```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
```

En este bloque, se importan los módulos necesarios: numpy para operaciones matemáticas, LinearRegression de scikit-learn para realizar regresión lineal, PolynomialFeatures de scikit-learn para generar características polinómicas, y matplotlib para visualizar los resultados.

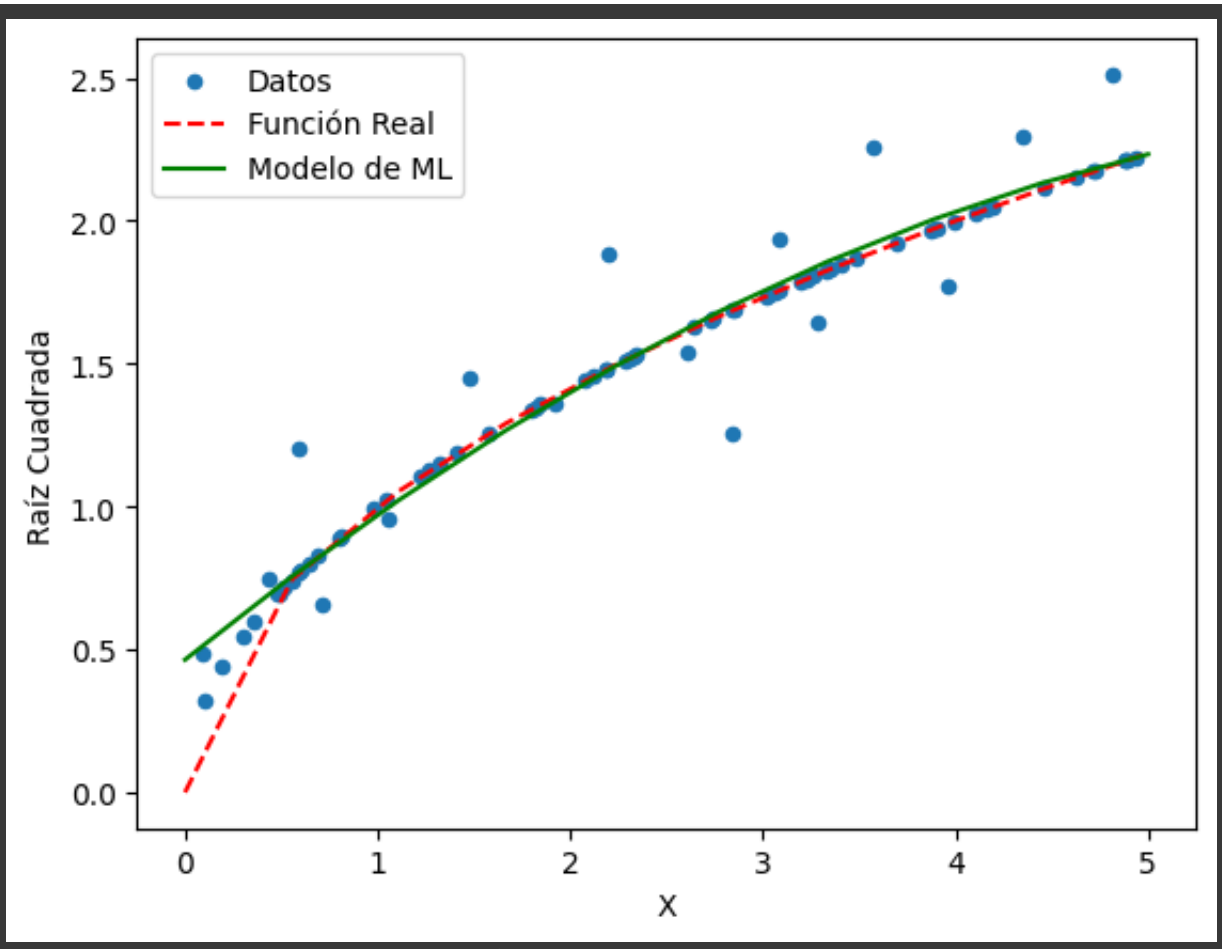
```
# Generar un conjunto de datos con valores aleatorios
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sqrt(X).ravel()

# Agregar ruido al conjunto de datos
y[::5] += 1 * (0.5 - np.random.rand(16))
```

Se genera un conjunto de datos X y y. X se genera aleatoriamente como 80 puntos en el rango de [0, 5], y y se calcula como la raíz cuadrada de X. Se agrega ruido a los datos de salida y en intervalos regulares. Esto simula la variabilidad en los datos reales.

```
# Entrenar un modelo de regresión polinómica
degree = 2  # Grado del polinomio
polyreg = PolynomialFeatures(degree=degree)
X_poly = polyreg.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, y)

# Crear un conjunto de datos de prueba
X_test = np.linspace(0, 5, 10).reshape(-1, 1)
X_test_poly = polyreg.transform(X_test)
```

Se entrena un modelo de regresión polinómica de grado 2. Se crean características polinómicas a partir de las características originales X. Luego, se ajusta el modelo de regresión lineal a las características polinómicas. Se crea un conjunto de datos de prueba X_test que contiene 10 puntos equidistantes en el rango de [0, 5]. Las características polinómicas se generan a partir de X_test.

```
# Predecir los valores de la raíz cuadrada utilizando el modelo
y_pred = model.predict(X_test_poly)

# Mostrar los resultados
plt.scatter(X, y, s=20, label="Datos")
plt.plot(X_test, np.sqrt(X_test), color='r', linestyle='--', label="Función Real")
plt.plot(X_test, y_pred, color='g', label="Modelo de ML")
plt.xlabel("X")
plt.ylabel("Raíz Cuadrada")
plt.legend()
plt.show()

# Comparar con la función nativa del lenguaje Python
true_values = np.sqrt(X_test)
for i in range(len(X_test)):
    print(f"Valor real: {true_values[i][0]}, Valor predicho: {y_pred[i]:.4f}")
```




```
Valor real: 0.0, Valor predicho: 0.4638
Valor real: 0.7453559924999299, Valor predicho: 0.7551
Valor real: 1.0540925533894598, Valor predicho: 1.0228
Valor real: 1.2909944487358056, Valor predicho: 1.2667
Valor real: 1.4907119849998598, Valor predicho: 1.4869
Valor real: 1.6666666666666667, Valor predicho: 1.6834
Valor real: 1.8257418583505538, Valor predicho: 1.8562
Valor real: 1.9720265943665387, Valor predicho: 2.0054
Valor real: 2.1081851067789197, Valor predicho: 2.1308
Valor real: 2.23606797749979, Valor predicho: 2.2325
```

Se utilizan el modelo entrenado y las características polinómicas de X_test para realizar predicciones en y_pred. Se visualizan los resultados utilizando matplotlib. Los datos reales se muestran como puntos azules, la función real (raíz cuadrada) se muestra como una línea roja punteada, y las predicciones del modelo se muestran como una línea verde. Se agrega etiquetas y una leyenda para mayor claridad. Se comparan las predicciones del modelo con los valores reales calculados mediante la función nativa de Python (raíz cuadrada). Los resultados se imprimen en pantalla para su comparación.

En resumen, este código realiza regresión polinómica de grado 2 en un conjunto de datos generados aleatoriamente. Se crea un modelo de regresión, se generan características polinómicas, se entrena el modelo y se realizan predicciones. Luego, se visualizan los resultados y se comparan con los valores reales. Este código es un ejemplo de ajuste polinómico y visualización de resultados de regresión.