Marwadi University	**Marwadi University** **Faculty of Technology** **Department of Information and Communication Technology**
Subject: Artificial Intelligence(01CT0616)	**Aim: To obtain the best fit line over single feature scattered datapoints using Linear Regression**

Experiment No: 02	**Date:**	**Enrollment No: 92200133003**

Aim: To obtain the best fit line over single feature scattered datapoints using Linear Regression

IDE: Google Colab

Theory:

Linear regression is a method for determining the best linear relationship between two variables *X* and *Y*. If variables *X* and *Y* are uncorrelated, it is pointless embarking upon linear regression. However, if a reasonable degree of correlation exists between *X* and *Y* then linear regression may be a useful means to describe the relationship between the two variables. The usual approach is to use the *least-squares* method, which minimizes the squared difference between the actual data points and a straight line. Let $[x_i, y_i]$, $i = 1,2,3,....,N$ be the *N* pairs of data values of the variables *X* and *Y*. The straight-line relating *X* and *Y* is $y = mx + c$, where *m* and *c* are the gradient and constant values (to be determined) defining the straight line. Thus, $y(x_i) - y_i$ is the difference between the line and data point *i* (see Fig. 1). Taking all the data points, we seek values of *m* and *c* that minimize the squared difference *SD*.

$$\sum_{1}^{N}[y(x_i) - y_i]^2$$

This is achieved by calculating the partial derivatives of *SD* with respect to *m* and *c* and finding the pair $[m,c]$ such that *SD* is at a minimum.

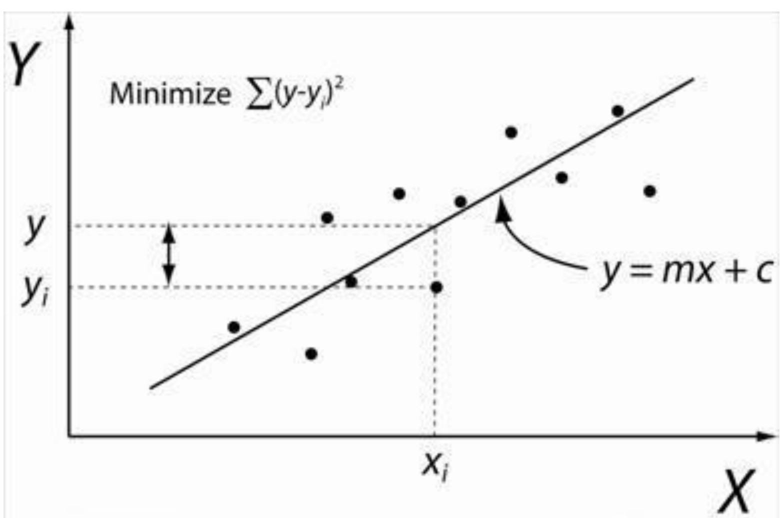


Figure 1: Illustration of Linear Regression. Linear least squares regression, the idea is to find the line y = mx + c that minimizes the mean squared difference between the line and the data points

Batch Gradient Descent:

Gradient Descent is an optimization algorithm used for minimizing the cost function in various machine learning algorithms. It is basically used for updating the parameters of the learning model. Batch gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive.

Let m be the number of training examples. Let n be the number of features.

Algorithm for batch gradient descent :

Let $h_\theta(x)$ be the hypothesis for linear regression. Then, the cost function is given by:
Let Σ represents the sum of all training examples from i=1 to m.
$J_{train}(\theta) = (1/2m)\ \Sigma(\ h_\theta(x^{(i)})\ - y^{(i)})^2$

Repeat {
$\theta j = \theta j -$ (learning rate/m) * $\Sigma(\ h_\theta(x^{(i)})\ - y^{(i)})x_j^{(i)}$
For every j =0 …n
}
Where $x_j^{(i)}$ Represents the j^{th} feature of the i^{th} training example. So if *m* is very large(e.g. 5 million training samples), then it takes hours or even days to converge to the global minimum.That's why for large datasets, it is not recommended to use batch gradient descent as it slows down the learning.

Pre Lab Exercise:

a. Explain the meaning of linear regression

____

____

____

____

b. Write three applications of linear regression

____

____

____

____

c. Write three advantages of linear regression

____

____

____

____

d. Write three limitations of linear regression

____

____

____

____

Methodology:

1. Load the basic libraries and packages
2. Load the dataset
3. Analyse the dataset
4. Pre-process the data
5. Visualize the Data
6. Separate the feature and prediction value columns
7. Write the Hypothesis Function
8. Write the Cost Function
9. Write the Gradient Descent optimization algorithm
10. Apply the training over the dataset to minimize the loss
11. Find the best fit line to the given dataset
12. Observe the cost function vs iterations learning curve

Program (Code):

1. Load the basic libraries and packages

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

2. Load the dataset

```python
dataset=pd.read_csv("/content/train.csv")
dataset
```

	x	y
0	24.0	21.549452
1	50.0	47.464463
2	15.0	17.218656
3	38.0	36.586398
4	87.0	87.288984
...	...	...
695	58.0	58.595006
696	93.0	94.625094
697	82.0	88.603770
698	66.0	63.648685
699	97.0	94.975266

700 rows × 2 columns

3. Analyse the dataset

```
[ ] dataset.shape
```

```
(700, 2)
```

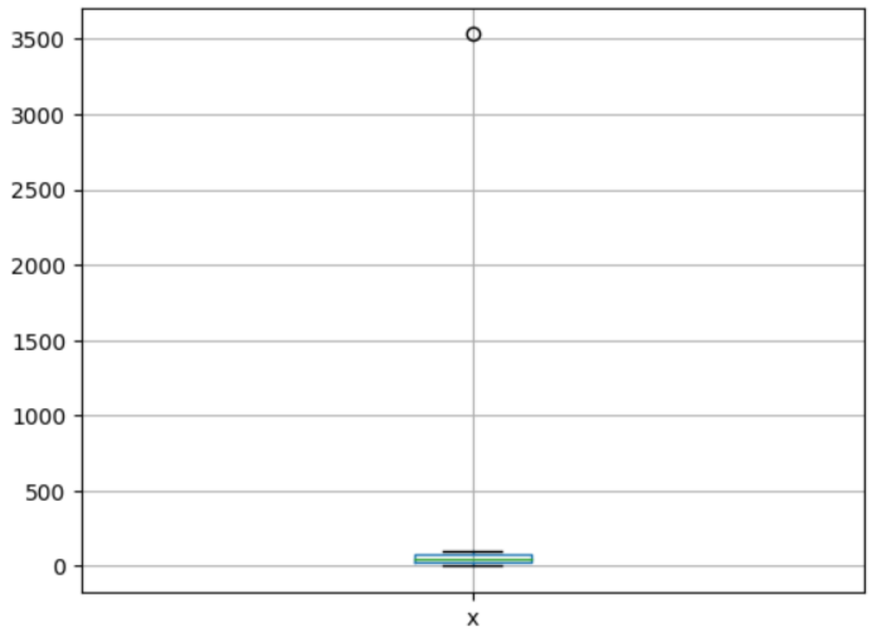
```
[ ] dataset.describe()
```

	x	y
count	700.000000	699.000000
mean	54.985939	49.939869
std	134.681703	29.109217
min	0.000000	-3.839981
25%	25.000000	24.929968
50%	49.000000	48.973020
75%	75.000000	74.929911
max	3530.157369	108.871618

4. Pre-process the data

```
x_value=dataset.iloc[0:700,0:1]
y_value=dataset.iloc[0:700,1:2]
x_value.boxplot(column=['x'])
```

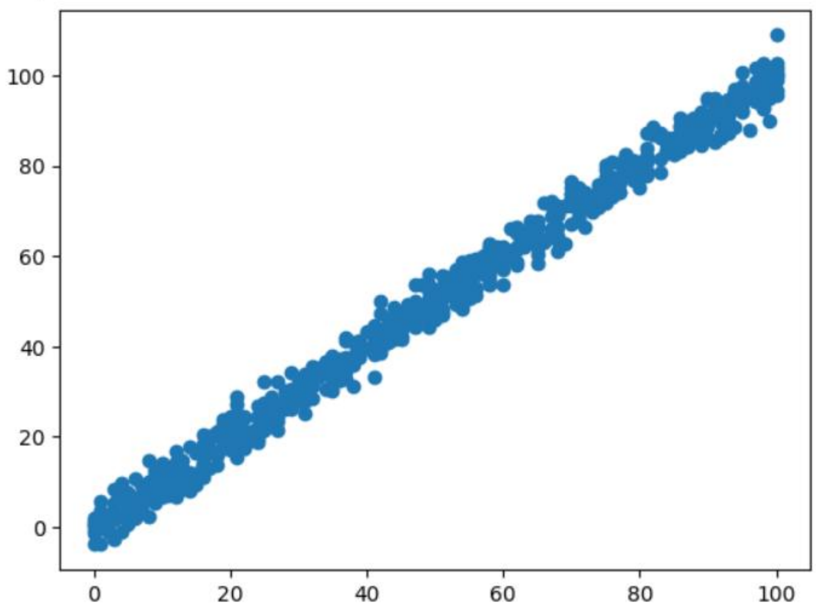
<Axes: >



5. Visualize the Data

```
plt.scatter(x_value,y_value)
```

<matplotlib.collections.PathCollection at 0x796eb8128c10>



6. Write the Hypothesis Function

```
def hypothesis(theta_array,x):
  h=theta_array[0]+theta_array[1]*x
  return h
```

10. Write the Cost Function

```
def costfunction(theta_array,x,y,m):
  total_cost=0
  for i in range(m):
    total_cost+=((theta_array[0]+theta_array[1]*x[i])-y[i])**2
  return total_cost/(2*m)
```

16. Write the Gradient Descent optimization algorithm

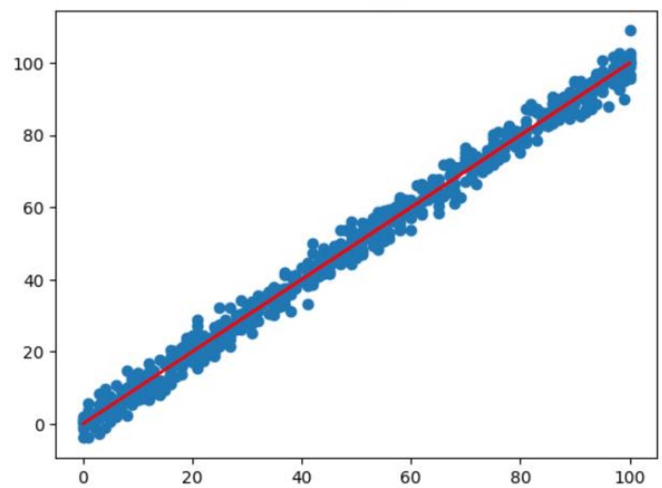
```
def gradient_descent(theta_array,x,y,m,alpha):
  summation_0=0
  summation_1=0
  for i in range(m):
    summation_0+=((theta_array[0]+theta_array[1]*x[i])-y[i])
    summation_1+=((theta_array[0]+theta_array[1]*x[i])-y[i])*x[i]
  new_theta0=theta_array[0]-(summation_0*alpha)/m
  new_theta1=theta_array[1]-(summation_1*alpha)/m
  improvised_theta=[new_theta0,new_theta1]
  print(improvised_theta)
  return improvised_theta
```

17. . Apply the training over the dataset to minimize the loss

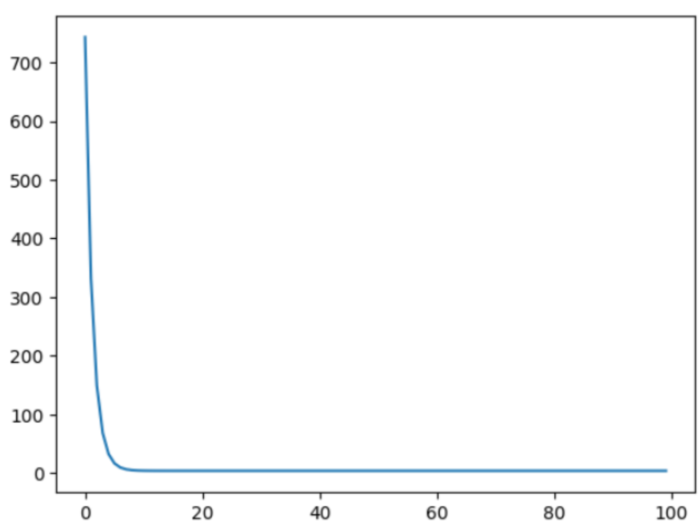
```
def training(x,y,alpha,epochs):
  theta_0=0
  theta_1=0
  m=x.size
  cost_values=[]
  theta_array=[theta_0,theta_1]
  for i in range(epochs):
    theta_array=gradient_descent(theta_array,x,y,m,alpha)
    loss=costfunction(theta_array,x,y,m)
    cost_values.append(loss)
    y_new=theta_array[0]+theta_array[1]*x

  plt.scatter(x,y)
  plt.plot(x,y_new,'r')
  plt.show()

  x=np.arange(0,epochs)
  plt.plot(x,cost_values)
  plt.show()
```

<table>
<tr><td></td><td colspan="2">Marwadi University
Faculty of Technology
Department of Information and Communication Technology</td></tr>
<tr><td>Subject: Artificial Intelligence(01CT0616)</td><td colspan="2">Aim: To obtain the best fit line over single feature scattered datapoints using Linear Regression</td></tr>
<tr><td>Experiment No: 02</td><td>Date:</td><td>Enrollment No: 92200133003</td></tr>
</table>

18. **Find the best fit line to the given dataset**



19. **Observe the cost function vs iterations learning curve**



Observation and Result Analysis:

a. Nature of the dataset

____

____

____

____

b. During Training Process

____

____

____

____

c. After the training Process

____

____

____

____

d. Observation over the Learning Curve

____

____

____

Post Lab Exercise:

a. What are the major assumptions considered in linear regression

____

____

____

b. Why MSE is used instead of MAE for calculating the loss function

____

____

____

c. How can the behaviour of outliers be understood while dealing with the unseen dataset

____

____

____

d. Derive the Normal Equation for the Linear Regression.