Lab : 10 PERFORM COMPRESSION ON MNIST DATASET
09-10-25 USING AUTOENCODER.

AIM
To implement an autoencoder for compressing and reconstructing images from the MNIST dataset.

Objectives :
1. To understand the concept and working of autoencoders
2. To perform image compression using the encoder-decoder.
3. To visualize and analyze reconstructed images and the quality
4. To explore dimensionality reduction through neural networks

pseudocode

```
start
    Import necessary libraries (tensorflow/
Keras, numpy, matplotlib)
    Load MNIST dataset
    Normalise pixel values (0-1)
    flatten images into 784-dimensional
    vectors
    Define autoencoder architecture
    Encoder
        Input layer (784)
        Dense (128, activation = 'relu')
        Dense (64, activation = 'relu')
        Dense (32, activation = 'relu')
    Decoder
        Dense (64, activation = 'relu')
```

Dense(784, activation ='sigmoid')

compile the model

Loss : MSE

optimizer : Adam

Evaluate model performance on test data.

plot training loss curve

END

Observation

Training Behavior: Autoencoder learned to reconstruct digits over epochs. Training loss decreased steadily.

Compression effect : 784D inputs Compressed to 32D latent space. reconstructed images kept key digit details, with slight loss of sharpness.

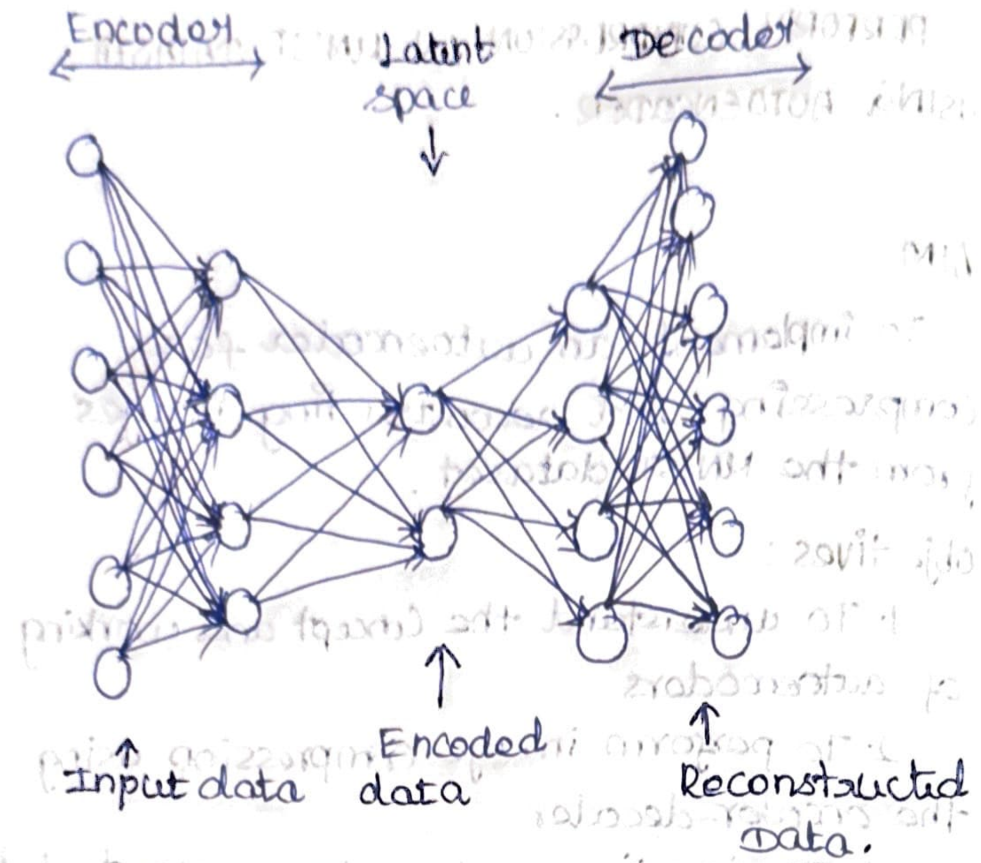
Visualization: orginal vs reconstructed images showed strong similarity. latent space captured distinct digit patterns.

Interpretation: performed non-linear dimensionality reduction. learned features useful for clustering or classification

[illegible]

Result:

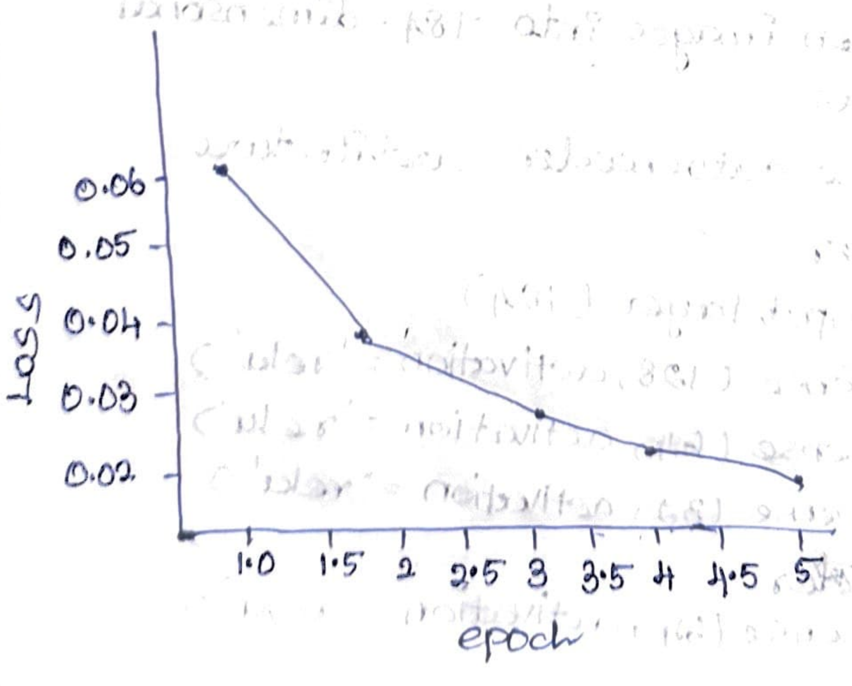
Implement perform compression on MNIST dataset using autoencoder.

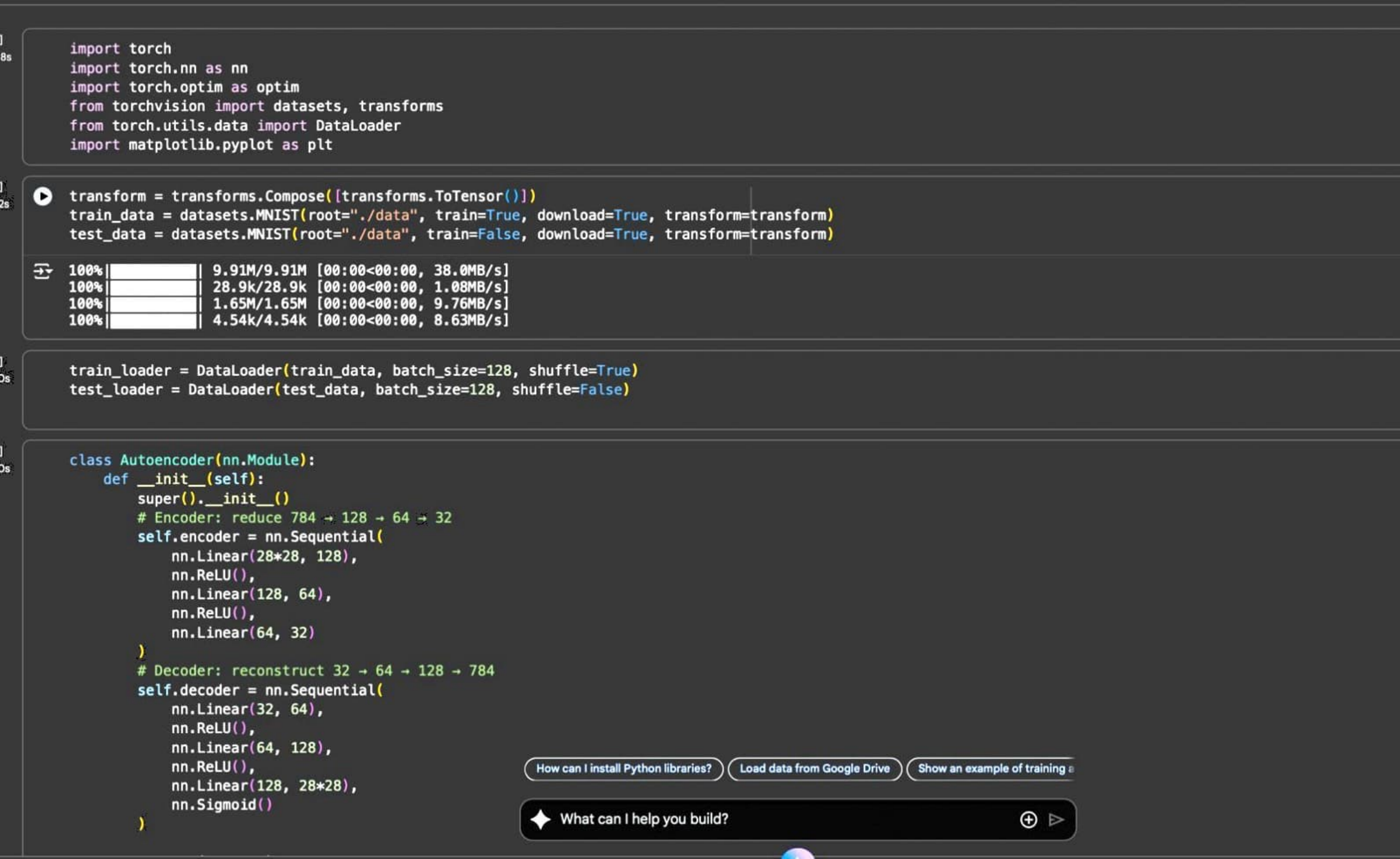


Output

```
Epoch [1/5], Loss : 0.0621
Epoch [2/5], Loss : 0.0321
Epoch [3,5], Loss : 0.0249
Epoch [4,5], Loss : 0.0219
Epoch [5,5], Loss : 0.0197
```

Accuracy : 98.17 %




8s
2s
0s
0s
How can I install Python libraries?
Load data from Google Drive
Show an example of training a
What can I help you build?

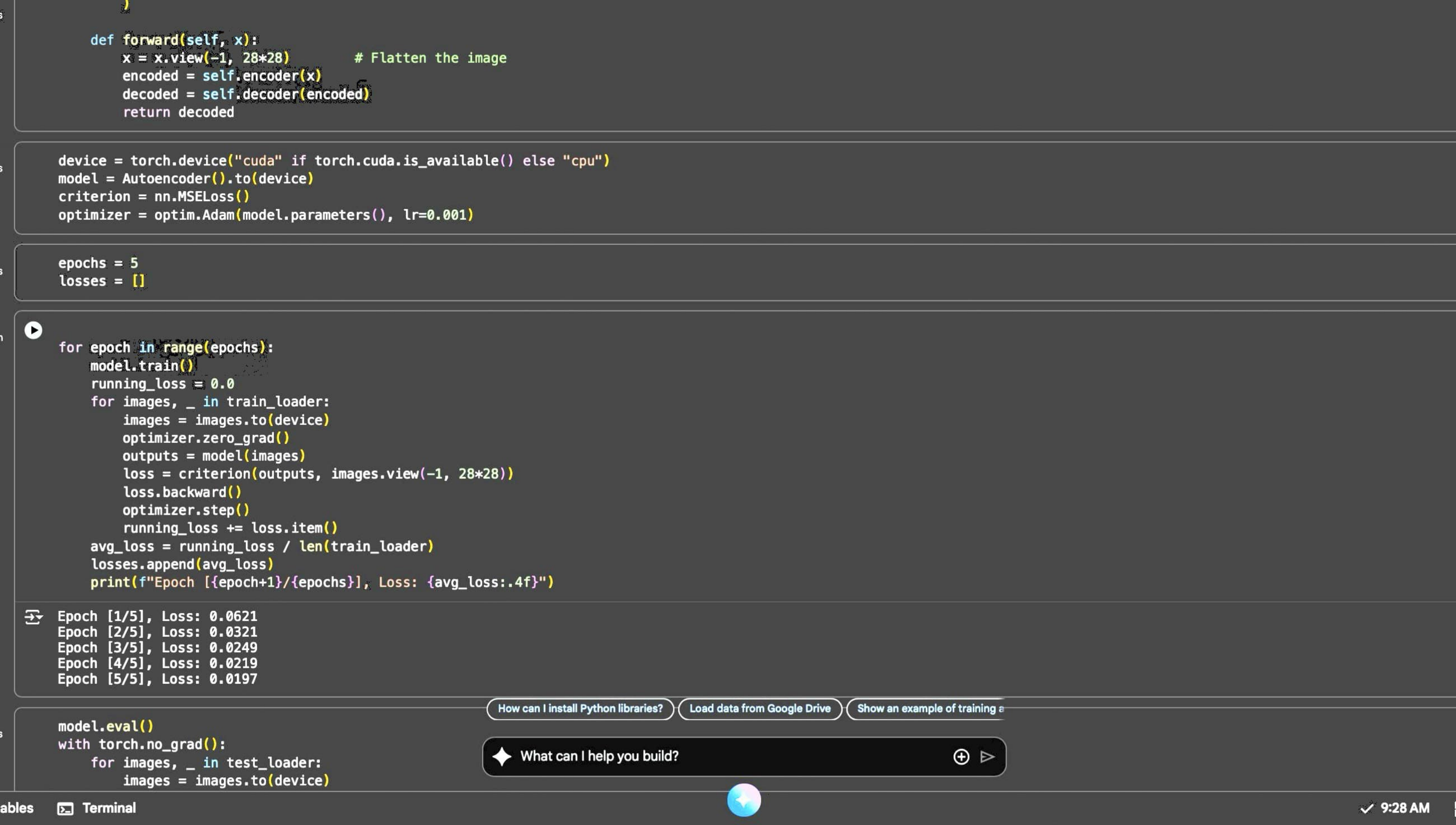
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
```

```python
transform = transforms.Compose([transforms.ToTensor()])
train_data = datasets.MNIST(root="./data", train=True, download=True, transform=transform)
test_data = datasets.MNIST(root="./data", train=False, download=True, transform=transform)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 38.0MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.08MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 9.76MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 8.63MB/s]
```

```python
train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=False)
```

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        # Encoder: reduce 784 → 128 → 64 → 32
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32)
        )
        # Decoder: reconstruct 32 → 64 → 128 → 784
        self.decoder = nn.Sequential(
            nn.Linear(32, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid()
        )
```



```
        )

    def forward(self, x):
        x = x.view(-1, 28*28)        # Flatten the image
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
epochs = 5
losses = []
```

```
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for images, _ in train_loader:
        images = images.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, images.view(-1, 28*28))
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_loss = running_loss / len(train_loader)
    losses.append(avg_loss)
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {avg_loss:.4f}")
```

```
Epoch [1/5], Loss: 0.0621
Epoch [2/5], Loss: 0.0321
Epoch [3/5], Loss: 0.0249
Epoch [4/5], Loss: 0.0219
Epoch [5/5], Loss: 0.0197
```

```
model.eval()
with torch.no_grad():
    for images, _ in test_loader:
        images = images.to(device)
```

How can I install Python libraries?
Load data from Google Drive
Show an example of training a
What can I help you build?
ables
Terminal
9:28 AM

ds + Code + Text ▷ Run all

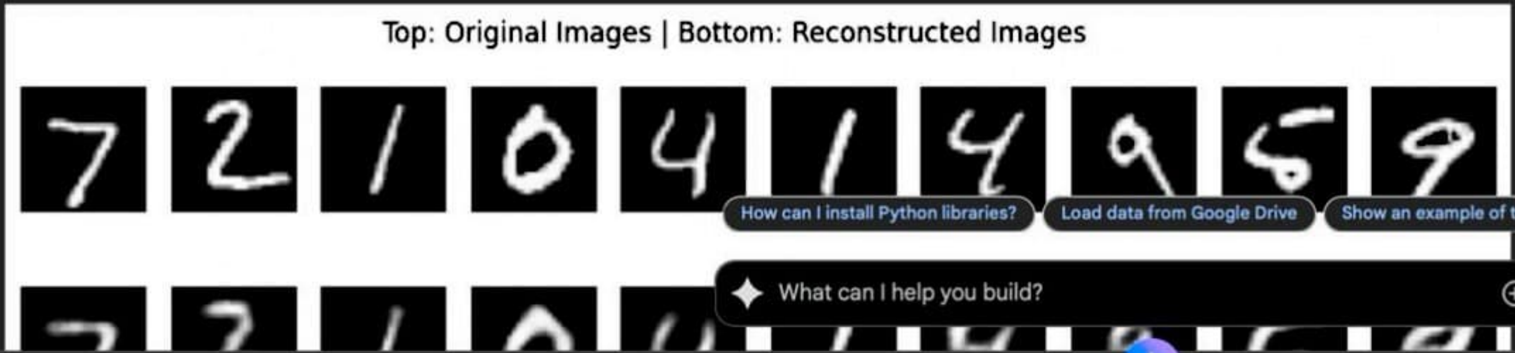
```
model.eval()
with torch.no_grad():
    for images, _ in test_loader:
        images = images.to(device)
        outputs = model(images)
        break
```

```
model.eval()
total_loss = 0
with torch.no_grad():
    for images, _ in test_loader:
        images = images.to(device)
        outputs = model(images)
        loss = criterion(outputs, images.view(-1, 28*28))
        total_loss += loss.item()

avg_mse = total_loss / len(test_loader)
accuracy = (1 - avg_mse) * 100
print(f"Approximate Reconstruction Accuracy: {accuracy:.2f}%")
```

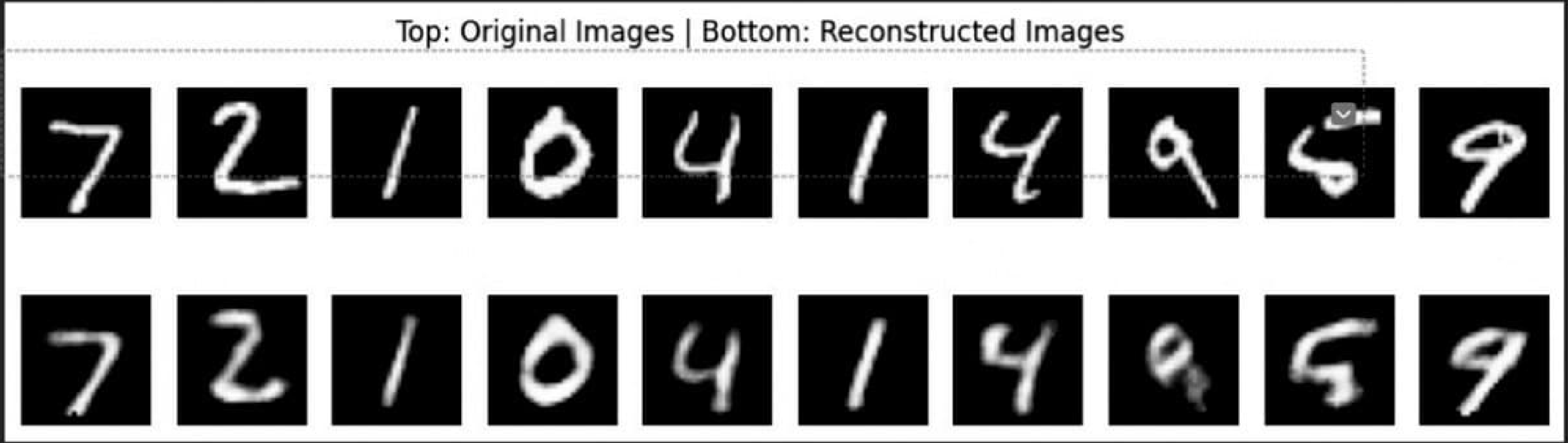
```
Approximate Reconstruction Accuracy: 98.17%
```

```
f, axarr = plt.subplots(2, 10, figsize=(12, 3))
for i in range(10):
    axarr[0][i].imshow(images[i].cpu().squeeze(), cmap="gray")
    axarr[0][i].axis("off")
    axarr[1][i].imshow(outputs[i].cpu().view(28, 28), cmap="gray")
    axarr[1][i].axis("off")
plt.suptitle("Top: Original Images | Bottom: Reconstructed Images")
plt.show()
```

