**Branch: - Computer Science and Engineering Class: - III Year**

**Subject: - Big Data Analytics Lab Sem: - VI**

**Teacher Manual**

**PRACTICAL NO. 4**

**Aim:** Write a program to implement Linear Regression.

**Software Requirement**: Jupyter

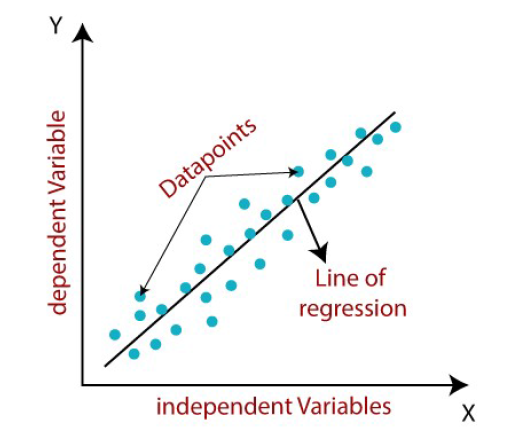
**Theory:**

**Linear Regression:**

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image.



Mathematically, we can represent a linear regression as:

**y= a0+a1x+ ε**

Where

y= Dependent Variable (Target Variable)

x= Independent Variable (predictor Variable)

a0= intercept of the line (Gives an additional degree of freedom)

a1 = Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

**Types of Linear Regression**

Linear regression can be further divided into two types of the algorithm:

o ***Simple Linear Regression:***

If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

o ***Multiple Linear regression:***

If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

**Finding the best fit line:**

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a0, a1) gives a different line of regression, so we need to calculate the best values for a0 and a1 to find the best fit line, so to calculate this we use cost function.

**Cost function**-

o The different values for weights or coefficient of lines (a0, a1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.

o Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.

o We can use the cost function to find the accuracy of the mapping function, which maps the input variable to the output variable. This mapping function is also known as Hypothesis function.

For Linear Regression, we use the Mean Squared Error (MSE) cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

**For the above linear equation, MSE can be calculated as:**



Where,

N=Total number of observation

yi = Actual value

(a1 xi+a0)= Predicted value.

**Residuals**: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

**Program:**

* Here, we will call some basic and important libraries to work.

|  |
| --- |
| import pandas as pd import numpy as np  import matplotlib.pyplot as plt |

* load the file into DataFrame object

|  |
| --- |
| df=pd.read\_csv('Filename.csv') |

* Head method shows us only the first 5 Rows

|  |
| --- |
| df.head() |

* To get column names

|  |
| --- |
| df.columns |

* To draw a scatter plot we used scatter() function, scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x- axis, and one for values on the y-axis

|  |
| --- |
| plt.scatter(df[' column\_name1'],df[' column\_name2']) |

|  |
| --- |
| X = df[['column\_name1']]  y = df[['column\_name2']] |

* sklearn is one of the most important packages in machine learning and big data analytics & it provides the maximum number of functions and algorithms. To use **Linear Regression class we need to call it from sklearn package**.

|  |
| --- |
| from sklearn.linear\_model import LinearRegression |

* Create a model using following command and train the model by using '.fit()' function

|  |
| --- |
| Model = LinearRegression() Model.fit(X,y) |

* The linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data. To see the value of the intercept and slop calculated by the linear regression algorithm for our dataset, execute the following code.

|  |
| --- |
| *#To retrieve the intercept:*  Model.intercept\_ |

|  |
| --- |
| *#For retrieving the slope*  Model.coef\_ |

* To create a x axis line we use .arange() & it is 1D x1 not been use so we reshape this

|  |
| --- |
| x1 = np.arange(1,11) |

* Using reshape( ) function

|  |
| --- |
| x1.reshape(2,5) |

|  |
| --- |
| y1 = Model.predict(x1)  y1 |

|  |
| --- |
| plt.scatter(X,y)  plt.plot(x1,y1) |

* **Split arrays or matrices into random train and test subsets**, so we need to import **train**\_**test\_split** **from sklearn package**.

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split |

* Next, we split 80% of the data to the training set while 20% of the data to test set using below code.
* The test\_size variable is where we actually specify the proportion of the test set.

|  |
| --- |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2) |

* After splitting the data into training and testing sets, finally, the time is to train our algorithm. For that, we need to import LinearRegression class, instantiate it, and call the fit() method along with our training data.

|  |
| --- |
| Model1 = LinearRegression()  Model1.fit(X\_train,y\_train) |

* Now that we have trained our algorithm, it’s time to make some predictions. To do so, we will use our test data and see how accurately our algorithm predicts the percentage score. To make predictions on the test data, execute the following script

|  |
| --- |
| y\_pred = Model1.predict(X\_test) |

|  |
| --- |
| Model1.score(X\_test,y\_test) |

|  |
| --- |
| Model1.score(X\_train,y\_train) |

|  |
| --- |
| plt.scatter(X\_train,y\_train) |

|  |
| --- |
| plt.scatter(X\_test,y\_test)  plt.plot(X\_test, y\_pred, color='red', linewidth=2) |

|  |
| --- |
| plt.plot(X\_test,y\_test)  plt.plot(X\_test, y\_pred, color='red', linewidth=2) |

**Result:**