Experiment 1

Name: Chirag Poornamath
Div: D15A
Roll No.: 59

Aim: To understand data handling, visualization, and EDA using Python libraries essential for Machine Learning.

Components:

1. Dataset Source

Source: User Uploaded / Local Repository

Filename: student_performance.csv

2. Dataset Description

The dataset contains quantitative academic information for **20 students** and consists of **five numerical attributes** related to learning effort and examination performance.

Attributes and Their Roles

Column Name	Description
Hours_Studied	Number of hours a student studied for the subject
Attendance	Percentage of classes attended
Assignment_Score	Marks obtained in internal assignments
Midterm_Score	Score obtained in the midterm examination
Final_Score	Score obtained in the final examination

Dataset Characteristics:

- Type: Structured, numerical, tabular
- Rows: 20
- Columns: 5
- No missing values
- Suitable for statistical analysis and exploratory data analysis (EDA)

The dataset is designed to analyze how study habits and academic engagement influence final performance.

3. Mathematical Formulation Used in the Experiment

The experiment applies fundamental statistical measures and data normalization techniques to understand the distribution and scale of the `Final_Score` variable.

a. Mean

The arithmetic average of the data values.

μ=1n∑i=1nxi\mu = \frac{1}{n} \sum_{i=1}^{n} x_iμ=n1i=1∑nxi

b. Median

The central value of the sorted dataset.

c. Standard Deviation

A measure of how spread out the values are around the mean.

σ=1n∑i=1n(xi−μ)2\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2}σ=n1i=1∑n(xi−μ)2

d. Min–Max Normalization

Scales values into the range [0, 1].

x′=x−xminxmax−xminx' = \frac{x - x_{min}}{x_{max} - x_{min}}x′=xmax−xminx−xmin

These formulations help summarize the data and prepare it for further analytical tasks.

4. Limitations of the Approach

Although EDA is essential, this experiment has certain limitations:

- No predictive model is developed.
- Results cannot be generalized beyond this dataset.
- Small dataset size limits statistical robustness.
- Correlation analysis does not establish cause-and-effect relationships.
- The experiment is exploratory and not suitable for automated decision systems.

5. Methodology / Workflow

This experiment follows a systematic process to explore and visualize the dataset.

Step-by-Step Procedure

1. Import NumPy, Pandas, Matplotlib, and Seaborn libraries.
2. Load the CSV file into a Pandas DataFrame.
3. Convert the `Final_Score` column into a NumPy array.
4. Compute statistical measures: mean, median, and standard deviation.
5. Apply Min–Max normalization to scale the `Final_Score`.
6. Create a new categorical column called `Performance` based on score ranges.
7. Generate line plots and histograms using Matplotlib.
8. Create scatter plots, heatmaps, and boxplots using Seaborn.
9. Interpret patterns and relationships from the visualizations.

Workflow Representation

`Data Acquisition → Data Inspection → Statistical Analysis → Normalization → Feature Engineering → Visualization → Interpretation`

6. Performance Analysis (Interpretation Based on EDA)

Since no machine learning model is trained, performance is assessed in terms of **analytical insight**.

• A clear positive trend is observed between `Hours_Studied` and `Final_Score`.
• Students with higher attendance and better assignment scores tend to perform better in the final exam.
• The correlation heatmap shows strong relationships between `Assignment_Score`, `Midterm_Score`, and `Final_Score`.
• The boxplot illustrates distinct score distributions across performance categories such as Excellent, Good, Average, and Below Average.

These results validate that study effort and continuous assessment significantly influence final outcomes.

7. Hyperparameter Tuning

Hyperparameter tuning is not applicable in this experiment because no machine learning algorithm is implemented.
The focus is strictly on data handling, statistical analysis, and visualization.

Theory:

1. **NumPy**

   NumPy is a fundamental package for numerical computation in Python. It provides support for multi-dimensional arrays and mathematical operations. In this experiment, NumPy is used to compute statistical measures and normalize numerical data efficiently.

2. **Pandas**

   Pandas is a powerful data manipulation library that offers data structures such as DataFrames for handling structured data. It is used to load the dataset, inspect its structure, manage columns, and create new derived features.

3. **Matplotlib**

   Matplotlib is a core plotting library used to create static visualizations. In this experiment, it is used to plot line graphs and histograms to study trends and distributions in student performance.

4. **Seaborn**

   Seaborn is a high-level visualization library built on Matplotlib. It simplifies the creation of complex statistical plots. Here, it is used for scatter plots, correlation heatmaps, and boxplots to analyze relationships and group-wise variations.

Code & Output:

1. Uploading & reading .csv file:

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

# Style
plt.style.use('default')
sns.set_palette("husl")

# NumPy
print("\n" + "="*20 + " NumPy " + "="*20)

# 1. Load Final_Score as NumPy array
data = pd.read_csv('student_performance.csv')
final_scores = np.array(data['Final_Score'])
```

2. Using numpy functions to calculate mean, median, mode and normalized the score.

```python
# 2. Mean, median, and standard deviation
mean_score = np.mean(final_scores)
median_score = np.median(final_scores)
std_score = np.std(final_scores)

print(f"\n2. Statistical Measures:")
print(f"   Mean: {mean_score:.2f}")
print(f"   Median: {median_score:.2f}")
print(f"   Standard Deviation: {std_score:.2f}")

# 3. Min-Max normalization
min_score = np.min(final_scores)
max_score = np.max(final_scores)
normalized_scores = (final_scores - min_score) / (max_score - min_score)

print(f"\n3. Min-Max Normalization:")
print(f"   Original range: [{min_score}, {max_score}]")
print(f"   Normalized range: [{np.min(normalized_scores):.3f}, {np.max(normalized_scores):.3f}]")
print(f"   First 5 normalized scores: {normalized_scores[:5]}")
```

```
==================== NumPy Basics ====================

1. Final Scores Array (first 10 values): [52 57 60 64 68 71 74 77 79 83]
   Array shape: (20,)
   Data type: int64

2. Statistical Measures:
   Mean: 68.95
   Median: 70.50
   Standard Deviation: 8.71

3. Min-Max Normalization:
   Original range: [52, 83]
   Normalized range: [0.000, 1.000]
   First 5 normalized scores: [0.         0.16129032 0.25806452 0.38709677 0.51612903]
```

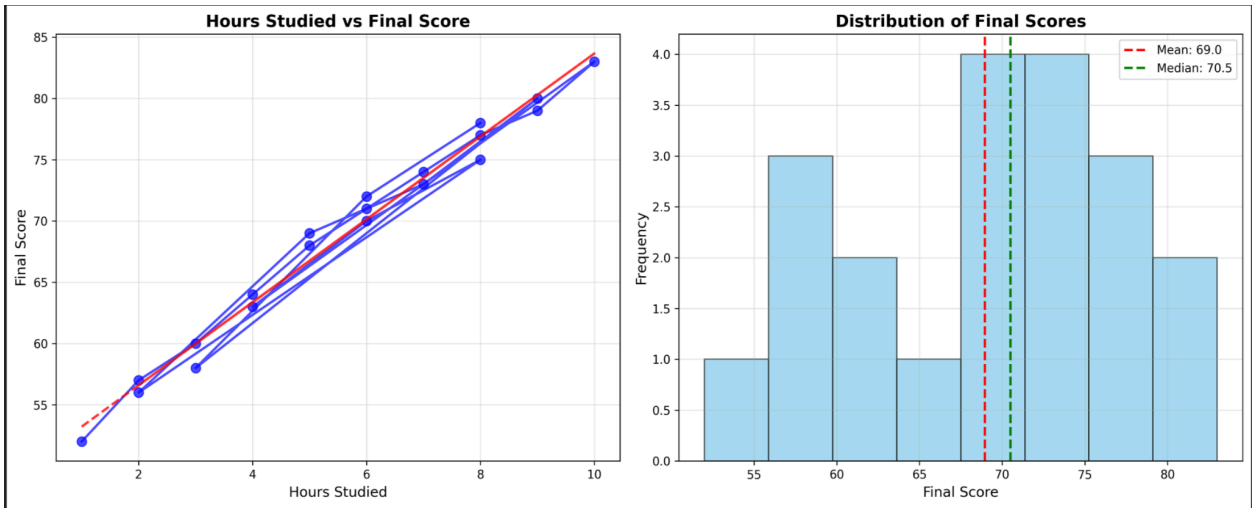
3. Checking shape, columns, and missing values and creating Performance label based on Final_Score.

```python
# 1. Load CSV file using Pandas
print(f"\n1. Dataset loaded successfully!")
print(f"   File: student_performance.csv")

# 2. Check shape, columns, and missing values
print(f"\n2. Dataset Information:")
print(f"   Shape: {data.shape}")
print(f"   Columns: {list(data.columns)}")
print(f"\n   Data Types:")
print(data.dtypes)
print(f"\n   Missing Values:")
print(data.isnull().sum())

print(f"\n   Basic Statistics:")
print(data.describe())

# 3. Create Performance label based on Final_Score
# Define performance categories based on Final_Score
def categorize_performance(score):
    if score >= 75:
        return 'Excellent'
    elif score >= 65:
        return 'Good'
    elif score >= 55:
        return 'Average'
    else:
        return 'Below Average'

data['Performance'] = data['Final_Score'].apply(categorize_performance)

print(f"\n3. Performance Categories:")
print(data['Performance'].value_counts())
print(f"\n   Sample data with Performance labels:")
print(data[['Final_Score', 'Performance']].head(10))
```

