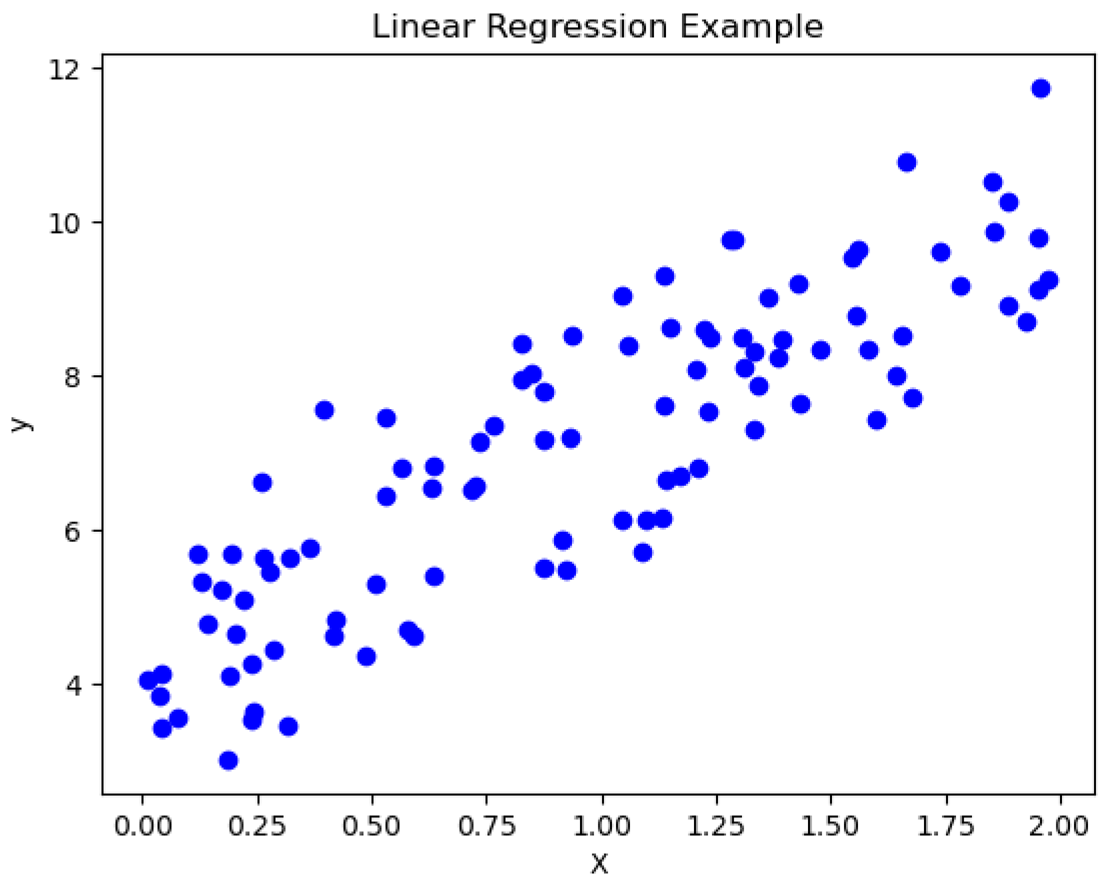
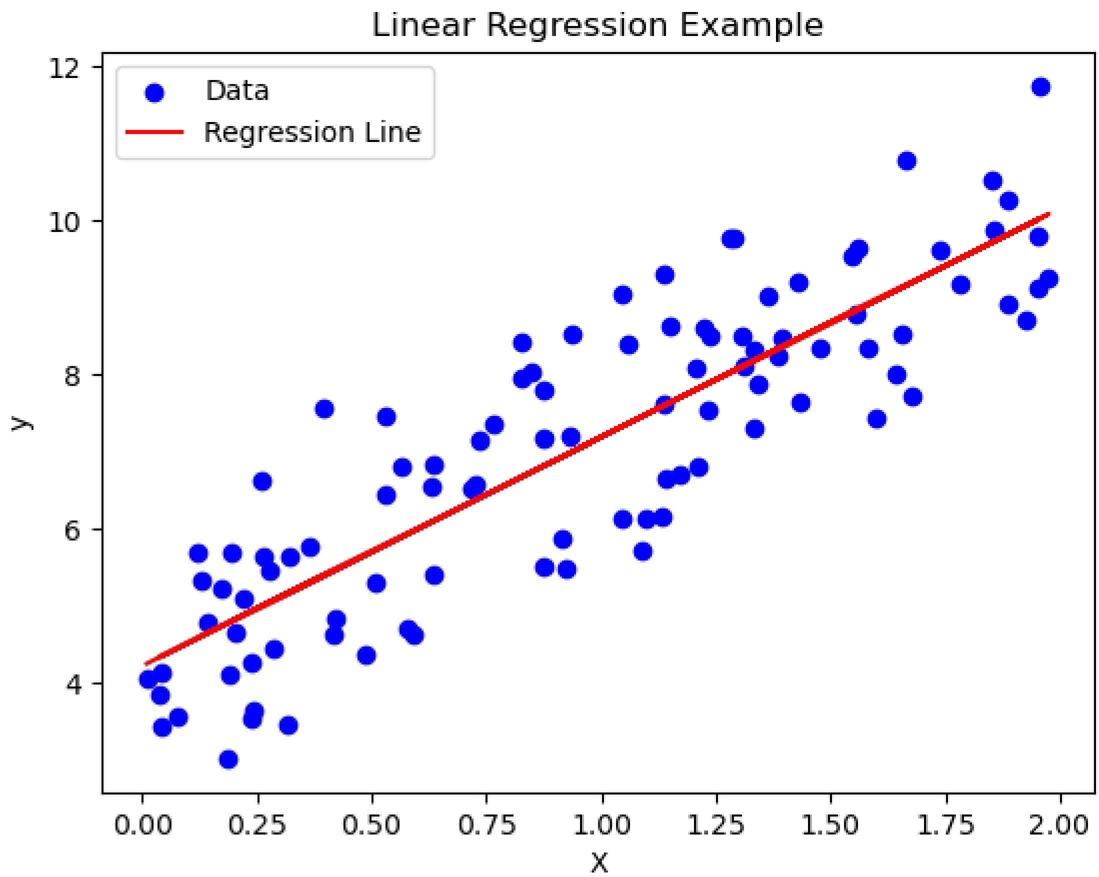
In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
# Generate synthetic data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
# Plot the data
plt.scatter(X, y, label='Data',color='blue')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Example')
plt.show()
# Split the data into training and testing sets (if needed)
# from sklearn.model_selection import train_test_split
# X_train, X_test, y_train, y_test = train_test_split
#X, y,test_size==0.2, random_state==42)
# Perform linear regression
from sklearn.linear_model import LinearRegression
# Create a Linear Regression model
model = LinearRegression()
# Fit the model to the data
model.fit(X, y)
# Print the coefficients (slope and intercept)
print("Slope (Coefficient):", model.coef_)
print("Intercept:", model.intercept_)
# Plot the regression line
plt.scatter(X, y, label='Data',color='blue')
plt.plot(X, model.predict(X), color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Example')
plt.legend()
plt.show()
```

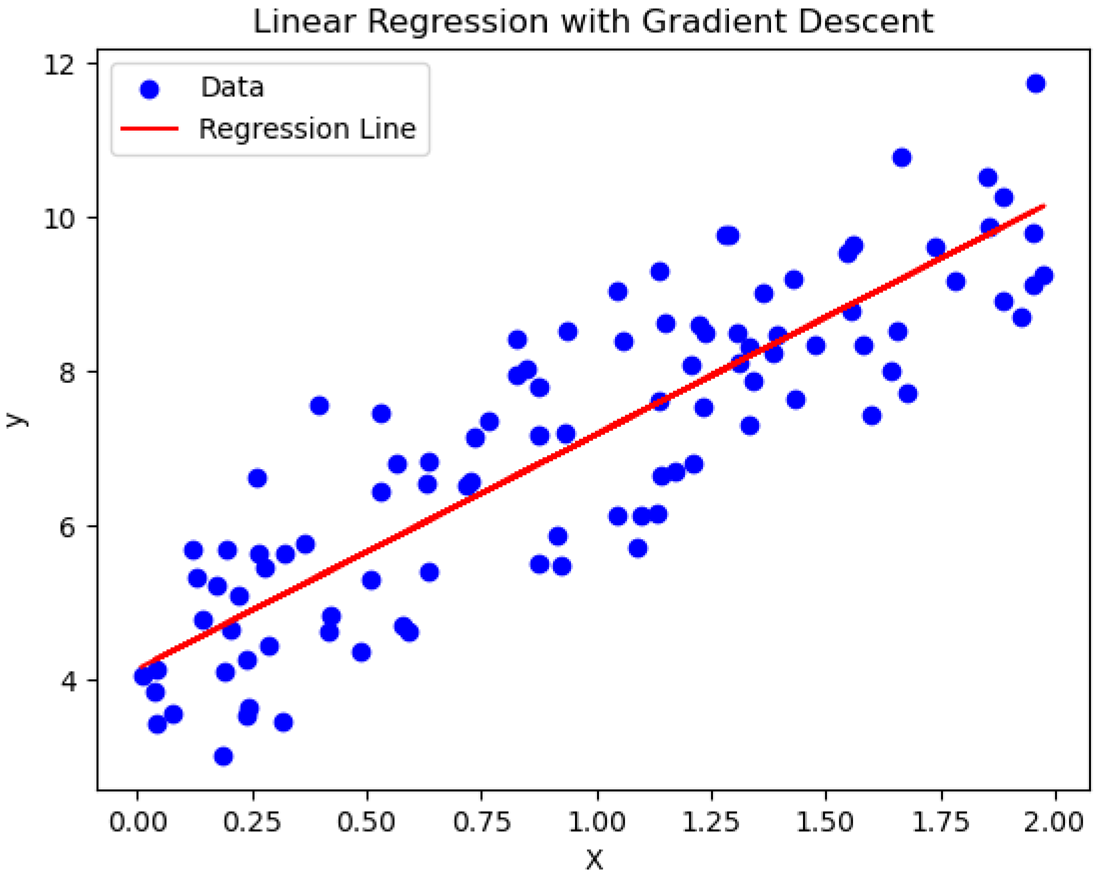



```
Slope (Coefficient): [[2.96846751]]
Intercept: [4.22215108]
```



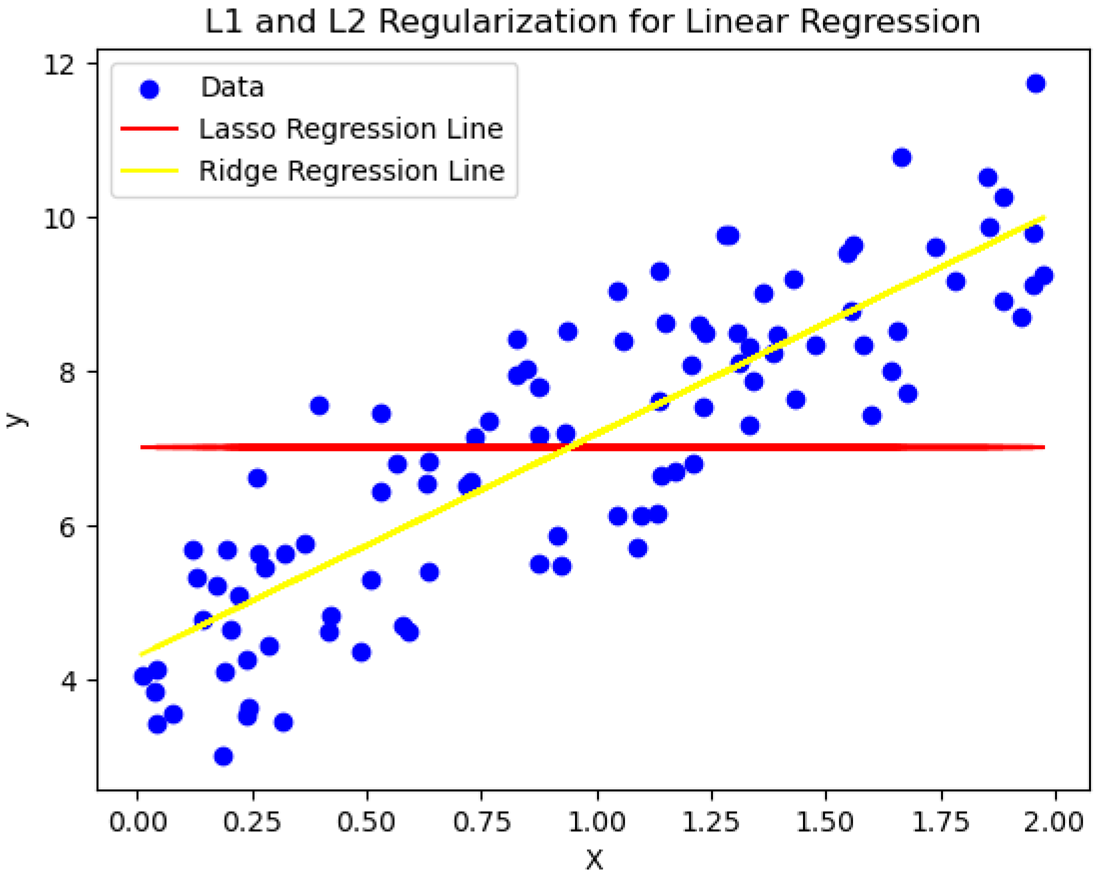
In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
# Generate synthetic data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
# Define the learning rate and number of iterations
learning_rate = 0.01
num_iterations = 1000
# Initialize the slope (theta1) and intercept (theta0) with random
#values
theta0 = np.random.randn()
theta1 = np.random.randn()
# Perform gradient descent
for i in range(num_iterations):
 # Calculate the predictions
 y_pred = theta0 + theta1 * X

 # Calculate the error
 error = y_pred - y

 # Calculate the gradient with respect to theta0 and theta1
 gradient_theta0 = (1/len(X)) * np.sum(error)
 gradient_theta1 = (1/len(X)) * np.sum(error * X)

 # Update the parameters using the gradient and learning rate
 theta0 = theta0 - learning_rate * gradient_theta0
 theta1 = theta1 - learning_rate * gradient_theta1
# Print the final parameters
print("Intercept (theta0):", theta0)
print("Slope (theta1):", theta1)
# Plot the data and the regression line
plt.scatter(X, y, label='Data',color='blue')
plt.plot(X, theta0 + theta1 * X, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression with Gradient Descent')
plt.legend()
plt.show()
```

```
Intercept (theta0): 4.14146131326896
Slope (theta1): 3.040065929195248
```


Linear Regression with Gradient Descent
Data
Regression Line
12
10
8
6
4
y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
X

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import mean_squared_error
# Generate synthetic data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
# Create models for L1 and L2 regularization
lasso_model = Lasso(alpha=1.0) # L1 regularization (alpha parameter
#controls the strength)
ridge_model = Ridge(alpha=1.0) # L2 regularization (alpha parameter
#Econtrols the strength)
# Fit the models to the data
lasso_model.fit(X, y)
ridge_model.fit(X, y)
# Print the coefficients and intercepts
print("Lasso Model Coefficients:", lasso_model.coef_)
print("Lasso Model Intercept:", lasso_model.intercept_)
print("Ridge Model Coefficients:", ridge_model.coef_)
print("Ridge Model Intercept:", ridge_model.intercept_)
# Plot the data and regression lines for Lasso and Ridge
plt.scatter(X, y, label='Data',color='blue')
plt.plot(X, lasso_model.predict(X), color='red', label='Lasso Regression Li
plt.plot(X, ridge_model.predict(X), color='yellow', label='Ridge Regressior
plt.xlabel('X')
plt.ylabel('y')
plt.title('L1 and L2 Regularization for Linear Regression')
plt.legend()
plt.show()
# Calculate mean squared error for both models
lasso_mse = mean_squared_error(y, lasso_model.predict(X))
ridge_mse = mean_squared_error(y, ridge_model.predict(X))
print("Lasso Mean Squared Error:", lasso_mse)
print("Ridge Mean Squared Error:", ridge_mse)
```

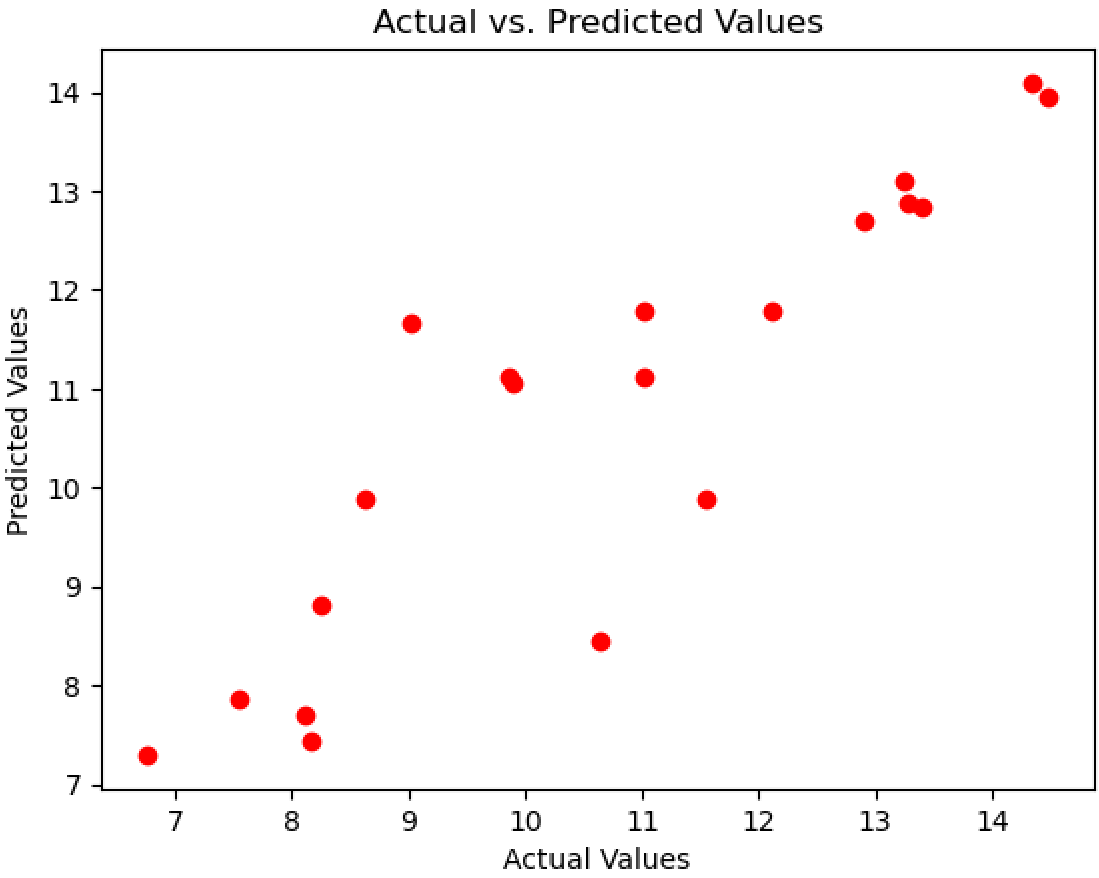
```
Lasso Model Coefficients: [0.]
Lasso Model Intercept: [7.02909738]
Ridge Model Coefficients: [[2.88178965]]
Ridge Model Intercept: [4.30411259]
```



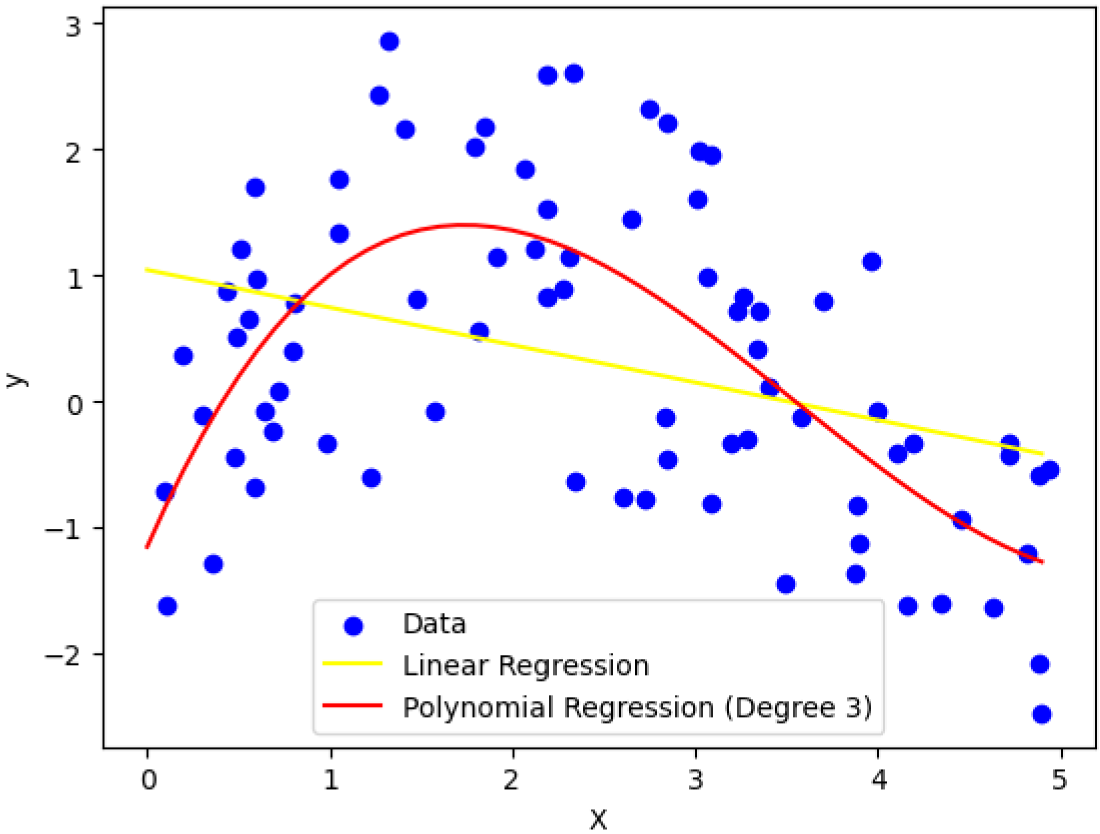

```
Lasso Mean Squared Error: 3.9221087015211338
Ridge Mean Squared Error: 0.9949365222436888
```

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
# Generate synthetic data with multiple features
np.random.seed(0)
X = 2 * np.random.rand(100, 3)
y = 4 + 3 * X[:, 0] + 2 * X[:, 1] + 1.5 * X[:, 2] +np.random.randn(100)
# Split the data 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)
# Create a Multivariate Linear Regression model
model = LinearRegression()
# Fit the model to the training data
model.fit(X_train, y_train)
# Make predictions on the test data
y_pred = model.predict(X_test)
# Print the coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
# Calculate mean squared error and R-squared (coefficient of
#determination)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared (Coefficient of Determination):", r2)
# Plot the data and the predicted values
plt.scatter(y_test, y_pred,color='red')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.show()
```

```
Coefficients: [2.93437988 1.88003911 1.54355848]
Intercept: 4.0163370124709346
Mean Squared Error: 1.0964652490522462
R-squared (Coefficient of Determination): 0.7987189077222158
```


Actual vs. Predicted Values
Predicted Values
14
13
12
11
10
9
8
7
7
8
9
10
11
12
13
14
Actual Values

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Generate some sample data
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + np.random.randn(80)

# Fit a linear regression model to the data
linear_reg = LinearRegression()
linear_reg.fit(X, y)

# Generate a range of X values for prediction
X_range = np.arange(0, 5, 0.1)[:, np.newaxis]

# Transform the input data to include polynomial features (e.g., X^2)
degree = 3  # You can change the degree to control the polynomial order
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)

# Fit a polynomial regression model to the transformed data
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)

# Predict the values for the X_range using the polynomial regression model
X_range_poly = poly_features.transform(X_range)
y_poly_pred = poly_reg.predict(X_range_poly)

# Plot the data and regression lines
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X_range, linear_reg.predict(X_range), color='yellow', label='Linea
plt.plot(X_range, y_poly_pred, color='red', label='Polynomial Regression (D
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```



3
2
1
0
−1
−2
y
0
1
2
3
4
5
X
Data
Linear Regression
Polynomial Regression (Degree 3)

In [ ]: