LAB Assignment No 2

Topic: Multiple and Logistic Linear Regression

Lab Practice Questions

Q1. Multiple Linear Regression – House Price Prediction

A dataset contains:

- Size (sqft),
- Number of Bedrooms,
- Age of House (years)

and the target variable is **House Price**.

👉 Task:

1. Fit a **multiple linear regression model**.
2. Predict the price of a house with: **Size = 2000 sqft**, **Bedrooms = 3**, Age = **10 years.**
3. Print coefficients and interpret them.

assignment2Q1.ipynb > import pandas as pd

+ Code + Markdown | Run All Clear All Outputs | Outline ··· Python 3.13.7

[23]
```python
import pandas as pd
```

[24]
```python
data = pd.read_csv("House price.csv")
print(data.head())
```

```
   Size  Bedrooms  Age   Price
0  2745         2    7  390746
1  3569         5   16  520069
2  1584         4   27  250234
3  1904         5   15  311628
4  2541         5    5  406236
```

[25]
```python
X = data[['Size', 'Bedrooms', 'Age']]  # features
y = data['Price']                      # target
```

```python
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)
```

```
LinearRegression
▶ Parameters
```

```python
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)

# Display them nicely
for feature, coef in zip(X.columns, model.coef_):
    print(f"{feature}: {coef:.2f}")
```

```
Intercept: 59495.351393212506
Coefficients: [  103.87169458 12545.52690969   -370.02435464]
Size: 103.87
Bedrooms: 12545.53
Age: -370.02
```

```python
sample = pd.DataFrame({'Size': [2000], 'Bedrooms': [3], 'Age': [10]})
predicted_price = model.predict(sample)
print("Predicted Price for given house:", predicted_price[0])
```

```
Predicted Price for given house: 301175.0777377575
```

```python
import matplotlib.pyplot as plt

plt.scatter(data['Size'], data['Price'], color='blue', label='Actual Data')
plt.xlabel('House Size (sqft)')
plt.ylabel('House Price')
plt.title('House Size vs Price')
plt.legend()
plt.show()
```

Q2. Multiple Linear Regression – Student Performance

Dataset columns:

- Hours Study,
- Hours Sleep,

- Attendance (%),
  Target: **Marks in Exam**

👉 Task:

1. Train a regression model.
2. Plot actual vs predicted marks.
3. Compute **R² score** and **Mean Squared Error (MSE)**.

Q2.ipynb > import pandas as pd

\+ Code + Markdown | Run All | Clear All Outputs | Outline ... Python 3.13.7

[68]

```
   Hours_Study  Hours_Sleep  Attendance  Marks
0            2            5          65     48
1            3            6          70     55
2            1            4          60     45
3            4            7          80     66
4            5            8          85     74
```

```python
X = data[['Hours_Study', 'Hours_Sleep', 'Attendance']]   # Features
y = data['Marks']                                         # Target
```

[69]

```python
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)
print("Model trained successfully!")
```

[70]

```
Model trained successfully!
```

\+ Code + Markdown | Run All | Clear All Outputs | Outline ... Python 3.13.7

```python
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

[71]

```
Intercept: 4.535027794866835
Coefficients: [3.24798118 0.83266363 0.54692488]
```

```python
y_pred = model.predict(X)
print("Predicted Marks (first 5):", y_pred[:5])
```

[72]

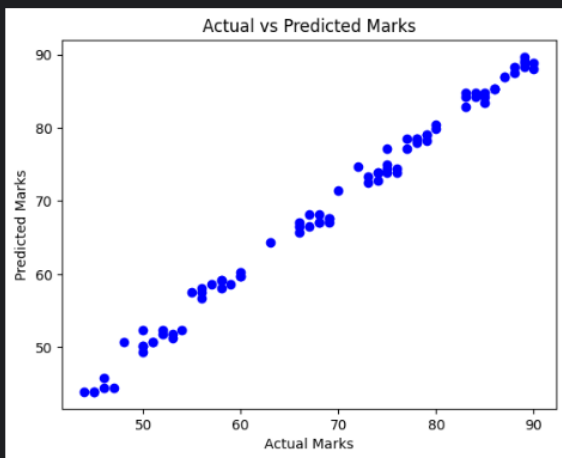
```
Predicted Marks (first 5): [50.7444254 57.5596946 43.9291562 67.1095882 73.9248574]
```

```python
from sklearn.metrics import r2_score, mean_squared_error

r2 = r2_score(y, y_pred)
mse = mean_squared_error(y, y_pred)

print("R² Score:", r2)
print("Mean Squared Error:", mse)
```

[73]

```
R² Score: 0.993404159089959
Mean Squared Error: 1.2634734555922102
```

```python
import matplotlib.pyplot as plt

plt.scatter(y, y_pred, color='blue')
plt.xlabel("Actual Marks")
plt.ylabel("Predicted Marks")
plt.title("Actual vs Predicted Marks")
plt.show()
```




```python
new_data = [[5, 7, 85]]  # 5 hours study, 7 hours sleep, 85% attendance
predicted_marks = model.predict(new_data)
print("Predicted Marks for new student:", predicted_marks)
```

```
Predicted Marks for new student: [73.09219377]
c:\Users\hp\Desktop\Assignment\venv\Lib\site-packages\sklearn\utils\validation.py:2749: UserWarning: X does not have valid feature names, but LinearRegression
  warnings.warn(
```

```python
data['Predicted_Marks'] = y_pred
data.to_csv("StudentPerformance_Predicted.csv", index=False)
print("Predictions saved successfully!")
```

```
Predictions saved successfully!
```

Q3. Logistic Regression – Pass/Fail Classification

Dataset columns:

- Hours Study
- Hours Sleep
  Target: Pass (1) / Fail (0)

👉 Task:

1. Fit a **logistic regression classifier**.

2. Predict the probability of passing if a student studies 30 hours and sleeps 6 hours.
3. Plot the **decision boundary** (pass vs fail).

Code Markdown Run All Clear All Outputs Outline ··· Python 3.13.7

[16]
```python
import pandas as pd
```

[17]
```python
data = pd.read_csv("PassFail.csv")
print(data.head())
```

```
   Hours_Study  Hours_Sleep  Pass
0            2            5     0
1            3            6     0
2            1            4     0
3            4            7     1
4            5            6     1
```

[18]
```python
X = data[['Hours_Study', 'Hours_Sleep']]
y = data['Pass']
```

[19]
```python
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X, y)

print("Model trained successfully!")
```

```
Model trained successfully!
```

Code Markdown Run All Clear All Outputs Outline ··· Python 3.13.7

[20]
```python
new_data = [[30, 6]]  # 30 hours study, 6 hours sleep
prob = model.predict_proba(new_data)
print("Probability of Passing:", prob[0][1])
```

```
Probability of Passing: 1.0
c:\Users\hp\Desktop\Assignment\venv\Lib\site-packages\sklearn\utils\validation.py:2749: UserWarning: X does not have valid feature names, but LogisticRegressio
  warnings.warn(
```

[21]
```python
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot of data
plt.scatter(data['Hours_Study'], data['Hours_Sleep'], c=data['Pass'], cmap='bwr')
plt.xlabel('Hours Study')
plt.ylabel('Hours Sleep')
plt.title('Pass vs Fail Decision Boundary')

# Create grid
x_min, x_max = data['Hours_Study'].min()-1, data['Hours_Study'].max()+1
y_min, y_max = data['Hours_Sleep'].min()-1, data['Hours_Sleep'].max()+1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))

# Predictions for grid
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Decision boundary
plt.contourf(xx, yy, Z, alpha=0.3, cmap='bwr')
plt.show()
```

Q4. Logistic Regression – Diabetes Prediction (Binary Classification)

Use a small dataset with:

- BMI,
- Age,
- Glucose Level
  Target: **Diabetic (1) or Not (0)**

👉 Task:

1. Fit logistic regression.
2. Find accuracy, precision, recall.
3. Predict whether a patient (BMI=28, Age=45, Glucose=150) is diabetic.

```python
import pandas as pd
```

[34]

```python
data = pd.read_csv("Diabetes.csv")
print(data.head())
```

[35]

```
   Glucose  BloodPressure   BMI  Age  Diabetes
0       85             66  26.6   31         0
1       89             68  28.1   33         0
2       78             50  31.0   26         0
3      115             70  34.6   35         1
4      120             80  36.5   29         1
```

```python
X = data[['Glucose', 'BloodPressure', 'BMI', 'Age']]  # Features
y = data['Diabetes']  # Target variable
```

[36]

```python
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)
```

[37]

```python
from sklearn.linear_model import LogisticRegression

# Create model
model = LogisticRegression()
```

```python
from sklearn.linear_model import LogisticRegression

# Create model
model = LogisticRegression()

# Train (fit) the model on training data
model.fit(X_train, y_train)

print("✅ Model training completed successfully!")
```

[38]

```
✅ Model training completed successfully!
```

```python
# Predict outcomes on test data
y_pred = model.predict(X_test)

# Show first 10 predictions
print("Predicted values:", y_pred[:10])
print("Actual values:", list(y_test[:10]))
```

[39]

```
Predicted values: [1 0 1 1 0 1 0 1 0 0]
Actual values: [1, 0, 1, 1, 0, 1, 0, 1, 0, 0]
```

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Calculate Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("✅ Accuracy:", accuracy)

# Confusion Matrix
print("\n📊 Confusion Matrix:")
```

[40]

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Calculate Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("✅ Accuracy:", accuracy)

# Confusion Matrix
print("\n📊 Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Classification Report
print("\n📋 Classification Report:")
print(classification_report(y_test, y_pred))
```

```
✅ Accuracy: 1.0

📊 Confusion Matrix:
[[ 9  0]
 [ 0 12]]

📋 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        12

    accuracy                           1.00        21
   macro avg       1.00      1.00      1.00        21
weighted avg       1.00      1.00      1.00        21
```

```
+ Code  + Markdown  | ▷ Run All  ☰ Clear All Outputs  | ☰ Outline  ···          Python 3.13.7

    accuracy                           1.00        21
   macro avg       1.00      1.00      1.00        21
weighted avg       1.00      1.00      1.00        21
```

