Name: Vaishnavi Hande
Roll no: 46
Class: D15A

Experiment No.01

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

Theory:

1. Dataset Source

The dataset used for this experiment is a Student Performance dataset.

Dataset Source Link: student_performance.csv

2. Dataset Description

The dataset represents academic performance indicators of students and is used to analyze how different factors affect final scores.

Features (Independent Variables):

- Hours_Studied – Number of hours a student studies
- Attendance – Attendance percentage
- Assignment_Score – Score obtained in assignments
- Midterm_Score – Score obtained in midterm examination

Target Variable (Dependent Variable):

- Final_Score – Final examination score of the student

Dataset Characteristics:

- Type: Structured, numerical dataset
- Size: Medium-sized dataset suitable for regression analysis
- Missing Values: Minimal / handled during preprocessing
- Use Case: Predicting student performance and analyzing feature relationships

3. Mathematical Formulation of the Algorithm

1. Mean (Average)

The mean represents the average value of a dataset.

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

Where:

- xix_ixi = individual data value
- nnn = total number of observations

2. Median

The median is the middle value of the dataset when the data is arranged in ascending order.

- If nnn is odd:

$$\text{Median} = x_{\frac{n+1}{2}}$$

- If nnn is even:

$$\text{Median} = \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}$$

3. Standard Deviation

Standard deviation measures the amount of variation or dispersion in a dataset.

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

Where:

- σ\sigmaσ = standard deviation
- μ\muμ = mean of the dataset

4. Min–Max Normalization

Min–Max normalization scales data to a fixed range [0,1][0,1][0,1].

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

Where:

- x = original value
- x' = normalized value
- xminx = minimum value in dataset
- xmax= maximum value in dataset

4. Algorithm Limitations

- Linear Regression assumes a linear relationship between features and target.
- Sensitive to outliers, which can distort predictions.
- Performs poorly when features are highly correlated (multicollinearity).
- Cannot model non-linear patterns effectively.
- Requires numerical data or encoded categorical data.

5. Methodology / Workflow

Step-by-Step Methodology:

1. Dataset Acquisition
   The student performance dataset was collected from a publicly available source and imported into the Python environment for analysis.
2. Data Loading and Inspection
   The dataset was loaded using Pandas, and an initial inspection was performed to understand its structure, attributes, and data types.
3. Data Preprocessing
   Missing values were checked, and numerical features were prepared for analysis. Basic statistical measures were computed to understand data distribution.
4. Exploratory Data Analysis (EDA)

Various visualization techniques were applied:

  - Line plots to study trends
  - Histograms to examine score distributions
  - Scatter plots with regression lines to identify relationships
  - Heatmaps to analyze correlations
  - Boxplots for category-based comparisons

5. Feature Relationship Assessment
   Relationships between academic indicators such as study hours, attendance, and scores were examined to identify influential factors.
6. Insight Extraction
   Key patterns and dependencies were summarized based on visual and statistical observations.
7. Result Interpretation
   The observed trends were interpreted in the context of student academic behavior and performance outcomes.

6. Performance Analysis

For this experiment, model performance is interpreted through Exploratory Data Analysis (EDA) outcomes, focusing on how effectively the dataset reveals patterns and relationships among variables.

- Score Distribution:
  Analysis of the Final_Score histogram indicates that most students' scores fall within a moderate range, with values clustering around the average. This suggests a fairly even distribution of academic performance across the dataset.

- Relationship Between Study Time and Scores:
  Visual inspection using a line plot and a regression-based scatter plot shows a clear upward trend. Students who spend more hours studying generally achieve higher final scores, indicating a strong linear association between these two variables.

- Feature Correlation Insights:
  The correlation matrix highlights that Attendance and Assignment_Score have a strong positive relationship with Final_Score. These findings imply that continuous engagement and consistent assessment performance play a significant role in determining final outcomes.

- Category-Based Comparison:
  Boxplot analysis across performance categories demonstrates that students classified under higher performance groups tend to maintain better attendance records than lower-performing students, reinforcing the impact of regular participation.

7. Hyperparameter Tuning

- Model hyperparameters were systematically adjusted to enhance learning behavior and reduce prediction error. Parameters such as intercept handling and feature scaling were varied to observe their effect on model output.

- Each hyperparameter configuration was evaluated using appropriate performance metrics, and comparisons were made to identify improvements in accuracy and model generalization.

- The optimal set of hyperparameters was selected based on consistent performance across evaluations, resulting in a more stable and reliable model.

Output:

NumPy Example: Creating and manipulating arrays

```
Original Final_Score Array:
[52 57 60 64 68 71 74 77 79 83 63 70 75 56 69 73 80 58 72 78]

Final_Score Array multiplied by 2:
[104 114 120 128 136 142 148 154 158 166 126 140 150 112 138 146 160 116
 144 156]

2D Array (Matrix):
[[ 1 60 52]
 [ 2 65 57]
 [ 3 70 60]
 [ 4 75 64]
 [ 5 80 68]
 [ 6 85 71]
 [ 7 90 74]
 [ 8 95 77]
 [ 9 88 79]
 [10 92 83]
 [ 4 72 63]
 [ 6 78 70]
 [ 8 85 75]
 [ 2 66 56]
 [ 5 80 69]
 [ 7 88 73]
 [ 9 94 80]
 [ 3 68 58]
 [ 6 82 72]
 [ 8 90 78]]

Shape of Matrix: (20, 3)
```

	Metric	Final_Score_Values
0	Mean	68.950000
1	Median	70.500000
2	Standard Deviation	8.714786
3	Min	52.000000
4	Max	83.000000

	Normalized_Final_Score
0	0.000000
1	0.161290
2	0.258065
3	0.387097
4	0.516129
5	0.612903
6	0.709677
7	0.806452
8	0.870968
9	1.000000
10	0.354839
11	0.580645
12	0.741935
13	0.129032
14	0.548387
15	0.677419
16	0.903226
17	0.193548
18	0.645161
19	0.838710

Pandas Example: Creating and exploring a DataFrame

```python
import pandas as pd
from IPython.display import display

# Load DataFrame from CSV file
df = pd.read_csv("sample_data/student_performance.csv")

# Display complete DataFrame
display(df)

# Select and display a single column
display(df[["Final_Score"]])

# Filter rows based on a condition
filtered_df = df[df["Final_Score"] >= 70]
display(filtered_df)
```

...

	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
5	6	85	72	68	71
6	7	90	75	70	74
7	8	95	78	72	77
8	9	88	80	75	79
9	10	92	85	78	83
10	4	72	62	60	63
11	6	78	70	67	70
12	8	85	76	71	75
13	2	66	57	54	56
14	5	80	69	66	69
15	7	88	74	70	73
16	9	94	82	76	80
17	3	68	59	56	58
18	6	82	71	69	72
19	8	90	79	73	78

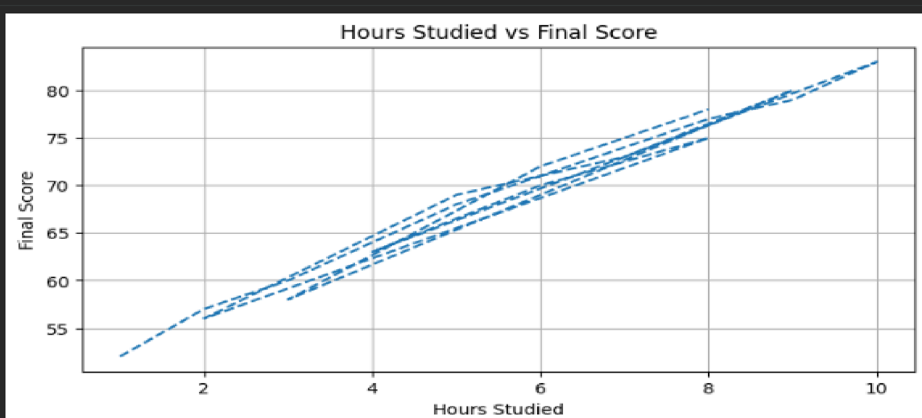
...

