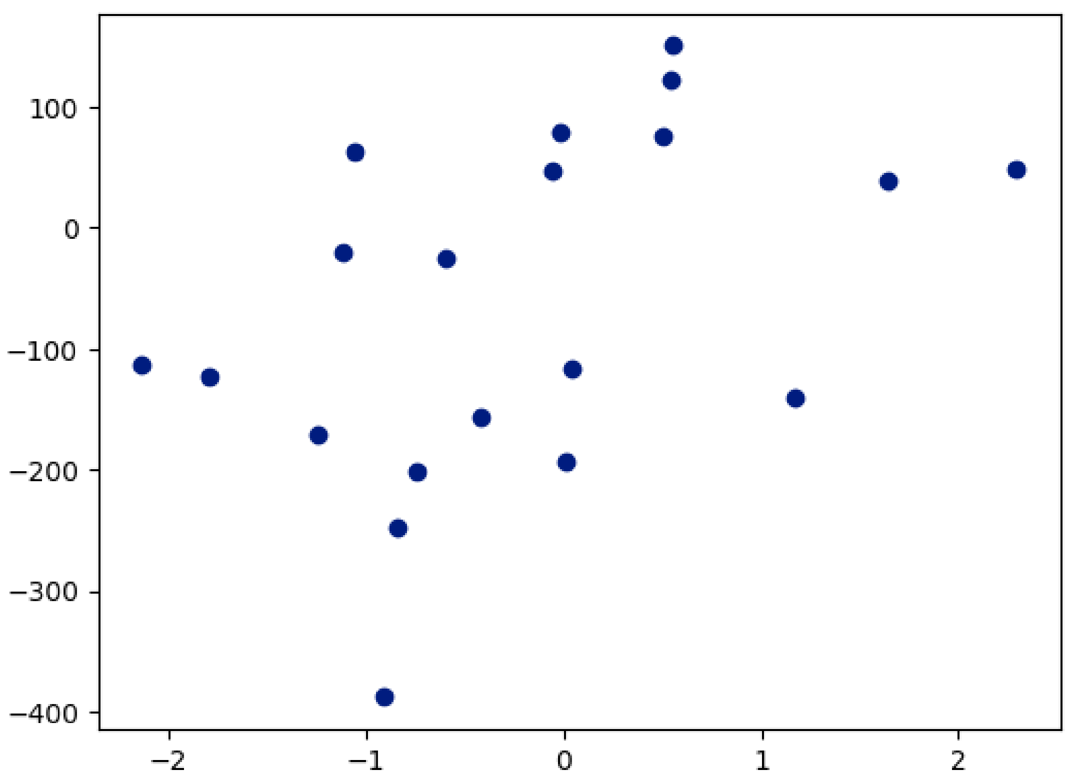
```
# Importing all the required liabraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression

# Using make_regression class of sklearn.datasets to generate a dummy
dataset

x,y = make_regression(n_samples=20 , n_features=1 , n_targets=1 ,
n_informative=1 , noise=100 , random_state=2)

# Plotting the dataset on a graph
plt.scatter(x,y)
```

```
<matplotlib.collections.PathCollection at 0x79043acd75e0>
```




```
# First training the model so that initially we get the values of
actual parameters so that later on we can com
lr = LinearRegression()
lr.fit(x,y)
```

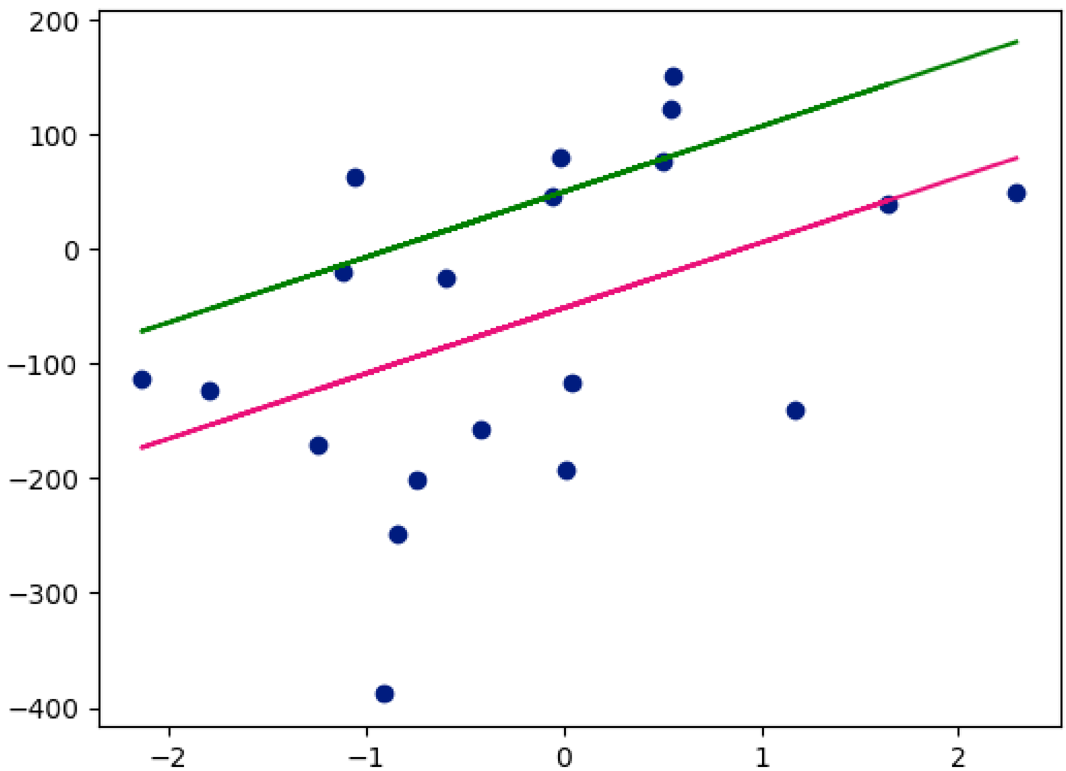
```
LinearRegression()
```

```python
# Getting the parameter values according the equation -> (mx + b)
# m value
m = lr.coef_

# b value
b = lr.intercept_

print(m)
print(b)
```

```
[57.03069586]
-51.38704024529875
```

```python
