```
import torch
import torch.nn as nn  # Neural network module
import numpy as np
import matplotlib.pyplot as plt  # For plotting
%matplotlib inline
```

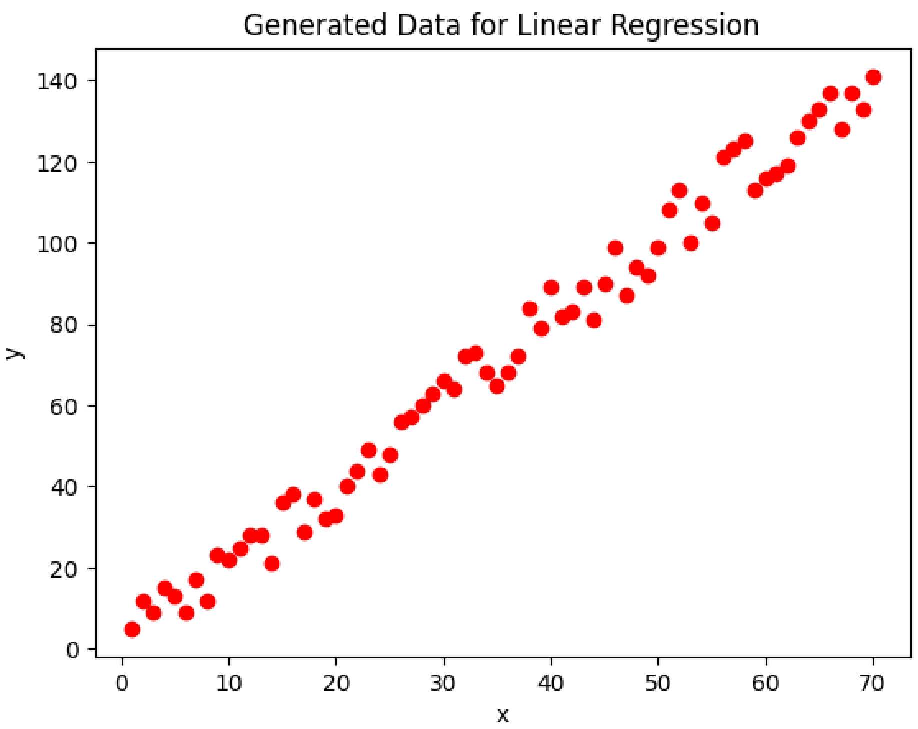
```
X = torch.linspace(1,70,70).reshape(-1,1)
```

```
torch.manual_seed(71) # to obtain reproducible results
e = torch.randint(-8,9,(70,1),dtype=torch.float)
#print(e.sum())
```

```
y = 2*X + 1 + e
print(y.shape)
```

```
torch.Size([70, 1])
```

```
plt.scatter(X.numpy(), y.numpy(),color='red')  # Scatter plot of data points
plt.xlabel('x')
plt.ylabel('y')
plt.title('Generated Data for Linear Regression')
plt.show()
```




```
torch.manual_seed(59)

# Defining the model class
class Model(nn.Module):
    def __init__(self, in_features, out_features):
        super().__init__()
        self.linear = nn.Linear(in_features, out_features)

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred
```

```
torch.manual_seed(59)
model = Model(1, 1)
print('Weight:', model.linear.weight.item())
print('Bias:  ', model.linear.bias.item())
```

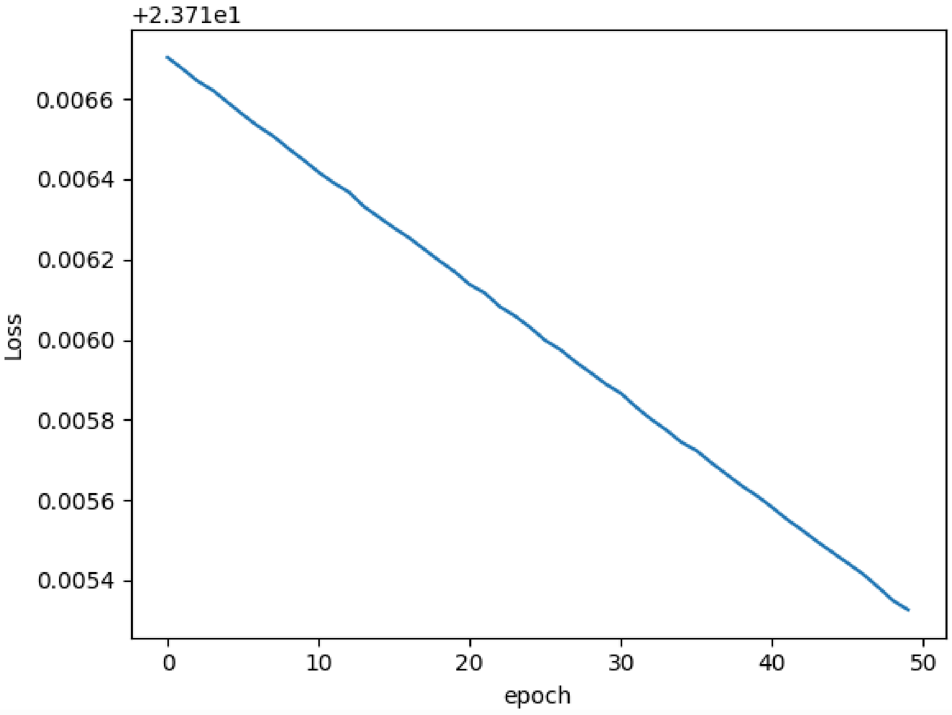
```
Weight: 0.10597813129425049
Bias:   0.9637961387634277
```

```
loss_function = nn.MSELoss()  # Mean Squared Error (MSE) loss

optimizer = torch.optim.SGD(model.parameters(), lr=0.0001)
```

```
epochs = 50  # Number of training iterations
losses = []  # List to store loss values

for epoch in range(1, epochs + 1):  # Start from 1 to 50
    optimizer.zero_grad()  # Clear previous gradients
    y_pred = model(X)  # Forward pass
    loss = loss_function(y_pred, y)  # Compute loss
    losses.append(loss.item())  # Store loss value

    loss.backward()  # Compute gradients
    optimizer.step()  # Update weights

    # Print loss, weight, and bias for EVERY epoch (1 to 50)
    print(f'epoch: {epoch:2}  loss: {loss.item():10.8f}  '
          f'weight: {model.linear.weight.item():10.8f}  '
          f'bias: {model.linear.bias.item():10.8f}')
```

```
epoch:  1  loss: 23.71670341  weight: 1.99034202  bias: 1.00660098
epoch:  2  loss: 23.71667480  weight: 1.99034083  bias: 1.00665390
epoch:  3  loss: 23.71664429  weight: 1.99033976  bias: 1.00670683
epoch:  4  loss: 23.71662140  weight: 1.99033856  bias: 1.00675976
epoch:  5  loss: 23.71659088  weight: 1.99033749  bias: 1.00681269
epoch:  6  loss: 23.71656036  weight: 1.99033642  bias: 1.00686562
epoch:  7  loss: 23.71653175  weight: 1.99033523  bias: 1.00691855
epoch:  8  loss: 23.71650696  weight: 1.99033415  bias: 1.00697148
epoch:  9  loss: 23.71647644  weight: 1.99033296  bias: 1.00702441
epoch: 10  loss: 23.71644783  weight: 1.99033189  bias: 1.00707734
epoch: 11  loss: 23.71641731  weight: 1.99033070  bias: 1.00713027
epoch: 12  loss: 23.71639061  weight: 1.99032962  bias: 1.00718319
epoch: 13  loss: 23.71636772  weight: 1.99032843  bias: 1.00723612
epoch: 14  loss: 23.71633148  weight: 1.99032736  bias: 1.00728905
epoch: 15  loss: 23.71630478  weight: 1.99032629  bias: 1.00734198
epoch: 16  loss: 23.71627808  weight: 1.99032509  bias: 1.00739491
epoch: 17  loss: 23.71625328  weight: 1.99032402  bias: 1.00744784
epoch: 18  loss: 23.71622467  weight: 1.99032283  bias: 1.00750077
epoch: 19  loss: 23.71619606  weight: 1.99032176  bias: 1.00755370
epoch: 20  loss: 23.71616936  weight: 1.99032056  bias: 1.00760663
epoch: 21  loss: 23.71613693  weight: 1.99031949  bias: 1.00765955
epoch: 22  loss: 23.71611595  weight: 1.99031830  bias: 1.00771248
epoch: 23  loss: 23.71608162  weight: 1.99031723  bias: 1.00776541
epoch: 24  loss: 23.71605873  weight: 1.99031603  bias: 1.00781834
epoch: 25  loss: 23.71603012  weight: 1.99031496  bias: 1.00787127
epoch: 26  loss: 23.71599770  weight: 1.99031377  bias: 1.00792420
epoch: 27  loss: 23.71597481  weight: 1.99031270  bias: 1.00797713
epoch: 28  loss: 23.71594429  weight: 1.99031150  bias: 1.00803006
epoch: 29  loss: 23.71591759  weight: 1.99031043  bias: 1.00808299
epoch: 30  loss: 23.71588898  weight: 1.99030936  bias: 1.00813591
epoch: 31  loss: 23.71586609  weight: 1.99030828  bias: 1.00818884
epoch: 32  loss: 23.71583176  weight: 1.99030709  bias: 1.00824177
epoch: 33  loss: 23.71580124  weight: 1.99030602  bias: 1.00829470
epoch: 34  loss: 23.71577454  weight: 1.99030483  bias: 1.00834763
