Experiment no 6:- Implementation of Prediction Algorithm (Linear Regression)

Name: Krish Nisar

Div: TY3

Batch: B

Roll no: 41

Subject : Data WareHouse And Mining

Aim: Implementation of Prediction Algorithm (Linear Regression) using Python

Introduction (Theory): Linear Regression is a fundamental supervised machine learning algorithm used for predicting a continuous dependent variable based on one or more independent variables. In its simplest form, known as Simple Linear Regression, it models the relationship between two variables by fitting a straight line to the observed data. The line is defined by the equation $Y = b_0 + b_1 X$, where Y is the predicted value, X is the input, b_0 is the intercept, and b_1 is the slope or coefficient.

The main objective of Linear Regression is to determine the best-fit line that minimizes the error between the predicted and actual values. This is typically achieved using the Least Squares Method, which reduces the Mean Squared Error (MSE). Linear Regression is widely used in forecasting, trend analysis, and real-world applications like predicting sales, prices, or performance scores based on input features.

Procedure:

Import Required Libraries

Import libraries like pandas, numpy, matplotlib, and sklearn for data handling, visualization, and modeling.

Load or Create the Dataset

Prepare your dataset with independent variable(s) (X) and the dependent variable (Y) for prediction.

Visualize the Data (Optional but Recommended)

Plot a scatter graph to observe the relationship between the input and output variables.

Split the Dataset

Divide the dataset into training and testing sets (commonly 80% training and 20% testing) using train_test_split().

Create the Model

Initialize the Linear Regression model using LinearRegression() from sklearn.linear_model.

Train the Model

Fit the model on the training data using .fit(X_train, y_train) to learn the relationship.

Make Predictions

Use .predict(X_test) to predict values on test data and compare them with actual values.

Visualize the Regression Line

Plot the regression line on the graph to see how well it fits the data.

Evaluate the Model

Calculate performance metrics like Mean Squared Error (MSE) and R^2 Score using mean_squared_error() and r2_score().

Make New Predictions {Optional)

Use the trained model to predict outcomes for new input data points.

Program Codes:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2 score
```

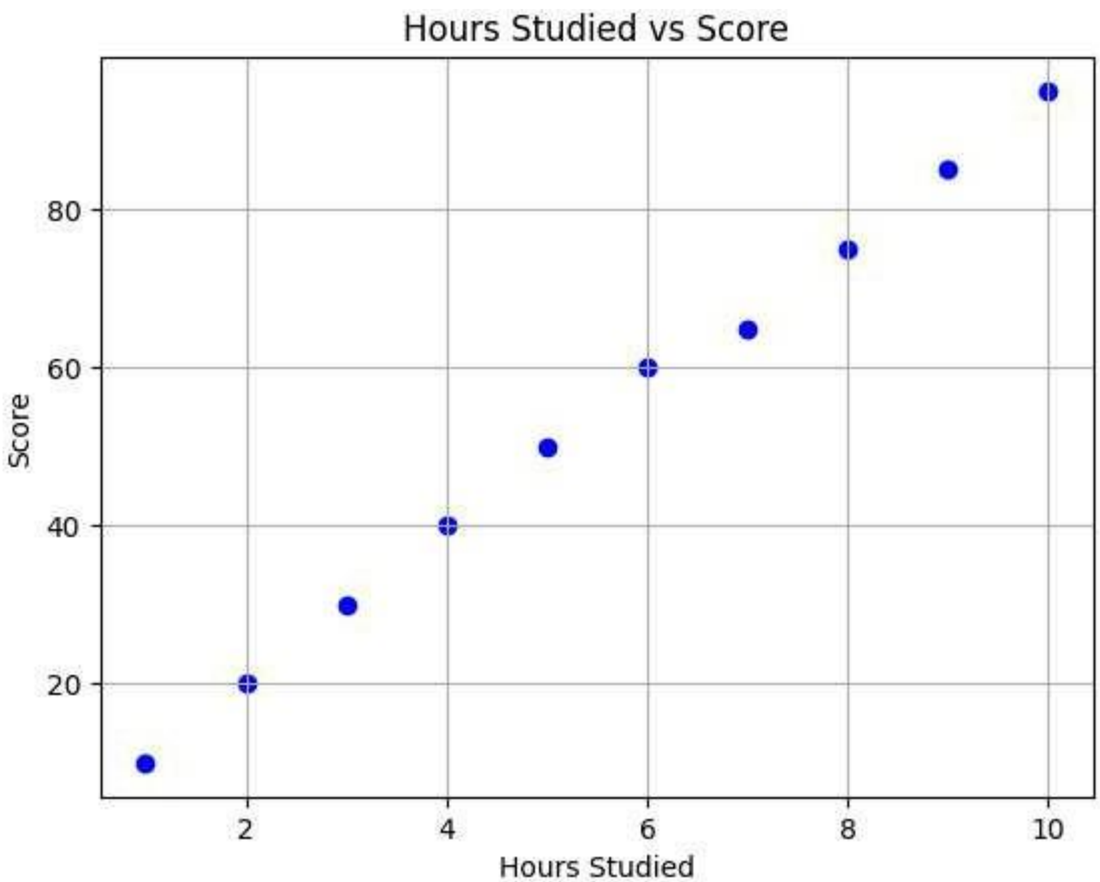
```
# Creating a dataset
data= {
    'Hours Studied': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Scores': [10, 20, 30, 40, 50, 60, 65, 75, 85, 95]
}
df = pd.DataFrame(data)

# Display the data
print(df)
```

	Hours_Studied	Scores
0	1	10
1	2	20

2	3	30
3	4	40
4	5	50
5	6	60
6	7	65
7	8	75
8	9	85
9	10	95

Implementation/Output snap shot:

```
plt.scatter(df['Hours_Studied'], df['Scores'], color='blue')
plt.title('Hours Studied vs Score')
plt.xlabel('Hours Studied')
plt.ylabel('Score')
plt.grid(True)
plt.show()
```



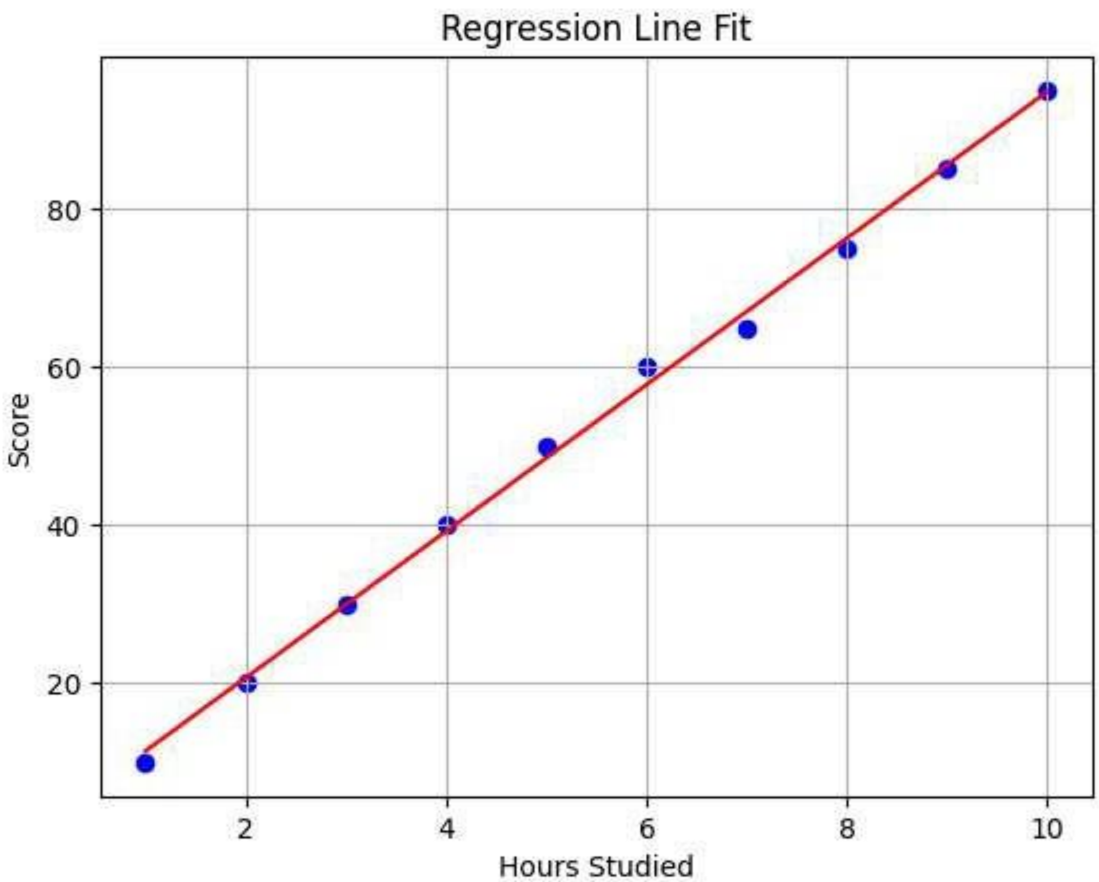
```
# Features and labels
X = df[['Hours_Studied']] # Independent variable
y = df['Scores']          # Dependent variable

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# Create a model
model= LinearRegression()

# Fit the model
model.fit(X_train, y_train)
```

```
.. LinearRegression i ?

LinearRegression()
```

```
# Predict test set results
y_pred = model.predict(X_test)

# Compare actual vs predicted
df_compare = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df_compare)
```

```
   Actual  Predicted
8      85  85.560345
1      20  20.689655
```

```
# Plot regression line
plt.scatter(X, y, color='blue')
plt.plot(X, model.predict(X), color='red') # Regression line
plt.title('Regression Line Fit')
plt.xlabel('Hours Studied')
plt.ylabel('Score')
plt.grid(True)
plt.show()
```




```
# Evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse:.2f}')
print(f'R² Score: {r2:.2f}')
```

```
Mean Squared Error: 0.39
R² Score: 1.00
```

```
# Predict for a new value
hours= 7.5
predicted_score = model.predict([[hours]])
print(f"Predicted Score for studying {hours} hours: {predicted_score[0]:.2f}")
```

```
Predicted Score for studying 7.5 hours: 71.66
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: >
  warnings.warn(
```

Conclusion:

Linear Regression is a simple yet powerful algorithm used for predicting continuous outcomes based on the relationship between variables. Through this implementation, we successfully trained a model to understand and capture the pattern between hours studied and student scores. The model's performance was evaluated using error metrics like Mean Squared Error and R^2 Score, confirming its accuracy and reliability for prediction. This practical application demonstrates how Linear Regression can be effectively used in real-world scenarios such as forecasting, analytics, and decision-making.

Review Questions:

1. What are the key steps involved in implementing a simple linear regression model using Python and scikit-learn?

Here are the key steps:

Import Libraries

Import required libraries like pandas, numpy, matplotlib.pyplot, and sklearn.linear_model.

Load or Create Dataset

Prepare the dataset with one independent variable (X) and one dependent variable (V).

Split Dataset

Use train_test_split() from sklearn.model_selection to divide the dataset into training and testing sets.

1. **Initialize the Model**

Create a LinearRegression() object.

2. **Train the Model**

Fit the model on training data using .fit(X_train, y_train).

3. **Make Predictions**

Predict the test data using .predict(X_test).

4. **Visualize the Results**

Plot the regression line and scatter plot to see the fit.

5. **Evaluate the Model**

Use evaluation metrics like Mean Squared Error (MSE) and R^2 Score to measure model performance.

2. How can you evaluate the performance of a linear regression model in Python? List and explain at least two metrics.

Two common evaluation metrics are:

a) Mean Squared Error (MSE}

Definition: Average of the squares of the errors (difference between actual and predicted values).

Formula:

MSE=1 n2:i=1 n(yi-yAi)2

Interpretation: Lower MSE means better performance. It penalizes larger errors more than smaller ones.

b} R^2 Score (Coefficient of Determination)

Definition: Indicates how well the independent variable(s) explain the variation in the dependent variable.

Range: 0 to 1 (closer to 1 is better)

Interpretation:

R^2 = 1 - perfect flt

R^2 = 0 - model explains none of the variability

from sklearn.metrics import mean_squared_error, r2_score mse = mean_squared_error(y_test, y_pred) r2 = r2_score(y_test, y_pred)

3. What is the role of the train_test_split() function in building a linear regression model, and why is it important?

Role of train_test_split(): It splits the dataset into training and testing subsets.

Syntax:

from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Why It's Important:

1. Avoids Overfltting: Ensures the model doesn't just memorize the training data.
2. Evaluates Generalization: Helps test the model on unseen data, giving a better estimate of real-world performance.
3. Improves Model Validation: Separates model development from model evaluation.

GitHub Link :-

https://github.com/Krish26-st/DWM-Assignments