Name: Sudarshan Tharmaraj

Roll no :66

Batch: B3

Expt-3: Linear Regression: Parameter Estimation using OLS, MLE, and Gradient Descent

In [87]:
```
# Step 1: Import Libraries
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [88]:
```
# Step 2:Input Data
z = np.array([1, 2, 3]).reshape(-1,1)  # input
y = np.array([2, 3, 5])    # output
```

In [89]:
```
# Step 3: Create model and fit
model= LinearRegression()
model.fit(z,y)
```

Out[89]:

▾ LinearRegression

Parameters

fit_intercept	True
copy_X	True
tol	1e-06
n_jobs	None
positive	False

In [90]:
```
# Step 4 : Get coefficients (MLE estimates)
w = model.coef_[0]
b = model.intercept_


print(f"w(slope) = {w}")
print(f"b(intercept) = {b}")
```

```
w(slope) = 1.4999999999999993
b(intercept) = 0.3333333333333348
```

In [91]:
```
y_pred=model.predict(z)
```

```
print (z)
```

```
[[1]
 [2]
 [3]]
```

In [92]:
```
print("\nPrediction for training data:")
for zi, yi, ypi in zip(z.flatten(), y, y_pred):
    print(f"x={zi}, Actual sales ={yi},Predicted Sales = {ypi:2f}")
```

```
Prediction for training data:
x=1, Actual sales =2,Predicted Sales = 1.833333
x=2, Actual sales =3,Predicted Sales = 3.333333
x=3, Actual sales =5,Predicted Sales = 4.833333
```

In [93]:
```
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print(f"\nMean Squared Error (MSE) = {mse:.4f}")
print(f"R^2 Score = {r2:.4f}")
```

```
Mean Squared Error (MSE) = 0.0556
R^2 Score = 0.9643
```

MLE

In [94]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from scipy.optimize import minimize
```

In [95]:
```
# Step 2: Input data
X = np.array([1,2,3])
y = np.array([2,3,5])
```

In [96]:
```
# Step 3: Negative Log Likelihood function
def neg_log_likelihood(parms):
    w, b = parms
    sigma2 = 1  #assume varience = 1
    y_pred = w*X + b
    nll = 0.5*np.sum((y-y_pred)**2 / sigma2)
    return nll
```

In [97]:
```
#Initial values for w (weight) and b (bias)
```

In [98]:
```
# Step 4: Minimize nll
result = minimize(neg_log_likelihood, initial_guess)
w_mle, b_mle = result.x
print(f"Slope is: {w_mle}\nIntercept is: {b_mle}")
```

```
Slope is: 1.500000003897125
Intercept is: 0.3333333918730798
```

In [99]:
```
# Step 5: Prediction
y_pred = w_mle*X + b_mle
print("Predictions for training data\n")
for xi, yi, ypi in zip(X.flatten(), y, y_pred):
    print(f"X = {xi}, Actual y = {yi}, Predicted y = {ypi}")
```

```
Predictions for training data

X = 1, Actual y = 2, Predicted y = 1.833333395770205
X = 2, Actual y = 3, Predicted y = 3.33333339966733
X = 3, Actual y = 5, Predicted y = 4.833333403564454
```

In [100…
```
# Step 6: Error Calculation
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f"Mean Squared Error is: {mse}")
print(f"R2_Score is: {r2}")
```

```
Mean Squared Error is: 0.05555555555556004
R2_Score is: 0.9642857142857114
```

Gradient Descent

In [101…
```
# Step 1: Imports
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
```

In [102…
```
# Step 2: Input data
X = np.array([1,2,3])
y = np.array([2,3,5])
```

In [103…
```
# Step 3:
w, b = 0, 0
alpha = 0.001
n_iter = 100000
n = len(X)

for i in range(n_iter):
    y_pred = w*X.flatten() + b
    dw = (-2/n)*np.sum(X.flatten()*(y-y_pred))
    db = (-2/n)*np.sum(y-y_pred)
    w -= alpha*dw
    b -= alpha*db

print(f"Slope is: {w}\nIntercept is: {b}")
```

```
Slope is: 1.49999999999562
Intercept is: 0.33333333334329107
```

In [104…
```
# Step 6 :Make Predictions
# Make predictions using estimated parameters
y_pred = w *X + b
```

```
print("Predictions for training data\n")
for xi, yi, ypi in zip(X.flatten(), y, y_pred):
    print(f"X = {xi}, Actual y = {yi}, Predicted y = {ypi}")
```

```
Predictions for training data

X = 1, Actual y = 2, Predicted y = 1.833333333338911
X = 2, Actual y = 3, Predicted y = 3.333333333334531
X = 3, Actual y = 5, Predicted y = 4.833333333330152
```

In [105…

```
# Step 7: Calculate metrics
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f"Mean Squared Error is: {mse}")
print(f"R2_Score is: {r2}")
```

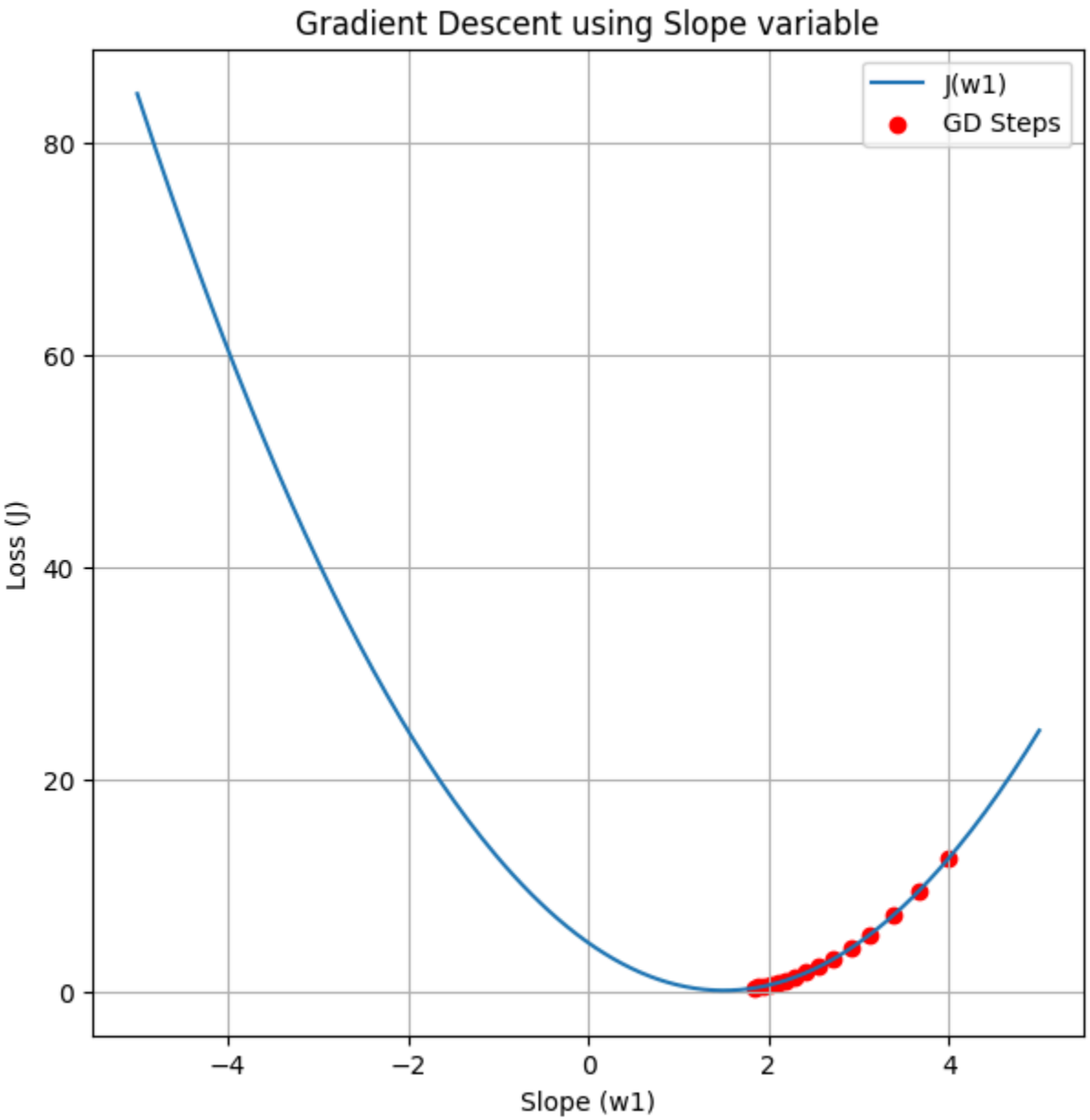
```
Mean Squared Error is: 0.05555555555555553
R2_Score is: 0.9642857142857143
```

Gradient Descent with single Parameter

In [106…

```
# Step 2: Data
z = np.array([1, 2, 3])
y = np.array([2, 3, 5])
n = len(X)

# Step 3: Calculate loss function
def loss(w1):
    w0 = np.mean(y) - w1*np.mean(X)
    y_pred = w1*X + w0
    return np.sum((y-y_pred)**2)

 # Step 4: Calculate Gradient of J wrt w1
def gradient(w1):
    w0 = np.mean(y) - w1*np.mean(X)
    y_pred = w1*X + w0
    return (-2/n)*np.sum(X*(y-y_pred))


# Step 5: Gradient Descent Algorithm
lr = 0.1
w1 = 4
n_iter = 15
w1_values = []
loss_values = []

for i in range(n_iter):
    w1_values.append(w1)
    loss_values.append(loss(w1))
    grad = gradient(w1)
    w1 -= lr*grad

# Step 6: Plot Loss function and GD
```

```
w_space = np.linspace(-5, 5, 200)
loss_space = [loss(w) for w in w_space]

# Plot
plt.figure(figsize=(7, 7))
plt.plot(w_space, loss_space, label="J(w1)")
plt.scatter(w1_values, loss_values, color="red", label="GD Steps")
plt.xlabel("Slope (w1)")
plt.ylabel("Loss (J)")
plt.title("Gradient Descent using Slope variable")
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:

In [ ]: