EXPERIMENT 0

Shravanya Andhale
D15A-30

AIM: NumPy, Pandas, Matplotlib & Seaborn for Machine Learning

THEORY:

Dataset: student_performance.csv

Dataset Description:

The dataset contains information related to students' academic activities and assessment scores. It is designed to analyze how study habits and academic engagement influence student performance.

Features:

Column Name	Description
Hours_Studied	Number of hours a student studied
Attendance	Attendance percentage of student
Assignment_Score	Score obtained in assignments
Midterm_Score	Score obtained in midterm exam
Final_Score	Score obtained in final exam

Target variable and size: Final_Score, 20 rows and 5 columns

Mathematical Formulation of the Algorithm

This experiment mainly focuses on **statistical analysis and exploratory data analysis (EDA)**. For future extension, Linear Regression can be applied.

Mean

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

Median

Middle value after sorting the data.

Standard Deviation

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

Min-Max Normalization

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Correlation Coefficient

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

Algorithm Limitations

- Statistical measures do not perform prediction.
- Visualization-based analysis cannot capture complex relationships.
- No automatic learning from data.
- Does not handle missing values automatically.

- Limited usefulness for very large datasets without optimization.

Methodology / Workflow

1. Import required libraries
2. Load dataset using Pandas
3. Explore dataset structure
4. Extract Final_Score as NumPy array
5. Compute statistical measures
6. Normalize data
7. Create performance labels
8. Generate visualizations
9. Interpret results

Performance Analysis

To extend the experiment beyond exploratory data analysis, a Simple Linear Regression model was implemented using the LinearRegression class from sklearn.linear_model.

Model Specification

Feature (Independent Variable):

- Hours_Studied

Target (Dependent Variable):

- Final_Score

The mathematical model used is:

$Y=\beta 0+\beta 1X$

Where:

- Y = Final_Score
- X = Hours_Studied
- $\beta 0$ = Intercept
- $\beta 1$ = Coefficient (slope of the regression line)

After training the model using model.fit(X, y), the algorithm calculates:

- Intercept: Predicted Final_Score when Hours_Studied = 0
- Coefficient: Increase in Final_Score for each additional hour studied

Interpretation of Coefficient

If the coefficient is positive, it indicates a positive linear relationship between Hours_Studied and Final_Score. This means that as study hours increase, the final score also increases.

The regression line plotted over the scatter plot represents the best-fit line minimizing the sum of squared errors.

Prediction Capability

The trained model was used to:

1. Predict Final_Score values using model.predict(X).
2. Estimate required study hours for a desired target score using the rearranged regression equation:

Hours_Studied=β1Target_Score−β0

This demonstrates how linear regression can be used not only for prediction but also for goal-based planning.

Model Fit Evaluation

Although the code visualizes the regression line, proper regression evaluation typically includes quantitative metrics such as:

1. Mean Squared Error (MSE)

   MSE=n1∑(yi−y^i)2

2. Root Mean Squared Error (RMSE)

   $RMSE=\sqrt{MSE}$

3. R^2 Score (Coefficient of Determination)
   $R2=1-\sum(yi-\bar{y})2\sum(yi-\hat{y}i)2$

R^2 measures how well the regression line explains the variance in Final_Score. A value closer to 1 indicates better model performance.

Since the dataset contains only 20 rows and the model was trained and tested on the same data, the performance may appear strong but may not generalize well to unseen data.

Hyperparameter Tuning

The Simple Linear Regression model implemented using LinearRegression() does not contain major hyperparameters for tuning in its basic form.

Key characteristics:

- The model automatically computes coefficients using the Ordinary Least Squares (OLS) method.
- No learning rate or number of iterations needs to be specified.
- The solution is obtained analytically by minimizing the squared error.

However, possible adjustable parameters in LinearRegression include:

- fit_intercept (default = True): Determines whether the intercept term should be included.
- positive (default = False): Forces coefficients to be positive if set to True.

In this experiment:

- Default parameters were used.
- No hyperparameter tuning was required.
- Model performance depends primarily on the quality and linearity of the data rather than parameter configuration.