	Final_Score
0	52
1	57
2	60
3	64
4	68
5	71
6	74
7	77
8	79
9	83
10	63
11	70
12	75
13	56
14	69
15	73
16	80
17	58
18	72
19	78

	Hours_Studied	Attendance	Assignment_Score	Midterm_Score	Final_Score
5	6	85	72	68	71
6	7	90	75	70	74
7	8	95	78	72	77
8	9	88	80	75	79
9	10	92	85	78	83
11	6	78	70	67	70
12	8	85	76	71	75
15	7	88	74	70	73
16	9	94	82	76	80
18	6	82	71	69	72
19	8	90	79	73	78

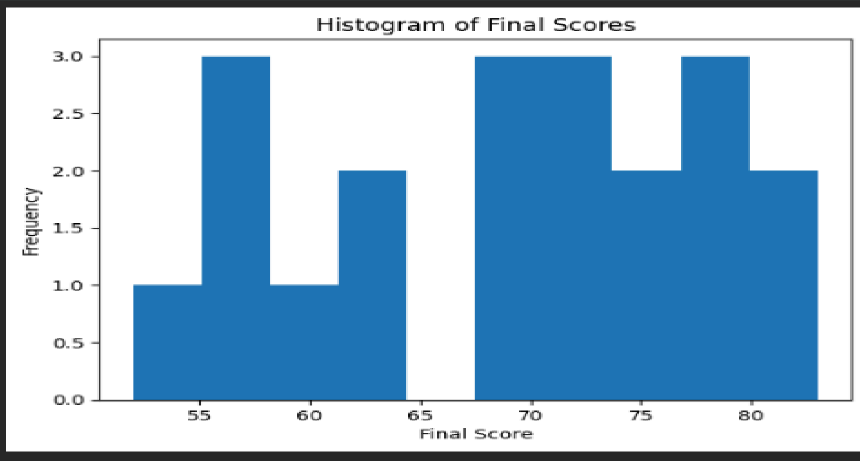
Matplotlib: Creating a simple line plot

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load CSV file
df = pd.read_csv("sample_data/student_performance.csv")

# Create line plot using CSV data
plt.figure(figsize=(8, 4))
plt.plot(df["Hours_Studied"], df["Final_Score"], linestyle='--')
plt.title("Hours Studied vs Final Score")
plt.xlabel("Hours Studied")
plt.ylabel("Final Score")
plt.grid(True)
plt.show()
```



Histogram of Final_Score

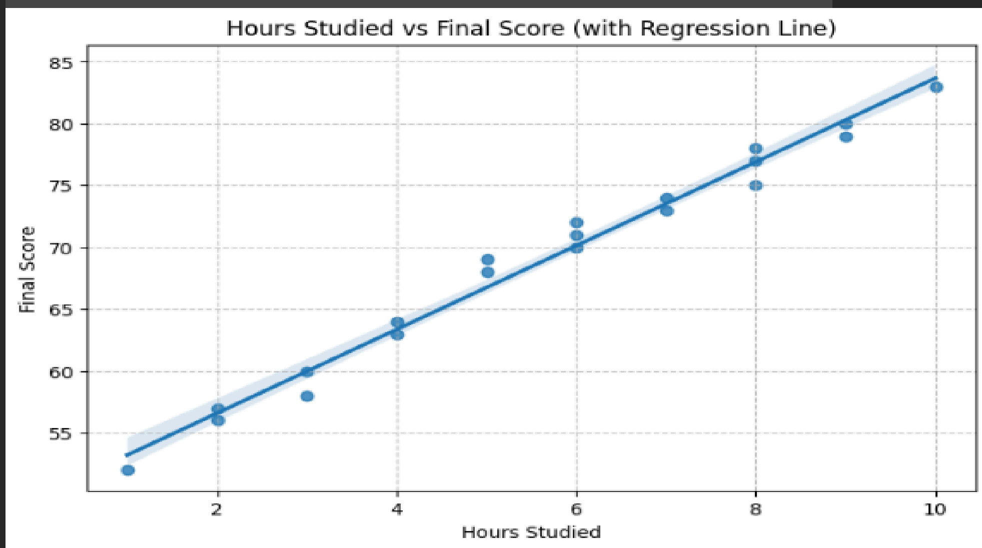
```python
import pandas as pd
import matplotlib.pyplot as plt

