import numpy as np

This line imports the NumPy library, which is used for numerical operations in Python. It is often used for handling arrays and matrices.

from sklearn.linear\_model import LinearRegression

This imports the LinearRegression class from the sklearn.linear\_model module. This class is used to create and fit a linear regression model.

from sklearn.datasets import make\_regression

This imports the make\_regression function from the sklearn.datasets module. This function generates a synthetic dataset for regression tasks.

from sklearn.model\_selection import train\_test\_split

This line imports the train\_test\_split function from sklearn.model\_selection. This function is used to split the dataset into training and testing subsets.

from sklearn.metrics import mean\_squared\_error, r2\_score

This imports two metrics: mean\_squared\_error and r2\_score from sklearn.metrics. These metrics will be used to evaluate the performance of the regression model.

import matplotlib.pyplot as plt

This imports the pyplot module from matplotlib, which is a plotting library in Python. It is used for creating static, animated, and interactive visualizations in Python.

# Generate a synthetic dataset

X,y = make\_regression(n\_samples=1000, n\_features=1, noise=20, random\_state=42)

This line generates a synthetic dataset using the make\_regression function. Here:

* n\_samples=1000 specifies that we want 1000 samples.
* n\_features=1 means there is 1 feature (independent variable).
* noise=20 adds some random noise to the data, which makes the regression task more realistic.
* random\_state=42 ensures that the random generation is reproducible.

The generated features are stored in X, and the target variable in y.

# Split the dataset into training and testing sets

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2, random\_state=42)

This line splits the dataset into training and testing sets using the train\_test\_split function:

* X\_train and y\_train will be the training data (80% of the original data), while X\_test and y\_test will be the testing data (20% of the original data).
* test\_size=0.2 indicates that 20% of the data should be used for testing.
* random\_state=42 ensures that the split is reproducible.

# Initialize the Linear Regression model

linear\_reg = LinearRegression()

This line creates an instance of the LinearRegression class, initializing a linear regression model called linear\_reg.

# Train the model on the training data

linear\_reg.fit(X\_train, y\_train)

This line fits the linear regression model to the training data. The model learns the relationship between the features in X\_train and the target variable y\_train.

# Make predictions on the testing data

predictions = linear\_reg.predict(X\_test)

This line uses the fitted linear regression model to make predictions on the testing set X\_test. The predicted values are stored in the predictions variable.

# Evaluate the model's performance

mse = mean\_squared\_error(y\_test, predictions)

This line calculates the Mean Squared Error (MSE) between the actual target values y\_test and the predicted values predictions. MSE measures the average of the squared differences between the actual and predicted values.

r2 = r2\_score(y\_test, predictions)

This line computes the R-squared (R²) score, which indicates how well the predicted values approximate the actual values. It is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable.

print("Mean Squared Error (MSE):\n",mse)

This line prints the Mean Squared Error calculated earlier.

print("R-squared:\n",r2)

This line prints the R-squared score calculated earlier.

# Plotting the regression line (optional)

plt.scatter(X\_test, y\_test, color='blue')

This line creates a scatter plot of the actual testing data (X\_test, y\_test), where each point represents an actual observation, colored blue.

plt.plot(X\_test, predictions, color='red', linewidth=3)

This line plots the predicted values (from the model) as a line on the same plot, using red color and a specified line width.

plt.xlabel('X')

This line labels the x-axis of the plot.

plt.ylabel('y')

This line labels the y-axis of the plot.

plt.title('Linear Regression')

This line adds a title to the plot.

plt.show()

This line displays the plot. It will show the scatter plot of actual data points and the regression line fitted to those points.