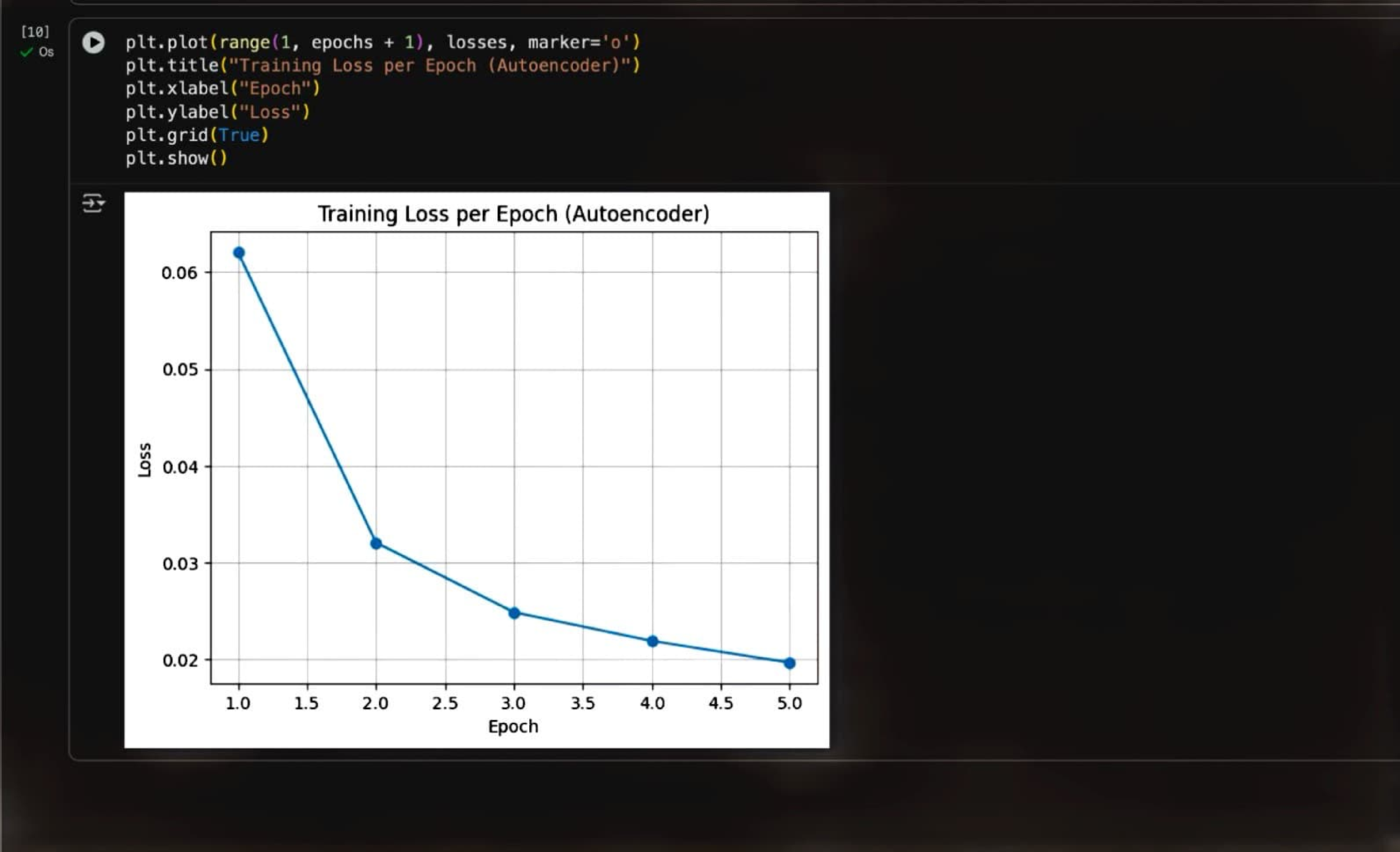


How can I install Python libraries?
Load data from Google Drive
Show an example of training a
What can I help you build?

```python
f, axarr = plt.subplots(2, 10, figsize=(12, 3))
for i in range(10):
    axarr[0][i].imshow(images[i].cpu().squeeze(), cmap="gray")
    axarr[0][i].axis("off")
    axarr[1][i].imshow(outputs[i].cpu().view(28, 28), cmap="gray")
    axarr[1][i].axis("off")
plt.suptitle("Top: Original Images | Bottom: Reconstructed Images")
plt.show()
```