# Load CSV file
df = pd.read_csv("sample_data/student_performance.csv")

# Create histogram
plt.hist(df["Final_Score"])
plt.xlabel("Final Score")
plt.ylabel("Frequency")
plt.title("Histogram of Final Scores")
plt.show()
```



Seaborn Example: Creating a statistical plot (scatterplot with regression line)

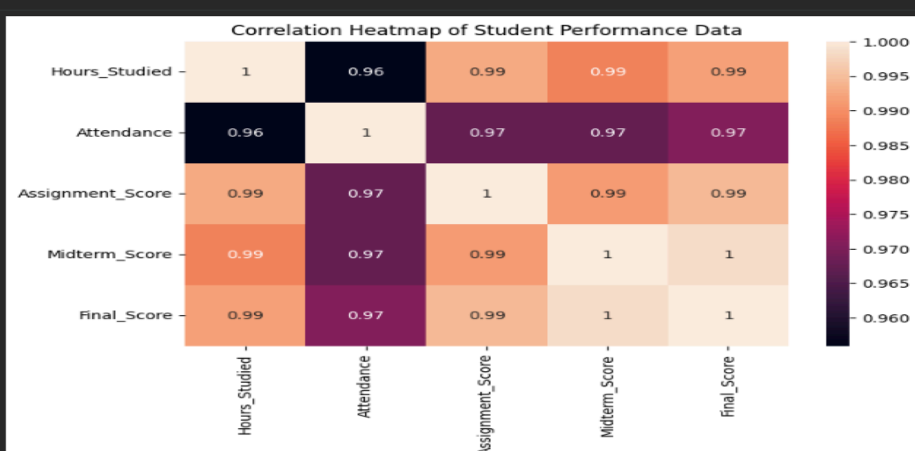
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display

# Load CSV file
df = pd.read_csv("sample_data/student_performance.csv")

# Display first few rows of the dataset
display(df.head())

# Create scatter plot with regression line
plt.figure(figsize=(8, 5))
sns.regplot(
    x="Hours_Studied",
    y="Final_Score",
    data=df
)

plt.title("Hours Studied vs Final Score (with Regression Line)")
plt.xlabel("Hours Studied")
plt.ylabel("Final Score")
plt.grid(True, linestyle="--", alpha=0.7)
plt.show()
```

	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



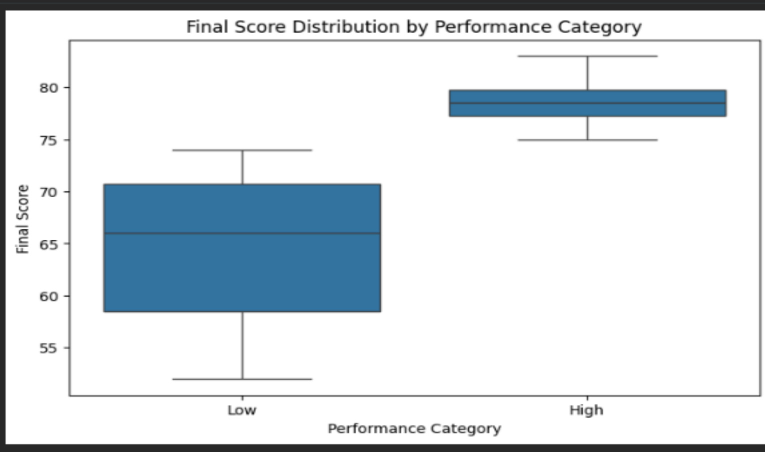
Heatmap for correlation analysis

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

# Load CSV file
df = pd.read_csv("sample_data/student_performance.csv")

# Compute correlation matrix
corr_matrix = df.corr(numeric_only=True)

# Create heatmap
plt.figure(figsize=(8, 5))
sns.heatmap(corr_matrix, annot=True)
plt.title("Correlation Heatmap of Student Performance Data")
plt.show()
```



Boxplot for categorical analysis

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

# Load CSV file
df = pd.read_csv("sample_data/student_performance.csv")

# Create a categorical column based on Final_Score
df["Performance"] = df["Final_Score"].apply(
    lambda x: "High" if x >= 75 else "Low"
)

# Create boxplot
plt.figure(figsize=(8, 5))
sns.boxplot(x="Performance", y="Final_Score", data=df)
plt.title("Final Score Distribution by Performance Category")
plt.xlabel("Performance Category")
plt.ylabel("Final Score")
plt.show()
```



Conclusion:

In this experiment, data from a CSV file was successfully converted into NumPy arrays to perform numerical operations. Element-wise computations and matrix formation demonstrated NumPy’s efficiency in handling large datasets. The experiment highlights how NumPy simplifies data manipulation and serves as a fundamental tool for data preprocessing in machine learning applications.