

EXPERIMENT - 1

AIM:

To Implement Linear and Logistic Regression on real-world datasets.

1. Dataset Source

The dataset used in this experiment is a Student Performance Dataset, containing academic and behavioral attributes of students.

https://github.com/tanaya2005/MLDL-lab/blob/main/Experiment-%201/expanded_student_performance.csv

2. Dataset Description

The dataset contains academic performance data of students.

- Dataset Characteristics

- Type: Tabular (Numerical)
- Domain: Education / Academic Performance
- Target Variable: Final_Score
- Size: Medium-sized dataset
- Data Quality: Contains missing (null) values

- Feature Description

Feature Name	Description
Hours_Studied	Number of hours studied
Attendance	Attendance percentage
Assignment_Score	Assignment marks
Midterm_Score	Midterm exam marks
Final_Score	Final exam marks (Target)

- Handling Missing Values

Since null values were present in my dataset:

- Numerical columns were filled using **mean imputation**
- This **prevents loss of data** and maintains distribution balance.

3. Mathematical Formulation of the Algorithm

- Mean

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i$$

- Standard Deviation

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

- Min-Max Normalization

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Where:

- y = Final Score
- β_0 = Intercept
- β_i = Coefficients

Loss Function (MSE):

$$MSE = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$$

- Logistic Regression

$$P(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)}}$$

(Sigmoid function converts output to probability between 0 and 1)

4. Algorithm Limitations

- Linear Regression
 - Assumes linear relationship
 - Sensitive to outliers
 - Performs poorly if data is non-linear
- Logistic Regression
 - Only suitable for binary classification
 - Sensitive to multicollinearity
 - Requires sufficient data for stable decision boundary
- General EDA Limitations
 - Visualization interpretation may be subjective
 - Correlation does not imply causation

5. Methodology / Workflow

Step 1: Data Loading: Load dataset using Pandas.

Step 2: Data Cleaning

- Check missing values
- Apply mean imputation

Step 3: Statistical Analysis

- Compute mean, median, std deviation

Step 4: Normalization

- Apply Min-Max scaling

Step 5: Label Creation

Create "Performance" category

Step 6: Visualization

- Line plot
- Histogram
- Scatter plot
- Heatmap
- Boxplot

Step 7: Model Training

- Linear Regression
- Logistic Regression

Step 8: Evaluation

- MSE
- R^2 Score
- Accuracy
- Confusion Matrix
- ROC Curve

Step 9: Hyperparameter Tuning

- Logistic Regression (C, solver)
- Compare model performance

6. Performance Analysis

- Linear Regression
 - MSE indicates prediction error.
 - R^2 score shows proportion of variance explained.
Higher $R^2 \rightarrow$ Better model fit.
- Logistic Regression
 - Accuracy measures classification correctness.
 - Confusion matrix shows TP, TN, FP, FN.
 - ROC curve evaluates classifier performance.
 - AUC closer to 1 indicates better model.
- Observations:
 - Hours studied strongly influences final score.
 - Assignment and midterm scores have high correlation.
 - Logistic model performs well in classifying Pass/Fail.

7. Hyperparameter Tuning

For Logistic Regression:

- **Tuned parameters:**
 - C (Regularization strength)
 - solver

GridSearchCV was used.

- **Effect:**
 - Improved accuracy
 - Reduced overfitting
 - Better ROC-AUC score

CODE

Google Colab Link:

<https://colab.research.google.com/drive/1IwHYRFBgGMAo657udEBYs7QOt7GDuSSL?usp=sharing>

1. Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
```

2. Load Dataset

```
df = pd.read_csv('/content/expanded_student_performance.csv')
print("First 5 rows:")
print(df.head())
```

```
print("\nDataset Shape:", df.shape)
print("\nMissing Values:\n", df.isnull().sum())
```

Output:

```
First 5 rows:
  Hours_Studied  Attendance  Assignment_Score  Midterm_Score  Final_Score
0              1         60.0              55.0            50.0           52
1              2         65.0              58.0            55.0           57
2              3         70.0              60.0            58.0           60
3              4         75.0              65.0            62.0           64
4              5         80.0              68.0            65.0           68

Dataset Shape: (100, 5)

Missing Values:
Hours_Studied      0
Attendance         11
Assignment_Score   12
Midterm_Score     19
Final_Score        0
dtype: int64
```

3. Handle Missing Values

```
# Fill numerical missing values using mean
df.fillna(df.mean(), inplace=True)

print("\nMissing Values After Handling:\n", df.isnull().sum())
```

Output:

```
Missing Values After Handling:
Hours_Studied      0
Attendance          0
Assignment_Score    0
Midterm_Score       0
Final_Score         0
dtype: int64
```

4. NumPy Statistical Analysis

```
final_scores = df['Final_Score'].to_numpy()

print("Mean:", np.mean(final_scores))
print("Median:", np.median(final_scores))
print("Standard Deviation:", np.std(final_scores))
```

Output:

```
Mean: 76.12
Median: 77.0
Standard Deviation: 11.458865563396754
```

5. Min-Max Normalization

```
min_val = np.min(final_scores)
max_val = np.max(final_scores)

normalized = (final_scores - min_val) / (max_val - min_val)

print("First 5 Normalized Values:")
print(normalized[:5])
```

Output:

```
First 5 Normalized Values:
[0.          0.10416667 0.16666667 0.25          0.33333333]
```

6. Create Performance Label

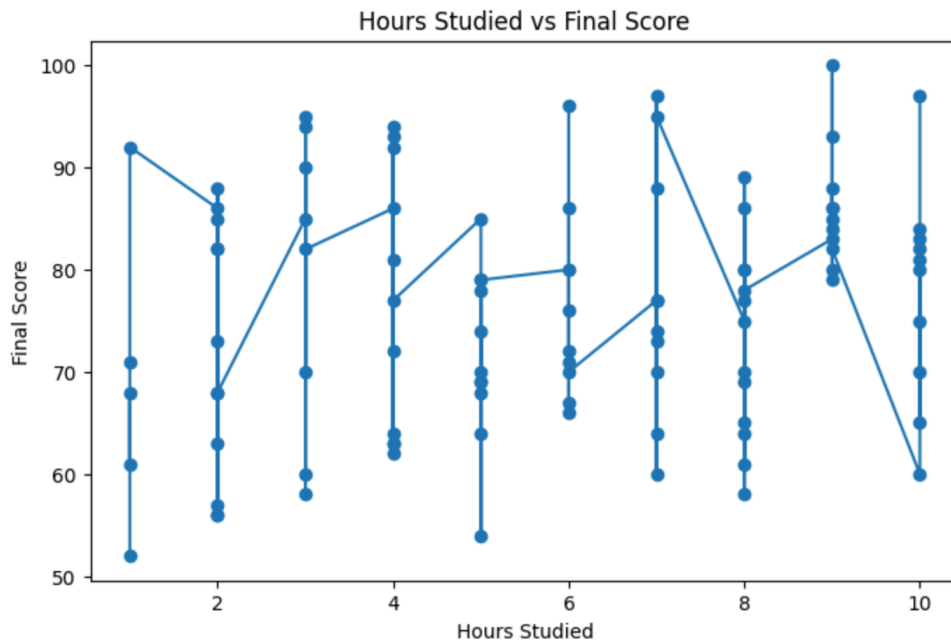
```
def get_performance(score):  
    if score >= 80:  
        return 'Excellent'  
    elif score >= 70:  
        return 'Good'  
    elif score >= 60:  
        return 'Average'  
    else:  
        return 'Low'  
df['Performance'] = df['Final_Score'].apply(get_performance)
```

7. Matplotlib Visualizations

Line Plot

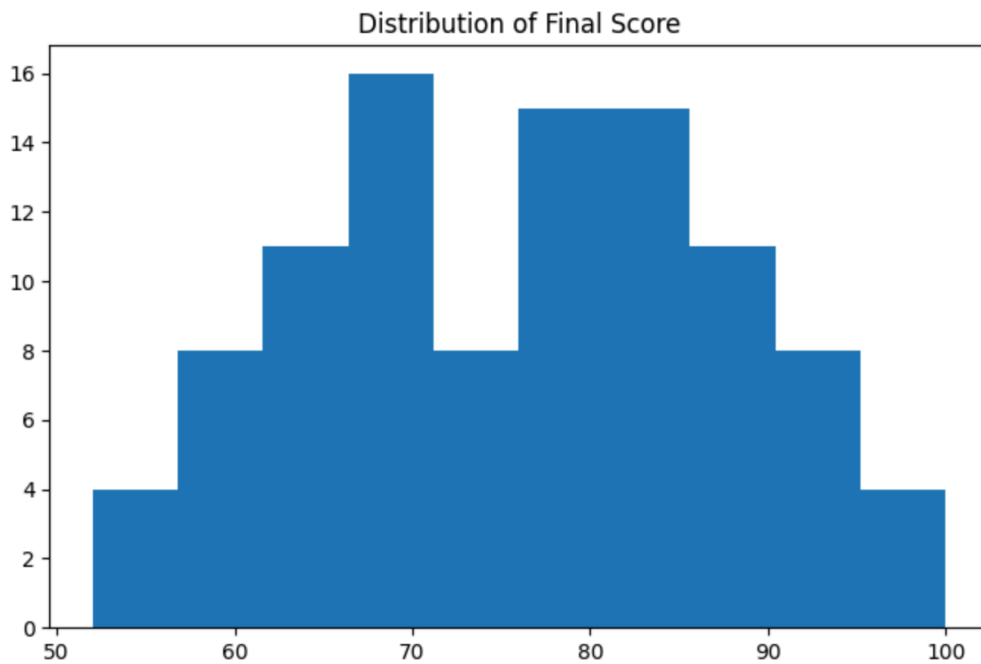
```
df_sorted = df.sort_values('Hours_Studied')
```