epoch: 35  loss: 23.71574402  weight: 1.99030375  bias: 1.00840056
epoch: 36  loss: 23.71572304  weight: 1.99030256  bias: 1.00845349
epoch: 37  loss: 23.71569252  weight: 1.99030149  bias: 1.00850642
epoch: 38  loss: 23.71566391  weight: 1.99030030  bias: 1.00855935
epoch: 39  loss: 23.71563530  weight: 1.99029922  bias: 1.00861228
epoch: 40  loss: 23.71561050  weight: 1.99029803  bias: 1.00866508
epoch: 41  loss: 23.71558189  weight: 1.99029696  bias: 1.00871789
epoch: 42  loss: 23.71555138  weight: 1.99029577  bias: 1.00877070
epoch: 43  loss: 23.71552467  weight: 1.99029458  bias: 1.00882351
epoch: 44  loss: 23.71549606  weight: 1.99029350  bias: 1.00887632
epoch: 45  loss: 23.71546936  weight: 1.99029243  bias: 1.00892913
epoch: 46  loss: 23.71544266  weight: 1.99029136  bias: 1.00898194
epoch: 47  loss: 23.71541595  weight: 1.99029016  bias: 1.00903475
epoch: 48  loss: 23.71538353  weight: 1.99028909  bias: 1.00908756
epoch: 49  loss: 23.71534920  weight: 1.99028790  bias: 1.00914037
epoch: 50  loss: 23.71532631  weight: 1.99028683  bias: 1.00919318
```

```
plt.plot(range(epochs), losses)
plt.ylabel('Loss')
plt.xlabel('epoch');
plt.show()
```

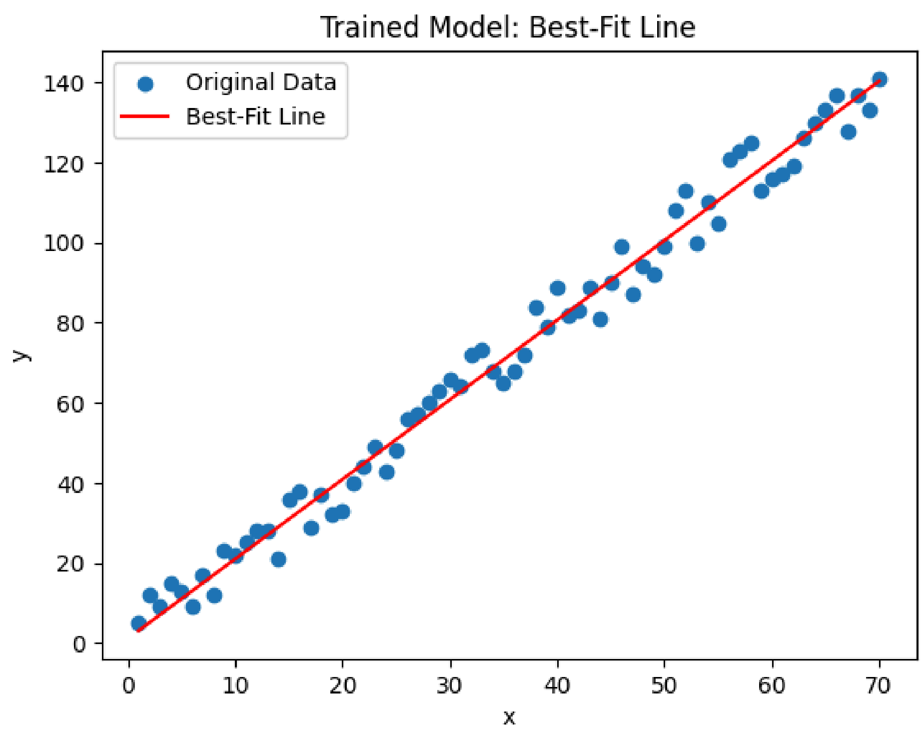



```
# Automatically determine x-range
x1 = torch.tensor([X.min().item(), X.max().item()])

# Extract model parameters
w1, b1 = model.linear.weight.item(), model.linear.bias.item()

# Compute y1 (predicted values)
y1 = x1 * w1 + b1
```

```
# Print weight, bias, and x/y values
print(f'Final Weight: {w1:.8f}, Final Bias: {b1:.8f}')
print(f'X range: {x1.numpy()}')
print(f'Predicted Y values: {y1.numpy()}')
```

```
Final Weight: 1.99028683, Final Bias: 1.00919318
X range: [ 1. 70.]
Predicted Y values: [  2.99948 140.32927]
```

```
# Plot original data and best-fit line
plt.scatter(X.numpy(), y.numpy(), label="Original Data")
plt.plot(x1.numpy(), y1.numpy(), 'r', label="Best-Fit Line")
plt.xlabel('x')
plt.ylabel('y')
plt.title('Trained Model: Best-Fit Line')
plt.legend()
plt.show()
```




```
torch.save(model.state_dict(),'Sana Fathima H regression.pt')
```