```
import numpy as np
import pandas as pd
from keras import Sequential
from keras.layers import Dense,LSTM,Embedding
```

```
df = pd.read_csv('/content/poems-100.csv')
```

```
df.head(10)
```

	text
0	O my Luve's like That’s newly spr O my Luve's ...
1	The rose is red, The violet's blue, Sugar is s...
2	How do I love the I love thee to the My soul c...
3	Had I the heaven Enwrought with g The blue and...
4	I.\nEnough! we're We sit beside And wish that ...
5	She walks in bea Of cloudless clim And all tha...
6	GIVE all to love; Obey thy heart; Friends, kin...
7	THE gray sea an And the yellow h And the start...
8	There! See the li A chain of stars Why can't y...
9	There was a ne I wish you could With stars like d

Next steps: Generate code with df New interactive sheet

```
df.duplicated().sum()
```

```
np.int64(0)
```

```
df.shape
```

```
(82, 1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82 entries, 0 to 81
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    82 non-null     object
dtypes: object(1)
memory usage: 788.0+ bytes
```

```
df.isnull()
```

	text
0	False
1	False
2	False
3	False
4	False

...	...
77	False
78	False
79	False
80	False
81	False

82 rows × 1 columns

```
df.notnull().sum()
```

	0
text	82

dtype: int64

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.1
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/pyth
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.2
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (fr
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.12/dist
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from kera
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from ker
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from ke
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/py
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (fro
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(oov_token='<nothing')
```

```
tokenizer.fit_on_texts(df['text'])
```

```
tokenizer.word_index
```

```
{'<nothing': 1,
 'the': 2,
 'i': 3,
 'and': 4,
 'a': 5,
 'of': 6,
 'my': 7,
 'is': 8,
 'to': 9,
 'you': 10,
 'in': 11,
 'not': 12,
 'it': 13,
 'that': 14,
 't': 15,
 'we': 16,
 'for': 17,
 'with': 18,
 'they': 19,
 'o': 20,
 'do': 21,
 'th': 22,
 'but': 23,
 'have': 24,
 'all': 25,
 'he': 26,
 'me': 27,
 'am': 28,
 'will': 29,
 'this': 30,
 's': 31,
 'as': 32,
 'so': 33,
 'what': 34,
 'm': 35,
 'be': 36,
 'love': 37,
 'at': 38,
 'where': 39,
 'on': 40,
 'are': 41,
 'w': 42,
 'one': 43,
 'if': 44,
 'by': 45,
 'when': 46,
 'was': 47,
 'b': 48,
 'our': 49,
 'no': 50,
 'or': 51,
 'an': 52,
 'his': 53,
 '–': 54,
 'shall': 55,
 'see': 56,
 'nor': 57,
 'know': 58,
```

```
tokenizer.document_count
```

```
82
```

```
sequences = tokenizer.texts_to_sequences(df['text'])
sequences
```

```
[[20,
  7,
  369,
  61,
  370,
  640,
  371,
  20,
  7,
  369,
  61,
  370,
  641,
  372,
  32,
  262,
  373,
  68,
  33,
  263,
  11,
  264,
  5,
  4,
  3,
  29,
  264,
  22,
  94,
  374,
  2,
  375,
  376,
  94,
  374,
  2,
  375,
  376,
  4,
  2,
  642,
  35,
  3,
  29,
  264,
  73,
  643,
  175,
  2,
  644,
  4,
  265,
  73,
  16,
  4,
  265,
  73,
  16,
```

```
from keras.utils import pad_sequences
```

```
sequences = pad_sequences(sequences,padding='post')
sequences
```

```
array([[ 20,   7, 369, ...,   0,   0,   0],
       [  2, 266,   8, ...,   0,   0,   0],
       [ 69,  21,   3, ...,   0,   0,   0],
       ...,
       [ 20, 496,   6, ...,   0,   0,   0],
       [  2, 342,   8, ...,   0,   0,   0],
       [ 13,  47,  52, ...,   0,   0,   0]], dtype=int32)
```

```
# from sklearn.model_selection import train_test_split

# y = np.zeros(len(sequences))

# X_train, X_test, y_train, y_test = train_test_split(sequences, y, test_size=0.2, rando
df["text"] = df["text"].astype(str)
text = " ".join(df["text"].values).lower()

import re
text = re.sub(r'[^a-zA-Z\s]', '', text)

words = text.split()
```

```
# text = " ".join(df.iloc[:,0].astype(str))
```

```
# text = text.lower()
```

```
# text
```

```
# words = text.split()
```

```
from collections import Counter

word_counts = Counter(words)

words = [w for w in words if word_counts[w] > 2]

# Build new vocabulary
vocab = sorted(set(words))
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("New Vocabulary Size:", vocab_size)
```

```
New Vocabulary Size: 354
```

```
def onehot(idx,vocab_size):
  vec = np.zeros(vocab_size,1)
  vec[idx] =1
  return vec
```

```
sequence_length = 5

encoded = [word_to_idx[w] for w in words]

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)

# One-hot encode
X_onehot = torch.nn.functional.one_hot(X, num_classes=vocab_size).float()
```

```
class SimpleRNN:
    def __init__(self, input_size, hidden_size, output_size, lr=0.01):
        self.hidden_size = hidden_size
        self.lr = lr

        # Initialize weights
        self.Wxh = np.random.randn(hidden_size, input_size) * 0.01
        self.Whh = np.random.randn(hidden_size, hidden_size) * 0.01
        self.Why = np.random.randn(output_size, hidden_size) * 0.01

        self.bh = np.zeros((hidden_size, 1))
        self.by = np.zeros((output_size, 1))

    def forward(self, inputs):
        h = np.zeros((self.hidden_size, 1))
        for idx in inputs:
            x = one_hot(idx, vocab_size)
            h = np.tanh(self.Wxh @ x + self.Whh @ h + self.bh)
        y = self.Why @ h + self.by
        return y, h
```

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

def generate_text(start_text, length=20, temperature=0.8):
    model.eval()

    words_list = start_text.lower().split()
    input_seq = [word_to_idx[w] for w in words_list]

    for _ in range(length):
        x = torch.tensor([input_seq[-sequence_length:]])
        x = torch.nn.functional.one_hot(x, num_classes=vocab_size).float()

        output = model(x)
        output = output / temperature

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

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

    return " ".join(words_list)
```

```
X_onehot = torch.nn.functional.one_hot(X, num_classes=vocab_size).float()
```

Start coding or generate with AI.

```
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNNModel, self).__init__()

        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, hidden = self.rnn(x)
        out = self.fc(out[:, -1, :])  # last time step
        return out
```

```
hidden_size = 128

model = RNNModel(vocab_size, hidden_size, vocab_size)

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

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

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

dataset = TensorDataset(X_onehot, y)
loader = DataLoader(dataset, batch_size=32, shuffle=True)

hidden_size = 64

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

epochs = 80

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()

        # ✅ Gradient Clipping
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=5)

        optimizer.step()

        total_loss += loss.item()

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

```
Epoch 1, Loss: 5.0116
Epoch 2, Loss: 4.7716
Epoch 3, Loss: 4.5824
Epoch 4, Loss: 4.3745
Epoch 5, Loss: 4.1462
Epoch 6, Loss: 3.9081
Epoch 7, Loss: 3.6728
Epoch 8, Loss: 3.4405
Epoch 9, Loss: 3.2039
Epoch 10, Loss: 2.9775
Epoch 11, Loss: 2.7692
Epoch 12, Loss: 2.5655
Epoch 13, Loss: 2.3764
Epoch 14, Loss: 2.2031
Epoch 15, Loss: 2.0294
Epoch 16, Loss: 1.8751
Epoch 17, Loss: 1.7300
Epoch 18, Loss: 1.5861
Epoch 19, Loss: 1.4583
Epoch 20, Loss: 1.3594
Epoch 21, Loss: 1.2290
Epoch 22, Loss: 1.1291
Epoch 23, Loss: 1.0309
Epoch 24, Loss: 0.9477
Epoch 25, Loss: 0.8618
Epoch 26, Loss: 0.7933
Epoch 27, Loss: 0.7168
Epoch 28, Loss: 0.6571
Epoch 29, Loss: 0.6024
Epoch 30, Loss: 0.5541
Epoch 31, Loss: 0.5146
Epoch 32, Loss: 0.4638
Epoch 33, Loss: 0.4278
Epoch 34, Loss: 0.3903
Epoch 35, Loss: 0.3590
Epoch 36, Loss: 0.3178
Epoch 37, Loss: 0.2894
Epoch 38, Loss: 0.2612
Epoch 39, Loss: 0.2430
Epoch 40, Loss: 0.2363
Epoch 41, Loss: 0.2276
Epoch 42, Loss: 0.1969
Epoch 43, Loss: 0.1959
Epoch 44, Loss: 0.1854
Epoch 45, Loss: 0.1647
Epoch 46, Loss: 0.1339
Epoch 47, Loss: 0.1394
Epoch 48, Loss: 0.1197
Epoch 49, Loss: 0.1060
Epoch 50, Loss: 0.1058
Epoch 51, Loss: 0.1042
Epoch 52, Loss: 0.1224
Epoch 53, Loss: 0.1206
Epoch 54, Loss: 0.1032
Epoch 55, Loss: 0.1223
Epoch 56, Loss: 0.1687
Epoch 57, Loss: 0.1490
Epoch 58, Loss: 0.1196
```

```python
import torch.nn.functional as F

def generate_text(start_words, length=20, temperature=0.8):
    model.eval()

    words_list = start_words.split()
    # Filter out start words not in the vocabulary
    input_seq = [word_to_idx[w] for w in words_list if w in word_to_idx]

    # If no valid starting words are found, return an error or empty string
    if not input_seq:
        return "No valid starting words found in vocabulary."

    for _ in range(length):
        x = torch.tensor([input_seq[-sequence_length:]])
        x = torch.nn.functional.one_hot(x, num_classes=vocab_size).float()

        output = model(x)
        output = output / temperature

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

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

    return " ".join(words_list)
```

```python
print(generate_text("the moon", 30, temperature=0.9))
```

```
the moon f of mine to be b sea had i love the to th and the from a the i h not t the o my
```

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