Experiment 2

Aim

To implement and analyze Linear Regression and Logistic Regression models on real-world datasets using Python and evaluate their performance using appropriate metrics.

---

1. Dataset Source

Dataset 1 – Linear Regression

Student Performance Dataset
Target Variable: `Final_Score`
Source:
https://github.com/jaykerkar0405/MLDL-Lab/blob/main/datasets/student_performance.csv

Dataset 2 – Logistic Regression

Breast Cancer Wisconsin Dataset (Binary Classification)
Source:
https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html

---

2. Dataset Description

Dataset 1 – Linear Regression

Features:

- Hours_Studied
- Attendance

- Assignment_Score
- Midterm_Score

Target:

- Final_Score (Continuous)

Characteristics:

- Type: Numerical tabular dataset
- Problem Type: Regression
- No missing values
- Used to predict continuous output

Dataset 2 – Logistic Regression

Features:

- 30 numerical medical attributes (e.g., radius, texture, perimeter)

Target:

- Diagnosis (0 = Malignant, 1 = Benign)

Characteristics:

- Type: Numerical tabular dataset
- Problem Type: Binary Classification
- Balanced dataset
- Suitable for logistic regression

---

3. Mathematical Formulation of the Algorithm

Linear Regression

Hypothesis Function:

$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta n x n$

Cost Function (Mean Squared Error):

$MSE = (1 / N) \Sigma (yi - \hat{y}i)^2$

Gradient Descent Update Rule:

$\beta j = \beta j - \alpha (\partial J / \partial \beta j)$

Where:

- β = coefficients
- α = learning rate
- N = number of samples

Logistic Regression

Hypothesis (Sigmoid Function):

$\sigma(z) = 1 / (1 + e^{-z})$

where
$z = \beta_0 + \beta_1 x_1 + \ldots + \beta n x n$

Cost Function (Log Loss):

$J = - (1/N) \Sigma [yi \log(\hat{y}i) + (1 - yi) \log(1 - \hat{y}i)]$

Decision Boundary:

If $\hat{y} \geq 0.5 \rightarrow$ Class 1
Else $\rightarrow$ Class 0

---

4. Algorithm Limitations

Linear Regression

- Sensitive to outliers
- Assumes linear relationship

- Poor performance on non-linear data

Logistic Regression

- Only suitable for linear decision boundaries
- Cannot handle complex non-linear separations
- Assumes independence between features

---

5. Methodology / Workflow

Linear Regression Workflow

1. Load dataset using Pandas
2. Select independent and dependent variables
3. Split dataset into training and testing sets
4. Train Linear Regression model
5. Predict test values
6. Evaluate using MSE and R^2 score

Logistic Regression Workflow

1. Load dataset
2. Split into train-test sets
3. Standardize features
4. Train Logistic Regression model
5. Predict test labels
6. Evaluate using Accuracy, Confusion Matrix

---

6. Performance Analysis

Linear Regression Metrics

Mean Squared Error (MSE)
R^2 Score

Interpretation:

- Lower MSE indicates better model
- R^2 close to 1 indicates good fit

Logistic Regression Metrics

Accuracy
Confusion Matrix
Precision
Recall
F1 Score

Interpretation:

- Higher accuracy indicates better classification
- Confusion matrix shows TP, TN, FP, FN

---

7. Hyperparameter Tuning

Linear Regression

- Fit intercept (True/False)

Logistic Regression

- Regularization parameter (C)
- Penalty (L1 / L2)
- Solver type

Hyperparameter tuning improves generalization and prevents overfitting.

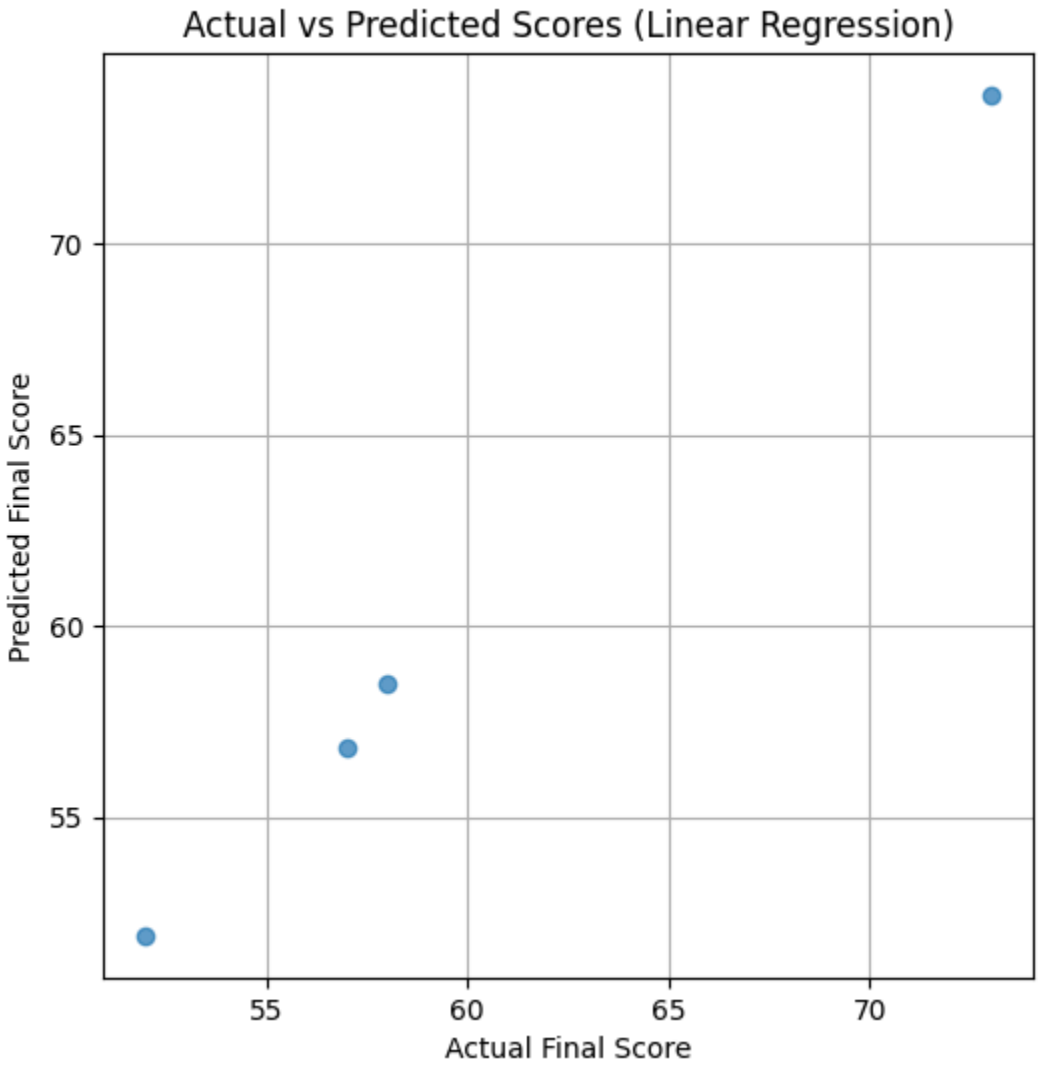
Exercise 1: Linear Regression

Code

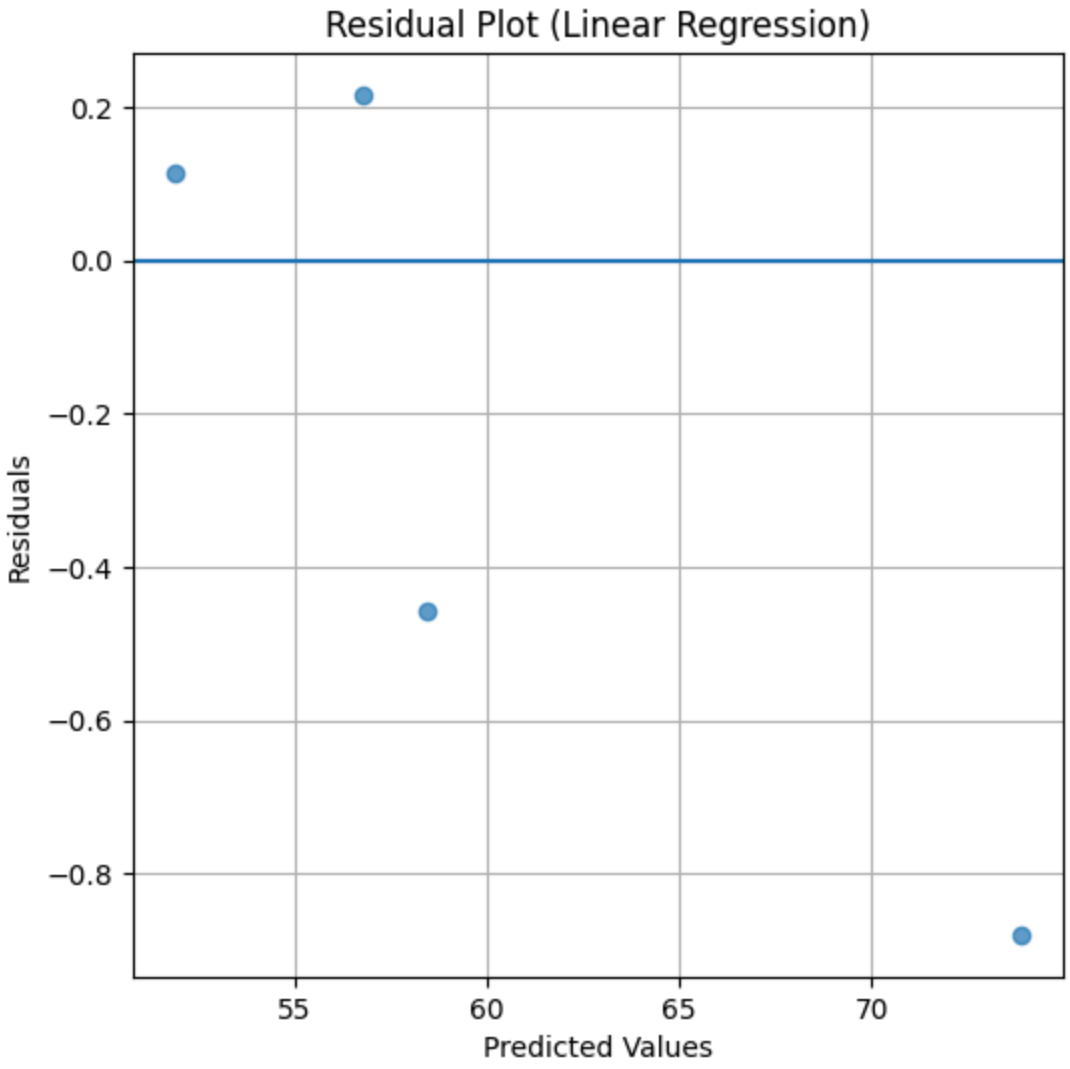
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
df = pd.read_csv('student_performance.csv')