```
================= Pandas Data Handling =================

1. Dataset loaded successfully!
   File: student_performance.csv

2. Dataset Information:
   Shape: (20, 5)
   Columns: ['Hours_Studied', 'Attendance', 'Assignment_Score', 'Midterm_Score', 'Final_Score']

   Data Types:
Hours_Studied       int64
Attendance          int64
Assignment_Score    int64
Midterm_Score       int64
Final_Score         int64
dtype: object

   Missing Values:
Hours_Studied       0
Attendance          0
Assignment_Score    0
Midterm_Score       0
Final_Score         0
dtype: int64

   Basic 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

3. Performance Categories:
Performance
Good            7
Average         6
Excellent       6
Below Average   1
Name: count, dtype: int64

   Sample data with Performance labels:
   Final_Score    Performance
0           52  Below Average
1           57        Average
2           60        Average
3           64        Average
4           68           Good
5           71           Good
6           74           Good
7           77      Excellent
8           79      Excellent
9           83      Excellent
```

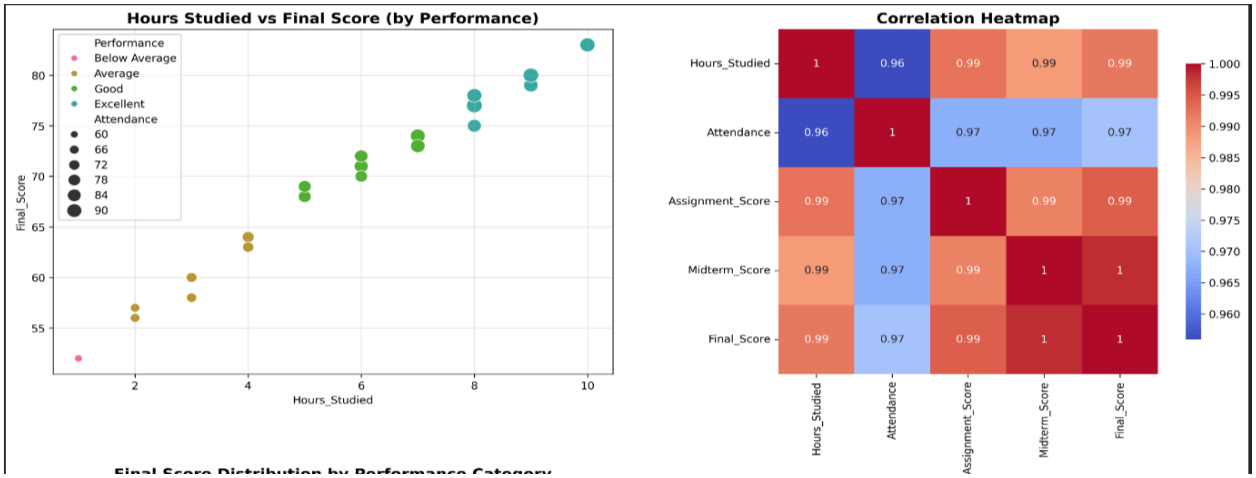
4. Plotting Line plot: Hours_Studied vs Final_Score and Histogram of Final_Score.

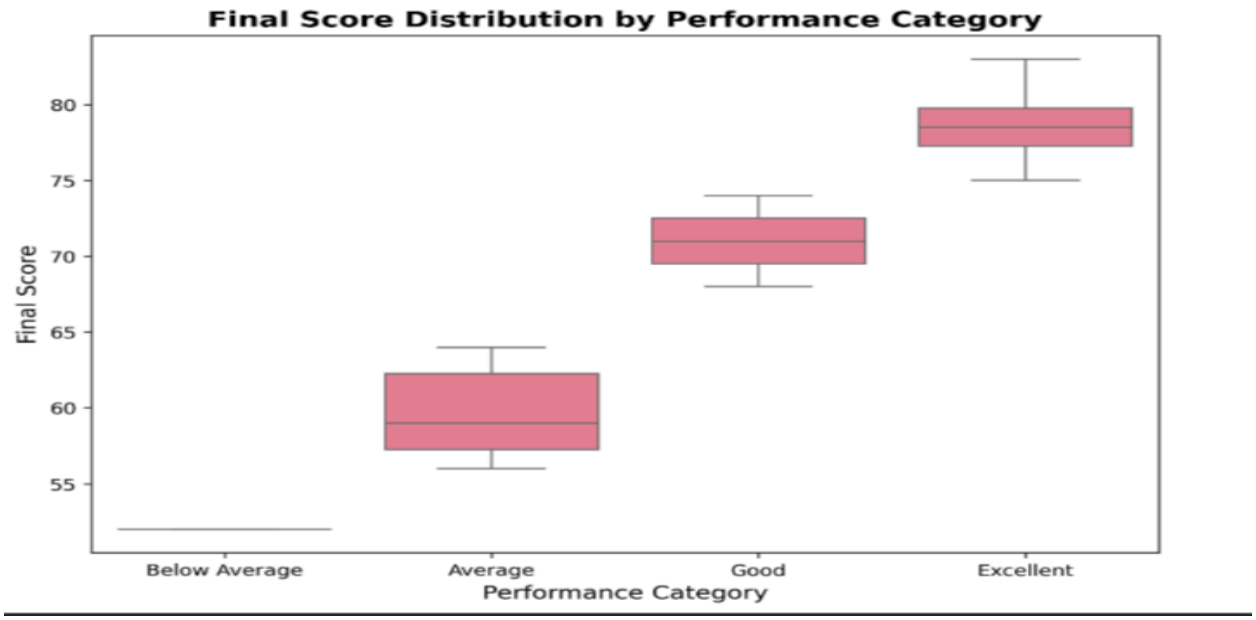
```python
# Matplotlib Visualization
print("\n" + "="*16 + " Matplotlib Visualization " + "="*16)

# Create figure with subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# 1. Line plot: Hours_Studied vs Final_Score
ax1.plot(data['Hours_Studied'], data['Final_Score'], 'bo-', linewidth=2, markersize=8, alpha=0.7)
ax1.set_xlabel('Hours Studied', fontsize=12)
ax1.set_ylabel('Final Score', fontsize=12)
ax1.set_title('Hours Studied vs Final Score', fontsize=14, fontweight='bold')
ax1.grid(True, alpha=0.3)

# Add trend line
z = np.polyfit(data['Hours_Studied'], data['Final_Score'], 1)
p = np.poly1d(z)
ax1.plot(data['Hours_Studied'], p(data['Hours_Studied']), "r--", alpha=0.8, linewidth=2)

# 2. Histogram of Final_Score
ax2.hist(data['Final_Score'], bins=8, edgecolor='black', alpha=0.7, color='skyblue')
ax2.set_xlabel('Final Score', fontsize=12)
ax2.set_ylabel('Frequency', fontsize=12)
ax2.set_title('Distribution of Final Scores', fontsize=14, fontweight='bold')
ax2.axvline(mean_score, color='red', linestyle='--', linewidth=2, label=f'Mean: {mean_score:.1f}')
ax2.axvline(median_score, color='green', linestyle='--', linewidth=2, label=f'Median: {median_score:.1f}')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('matplotlib_plots.png', dpi=300, bbox_inches='tight')
plt.show()

print("Line plot and histogram created and saved as 'matplotlib_plots.png'")
```



5. Plotting scatter plot using seaborn,heatmap for correlation analysis and boxplot for categorical analysis.

```
# Seaborn Visualization
print("\n" + "="*17 + " Seaborn Visualization " + "="*17)

# Create figure with subplots for seaborn plots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))

# 1. Scatter plot using seaborn
sns.scatterplot(data=data, x='Hours_Studied', y='Final_Score',
                hue='Performance', size='Attendance', sizes=(50, 200), ax=ax1)
ax1.set_title('Hours Studied vs Final Score (by Performance)', fontsize=14, fontweight='bold')
ax1.grid(True, alpha=0.3)

# 2. Heatmap for correlation analysis
# correlation matrix
correlation_matrix = data[['Hours_Studied', 'Attendance', 'Assignment_Score',
                           'Midterm_Score', 'Final_Score']].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
            square=True, ax=ax2, cbar_kws={'shrink': 0.8})
ax2.set_title('Correlation Heatmap', fontsize=14, fontweight='bold')

# 3. Boxplot for categorical analysis
sns.boxplot(data=data, x='Performance', y='Final_Score', ax=ax3)
ax3.set_title('Final Score Distribution by Performance Category', fontsize=14, fontweight='bold')
ax3.set_xlabel('Performance Category', fontsize=12)
ax3.set_ylabel('Final Score', fontsize=12)

# Remove the 4th subplot since we're not using it
fig.delaxes(ax4)

plt.tight_layout()
plt.savefig('seaborn_plots.png', dpi=300, bbox_inches='tight')
plt.show()

print("Scatter plot, heatmap, and boxplots created and saved as 'seaborn_plots.png'")
```





Conclusion:

This experiment successfully demonstrated the practical use of NumPy, Pandas, Matplotlib, and Seaborn for data handling and exploratory data analysis.

The experiment reinforced the importance of:

- Understanding data distributions
- Identifying relationships between variables
- Using visualization as a key step before applying machine learning models