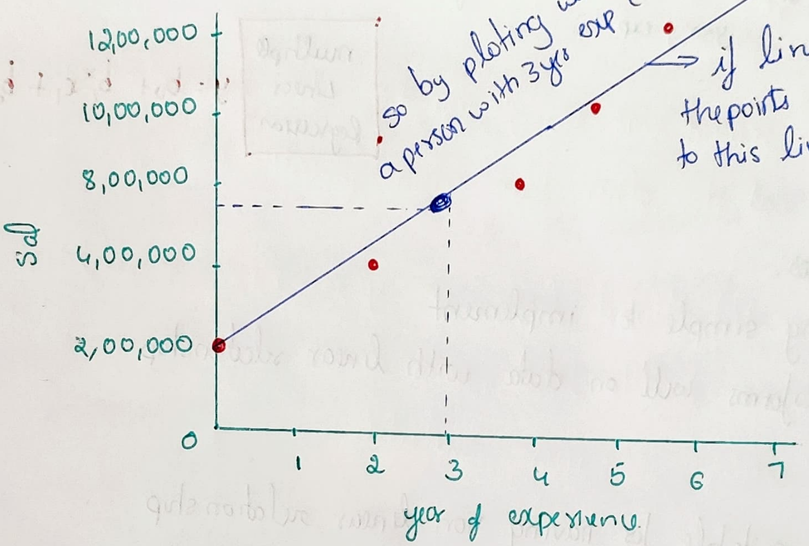
Linear Regression:-

data:

experience in Year	0	2	4	5	6
sal	200,000	4,00,000	8,00,000	10,00,000	12,00,000

what would be the salary of person in 3 yr of experience



so by ploting we can say that a person with 3yrs exp can earn about 6.5L

→ if line is draw the points are nearer to this line

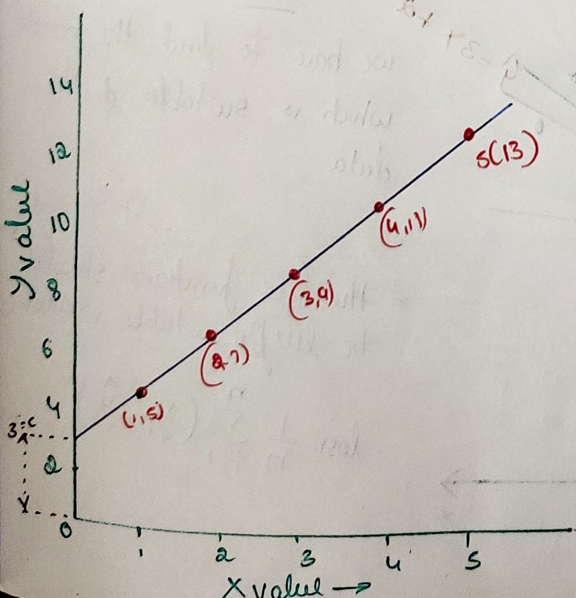
equation of the line:

$y = mx + c$

$x \to$ x value
$y -$ y value
$m \to$ slope
$c \to$ Intercept

data:-

x	1	2	3	4	5
y	5	7	9	11	13



Find the value of m and c

Point $P_1(\overset{x_1}{2}, \overset{y_1}{7})$
Point $P_2(\overset{x_2}{3}, \overset{y_2}{9})$

$$\boxed{\text{slope, } m = \frac{y_2 - y_1}{x_2 - x_1}} = \frac{9-7}{3-2} = \underline{\underline{2}}$$

$m = 2$

Intercept c.

Point(4,11) $\longrightarrow$ $y = 2x + c$

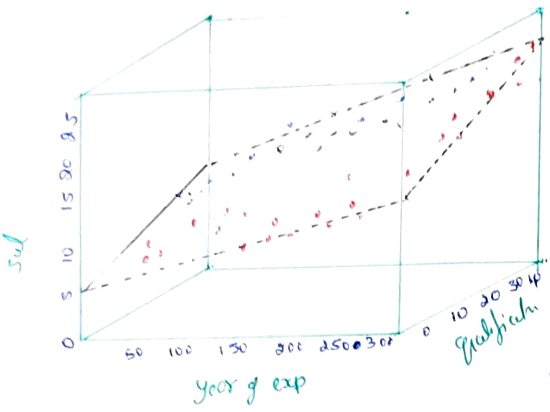
$11 = 2(4) + c$

$\underline{\underline{c = 3}}$

Multiple Linear Regression:

Multiple linear regression is a model for predicting the value of one dependent variable based on two or more independent variable

if we have more than 3 variables we use Multiple linear Regression.



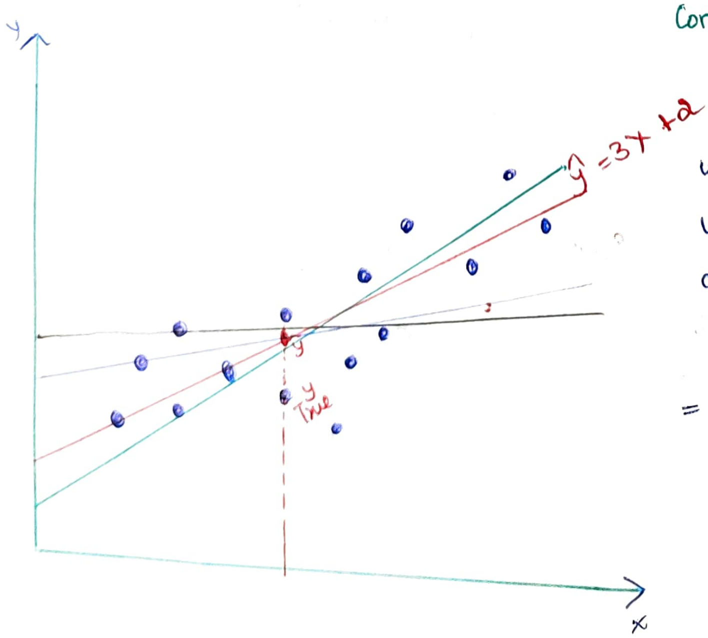
Simple Linear Regression	$y = b_0 + b \times x_1$
multiple Linear Regression	$y = b_0 + b_1^* x_1 + b_2^* x_2 + b_n^* x_n$

Advantages:

1) Very simple to implement
2) Performs well on data with linear relationship

Disadvantages

1) Not suitable for having non linear relationship
2) Underfitting issue
3) Sensitive to outliers



Consider we have 4 models/lines

we have to find the model which is suitable of the data

= the loss function should be less for suitable model

$$loss = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Randomly assigned Parameter $m=3$ $c=2$

i.e, $y = mx + c$

$\hat{y} = 3x + 2$

For example consider just 5 data to find the loss function

less the loss function more the accuracy

x	y	$\hat{y}$
2	10	8
3	14	11
4	18	14
5	22	17
6	26	20

data according to random parameters

$$loss = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$loss = [(10-8)^2 + (14-11)^2 + (18-14)^2 + (22-17)^2 + (26-20)^2] / 5$$

$$loss = (4 + 9 + 16 + 25 + 36)/5$$

$$loss = 18$$

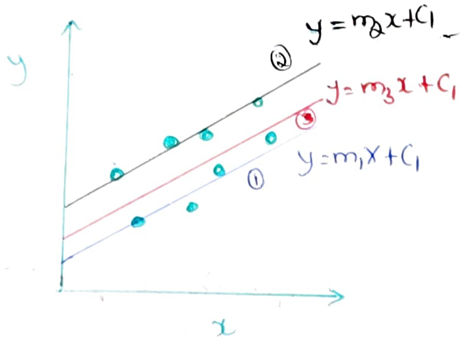
so here we got the loss function as 18 which is totally high and is not suitable so our model will change the previous ($m=3$ $c=2$) assigned parameter to a new random value ($m=?$ $c=?$) and check the loss function for it.

So by go doing it repeatedly the model will find the best suitable parameters (this is how linear regression model fits to data)

Gradient Descent for Linear Regression:

model optimization:

Optimization refers to determining best parameters for a model, such that the loss function of the model decreases, as a result of which can predict more accurately



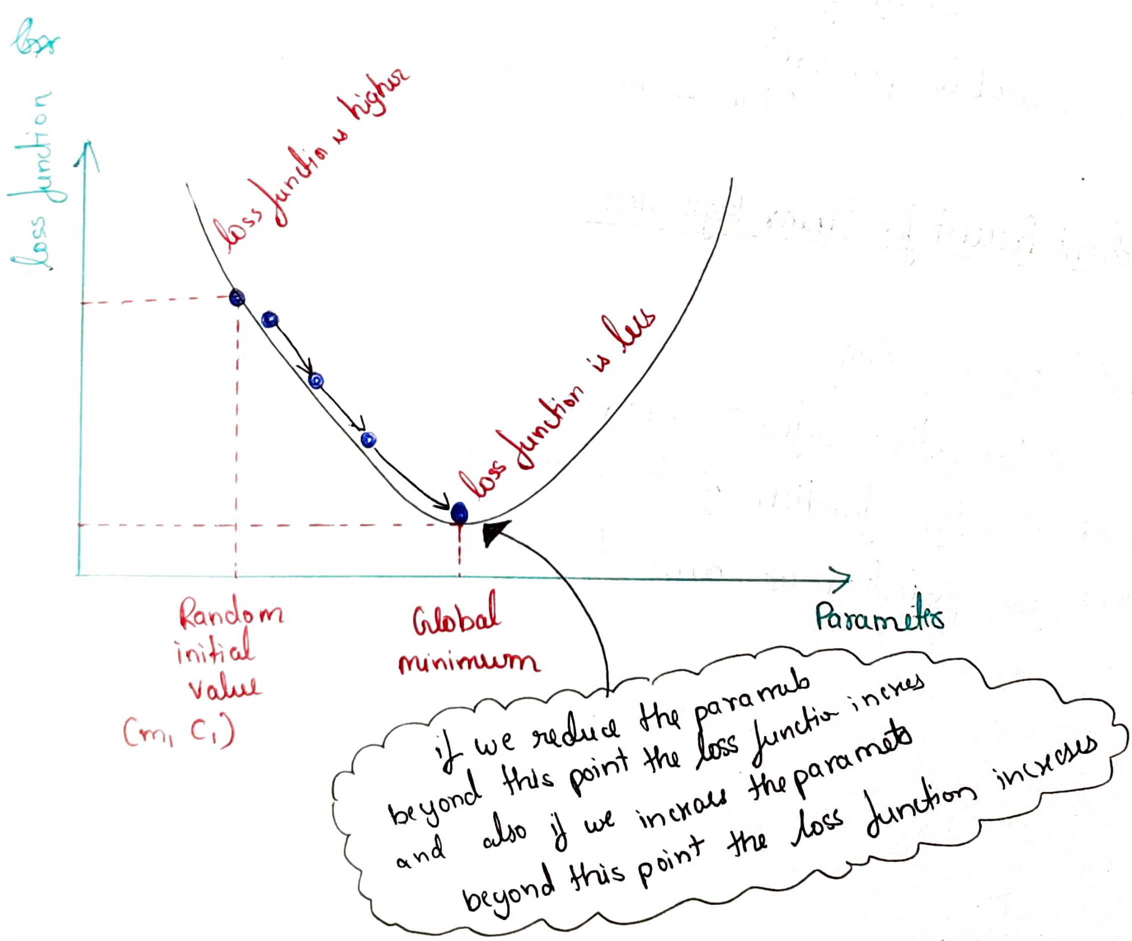
So by looking at the data we can say that as x value increases y value also increases

Hence we can fit this data to a stright line

for the first line i.e, $y = m_1x + c_1$ the four data point are nearer to it but the other four data points are far from it

for the second line i.e, $y = m_2x + c_2$ the four data points are nearer to it but the other four data points are far from it

And for the third line i.e, $y = m_3x + c_3$ ~~p~~ all the 8 points are nearer to the line, so now I can say that the third line is more suitable



Gradient Descent is an optimization algorithm used for minimizing the loss function in various machine learning algorithm. It is used for updating the parameters of learning model

$$m = m - L D_m$$
$$c = c - L D_c$$

$m \rightarrow$ slope
$c \rightarrow$ intercept
$L \rightarrow$ Learning Rate:- It is the parameter to determine the step size at each iteration
$D_m \rightarrow$ Partial Derivative of loss function with respect to m
$D_c \rightarrow$ Partial Derivative of loss function with respect to c

$$D_m = \frac{\partial(\text{Cost Function})}{\partial m} = \frac{\partial}{\partial m}\left(\frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2\right)$$

$$= \frac{1}{n}\frac{\partial}{\partial m}\left(\sum_{i=0}^{n}(y_i - (mx_i + c))^2\right)$$

$$= \frac{1}{n}\frac{\partial}{\partial m}\left(\sum_{i=0}^{n}(y_i^2 + m^2x_i^2 + c^2 + 2mx_ic - 2y_ix_im - 2y_ic)\right)$$

$$= \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - (mx_i + c))$$

$$= \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \hat{y}_i)$$

$$D_c = \frac{\partial(\text{Cost Function})}{\partial c} = \frac{\partial}{\partial c}\left(\frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2\right)$$

$$= \frac{1}{n}\frac{\partial}{\partial c}\left(\sum_{i=0}^{n}(y_i - (mx_i + c))^2\right)$$

$$= \frac{1}{n}\frac{\partial}{\partial c}\left(\sum_{i=0}^{n}(y_i^2 + m^2x_i^2 + c^2 + 2mx_ic - 2y_imx_i - 2y_ic)\right)$$

$$= \frac{-2}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))$$

$$= \frac{-2}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)$$

```
# importing numpy library

import numpy as np
```

Linear Regression

Building Linear Regression

```
class Linear_Regression():

  def __init__(self, learning_rate, no_of_iterations):

    self.learning_rate = learning_rate

    self.no_of_iterations = no_of_iterations

  def fit(self, x, y):

    # no_of_training examples and number of features


    self.m , self.n = x.shape # number of m(rows) and n(columns)


    # initiating the weight and bias


    self.w = np.zeros(self.n)   # no of features
    self.b = 0
    self.x = x
    self.y = y

    # Implementing Gradient Descent


    for i in range(self.no_of_iterations):
      self.update_weights()

  def update_weights(self,):
    y_prediction = self.predict(self.x)


    # calculating gradients

    dw = -(2*(self.x.T).dot(self.y - y_prediction))  / self.m
    db = -2 * np.sum(self.y - y_prediction) / self.m

      # Updating the weights

    self.w = self.w - self.learning_rate*dw
    self.b = self.b - self.learning_rate*db


  def predict(self, x):
    return x.dot(self.w) + (self.b)
```

Checking the model

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
salary = pd.read_csv('/content/salary_data.csv')
```

```
salary.head()
```

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

```
salary.shape
```

```
(30, 2)
```

```
salary.isna().sum()
```

```
YearsExperience    0
Salary             0
dtype: int64
```

```
x = salary.drop(columns = 'Salary', index = None)
y = salary['Salary']
```

```
print(x)
```

```
    YearsExperience
0               1.1
1               1.3
2               1.5
3               2.0
4               2.2
5               2.9
6               3.0
7               3.2
8               3.2
9               3.7
10              3.9
11              4.0
12              4.0
13              4.1
14              4.5
15              4.9
16              5.1
17              5.3
```

```
18     5.9
19     6.0
20     6.8
21     7.1
22     7.9
23     8.2
24     8.7
25     9.0
26     9.5
27     9.6
28    10.3
29    10.5
```

```
print(y)
```

```
0      39343
1      46205
2      37731
3      43525
4      39891
5      56642
6      60150
7      54445
8      64445
9      57189
10     63218
11     55794
12     56957
13     57081
14     61111
15     67938
16     66029
17     83088
18     81363
19     93940
20     91738
21     98273
22    101302
23    113812
24    109431
25    105582
26    116969
27    112635
28    122391
29    121872
Name: Salary, dtype: int64
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25, random_state=2)
```

```
model = Linear_Regression(0.02,500)
```

```
model.fit(x_train,y_train)
```

```
# printing the parameter values ( Weights & bias )
print("weight = ", model.w[0])
print("bias = ", model.b)
```

```
weight =  9578.952732154148
bias =  23438.08204654618
```

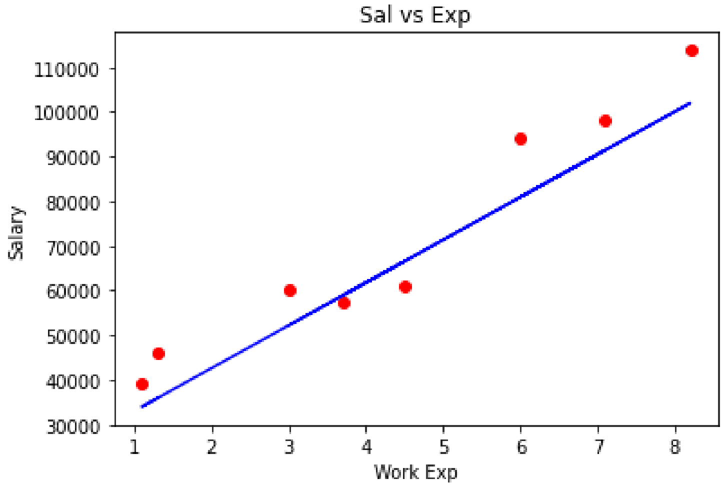
```
test_data_prediction = model.predict(x_test)
```

```
print(test_data_prediction)
```

```
1      35890.720598
0      33974.930052
14     66543.369341
9      58880.207156
21     91448.646445
19     80911.798439
23    101985.494450
6      52174.940243
dtype: float64
```

```
plt.scatter(x_test,y_test, color = 'red')
plt.plot(x_test, test_data_prediction, color = 'blue')
plt.xlabel('Work Exp')
plt.ylabel('Salary')
plt.title('Sal vs Exp')
```

```
Text(0.5, 1.0, 'Sal vs Exp')
```



Testing

```
input_data = (10.5)
input_data_np = np.asarray(input_data)
input_data_reshaped = input_data_np.reshape(1,-1)
prediction = model.predict(input_data_reshaped)
print(prediction)
```

```
[124017.08573416]
```

0s completed at 20:14