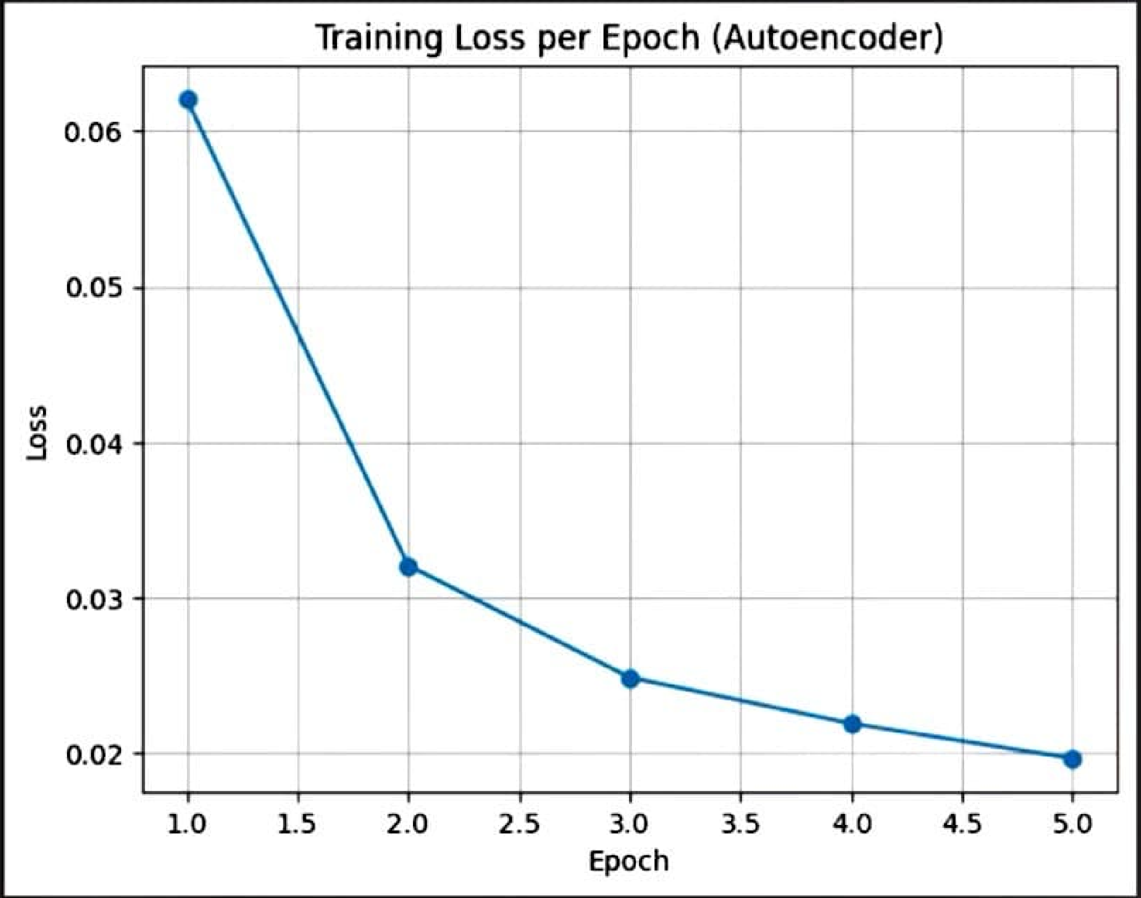


```python
plt.plot(range(1, epochs + 1), losses, marker='o')
plt.title("Training Loss per Epoch (Autoencoder)")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```



[10]
0s

```python
plt.plot(range(1, epochs + 1), losses, marker='o')
plt.title("Training Loss per Epoch (Autoencoder)")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```


Training Loss per Epoch (Autoencoder)
Loss
Epoch