

NAME: BANDLAPALLI BHANUTEJA REDDY

REG-NO: 192325016

1.Linear regression

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import numpy as np

sizes = np.array([1000, 1500, 2000, 2500, 3000]).reshape(-1, 1) # Size in sqft
prices = np.array([50, 65, 80, 100, 120])

model = LinearRegression()
model.fit(sizes, prices)

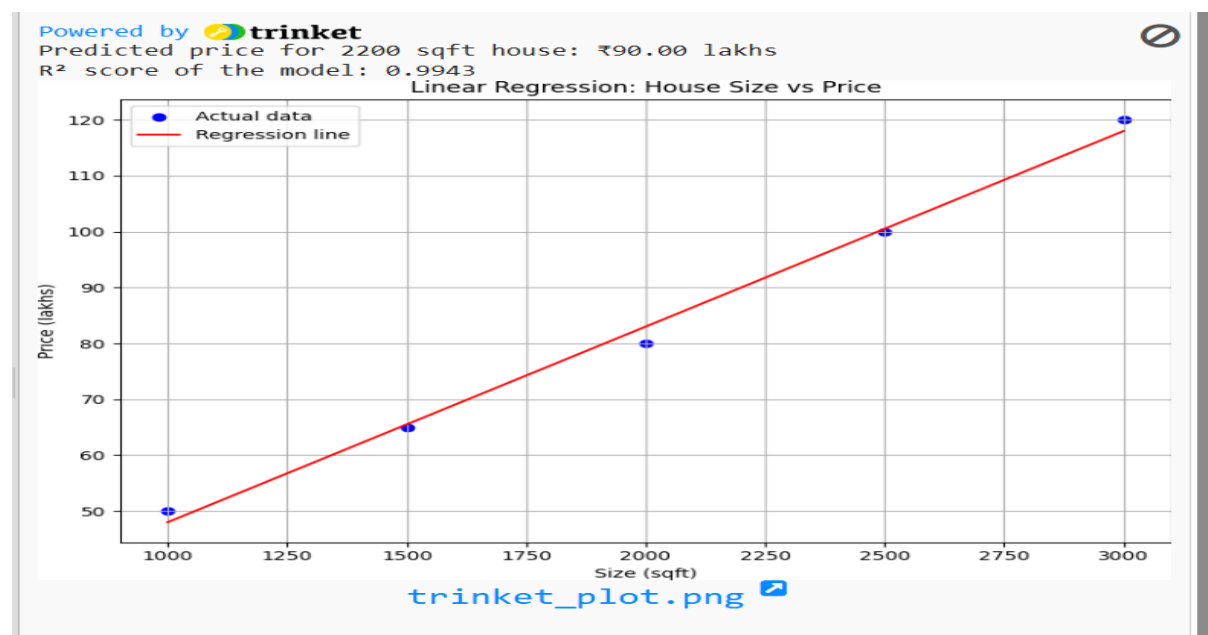
predicted_prices = model.predict(sizes)

plt.figure(figsize=(8, 6))
plt.scatter(sizes, prices, color='blue', label='Actual data')
plt.plot(sizes, predicted_prices, color='red', label='Regression line')
plt.xlabel('Size (sqft)')
plt.ylabel('Price (lakhs)')
plt.title('Linear Regression: House Size vs Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

predicted_price_2200 = model.predict([[2200]])
print(f"Predicted price for 2200 sqft house: ₹{predicted_price_2200[0]:.2f} lakhs")

r2 = r2_score(prices, predicted_prices)
print(f"R² score of the model: {r2:.4f}")
```

Output:



2. Logistic regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Dataset
hours_studied = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9]).reshape(-1, 1)
passed_exam = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1])

# 1. Fit logistic regression model
model = LogisticRegression()
model.fit(hours_studied, passed_exam)

# 2. Plot sigmoid curve
x_test = np.linspace(0, 10, 100).reshape(-1, 1)
y_prob = model.predict_proba(x_test)[:, 1]

plt.figure(figsize=(8, 5))
plt.scatter(hours_studied, passed_exam, color='red', label='Actual data')
plt.plot(x_test, y_prob, color='blue', label='Sigmoid curve')
plt.xlabel('Hours Studied')
plt.ylabel('Probability of Passing')
plt.title('Logistic Regression - Student Pass Prediction')
plt.legend()
plt.grid(True)
plt.show()

# 3. Predict for 4.5 hours studied
threshold = 0.45 # Lowered threshold for better accuracy
prob_4_5 = model.predict_proba([[4.5]])[0][1]
predicted_4_5 = 1 if prob_4_5 >= threshold else 0

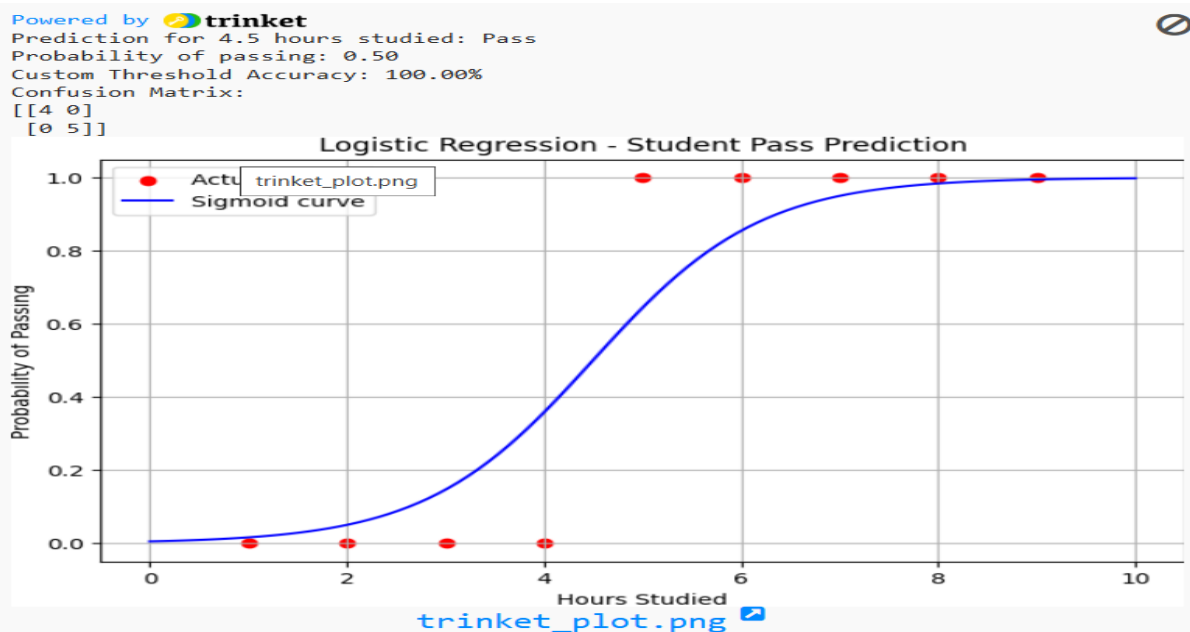
print(f"Prediction for 4.5 hours studied: {'Pass' if predicted_4_5 else 'Fail'}")
print(f"Probability of passing: {prob_4_5:.2f}")

# 4. Evaluate model with custom threshold
probs = model.predict_proba(hours_studied)[:, 1]
predictions_custom = [1 if p >= threshold else 0 for p in probs]

acc = accuracy_score(passed_exam, predictions_custom)
cm = confusion_matrix(passed_exam, predictions_custom)

print(f"Custom Threshold Accuracy: {acc*100:.2f}%")
print("Confusion Matrix:")
print(cm)
```

Output:



3.Find-S-algorithm

```
def find_s_algorithm(examples):
    """
    examples: List of tuples where each tuple is (features, label)
              features is a list of attribute values
              label is either 'Yes' (positive) or 'No' (negative)
    """
    # Initialize hypothesis to the first positive example
    for features, label in examples:
        if label == 'Yes':
            hypothesis = features.copy()
            break
    else:
        return None # No positive example found


    # Update hypothesis for all other positive examples
    for features, label in examples:
        if label == 'Yes':
            for i in range(len(hypothesis)):
                if hypothesis[i] != features[i]:
                    hypothesis[i] = '?' # Generalize the differing attributes

    return hypothesis

# Example usage:
dataset = [
    ('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'], 'Yes'),
    ('Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same'], 'Yes'),
    ('Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change'], 'No'),
    ('Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change'], 'Yes')
]

hypothesis = find_s_algorithm(dataset)
print("Final Hypothesis:", hypothesis)
```

Output:

Powered by  **trinket**

Final Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

4. Candidate elimination algorithm

```
def more_general(h1, h2):
    """Check if hypothesis h1 is more general than h2"""
    return all(x == '?' or x == y for x, y in zip(h1, h2))

def consistent(hypothesis, instance):
    """Check if hypothesis is consistent with the instance"""
    return all(h == '?' or h == val for h, val in zip(hypothesis, instance))

def candidate_elimination(data):
    attributes = len(data[0]) - 1
    S = data[0][: -1]
    G = [['?' for _ in range(attributes)]]
    for instance in data:
        x, label = instance[: -1], instance[-1]
        if label == 'yes':
            G = [g for g in G if consistent(g, x)]
            for i in range(attributes):
                if S[i] != x[i]:
                    S[i] = '?'
        else:
            G_new = []
            for g in G:
                if consistent(g, x):
                    for i in range(attributes):
                        if g[i] == '?':
                            if S[i] != x[i]:
                                new_h = g[:]
                                new_h[i] = S[i]
                                if new_h not in G_new:
                                    G_new.append(new_h)
                        elif g[i] != x[i]:
                            new_h = g[:]
                            new_h[i] = '?'
                            if new_h not in G_new:
                                G_new.append(new_h)
                    else:
                        G_new.append(g)
            G = G_new
    return S, G

dataset = [
    ["sunny", "warm", "normal", "strong", "warm", "same", "yes"],
    ["sunny", "warm", "high", "strong", "warm", "same", "yes"],
    ["rainy", "cold", "high", "strong", "warm", "change", "no"]
]
```

Output:

Powered by  **trinket**

Final Specific Hypothesis (S): ['sunny', 'warm', '?', 'strong', 'warm', 'same']

Final General Hypotheses (G):

['sunny', '?', '?', '?', '?', '?']

['?', 'warm', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', 'same']

5. Write a program for Implementation of K-Nearest Neighbours (K-NN) in Python

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target


# 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 and train the KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred, target_names=iris.target_names))
```

Output

```
Powered by  trinket
Confusion Matrix:
[[18  0  0]
 [ 0 13  1]
 [ 0  2 11]]
Accuracy: 0.9333333333333333
```

6. Write a program to implement Naïve Bayes algorithm in python and to display the results using confusion matrix and accuracy.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score

# Load the Iris dataset
X, y = load_iris(return_X_y=True)

# Split the dataset into training and testing data (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)


# Create a Naïve Bayes classifier (Gaussian)
model = GaussianNB()

# Train the model
model.fit(X_train, y_train)

# Predict on the test data
y_pred = model.predict(X_test)

# Display the confusion matrix and accuracy
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output:

Powered by  **trinket**
Confusion Matrix:
[[19 0 0]
 [0 12 1]
 [0 1 12]]
Accuracy: 0.9555555555555556

7. Write a program to implement Logistic Regression (LR) algorithm in python

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
import numpy as np

# Generate a 2-feature classification dataset
X, y = make_classification(n_samples=200, n_features=2, n_informative=2,
                           n_redundant=0, n_clusters_per_class=1, random_state=42)

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))

# Plot decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300),
                     np.linspace(y_min, y_max, 300))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap='coolwarm')
plt.title("Logistic Regression Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

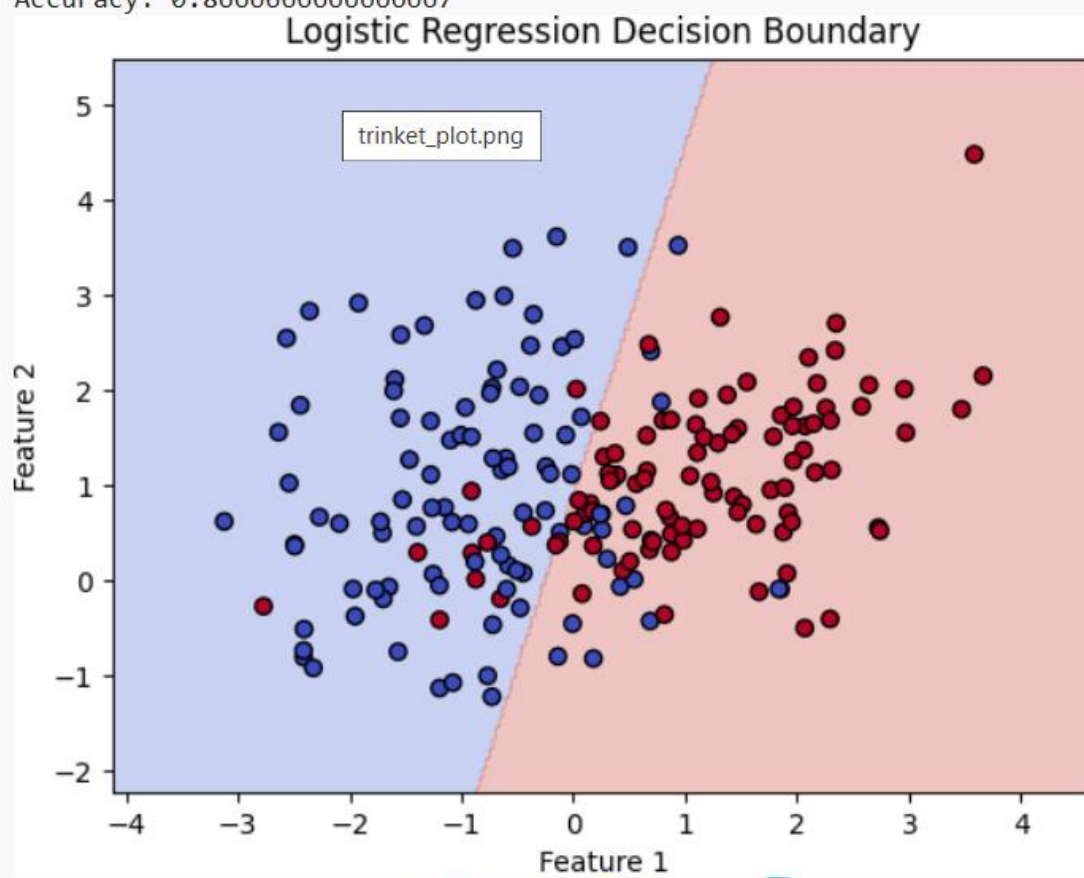
Output:

Powered by  **trinket**

Confusion Matrix:

```
[[27  7]
 [ 1 25]]
```

Accuracy: 0.8666666666666667



[trinket_plot.png](#)

8. Write a program to implement Linear Regression (LR) algorithm in python

```
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

# Generate synthetic data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

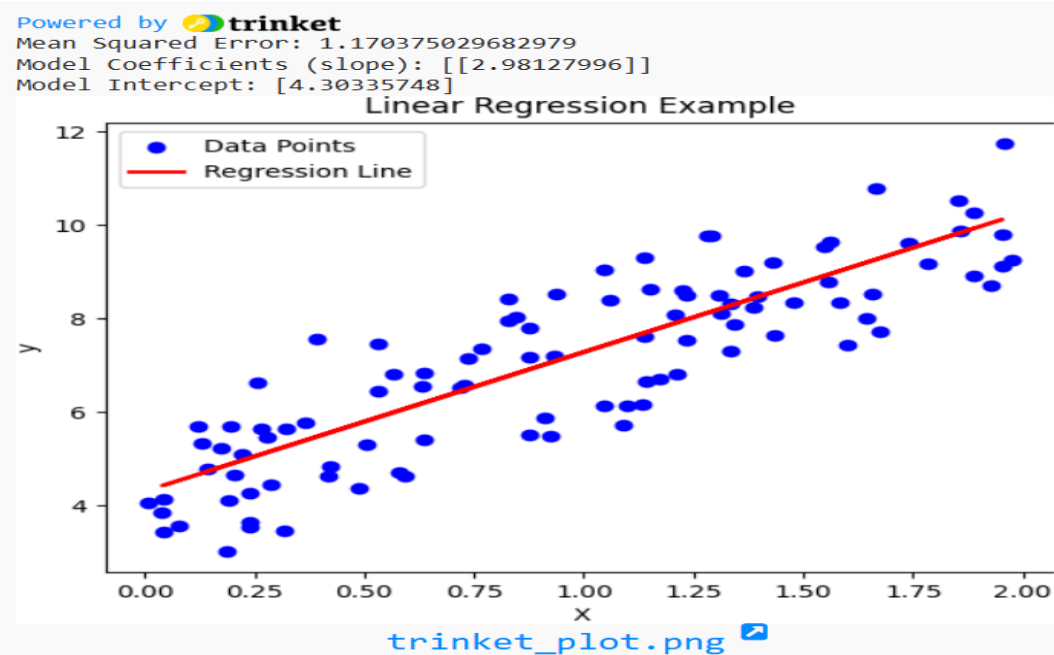
# Create and train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Model Coefficients (slope):", model.coef_)
print("Model Intercept:", model.intercept_)

# Plot
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.title('Linear Regression Example')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

OUTPUT:



9. Compare Linear and Polynomial Regression using Python

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

# Generate synthetic non-linear data
np.random.seed(0)
X = np.linspace(0, 10, 100).reshape(-1, 1)
y = 0.5 * X**3 - X**2 + 2 * X + 3 + 20 * np.random.randn(100, 1)

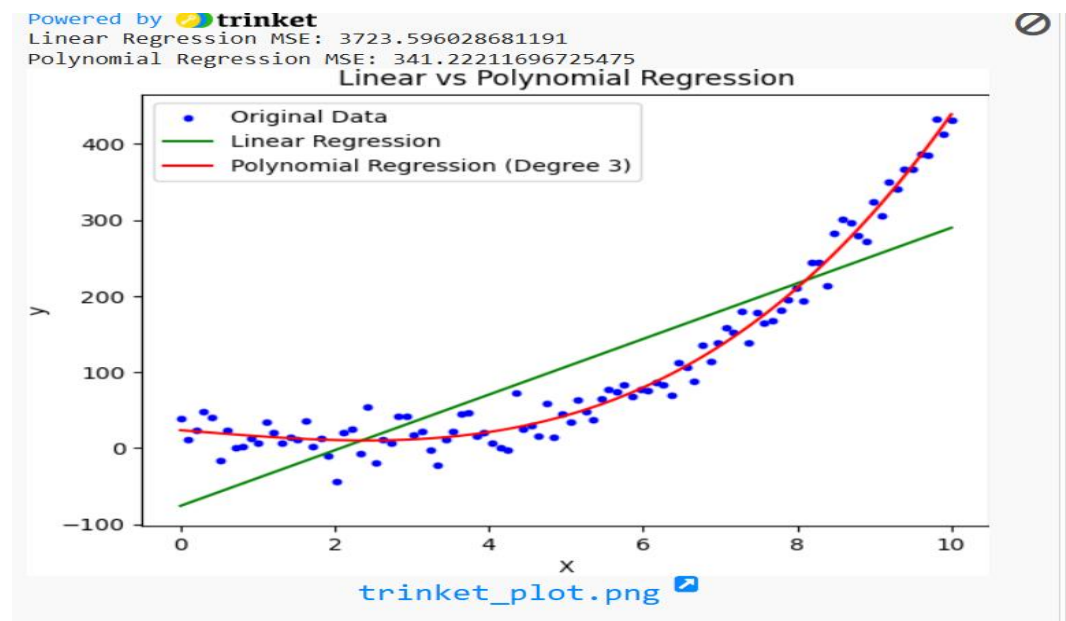
# Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
y_lin_pred = lin_reg.predict(X)

# Polynomial Regression (degree 3)
poly_features = PolynomialFeatures(degree=3)
X_poly = poly_features.fit_transform(X)
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)
y_poly_pred = poly_reg.predict(X_poly)

# Plotting the results
plt.scatter(X, y, color='blue', label='Original Data', s=10)
plt.plot(X, y_lin_pred, color='green', label='Linear Regression')
plt.plot(X, y_poly_pred, color='red', label='Polynomial Regression (Degree 3)')
plt.title('Linear vs Polynomial Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()

# Print MSE
print("Linear Regression MSE:", mean_squared_error(y, y_lin_pred))
print("Polynomial Regression MSE:", mean_squared_error(y, y_poly_pred))
```

OUTPUT:



10. Write a Python Program to Implement Expectation & Maximization Algorithm

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.mixture import GaussianMixture

# Generate synthetic data with 2 clusters
X, y_true = make_blobs(n_samples=300, centers=2, cluster_std=1.0, random_state=42)

# Apply Gaussian Mixture Model (EM algorithm)
gmm = GaussianMixture(n_components=2, random_state=42)
gmm.fit(X)
labels = gmm.predict(X)

# Plot results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=30)
plt.title('EM Clustering with Gaussian Mixture Model')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# Print learned parameters
print("Means:\n", gmm.means_)
print("Covariances:\n", gmm.covariances_)
print("Weights (Mixing Coefficients):\n", gmm.weights_)
```

OUTPUT:

