30/9/25

Lab-9 (Build a Recurrent Neural Networ

Aim:- To design, implement and evaluate a RNN model for sequential data, such as text, and analyze its performance.

Pseudo Code:-

```
• Load the dataset.
• Preprocess data.
  → Convert sequences into input-output pairs.
• Define RNN model:
  → RNN layer + Dense output layer with activation.
• Compile Model:
  → Select optimizer
• Train Model:
  → Fit data into RNN for given epoch & batch size
  → Monitor validation Loss.
• Evaluate Model:
  → Test data.
• Visualize results:
  → Plot accuracy & loss curves.
• Conclude observation & Results.
```

(Observation)

. The training acuracy increases with epochs, while the loss decreases.

. overfitting can ocur if too many epochs are used without regulagiation (dropout).

. RNN captures seq. dependencies better than feedforward networks.

. LSTM variants perform more effictively on long sequences due to vanishing gradient mitigation

Validation performance depends on dataset complexity & preprocessing quality.

Result) A R.N.N was successfully built and trained on sequential data. "successfully implem

[illegible] 30/9/25

Output:-

Epoch 1:- Loss: 0.6832 - Accuracy: 0.4938

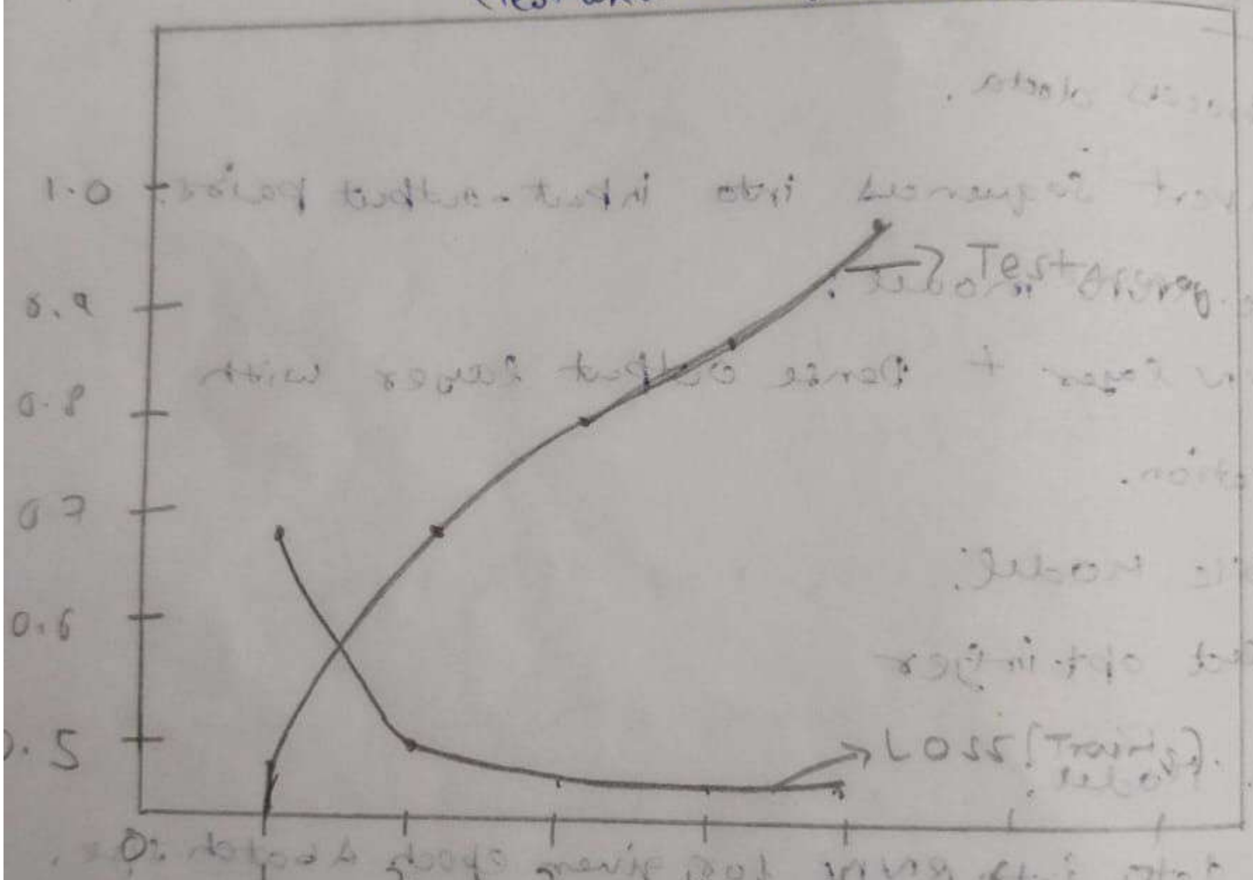
Epoch 2:- Loss: 0.0474 - Accuracy: 0.6832

Epoch 3:- Loss: 0.0019 - Accuracy: 0.7848

Epoch 4:- Loss:- 0.0011 - Accuracy: 0.8492

Epoch 5:- Loss:- 0.0009 - Accuracy:- 0.96.

(Test and Training Accuracy)..



30-9-25 Lab 8:- (Experiment Using LSTM)

Aim:- To build and implement a L.S.T.M (Long short-term-memory) model for seq. prediction.

Pseudo code:-

```
· Import req. libraries.
· Load & preprocess the sequential dataset.
- Normalize the data.
- Create inp-output pairs.
- Reshape X into samples.
Define LSTM model:
- Initialize seq. model
- Add LSTM layer with req. units.
- Add Dense output layer.
Compile the model with optimizer & loss.
Train the model using model.fit().
Evaluate model performance on test dat
· Predict future or test samples.
· Visualize predicted v/s actual output.
```

(Observation)

· The training loss decreases gradually with epoch, indicating that the model is learn the sequence pattern.

- LSTM performs better than simple RNNs when dealing with long-term dependencies,
- The predicted output closely follows the trend of actual data, demonstrating the model's ability to remeber previous context.
- However, training time is higher compared to standard RNN due to more complex computation

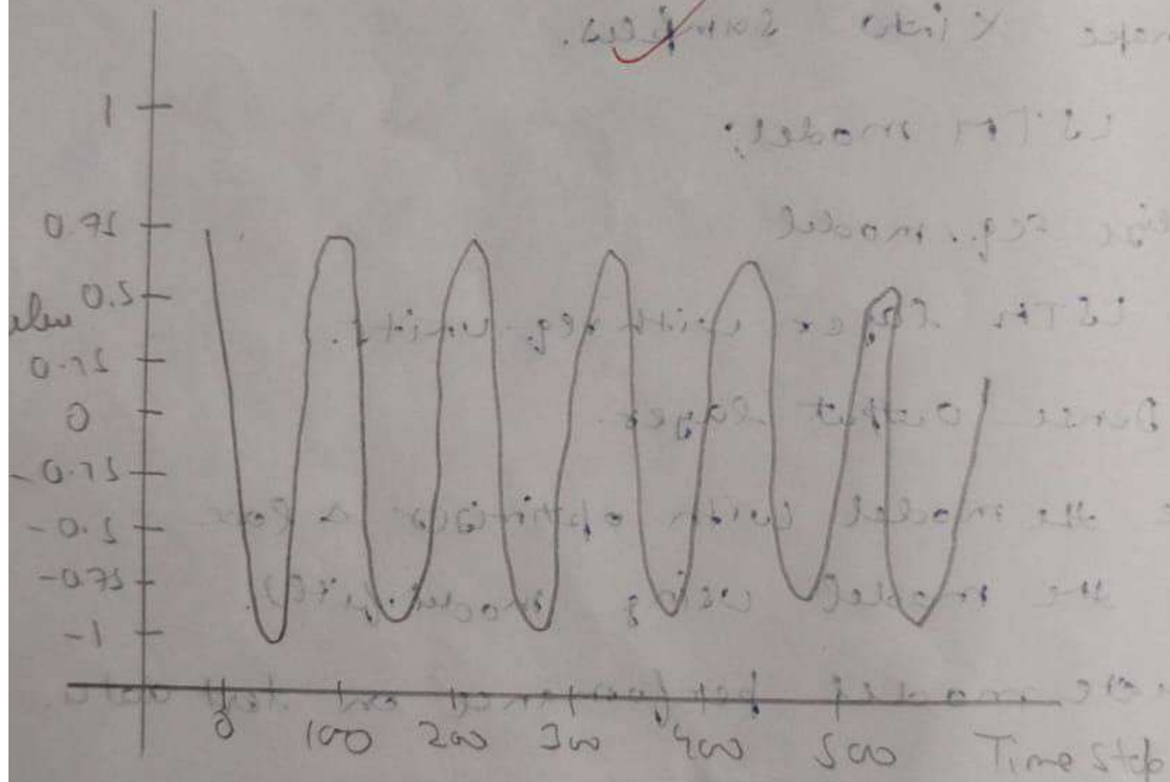
(Result)

"The experiment was succesfully carried out and LSTM model was implemented to lea and predict seq pattern effectively."

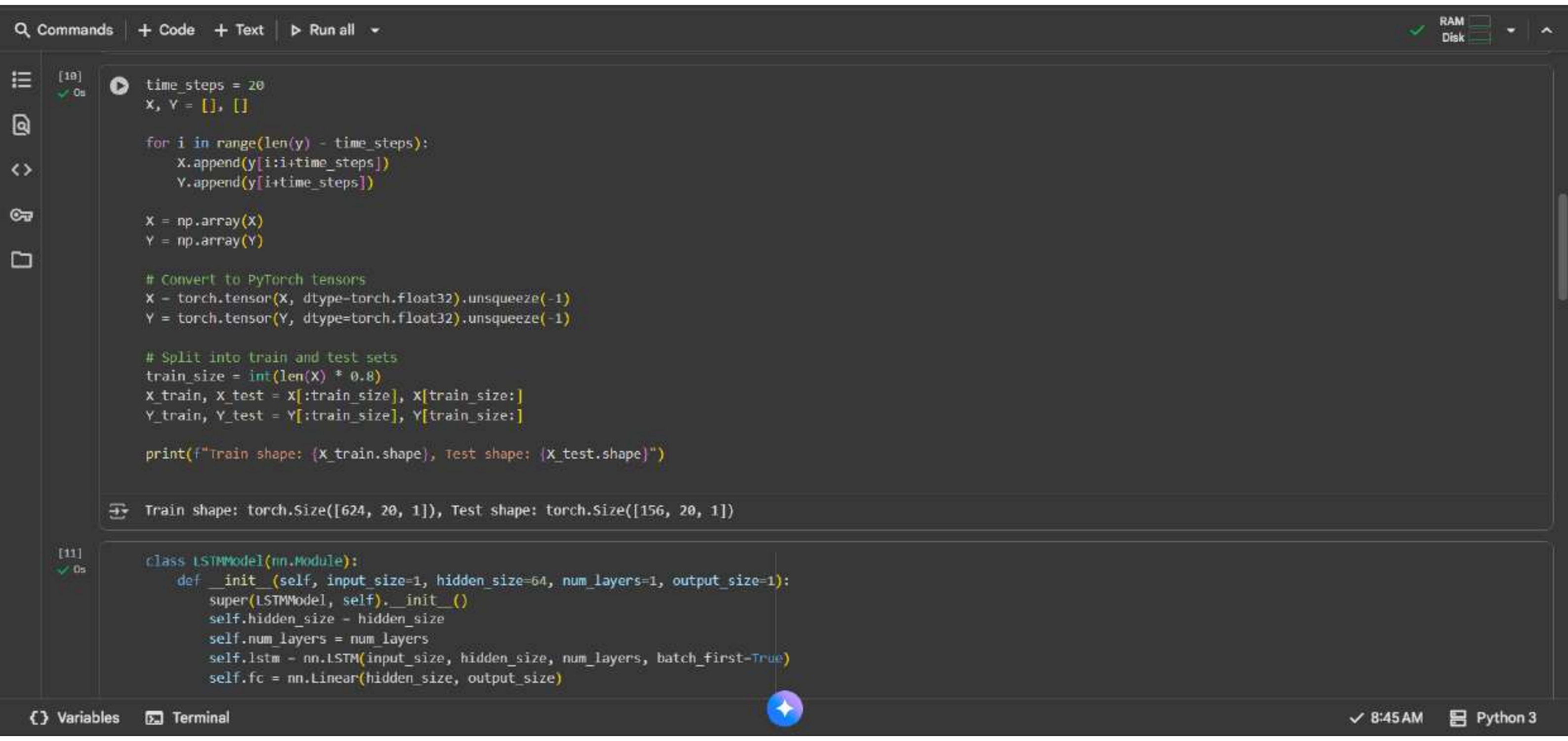
[illegible] 9/10/25

Output:-

```
Epoch [10,100], Loss = 0.032186.
Epoch [20,100], Loss = 0.017898
Epoch [30,100], Loss : 0.018660
Epoch [40,100], Loss = 0.016763.
Epoch [50,100], Loss = 0.015243
Epoch [60,100], Loss = 0.014493
Epoch [70,100], Loss = 0.013756
Epoch [80,100], Loss = 0.016046
Epoch [90,100], Loss = 0.13142
Epoch [100,100], Loss = 0.012629.
```



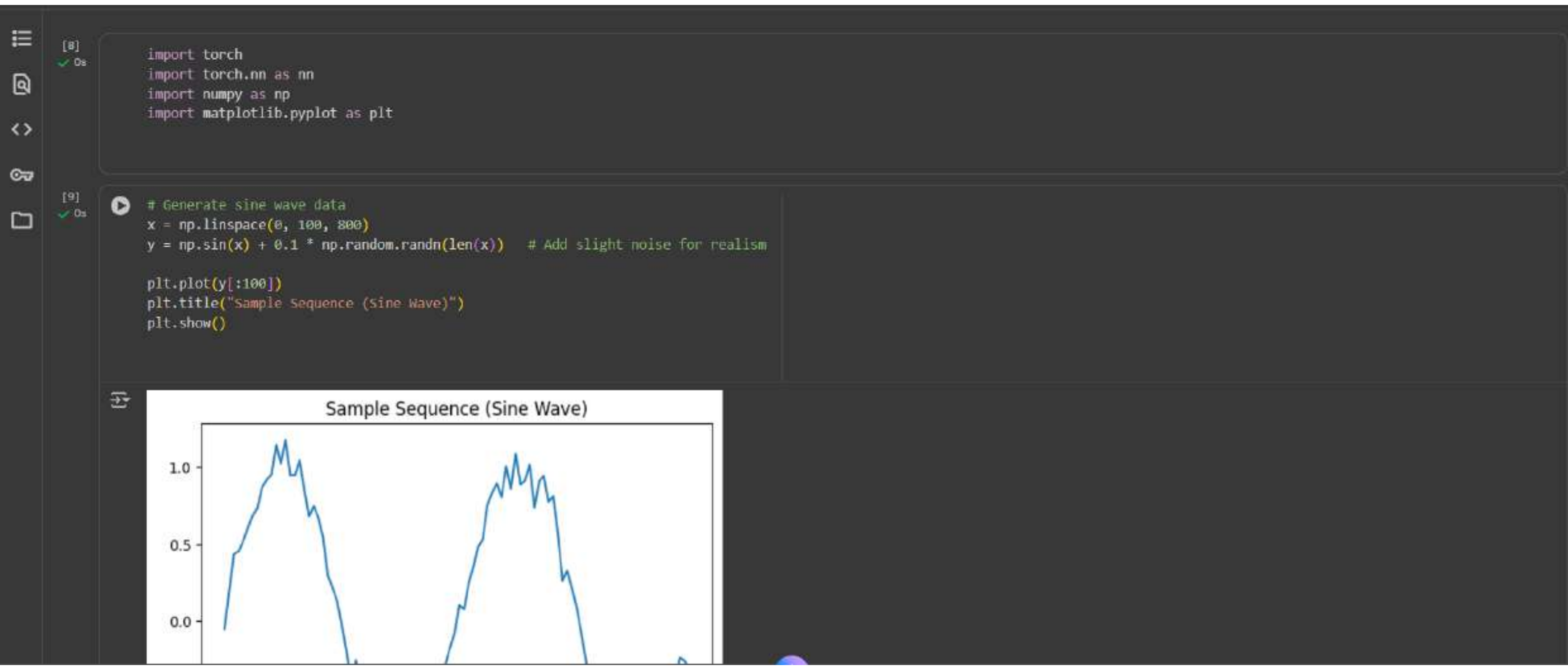
?



```python
time_steps = 20
X, Y = [], []

for i in range(len(y) - time_steps):
    X.append(y[i:i+time_steps])
    Y.append(y[i+time_steps])

X = np.array(X)
Y = np.array(Y)

# Convert to PyTorch tensors
X = torch.tensor(X, dtype=torch.float32).unsqueeze(-1)
Y = torch.tensor(Y, dtype=torch.float32).unsqueeze(-1)

# Split into train and test sets
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]

print(f"Train shape: {X_train.shape}, Test shape: {X_test.shape}")
```

```
Train shape: torch.Size([624, 20, 1]), Test shape: torch.Size([156, 20, 1])
```

```python
class LSTMModel(nn.Module):
    def __init__(self, input_size=1, hidden_size=64, num_layers=1, output_size=1):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
```

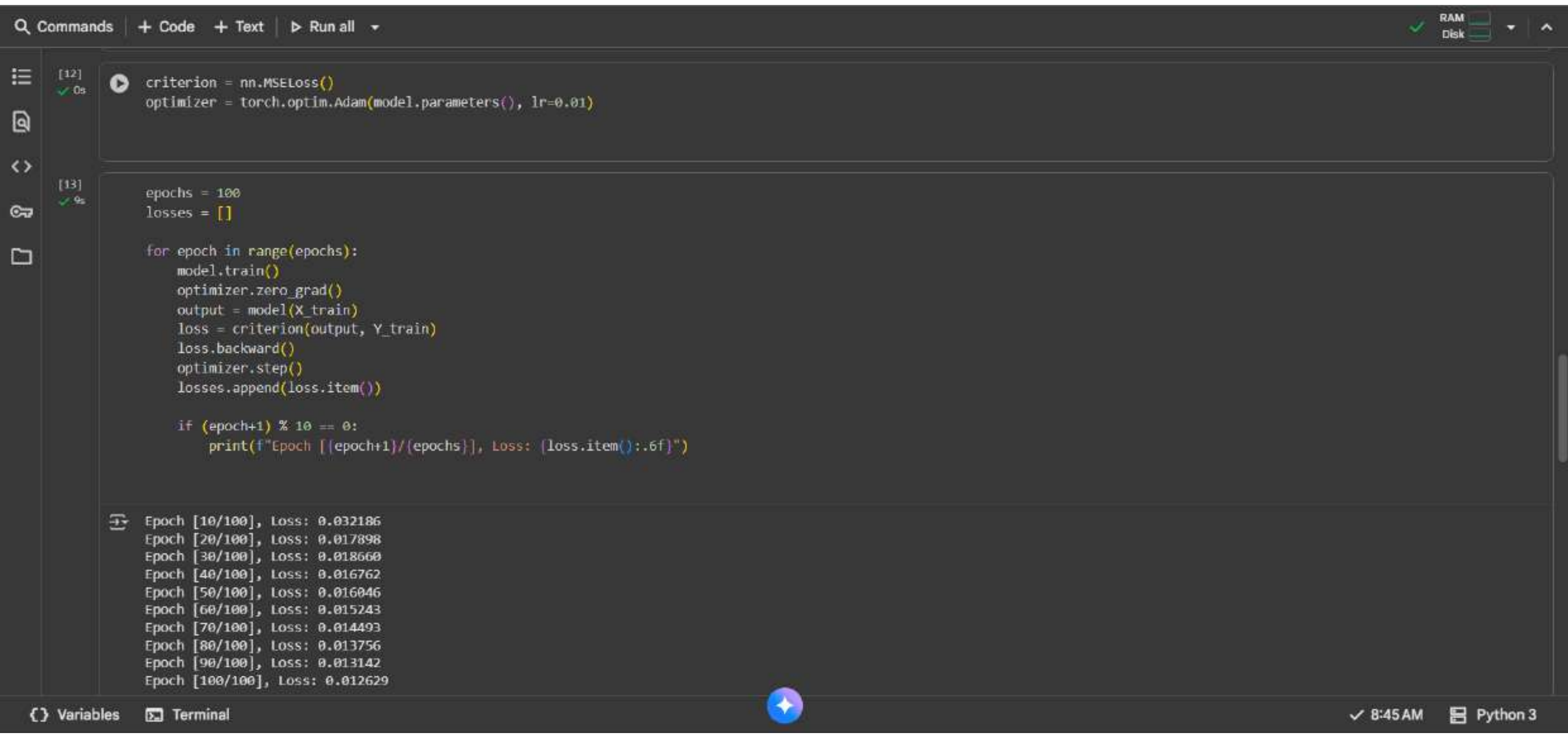


[8]
✓ 0s

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

[9]
✓ 0s

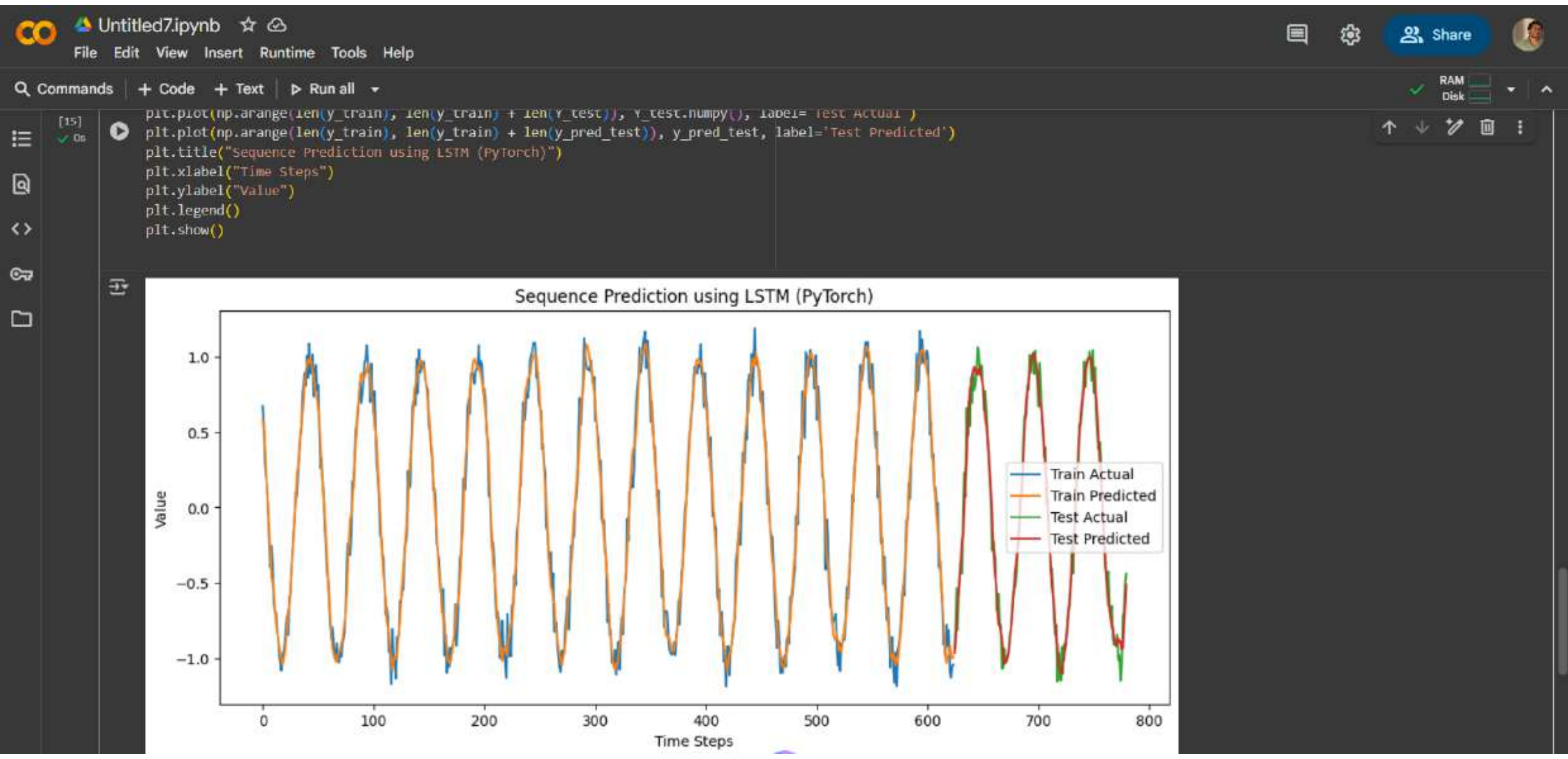
```
# Generate sine wave data
x = np.linspace(0, 100, 800)
y = np.sin(x) + 0.1 * np.random.randn(len(x))   # Add slight noise for realism

plt.plot(y[:100])
plt.title("Sample Sequence (Sine Wave)")
plt.show()
```



```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
epochs = 100
losses = []

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    output = model(X_train)
    loss = criterion(output, Y_train)
    loss.backward()
    optimizer.step()
    losses.append(loss.item())

    if (epoch+1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.6f}")
```

```
Epoch [10/100], Loss: 0.032186
Epoch [20/100], Loss: 0.017898
Epoch [30/100], Loss: 0.018660
Epoch [40/100], Loss: 0.016762
Epoch [50/100], Loss: 0.016046
Epoch [60/100], Loss: 0.015243
Epoch [70/100], Loss: 0.014493
Epoch [80/100], Loss: 0.013756
Epoch [90/100], Loss: 0.013142
Epoch [100/100], Loss: 0.012629
```



```
plt.plot(np.arange(len(y_train), len(y_train) + len(Y_test)), Y_test.numpy(), label='Test Actual')
plt.plot(np.arange(len(y_train), len(y_train) + len(y_pred_test)), y_pred_test, label='Test Predicted')
plt.title("Sequence Prediction using LSTM (PyTorch)")
plt.xlabel("Time Steps")
plt.ylabel("Value")
plt.legend()
plt.show()
```