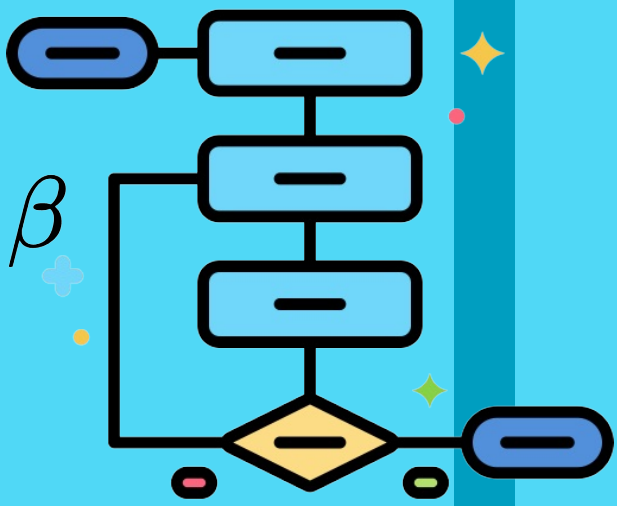
Simple Linear Regression Algorithm from scratch.

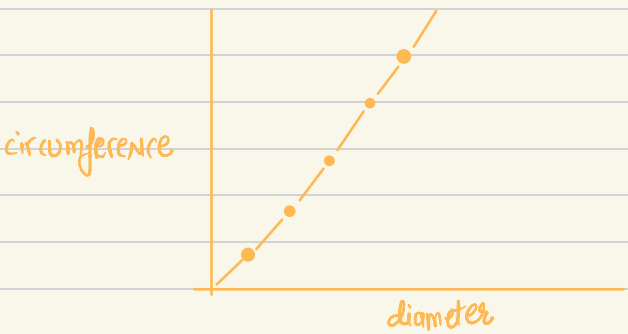
$$y = \alpha * x + \beta$$



What is Simple Linear Regression?

Simple linear Regression is a statistical method that helps us to find out the mathematical relationship between two continuous variables.

variable x → Independent variable
variable y → Dependent variable



⇒ circumference = π x diameter

⇓

already define relationship

⇓

But In reallife we have to find the relationship.

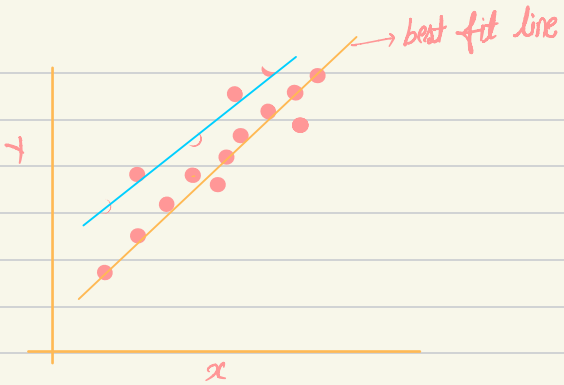
⇓

for example ⇒ weight - height
speed - mileage
Iq - placement

Here, We have to find a best fit line using slop-Intercept concept.

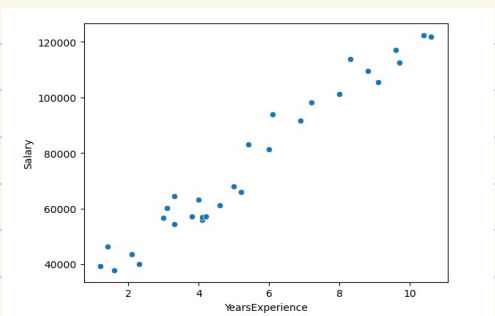
$y = mx + b$ ⇒ m = slop, b = Intercept

Best Fit line ⇒ A Line which have lowest error on it. which means that this line cover most of the points on the plan.



For better understanding, I am taking a simple example.

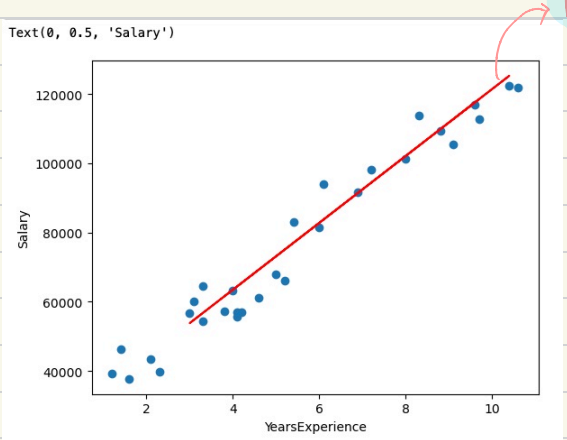
	Unnamed: 0	YearsExperience	Salary
24	24	8.8	109432.0
28	28	10.4	122392.0
25	25	9.1	105583.0
16	16	5.2	66030.0
3	3	2.1	43526.0
1	1	1.4	46206.0
8	8	3.3	64446.0
21	21	7.2	98274.0
7	7	3.3	54446.0
18	18	6.0	81364.0



Now for given data we will find Best fit Line

Train the model ⇒

```
1 from sklearn.model_selection import train_test_split
1 test, y_train, y_test = train_test_split(df[['YearsExperience']], df[['Salary']], test_size=0.25, random_state=10
1 from sklearn.linear_model import LinearRegression
1 lr = LinearRegression()  ⇒ our model
1 lr.fit(x_train,y_train)  ⇒ Train the model
▾ LinearRegression
LinearRegression()
```



Best fit Line

Salary = m(YearsExperience) + b

by give code we find out the value of Intercept(b) and slope(m)

```
1 lr.coef_
array([[9657.22661273]])
```

⇒ slop

```
1 lr.intercept_
array([24857.13238724])
```

⇒ Intercept

So equation of Best fit line ⇒

Salary = (9657.22)x + 24857.132

If YearsExperience = 0 then Salary = 24857.132,

∴ b is also know as offset.

m, b are called weights

Now, one question should be in your mind that how this function has find these values. and the true maths comes in the picture.

How to find m and b?

There are two main methods to find out the values or weights.

① Closed-form method ⇒ create a formula using any logical formula with NO free variables. Can not use limit, differentiation or Integration. Linear Regression class.
(ordinary least square OLS) ←

② Non Closed-form method ⇒ using Gradient descent, can use limit, differentiation or Integration. SGD Regression class.

For small features OLS appoarch is effective but if we have higher dimension of data then Gradient descent have better performance.

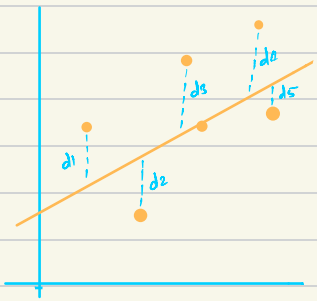
OLS method to find m and b ⇒

$$b = \bar{y} - m\bar{x} \qquad , \quad m = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

all the points we have they are sort of linear. and we want to draw a best fit line.

↓↓

try to cover maximum points.
and reduce the error.



$Error(E) = d_1 + d_2 + d_3 + \cdots + d_n$ ⇒ in this equation, value of d may be +ve, and -ve and that may not gives correct output, therefore we are taking square of each and every distance to make positive.

$E = d_1^2 + d_2^2 + d_3^2 + \cdots + d_n^2$

$E = \sum_{i=1}^{n} d_i^2$ ⇒ Error function (J)

Why not mode?

$E = |d_1| + |d_2| + \cdots + |d_n|$

Reason ① ⇒ we want to penalize the outliers.

Reason ② ⇒ differentiation of mode is not possible at origin.

Want the value of m and b which have minimum error (E).

$E = \sum_{i=1}^{n} d_i$ (this is always in terms of y)

$\therefore \quad d_i = y_i - \hat{y}_i$

here y_i = actual value

$\hat{y}_i$ = Predicted value

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

$$\hat{y}_i = mx_i + b$$

$$E(m,b) = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

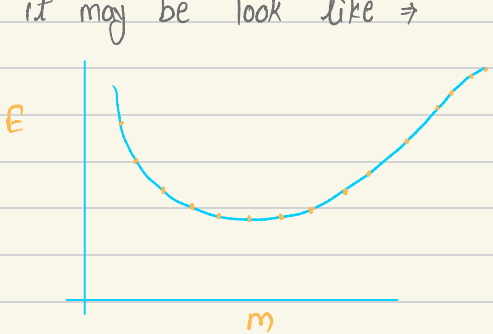
$y_i, x_i \Rightarrow$ constant $\Rightarrow$ cannot be change, we can change m, b to modify the error.

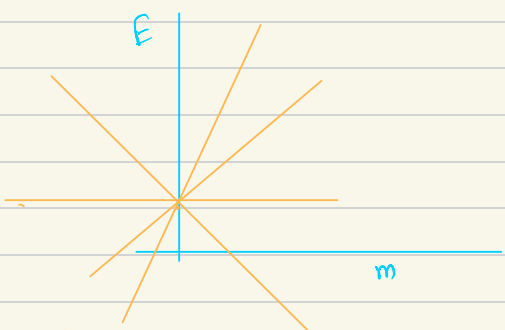
Our error function is depends on m and b. $(E(m,b))$

① Assume $b = 0$

$$E(m) = \sum_{i=1}^{n} (y_i - mx_i)^2$$

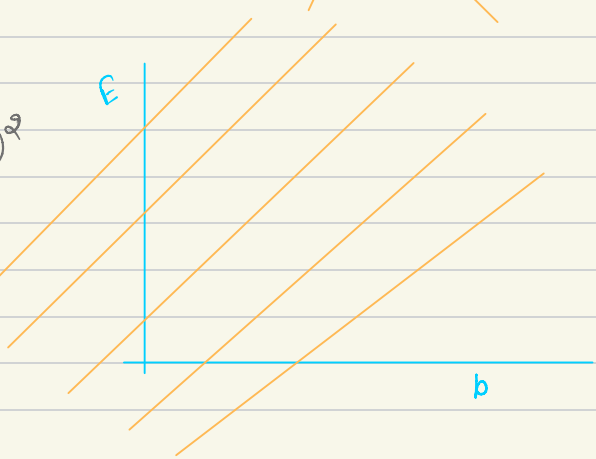
if we create a plot between m and E, then we will get a curve plot it may be look like $\Rightarrow$

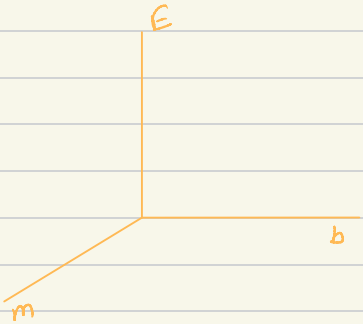


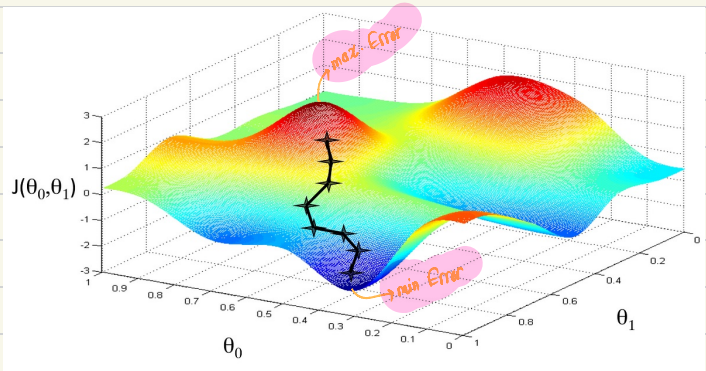


② Assume $m = 1$

$$E(b) = \sum_{i=1}^{n} (y_i - x_i - b)^2$$







we can find minimum point using maxima - minima.

↓

find out the derivative of error function with respect to m and b.

Now $\Rightarrow \frac{\partial E}{\partial b} \Rightarrow \frac{\partial}{\partial b}\left(\sum_{i=1}^{n}(y_i - mx_i - b)^2\right) = 0$

$$\sum \frac{\partial}{\partial b}(y_i - mx_i - b)^2 = 0$$

$$\sum 2(y_i - mx_i - b)(-1) = 0$$

$$\sum (y_i - mx_i - b) = 0$$

$$\frac{\sum y_i}{n} - \frac{\sum mx_i}{n} - \frac{\sum b}{n} = 0$$

$$\bar{y} - m\bar{x} - b = 0$$

$$b = \bar{y} - m\bar{x}$$

$$E = \sum (y_i - mx_i - \bar{y} + m\bar{x})^2$$

$$\frac{\partial E}{\partial m} = \sum \frac{\partial}{\partial m} (y_i - mx_i - \bar{y} + m\bar{x})^2 = 0$$

$$\Rightarrow \quad -\sum 2 (y_i - mx_i - \bar{y} + m\bar{x})^2 (x_i - \bar{x}) = 0$$

$$\Rightarrow \quad \sum \left[(y_i - mx_i - \bar{y} + m\bar{x}) \right]^2 (x_i - \bar{x}) = 0$$

$$\Rightarrow \quad \sum \left[(y_i - \bar{y}) - m(x_i - \bar{x}) \right] (x_i - \bar{x}) = 0$$

$$\Rightarrow \sum (y_i - \bar{y})(x_i - \bar{x}) - m(x_i - \bar{x})^2 = 0$$

$$\Rightarrow \sum (y_i - \bar{y})(x_i - \bar{x}) = m \sum (x_i - \bar{x})^2$$

$$m = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

```python
class LinearClass:
    '''
    This class implements the linear regression algorithm
    '''
    def __init__(self):
        '''
        Initializes the model parameters
        '''
        self.slope = None # Slope of the regression line(m)
        self.intercept = None # Intercept of the regression line(b)

    def fit(self, X_train,y_train):
        '''
        Fits the linear regression model to the training data

        Args:
            X_train (Numpy.ndarray) :  the traning features
            y_train (Numpy.ndarray) :  the traning labels
        '''
        # Have to calculate the numerator and denominator for the slope
        numerator = 0
        denominator = 0

        for i in range(X_train.shape[0]):
            numerator += ((X_train[i] - X_train.mean()) * (y_train[i] - y_train.mean()))
            denominator += ((X_train[i] - X_train.mean()) ** 2)

        # Calculate the slope and intercept
        self.slope = numerator/denominator
        self.intercept = y_train.mean() - (self.slope * X_train.mean())

        # Print the slope and intercept for information purposes
        print("Slop m", self.slope)
        print("Intercept b", self.intercept)

    def predict(self, X_test):
        '''
        Predicts the labels for new data points
        Args :
            X_test (Numpy.ndarray)  :  The test features
        Returns :
            Numpy.ndarray : the predicted labels
        '''
        return self.slope*X_test + self.intercept
```

```python
lr.fit(x_train['YearsExperience'].values, y_train['Salary'].values)
```

```
Slop m 9657.226612725688
Intercept b 24857.132387239224
```