# Now we will see as we iterate to update the values of parameters how they move to local minima

# Firstly we will see the study with respect to b paramater so we assume that m is constant at this moment which we are going to take m = 57.03

# Let us first discuss how we actually perform Gradient Descent

# Step-1 : start with random value of b
# Step-2 : In iterations(eg. 500) or till b(new) - b(old) becomes very small , perform blow step:
#                          b(new) = b(old) - learning_rate * slope_of_function_at_particular_value_of_parameter
# Step-3 : finally we have got optimized value of our parameter

b = 50

plt.style.use('seaborn-v0_8-dark-palette')
plt.scatter(x,y)
plt.plot(x,lr.predict(x),color="#EA1179")
y_pred = m*x + b
plt.plot(x,y_pred,color='green')
```

```
[<matplotlib.lines.Line2D at 0x79043a8b96f0>]
```

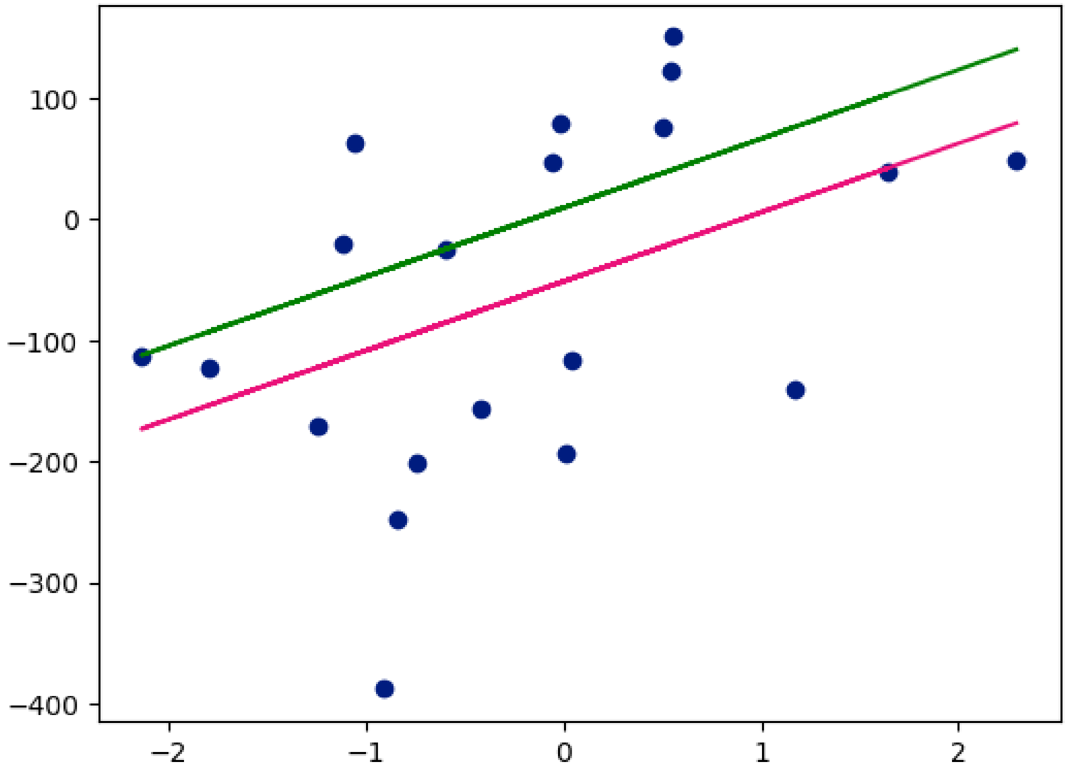



