SESIÓN REGRESIONES

CONTENIDOS:

- Problemas de regresión.
- Tipos de problemas de regresión: lineales. No lineales.
- Librería scikit-learn para modelos de regresión.
- Entrenamiento de algoritmos de regresión.
- Predicciones con el modelo de regresión.
- Selección del modelo de regresión.

PROBLEMAS DE REGRESIÓN

Los problemas de regresión son una categoría fundamental en el campo del **aprendizaje automático (Machine Learning)**, donde el objetivo principal es predecir un valor continuo (una cantidad numérica) a partir de una o más variables de entrada (también conocidas como características o predictores). A diferencia de los problemas de **clasificación**, donde se busca predecir una etiqueta discreta (como "sí" o "no", "perro" o "gato"), en la regresión se trabaja con valores numéricos que pueden tomar cualquier valor dentro de un rango continuo.

Características de los problemas de regresión:

- **Variable dependiente (objetivo)**: Es la variable que se desea predecir. Es continua, lo que significa que puede tomar cualquier valor dentro de un rango (por ejemplo, el precio de una casa, la temperatura, el tiempo de entrega, etc.).
- **Variables independientes (características)**: Son las variables que se utilizan para predecir la variable dependiente. Pueden ser una o varias, y pueden ser tanto numéricas como categóricas (aunque las categóricas suelen codificarse numéricamente).
- **Relación entre variables**: En los problemas de regresión, se busca modelar la relación entre las variables independientes y la variable dependiente. Esta relación puede ser lineal o no lineal, dependiendo de la naturaleza del problema.

Ejemplos de problemas de regresión:

1. **Predicción del precio de una casa**:
    - **Variables de entrada**: Tamaño de la casa, número de habitaciones, ubicación, antigüedad, etc.
    - **Variable de salida**: Precio de la casa.
    - **Aplicación**: Este tipo de modelo es útil para empresas inmobiliarias o compradores que desean estimar el valor de una propiedad.
2. **Estimación del consumo de combustible de un vehículo**:
    - **Variables de entrada**: Peso del vehículo, potencia del motor, tipo de combustible, condiciones de conducción, etc.
    - **Variable de salida**: Consumo de combustible (litros por kilómetro).
    - **Aplicación**: Este modelo puede ser utilizado por fabricantes de automóviles para optimizar el diseño de vehículos más eficientes.
3. **Predicción de ventas de un producto**:
    - **Variables de entrada**: Inversión en publicidad, precio del producto, temporada del año, etc.
    - **Variable de salida**: Cantidad de unidades vendidas.
    - **Aplicación**: Las empresas pueden utilizar este modelo para planificar estrategias de marketing y optimizar sus campañas publicitarias.

¿Por qué es importante la regresión?

1. **Predicciones cuantitativas**: La regresión permite hacer predicciones numéricas basadas en datos históricos. Esto es esencial en muchas áreas donde se necesita estimar valores futuros o desconocidos.
2. **Toma de decisiones basada en datos**: Al proporcionar estimaciones precisas, los modelos de regresión ayudan a las organizaciones a tomar decisiones informadas. Por ejemplo, una empresa puede decidir cuánto invertir en publicidad basándose en las predicciones de ventas.

3. **Aplicaciones en diversas áreas**:
    - **Finanzas**: Predicción de precios de acciones, tasas de interés, etc.
    - **Economía**: Estimación del PIB, tasas de desempleo, etc.
    - **Ingeniería**: Predicción de fallos en maquinaria, estimación de vida útil de componentes, etc.
    - **Ciencias sociales**: Predicción de comportamientos humanos, como la tasa de criminalidad en función de factores socioeconómicos.
4. **Optimización de recursos**: Los modelos de regresión pueden ayudar a optimizar recursos al predecir demandas futuras, costos o rendimientos. Por ejemplo, en la logística, se puede predecir el tiempo de entrega de mercancías para optimizar rutas y reducir costos.

TIPOS DE PROBLEMAS DE REGRESIÓN: LINEALES Y NO LINEALES

Los problemas de regresión se pueden clasificar en dos categorías principales según la naturaleza de la relación entre las variables de entrada y la variable de salida: **regresión lineal** y **regresión no lineal**.

Regresión Lineal

La **regresión lineal** es uno de los modelos más simples y ampliamente utilizados en Machine Learning. Se basa en la suposición de que existe una **relación lineal** entre las variables de entrada (características) y la variable de salida (objetivo). Esto significa que la relación entre las variables puede ser representada por una **línea recta** en un espacio de dos dimensiones (en el caso de una sola variable de entrada) o por un **hiperplano** en espacios de mayor dimensión (cuando hay múltiples variables de entrada).

Fórmula de la regresión lineal simple:

$$Y = \beta_0 + \beta_1 x + \epsilon$$

- ***y***: Variable dependiente (objetivo). Es el valor que se desea predecir.
- ***x***: Variable independiente (característica). Es la variable que se utiliza para predecir *y*.

- $\boldsymbol{\beta_0}$: Intercepto. Es el valor de *y* cuando x = 0. Representa el punto donde la línea de regresión cruza el eje *y*.
- $\boldsymbol{\beta_1}$: Pendiente. Representa el cambio en *y* por cada unidad de cambio en *x*. Indica la dirección y la inclinación de la línea de regresión.
- $\boldsymbol{\epsilon}$: Error. Representa la diferencia entre el valor real y el valor predicho por el modelo. Este término captura el ruido o la variabilidad que no puede ser explicada por el modelo.

Ejemplo de regresión lineal simple:

- **Problema**: Predecir el precio de una casa en función de su tamaño.
- **Variable de entrada (*x*)**: Tamaño de la casa (en metros cuadrados).
- **Variable de salida (*y*)**: Precio de la casa (en dólares).
- **Modelo**: El modelo de regresión lineal ajustará una línea recta que mejor represente la relación entre el tamaño de la casa y su precio.

Ventajas de la regresión lineal:

- **Simplicidad**: Es fácil de entender e implementar.
- **Interpretabilidad**: Los coeficientes β_0 y β_1 tienen una interpretación clara.
- **Eficiencia**: Es computacionalmente eficiente, incluso con grandes conjuntos de datos.

Limitaciones de la regresión lineal:

- **Supuesto de linealidad**: Solo puede modelar relaciones lineales. Si la relación entre las variables no es lineal, el modelo no será adecuado.
- **Sensibilidad a valores atípicos**: Los valores atípicos pueden afectar significativamente la línea de regresión.

Regresión No Lineal

La **regresión no lineal** se utiliza cuando la relación entre las variables de entrada y la variable de salida no es lineal. En estos casos, se necesitan modelos más complejos que puedan capturar relaciones curvilíneas o patrones más intrincados. Algunos ejemplos de modelos no lineales incluyen **regresión polinómica**, **regresión exponencial**, **regresión logarítmica** y **modelos basados en redes neuronales**.

Ejemplo de regresión no lineal:

- **Problema**: Predecir el crecimiento de una población en función del tiempo.
- **Variable de entrada (*x*)**: Tiempo (en años).
- **Variable de salida (*y*)**: Tamaño de la población.
- **Modelo**: En este caso, la relación entre el tiempo y el tamaño de la población puede ser exponencial, por lo que un modelo lineal no sería adecuado. En su lugar, se podría utilizar un modelo exponencial como $y = \beta_0 e^{\beta 1 x}$.

Diferencias clave entre regresión lineal y no lineal:

1. **Forma de la relación**:
    - **Lineal**: La relación entre las variables es una línea recta.
    - **No lineal**: La relación puede ser curva, exponencial, logarítmica, etc.
2. **Complejidad del modelo**:
    - **Lineal**: Modelos más simples y fáciles de interpretar.
    - **No lineal**: Modelos más complejos que pueden capturar relaciones más intrincadas.
3. **Aplicabilidad**:
    - **Lineal**: Adecuado para problemas donde la relación entre las variables es lineal o aproximadamente lineal.
    - **No lineal**: Adecuado para problemas donde la relación es claramente no lineal.

Ejemplos de modelos no lineales:

- **Regresión polinómica**: Modela la relación entre las variables como un polinomio de grado n. Por ejemplo, $y = \beta_0+\beta_1 x+\beta_2 x^2$.
- **Regresión exponencial**: Modela la relación como una función exponencial. Por ejemplo, $y = \beta_0 e^{\beta 1 x}$.
- **Regresión logarítmica**: Modela la relación como una función logarítmica. Por ejemplo, $y = \beta_0 +\beta_1 \ln(x)$.

Ventajas de la regresión no lineal:

- **Flexibilidad**: Puede modelar relaciones complejas y no lineales.
- **Precisión**: En muchos casos, puede proporcionar predicciones más precisas que la regresión lineal.

Limitaciones de la regresión no lineal:

- **Complejidad**: Los modelos no lineales son más difíciles de interpretar y ajustar.
- **Riesgo de sobreajuste**: Los modelos no lineales pueden sobreajustarse a los datos de entrenamiento, lo que reduce su capacidad de generalización.

LIBRERÍA SCIKIT-LEARN PARA MODELOS DE REGRESIÓN

Scikit-learn es una de las librerías más populares y ampliamente utilizadas en Python para el desarrollo de modelos de **Machine Learning**, incluyendo modelos de regresión. Es una herramienta poderosa que ofrece una amplia gama de algoritmos y funcionalidades para implementar, entrenar y evaluar modelos de regresión de manera eficiente. Scikit-learn es especialmente valorada por su facilidad de uso, su integración con otras librerías de Python (como NumPy, Pandas y Matplotlib), y su extensa documentación.

Funcionalidades principales de Scikit-learn para regresión:

1. **Regresión Lineal (LinearRegression)**:
    - **Descripción**: Es el modelo más básico y común para problemas de regresión lineal. Ajusta una línea recta (o un hiperplano en múltiples dimensiones) a los datos de entrenamiento.
    - **Uso**: Ideal para problemas donde la relación entre las variables de entrada y la variable de salida es lineal.

      Ejemplo de uso:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Ilustración 1 Regresión lineal.

2. **Regresión Polinómica (PolynomialFeatures + LinearRegression)**:
    - **Descripción**: Permite modelar relaciones no lineales entre las variables de entrada y la variable de salida. Primero, se transforman las características en un polinomio de grado n*n*, y luego se aplica una regresión lineal sobre estas características transformadas.
    - **Uso**: Útil cuando la relación entre las variables no es lineal, pero puede ser aproximada por un polinomio.

      Ejemplo de uso:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_train)
model = LinearRegression()
model.fit(X_poly, y_train)
y_pred = model.predict(poly.transform(X_test))
```

Ilustración 2 Regresión polinómica.

3. **Regresión Ridge y Lasso (Regularización)**:
    - **Descripción**: Son variantes de la regresión lineal que incluyen términos de regularización para evitar el sobreajuste (overfitting).
        - **Ridge Regression**: Añade una penalización L2$L2$ (suma de los cuadrados de los coeficientes) a la función de costo.
        - **Lasso Regression**: Añade una penalización L1$L1$ (suma de los valores absolutos de los coeficientes), lo que puede llevar a la eliminación de algunas características (feature selection).
    - **Uso**: Útil cuando hay multicolinealidad (correlación alta entre las variables de entrada) o cuando se desea reducir la complejidad del modelo.

      Ejemplo de uso:

```
from sklearn.linear_model import Ridge, Lasso

model_ridge = Ridge(alpha=1.0)
model_lasso = Lasso(alpha=1.0)
model_ridge.fit(X_train, y_train)
model_lasso.fit(X_train, y_train)
```

Ilustración 3 Regresión Ridge y Lasso.

4. **Regresión de Soporte Vectorial (SVR - Support Vector Regression)**:
    - **Descripción**: Es una extensión de las Máquinas de Soporte Vectorial (SVM) para problemas de regresión. SVR busca encontrar una función que se desvíe lo menos posible de los valores reales, mientras mantiene un margen de tolerancia.
    - **Uso**: Adecuado para problemas de regresión no lineal, especialmente cuando los datos tienen una estructura compleja.

      Ejemplo de uso:

```
from sklearn.svm import SVR

model = SVR(kernel='rbf')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Ilustración 4 Regresión de soporte vectorial.

Ejemplo completo de implementación de regresión lineal con Scikit-learn:

```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear y entrenar el modelo
model = LinearRegression()
model.fit(X_train, y_train)

# Predecir
y_pred = model.predict(X_test)

# Evaluar el modelo
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Error cuadrático medio (MSE): {mse}")
print(f"Coeficiente de determinación (R²): {r2}")
```

Ilustración 5 Implementación completa regresión lineal.

ENTRENAMIENTO DE ALGORITMOS DE REGRESIÓN

El entrenamiento de un modelo de regresión es el proceso mediante el cual se ajustan los parámetros del modelo para minimizar el error entre las predicciones y los valores reales. Este proceso es fundamental para garantizar que el modelo pueda generalizar bien a nuevos datos y hacer predicciones precisas.

Pasos para entrenar un modelo de regresión:

1. **Preprocesamiento de datos**:
    - **Limpieza de datos**: Eliminar valores nulos, corregir errores y manejar valores atípicos.
    - **Normalización/Escalado**: Asegurar que todas las características estén en la misma escala, especialmente importante para modelos sensibles a la magnitud de los datos (como regresión lineal o SVM).
    - **Transformación de datos**: Aplicar transformaciones como logarítmicas o polinómicas si es necesario.

- **Codificación de variables categóricas**: Convertir variables categóricas en numéricas mediante técnicas como **One-Hot Encoding** o **Label Encoding**.

2. **División de datos**:
   - **Conjunto de entrenamiento**: Se utiliza para ajustar los parámetros del modelo.
   - **Conjunto de prueba**: Se utiliza para evaluar el rendimiento del modelo en datos no vistos.
   - **Validación cruzada**: Técnica que divide los datos en múltiples subconjuntos para evaluar el modelo de manera más robusta.
3. **Selección del modelo**:
   - Elegir el tipo de regresión adecuado según la naturaleza del problema (lineal, no lineal, regularizado, etc.).
   - Considerar la complejidad del modelo y el riesgo de sobreajuste.
4. **Entrenamiento del modelo**:
   - Ajustar los parámetros del modelo utilizando el conjunto de entrenamiento.
   - Minimizar una función de costo (como el Error Cuadrático Medio) para encontrar los mejores parámetros.
5. **Validación del modelo**:
   - Evaluar el rendimiento del modelo utilizando el conjunto de prueba.
   - Utilizar métricas como el **Error Cuadrático Medio (MSE)** y el **Coeficiente de Determinación (R^2)** para medir la precisión del modelo.

Métricas comunes para evaluar modelos de regresión:

1. **Error Cuadrático Medio (MSE)**:
   - **Definición**: Mide el promedio de los errores al cuadrado entre los valores reales y los predichos.

- **Fórmula**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Ilustración 6 Fórmula MSE.

- **Interpretación**: Un MSE más bajo indica un mejor ajuste del modelo. Sin embargo, es sensible a valores atípicos.

2. **Coeficiente de Determinación (R^2)**:
   - **Definición**: Indica la proporción de la varianza en la variable dependiente que es explicada por el modelo.
   - **Fórmula**:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Ilustración 7 Fórmula R^2.

   - **Interpretación**: Un valor de $R2$ cercano a 1 indica que el modelo explica una gran parte de la varianza, mientras que un valor cercano a 0 indica un mal ajuste.

3. **Error Absoluto Medio (MAE)**:
   - **Definición**: Mide el promedio de los errores absolutos entre los valores reales y los predichos.
   - **Fórmula**:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Ilustración 8 Fórmula MAE.

   - **Interpretación**: Es menos sensible a valores atípicos que el MSE.

PREDICCIONES CON EL MODELO DE REGRESIÓN

Una vez que un modelo de regresión ha sido entrenado, su principal función es hacer **predicciones** sobre nuevos datos. Estas predicciones son valores continuos que representan la estimación del modelo basada en las variables de entrada (características). Este proceso es fundamental en aplicaciones prácticas, ya que permite utilizar el modelo para tomar decisiones informadas o realizar estimaciones en tiempo real.

Proceso de predicción:

1. **Entrada de nuevos datos**:
    - Los nuevos datos deben tener el mismo formato que los datos utilizados para entrenar el modelo. Es decir, deben incluir las mismas características (variables de entrada) en el mismo orden.
    - **Ejemplo**: Si el modelo fue entrenado con dos características (por ejemplo, tamaño de la casa y número de habitaciones), los nuevos datos deben incluir estas dos características.
2. **Uso del modelo para predecir**:
    - El modelo utiliza las características de los nuevos datos para calcular la predicción. En el caso de la regresión lineal, esto implica multiplicar las características por los coeficientes aprendidos durante el entrenamiento y sumar el intercepto.
    - **Ejemplo en código**:

```
# Predecir con nuevos datos
nuevos_datos = [[100, 200]]  # Ejemplo de nuevas características
prediccion = model.predict(nuevos_datos)
print(f"Predicción: {prediccion}")
```

Ilustración 9 modelo para predecir.

3. **Interpretación de la predicción**:
    - La salida del modelo es un valor continuo que representa la estimación de la variable dependiente (objetivo).

- **Ejemplo**: Si el modelo predice un valor de 150,000 para el precio de una casa, esto significa que, según las características proporcionadas (por ejemplo, tamaño y número de habitaciones), el modelo estima que el precio de la casa es de 150,000 dólares.

Consideraciones importantes:

1. **Calidad del modelo**:
   - Las predicciones son tan buenas como el modelo y los datos utilizados para entrenarlo. Si el modelo no fue entrenado adecuadamente o los datos tienen problemas (como ruido o valores atípicos), las predicciones pueden ser inexactas.
   - **Validación del modelo**: Es crucial evaluar el modelo con datos no vistos (conjunto de prueba) para asegurar que generaliza bien a nuevos datos. Métricas como el **Error Cuadrático Medio (MSE)** y el **Coeficiente de Determinación (R^2)** son útiles para esta evaluación.
2. **Generalización**:
   - Un modelo bien entrenado debe ser capaz de generalizar, es decir, hacer predicciones precisas en datos que no ha visto antes. Si el modelo está sobreajustado (overfitting), puede funcionar muy bien en los datos de entrenamiento pero mal en datos nuevos.
   - **Técnicas para evitar el sobreajuste**: Regularización (Ridge, Lasso), validación cruzada y selección de características.
3. **Preprocesamiento de nuevos datos**:
   - Los nuevos datos deben ser preprocesados de la misma manera que los datos de entrenamiento. Esto incluye normalización, escalado y codificación de variables categóricas.
   - **Ejemplo**: Si se utilizó StandardScaler para normalizar los datos de entrenamiento, se debe aplicar el mismo escalador a los nuevos datos antes de hacer la predicción.

4. **Interpretabilidad**:
    - En algunos casos, es importante no solo obtener la predicción, sino también entender cómo el modelo llegó a esa predicción. Esto es especialmente relevante en aplicaciones donde la transparencia es crucial, como en finanzas o medicina.

SELECCIÓN DEL MODELO DE REGRESIÓN

La **selección del modelo de regresión** es un paso crítico en el proceso de Machine Learning. Elegir el modelo adecuado puede marcar la diferencia entre un modelo que hace predicciones precisas y uno que no es útil. La selección del modelo depende de varios factores, incluyendo la naturaleza de los datos, la relación entre las variables y el objetivo del análisis.

Factores a considerar al seleccionar un modelo de regresión:

1. **Linealidad**:
    - **Relación lineal**: Si la relación entre las variables de entrada y la variable de salida es lineal, un modelo de **regresión lineal** puede ser suficiente.
    - **Relación no lineal**: Si la relación es no lineal, se deben considerar modelos más complejos, como regresión polinómica, regresión de soporte vectorial (SVR) o redes neuronales.
2. **Complejidad del modelo**:
    - **Modelos simples**: Los modelos simples, como la regresión lineal, son fáciles de interpretar y menos propensos al sobreajuste, pero pueden no capturar relaciones complejas en los datos.
    - **Modelos complejos**: Los modelos más complejos, como las redes neuronales o los modelos de ensamble (por ejemplo, Random Forest o Gradient Boosting), pueden capturar relaciones no lineales y patrones más intrincados, pero tienen un mayor riesgo de sobreajuste y son más difíciles de interpretar.
3. **Regularización**:
    - **Ridge Regression**: Añade una penalización L2*L*2 a los coeficientes del modelo, lo que ayuda a reducir la magnitud de los coeficientes y evitar el sobreajuste.

- **Lasso Regression**: Añade una penalización L1$L1$, lo que no solo reduce la magnitud de los coeficientes, sino que también puede eliminar algunas características (feature selection), lo que resulta en un modelo más simple.
- **Elastic Net**: Combina las penalizaciones L1$L1$ y L2$L2$ de Ridge y Lasso, lo que permite un equilibrio entre ambas.

4. **Evaluación del modelo**:
   - **Métricas de evaluación**: Es importante utilizar métricas como el Error Cuadrático Medio (MSE), el Coeficiente de Determinación (R^2) y el Error Absoluto Medio (MAE) para comparar el rendimiento de diferentes modelos.
   - **Validación cruzada**: La validación cruzada (cross-validation) es una técnica que permite evaluar el rendimiento del modelo en múltiples subconjuntos de datos, lo que proporciona una estimación más robusta de su capacidad de generalización.
5. **Interpretabilidad**:
   - En algunos casos, es importante que el modelo sea interpretable, es decir, que se pueda entender cómo está haciendo las predicciones. Los modelos lineales son más interpretables que los modelos no lineales o basados en ensambles.
6. **Escalabilidad**:
   - Algunos modelos, como las redes neuronales, pueden ser computacionalmente costosos de entrenar, especialmente con grandes conjuntos de datos. Es importante considerar la escalabilidad del modelo en función del tamaño de los datos y los recursos disponibles.

ACTIVIDAD PRÁCTICA GUIADA: Modelos de Regresión con Scikit-Learn

En este ejercicio es comprenderemos y aplicaremos modelos de regresión para resolver problemas de predicción utilizando la librería Scikit-Learn. Abordaremos distintos tipos de regresión y compararemos los resultados para seleccionar el modelo más adecuado.

Paso 1: Importar librerías

Antes de comenzar, importemos las librerías necesarias:

```
# Paso 1: Importar librerías
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
```

- **numpy (np)**: Biblioteca para trabajar con arrays y funciones matemáticas.
- **matplotlib.pyplot (plt)**: Permite crear gráficos y visualizar datos.
- **pandas (pd)**: Se usa para manipulación y análisis de datos (aunque no se usó directamente aquí).
- **train_test_split**: Divide los datos en conjuntos de entrenamiento y prueba.
- **LinearRegression**: Implementa el modelo de regresión lineal.
- **PolynomialFeatures**: Transforma datos para aplicar regresión polinómica.
- **DecisionTreeRegressor**: Implementa un modelo de regresión basado en árboles de decisión.
- **mean_squared_error**: Calcula el error cuadrático medio (MSE) para evaluar modelos.

Paso 2: Generar datos sintéticos

Creamos un conjunto de datos que muestre una relación no lineal entre la variable independiente X y la variable dependiente y.

```
# Paso 2: Generar datos sintéticos
np.random.seed(42)
X = np.linspace(0, 10, 100).reshape(-1, 1)
y = 3 * X**2 + 2 * X + np.random.randn(100, 1) * 10

plt.scatter(X, y, color="blue", label="Datos reales")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

Salida:

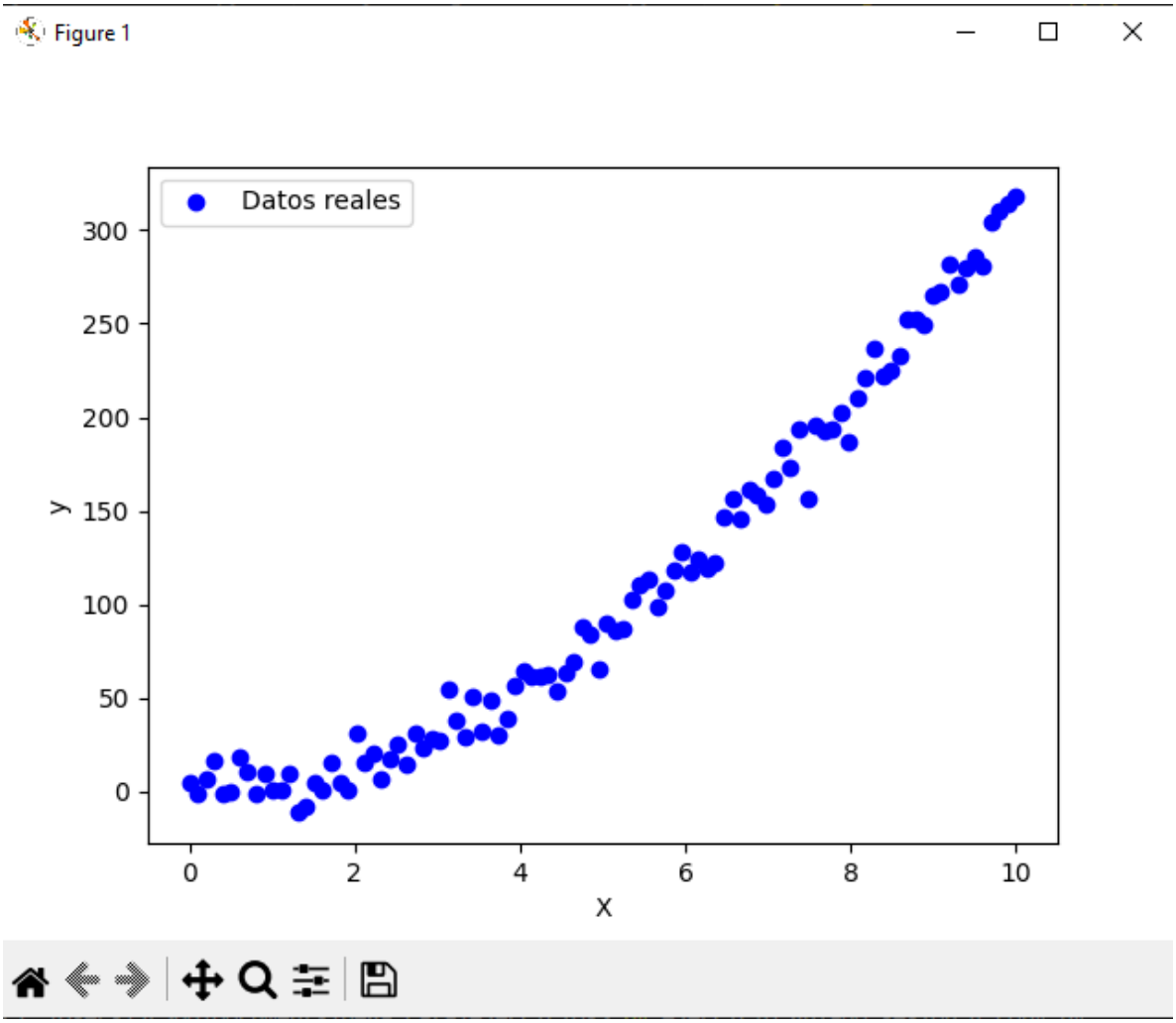

Ilustración 10 Gráfico "Datos reales"

- **np.random.seed(42):** Fija la semilla aleatoria para reproducibilidad.
- **np.linspace(0, 10, 100):** Crea 100 valores equidistantes entre 0 y 10.
- **.reshape(-1, 1):** Convierte el array en una matriz de una columna.
- **np.random.randn(100, 1) * 10:** Genera ruido aleatorio normal para simular datos reales.

Paso 3: División del conjunto de datos

Dividimos los datos en conjuntos de entrenamiento y prueba.