```python
# Create new patient data
new_patient = pd.DataFrame({
    'Glucose': [150],
    'BloodPressure': [75],
    'BMI': [28],
    'Age': [45]
})

# Predict diabetes
prediction = model.predict(new_patient)
probability = model.predict_proba(new_patient)

print("Predicted Diabetes:", "Yes (1)" if prediction[0] == 1 else "No (0)")
print("Probability [Not Diabetic, Diabetic]:", probability[0])
```

```
Predicted Diabetes: Yes (1)
Probability [Not Diabetic, Diabetic]: [0. 1.]
```

Q5. Comparison – Linear vs Logistic Regression

Dataset columns:

- Hours Study,
- Exam Score,
- Pass/Fail

👉 Task:

1. Use **Linear Regression** to predict exam scores.
2. Use **Logistic Regression** to predict pass/fail.
3. Compare results — explain why linear regression is unsuitable for classification.

+ Code + Markdown | Run All | Clear All Outputs | Outline ... Python 3.13.7

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
```

[31] Python

```python
data = pd.read_csv("student_lab2 q5.csv")
data.head()
```

[32] Python

	Hours_Study	Exam_Score	Pass_Fail
0	1.0	32	0
1	1.5	35	0
2	2.0	38	0
3	2.5	40	0
4	3.0	42	0

```python
data.columns = data.columns.str.strip()
print(data.columns.tolist())
```

[33] Python

```
['Hours_Study', 'Exam_Score', 'Pass_Fail']
```

```python
X = data[['Hours_Study']]
y_score = data['Exam_Score']
y_class = data['Pass_Fail']
```

[34] Python

+ Code + Markdown | Run All | Clear All Outputs | Outline ... Python 3.13.7

```python
X = data[['Hours_Study']]
y_score = data['Exam_Score']
y_class = data['Pass_Fail']
```

[34] Python

```python
X_train, X_test, y_train_score, y_test_score = train_test_split(X, y_score, test_size=0.2, random_state=42)
X_train2, X_test2, y_train_class, y_test_class = train_test_split(X, y_class, test_size=0.2, random_state=42)
```

[35] Python

```python
lin_model = LinearRegression()
lin_model.fit(X_train, y_train_score)

y_pred_score = lin_model.predict(X_test)
r2 = r2_score(y_test_score, y_pred_score)
mse = mean_squared_error(y_test_score, y_pred_score)

# Convert continuous exam scores to pass/fail for accuracy comparison
y_pred_class_from_linear = (y_pred_score >= 50).astype(int)
linear_acc = accuracy_score(y_test_class, y_pred_class_from_linear)

print("📘 Linear Regression Results:")
print("R² Score:", round(r2, 3))
print("Mean Squared Error:", round(mse, 3))
print("Accuracy (converted to Pass/Fail):", round(linear_acc, 3))
```

[36] Python

```
📘 Linear Regression Results:
R² Score: 0.986
Mean Squared Error: 6.816
Accuracy (converted to Pass/Fail): 0.952
```

+ Code + Markdown | Run All | Clear All Outputs | Outline ... Python 3.13.7

```python
log_model = LogisticRegression()
log_model.fit(X_train2, y_train_class)

y_pred_class = log_model.predict(X_test2)
y_pred_prob = log_model.predict_proba(X_test2)[:,1]

acc = accuracy_score(y_test_class, y_pred_class)
mse_log = mean_squared_error(y_test_class, y_pred_prob)

print("🟩 Logistic Regression Results:")
print("Accuracy:", round(acc, 3))
print("Mean Squared Error:", round(mse_log, 3))
```

[37] Python

```
🟩 Logistic Regression Results:
Accuracy: 1.0
Mean Squared Error: 0.009
```

```python
plt.figure(figsize=(10,5))

# Linear Regression
plt.subplot(1,2,1)
plt.scatter(X, y_score, color='blue', label='Actual Scores')
plt.plot(X, lin_model.predict(X), color='red', label='Predicted Line')
plt.xlabel("Hours Study")
plt.ylabel("Exam Score")
plt.title("Linear Regression (Exam Score)")
plt.legend()

# Logistic Regression
plt.subplot(1,2,2)
plt.scatter(X, y_class, c=y_class, cmap='bwr', label='Actual')
plt.plot(X, log_model.predict_proba(X)[:,1], color='black', label='Probability Curve')
plt.xlabel("Hours Study")
plt.ylabel("Pass Probability")
```

[38]

Spaces: 4 Cell 1 of 8

+ Code + Markdown | Run All | Clear All Outputs | Outline ... Python 3.13.7

```python
plt.subplot(1,2,2)
plt.scatter(X, y_class, c=y_class, cmap='bwr', label='Actual')
plt.plot(X, log_model.predict_proba(X)[:,1], color='black', label='Probability Curve')
plt.xlabel("Hours Study")
plt.ylabel("Pass Probability")
```


```python
plt.title("Logistic Regression (Pass/Fail)")
plt.legend()

plt.tight_layout()
plt.show()
```

[38] Python