```
learning_rate = 0.01

# Changing parameter value in First iteration    --   Iteration-1
# Calculating slope
slope = -2 * np.sum(y - 57.03 * np.ravel(x) - b)

step_size = slope*learning_rate

# Updating value of b
b = b - step_size
print(b)
```

```
9.445125640447138
```

```
plt.scatter(x,y)
plt.plot(x,lr.predict(x),color="#EA1179")
y_pred1 = m*x + b
plt.plot(x,y_pred1,color='green')
```

```
[<matplotlib.lines.Line2D at 0x79043a8b6f20>]
```

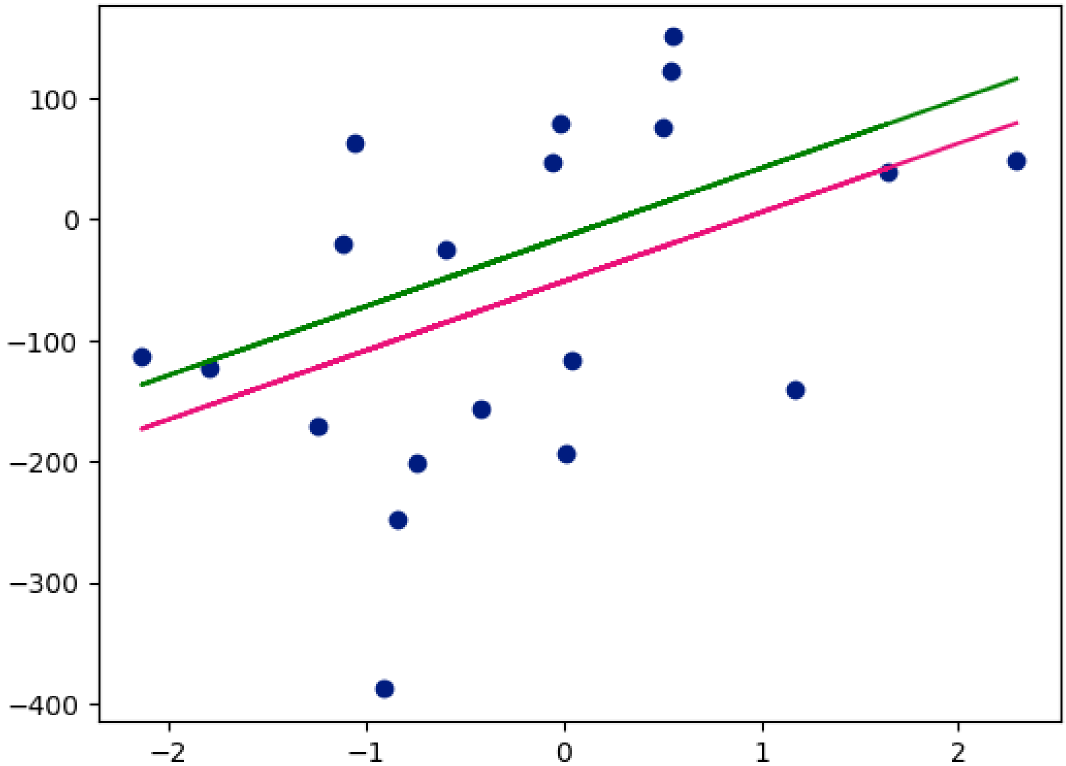



```
# Changing parameter value in Second iteration    --   Iteration-2
# Calculating slope
slope = -2 * np.sum(y - 57.03 * np.ravel(x) - b)

step_size = slope*learning_rate

# Updating value of b
b = b - step_size
print(b)
```

```
-14.887798975284575
```

```
plt.scatter(x,y)
plt.plot(x,lr.predict(x),color="#EA1179")
y_pred2 = m*x + b
plt.plot(x,y_pred2,color='green')
```

```
[<matplotlib.lines.Line2D at 0x79043a7b3b80>]
```

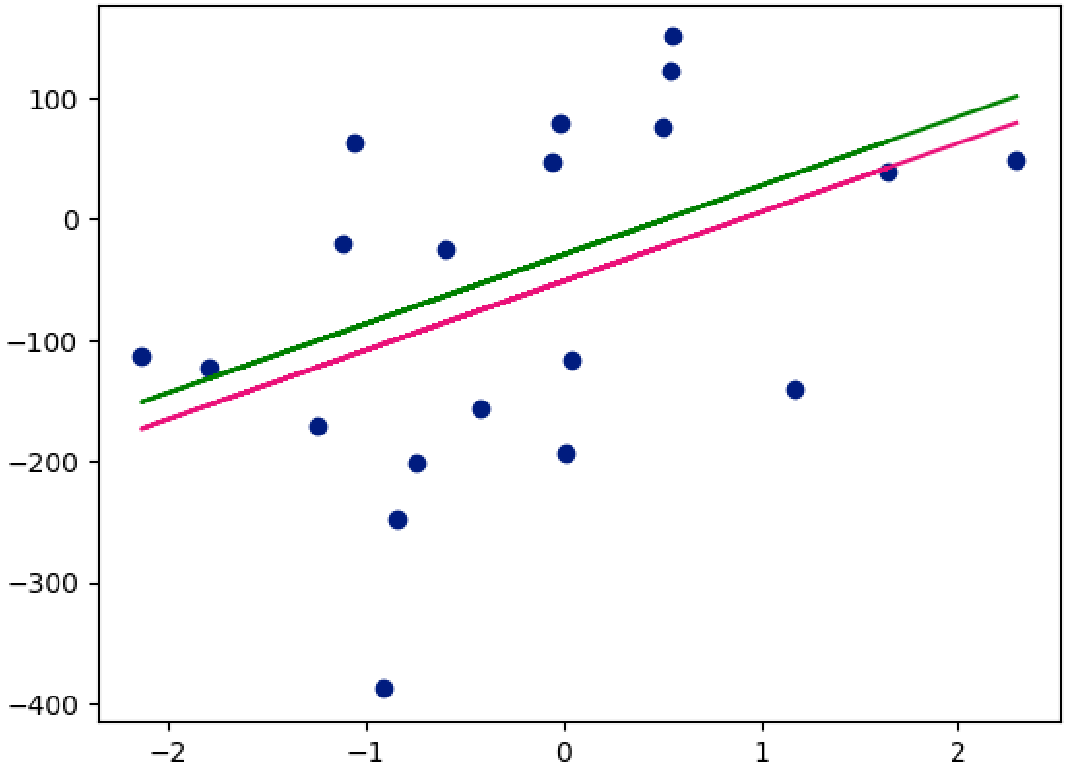



```
# Changing parameter value in Third iteration    --   Iteration-3
# Calculating slope
slope = -2 * np.sum(y - 57.03 * np.ravel(x) - b)

step_size = slope*learning_rate

# Updating value of b
b = b - step_size
print(b)
```

```
-29.487553744723606
```

```
plt.scatter(x,y)
plt.plot(x,lr.predict(x),color="#EA1179")
y_pred3 = m*x + b
plt.plot(x,y_pred3,color='green')
```

```
[<matplotlib.lines.Line2D at 0x79043a554d00>]
```

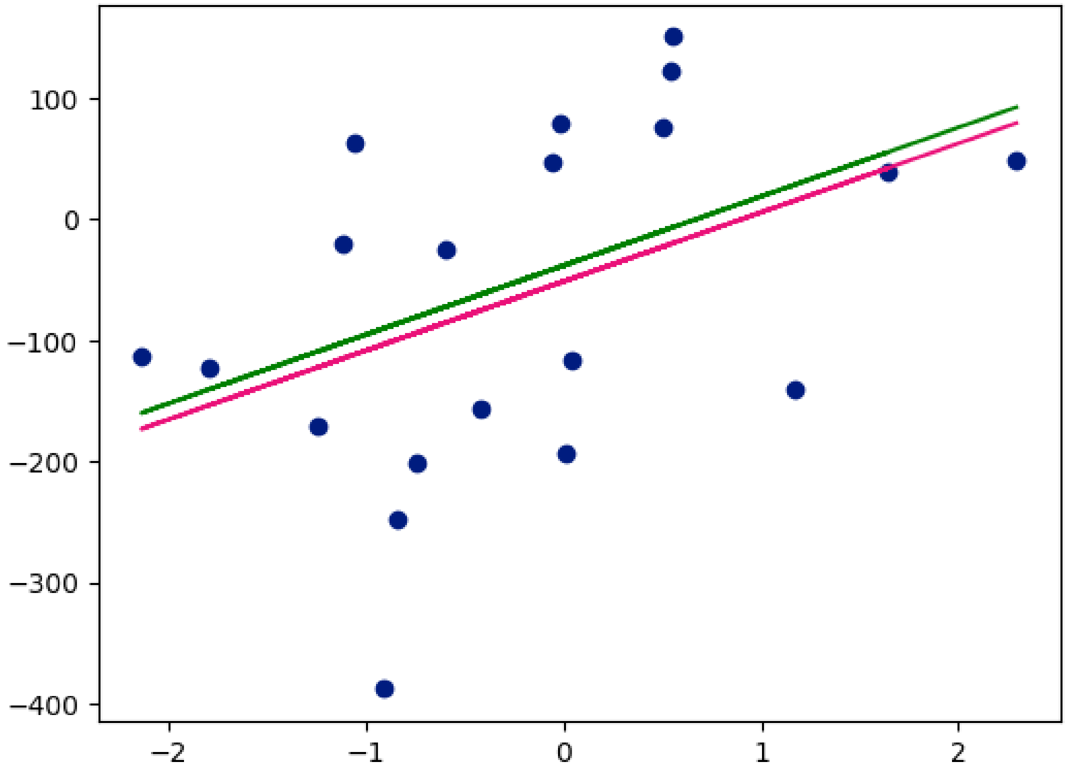



```
# Changing parameter value in Fourth iteration    --   Iteration-4
# Calculating slope
slope = -2 * np.sum(y - 57.03 * np.ravel(x) - b)

step_size = slope*learning_rate

# Updating value of b
b = b - step_size
print(b)
```

```
-38.24740660638702
```

```
plt.scatter(x,y)
plt.plot(x,lr.predict(x),color="#EA1179")
y_pred4 = m*x + b
plt.plot(x,y_pred4,color='green')
```

```
[<matplotlib.lines.Line2D at 0x79043a5a54e0>]
```

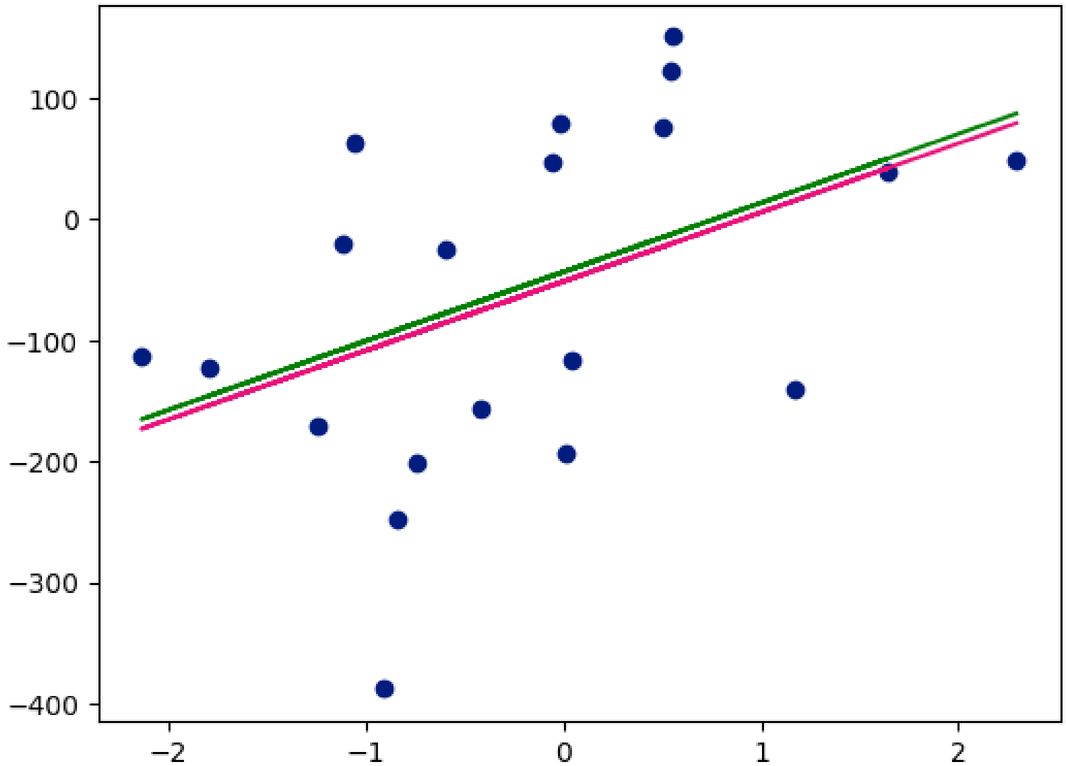



```
# Changing parameter value in Fifth iteration    --   Iteration-5
# Calculating slope
slope = -2 * np.sum(y - 57.03 * np.ravel(x) - b)

step_size = slope*learning_rate

# Updating value of b
b = b - step_size
print(b)
```

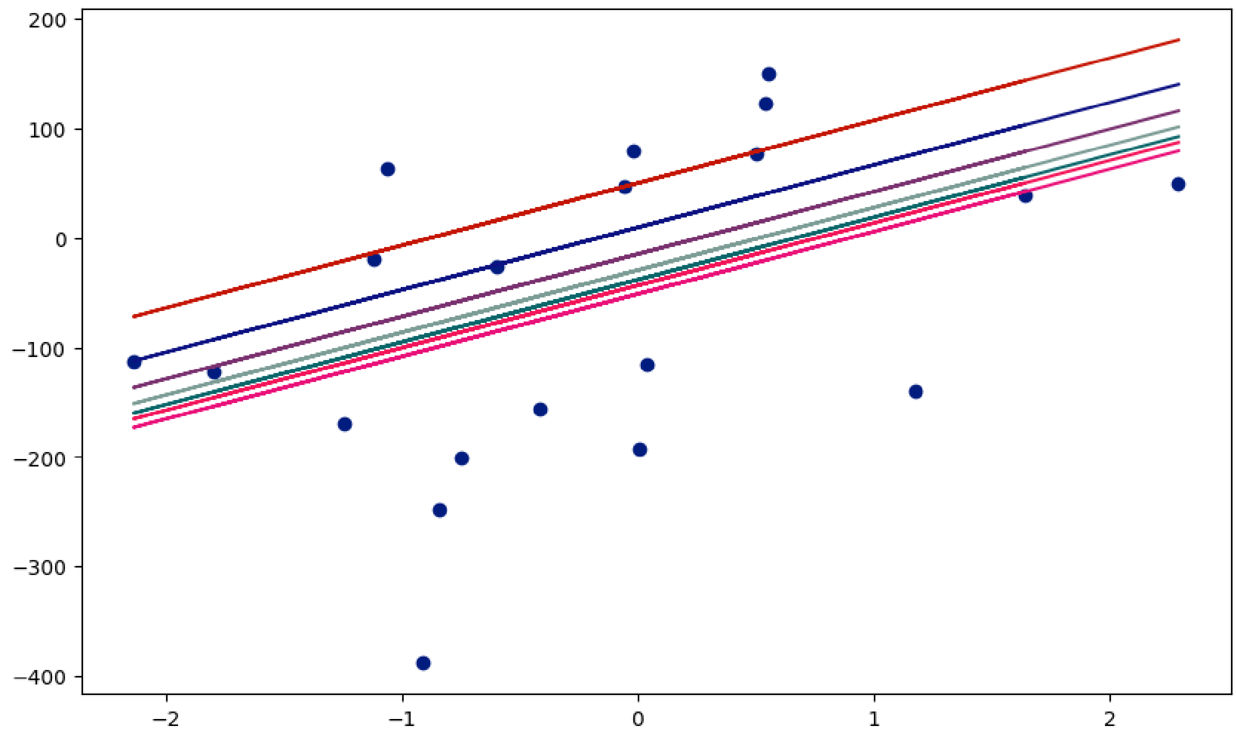
```
-43.503318323385066
```

```
plt.scatter(x,y)
plt.plot(x,lr.predict(x),color="#EA1179")
y_pred5 = m*x + b
plt.plot(x,y_pred5,color='green')
```

```
[<matplotlib.lines.Line2D at 0x79043a63e6b0>]
```



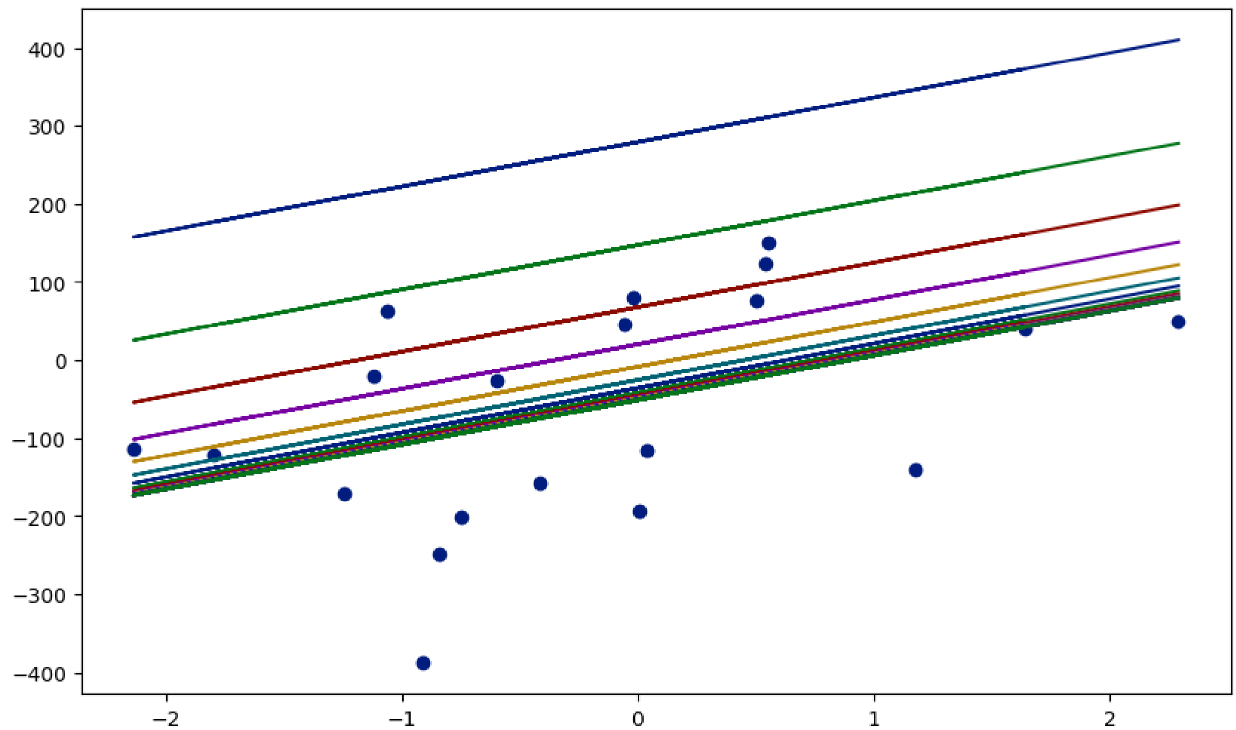

```
plt.figure(figsize=(10,6))
plt.scatter(x,y)
plt.plot(x,lr.predict(x),color="#EA1179")
y_pred5 = m*x + b
plt.plot(x,y_pred,color='#C51605')
plt.plot(x,y_pred1,color='#0D1282')
plt.plot(x,y_pred2,color='#7A316F')
