

## ✓ Activity 1

```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

## ✓ Activity 2

```
df = pd.read_csv('50_Startups.csv')
df.head()
```



	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94



Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

```
X = df.iloc[:, [0, 1, 2]].values
y = df.iloc[:, -1:].values
```

```
print(X.shape)
print(y.shape)
```



```
(50, 3)
(50, 1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```



```
(40, 3) (10, 3)
(40, 1) (10, 1)
```

```
X_max = X_train.max(axis=0, keepdims=True)
X_min = X_train.min(axis=0, keepdims=True)
print(X_max.shape, X_min.shape)
print(X_max)
print(X_min)
```

```
↔ (1, 3) (1, 3)
[[165349.2  182645.56 471784.1 ]]
[[      0.   51283.14      0.  ]]
```

```
X_train_scaled = (X_train - X_min) / X_max
X_test_scaled = (X_test - X_min) / X_max

print(X_train_scaled.min(), X_train_scaled.max())
print(X_test_scaled.min(), X_test_scaled.max())
```

```
↔ 0.0 1.0
   0.0 0.5989483537067061
```

```
y_mean = y_train.mean(axis=0, keepdims=True)
y_std = y_train.std(axis=0, keepdims=True)

y_train_scaled = (y_train - y_mean) / y_std
y_test_scaled = (y_test - y_mean) / y_std

print(y_train_scaled.max(), y_train_scaled.min())
print(y_test_scaled.max(), y_test_scaled.min())
```

```
↔ 1.8464513788878065 -2.433579441809424
   0.449636647684142 -1.9276315184465596
```

## ✓ Activity 3

```
def predict(X, w):
    y_pred = np.dot(X, w.T)
    return y_pred
```

```
def loss(X, y, w):
    y_pred = predict(X, w)
    loss = np.mean((y_pred - y)**2)
    return loss
```

```
def grad(X, y, w):
    y_pred = predict(X, w)
    delta = y_pred - y
    dw = np.dot(X.T, delta)
    return dw.T
```

```
w = np.zeros((1, 3))
lr = 0.1
epochs = 100
```

```
for i in range(epochs):
    l = loss(X_train_scaled, y_train_scaled, w)
    dw = grad(X_train_scaled, y_train_scaled, w)
    w -= lr * dw
    print(l, dw.ravel())
```

```
9.843947890040512e+21 [-2.13466656e+12 -1.45725099e+12 -2.15216864e+12]
3.452959401618463e+22 [3.99798472e+12 2.72926334e+12 4.03076410e+12]
1.2111937976925097e+23 [-7.48776513e+12 -5.11159603e+12 -7.54915712e+12]
4.248501777580128e+23 [1.40237221e+13 9.57343090e+12 1.41387022e+13]
1.4902460191332526e+24 [-2.62648170e+13 -1.79299340e+13 -2.64801616e+13]
5.227332631145692e+24 [4.91909786e+13 3.35807022e+13 4.95942942e+13]
1.833590298904683e+25 [-9.21290399e+13 -6.28927893e+13 -9.28844035e+13]
6.431680594048002e+25 [1.72547086e+14 1.17790954e+14 1.73961795e+14]
2.256039164723136e+26 [-3.23160827e+14 -2.20608897e+14 -3.25810415e+14]
7.913503536656938e+26 [6.05243027e+14 4.13175066e+14 6.10205399e+14]
2.7758178671675273e+27 [-1.13355051e+15 -7.73829333e+14 -1.14284446e+15]
9.736730129701227e+27 [2.12300961e+15 1.44929326e+15 2.14041611e+15]
3.415350651783596e+28 [-3.97615260e+15 -2.71435942e+15 -4.00875297e+15]
1.198001784917147e+29 [7.44687607e+15 5.08368272e+15 7.50793282e+15]
4.2022281838474256e+29 [-1.39471416e+16 -9.52115249e+15 -1.40614938e+16]
```

## ✓ Ex

## ✓ Cau 1:

```
y_pred = predict(X_test, w)

rmse = math.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("RMSE:", rmse)
print("MAE:", mae)
print("R2 Score:", r2)
```

```
➡ RMSE: 1.3596991262360576e+32
MAE: 1.2078222719162589e+32
R2 Score: -2.2830326128627455e+55
```

## ✓ Cau 2:

```
class LinearRegression:
    def __init__(self, learning_rate=0.1, epochs=100):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None

    def predict(self, X):
        y_pred = np.dot(X, self.weights.T)
        return y_pred

    def loss(self, X, y):
        y_pred = self.predict(X)
        loss = np.mean((y_pred - y)**2)
        return loss

    def grad(self, X, y):
        y_pred = self.predict(X)
        delta = y_pred - y
        dw = np.dot(X.T, delta)
        return dw.T

    def fit(self, X, y):
        self.weights = np.zeros((1, X.shape[1]))
        for i in range(self.epochs):
            l = self.loss(X, y)
            dw = self.grad(X, y)
            self.weights -= self.learning_rate * dw

    def evaluate(self, X_test, y_test):
```

```


y_pred = self.predict(X_test)
rmse = math.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("RMSE:", rmse)
print("MAE:", mae)
print("R2 Score:", r2)

```

```

model = LinearRegression(learning_rate=0.1, epochs=100)
model.fit(X_train_scaled, y_train_scaled)
model.evaluate(X_test_scaled, y_test_scaled)

```

 RMSE: 4.560537851411792e+26  
 MAE: 4.058137603814759e+26  
 R2 Score: -4.421351281112377e+53

### ✓ Cau 3:

```

class LinearRegressionWithBias:
    def __init__(self, learning_rate=0.1, epochs=100):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None

    def predict(self, X):
        X_with_bias = np.c_[np.ones(X.shape[0]), X]
        y_pred = np.dot(X_with_bias, self.weights.T)
        return y_pred

    def loss(self, X, y):
        y_pred = self.predict(X)
        loss = np.mean((y_pred - y)**2)
        return loss

    def grad(self, X, y):
        X_with_bias = np.c_[np.ones(X.shape[0]), X]
        y_pred = self.predict(X)
        delta = y_pred - y
        dw = np.dot(X_with_bias.T, delta)
        return dw.T

    def fit(self, X, y):
        X_with_bias = np.c_[np.ones(X.shape[0]), X]
        self.weights = np.zeros((1, X_with_bias.shape[1]))
        for i in range(self.epochs):
            l = self.loss(X, y)
            dw = self.grad(X, y)
            self.weights -= self.learning_rate * dw

    def evaluate(self, X_test, y_test):
        y_pred = self.predict(X_test)
        rmse = math.sqrt(mean_squared_error(y_test, y_pred))
        mae = mean_absolute_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)

```

```
print("RMSE:", rmse)
print("MAE:", mae)
print("R2 Score:", r2)
```

```
model = LinearRegressionWithBias(learning_rate=0.1, epochs=100)
model.fit(X_train_scaled, y_train_scaled)
model.evaluate(X_test_scaled, y_test_scaled)
```

➡ RMSE: 1.337540350839384e+74  
MAE: 1.320878178392196e+74  
R2 Score: -3.8030906517067147e+148