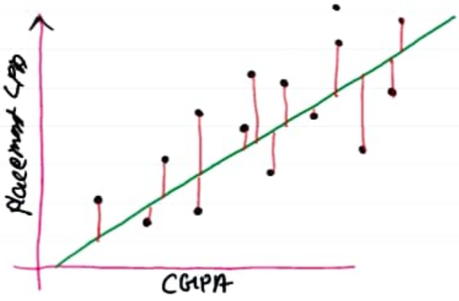
Linnear regressions mathematical intuitions

Wednesday, 3 May 2023 8:07 PM

Linear Regrations

Introduction :→ Linear regression is a basic machine learning model. which draw a best fit Line between based on trained data. Projection of Test data point on best fit line and make prediction according with that.

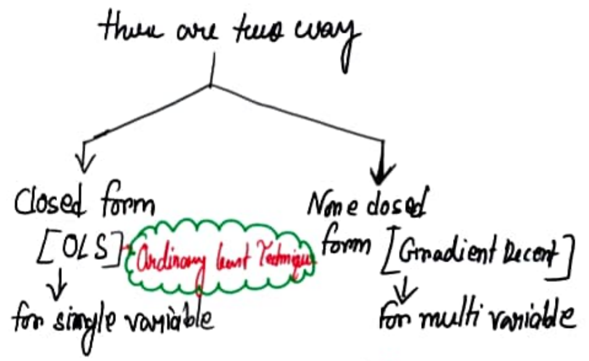


Type of Machine Learning Model :→

- (1) Simple Linear Regression
- (2) Multiple Linear Regression
- (3) polynomial Linear regression.

Simple Linear regression :- Predict Target variable one the Basis on one Linear variable

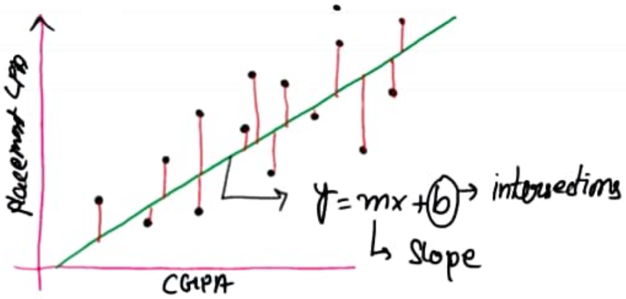
How to find Best fit Line :-



there are two way

Closed form [OLS] (Ordinary least Technique) → for single variable

None closed form [Gradient Decent] → for multi variable

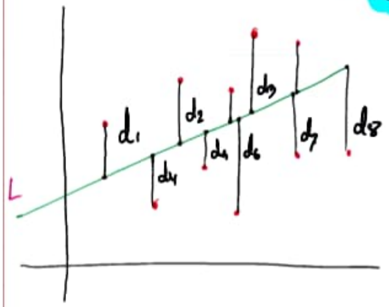


Solving Using Ordinary least Technique

Ordinary least technique used when one indipended variable present in our data.

X → input, $\bar{x}$ mean of x y → input, $\bar{y}$ mean of y, $\bar{y} = m\bar{x} + b$

→ $b = \bar{y} - m\bar{x}$ $M = \dfrac{\sum_{i=1}^{i=n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{i=n}(x_i - \bar{x})^2}$,



Let, assume that the line L, $y = mx + b$ is the best fit Line. now we need to find out the funtion of m(slope) and b(intercept)

for every point i on L, $y_i = mx_i + b$

$$\boxed{E = \sum_{i=1}^{n} d_i^2} \quad \text{(1)}$$

→ taking sum of squre of distances from L(straight Line) from all point.

↳ This is called error function or cost ferndions

$d_i = (y_i - \hat{y})$ → $\hat{y}$ is the predicted value of y_i on the straight Line

Putting $d_i = (y_i - \hat{y}_i)$ in equation (1)

Putting $d_i = (y_i - \hat{y}_i)$ in equation (1)

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

we know that, $\hat{y}_i = mx_i + b$

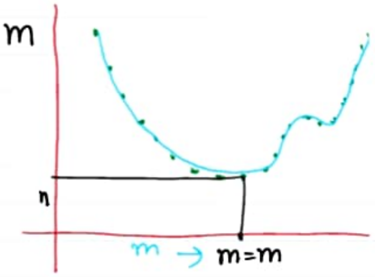
so, $$E(m,b) = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

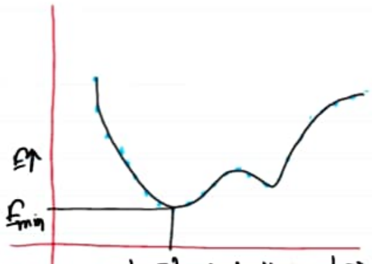
Error functin $E(m,b)$ is a function of m, b, so it depends on m, b

Now we need to find $E(m,b)_{min}$ → that is mean we need to find m, b value for which $E(m,b)$ will be minimum, to find follow below.

let suppose $b=0$, then $E(m,b) = E(m)$

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





b = b → Such that b→ E_{min}.

let, Suppose $m=1$ Constant, then, $E(m,b) = \sum_{i=1}^{n} (y_i - x_i - b)$

Mathematical methode of Calculating $E(m,b)_{min}$.

at, min Slope of $\frac{dE}{dy} = 0$, So,

So,

$$E(m,b) = \sum_{i=1}^{n} (y_i - mx_i - b)^2 = 0$$

taking derivative wrt b of this equations.

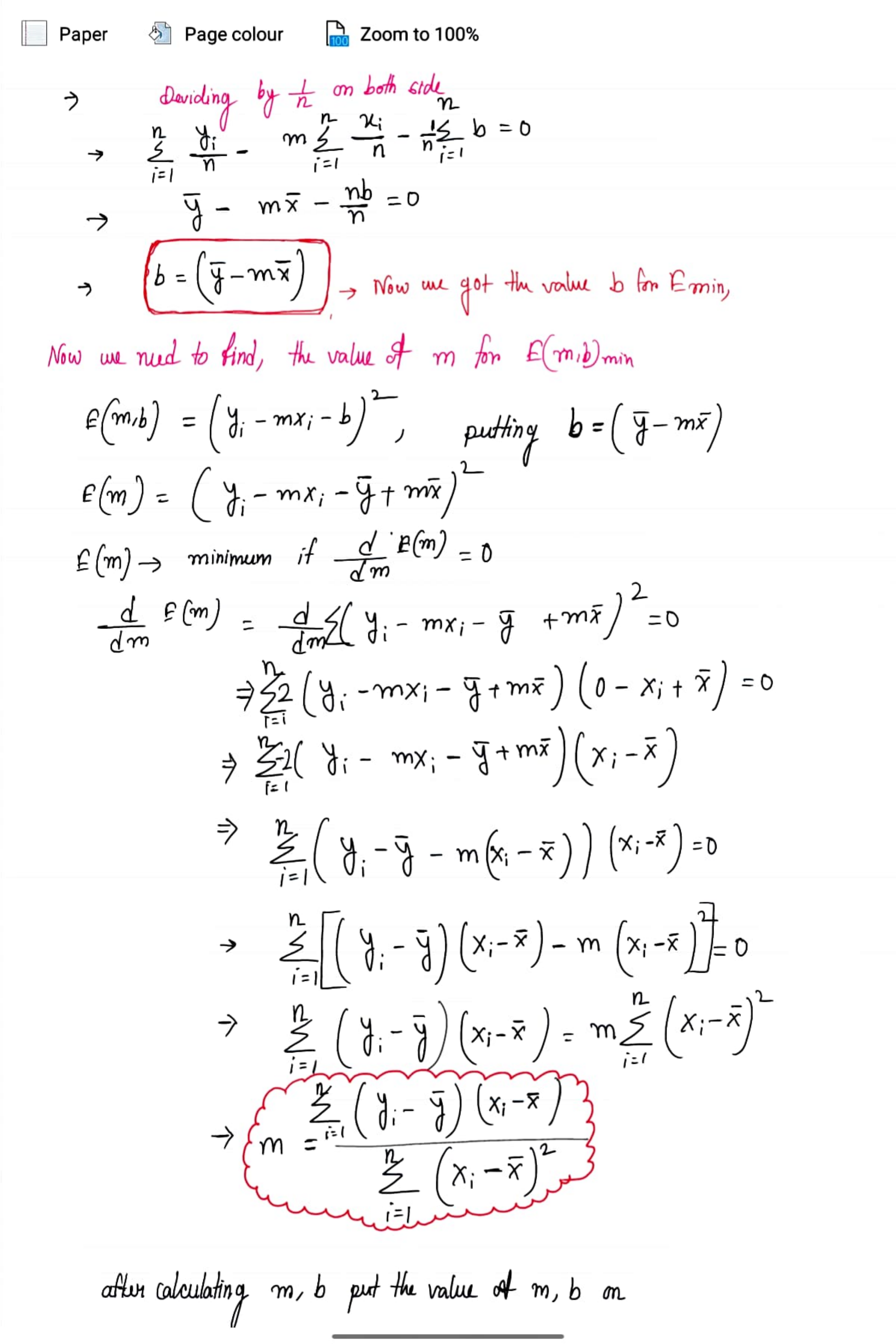
$$\frac{\partial E}{\partial b} = \frac{\partial}{\partial b} \sum_{i=1}^{n} (y_i - mx_i - b)^2 = 0$$

$$\rightarrow \sum_{i=1}^{n} 2(y_i - mx_i - b)(0 + 0 - 1) = 0$$

$$\rightarrow \sum_{i=1}^{n} -2(y_i - mx_i - b) = 0$$

$$\rightarrow \sum_{i=1}^{n} (y_i - mx_i - b) = 0$$

→ Deviding by $\frac{1}{n}$ on both side n



→ Deviding by $\frac{1}{n}$ on both side

$$\rightarrow \sum_{i=1}^{n} \frac{y_i}{n} - m\sum_{i=1}^{n} \frac{x_i}{n} - \frac{1}{n}\sum_{i=1}^{n} b = 0$$

$$\rightarrow \bar{y} - m\bar{x} - \frac{nb}{n} = 0$$

$$\rightarrow \boxed{b = (\bar{y} - m\bar{x})}$$

→ Now we got the value b for E min,

Now we need to find, the value of m for E(m,b) min

$E(m,b) = (y_i - mx_i - b)^2$, putting $b = (\bar{y} - m\bar{x})$

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

$E(m) \rightarrow$ minimum if $\frac{d}{dm}E(m) = 0$

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

$$\Rightarrow \sum_{i=1}^{n} 2(y_i - mx_i - \bar{y} + m\bar{x})(0 - x_i + \bar{x}) = 0$$

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

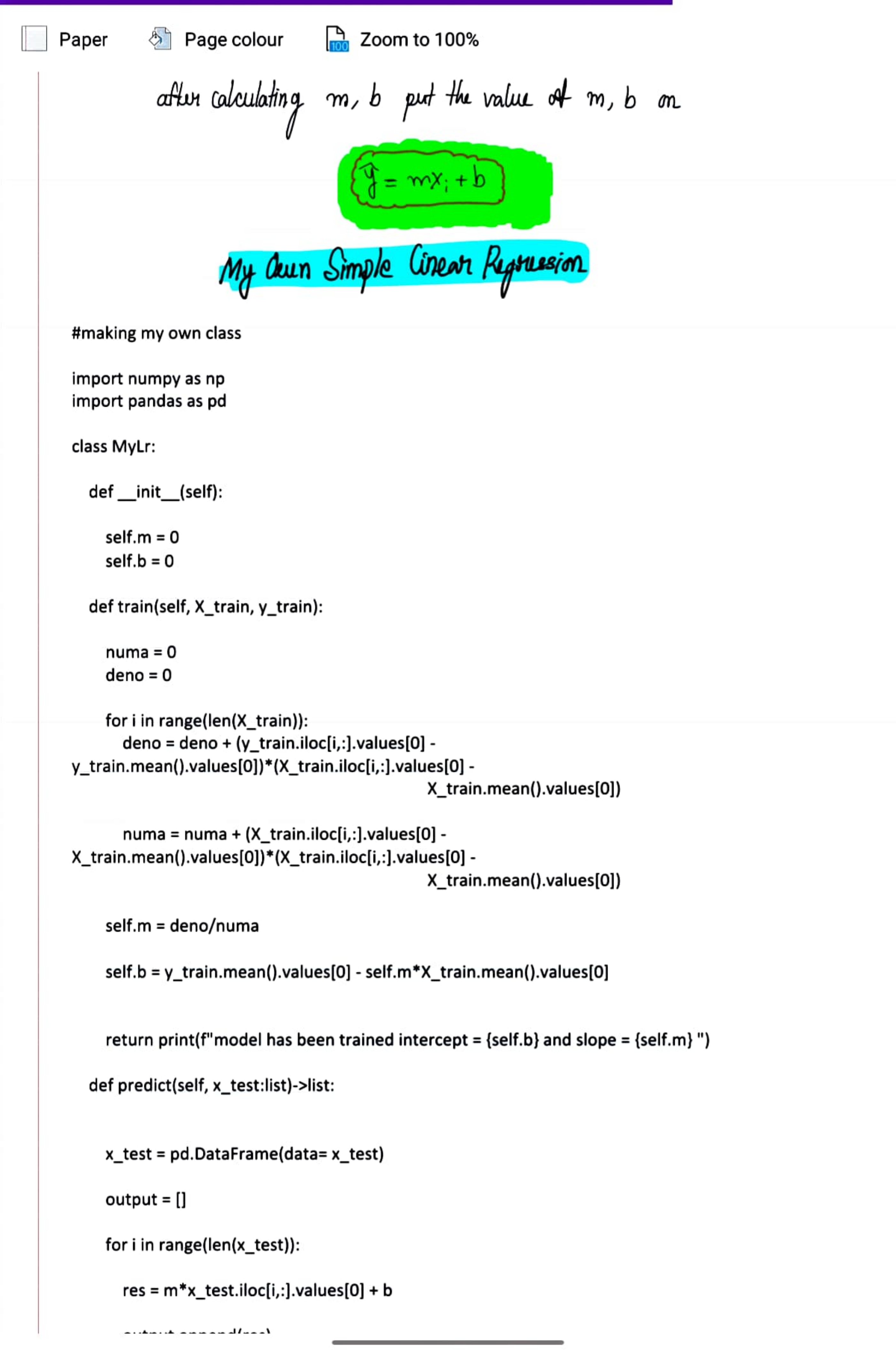
$$\Rightarrow \sum_{i=1}^{n}(y_i - \bar{y} - m(x_i - \bar{x}))(x_i - \bar{x}) = 0$$

$$\rightarrow \sum_{i=1}^{n}[(y_i - \bar{y})(x_i - \bar{x}) - m(x_i - \bar{x})^2] = 0$$

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

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

after calculating m, b put the value of m, b on



after calculating m, b put the value of m, b on

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

My Own Simple Linear Regression

```
#making my own class

import numpy as np
import pandas as pd

class MyLr:

   def __init__(self):

      self.m = 0
      self.b = 0

   def train(self, X_train, y_train):

      numa = 0
      deno = 0

      for i in range(len(X_train)):
         deno = deno + (y_train.iloc[i,:].values[0] -
y_train.mean().values[0])*(X_train.iloc[i,:].values[0] -
                                          X_train.mean().values[0])

         numa = numa + (X_train.iloc[i,:].values[0] -
X_train.mean().values[0])*(X_train.iloc[i,:].values[0] -
                                          X_train.mean().values[0])

      self.m = deno/numa

      self.b = y_train.mean().values[0] - self.m*X_train.mean().values[0]


      return print(f"model has been trained intercept = {self.b} and slope = {self.m} ")

   def predict(self, x_test:list)->list:


      x_test = pd.DataFrame(data= x_test)

      output = []

      for i in range(len(x_test)):

         res = m*x_test.iloc[i,:].values[0] + b

         output.append(res)
```

Paper Page colour Zoom to 100%

```
        res = m*x_test.iloc[i,:].values[0] + b

        output.append(res)

    return np.array([output]).reshape(len(output),1)
```

Multiple linear Regressions

In simple linnear regression we have only one input features. But what if if we have multiple input features?

The answear is → we need to calculate weights (like m) for each term.

simple linnear Regression

X	$\hat{Y}$
-	-
-	-
-	-
-	-
-	-
-	-
-	-

$$\hat{Y} = mx + b$$

or

$$\hat{Y} = \theta_1 x + \theta_0$$

Multiple linnear Regression

x_1 x_2 x_3 .... x_n	$\hat{Y}$

$$\hat{Y} = b + m_1x_1 + m_2x_2 + m_3x_3 \dots + m_nx_n$$

so, we need to only calculate the value of $(b, m_1, m_2 \dots m_n)$ with the help of Y test value (y)

we can also re-write the equations as

$$\hat{Y} = \theta_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_3 + \dots + \theta_nx_n$$

Derivation is not needed at the point. But I keeping Blank for Derivations

Data sample

x_1	x_2	x_3	x_n	Y
-	-	-	-	-

the equation of predicted value will be

$$\hat{Y} = \beta_0 + x_1\beta_1 + x_2\beta_2 + \dots x_n\beta_n \rightarrow \text{General Solution}$$

for each predictions.

$$\hat{y}_1 = \beta_0 + \beta_1x_{11} + \beta_2x_{12} \dots + \beta_n x_{1n}$$

$$\hat{y}_2 = \beta_0 + \beta_1x_{21} + \beta_2x_{22} \dots B_n x_{2n} \xrightarrow{\text{Converting it into matrix}} \hat{Y} = \begin{bmatrix} \beta_0 & \beta_1x_{11} & \beta_2x_{12} \dots & \beta_nx_{1n} \\ \beta_0 & \beta_1x_{21} & \beta_2x_{22} \dots & \beta_nx_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_0 & \beta_1x_{m1} & \beta_2x_{m2} & \beta_nx_{mn} \end{bmatrix}$$

$$\vdots$$

$$\hat{y}_m = \beta_0 + \beta_1x_{m1} + \beta_2x_{m2} \dots B_n x_{mn}$$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} \dots & x_{1n} \\ 1 & x_{21} & x_{22} & x_{2n} \\ 1 & x_{31} & x & x \end{bmatrix} * \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

Paper Page colour Zoom to 100%

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & & x_{2n} \\ 1 & x_{31} & x_{32} & & x_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{m1} & x_{m2} & & x_{mn} \end{bmatrix}_{(m,n+1)} * \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}_{(n+1,1)}$$

$$\boxed{\hat{Y}_{(m,1)} = X_{(m,n+1)} \beta_{(n+1,1)}} \text{—}(i)$$

Multiple Linear Regression equations in matrix form.

to calculate $\hat{Y}_{(m,1)}$ we have $X_{(m,n+1)}$, we just need to calculate $\beta_{(n+1,1)}$ matrix so, that the $\boxed{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2} = \text{minimum} \rightarrow (i)$

Transforming Ino functions into matrix form. so,

$$\sum_{i=1}^{n}(y_1-\hat{y}_1)^2 = \begin{bmatrix} (y_1-\hat{y}_1) \\ (y_2-\hat{y}_2) \\ (y_3-\hat{y}_3) \\ \vdots \\ (y_n-\hat{y}_3) \end{bmatrix} * \begin{bmatrix} (y_1-\hat{y}_1) & (y_2-\hat{y}_2) & (y_3-\hat{y}_3) & \cdots & (y_n-\hat{y}_n) \end{bmatrix}$$

$$\| \qquad\qquad = e^T \qquad\qquad \|$$

$$E \qquad\qquad e = [y_i - \hat{y}_i] \qquad\qquad e$$

$$E = e^Te = (y-\hat{y})^T(y-\hat{y})$$

$$E = (y^T - \hat{y}^T)(y-\hat{y})$$ we know that, $\hat{y} = X\beta$

$$[(y^T-(X\beta)^T)](y-X\beta)$$

$$E = y^Ty - (X\beta)^Ty - (X\beta)y^T + (X\beta)^T(X\beta) \qquad \because A^TB = B^TA$$

$$= y^Ty - (X\beta)y^T - (X\beta)y^T + (X\beta)^T(X\beta)$$

$$E = y^Ty - 2y^T(X\beta) + \beta^TX^TX\beta$$ [This will be minimum if $\frac{dE}{d\beta} = 0$]

$$\frac{dE}{d\beta} = \frac{d}{d\beta}\left[y^Ty - 2y^TX\beta + \beta^TX^TX\beta\right] = 0$$

$$= 0 - 2y^TX + \frac{d}{d\beta}\left[\beta^TX^TX\beta\right] = 0$$

$$\Rightarrow -2y^TX + 2X^TX\beta^T = 0$$

$$\rightarrow \cancel{2}X^TX\beta^T = \cancel{2}y^TX$$

$$^{-1}$$

$$\Rightarrow -2y^T x + 2x^T x \beta^T = 0$$

$$\rightarrow \cancel{2}x^T x \beta^T = \cancel{2}y^T x$$

$$\rightarrow \beta^T = y^T x (x^T x)^{-1}$$

$$\rightarrow (\beta^T)^T = \left[y^T x (x^T x)^{-1}\right]^T$$

$$\rightarrow \beta = \left[(x^T x)^{-1}\right]^T (x^T y)$$

$$\rightarrow \boxed{\beta = (x x^T)^{-1} (x^T y)}$$

Calculating shape of β

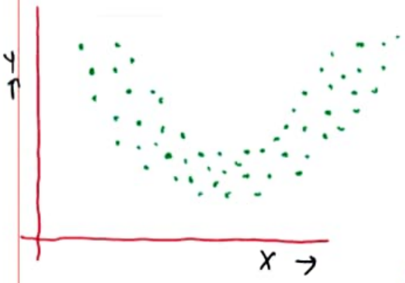
$$(n+1, n+1) \qquad \underline{(n+1, m)(m, 1)}$$

$$= (n+1, n+1)(n+1, 1)$$

$$\text{Shape}(\beta) = (n+1, 1)$$

Polynomial Functions

we have seen in Linear Regession that if the relation of dependant and indipendent variables is Linear then we are using Linear Regressions or multiple Linear Regressions. But what if? if we do not Linear relationship. if we have parbolic relationship Between input and output featurs. Like Below



→ Linear Regression will not work here.

need to use polynomial function to find the best fit Curve.

Polynomial functions are two types

Simple polynomial → Only one input features

multiple polynomial → Multiple input features.

Equations $\left(\hat{y} = b + m_1 x + m_2 x^2\right)$

$\left(\hat{y} = b + m_1 x_1 + m_2 x_1^2 + m_3 x_2^2 + m_4 x_1 x_2 + m_5 x_2\right)$

It basically Convert input functions into polynomial equations then apply Linear regressions.

(Converting X_1 into x_1, x_1^2
Converting X_2 into x_1, x_2^2)

Sk-learn have a methods to convert inputs into a polynomia equations.

from sklearn.preprocessing import polynomial features.

polinomialobject = Polynomialfeatures.fit_transform ;

:- python code :-

it return an standardrise form of input data and then convert int an polynomial functions with degree which you specified

polyno_x_data = polynomial_object.fit_transform (x_data)

Standard_Ydata = polynomial_object.transform (Y data)

Conversation of single input column into polynomial function.

X input	y output
x_1	-
x_2	-
x_3	-

Converting →

input			output
$\theta_0 x_1^0$	$\theta_1 x_1^1$	$\theta_2 x_1^2$	y
$\theta_0 x_2^0$	$\theta_1 x_2^1$	$\theta_2 x_2^2$	-
θx_3^0	$\theta_1 x_3^1$	$\theta_2 x_3^2$	-
θx_3^0	$\theta_1 x_3^1$	$\theta_2 x_3^2$	-

if $y = f(x)$ → is polynomial of degree 2

I am creating my own Linear Regression model

In [419]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings("ignore")
import matplotlib as plt
import seaborn as sns
```

In [420]:

```
from sklearn.datasets import load_boston
boston_data = load_boston()
values = boston_data.data
columnss = boston_data.feature_names
boston_df = pd.DataFrame(data= values, columns=columnss)
```

In [421]:

```
target = pd.DataFrame(boston_data.target)
target.set_axis(['price'], axis=1,inplace=True)
```

Simple Linear regressions

In [422]:

```
#Average room per house as x data fpr linear regression.
X = boston_df['RM']
y = target
```

In [423]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

In [424]:

```
SkLr = LinearRegression()
```

In [425]:

```
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)
```

In [426]:

```
SkLr.fit(X_train,y_train)

y_predict = SkLr.predict(X_test)

from sklearn.metrics import r2_score

r2_score(y_test, y_predict)

print(SkLr.intercept_, SkLr.coef_)
```

```
[-32.55158437] [[8.74934434]]
```

Creating My own LR Class

In [427]:

```
#making my own class

import numpy as np
import pandas as pd

class MyLr:

    def __init__(self):

        self.m = 0
        self.b = 0

    def train(self, X_train, y_train):

        numa = 0
        deno = 0

        for i in range(len(X_train)):
            deno = deno + (y_train.iloc[i,:].values[0] - y_train.me


            numa = numa + (X_train.iloc[i,:].values[0] - X_train.me


        self.m = deno/numa

        self.b = y_train.mean().values[0] - self.m*X_train.mean().v


        return print(f"model has been trained intercept = {self.b}

    def predict(self, x_test:list)->list:


        x_test = pd.DataFrame(data= x_test)

        output = []

        for i in range(len(x_test)):

            res = self.m*x_test.iloc[i,:].values[0] + self.b

            output.append(res)

        return np.array([output]).reshape(len(output),1)
```

In [428]:

```
myobj = MyLr()
```

```
In [429]: myobj.train(X_train, y_train)
```

```
model has been trained intercept = -32.5515843678096 and slope = 8
.749344338735002
```

```
In [430]: print(f"Intercept and cofficent from sk learn model {SkLr.intercept
          print(f"Intercept and cofficent from my own created model {myobj.b}
```

```
Intercept and cofficent from sk learn model [-32.55158437]  [[8.74
934434]]
Intercept and cofficent from my own created model -32.551584367809
6  8.749344338735002
```

See both the value is same. Hurray I have made my own LR algorithm!

```
In [431]: print(f"r_2 score of Scikit Learn model = {r2_score(y_test, SkLr.pr
          print(f"r_2 score of my own created model  = {r2_score(y_test, myob
```

```
r_2 score of Scikit Learn model = 0.6335439948424493
r_2 score of my own created model  = 0.6335439948424488
```

See both the value is same. Hurray I have made my own LR algorithm!

Multiple Linear Regressions.

I can create my own MLR model but in sk learn it already available

```
In [432]: X_train,X_test,y_train, y_test = train_test_split(boston_df, target
```

```
In [433]: mlr = LinearRegression()
          mlr.fit(X_train, y_train)
          y_predict = mlr.predict(X_test)
          print(f" The r2 score is {r2_score(y_test, y_predict)}")
```

```
 The r2 score is 0.7789207451814428
```

Ploynomial Linear Regressions.

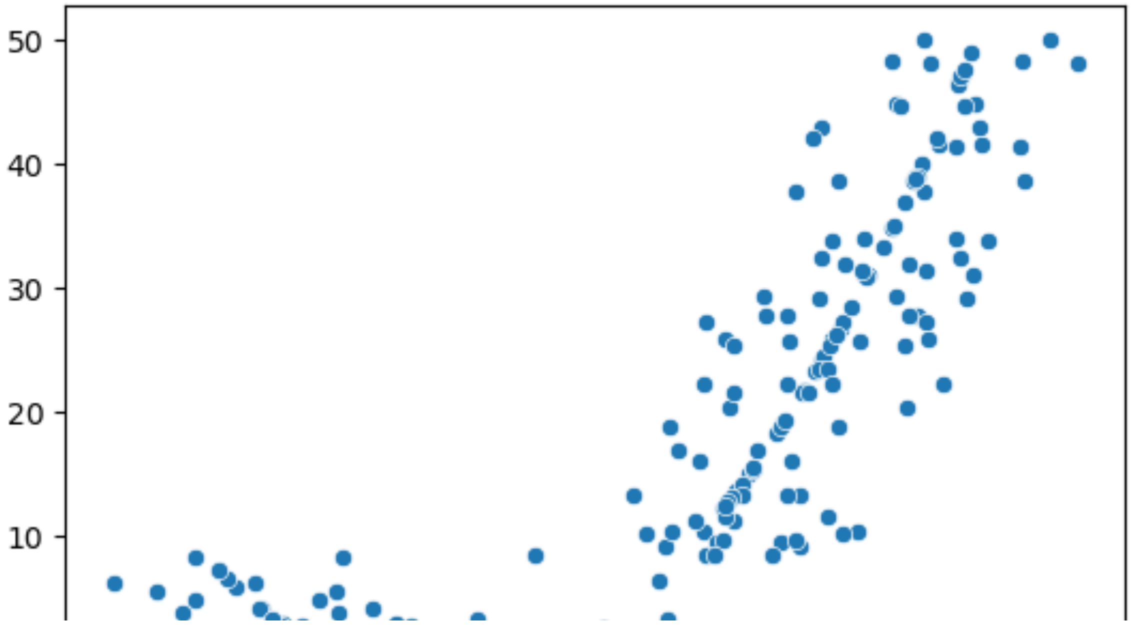
Makung a polynomially related X and Y Data Point.

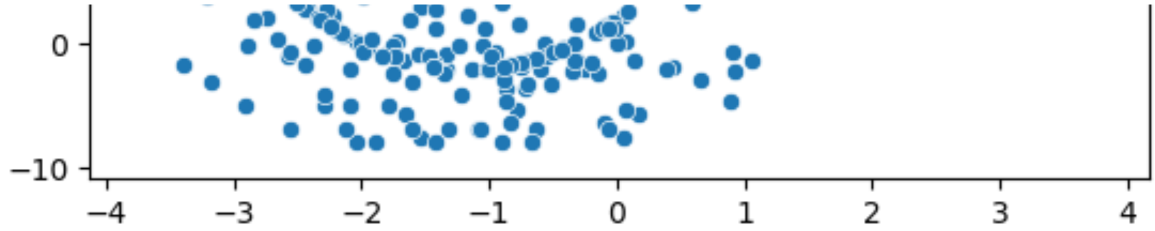
In [476]:

```
a = np.random.randint(2,10)
b = np.random.randint(2,10)
c = np.random.randint(2,10)

X = []
Y = []
for i in range(100):
    x = np.random.randint(-3,3) + np.random.random(1)
    x_noise1 = x[0] - np.random.random(1)[0]
    x_noise2 = x[0] + np.random.random(1)[0]


    X.append(x[0])
    X.append(x_noise1)
    X.append(x_noise2)

    y = a*x*x + b*x + c
    y_noise1 = y[0]+ np.random.randint(-6,5)
    y_noise2 = y[0]+ np.random.randint(-6,5)


    Y.append(y[0])
    Y.append(y_noise1)
    Y.append(y_noise1)

sns.scatterplot(X,Y)

# Ploynomyal Functions is created you can see in graph
```

Out[476]:

```
<AxesSubplot:>
```





In [477]:

```
X = pd.DataFrame(X)
Y = pd.DataFrame(Y)
```

In [478]:

```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=
```

We will use 1st linear regression then calculate the r2_score and then Polynomial Regressions then Calculate r_score then we compare

In [479]:

```
Linear_model = LinearRegression()
Linear_model.fit(X_train, y_train)
y_predict = Linear_model.predict(X_test)
r2_score(y_test, y_predict)
```

Out[479]:

```
0.6766514586288175
```

Can see very low accuracy

Now we will use polynomial linear regressions.

In [480]:

```
from sklearn.preprocessing import PolynomialFeatures
```

In [481]:

```
polynomial_object = PolynomialFeatures(degree=2)

X_train = polynomial_object.fit_transform(X_train)

X_test = polynomial_object.transform(X_test)

X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)
```

In [482]:

```
Lin_model = LinearRegression()

Lin_model.fit(X_train, y_train)

y_predict = Lin_model.predict(X_test)

r2_score(y_test, y_predict)
```

Out[482]:

```
0.8513585489807223
```

In [ ]:

```
# See very good score
```