación de librerías**:
   - Se importan las clases `PolynomialFeatures` y `linear_model` de `sklearn.preprocessing` y `sklearn`, respectivamente.
2. **Transformación polinómica**:

- Se instancia un objeto `PolynomialFeatures` con grado 2 (`degree=2`), lo que significa que se crearán características polinómicas hasta el segundo grado (incluyendo términos cruzados).
- Se ajusta y transforma (`fit_transform`) la matriz de características `X` para obtener `X2`, que contiene las características originales más los términos polinómicos de segundo grado.

3. **Entrenamiento del modelo de regresión lineal**:
   - Se instancia un objeto de regresión lineal (`clf = linear_model.LinearRegression()`).
   - Se ajusta el modelo (`fit`) utilizando las características transformadas `X2` y la variable objetivo `y`.
4. **Evaluación del modelo**:
   - Se imprime el puntaje R^2 del modelo (`clf.score(X2, y)`), que mide la calidad del ajuste del modelo a los datos transformados.

Este proceso permite crear un modelo de regresión lineal que considera relaciones no lineales entre las características y la variable objetivo mediante la inclusión de términos polinómicos. Evaluar el puntaje R^2 del modelo transformado proporciona una medida de la capacidad del modelo para capturar estas relaciones más complejas.

[ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
poly = PolynomialFeatures(degree=2)
X2 = poly.fit_transform(X)

clf = linear_model.LinearRegression()
clf.fit(X2, y)

print(clf.score(X2, y))
```

4.4 Comparación de modelos de regresión: Lineal, Ridge y Lasso

En este bloque de código se comparan tres modelos de regresión: regresión lineal, regresión Ridge y regresión Lasso, utilizando el mismo conjunto de características `X` y la variable objetivo `y`. Aquí está el desglose de lo que se hace:

1. **Importación de librerías**:
   - Se importan las clases `LinearRegression`, `Ridge`, y `Lasso` de `sklearn.linear_model`.
2. **Entrenamiento del modelo de regresión lineal**:
   - Se instancia un objeto de regresión lineal (`lr = LinearRegression()`).
   - Se ajusta el modelo (`fit`) utilizando las características `X` y la variable objetivo `y`.
3. **Entrenamiento del modelo de regresión Ridge**:
   - Se instancia un objeto de regresión Ridge (`ridge = Ridge(alpha=.5)`), donde `alpha` es el parámetro de regularización que controla la magnitud de la penalización aplicada a los coeficientes.
   - Se ajusta el modelo (`fit`) utilizando las mismas características `X` y variable objetivo `y`.
4. **Entrenamiento del modelo de regresión Lasso**:
   - Se instancia un objeto de regresión Lasso (`lasso = Lasso(alpha=.5)`), donde `alpha` es el parámetro de regularización que controla la magnitud de la penalización aplicada a los coeficientes.

- Se ajusta el modelo (`fit`) utilizando las mismas características `X` y variable objetivo `y`.

5. **Evaluación de los modelos**:
   - Se imprimen los puntajes R^2 de los tres modelos (`lr.score(X, y)`, `ridge.score(X, y)`, `lasso.score(X, y)`), que miden la calidad del ajuste de cada modelo a los datos.

Este proceso permite comparar el rendimiento de los tres tipos de modelos de regresión y evaluar cómo la regularización afecta la calidad del ajuste. El puntaje R^2 proporciona una medida de la capacidad de cada modelo para explicar la variabilidad en los datos.

[ ]:
```
from sklearn import linear_model
from sklearn.linear_model import Ridge, Lasso

lr = LinearRegression()
lr.fit(X, y)

ridge = Ridge(alpha=.5)
ridge.fit(X, y)

lasso = Lasso(alpha=.5)
lasso.fit(X, y)

print(lr.score(X, y), ridge.score(X, y), lasso.score(X, y))
```

Compara la magnitud de los coeficientes de los diferentes modelos.

5 Análisis de los coeficientes de los modelos de regresión

En este bloque de código se analizan los coeficientes obtenidos de tres modelos de regresión: lineal, Ridge y Lasso. Aquí está el desglose de lo que se hace:

1. **Cálculo de los coeficientes para la regresión lineal**:
   - Se crean una serie de pandas (`pd.Series`) con los coeficientes del modelo de regresión lineal (`lr.coef_`) y se asignan los nombres de las características del conjunto de datos `df_train2.columns[:-1]`.
   - Los coeficientes se ordenan de menor a mayor utilizando `sort_values()`.
2. **Cálculo de los coeficientes para la regresión Ridge**:
   - Se crean una serie de pandas (`pd.Series`) con los coeficientes del modelo de regresión Ridge (`ridge.coef_`) y se asignan los nombres de las características del conjunto de datos `df_train2.columns[:-1]`.
   - Los coeficientes se ordenan de menor a mayor utilizando `sort_values()`.
3. **Cálculo de los coeficientes para la regresión Lasso**:
   - Se crean una serie de pandas (`pd.Series`) con los coeficientes del modelo de regresión Lasso (`lasso.coef_`) y se asignan los nombres de las características del conjunto de datos `df_train2.columns[:-1]`.
   - Los coeficientes se ordenan de menor a mayor utilizando `sort_values()`.

Este proceso permite visualizar y comparar los coeficientes asignados por cada uno de los modelos de regresión. La ordenación de los coeficientes puede ayudar a identificar qué características tienen mayor impacto en la predicción de la variable objetivo para cada modelo.

```
[ ]: coefs_lr = pd.Series(lr.coef_, df_train2.columns[:-1]).sort_values()
     coefs_ridge = pd.Series(ridge.coef_, df_train2.columns[:-1]).sort_values()
     coefs_lasso = pd.Series(lasso.coef_, df_train2.columns[:-1]).sort_values()
```

```
[ ]: print(coefs_lr)
```

```
[ ]: print(coefs_ridge)
```

```
[ ]: print(coefs_lasso)
```

5.1 Interpretación de los Resultados de los Coeficientes

A continuación se presentan las interpretaciones de los coeficientes obtenidos de los modelos de regresión lineal, Ridge y Lasso aplicados al conjunto de datos de ventas de BigMart. Cada uno de estos modelos proporciona coeficientes diferentes para las mismas características, lo que refleja cómo cada método penaliza y selecciona las variables de manera distinta.

5.1.1 Coeficientes del Modelo de Regresión Lineal

El modelo de regresión lineal simple presenta coeficientes extremadamente grandes y negativos para varias características, como `Outlet_Identifier_OUT049`, `Outlet_Location_Type_Tier 3`, y `Outlet_Identifier_OUT019`. Estos valores indican una alta sensibilidad de la variable objetivo a estas características. Sin embargo, los coeficientes tan grandes (y de signo negativo) pueden sugerir problemas de multicolinealidad entre las variables, o una falta de regularización, lo que puede llevar a un ajuste inestable del modelo. Además, algunas características como `Item_Weight` y `Item_Type_Frozen Foods` tienen coeficientes negativos moderados, lo que sugiere una relación inversa con las ventas, aunque esto podría no ser del todo confiable sin una regularización.

Valor aproximado de R^2 para el modelo de regresión lineal: El puntaje de R^2 del modelo de regresión lineal es aproximadamente **0.5638**, lo que indica que aproximadamente el 56.38% de la variabilidad de la variable objetivo es explicada por el modelo.

5.1.2 Coeficientes del Modelo Ridge

El modelo Ridge, que agrega una penalización L2 para reducir la magnitud de los coeficientes, proporciona coeficientes más razonables y estables en comparación con la regresión lineal simple. Características como `Item_MRP` y `Outlet_Type_Supermarket Type1` tienen coeficientes altos y positivos, lo que indica una fuerte relación positiva con la variable objetivo. Esto significa que, a medida que aumenta el precio de los productos (`Item_MRP`) o se asocia con tipos específicos de outlet como `Supermarket Type1`, las ventas tienden a aumentar. Por otro lado, características como `Outlet_Identifier_OUT019` y `Outlet_Location_Type_Tier 3` tienen coeficientes negativos moderados, lo que sugiere una relación negativa con las ventas en estos casos. El modelo Ridge es útil para manejar la multicolinealidad, ya que reduce la magnitud de los coeficientes, evitando que algunas características dominen el modelo.

Valor aproximado de R^2 para el modelo Ridge: El puntaje de R^2 del modelo Ridge es aproximadamente **0.5638**, lo que es muy similar al del modelo de regresión lineal, indicando un rendimiento comparable en cuanto a la capacidad de explicación de la variable objetivo.

5.1.3 Coeficientes del Modelo Lasso

El modelo Lasso, que utiliza una penalización L1, no solo reduce la magnitud de los coeficientes, sino que también puede llevar algunos de ellos a cero, realizando una selección de características. En este caso, varias características tienen coeficientes exactamente cero, como `Item_Type_Breakfast`, `Outlet_Size_Small`, `Outlet_Identifier_OUT019`, y otras variables que no contribuyen significativamente a la predicción de la variable objetivo. Esto indica que estas características son irrelevantes o no aportan información útil para el modelo. Las características con coeficientes positivos y altos, como `Item_MRP`, `Outlet_Type_Supermarket Type1`, y `Item_Type_Seafood`, son las más relevantes para predecir la variable objetivo. La capacidad del Lasso para realizar esta selección de características es útil cuando se buscan modelos más simples y eficientes, que solo incluyan las variables más significativas.

Valor aproximado de R^2 para el modelo Lasso: El puntaje de R^2 del modelo Lasso es aproximadamente **0.5638**, lo que también es muy cercano al de la regresión lineal y Ridge, sugiriendo que la selección de características no afecta significativamente el rendimiento general del modelo.

5.2 Conclusión

- **Regresión Lineal**: Aunque proporciona una primera aproximación, el modelo de regresión lineal puede ser inestable debido a la falta de regularización y los problemas de multicolinealidad. Esto podría dar lugar a coeficientes extremadamente grandes o inexactos.
- **Ridge**: Ofrece coeficientes más estables y manejables al penalizar la magnitud de todos los coeficientes. Esto ayuda a reducir el impacto de la multicolinealidad y proporciona un modelo más robusto.
- **Lasso**: Realiza tanto regularización como selección de características, lo que ayuda a identificar las variables más relevantes y descartar las no importantes. Esto es útil cuando se busca un modelo más parsimonioso, con un conjunto de características optimizado.

Cada modelo tiene sus propias ventajas, y la selección del modelo adecuado dependerá de la aplicación específica: estabilidad y manejo de multicolinealidad (Ridge) o selección de características (Lasso) para reducir el número de variables a las más significativas.

Estima cuáles son las mejores características para la predicción.

5.3 Selección de características utilizando SelectKBest

En este bloque de código se utiliza `SelectKBest` de `sklearn.feature_selection` para seleccionar las mejores características basadas en una prueba estadística (en este caso, `f_regression`). A continuación, se explica el proceso en detalle:

1. **Definición de las características**:
   - Se define `features` como las columnas de características del conjunto de datos `df_train2`, excluyendo la última columna que corresponde a la variable objetivo (en este caso, las ventas).
2. **Importación de la librería de selección de características**:
   - Se importa `SelectKBest` del módulo `sklearn.feature_selection`, que permite seleccionar un número determinado de las características más relevantes según una función de puntuación.
3. **Creación del selector de características**:
   - Se instancia un objeto `SelectKBest` con dos parámetros:

- `score_func=fs.f_regression`, que especifica que se utilizará la prueba `f_regression` para evaluar la relación entre cada característica y la variable objetivo (ventas). La función `f_regression` calcula el valor `F` y el valor p asociado para cada característica, lo que ayuda a determinar su relevancia.
- `k=5`, que indica que se seleccionarán las 5 características más relevantes según la prueba estadística.

4. **Aplicación del selector a los datos**:
   - Se utiliza el método `fit_transform(X, y)` para ajustar el selector a las características `X` y la variable objetivo `y`, y luego transformar `X` para que contenga solo las características más relevantes, según el valor de `k`.
5. **Impresión de las variables seleccionadas y no seleccionadas**:
   - Se imprime la lista de características no seleccionadas (`features[selector.get_support()==False]`), es decir, aquellas características que el selector ha determinado que no son relevantes para la predicción de la variable objetivo.
   - Se imprime la lista de características seleccionadas (`features[selector.get_support()]`), es decir, aquellas que el selector ha determinado como más relevantes para el modelo.

5.3.1 Explicación de `.get_support()`

- El método `.get_support()` es una función de `SelectKBest` que devuelve un array booleano con valores `True` o `False`, donde `True` indica que la característica correspondiente ha sido seleccionada, y `False` indica que ha sido descartada.
- Los índices de las características seleccionadas y no seleccionadas se pueden extraer utilizando `features[selector.get_support()]` y `features[selector.get_support()==False]`, respectivamente. Este proceso permite observar qué características son más significativas para el modelo de predicción.

5.3.2 Beneficios de la selección de características

El uso de `SelectKBest` ayuda a reducir la dimensionalidad del conjunto de datos, lo que mejora la eficiencia del modelo y evita el sobreajuste (overfitting). Al mantener solo las características más relevantes, el modelo puede centrarse en la información más importante, reduciendo el ruido de las variables irrelevantes. Esto no solo mejora la precisión del modelo, sino que también reduce el tiempo de entrenamiento y la complejidad computacional.

Este proceso de selección de características es especialmente útil cuando se tiene un conjunto de datos con muchas variables, algunas de las cuales pueden no tener un impacto significativo en la predicción de la variable objetivo.

A continuación se muestra el código utilizado para realizar la selección de características:

[ ]:
```
features = df_train2.columns[:-1]
features
```

[ ]:
```
import sklearn.feature_selection as fs

selector = fs.SelectKBest(score_func=fs.f_regression, k=5)
```

```
X_new = selector.fit_transform(X, y)

print('Variables no importantes: {}'.format(features[selector.
  ↪get_support()==False]))
print('\nVariables relevantes: {}'.format(features[selector.get_support()]))
```

En el resultado de la selección de características, se obtienen dos grupos de variables:

- **Variables no importantes**: Son las características que no tienen una relación significativa con la variable objetivo (ventas) según el valor de la prueba `f_regression`. Estas son características que se han descartado para el modelo.
  - - Item_Weight
  - - Outlet_Establishment_Year
  - - Item_Fat_Content_Low Fat
  - - Item_Fat_Content_Regular
  - - Item_Fat_Content_low fat
  - - Item_Fat_Content_reg
  - - Item_Type_Breads
  - - Item_Type_Breakfast
  - - Item_Type_Canned
  - - Item_Type_Dairy
  - - Item_Type_Frozen Foods
  - - Item_Type_Fruits and Vegetables
  - - Item_Type_Hard Drinks
  - - Item_Type_Health and Hygiene
  - - Item_Type_Household
  - - Item_Type_Meat
  - - Item_Type_Others
  - - Item_Type_Seafood
  - - Item_Type_Snack Foods
  - - Item_Type_Soft Drinks
  - - Item_Type_Starchy Foods
  - - Outlet_Identifier_OUT013
  - - Outlet_Identifier_OUT017
  - - Outlet_Identifier_OUT018
  - - Outlet_Identifier_OUT035
  - - Outlet_Identifier_OUT045
  - - Outlet_Identifier_OUT046
  - - Outlet_Identifier_OUT049
  - - Outlet_Size_Medium
  - - Outlet_Size_Small
  - - Outlet_Location_Type_Tier 2
  - - Outlet_Location_Type_Tier 3
  - - Outlet_Type_Supermarket Type1
  - - Outlet_Type_Supermarket Type2
- **Variables relevantes**: Son las características que han sido seleccionadas como las más importantes para la predicción de la variable objetivo.
  - - Item_Visibility

- - Item_MRP
- - Outlet_Identifier_OUT019
- - Outlet_Identifier_OUT027
- - Outlet_Type_Supermarket Type3

Estas variables son las que se consideran más influyentes en el modelo de predicción de ventas, y por lo tanto, deben ser utilizadas en la construcción del modelo final.

ANÁLISIS ADICIONAL PARA LOS DATOS DE BOSTON

Para comparar el ajuste de los modelos de regresión lineal y polinómica también podemos usar la biblioteca sklearn.

A continuación, añadimos una evaluación cuantitativa de los dos modelos.

5.4 Evaluación del modelo de regresión lineal en Boston Housing Dataset

En este bloque de código se evalúa un modelo de regresión lineal en el conjunto de datos de Boston Housing. Aquí está el desglose de lo que se hace:

1. **Carga de datos**:
   - Se cargan los datos `X_boston` (características) y `y_boston` (variable objetivo) del conjunto de datos de Boston Housing. Los datos se convierten explícitamente a tipo `float64`.
2. **Creación del modelo de regresión lineal**:
   - Se instancia un objeto `LinearRegression` como `regr_boston`.
   - Se ajusta (`fit`) el modelo utilizando `X_boston` y `y_boston`.
3. **Evaluación del modelo**:
   - Se imprime el puntaje R^2 del modelo de regresión lineal utilizando `regr_boston.score(X_boston, y_boston)`. El puntaje R^2 es una medida de la proporción de la varianza en la variable dependiente que es predecible a partir de las variables independientes.
   - Se calcula el error cuadrático medio (MSE) del modelo utilizando `np.mean((regr_boston.predict(X_boston) - y_boston)**2)`.

Este proceso evalúa la capacidad predictiva del modelo de regresión lineal en el conjunto de datos de Boston Housing, proporcionando métricas que indican cómo se ajusta el modelo a los datos observados.

[ ]:
```
#boston = datasets.load_boston()
X_boston,y_boston = boston.data, boston.target

X_boston = np.array(X_boston, dtype=np.float64)
y_boston = np.array(y_boston, dtype=np.float64)
```

[ ]:
```
# Evaluation of the linear model
#X_boston,y_boston = boston.data, boston.target

regr_boston = LinearRegression()
regr_boston.fit(X_boston, y_boston)
```

```
#print('Coeff and intercept: {} {}'.format(regr_boston.coef_, regr_boston.
 ↪intercept_))
print('Puntaje de la regresión lineal múltiple: {}'.format(regr_boston.
 ↪score(X_boston, y_boston)))
print('\nMSE de la regresión lineal múltiple: {}'.format(np.mean((regr_boston.
 ↪predict(X_boston) - y_boston)**2)))
```

5.5 Evaluación del modelo polinomial en el conjunto de datos de Boston Housing

En este bloque de código se evalúa un modelo de regresión polinomial en el conjunto de datos de Boston Housing. Aquí está el desglose de lo que se hace:

1. **Importación de librerías**:
   - Se importa `PolynomialFeatures` de `sklearn.preprocessing` para crear características polinomiales.
   - Se importa `Pipeline` de `sklearn.pipeline` para encadenar varias transformaciones y un estimador final.
2. **Creación del modelo de regresión polinomial**:
   - Se instancia un objeto `Pipeline` llamado `regr_pol` que consta de dos pasos:
     - `poly`: `PolynomialFeatures` con grado 2 para generar características polinomiales.
     - `linear`: `LinearRegression` sin intercepto (`fit_intercept=False`) como el modelo final.
   - Se ajusta (`fit`) el modelo polinomial utilizando `X_boston` y `y_boston`.
3. **Evaluación del modelo**:
   - Se imprime el puntaje R^2 del modelo de regresión polinomial utilizando `regr_pol.score(X_boston, y_boston)`. El puntaje R^2 es una medida de la proporción de la varianza en la variable dependiente que es predecible a partir de las variables independientes.
   - Se calcula el error cuadrático medio (MSE) del modelo utilizando `np.mean((regr_pol.predict(X_boston) - y_boston)**2)`.

Este proceso evalúa la capacidad predictiva del modelo de regresión polinomial (grado 2) en el conjunto de datos de Boston Housing, proporcionando métricas que indican cómo se ajusta el modelo a los datos observados.

[ ]:
```
# Evaluation of the polynomial model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

regr_pol = Pipeline([('poly', PolynomialFeatures(degree=2)),('linear',␣
 ↪LinearRegression(fit_intercept=False))])
regr_pol.fit(X_boston, y_boston)

#print('Coeff and intercept: {} {}'.format(regr_pol.named_steps['linear'].
 ↪coef_, regr_pol.named_steps['linear'].intercept_))
print('Puntaje de la regresión polinomial múltiple: {}'.format(regr_pol.
 ↪score(X_boston, y_boston)))
```

```
print('\nMSE de la regresión polinomial múltiple: {}'.format(np.mean((regr_pol.
↪predict(X_boston) - y_boston)**2)))
```

Para la regresión simple, primero necesitamos extraer una de las características y luego usar los mismos métodos:

[ ]:
```
# Evaluación cuantitativa de la regresión lineal SIMPLE y polinomial:
bostonDF = pd.DataFrame(boston.data)
bostonDF.head()
```

[ ]:
```
bostonDF.columns=boston.feature_names
bostonDF.head()
```

5.6 Evaluación de Modelos de Regresión Lineal y Polinomial Simple en el Boston Housing Dataset

En este bloque de código se evalúan modelos de regresión lineal y polinomial simple en el conjunto de datos de Boston Housing utilizando la característica 'LSTAT'. Aquí está el desglose de lo que se hace:

1. **Preparación de Datos**:
   - Se selecciona la característica 'LSTAT' del DataFrame `bostonDF` como `x`.
   - Se selecciona la variable objetivo `boston.target` como `y`.
   - Ambos, `x` e `y`, se expanden a una dimensión adicional utilizando `np.expand_dims` para adecuar su forma requerida por `LinearRegression` y `PolynomialFeatures`.
2. **Modelo de Regresión Lineal Simple**:
   - Se instancia un objeto `LinearRegression` como `regr_boston`.
   - Se ajusta el modelo utilizando `x` e `y`.
   - Se imprime el puntaje R^2 del modelo utilizando `regr_boston.score(x, y)`.
   - Se calcula el error cuadrático medio (MSE) del modelo utilizando `np.mean((regr_boston.predict(x) - y)**2)`.
3. **Modelos de Regresión Polinomial Simple**:
   - Se instancia un objeto `Pipeline` llamado `regr_pol` con dos pasos:
     - `poly`: `PolynomialFeatures` con grado 2 para generar características polinomiales.
     - `linear`: `LinearRegression` sin intercepto (`fit_intercept=False`) como el modelo final.
   - Se ajusta el modelo utilizando `x` e `y`.
   - Se imprime el puntaje R^2 del modelo utilizando `regr_pol.score(x, y)`.
   - Se calcula el error cuadrático medio (MSE) del modelo utilizando `np.mean((regr_pol.predict(x) - y)**2)`.
4. **Modelo de Regresión Polinomial Simple (Orden 3)**:
   - Similar al modelo anterior pero con `PolynomialFeatures` de grado 3.
   - Se imprime el puntaje R^2 del modelo utilizando `regr_pol.score(x, y)`.
   - Se calcula el error cuadrático medio (MSE) del modelo utilizando `np.mean((regr_pol.predict(x) - y)**2)`.

Este proceso evalúa la capacidad predictiva de varios modelos de regresión lineal y polinomial simples en la característica 'LSTAT' del conjunto de datos de Boston Housing, proporcionando métricas que indican cómo se ajustan los modelos a los datos observados.

A continuación, se muestra el código utilizado para realizar la evaluación de los modelos:

[ ]:
```
x = bostonDF['LSTAT']
y = boston.target
x = np.expand_dims(x, axis=1)
y = np.expand_dims(y, axis=1)
```

[ ]:
```
regr_boston = LinearRegression()
regr_boston.fit(x, y)

print('Puntuación de la regresión lineal simple: {}'.format(regr_boston.
 ↪score(x, y)))
print('MSE de la regresión lineal simple: {}'.format(np.mean((regr_boston.
 ↪predict(x) - y)**2)))

regr_pol = Pipeline([('poly', PolynomialFeatures(degree=2)),('linear',␣
 ↪LinearRegression(fit_intercept=False))])
regr_pol.fit(x, y)

print('\nPuntuación de la regresión polinomial simple (orden 2): {}'.
 ↪format(regr_pol.score(x, y)))
print('MSE de la regresión polinomial simple (orden 2): {}'.format(np.
 ↪mean((regr_pol.predict(x) - y)**2)))

regr_pol = Pipeline([('poly', PolynomialFeatures(degree=3)),('linear',␣
 ↪LinearRegression(fit_intercept=False))])
regr_pol.fit(x, y)

print('\nPuntuación de la regresión polinomial simple (orden 3): {}'.
 ↪format(regr_pol.score(x, y)))
print('MSE de la regresión polinomial simple (orden 3): {}'.format(np.
 ↪mean((regr_pol.predict(x) - y)**2)))
```

5.7 Interpretación de los Resultados

5.7.1 Regresión Lineal Simple:

- **Puntuación (R^2)**: 0.5441
  - La regresión lineal simple explica aproximadamente el 54.41% de la variabilidad en la variable objetivo utilizando la característica 'LSTAT'.
  - Esto sugiere que el modelo lineal explica moderadamente bien la relación entre 'LSTAT' y la variable objetivo.
- **Error Cuadrático Medio (MSE)**: 38.483
  - El MSE de aproximadamente 38.483 indica que, en promedio, las predicciones del modelo lineal están desviadas por esta cantidad cuadrada.

5.7.2 Regresión Polinomial Simple (Orden 2):

- **Puntuación (R^2)**: 0.6407
  - La regresión polinomial de segundo orden explica aproximadamente el 64.07% de la variabilidad en la variable objetivo utilizando 'LSTAT' y sus términos polinomiales de grado 2.
  - Comparado con la regresión lineal simple, este modelo explica más variabilidad en los datos.
- **Error Cuadrático Medio (MSE)**: 30.331
  - El MSE de aproximadamente 30.331 es menor que el de la regresión lineal simple, lo que indica que las predicciones del modelo polinomial de segundo orden tienen menos error en promedio que las del modelo lineal simple.

5.7.3 Regresión Polinomial Simple (Orden 3):

- **Puntuación (R^2)**: 0.6578
  - La regresión polinomial de tercer orden explica aproximadamente el 65.78% de la variabilidad en la variable objetivo utilizando 'LSTAT' y sus términos polinomiales de grado 3.
  - Este modelo presenta una mejora adicional en la explicación de la variabilidad en comparación con el modelo de segundo orden.
- **Error Cuadrático Medio (MSE)**: 28.884
  - El MSE de aproximadamente 28.884 es ligeramente menor que el del modelo de segundo orden, indicando que las predicciones del modelo polinomial de tercer orden tienen un error promedio aún menor.

En resumen, los modelos polinomiales de orden 2 y 3 muestran una mejora significativa tanto en la capacidad explicativa (R^2 más alto) como en la precisión de las predicciones (MSE más bajo) en comparación con el modelo lineal simple. Esto sugiere que la relación entre la característica 'LSTAT' y la variable objetivo es mejor capturada por modelos polinomiales que por un modelo lineal simple.

EJERCICIO 3: Conjunto de Datos Macroeconómicos

En este ejercicio, trabajamos con el conjunto de datos Longley, que contiene datos macroeconómicos de EE. UU. desde 1947 hasta 1962. Utilizamos este conjunto para ajustar un modelo de regresión lineal múltiple y evaluar su rendimiento.

A continuación, se detalla el proceso seguido con los datos:

1. **Carga de Datos**:
   - Se lee el conjunto de datos desde la URL proporcionada utilizando `pd.read_csv`.
   - Se visualizan las primeras filas del DataFrame para asegurarnos de que los datos se han cargado correctamente.
2. **Limpieza de Nombres de Columnas**:
   - Se renombraron las columnas del DataFrame para tener nombres más descriptivos y claros.
   - Se definen las características (`features`) y el objetivo (`target`) del análisis.
3. **Preparación de los Datos**:
   - Se crea la matriz `X` con las características y el vector `y` con el objetivo a predecir, a partir del DataFrame cargado.

4. **Modelo de Regresión Lineal Múltiple**:
   - Se instancia un objeto `LinearRegression` como `lin_reg` para realizar la regresión lineal.
   - Se ajusta el modelo utilizando los datos de entrenamiento (`X` e `y`).
5. **Evaluación del Modelo**:
   - Se realizan predicciones (`predict`) utilizando el modelo ajustado.
   - Se calcula el Error Cuadrático Medio (ECM) y el Coeficiente de Determinación (R^2) para evaluar el rendimiento del modelo.
   - Se imprimen el intercepto y los coeficientes obtenidos del modelo.
   - Se muestran el ECM y el R^2 Score como métricas de evaluación del modelo.

Este ejercicio permite explorar cómo ajustar un modelo de regresión lineal múltiple a datos macroeconómicos utilizando Python y pandas, evaluando su rendimiento con métricas estándar como el ECM y el R^2 Score.

A continuación, se presenta el código utilizado para llevar a cabo este ejercicio:

[ ]:
```
# Leer los datos
df = pd.read_csv('http://vincentarelbundock.github.io/Rdatasets/csv/datasets/
 ↪longley.csv', index_col=0)
df.head()

# Limpiar nombres de columnas
df.columns = ['Deflactor_GNP', 'GNP', 'Desempleados', 'FuerzasArmadas',␣
 ↪'Poblacion', 'Anio', 'Empleados']
features = ['Deflactor_GNP', 'Desempleados', 'FuerzasArmadas', 'Poblacion',␣
 ↪'Anio', 'Empleados']
target = 'GNP'

# Crear la matriz X y el vector y a partir del dataset
X = df[features].values
y = df[target].values

print('Forma de los datos: {} {}'.format(X.shape, y.shape))
```

[ ]:
```
# Ajustando un modelo de regresión lineal múltiple
lin_reg = LinearRegression()  # Crear el estimador de Regresión Lineal
lin_reg.fit(X, y)  # Realizar el ajuste

# Coeficientes de la regresión
coefs = pd.Series(lin_reg.coef_, features).sort_values()

# Predicción
y_pred = lin_reg.predict(X)

# Evaluación
mse = mean_squared_error(y, y_pred)
r2score = lin_reg.score(X, y)
```

```
# Los coeficientes
print('\nIntercepto y coeficientes:\n{} {}'.format(lin_reg.intercept_, lin_reg.
 ↪coef_))
# Error cuadrático medio
print('\nECM: {}'.format(mse))
# Coeficiente de determinación: 1 es una predicción perfecta
print('R^2 Score: {}'.format(r2score))
```

5.8 Interpretación de los Resultados

5.8.1 Intercepto y Coeficientes:

El modelo de regresión lineal múltiple ajustado tiene un intercepto de aproximadamente -30213.91 y los siguientes coeficientes para cada una de las características:

- Deflactor_GNP: 1.5083
- Desempleados: -0.1859
- FuerzasArmadas: -0.0591
- Poblacion: 4.8293
- Anio: 15.4319
- Empleados: -3.1482

Estos coeficientes indican que, manteniendo todas las demás variables constantes, un aumento unitario en cada una de las características se asocia con un aumento o disminución en el GNP (Producto Nacional Bruto) en las cantidades indicadas por los coeficientes respectivos.

5.8.2 Error Cuadrático Medio (ECM):

El Error Cuadrático Medio obtenido en las predicciones del modelo es de aproximadamente 4.5946. Este valor representa el promedio de los errores al cuadrado entre las predicciones del modelo y los valores reales de GNP en el conjunto de datos.

5.8.3 Coeficiente de Determinación (R^2 Score):

El coeficiente de determinación, o R^2 Score, del modelo es aproximadamente 0.9995. Este valor indica que el modelo explica casi el 99.95% de la variabilidad de la variable objetivo (GNP) utilizando las características proporcionadas. Un valor cercano a 1.0 indica que el modelo ajustado se ajusta muy bien a los datos observados.

En resumen, el modelo de regresión lineal múltiple ajustado es extremadamente preciso, con un R^2 muy cercano a 1.0 y un ECM bajo. Esto sugiere que las características seleccionadas tienen una relación fuerte y lineal con el GNP, lo que permite realizar predicciones muy precisas.

[ ]:
```
plt.figure(figsize=(10, 6))
np.abs(coefs).sort_values().plot(kind='bar', title='Coeficientes del Modelo␣
 ↪(Valor Absoluto)')
plt.xlabel('Características')
plt.ylabel('Valor Absoluto del Coeficiente')
plt.show()
```

5.9 Interpretación de un R^2 alto en un Modelo de Regresión

Cuando el coeficiente de determinación R^2 es cercano a 1.0, indica que el modelo de regresión explica una gran parte de la variabilidad presente en los datos observados. Sin embargo, no se puede concluir automáticamente que existe un problema de sobreajuste basándose únicamente en un R^2 alto. Aquí se presentan algunos puntos clave a considerar:

- **Contexto del problema:** Es esencial entender el contexto y la naturaleza específica de los datos. En algunos casos, un R^2 alto puede ser completamente adecuado si el modelo está diseñado para capturar la variabilidad observada de manera precisa.
- **Validación del modelo:** Para evaluar la posibilidad de sobreajuste, es fundamental realizar una validación del modelo utilizando conjuntos de datos separados para entrenamiento y prueba. Un R^2 significativamente más bajo en el conjunto de prueba en comparación con el conjunto de entrenamiento puede indicar sobreajuste. Sin embargo, un alto R^2 en ambos conjuntos puede sugerir que el modelo generaliza bien.
- **Comparación con modelos alternativos:** Comparar el rendimiento del modelo actual con modelos más simples o más complejos puede proporcionar una perspectiva útil sobre si el modelo actual está sobreajustando los datos.
- **Inspección de los residuos:** Examinar los residuos (diferencias entre los valores predichos y observados) puede ser informativo. Patrones sistemáticos en los residuos o grandes fluctuaciones en relación con los valores predichos podrían indicar que el modelo no está capturando ciertos aspectos de los datos de manera adecuada.

En resumen, un R^2 cercano a 1.0 indica que el modelo explica bien la variabilidad de los datos, pero no es suficiente para determinar si existe sobreajuste. Es necesario considerar otros factores, como la validación del modelo y la inspección de los residuos, para evaluar correctamente si el modelo está sobreajustado a los datos.

5.9.1 Análisis de los residuos

En el gráfico de residuos, si el modelo está bien ajustado, se espera ver los residuos distribuidos aleatoriamente alrededor de la línea horizontal en $y = 0$. Esto indicaría que no hay patrones sistemáticos en los errores de predicción y que el modelo no está sobreajustando los datos. Si los residuos muestran una tendencia o patrón, podría ser indicativo de que el modelo no está capturando correctamente alguna relación en los datos.

[ ]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=y - y_pred, alpha=0.7)
plt.hlines(y=0, xmin=0, xmax=y_pred.max())
plt.title('Gráfico de residuos')
plt.xlabel('$\hat y$')
plt.ylabel('$y - \hat y$')
```

5.9.2 Gráfico de predicciones vs. valores reales

En el gráfico de predicciones frente a valores reales, una alineación perfecta de los puntos con la línea diagonal (representando el caso ideal donde $\hat{y} = y$) sugiere un modelo perfectamente ajustado.

Si los puntos se dispersan significativamente lejos de la línea, indica que las predicciones del modelo tienen errores notables, lo que implica que el modelo podría no ser tan preciso.

[ ]:
```
plt.figure(figsize=(10, 6))
plt.scatter(y, y_pred, alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--k')
plt.axis('tight')
plt.xlabel('$y$')
plt.ylabel('$\hat y$')
```