X = df[['Hours_Studied', 'Attendance', 'Assignment_Score',
'Midterm_Score']]
y = df['Final_Score']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Model Training
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
```

```
print("R2 Score:", r2)
```

Output




```
@jaykerkar0405 ➜ /workspaces/MLDL-Lab (main) $ python3 main.py
Mean Squared Error: 0.2612765457256682
R2 Score: 0.9957516008825095
```



Exercise 2: Logistic Regression

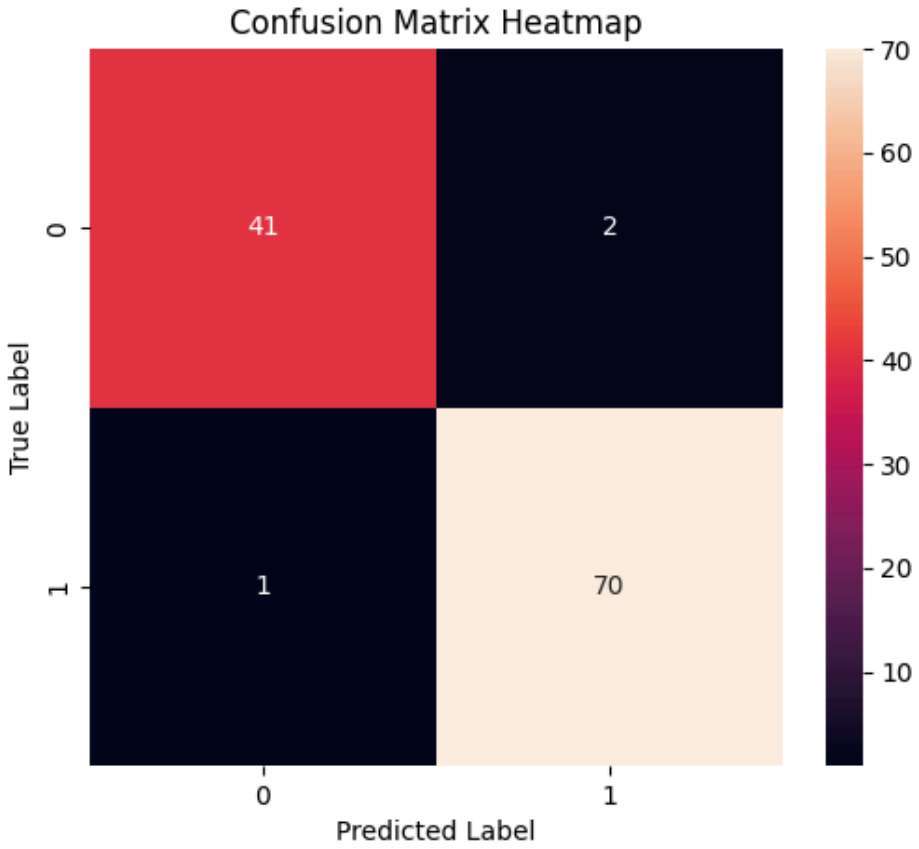
Code

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.preprocessing import StandardScaler

# Load dataset
```

```
data = load_breast_cancer()
X = data.data
y = data.target

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Model Training
model = LogisticRegression(max_iter=5000)
model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
```

Output

```
@jaykerkar0405 → /workspaces/MLDL-Lab (main) $ python3 main.py
Accuracy: 0.9736842105263158
Confusion Matrix:
 [[41  2]
 [ 1 70]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```



Conclusion

This experiment successfully implemented Linear and Logistic Regression on real-world datasets. Linear Regression effectively predicted continuous student scores, while Logistic Regression classified medical data into benign and malignant categories. Performance metrics such as MSE, R^2 score, Accuracy, and F1-score were used to evaluate the models. These algorithms form the foundation of supervised machine learning and are widely used in real-world predictive analytics and classification tasks.