```
# Paso 3: División del conjunto de datos
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

- **train_test_split**: Divide los datos en 80% entrenamiento y 20% prueba.
- **test_size=0.2**: Define el 20% de los datos como conjunto de prueba.
- **random_state=42**: Asegura que la división sea reproducible.

Paso 4: Aplicación de diferentes modelos de regresión

Implementamos modelos de regresión

1. Regresión Lineal

```
# Paso 4: Aplicación de diferentes modelos de regresión
# Regresión Lineal
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred_lin = lin_reg.predict(X_test)
error_lin = mean_squared_error(y_test, y_pred_lin)
print(f"Error de Regresión Lineal: {error_lin}")
```

Salida:

```
Error de Regresión Lineal: 544.1736611263539
```

Ilustración 11 Salida Regresión lineal.

- **LinearRegression():** Crea el modelo de regresión lineal.
- **.fit(X_train, y_train):** Entrena el modelo con los datos de entrenamiento.
- **.predict(X_test):** Predice valores usando el conjunto de prueba.
- **mean_squared_error(y_test, y_pred_lin):** Calcula el error cuadrático medio del modelo.

2. Regresión Polinómica

```
# Paso 4: Aplicación de diferentes modelos de regresión
# Regresión Polinómica
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)

poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)
y_pred_poly = poly_reg.predict(X_test_poly)
error_poly = mean_squared_error(y_test, y_pred_poly)
print(f"Error de Regresión Polinómica: {error_poly}")
```

Salida:

```
Error de Regresión Polinómica: 61.963304503949914
```

Ilustración 12 Salida Regresión polinómica.

- **PolynomialFeatures(degree=2)**: Convierte X en características polinómicas de segundo grado.
- **.fit_transform(X_train)**: Ajusta y transforma los datos de entrenamiento.
- **.transform(X_test)**: Transforma los datos de prueba con la misma escala.
- **.fit(X_train_poly, y_train)**: Entrena la regresión lineal sobre las características polinómicas.
- **.predict(X_test_poly)**: Predice valores con el modelo polinómico.
- **mean_squared_error(y_test, y_pred_poly)**: Calcula el error cuadrático medio.

3. Árbol de Decisión

```
# Paso 4: Aplicación de diferentes modelos de regresión
# Árbol de Decisión
tree_reg = DecisionTreeRegressor()
tree_reg.fit(X_train, y_train)
y_pred_tree = tree_reg.predict(X_test)
error_tree = mean_squared_error(y_test, y_pred_tree)
print(f"Error de Regresión con Árbol de Decisión: {error_tree}")
```

Salida:

```
Error de Regresión con Árbol de Decisión: 135.2008472515694
```

Ilustración 13 Salida Regresión con Árbol de Decisión.

- **DecisionTreeRegressor()**: Crea un modelo basado en árboles de decisión.
- **.fit(X_train, y_train)**: Entrena el modelo con los datos de entrenamiento.
- **.predict(X_test)**: Predice valores con el modelo entrenado.
- **mean_squared_error(y_test, y_pred_tree)**: Calcula el error del modelo de árbol de decisión.

Paso 5: Comparación de modelos

```
# Paso 5: Comparación de modelos
plt.scatter(X_test, y_test, color="blue", label="Datos Reales")
plt.scatter(X_test, y_pred_lin, color="red", label="Regresión Lineal")
plt.scatter(X_test, y_pred_poly, color="green", label="Regresión
Polinómica")
plt.scatter(X_test, y_pred_tree, color="purple", label="Árbol de Decisión")
plt.legend()
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```

Salida:

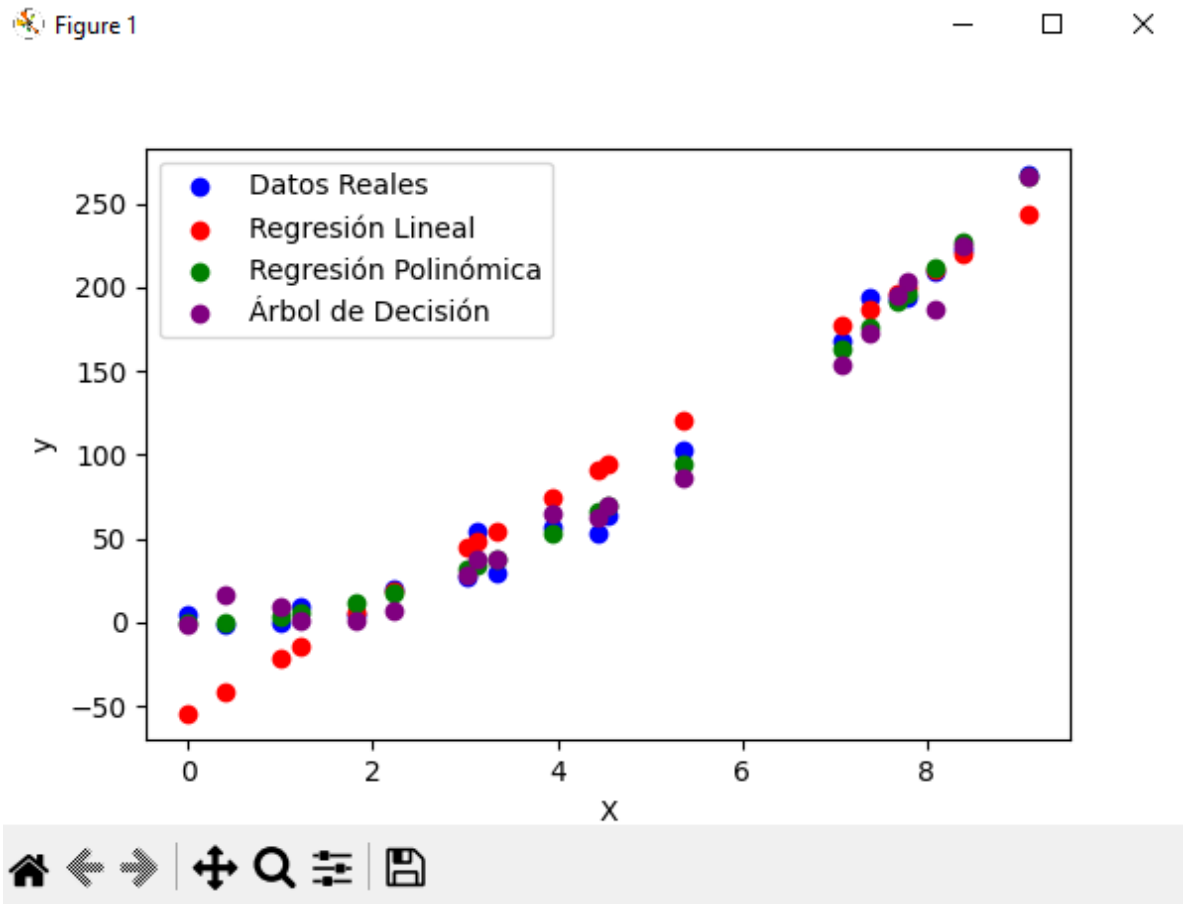

Ilustración 14 Gráfico comparación de modelos.

- **plt.scatter(X_test, y_test, color='blue', label='Datos Reales')**: Grafica los datos reales.
- **plt.scatter(X_test, y_pred_lin, color='red', label='Regresión Lineal')**: Grafica las predicciones de la regresión lineal.
- **plt.scatter(X_test, y_pred_poly, color='green', label='Regresión Polinómica')**: Grafica la regresión polinómica.
- **plt.scatter(X_test, y_pred_tree, color='purple', label='Árbol de Decisión')**: Grafica la predicción del árbol de decisión.
- **plt.legend()**: Agrega la leyenda.
- **plt.xlabel('X'), plt.ylabel('y')**: Etiquetas para los ejes.
- **plt.show()**: Muestra el gráfico.

Paso 6: Selección del mejor modelo

Comparamos los errores y seleccionamos el modelo con menor error.

```
errores = {
    "Regresión Lineal": error_lin,
    "Regresión Polinómica": error_poly,
    "Árbol de Decisión": error_tree,
}
mejor_modelo = min(errores, key=errores.get)
print(f"El mejor modelo es: {mejor_modelo}")
```

Salida:

```
El mejor modelo es: Regresión Polinómica
```

Ilustracion 15 selección major modelo.

- **errores = { ... }**: Guarda los errores de cada modelo en un diccionario.
- **min(errores, key=errores.get)**: Encuentra el modelo con el menor error.
- **print(f'El mejor modelo es: {mejor_modelo}')**: Muestra el mejor modelo según el error cuadrático medio.