

```
#NAME - GAURAV RAMESH GUPTA
```

```
#ROLL NO. - 27
```

```
#BATCH - 2
```

```
# Step 1: Imports
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from scipy.optimize import minimize

# Step 2: Input Data
x = np.array([1, 2, 3]).reshape((-1, 1))
y = np.array([2, 3, 5])

# Step 3: Create model and fit data
model = LinearRegression()
model.fit(x, y)

LinearRegression()

# Step 4: get coefficients and print
weight = model.coef_[0]
bias = model.intercept_
print(f"Weight: {weight}, Bias: {bias}")

Weight: 1.4999999999999993, Bias: 0.333333333333348

# Step 5: Make predictions
y_pred = model.predict(x)
print("Predictions for training data")
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 = 1.833333333333341
x = 2, Actual y = 3, Predicted y = 3.333333333333335
x = 3, Actual y = 5, Predicted y = 4.833333333333333

# Lab recitation question using OLS method
# input data
x = np.array([10, 20, 30, 50]).reshape((-1, 1))
y = np.array([12, 21, 29, 48])

# create model and fit data
model = LinearRegression()
model.fit(x, y)

# get coefficients and print
weight = model.coef_[0]
```

```

bias = model.intercept_
print(f"Weight: {weight}, Bias: {bias}")

# Make predictions
y_pred = model.predict(x)
print("\nPredictions for training data")
for xi, yi, ypi in zip(x.flatten(), y, y_pred):
    print(f"x = {xi}, Actual y = {yi}, Predicted y = {ypi}")

Weight: 0.8971428571428574, Bias: 2.828571428571422

Predictions for training data
x = 10, Actual y = 12, Predicted y = 11.799999999999995
x = 20, Actual y = 21, Predicted y = 20.77142857142857
x = 30, Actual y = 29, Predicted y = 29.742857142857144
x = 50, Actual y = 48, Predicted y = 47.68571428571429

```

#MLE

```

# Step 1: Imports
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 2: Input Data
x = np.array([1, 2, 3])
y = np.array([2, 3, 5])

# Step 3: Negative Log Likelihood
def negative_log_likelihood(params):
    weight, bias = params
    sigma_squared = 1.0 # Assume a fixed variance
    y_pred = weight*x + bias
    nll = 0.5*np.sum(((y-y_pred)**2)) / sigma_squared
    return nll

# initial values for w and b
initial_guess = [0, 0]

# Step 4: Minimize NLL to find optimal parameters
result = minimize(negative_log_likelihood, initial_guess)
w_mle, b_mle = result.x
print(f"Slope is: {w_mle}, intercept: {b_mle}")

Slope is: 1.50000003897125, intercept: 0.333333918730798

# Step 5: prediction
y_pred = w_mle*x + b_mle
print("Predictions for training data")
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

# Step 6: Error Calculation
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

Mean Squared Error: 0.05555555555556004
R^2 Score: 0.9642857142857114

# Recitation Question
# input data
x = np.array([10, 20, 30, 50])
y = np.array([12, 21, 29, 48])

# Negative Log Likelihood
def negative_log_likelihood(params):
    weight, bias = params
    sigma_squared = 1.0 # Assume a fixed variance
    y_pred = weight*x + bias
    nll = 0.5*np.sum(((y-y_pred)**2)) / sigma_squared
    return nll

# initial values for w and b
initial_guess = [0, 0]

# Minimize NLL to find optimal parameters
result = minimize(negative_log_likelihood, initial_guess)
w_mle, b_mle = result.x
print(f"Slope is: {w_mle}, intercept: {b_mle}")

# prediction
y_pred = w_mle*x + b_mle
print("\nPredictions for training data")
for xi, yi, ypi in zip(x.flatten(), y, y_pred):
    print(f"x = {xi}, Actual y = {yi}, Predicted y = {ypi}")

# Error Calculation
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f"\nMean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

Slope is: 0.8971428246616441, intercept: 2.828572315886469

Predictions for training data
x = 10, Actual y = 12, Predicted y = 11.800000562502909

```

```
x = 20, Actual y = 21, Predicted y = 20.77142880911935
x = 30, Actual y = 29, Predicted y = 29.74285705573579
x = 50, Actual y = 48, Predicted y = 47.68571354896867
```

```
Mean Squared Error: 0.18571428571451654
R^2 Score: 0.998946301925024
```

#LR Using Gradient Descent (GD) Optimization ALgorithm

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

# Step 2: Input Data
x = np.array([1, 2, 3])
y = np.array([2, 3, 5])

# step 3
weight, bias = 0, 0
alpha = 0.001
n_iter = 10000
n = len(x)

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

print(f"Slope is: {weight}, intercept: {bias}")

Slope is: 1.4891282998341364, intercept: 0.35804726319993174
```

#LR Using Gradient Descent (GD) Optimization Algorithm with Single Parameter

```
import numpy as np
import matplotlib.pyplot as plt

# Step 2: Input Data
x = 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
    grad = (-2/n) * np.sum((y - y_pred) * (x - np.mean(x)))
    return grad

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

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

# Step 6: Plotting Loss function and Gradient Descent
w_space = np.linspace(-2, 4, 200)
loss_space = [loss(w) for w in w_space]

plt.figure(figsize=(10, 6))
plt.plot(w_space, loss_space, label='Loss Function J(w1)')
plt.scatter(w1_values, loss_values, color='red', label='Gradient
Descent Steps')
plt.xlabel('w1 (slope)')
plt.ylabel('Loss J(w1)')
plt.title('Loss Function and Gradient Descent Steps')
plt.legend()
plt.grid(True)
plt.show()

```

Loss Function and Gradient Descent Steps