plt.plot(x,y_pred3,color='#7C9D96')
plt.plot(x,y_pred4,color='#0B666A')
plt.plot(x,y_pred5,color='#F31559')

[<matplotlib.lines.Line2D at 0x790438ba35b0>]
```




```
# From the above iterations we saw that how by applying gradient descent we are approaching tolocal minima as per iteration
#we kept on going updating value of b and in each iteration we came close to actual b parameter so our error decreases

# Now we will run through epochs to see how parameters change and reach minima

b = 500
m = 57.04
lr = 0.01

epochs = int(input("No. of epochs : "))
plt.figure(figsize=(10,6))
for i in range(epochs):

   slope = -2 * np.sum(y - 57.03 * np.ravel(x) - b)
   step_size = slope*learning_rate
   b = b - step_size

   y_pred = m*x + b

   plt.plot(x,y_pred)
plt.scatter(x,y)
```

```
No. of epochs : 50
```

```
<matplotlib.collections.PathCollection at 0x790438919720>
```




```
# In gradient descent, the parameters "m" and "b" (slope and
intercept) of the linear regression model change simultaneously with
each iteration.
#The algorithm iteratively updates the values of "m" and "b" based on
the gradient of the loss function with respect to these parameters.
# As the gradient descent progresses, "m" and "b" are adjusted in the
direction that minimizes the mean squared error between the predicted
values and the actual target values.
# This simultaneous update process continues until the algorithm
converges to the optimal values of "m" and "b,"
# resulting in the best-fitting line that accurately represents the
underlying relationship between the input features and the target
variable.

class GDRegressor:

    def __init__(self,learning_rate,epochs):
        self.m = 100
        self.b = -120
        self.lr = learning_rate
        self.epochs = epochs

    def fit(self,x,y):
        # calcualte 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.m = self.m - (self.lr * loss_slope_m)
        print(self.m,self.b)

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

lgd = GDRegressor(0.01,100)
lgd.fit(x,y)

57.030695858354925 -51.38704024529876

# see here we got m as 57.03 same as we got in our model using sklearn
and the b = -51.38

lgd.predict(x)

array([[  79.33917778],
       [ -85.38644277],
       [ -75.15503029],
       [-115.1431063 ],
       [-122.40668637],
       [ -19.93723235],
       [ -94.0386409 ],
       [ -50.8723239 ],
       [ -52.47806577],
       [ -49.01801976],
       [ -20.64416911],
       [-153.66791965],
       [ -54.59597656],
       [-173.21579007],
       [-111.72279147],
       [  42.15874535],
       [  15.62409694],
       [ -22.70736309],
       [ -99.39247825],
       [-103.22837707]])
```