```
plt.figure(figsize=(8,5))  
plt.plot(df_sorted['Hours_Studied'], df_sorted['Final_Score'], marker='o')  
plt.title("Hours Studied vs Final Score")  
plt.xlabel("Hours Studied")  
plt.ylabel("Final Score")  
plt.show()
```



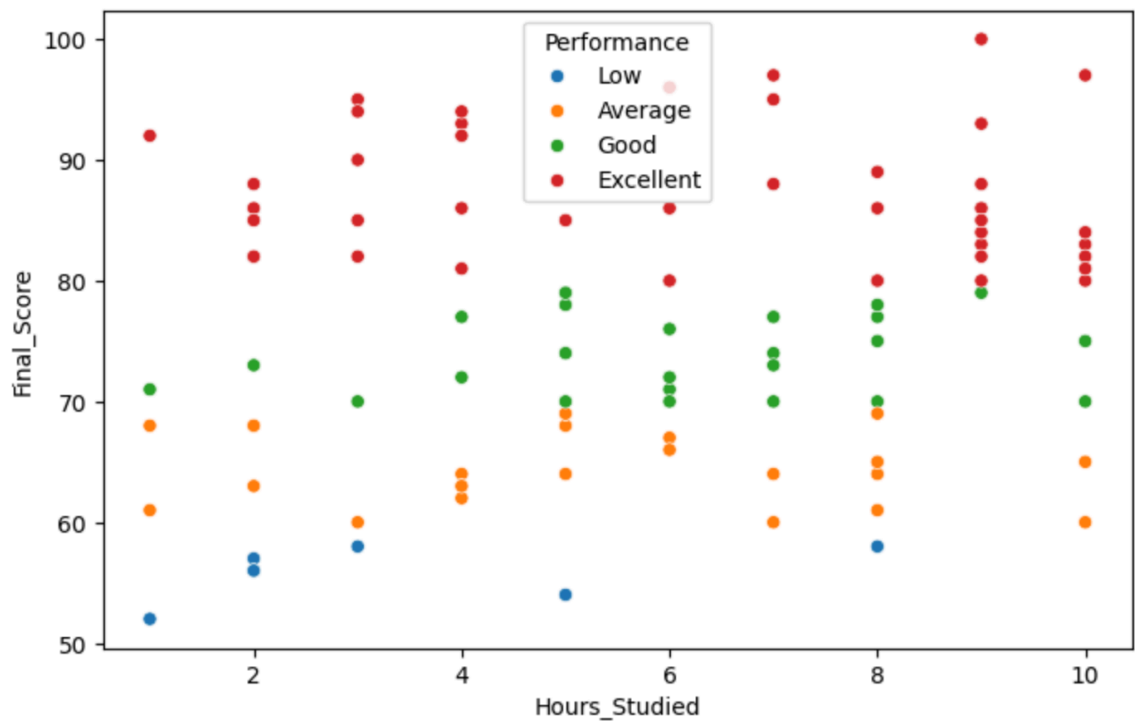
Histogram

```
plt.figure(figsize=(8,5))  
plt.hist(df['Final_Score'], bins=10)  
plt.title("Distribution of Final Score")  
plt.show()
```

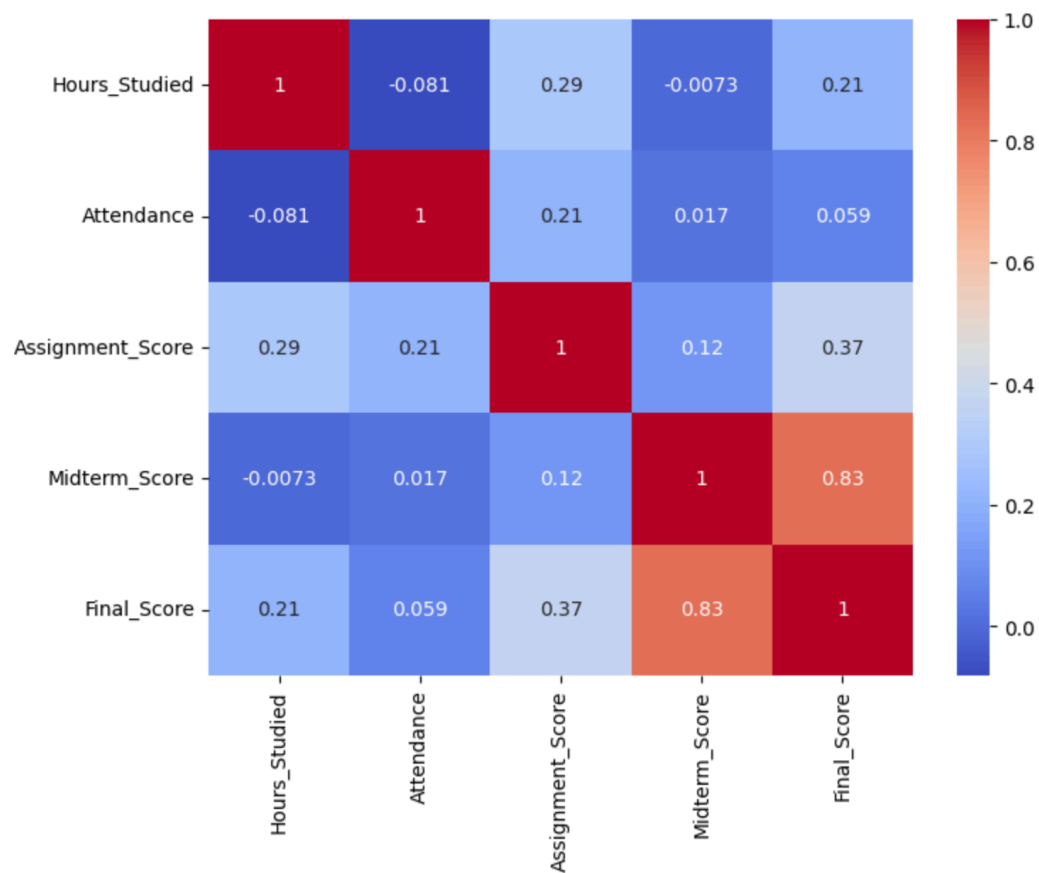


8. Seaborn Visualizations

```
# Scatter Plot
plt.figure(figsize=(8,5))
sns.scatterplot(data=df, x='Hours_Studied', y='Final_Score', hue='Performance')
plt.show()
```



```
# Heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
plt.show()
```

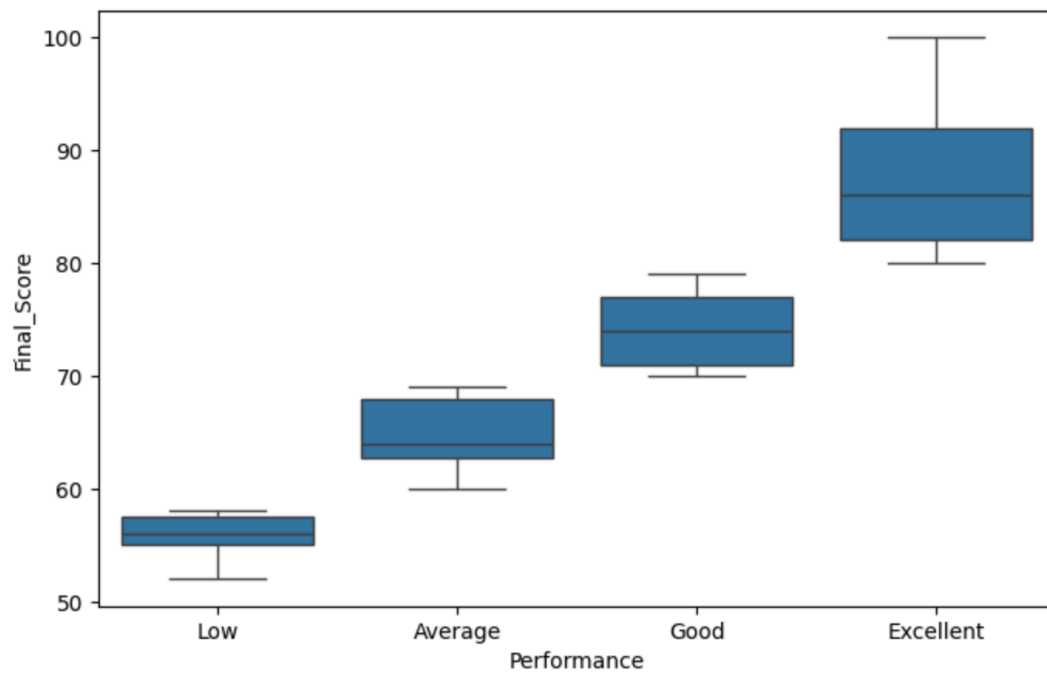



Boxplot

```
plt.figure(figsize=(8,5))
```

```
sns.boxplot(data=df, x='Performance', y='Final_Score')
```

```
plt.show()
```



9. Linear Regression

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
lin_model = LinearRegression()
lin_model.fit(X_train, y_train)
```

```
y_pred = lin_model.predict(X_test)
```

```
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

Output:

```
MSE: 35.62314724702965
R2 Score: 0.7102133797107029
```

10. Logistic Regression

```
df['Pass'] = (df['Final_Score'] >= 70).astype(int)
```

```
X_log = X
y_log = df['Pass']
```

```
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(
    X_log, y_log, test_size=0.3, random_state=42)
```

```
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_log, y_train_log)
```

```
y_pred_log = log_model.predict(X_test_log)
```

```
print("Accuracy:", accuracy_score(y_test_log, y_pred_log))
print("\nConfusion Matrix:\n", confusion_matrix(y_test_log, y_pred_log))
print("\nClassification Report:\n", classification_report(y_test_log, y_pred_log))
```

Accuracy: 0.8666666666666667

Confusion Matrix:

```
[[ 6  3]
 [ 1 20]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.86	0.67	0.75	9
1	0.87	0.95	0.91	21
accuracy			0.87	30
macro avg	0.86	0.81	0.83	30
weighted avg	0.87	0.87	0.86	30

11.ROC Curve

```
y_probs = log_model.predict_proba(X_test_log)[: , 1]
```

```
fpr, tpr, thresholds = roc_curve(y_test_log, y_probs)
```

```
roc_auc = auc(fpr, tpr)
```

```
plt.figure(figsize=(6,4))
```

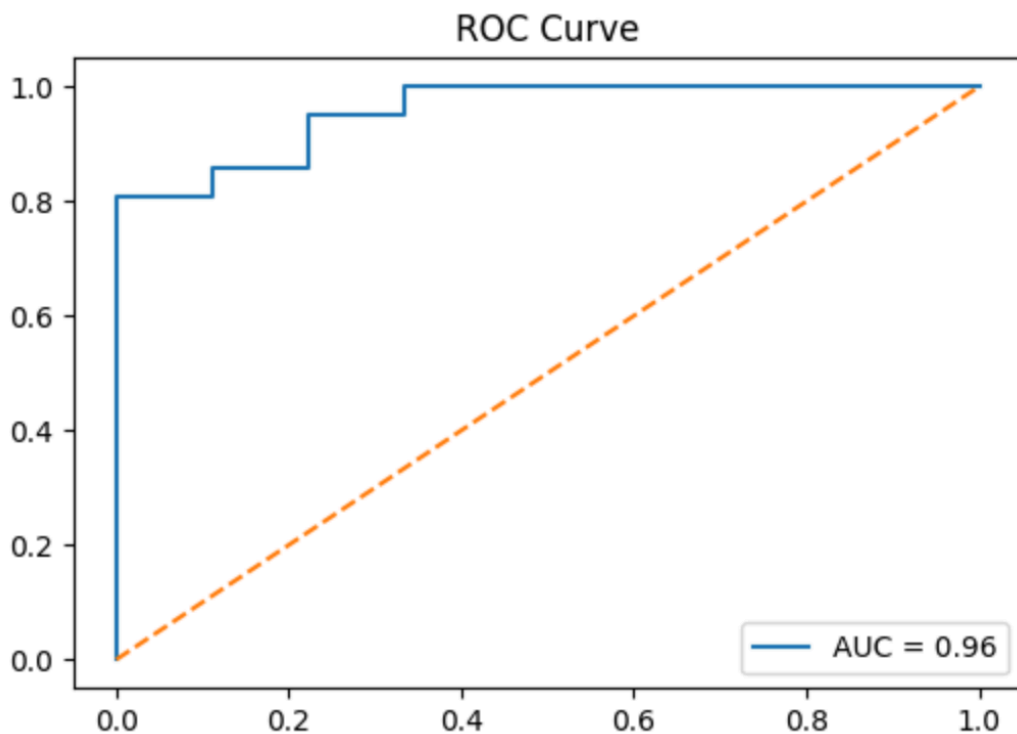
```
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
```

```
plt.plot([0,1],[0,1], '--')
```

```
plt.legend()
```

```
plt.title("ROC Curve")
```

```
plt.show()
```



12. Hyperparameter Tuning

```
param_grid = {  
    'C': [0.01, 0.1, 1, 10],  
    'solver': ['lbfgs', 'liblinear']  
}
```

```
grid = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5)  
grid.fit(X_train_log, y_train_log)
```

```
print("Best Parameters:", grid.best_params_)  
print("Best Score:", grid.best_score_)
```

```
Best Parameters: {'C': 10, 'solver': 'liblinear'}  
Best Score: 0.8428571428571429
```

CONCLUSION

This experiment demonstrated complete ML pipeline development including data cleaning, visualization, regression modeling, classification, evaluation, and hyperparameter tuning. Missing value handling improved data quality, while visualization revealed strong correlation between study hours and academic performance. Logistic regression successfully classified pass/fail students with good accuracy.