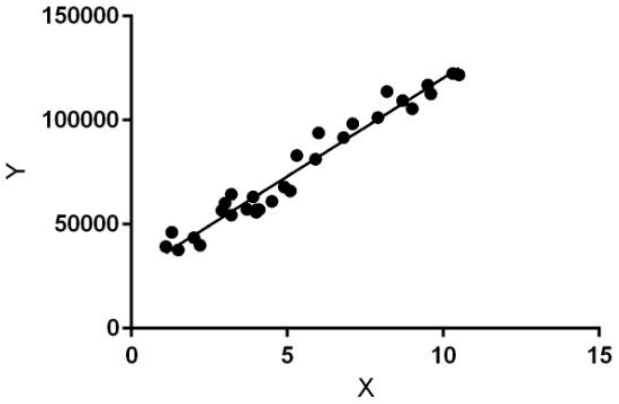
Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

JUNAID GIRKAR
60004190057
TE COMPS A4

EXPERIMENT - 1

AIM: Implement Multiple Linear Regression algorithm with gradient descent.

THEORY:
Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

MULTIPLE LINEAR REGRESSION:
Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression is to model the linear relationship between the explanatory (independent) variables and response (dependent) variables. In essence, multiple regression is the extension of ordinary least-squares (OLS) regression because it involves more than one explanatory variable.

Multiple linear regression (MLR) is used to determine a mathematical relationship among several random variables. In other terms, MLR examines how multiple independent variables are related to one dependent variable. Once each of the independent factors has been

determined to predict the dependent variable, the information on the multiple variables can be used to create an accurate prediction on the level of effect they have on the outcome variable. The model creates a relationship in the form of a straight line (linear) that best approximates all the individual data points.

Formula and Calculation of Multiple Linear Regression

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon$$

where, for $i = n$ observations:

y_i = dependent variable

x_i = explanatory variables

β_0 = y-intercept (constant term)

β_p = slope coefficients for each explanatory variable

ϵ = the model's error term (also known as the residuals)

The multiple regression model is based on the following assumptions:

- There is a linear relationship between the dependent variables and the independent variables
- The independent variables are not too highly correlated with each other
- yi observations are selected independently and randomly from the population
- Residuals should be normally distributed with a mean of 0 and variance σ

CODE:

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
# Extracting and Visualizing Data
data = pd.read_csv('Advertising.csv')
sns.pairplot(data)
data
# Classifying and Normalizing data into X(input) and Y(target)
X = data.iloc[:,2:]
X=(X-X.min())/(X.max()-X.min())
y = data.iloc[:,1]
```

```
y =(y-y.min())/(y.max()-y.min())
y = y.to_numpy()
yList = list()
for i in range(X.shape[0]):
    temp = list()
    temp.append(y[i])
    yList.append(temp)
yList
y = yList
# Initializing Variables
m = X.shape[0]
epochs = 10000
theta = np.random.rand(X.shape[1] + 1,1)
alpha = 0.01
print('Initial theta values', theta)
# Adding Bias
X = np.concatenate((np.ones([m,1]),X),axis=1)
# Cost Function
def computeCost(X,y,theta):
    m = X.shape[0]
    h_x = np.matmul(X,theta)
    J = (1/(2*m))*(sum(np.square(h_x-y)))
    return J
J = computeCost(X,y,theta)
J[0]
# Gradient Descent
X_transpose = np.transpose(X)
print(theta.shape)
for i in range(epochs):
    h_x = np.matmul(X,theta)
    theta = theta - ((alpha/m) * np.matmul(X_transpose,h_x-y))
print('Theta from Gradient Descent', theta)
# Checking Accuracy of Model by Mean Squared Error
from sklearn import metrics
y_pred = np.matmul(X,theta)
print("MEAN ABSOLUTE ERROR : ",metrics.mean_absolute_error(y,y_pred))
print("MEAN SQUARED ERROR : ", metrics.mean_squared_error(y,y_pred))
print("MEAN ERROR : ",np.sqrt(metrics.mean_squared_error(y,y_pred)))
```

OUTPUT:

```
Initial theta values [[0.39436149]
 [0.92955861]
```

```
 [0.90743643]
 [0.51255284]]
(4, 1)
Theta from Gradient Descent [[ 0.02655971]
 [-0.5052639 ]
 [ 0.02302333]
 [ 1.43141224]]
MEAN ABSOLUTE ERROR :  0.09025888384032164
MEAN SQUARED ERROR :  0.013561846240113393
MEAN ERROR :  0.11645534010990391
```

CONCLUSION: We learnt about Multiple Linear Regression with gradient descent and implemented its code from scratch in python.