In [1]:
```
# Gradient Descent in Machine Learning  ===>

# we know that equation of simple line => y = mx + b

# we know that
# loss function = Actual_value - Predicted_value
# loss with respect to mean squared error
# l = 1/n(y_actual - y_predicted)**2

# if n(features) = 1

# l = (y - mx-b)***2

# Case - 1 loss w.r.t intercept(b)

# dl/db = 2(y - mx-b)(d(y)/db - d(mx)/db - db/db)
# dl/db = 2(y - mx-b) (0-0-1)
# dl/db = -2(y - mx-b)  ........(1)

# Case - 1 loss w.r.t weights

# dl/dm = 2(y - mx-b)[(dy/dm - d(mx)/dm - db/dm)]
# dl/dm = 2(y - mx-b) [0-x-0]
# dl/dm = -2(y - mx-b)(x) .........(2)

# b_new = b_old - learning_rate * (dl/db)  .......(3)
# m_new = m_old - learning_rate ((dl/dm))  .......(4)

# from equation (3,4) newline will be

# y = m_new*X + b_new  ........(5)

# if learning_rate  >>>>0  then it will be GradientExploading

# if learning_rate  <<<<0  then it will be Vanishig Gradient.
```

Now this time to check both on m and b

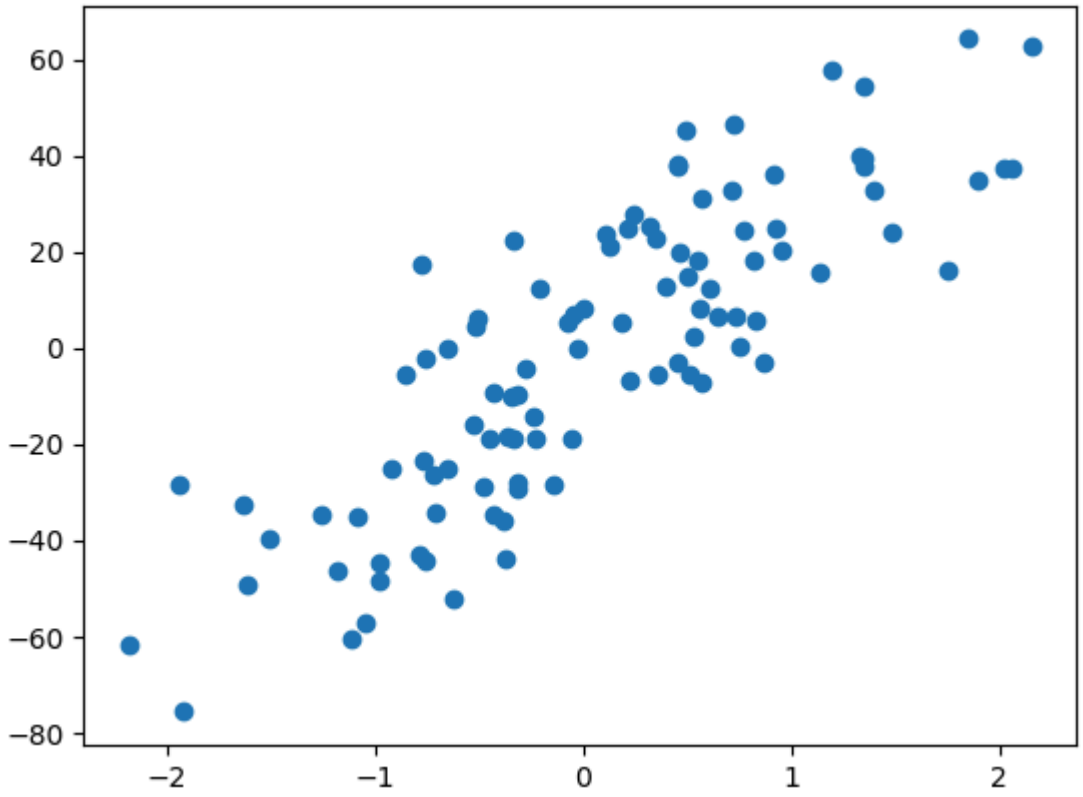
In [2]:
```
import numpy as np
```

In [3]:
```
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import cross_val_score
```

In [4]:
```
X,y = make_regression(n_samples = 100 , n_features = 1 ,
                      n_informative = 1 , n_targets = 1 , noise = 20 , random_st
```

In [5]:
```
plt.scatter(X,y)
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x25d9f66a410>
```



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 , ra
```

In [8]:
```
from sklearn.linear_model import LinearRegression
```

In [9]:
```
lr = LinearRegression()
```

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

```
[28.12597332]
-2.2710144261783816
```

In [13]:
```
# np.mean(cross_val_score(lr , X,y,scoring = 'r2' , cv = 10))

y_pred = lr.predict(X_test)
from sklearn.metrics import r2_score
r2_score(y_test , y_pred)
```

Out[13]:
```
0.6345158782661013
```

In [17]:
```
class GDRegressor :

    def __init__(self , learning_rate , epochs):
        self.m = 100  # we can start any number as like m = 0
        self.b = -120  # we can start any number as like b = 1
        self.lr = learning_rate
        self.epochs = epochs

    def fit(self , X,y):
        # Caluculate the b using GD
        for i in range(self.epochs):
            loss_slope_b = -2*np.sum(y - self.m*X.ravel() - self.b)
            loss_slope_m = -2*np.sum((y - self.m*X.ravel() - self.b)*X.ravel())

            self.b = self.b - (self.lr * loss_slope_b)
            self.b = self.m - (self.lr * loss_slope_m)
        print(self.b , self.m)

    def predict(self , X):
        return self.m * X + self.b
```

In [18]:
```
gd = GDRegressor(0.001 , 100)
```

In [19]:
```
gd.fit(X_train , y_train)
```

```
88.35787453246333 100
```

In [20]:
```
# gd.predict(X)
```

In [21]:
```
y_pred = gd.predict(X_test)
from sklearn.metrics import r2_score
r2_score(y_test , y_pred )
```

Out[21]:
```
-20.93645329631517
```

In [ ]: