```
import pandas as pd
import re

df = pd.read_csv("/content/poems-100.csv")
df["text"] = df["text"].astype(str)

text = " ".join(df["text"].values).lower()
text = re.sub(r'[^a-zA-Z\s]', '', text)

words = text.split()
```

```
from collections import Counter

word_counts = Counter(words)

vocab = sorted(set(words))
vocab.append("<UNK>")

word_to_idx = {w: i for i, w in enumerate(vocab)}
idx_to_word = {i: w for w, i in word_to_idx.items()}

vocab_size = len(vocab)

print("Vocabulary Size:", vocab_size)
```

```
Vocabulary Size: 2211
```

```
encoded = [word_to_idx.get(w, word_to_idx["<UNK>"]) for w in words]

sequence_length = 10

X = []
y = []

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

import torch

X = torch.tensor(X)
y = torch.tensor(y)

print("Dataset Shape:", X.shape)
```

```
Dataset Shape: torch.Size([7433, 10])
```

```
import torch.nn as nn

class EmbeddingRNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_size):
        super(EmbeddingRNN, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, x):
        x = self.embedding(x)
        out, _ = self.lstm(x)
        out = self.fc(out[:, -1, :])
        return out
```

```
        return out
```

```
embed_dim = 100
hidden_size = 128

model = EmbeddingRNN(vocab_size, embed_dim, hidden_size)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.003)
```

```
from torch.utils.data import DataLoader, TensorDataset

dataset = TensorDataset(X, y)
loader = DataLoader(dataset, batch_size=64, shuffle=True)

epochs = 50

for epoch in range(epochs):
    total_loss = 0

    for batch_x, batch_y in loader:
        optimizer.zero_grad()

        output = model(batch_x)
        loss = criterion(output, batch_y)

        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 5)

        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {total_loss/len(loader):.4f}")
```

```
Epoch 1, Loss: 6.6493
Epoch 2, Loss: 5.9190
Epoch 3, Loss: 5.3078
Epoch 4, Loss: 4.4946
Epoch 5, Loss: 3.5862
Epoch 6, Loss: 2.7240
Epoch 7, Loss: 1.9799
Epoch 8, Loss: 1.4104
Epoch 9, Loss: 0.9782
Epoch 10, Loss: 0.6637
Epoch 11, Loss: 0.4392
Epoch 12, Loss: 0.2912
Epoch 13, Loss: 0.1938
Epoch 14, Loss: 0.1354
Epoch 15, Loss: 0.0983
Epoch 16, Loss: 0.0720
Epoch 17, Loss: 0.0554
Epoch 18, Loss: 0.0449
Epoch 19, Loss: 0.0368
Epoch 20, Loss: 0.0313
Epoch 21, Loss: 0.0273
Epoch 22, Loss: 0.0234
Epoch 23, Loss: 0.0203
Epoch 24, Loss: 0.0179
Epoch 25, Loss: 0.0159
Epoch 26, Loss: 0.0142
Epoch 27, Loss: 0.0127
Epoch 28, Loss: 0.0115
Epoch 29, Loss: 0.0104
Epoch 30, Loss: 0.0094
```

```
Epoch 31, Loss: 0.0086
Epoch 32, Loss: 0.0079
Epoch 33, Loss: 0.0072
Epoch 34, Loss: 0.0066
Epoch 35, Loss: 0.0060
Epoch 36, Loss: 0.0055
Epoch 37, Loss: 0.0051
Epoch 38, Loss: 0.0047
Epoch 39, Loss: 0.0043
Epoch 40, Loss: 0.0040
Epoch 41, Loss: 0.0037
Epoch 42, Loss: 0.0034
Epoch 43, Loss: 0.0032
Epoch 44, Loss: 0.0029
Epoch 45, Loss: 0.0027
Epoch 46, Loss: 0.0025
Epoch 47, Loss: 0.0024
Epoch 48, Loss: 0.0022
Epoch 49, Loss: 0.0021
Epoch 50, Loss: 0.0019
```

```
import torch.nn.functional as F
import numpy as np

def generate_text(start_text, length=30, temperature=0.8):
    model.eval()

    words_list = start_text.lower().split()
    input_seq = [word_to_idx.get(w, word_to_idx["<UNK>"]) for w in words_list]

    for _ in range(length):

        seq = input_seq[-sequence_length:]

        if len(seq) < sequence_length:
            seq = [0]*(sequence_length - len(seq)) + seq

        x = torch.tensor([seq])

        output = model(x)
        output = output / temperature

        probs = F.softmax(output, dim=1).detach().numpy().ravel()
        predicted = np.random.choice(len(probs), p=probs)

        words_list.append(idx_to_word[predicted])
        input_seq.append(predicted)

    return " ".join(words_list)
```

```
print(generate_text("the moon", 30, temperature=0.8))
```

```
the moon ne of the fa here and the nor the negro of the wild gande yahonk he say the pert
```

```
Start coding or generate with AI.
```