Experiment - 0

Aim

To understand data handling, visualization, and exploratory data analysis using NumPy, Pandas, Matplotlib, and Seaborn for Machine Learning applications.

---

1. Dataset Source

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

Dataset Source Link:
https://github.com/rakshit533/MLDL-Lab/blob/main/Datasets/student_performance.csv

---

2. Dataset Description

The dataset consists of student academic performance records with the following features:

Feature Name	Description
Hours_Studied	Number of hours a student studied
Attendance	Attendance percentage

Assignment_Score	Assignment marks
Midterm_Score	Midterm examination marks
Final_Score	Final examination marks (Target variable)

Dataset Characteristics:

- Type: Tabular, numerical
- Size: Medium-sized dataset (suitable for EDA)
- Target Variable: `Final_Score`
- No missing values detected

This dataset is suitable for **exploratory data analysis**, feature correlation analysis, and data preprocessing for machine learning models.

---

3. Mathematical Formulation of the Algorithm

Although no predictive model is trained in this experiment, **statistical and mathematical operations** form the foundation of machine learning preprocessing.

Mean

$\mu = (1 / N) \Sigma x_i$

where $i = 1$ to N

Median

Middle value of sorted data.

Standard Deviation

$\sigma = \sqrt{(1 / N) \Sigma (x_i - \mu)^2}$

where i = 1 to N

Min-Max Normalization

$x' = (x - x_{min}) / (x_{max} - x_{min})$

Correlation Coefficient

$Cov(X, Y) = (1 / N) \Sigma (X_i - \mu_X)(Y_i - \mu_Y)$

$r = Cov(X, Y) / (\sigma_X \cdot \sigma_Y)$

These mathematical concepts are critical for **feature scaling, variance analysis, and pattern discovery** in machine learning workflows.

---

4. Algorithm Limitations

Since this experiment focuses on EDA rather than predictive modeling, it has the following limitations:

- No prediction or classification capability
- Cannot measure model accuracy or loss
- Results are descriptive, not inferential
- Sensitive to outliers during normalization
- Visualization interpretation may be subjective

EDA must always be followed by **model-based learning** for complete ML pipelines.

---

5. Methodology / Workflow

Step-by-Step Workflow

1. Load dataset using Pandas
2. Convert numerical columns to NumPy arrays
3. Perform statistical analysis (mean, median, standard deviation)
4. Normalize data using Min-Max scaling
5. Label student performance categories
6. Visualize data using Matplotlib and Seaborn
7. Analyze feature relationships using correlation heatmaps

---

6. Performance Analysis

Since no ML model is trained, performance is analyzed using **statistical insights and visual interpretation**:

- Higher study hours generally correspond to higher final scores
- Strong positive correlation observed between:
  - Assignment Score and Final Score
  - Midterm Score and Final Score
- Performance categories clearly separate student outcomes
- Boxplots show score spread and outliers

These insights help in **feature selection** for future ML models.

---

7. Hyperparameter Tuning

Hyperparameter tuning is **not applicable** in this experiment because:

- No machine learning algorithm is trained
- EDA focuses on data understanding rather than optimization

Exercise 1: NumPy Basics

Code

```python
import numpy as np
import pandas as pd

df = pd.read_csv('student_performance.csv')

# 1. Load Final_Score as NumPy array
final_scores = df['Final_Score'].values
print("Final Scores (first 10):", final_scores[:10])

# 2. Compute statistics
mean_score = np.mean(final_scores)
median_score = np.median(final_scores)
std_score = np.std(final_scores)
print(f"\nMean: {mean_score:.2f}")
print(f"Median: {median_score:.2f}")
print(f"Standard Deviation: {std_score:.2f}")

# 3. Min-Max Normalization
min_val = np.min(final_scores)
max_val = np.max(final_scores)
normalized_scores = (final_scores - min_val) / (max_val -
min_val)
print(f"\nOriginal range: [{min_val}, {max_val}]")
print(f"Normalized range: [0.0, 1.0]")
print("Normalized scores (first 10):", normalized_scores[:10])
```

Output

```
Final Scores (first 10): [52 57 60 64 68 71 74 77 79 83]

Mean: 68.95
Median: 70.50
Standard Deviation: 8.71

Original range: [52, 83]
Normalized range: [0.0, 1.0]
Normalized scores (first 10): [0.         0.16129032 0.25806452 0.38709677 0.51612903 0.61290323
 0.70967742 0.80645161 0.87096774 1.        ]
```

Exercise 2: Pandas Data Handling

Code

```
# 1. Load and explore dataset
df = pd.read_csv('student_performance.csv')
print("Shape:", df.shape)
print("\nColumns:", df.columns.tolist())
print("\nFirst 5 rows:\n", df.head())

# 2. Check data quality
print("\nData Types:\n", df.dtypes)
print("\nMissing Values:\n", df.isnull().sum())
print("\nStatistics:\n", df.describe())

# 3. Create Performance label
def categorize_performance(score):
    if score ⩾ 80:
        return 'Excellent'
    elif score ⩾ 60:
        return 'Good'
    elif score ⩾ 40:
        return 'Average'
    else:
        return 'Needs Improvement'

df['Performance'] =
df['Final_Score'].apply(categorize_performance)
print("\nPerformance Distribution:\n",
df['Performance'].value_counts())
```

Output

```
First 5 rows:
   Hours_Studied  Attendance  Assignment_Score  Midterm_Score  Final_Score
0              1          60                55             50           52
1              2          65                58             55           57
2              3          70                60             58           60
3              4          75                65             62           64
4              5          80                68             65           68
```

```
Statistics:
       Hours_Studied  Attendance  Assignment_Score  Midterm_Score  Final_Score
count      20.000000   20.000000         20.000000      20.000000    20.000000
mean        5.650000   80.150000         69.750000      65.750000    68.950000
std         2.621269   10.524533          9.025549       8.005755     8.941182
min         1.000000   60.000000         55.000000      50.000000    52.000000
25%         3.750000   71.500000         61.500000      59.500000    62.250000
50%         6.000000   81.000000         70.500000      67.500000    70.500000
75%         8.000000   88.500000         76.500000      71.250000    75.500000
max        10.000000   95.000000         85.000000      78.000000    83.000000

Performance Distribution:
 Performance
Good         14
Average       4
Excellent     2
Name: count, dtype: int64
```

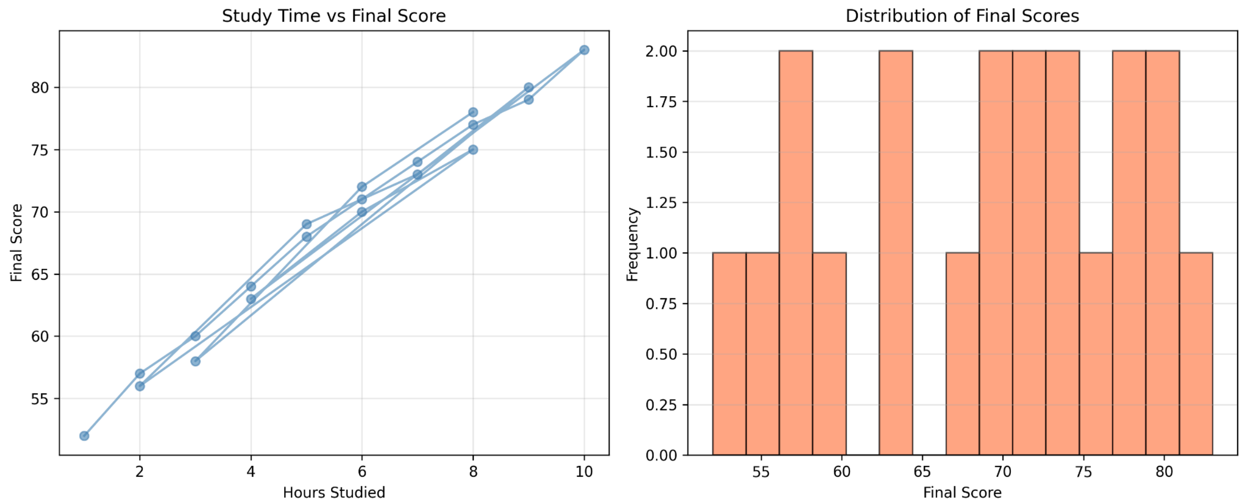
```
Updated DataFrame (first 10 rows):
   Hours_Studied  Attendance  Assignment_Score  Midterm_Score  Final_Score Performance
0              1          60                55             50           52     Average
1              2          65                58             55           57     Average
2              3          70                60             58           60        Good
3              4          75                65             62           64        Good
4              5          80                68             65           68        Good
5              6          85                72             68           71        Good
6              7          90                75             70           74        Good
7              8          95                78             72           77        Good
8              9          88                80             75           79        Good
9             10          92                85             78           83   Excellent
```

Exercise 3: Matplotlib Visualization

Code

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

# 1. Line plot: Hours Studied vs Final Score
plt.subplot(1, 2, 1)
plt.plot(df['Hours_Studied'], df['Final_Score'], 'o-', alpha=0.6)
plt.xlabel('Hours Studied')
plt.ylabel('Final Score')
plt.title('Study Time vs Final Score')
plt.grid(True, alpha=0.3)

# 2. Histogram of Final Scores
plt.subplot(1, 2, 2)
plt.hist(df['Final_Score'], bins=15, edgecolor='black',
alpha=0.7)
plt.xlabel('Final Score')
plt.ylabel('Frequency')
plt.title('Distribution of Final Scores')
plt.grid(True, alpha=0.3, axis='y')

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

Output



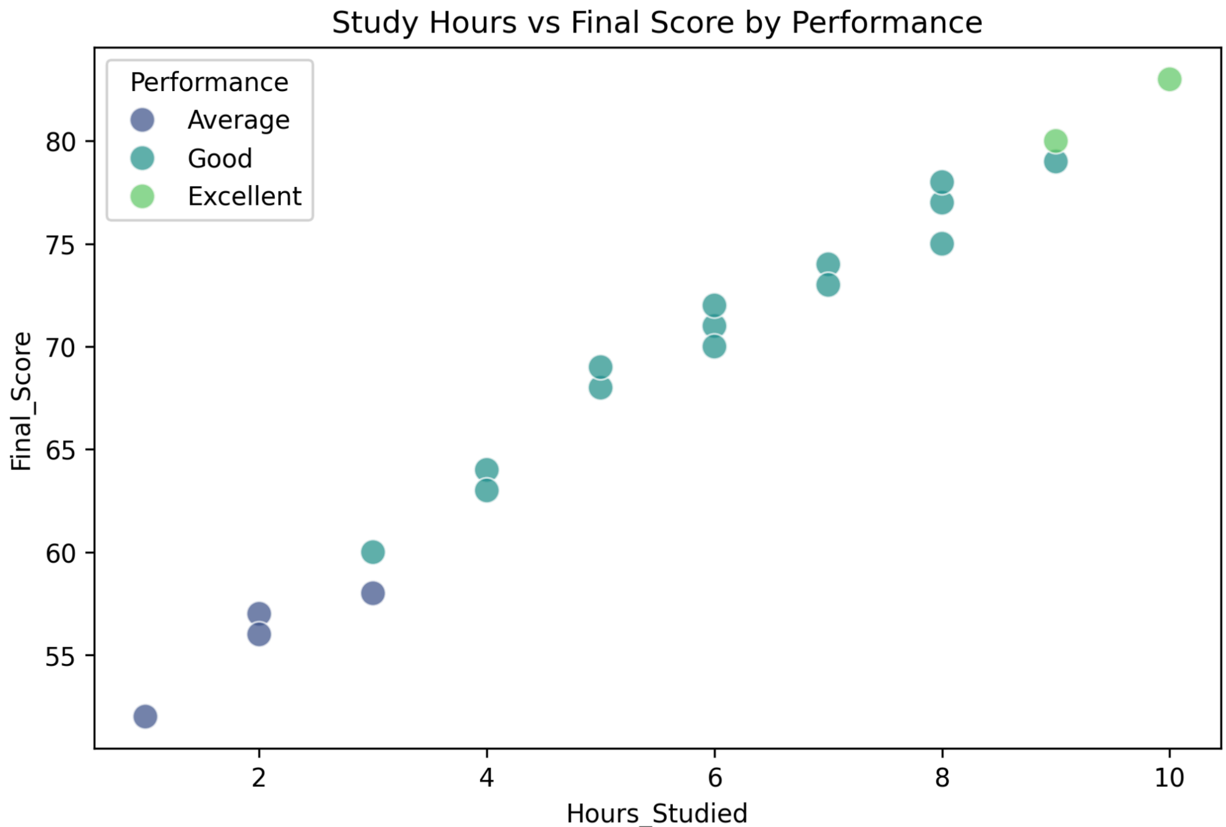
Study Time vs Final Score
Final Score
Hours Studied
Distribution of Final Scores
Frequency
Final Score

