D15A / 26

Experiment No: 0

AIM:

To perform data preprocessing and visualization using **NumPy, Pandas, Matplotlib, and Seaborn** in order to understand student performance data and prepare it for machine learning applications.

THEORY:

1. NumPy – Numerical Computation Foundation

NumPy is a core Python library designed for efficient numerical operations on large datasets. It supports multi-dimensional arrays and provides optimized mathematical functions that are faster than traditional Python loops.

In this experiment, NumPy plays a crucial role in **numerical analysis** by converting student scores into array format. This allows direct computation of statistical measures such as **mean, median, and standard deviation**, which help summarize overall student performance.

Additionally, **Min–Max normalization** is applied using NumPy to scale scores between 0 and 1. This step is important in machine learning because features with different value ranges can negatively affect model performance. Normalization ensures fairness by giving equal importance to all features.

2. Pandas – Data Handling and Preprocessing

Pandas is a powerful library used to manage and analyze structured datasets. It introduces the DataFrame, which makes data inspection, cleaning, and transformation simple and intuitive.

In this experiment, Pandas is used to load the student dataset and explore its structure by checking the number of rows, columns, and missing values. Missing values are handled using **mean imputation**, which replaces null entries with the average of that column. This ensures the dataset remains complete and usable.

A new categorical column called **Performance** is created based on students' final scores. This transformation converts numerical results into meaningful labels such as *Average*, *Good*, and *Excellent*, making the data easier to interpret and analyze.

3. Matplotlib – Basic Data Visualization

Matplotlib is a widely used Python library for creating visual representations of data. Visualizations help transform raw numbers into understandable patterns and trends.

In this experiment, Matplotlib is used to plot a **line graph** showing the relationship between hours studied and final scores, helping identify how study time impacts academic performance. A **histogram** is also generated to observe the distribution of final scores, revealing score concentration and spread among students.

These plots form an essential part of **Exploratory Data Analysis (EDA)**.

4. Seaborn – Advanced Statistical Visualization

Seaborn is built on top of Matplotlib and provides enhanced visual aesthetics along with statistical insights.

Here, Seaborn is used to create a **scatter plot** that clearly shows the relationship between study hours and final scores. A **correlation heatmap** is generated to identify the strength of relationships between numerical variables such as attendance, assignment scores, and exam marks.

A **boxplot** is also used to compare final scores across different performance categories, making it easier to observe variations and score ranges among student groups.

5. Exploratory Data Analysis (EDA)

EDA is the process of analyzing data before applying machine learning models. It helps uncover hidden patterns, relationships, and anomalies.

In this experiment, EDA includes:

- Computing statistical summaries
- Handling missing values
- Visualizing feature relationships
- Analyzing correlations between academic metrics

This step ensures that the dataset is clean, meaningful, and ready for future predictive modeling.

IMPLEMENTATION STEPS :

Step 1: Dataset Loading

- Import the required Python libraries.
- Load the student performance dataset using Pandas.
- Verify successful loading by inspecting initial records.

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

[1] ✓ 7.6s

```
df = pd.read_csv("student_performance.csv")

# Display first few rows
print(df.head())
```

[6] ✓ 0.0s

```
   Hours_Studied  Attendance  Assignment_Score  Midterm_Score  Final_Score
0              1          60              55.0           50.0         52.0
1              2          65              58.0           55.0         57.0
2              3          70              60.0           58.0         60.0
3              4          75              65.0           62.0         64.0
4              5          80              68.0           65.0         68.0
```

Step 2: Data Understanding and Cleaning

- Examine dataset dimensions and column names.
- Identify missing values in numerical columns.
- Replace missing values using mean imputation to maintain data consistency.

```
# Shape of dataset
print("Dataset Shape:", df.shape)

# Column names
print("Columns:", df.columns)

# Check missing values
print("Missing Values:\n", df.isnull().sum())
```

✓ 0.0s

```
Dataset Shape: (20, 5)
Columns: Index(['Hours_Studied', 'Attendance', 'Assignment_Score', 'Midterm_Score',
       'Final_Score'],
      dtype='object')
Missing Values:
 Hours_Studied       0
Attendance           0
Assignment_Score     1
Midterm_Score        1
Final_Score          2
dtype: int64
```

```
df.fillna(df.mean(numeric_only=True), inplace=True)

print("Missing Values After Cleaning:\n", df.isnull().sum())
```

[8] ✓ 0.0s

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

Step 3: Numerical Analysis Using NumPy

- Extract the final score column as a NumPy array.
- Calculate statistical measures such as mean, median, and standard deviation.
- Apply Min–Max normalization to scale score values.

```
# Convert Final_Score to NumPy array
final_scores = df["Final_Score"].values

# Statistical calculations
mean_score = np.mean(final_scores)
median_score = np.median(final_scores)
std_dev = np.std(final_scores)

print("Mean:", mean_score)
print("Median:", median_score)
print("Standard Deviation:", std_dev)
```

[9] ✓ 0.0s

```
Mean: 68.77777777777779
Median: 69.5
Standard Deviation: 7.749551958375113
```

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

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

print("Normalized Final Scores:\n", normalized_scores)
```

[10] ✓ 0.0s

```
Normalized Final Scores:
 [0.         0.17857143 0.28571429 0.42857143 0.57142857 0.67857143
 0.78571429 0.89285714 0.96428571 0.59920635 0.39285714 0.64285714
 0.82142857 0.14285714 0.60714286 0.75       1.         0.59920635
 0.71428571 0.92857143]
```

Step 4: Feature Engineering

- Define performance categories based on final score ranges.
- Create a new categorical column representing student performance levels.

```
def performance_label(score):
    if score >= 75:
        return "Excellent"
    elif score >= 60:
        return "Good"
    else:
        return "Average"

df["Performance"] = df["Final_Score"].apply(performance_label)

print(df.head())
```

] ✓ 0.0s

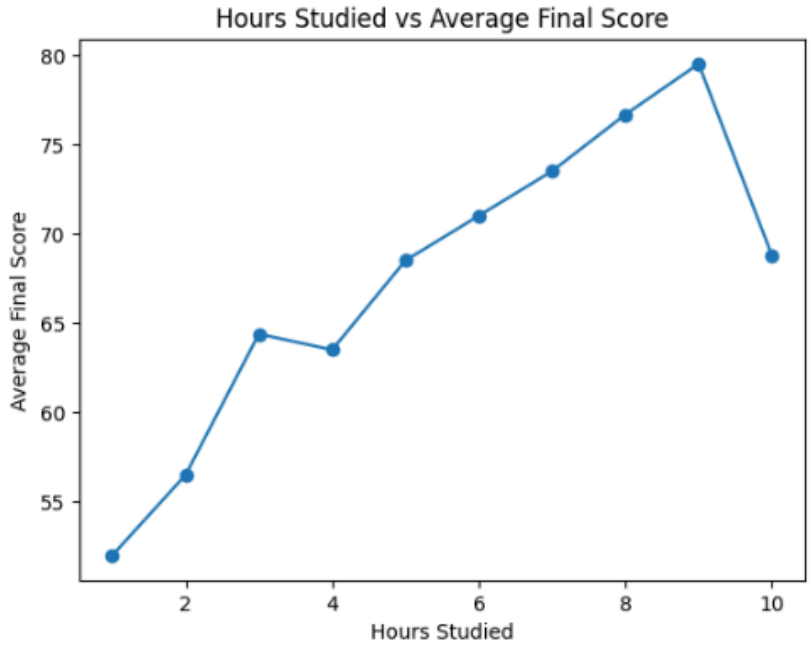
```
   Hours_Studied  Attendance  Assignment_Score  Midterm_Score  Final_Score  \
0              1          60              55.0           50.0         52.0
1              2          65              58.0           55.0         57.0
2              3          70              60.0           58.0         60.0
3              4          75              65.0           62.0         64.0
4              5          80              68.0           65.0         68.0

  Performance
0     Average
1     Average
2        Good
3        Good
4        Good
```

Step 5: Visualization with Matplotlib

- Plot a line graph to analyze the effect of study hours on final scores.
- Create a histogram to visualize score distribution.

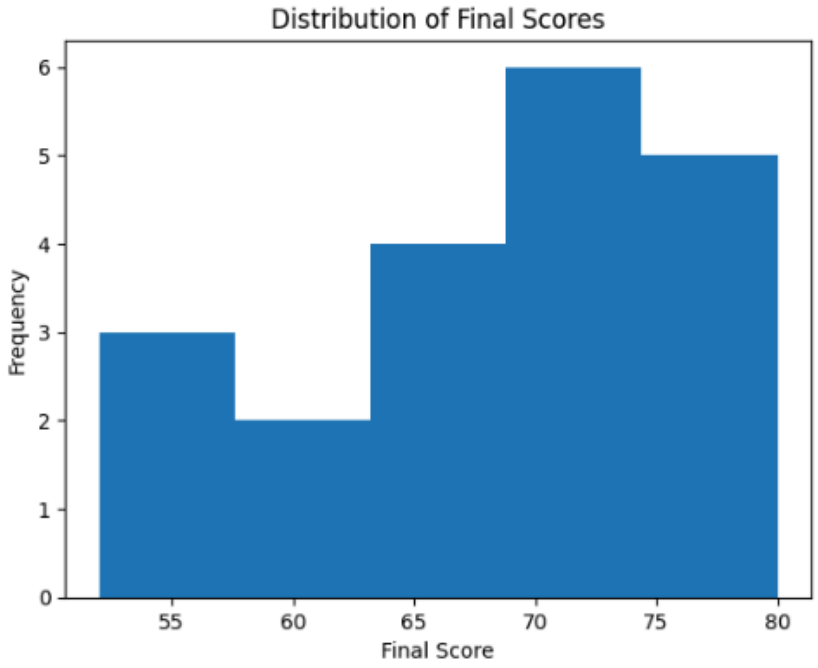
```
df_avg = df.groupby("Hours_Studied")["Final_Score"].mean().reset_index()

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

[19] ✓ 0.2s




```
plt.figure()
plt.hist(df["Final_Score"], bins=5)
plt.xlabel("Final Score")
plt.ylabel("Frequency")
plt.title("Distribution of Final Scores")
plt.show()
```

13] ✓ 0.4s

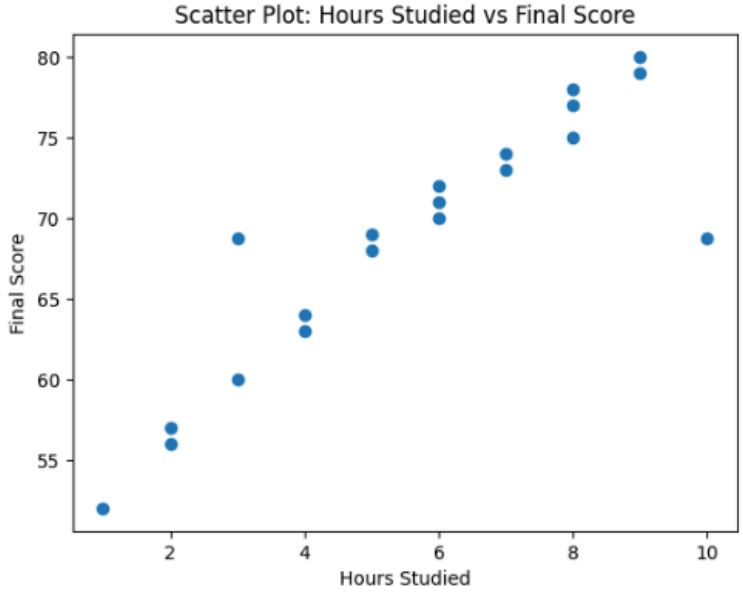


Step 6: Visualization with Seaborn

- Generate a scatter plot for study hours vs final scores.
- Create a heatmap to analyze correlations among all numerical features.
- Plot a boxplot to compare final scores across performance categories.

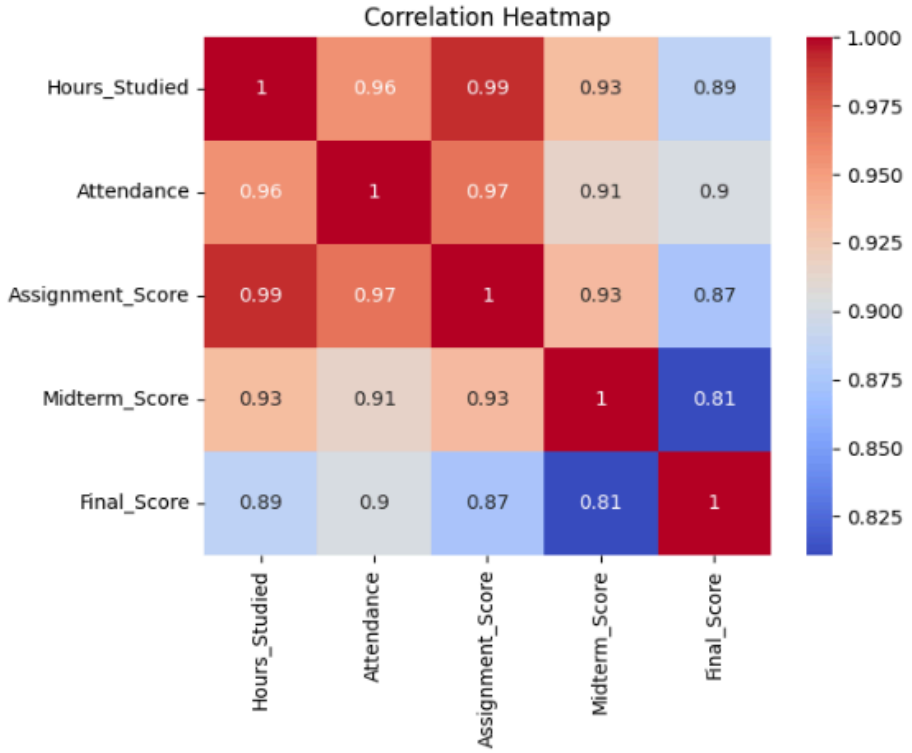
```
plt.figure()
plt.scatter(df["Hours_Studied"], df["Final_Score"])
plt.xlabel("Hours Studied")
plt.ylabel("Final Score")
plt.title("Scatter Plot: Hours Studied vs Final Score")
plt.show()
```

✓ 0.2s



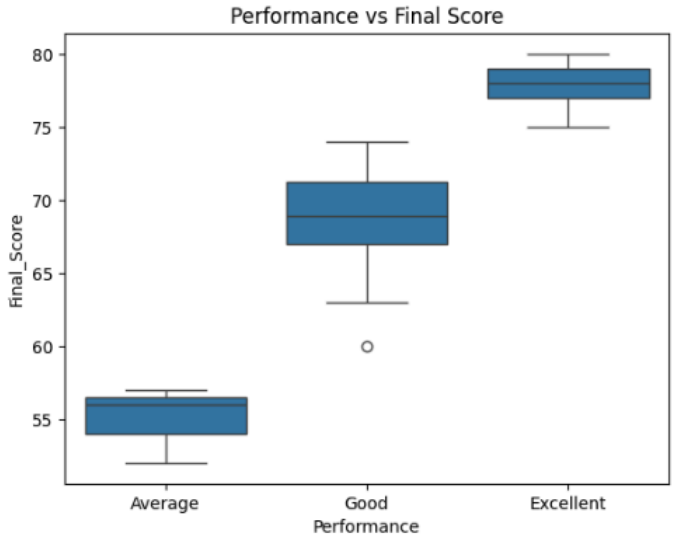
```
plt.figure()
correlation = df.corr(numeric_only=True)
sns.heatmap(correlation, annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

0.4s




```
plt.figure()
sns.boxplot(x="Performance", y="Final_Score", data=df)
plt.title("Performance vs Final Score")
plt.show()
```

[16] 0.2s



Conclusion :

The experiment successfully demonstrated the use of NumPy, Pandas, Matplotlib, and Seaborn for data preprocessing and visualization. Missing values were handled effectively, and statistical analysis was performed using NumPy. Visualizations helped understand the relationship between hours studied and final scores. The experiment highlights the importance of data cleaning and visualization before applying machine learning techniques.