```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("deepshah16/song-lyrics-dataset")
path = path + '/csv'

print("Path to dataset files:", path)
```

```
Path to dataset files: /root/.cache/kagglehub/datasets/deepshah16/song-lyrics-dataset/versions/5/csv
```

```
import os

# Define the folder path
folder_path = path

# List items in the folder
items = os.listdir(folder_path)

# Print the items
for item in items:
    print(item)
```

```
CharliePuth.csv
BTS.csv
TaylorSwift.csv
KatyPerry.csv
Khalid.csv
BillieEilish.csv
Maroon5.csv
Rihanna.csv
PostMalone.csv
Drake.csv
Eminem.csv
SelenaGomez.csv
JustinBieber.csv
LadyGaga.csv
ArianaGrande.csv
EdSheeran.csv
```

```
     CardiB.csv
     ColdPlay.csv
     NickiMinaj.csv
     Beyonce.csv
     DuaLipa.csv
```

```
seq_length = 50
epochs = 30
```

```
import pandas as pd
import re

def load_and_preprocess_data(data_folder):
    """
    Load artist lyrics from CSV files and preprocess them
    """
    artists_data = {}
    all_text = ""

    # Load each artist's lyrics from CSV files
    for filename in os.listdir(data_folder):
        if filename.endswith('.csv'):
            artist_name = filename.split('.')[0]  # Get artist name from filename
            file_path = os.path.join(data_folder, filename)
            print('artist_name: ', artist_name)

            try:
                df = pd.read_csv(file_path)

                # Extract lyrics column (adjust column name as needed)
                # Lyrics column is the last column in this dataset
                lyrics = df.iloc[:, -1].dropna().astype(str).tolist()

                # Clean lyrics
                cleaned_lyrics = []
                for lyric in lyrics:
                    # Basic cleaning
                    text = lyric.lower()
                    # Remove URLs, special characters, etc.
                    text = re.sub(r'http\S+', '', text)
                    text = re.sub(r'[^\w\s]', ' ', text)
                    text = re.sub(r'\s+', ' ', text).strip()

                    if len(text) > 10:  # Skip very short lyrics
                        cleaned_lyrics.append(text)

                artists_data[artist_name] = cleaned_lyrics
                all_text += " ".join(cleaned_lyrics) + " "
                print(f"Loaded {len(cleaned_lyrics)} songs for {artist_name}")

            except Exception as e:
                print(f"Error loading {filename}: {e}")

    return artists_data, all_text



print("Loading and preprocessing lyrics data...")
artists_data, all_text = load_and_preprocess_data(path)
```

```
Loading and preprocessing lyrics data...
artist_name:  CharliePuth
Loaded 75 songs for CharliePuth
artist_name:  BTS
Loaded 269 songs for BTS
artist_name:  TaylorSwift
Loaded 477 songs for TaylorSwift
artist_name:  KatyPerry
Loaded 322 songs for KatyPerry
artist_name:  Khalid
Loaded 64 songs for Khalid
artist_name:  BillieEilish
Loaded 145 songs for BillieEilish
artist_name:  Maroon5
Loaded 197 songs for Maroon5
artist_name:  Rihanna
Loaded 397 songs for Rihanna
```

```
artist_name:  PostMalone
Loaded 148 songs for PostMalone
artist_name:  Drake
Loaded 463 songs for Drake
artist_name:  Eminem
Loaded 516 songs for Eminem
artist_name:  SelenaGomez
Loaded 174 songs for SelenaGomez
artist_name:  JustinBieber
Loaded 345 songs for JustinBieber
artist_name:  LadyGaga
Loaded 390 songs for LadyGaga
artist_name:  ArianaGrande
Loaded 0 songs for ArianaGrande
artist_name:  EdSheeran
Loaded 292 songs for EdSheeran
artist_name:  CardiB
Loaded 74 songs for CardiB
artist_name:  ColdPlay
Loaded 333 songs for ColdPlay
artist_name:  NickiMinaj
Loaded 318 songs for NickiMinaj
artist_name:  Beyonce
Loaded 406 songs for Beyonce
artist_name:  DuaLipa
Loaded 239 songs for DuaLipa
```

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

def create_sequences(text, seq_length=50):
    """
    Create sequences of tokens from text for training
    """
    # Tokenize text
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts([text])
    total_words = len(tokenizer.word_index) + 1
    # Create sequences
    input_sequences = []
    for i in range(0, len(text.split()) - seq_length):
        sequence = text.split()[i:i + seq_length + 1]
        input_sequences.append(" ".join(sequence))

    # Tokenize sequences
    tokenized_sequences = tokenizer.texts_to_sequences(input_sequences)

    # Create input and output arrays
    X, y = [], []
    for sequence in tokenized_sequences:
        X.append(sequence[:-1])
        y.append(sequence[-1])

    X = np.array(pad_sequences(X, maxlen=seq_length))
    y = np.array(tf.keras.utils.to_categorical(y, num_classes=total_words))

    return X, y, tokenizer, total_words
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

def build_model(total_words, seq_length=50, embedding_dim=100):
    """
    Build LSTM model for text generation
    """
    model = Sequential([
        Embedding(total_words, embedding_dim, input_length=seq_length),
        LSTM(units=150, return_sequences=True),
        Dropout(0.2),
        LSTM(units=100),
        Dropout(0.2),
        Dense(total_words, activation='softmax')
    ])
```

```
        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        return model


    def prepare_artist_specific_data(artists_data, seq_length=50):
        """
        Prepare data for artist-specific training
        """
        all_X = []
        all_y = []
        artist_tokenizers = {}

        for artist, lyrics in artists_data.items():
            artist_text = " ".join(lyrics)
            if len(artist_text.split()) <= seq_length:
                print(f"Not enough data for {artist}, skipping")
                continue

            X, y, tokenizer, total_words = create_sequences(artist_text, seq_length)

            # Store tokenizer for this artist
            artist_tokenizers[artist] = {
                'tokenizer': tokenizer,
                'total_words': total_words
            }

            # Create artist-specific dataset
            artist_X = X
            artist_y = y

            all_X.append(artist_X)
            all_y.append(artist_y)

            print(f"Created {len(X)} sequences for {artist}")

        return artist_tokenizers
```

```
import matplotlib.pyplot as plt

def train_artist_specific_models(artists_data, seq_length=50, epochs=50):
    """
    Train a model for each artist
    """
    artist_models = {}

    for artist, lyrics in artists_data.items():
        print(f"\nTraining model for {artist}...")

        # Prepare data
        artist_text = " ".join(lyrics)
        if len(artist_text.split()) <= seq_length:
            print(f"Not enough data for {artist}, skipping")
            continue

        X, y, tokenizer, total_words = create_sequences(artist_text, seq_length)

        # Build and train model
        model = build_model(total_words, seq_length)

        history = model.fit(
            X, y,
            epochs=epochs,
            batch_size=128,
            validation_split=0.1,
            callbacks=[
                tf.keras.callbacks.EarlyStopping(
                    monitor='val_loss',
                    patience=5,
                    restore_best_weights=True
                )
            ]
        )

        # Save artist-specific model and tokenizer
        artist_models[artist] = {
            'model': model,
            'tokenizer': tokenizer,
            'total_words': total_words
        }

        # Plot training history
        plt.figure(figsize=(12, 4))
        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title(f'{artist} Model Accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.legend(['Train', 'Validation'])

        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title(f'{artist} Model Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend(['Train', 'Validation'])
        plt.tight_layout()
        plt.savefig(f"{artist}_training_history.png")
        plt.close()

    return artist_models


# Train separate models for each artist
'''
# tdata can be used to select a specific artist(s) to train model on
tdata = {
    'CharliePuth': artists_data['CharliePuth']
}
'''
artist_models = train_artist_specific_models(artists_data, seq_length, epochs)
```

```
Epoch 2/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 81s 365ms/step - accuracy: 0.0530 - loss: 5.6444 - val_accuracy: 0.0356 - val_loss: 6.06
Epoch 3/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 368ms/step - accuracy: 0.0507 - loss: 5.5491 - val_accuracy: 0.0376 - val_loss: 5.94
Epoch 4/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 65s 366ms/step - accuracy: 0.0622 - loss: 5.3359 - val_accuracy: 0.0550 - val_loss: 5.82
Epoch 5/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 65s 367ms/step - accuracy: 0.0747 - loss: 5.1312 - val_accuracy: 0.0743 - val_loss: 5.75
Epoch 6/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 366ms/step - accuracy: 0.0975 - loss: 4.9532 - val_accuracy: 0.0933 - val_loss: 5.67
Epoch 7/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 66s 368ms/step - accuracy: 0.1172 - loss: 4.8514 - val_accuracy: 0.0917 - val_loss: 5.64
Epoch 8/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 367ms/step - accuracy: 0.1239 - loss: 4.7538 - val_accuracy: 0.1072 - val_loss: 5.64
Epoch 9/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 367ms/step - accuracy: 0.1359 - loss: 4.6282 - val_accuracy: 0.1206 - val_loss: 5.57
Epoch 10/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 365ms/step - accuracy: 0.1553 - loss: 4.5450 - val_accuracy: 0.1321 - val_loss: 5.56
Epoch 11/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 363ms/step - accuracy: 0.1634 - loss: 4.4295 - val_accuracy: 0.1352 - val_loss: 5.54
Epoch 12/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 84s 376ms/step - accuracy: 0.1713 - loss: 4.3690 - val_accuracy: 0.1439 - val_loss: 5.55
Epoch 13/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 64s 361ms/step - accuracy: 0.1834 - loss: 4.2893 - val_accuracy: 0.1408 - val_loss: 5.51
Epoch 14/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 65s 364ms/step - accuracy: 0.1926 - loss: 4.2066 - val_accuracy: 0.1566 - val_loss: 5.50
Epoch 15/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 84s 374ms/step - accuracy: 0.2066 - loss: 4.1289 - val_accuracy: 0.1534 - val_loss: 5.48
Epoch 16/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 80s 363ms/step - accuracy: 0.2109 - loss: 4.0420 - val_accuracy: 0.1677 - val_loss: 5.48
Epoch 17/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 65s 364ms/step - accuracy: 0.2140 - loss: 3.9684 - val_accuracy: 0.1684 - val_loss: 5.47
Epoch 18/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 364ms/step - accuracy: 0.2312 - loss: 3.9196 - val_accuracy: 0.1692 - val_loss: 5.45
Epoch 19/30
```

```
178/178 ━━━━━━━━━━━━━━━━━━━━ 80s 368ms/step - accuracy: 0.2572 - loss: 3.6699 - val_accuracy: 0.1937 - val_loss: 5.49
Epoch 23/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 369ms/step - accuracy: 0.2602 - loss: 3.6260 - val_accuracy: 0.2143 - val_loss: 5.45
Epoch 24/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 66s 368ms/step - accuracy: 0.2744 - loss: 3.5866 - val_accuracy: 0.2119 - val_loss: 5.45
Epoch 25/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 65s 365ms/step - accuracy: 0.2810 - loss: 3.5262 - val_accuracy: 0.2127 - val_loss: 5.43
Epoch 26/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 368ms/step - accuracy: 0.2820 - loss: 3.4766 - val_accuracy: 0.2143 - val_loss: 5.45
Epoch 27/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 82s 368ms/step - accuracy: 0.2884 - loss: 3.4686 - val_accuracy: 0.2135 - val_loss: 5.45
Epoch 28/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 66s 371ms/step - accuracy: 0.2972 - loss: 3.3893 - val_accuracy: 0.2242 - val_loss: 5.43
Epoch 29/30
178/178 ━━━━━━━━━━━━━━━━━━━━ 84s 385ms/step - accuracy: 0.3143 - loss: 3.3210 - val_accuracy: 0.2250 - val_loss: 5.45
Epoch 30/30
```

```python
def generate_text(model, tokenizer, seed_text, artist, next_words=50, temperature=1.0):
    """
    Generate text in the style of a specific artist
    """
    generated_text = seed_text

    for _ in range(next_words):
        # Tokenize and pad the current text
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=model.input_shape[1], padding='pre')

        # Predict next word
        predicted_probs = model.predict(token_list, verbose=0)[0]

        # Apply temperature for randomness
        predicted_probs = np.log(predicted_probs) / temperature
        predicted_probs = np.exp(predicted_probs) / np.sum(np.exp(predicted_probs))

        # Sample from distribution
        predicted_index = np.random.choice(len(predicted_probs), p=predicted_probs)

        # Find the word
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted_index:
                output_word = word
                break

        # Add to the generated text
        seed_text += " " + output_word
        generated_text += " " + output_word

    return generated_text


# Example: Generate lyrics for specific artist
artist = "CharliePuth"  # Change to any artist in your dataset
seed_text = "I feel like"  # Starting text

# Using separate models
if artist in artist_models:
    model = artist_models[artist]['model']
    tokenizer = artist_models[artist]['tokenizer']
    generated_text = generate_text(model, tokenizer, seed_text, artist, next_words=100, temperature=0.7)
    print(f"\nGenerated lyrics in style of {artist}:\n{generated_text}")
else:
    print(f"No model trained for {artist}")
```

```
Generated lyrics in style of CharliePuth:
I feel like it s beautiful at the top up ooh as you re reluctant cause you had to thinking when why there s gonna have
```