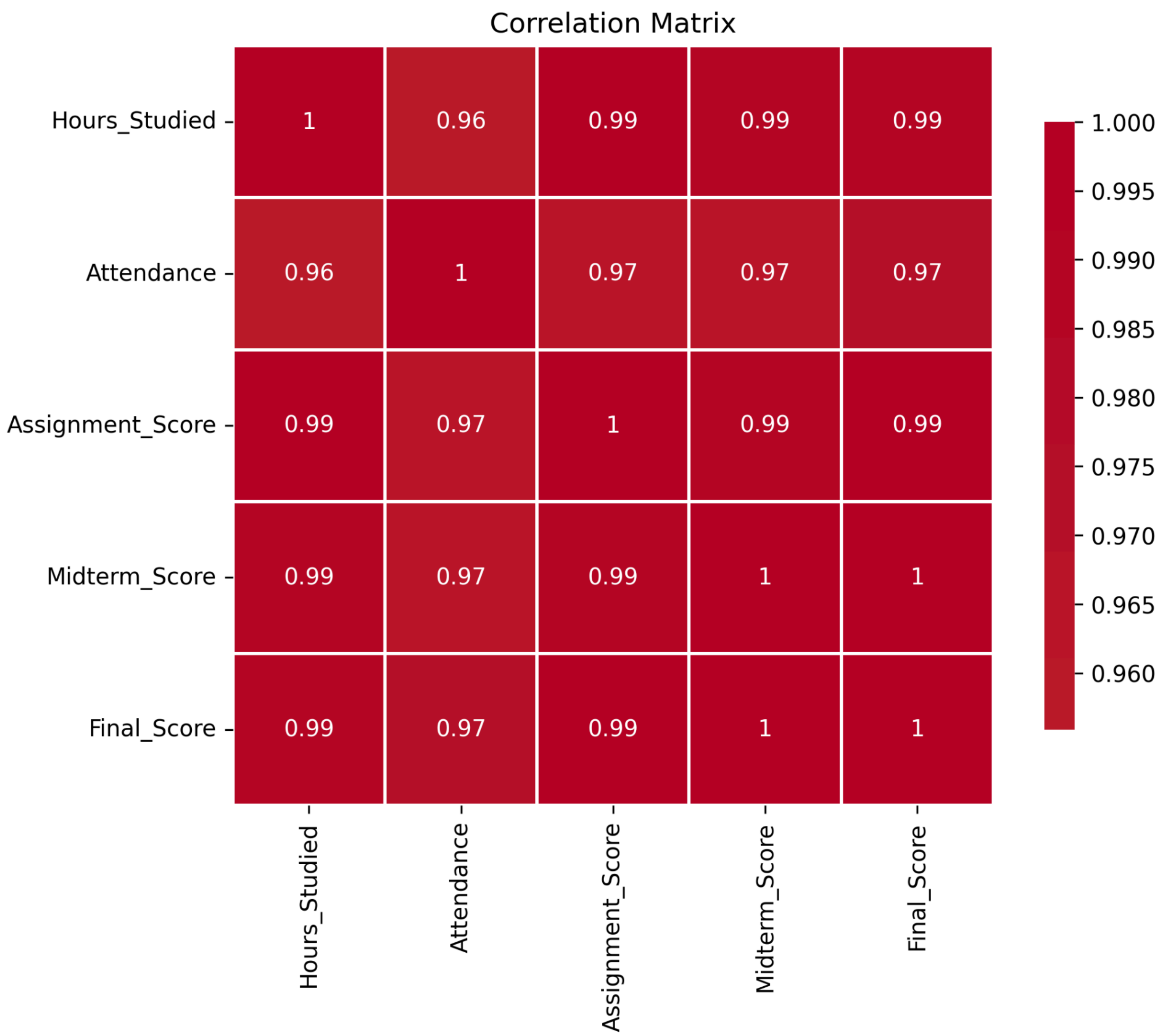
Exercise 4: Seaborn Visualization

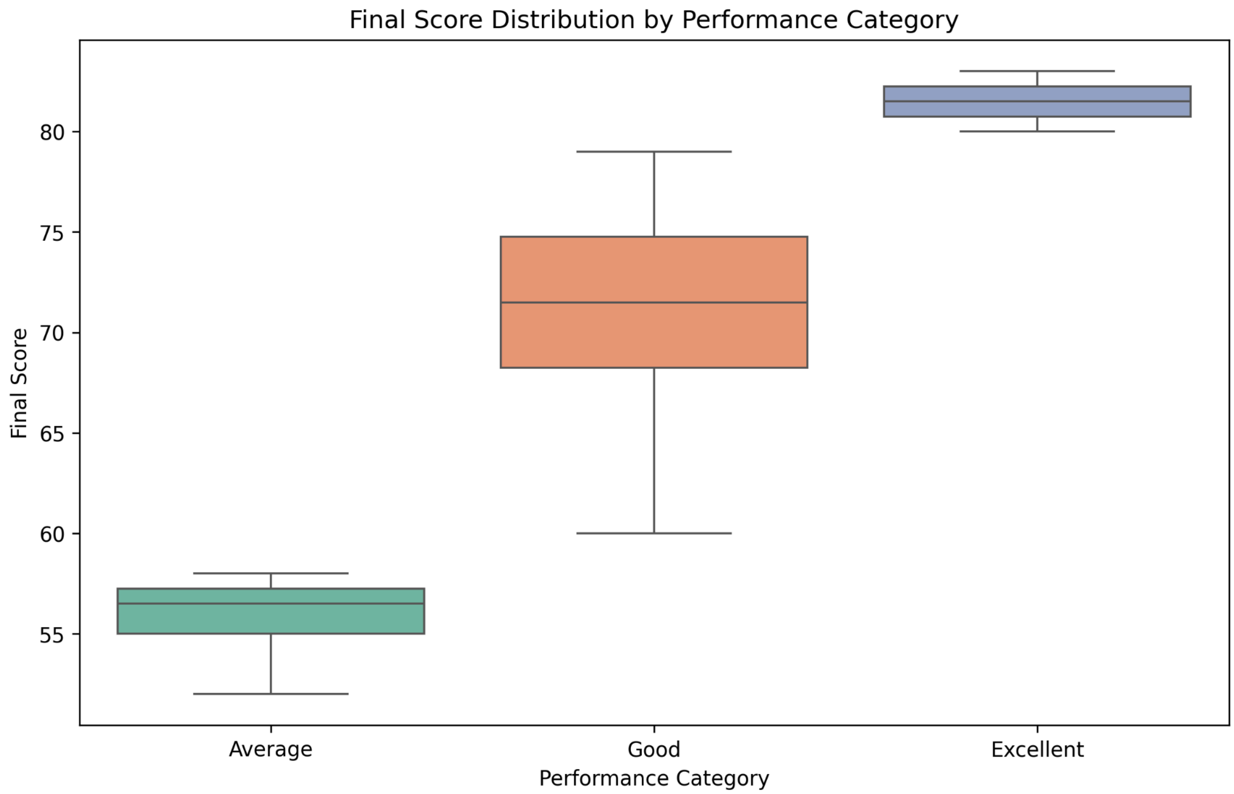
Code

```
import seaborn as sns

# 1. Scatter plot
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='Hours_Studied', y='Final_Score',
                hue='Performance', palette='viridis', s=100)
plt.title('Study Hours vs Final Score by Performance')
plt.show()

# 2. Correlation heatmap
plt.figure(figsize=(8, 6))
corr = df[['Hours_Studied', 'Attendance', 'Assignment_Score',
           'Midterm_Score', 'Final_Score']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', center=0,
square=True)
plt.title('Correlation Matrix')
plt.show()

# 3. Boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Performance', y='Final_Score',
palette='Set2')
plt.title('Final Score Distribution by Performance Category')
plt.show()
```

Output


Study Hours vs Final Score by Performance
Performance
Average
Good
Excellent
80
75
70
65
60
55
Final_Score
2
4
6
8
10
Hours_Studied


Correlation Matrix
Hours_Studied
Attendance
Assignment_Score
Midterm_Score
Final_Score
1
0.96
0.99
0.99
0.99
0.96
1
0.97
0.97
0.97
0.99
0.97
1
0.99
0.99
0.99
0.97
0.99
1
1
0.99
0.97
0.99
1
1
Hours_Studied
Attendance
Assignment_Score
Midterm_Score
Final_Score
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0.960



Conclusion

This lab demonstrated essential Python libraries for machine learning data analysis. NumPy enabled efficient numerical operations and normalization. Pandas facilitated data exploration and transformation. Matplotlib and Seaborn provided powerful visualization capabilities to identify patterns and relationships in the data. These skills are fundamental for effective machine learning preprocessing and exploratory data analysis.