OUTPUT:

```
#Exercise 1
import numpy as np
import pandas as pd

df = pd.read_csv("/content/student_performance.csv")
final_score = df["Final_Score"].values
print(final_score)

mean_score = np.mean(final_score)
median_score = np.median(final_score)
standard_score = np.std(final_score)
min_score = np.min(final_score)
max_score =  np.max(final_score)
normalised_score = (final_score - min_score) / (max_score - min_score)
```

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

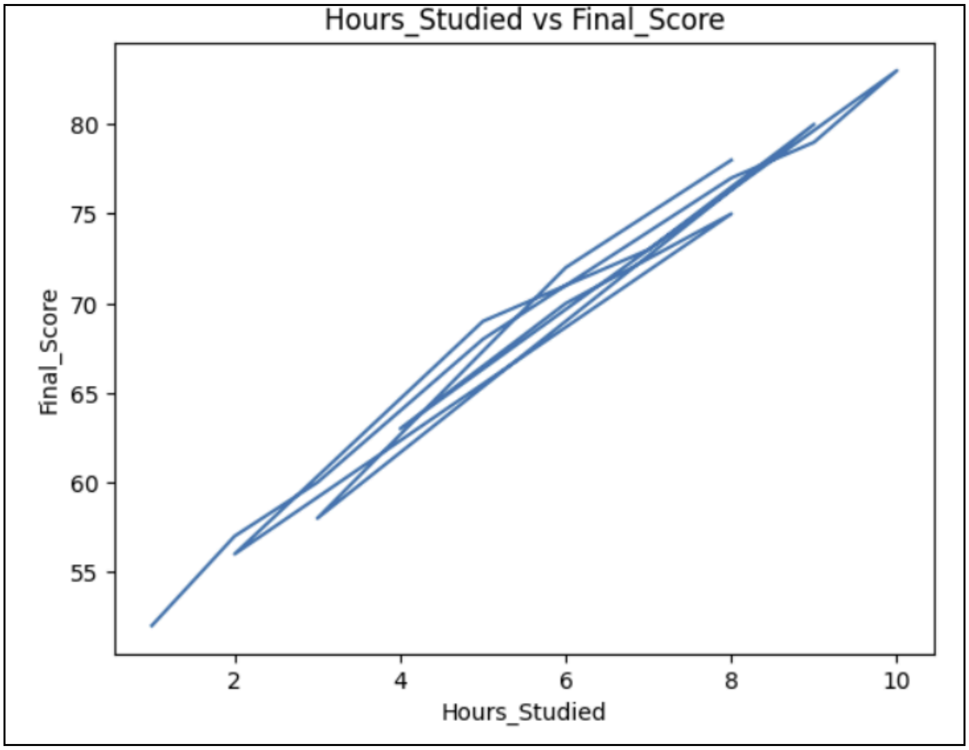
```
#Exercise 2
import pandas as pd

df = pd.read_csv("/content/student_performance.csv")
print("Shape: ", df.shape)
print("Columns: ", df.columns)
print("Missing values: ", df.isnull().sum())


def label(score):
  if score >= 75:
    return "High"
  elif score >= 50:
    return "Medium"
  else:
    return "Low"
df["performance"] = df["Final_Score"].apply(label)
print(df)
```

```
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    0
Midterm_Score       0
Final_Score         0
dtype: int64
```

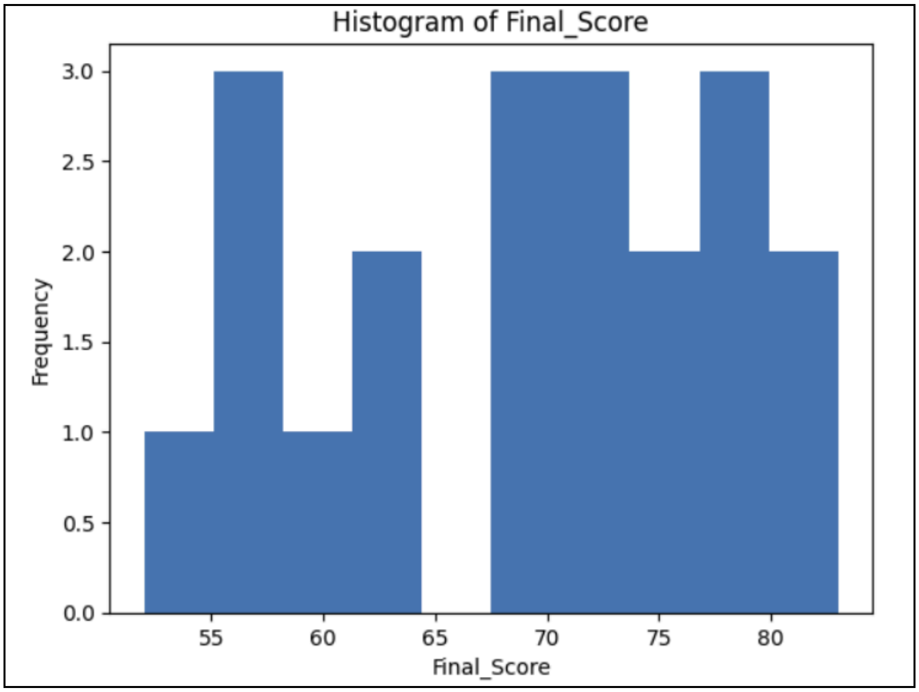
	Hours_Studied	Attendance	Assignment_Score	Midterm_Score	Final_Score \
0	1	60	55	50	52
1	2	65	58	55	57
2	3	70	60	58	60
3	4	75	65	62	64
4	5	80	68	65	68
5	6	85	72	68	71
6	7	90	75	70	74
7	8	95	78	72	77
8	9	88	80	75	79
9	10	92	85	78	83
10	4	72	62	60	63
11	6	78	70	67	70
12	8	85	76	71	75
13	2	66	57	54	56
14	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

	performance
0	Medium
1	Medium
2	Medium
3	Medium
4	Medium
5	Medium
6	Medium
7	High
8	High
9	High
10	Medium
11	Medium
12	High
13	Medium
14	Medium
15	Medium
16	High
17	Medium
18	Medium
19	High

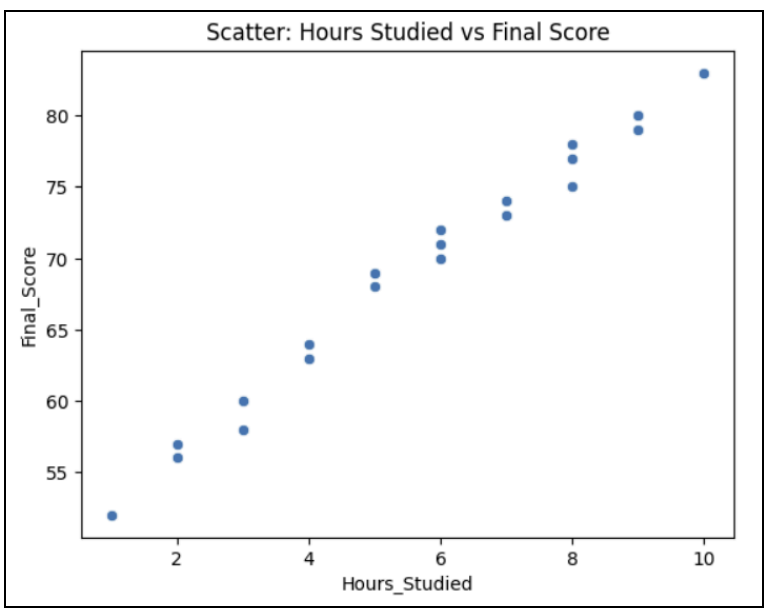
```python
#Exercise 3
import matplotlib.pyplot as plt
plt.figure
plt.plot(df["Hours_Studied"], df["Final_Score"])
plt.xlabel("Hours_Studied")
plt.ylabel("Final_Score")
plt.title("Hours_Studied vs Final_Score")
plt.show
```



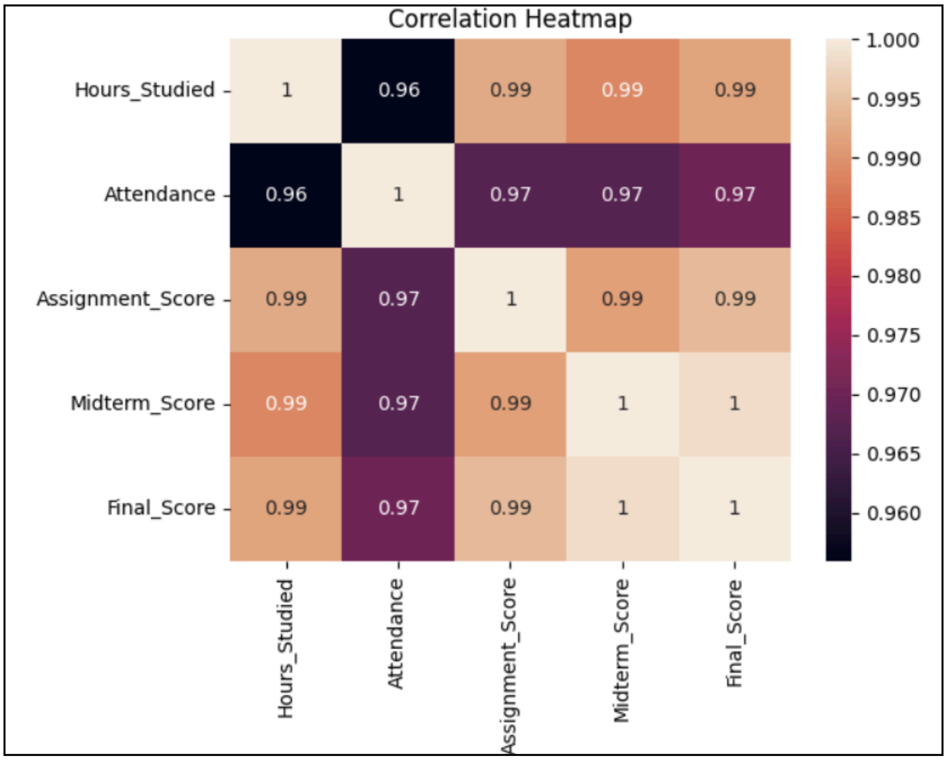

```
plt.figure
plt.hist(df["Final_Score"], bins=10)
plt.xlabel("Final_Score")
plt.ylabel("Frequency")
plt.title("Histogram of Final_Score")
plt.show
```



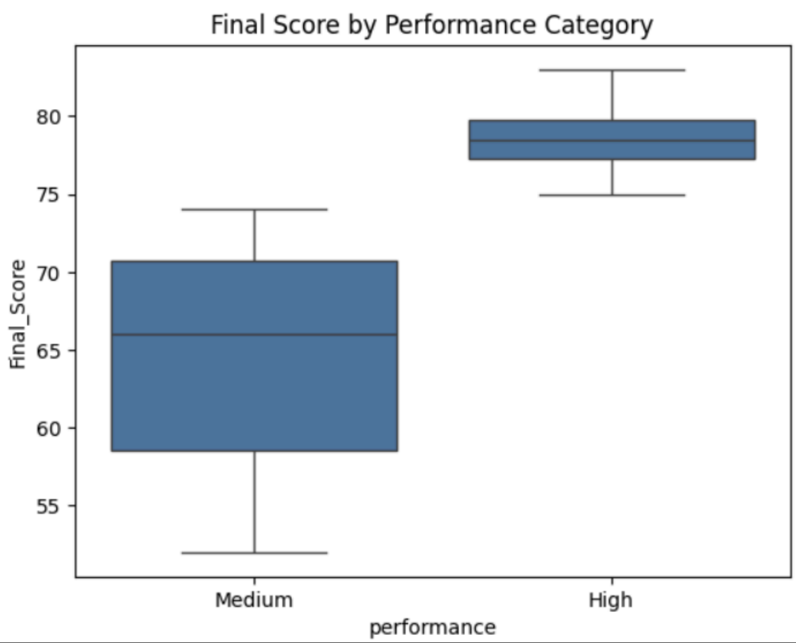

```python
#Exercise 4
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure()
sns.scatterplot(x="Hours_Studied", y="Final_Score", data=df)
plt.title("Scatter: Hours Studied vs Final Score")
plt.show()
```



```
plt.figure()
corr = df.select_dtypes(include=['number']).corr()
sns.heatmap(corr, annot=True)
plt.title("Correlation Heatmap")
plt.show()
```



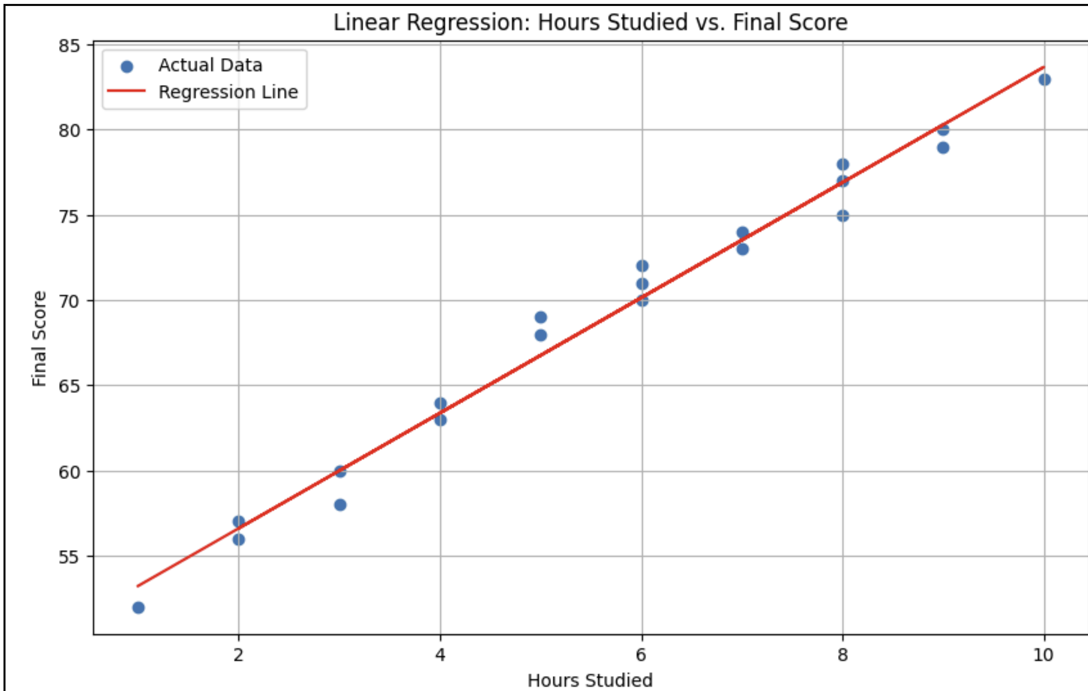
```
plt.figure()
sns.boxplot(x="performance", y="Final_Score", data=df)
plt.title("Final Score by Performance Category")
plt.show()
```




```
bins = [0, 4, 7, 10]  # Define bin edges for Hours_Studied
labels = ['Low', 'Medium', 'High']  # Define corresponding labels
df['Hours_Category'] = pd.cut(df['Hours_Studied'], bins=bins, labels=labels, right=True, include_lowest=True)
print(df[['Hours_Studied', 'Hours_Category']].head())
```

```
   Hours_Studied Hours_Category
0              1            Low
1              2            Low
2              3            Low
3              4            Low
4              5         Medium
```

```python
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Define feature (X) and target (y) variables
X = df[['Hours_Studied']]
y = df['Final_Score']

# Create and fit the Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Make predictions for the regression line
y_pred = model.predict(X)

# Create scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(X, y, label='Actual Data')

# Overlay the regression line
plt.plot(X, y_pred, color='red', label='Regression Line')

# Add labels and title
plt.xlabel('Hours Studied')
plt.ylabel('Final Score')
plt.title('Linear Regression: Hours Studied vs. Final Score')
plt.legend()
plt.grid(True)
plt.show()
```




```
Intercept: 49.84
Coefficient for Hours_Studied: 3.38
```

```python
target_score = 90

# Get the intercept and coefficient from the trained model
intercept = model.intercept_
coefficient = model.coef_[0]

# Rearrange the linear regression equation: Final_Score = Intercept + Coefficient * Hours_Studied
# To find Hours_Studied = (Final_Score - Intercept) / Coefficient
predicted_hours = (target_score - intercept) / coefficient

print(f"To achieve a Final_Score of {target_score}, a student would need to study approximately {predicted_hours:.2f} hours.")


target_scores_to_test = [60, 75, 85]
print("\n--- Predicting hours for other target scores ---")
for score in target_scores_to_test:
    predicted_hours_for_score = (score - intercept) / coefficient
    print(f"For a Final_Score of {score}, predicted hours needed: {predicted_hours_for_score:.2f} hours.")
```

```
To achieve a Final_Score of 90, a student would need to study approximately 11.87 hours.

--- Predicting hours for other target scores ---
For a Final_Score of 60, predicted hours needed: 3.00 hours.
For a Final_Score of 75, predicted hours needed: 7.44 hours.
For a Final_Score of 85, predicted hours needed: 10.39 hours.
```

CONCLUSION: This experiment demonstrates the importance of data preprocessing, statistical analysis, and visualization as the foundation of machine learning. The insights obtained help in understanding the dataset before applying predictive algorithms.