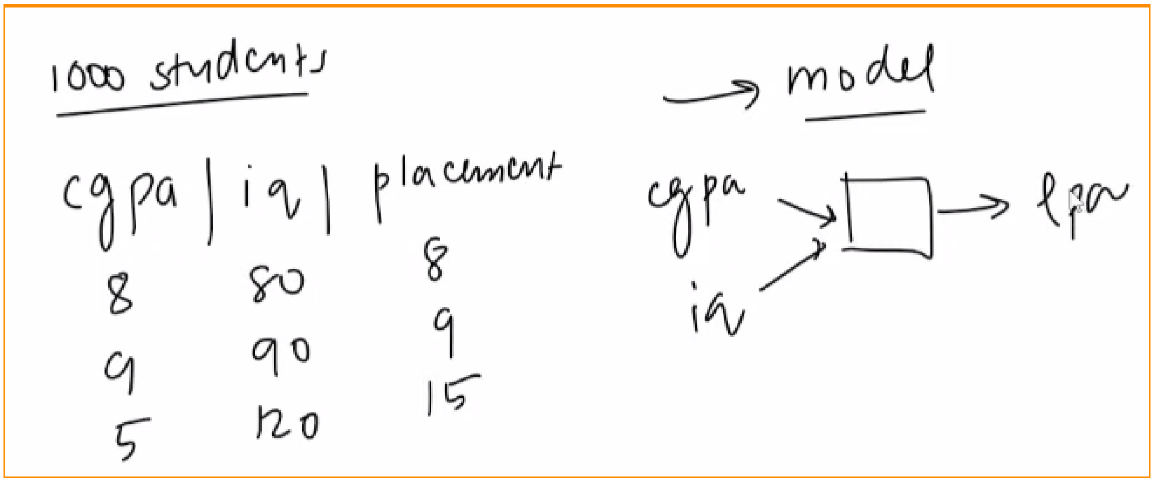
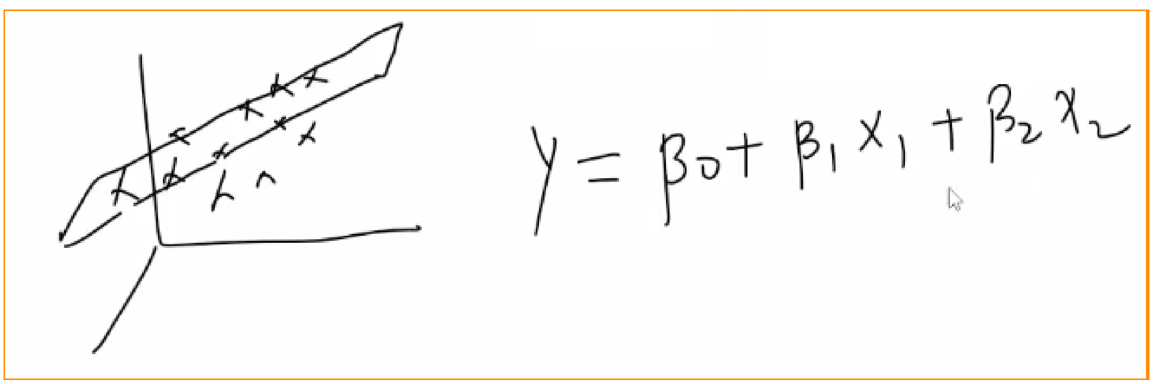
Multiple Linear Regression (OLS Method)



- Lets assume we have dtaa of 100 students & we need to create a model to predict package amount
- since we have more then 1 input cols, we will apply Multiple Linear Regression (which is a extension of Simple Linear Regression)
- In SLR, we try to find the line in a 2d environment which passes very closely from each data point
- but now we have more then 1 input cols, we have 3d environment & here we dont draw a line, instead we draw a plane



- where:
  - x1 cgpa
  - x2 iq
  - b0 intercept
  - b1 slope of x1
  - b1 slope of x2
- Same equation can be used for any number of cols by extending it to n cols
- below is equation of **hyperplane** in n dimensional environment
- In Multiple Linear Regression, we draw a hyper plane in higher dimension (n) which passes very closely from each data point

- In SLR we find the value of m & b and MLR we find the value of those β0, β1, ... βn for which loss function is very less means its passing very very closely possible from all data points
- β1, β2, ... βn are weighatges which tells how much target value relies on these cols
- lets say β1 is slope of cgpa & β2 is slope of iq, β1 is very high & β2 is low then it means package depends more on cgpa then iq

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n$$

Python Implimentation

In [2]:
```
from sklearn.datasets import make_regression
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [3]:
```
#creating dummy data
X,y = make_regression(n_samples=100, n_features=2, n_informative=2, n_targets=1, nois
```

In [4]:
```
df = pd.DataFrame({'feature1':X[:,0], 'feature2':X[:,1], 'target':y})
```

In [5]:
```
df.shape
```

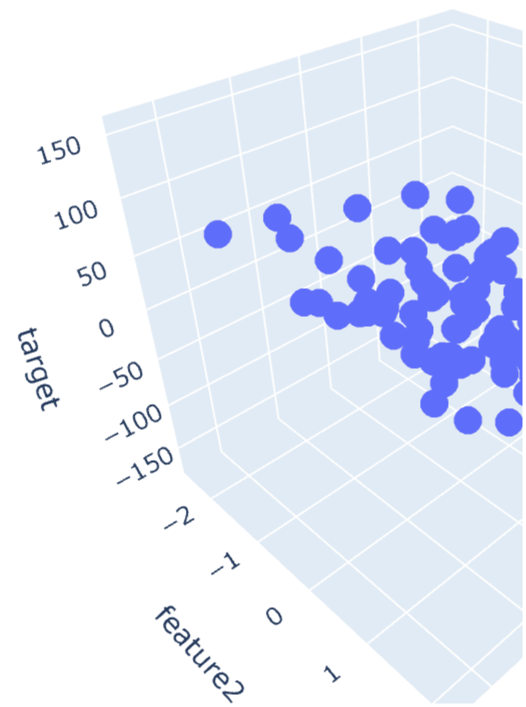
Out[5]:
```
(100, 3)
```

In [6]:
```
df.sample(3)
```

Out[6]:

	feature1	feature2	target
63	0.426679	0.112665	-46.196966
51	-0.505986	-0.482620	-84.460392
46	-0.053685	0.165948	30.636091

In [7]:
```
fig = px.scatter_3d(df, x='feature1', y='feature2', z='target')
fig.show()
```



In [8]:
```
#data is sort of linear so we can apply linear regression
```

In [9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=3
```

In [10]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [11]:
```
lr.fit(X_train,y_train)
```

Out[11]:
```
▾ LinearRegression
LinearRegression()
```

In [12]:
```
y_pred = lr.predict(X_test)
```

In [13]:
```
print("MAE: ", mean_absolute_error(y_test, y_pred))
print("MSE: ", mean_squared_error(y_test, y_pred))
print("R2 Score: ", r2_score(y_test, y_pred))
```

```
MAE:  30.071698510378713
MSE:  1505.808659888552
R2 Score:  0.7866799371185449
```

Our goal is to minimize 👆👆👆 loss function like MAE or MSE & Increase r2 score

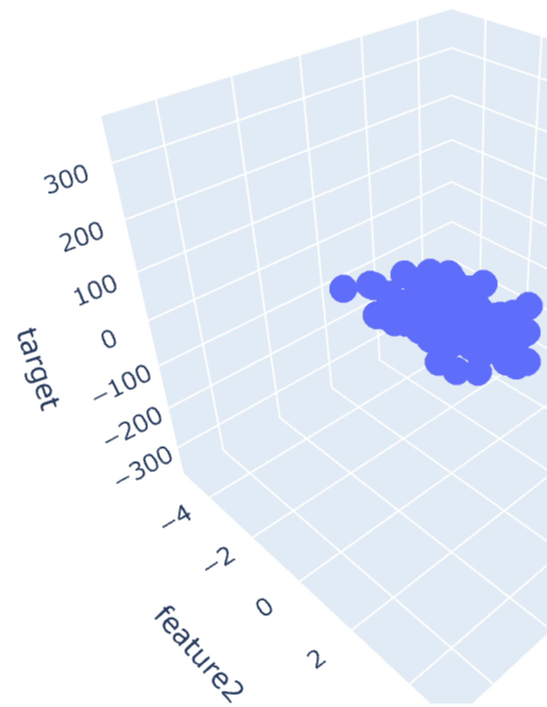
In [17]:
```
# Sample data points (creating a mesh grid)
x = np.linspace(-5, 5, 10)
y = np.linspace(-5, 5, 10)
xGrid, yGrid = np.meshgrid(y,x)

z_final = lr.predict(final).reshape(10,10)
z = z_final

# Reshape grid points for stacking
xGrid_flat = xGrid.ravel().reshape(1,100)
yGrid_flat = yGrid.ravel().reshape(1,100)

# Stack grid points into final input array
final = np.vstack((xGrid_flat, yGrid_flat)).T
```

In [19]:
```
fig = px.scatter_3d(df, x='feature1',y='feature2', z='target')
fig.add_trace(go.Surface(x=x,y=y, z=z))
fig.show()
```



In [20]:
```
lr.coef_
```

Out[20]:
```
array([51.0173524 , 19.16775978])
```

In [22]:
```
# there are 2 coefficient values β1, β2
```

In [24]:
```
lr.intercept_   #β0
```

Out[24]:
```
4.496459252446482
```

Mathematical Formulation

- Lets assume a test data of 3 students on which we have to apply linear regression

cgpa	iq	placement
8	80	8
7	70	7
5	120	15

- CGPA --> x1
- IQ --> x2
- Lets also assume that we already know the values of β0, β1, β2

	x_1 cgpa (β_1)	x_2 iq (β_2)	y placement
β_0	8	80	8
	7	70	7
	5	120	15

- And I want to do prediction for these same students using ml model even though we know the placement package

$$y_1 = 8 \quad y_2 = 7 \quad y_3 = 15$$

$$\hat{y}_1 = ? \quad \hat{y}_2 = ? \quad \hat{y}_3 = ?$$

$$\hat{y}_1 = \beta_0 + \beta_1 8 + \beta_2 80$$
$$\hat{y}_2 = \beta_0 + \beta_1 7 + \beta_2 70$$
$$\hat{y}_3 = \beta_0 + \beta_1 5 + \beta_2 120$$

- If we write above row values in numpy notation form then it will be X11, X12, X21, X22... and so on:

β_0	X_1 cgpa (β_1)	X_2 iq (β_2)	y placement
	8 X_{11}	80 X_{12}	8
	7 X_{21}	70 X_{22}	7
	5 X_{31}	120 X_{32}	15

- Prediction formula for 1st, 2nd & 3rd student will be:

$$\hat{y}_1 = \beta_0 + \beta_1 X_{11} + \beta_2 X_{12}$$
$$\hat{y}_2 = \beta_0 + \beta_1 X_{21} + \beta_2 X_{22}$$
$$\hat{y}_3 = \beta_0 + \beta_1 X_{31} + \beta_2 X_{32}$$

- lets assume we have **m** columns in input instead of just 2 so the predition formula will become:

$$
\begin{aligned}
\hat{y}_1 &= \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13} + \beta_4 x_{14} + \dots + \beta_m x_{1m} \\
\hat{y}_2 &= \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_m x_{2m} \\
\hat{y}_3 &= \beta_0 + \beta_1 x_{31} + \beta_2 x_{32} + \dots + \beta_m x_{3m}
\end{aligned}
$$

- lets also assume we have **n** students data instead of just 3 so the prediction formula will become:

$$
\begin{aligned}
\hat{y}_1 &= \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13} + \beta_4 x_{14} + \dots + \beta_m x_{1m} \\
\hat{y}_2 &= \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_m x_{2m} \\
\hat{y}_3 &= \beta_0 + \beta_1 x_{31} + \beta_2 x_{32} + \dots + \beta_m x_{3m} \\
\hat{y}_4 & \\
\vdots & \\
\hat{y}_n &= \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \dots + \beta_m x_{nm}
\end{aligned}
$$

- To simplify above we can make a numpy metrix:

$$
\boxed{\hat{Y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} =
\begin{matrix}
\beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13} + \beta_4 x_{14} + \dots + \beta_m x_{1m} \\
\beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_m x_{2m} \\
\beta_0 + \beta_1 x_{31} + \beta_2 x_{32} + \dots + \beta_m x_{3m} \\
\\
\beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \dots + \beta_m x_{nm}
\end{matrix}
$$

- shape of above both metrix will be n x 1
- we can represent above as 2 metrices & get above when we do dot product of both

$$
= \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & & & & \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}
$$

- It can be represented like below where X holds all input values & β holds all beta component
- this equation is true for any number of dimension.
- **This is 1st equation**

$$
\hat{y} = X\beta
$$

- In SLR, we try to find a line which passes very closely from each data point & our goal is to minimize the loss which is the square of sum of difference of actual vs predicted value
- This is true in MLR also, we're trying to minimize the square of sum of vertical distance b/w plane (predicted value) & actual value
- Error function remains the same

min

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

- we need to use the same equation but need to write in matrix form since we're dealing with multiple columns
- storing actual placement value as Y

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- I also have predicted value as $\hat{Y}$

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix}$$

- lets create a new metrix e:

$$e = Y - \hat{Y} = \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ \vdots \\ y_n - \hat{y}_n \end{bmatrix}$$

- lets create eTe (e-transpose-e)

$$e^Te = \begin{bmatrix} y_1-\hat{y}_1 & y_2-\hat{y}_2 & \cdots & y_n-\hat{y}_n \end{bmatrix}_{1\times n} \begin{bmatrix} y_1-\hat{y}_1 \\ y_2-\hat{y}_2 \\ \vdots \\ y_n-\hat{y}_n \end{bmatrix}$$

$$\overset{1\times 1}{e^Te = (y_1-\hat{y}_1)^2 + (y_2-\hat{y}_2)^2 + \cdots + (y_n-\hat{y}_n)^2}$$

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

$$\boxed{E = e^Te}$$

- **This is the 2nd equation**
- We have to minimize E (loss function).. we need that hyper plane which will minimize the value of E

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

- Upon simplifying it further.. we get:

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

$$E = y^T y - y^T \hat{y} - \hat{y}^T y + \hat{y}^T \hat{y}$$

- These 2 metrices in above formula is same:

$$\boxed{y^T \hat{y} - \hat{y}^T y}$$

- Lets assume them as A & B:

$$y = A \qquad \hat{y} = B$$

$$A^T B = B^T A$$

$$(A^T B)^T = B^T A$$

$$(AB)^T = B^T A^T$$

$$(A^T)^T = A$$

- so we have to proove this:

$$\underline{A^T B} = (\underline{A^T B})^T$$

- if we consider it as C then we have to proove:

$$A^T B = C$$

$$C = C^T$$

- so we have to proove AtB is a symmetric matrix
- A = Y & B = Y-hat

$$Y^T \hat{Y}$$

- Y-hat = Xβ

$$Y^T \hat{Y} = Y^T X \beta$$

$A^T B$ is symmetric

$1 \times 1 \rightarrow$ Scalar

$[1]$ $[2]$

$$y^T \hat{y} = y^T X \beta$$

$A^T B$ is sym

$(1 \times n)$ $n \times (m+1)$ $(m+1) \times 1$

$1 \times n$ $n \times 1$

- If its is symmetric then we have proved below relationship:

$$\boxed{y^T \hat{y} = \hat{y}^T y}$$

- So we can re-write this equation:

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

$$E = y^T y - \boxed{y^T \hat{y} - \hat{y}^T y} + \hat{y}^T \hat{y}$$

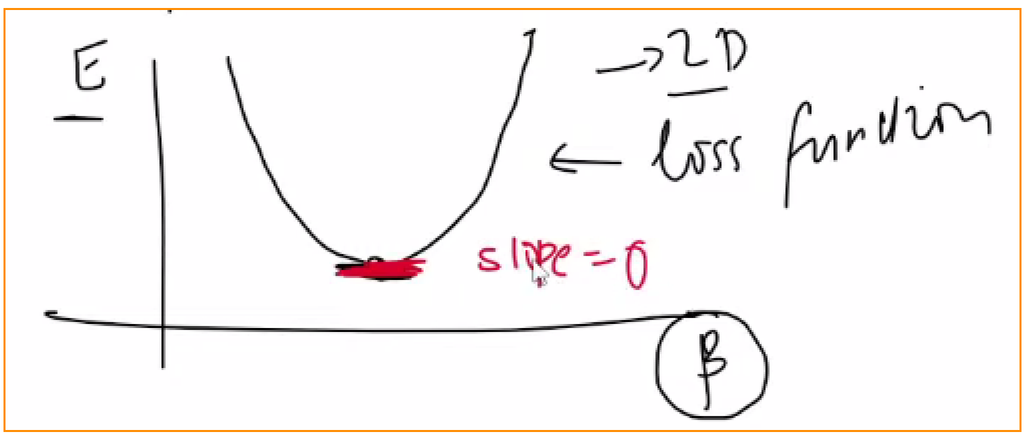
$$E = y^T y - 2 y^T \hat{y} + \hat{y}^T \hat{y}$$

- **This is 3rd equation**
- Replacing Y-hat with Xβ

$$E = y^T y - 2y^T X\beta + (X\beta)^T (X\beta)$$

$$\boxed{E = y^T y - 2y^T X\beta + \beta^T X^T X\beta} \quad eq^n\ 4$$

- **This is equation #4**
- now we can understand that error function is a function of β
- like y = f(x) if we change x then f(x) will change & it will change y
- similarly when we change β then error function will change bcoz y is output & x is input which will not change
- β is what can change
- now we have to find such value of β for which value of E is minimum
- Loss function of Linear Regreesion varies like this:



- so we have to arrive at the minimum point
- minimum point is a point where slope == 0
- so we have to diffrentiate loss function w.r.t to β & make it == 0

$$\frac{dE}{d\beta} = 0$$

- solve it for β & then we will get that value of β for which E is minimum
- so we have to diffrentiate below equation:

$$\boxed{E = y^T y - 2y^T X\beta + \beta^T X^T X\beta}$$

- there is diffrentiation rule:

$$\frac{d}{d\beta}\,\beta^T \underbrace{X^T X}\beta$$

$$\downarrow$$

$$x^T A x \qquad 2x^T A$$

$$\boxed{2\beta^T X^T X}$$

- Above is only true when A is symmetric
- so our diffrentiation will look like this:

$$\frac{dE}{d\beta} = 0 - 2y^T X + 2\beta^T X^T X = 0$$

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

$$\boxed{\beta^T X^T X = y^T X}$$

- we just need to find the value of β
- we'll multiply both side with inverse of XtX

$$\beta^T \underbrace{X^T X (X^T X)^{-1}} = y^T X (X^T X)^{-1}$$

$$\underbrace{\beta^T I} = y^T X (X^T X)^{-1}$$

- when mulitply with identity metrix we get the same value..

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

- Applying transpose both side

$$\underline{(\beta^T)}^T = \Big[\underbrace{y^T X}_{A} \big(\underbrace{X^T X}_{B}\big)^{-1}\Big]^T$$

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

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

- we will proove that this is symmetric:

$(X^T X)^{-1}$ symmetric

$$[(X^T X)^{-1}]^T = (X^T X)^{-1}$$

- so after that our final equation will become

$$\beta = [(X^T X)^{-1}]^T (Y^T X)^T$$

$$\beta = [(X^T X)^{-1}]^T X^T Y$$

$$\beta = (X^T X)^{-1} X^T Y$$

- **This is equation #5**
- This is called OLS (Ordinary Least Squares) method

Python Implimentation

In [1]:
```
import numpy as np
from sklearn.datasets import load_diabetes
```

In [2]:
```
X,y = load_diabetes(return_X_y=True)
```

In [3]:
```
X
```

Out[3]:
```
array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
         0.01990749, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
        -0.06833155, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
         0.00286131, -0.02593034],
       ...,
       [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
        -0.04688253,  0.01549073],
       [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
         0.04452873, -0.02593034],
       [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
        -0.00422151,  0.00306441]])
```

In [4]:
```
y
```

Out[4]:
```
array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
        69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
        68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
        87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
       259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
       128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
       150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
       200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
        42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
        83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
       104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
       173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
       107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
        60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
       197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
        59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
       237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
       143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
       142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
        77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
        78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
       154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
        71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  91.,
       150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 116.,
       145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  66.,
        94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 233.,
        60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  89.,
        31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 172.,
       114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 109.,
       191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 135.,
       244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 216.,
       263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  71.,
        77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 321.,
        58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 168.,
       140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  69.,
       219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 258.,
        43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  72.,
       140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  55.,
        84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 310.,
        94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 132.,
       220.,  57.])
```

In [5]:
```
X.shape
```

Out[5]:
```
(442, 10)
```

In [6]:
```
#since there are 10 cols we need 10 beta values
#it will 10 coefficients & 1 intercnenpt
```

In [7]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=
```

In [8]:
```
X_train.shape
```

Out[8]: (353, 10)

In [10]:
```
X_test.shape
```

Out[10]: (89, 10)

In [11]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [12]:
```
lr.fit(X_train, y_train)
```

Out[12]:
```
LinearRegression
LinearRegression()
```

In [13]:
```
y_pred = lr.predict(X_test)
```

In [14]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[14]: 0.439933866156897

In [15]:
```
lr.coef_
```

Out[15]:
```
array([  -9.15865318, -205.45432163,  516.69374454,  340.61999905,
       -895.5520019 ,  561.22067904,  153.89310954,  126.73139688,
        861.12700152,   52.42112238])
```

In [16]:
```
lr.intercept_
```

Out[16]: 151.88331005254167

Problem with OLS

- There are 2 method to solve Linear Regression Problems

  1. OLS: (Closed form solution): where we get the value of beta & formula
  2. Gradient Descent (Non closed form solution): There is no formula.. this is an approximation technique where we converge & arrive very close to correct solution

- The Ordinary Least Squares (OLS) method can be quite time-consuming because it requires computing the inverse of a matrix to find the optimal solution. This computational effort can be manageable with smaller datasets, but as the dataset grows, the process becomes increasingly cumbersome.

- In contrast, Gradient Descent is generally more efficient and scalable for larger datasets. This optimization algorithm approximates the solution iteratively, avoiding the need for matrix inversion and thus speeding up the computation process.

In [ ]: