CSV code

```
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from collections import defaultdict
import random
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
import numpy as np

nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.8.30)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.5)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kagg
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggl
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.10)
```

```
!kaggle datasets download -d thedevastator/tinystories-narrative-classification
!unzip tinystories-narrative-classification.zip
```

```
Dataset URL: https://www.kaggle.com/datasets/thedevastator/tinystories-narrative-classification
License(s): CC0-1.0
Downloading tinystories-narrative-classification.zip to /content
 97% 561M/576M [00:08<00:00, 77.2MB/s]
100% 576M/576M [00:08<00:00, 72.3MB/s]
Archive:  tinystories-narrative-classification.zip
  inflating: train.csv
  inflating: validation.csv
```

```
df = pd.read_csv('train.csv') #, encoding='ISO-8859-1') #, skiprows=303350, nrows=10)
```

```
import string

text_data = df['text'].str.cat(sep=' ')

# Preprocessing
def preprocess_text(text):
    text_no_punctuation = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text_no_punctuation.lower())
    return tokens

tokens = preprocess_text(text_data)
```

```
# Markov
class MarkovChain:
    def __init__(self, order=2):
        self.order = order
        self.transitions = defaultdict(list)
        self.start_states = []

    def train(self, tokens):
        for i in range(len(tokens) - self.order):
            state = tuple(tokens[i:i+self.order])
            next_token = tokens[i+self.order]
            self.transitions[state].append(next_token)
            if i == 0:
                self.start_states.append(state)

    def generate_sentence(self, max_length=20):
```

```
            state = random.choice(self.start_states)
            sentence = list(state)
            for _ in range(max_length - self.order):
                next_words = self.transitions.get(state)
                if not next_words:
                    break
                next_word = random.choice(next_words)
                sentence.append(next_word)
                state = tuple(sentence[-self.order:])
            return ' '.join(sentence).capitalize() + '.'


    # Train Markov
    markov = MarkovChain(order=2)
    markov.train(tokens)


    # Dataprep LSTM
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts([text_data])
    total_words = len(tokenizer.word_index) + 1
    max_sequence_len = max([len(word_tokenize(sent)) for sent in sent_tokenize(text_data)])


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

    # LSTM
    def build_lstm_model(vocab_size, max_sequence_len):
        model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.Embedding(vocab_size, 100, input_length=max_sequence_len-1))
        model.add(tf.keras.layers.LSTM(150, return_sequences=True))
        model.add(tf.keras.layers.LSTM(100))
        model.add(tf.keras.layers.Dense(vocab_size, activation='softmax'))

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

    def prepare_sequences(text, tokenizer, max_sequence_len, batch_size=1000):
        sequences = []
        lines = text.split('.')

        for line in lines:
            if line.strip():
                token_list = tokenizer.texts_to_sequences([line])[0]
                for i in range(1, len(token_list)):
                    seq = token_list[:i + 1]
                    sequences.append(seq)

                if len(sequences) >= batch_size:
                    sequences = pad_sequences(sequences, maxlen=max_sequence_len, padding='pre')
                    yield sequences[:, :-1], sequences[:, -1]

        if sequences:
            sequences = pad_sequences(sequences, maxlen=max_sequence_len, padding='pre')
            yield sequences[:, :-1], sequences[:, -1]

    def create_dataset(text_data, tokenizer, max_sequence_len, batch_size=1000):
        return tf.data.Dataset.from_generator(
            lambda: prepare_sequences(text_data, tokenizer, max_sequence_len, batch_size),
            output_signature=(
                tf.TensorSpec(shape=(None, max_sequence_len-1), dtype=tf.int32),
                tf.TensorSpec(shape=(None,), dtype=tf.int32)
            )
        ).prefetch(tf.data.AUTOTUNE)

    dataset = create_dataset(text_data, tokenizer, max_sequence_len)
    model = build_lstm_model(total_words, max_sequence_len)
    model.fit(dataset, epochs=10, verbose=1)
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is d
  warnings.warn(
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 268s 78ms/step - accuracy: 0.0978 - loss: 5.8366
Epoch 2/10
   1/3370 ━━━━━━━━━━━━━━━━━━━━ 9:26 168ms/step - accuracy: 0.2808 - loss: 4.1591/usr/lib/python3.10/contextlib.py:153: U
  self.gen.throw(typ, value, traceback)
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 268s 80ms/step - accuracy: 0.2513 - loss: 4.2976
Epoch 3/10
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 269s 80ms/step - accuracy: 0.2779 - loss: 3.9784
Epoch 4/10
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 268s 79ms/step - accuracy: 0.2928 - loss: 3.8172
Epoch 5/10
```

```
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 269s 80ms/step - accuracy: 0.3021 - loss: 3.7129
Epoch 6/10
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 269s 80ms/step - accuracy: 0.3091 - loss: 3.6373
Epoch 7/10
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 321s 80ms/step - accuracy: 0.3148 - loss: 3.5793
Epoch 8/10
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 323s 80ms/step - accuracy: 0.3192 - loss: 3.5344
Epoch 9/10
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 320s 79ms/step - accuracy: 0.3233 - loss: 3.4935
Epoch 10/10
3370/3370 ━━━━━━━━━━━━━━━━━━━━ 267s 79ms/step - accuracy: 0.3269 - loss: 3.4591
<keras.src.callbacks.history.History at 0x7bb2b89d1cc0>
```

```
markov_chain = MarkovChain()
lstm_model = build_lstm_model(total_words, max_sequence_len)
tokenizer = Tokenizer()
max_sequence_len = 20
num_sentences = 10
temperature = 1.0

def generate_seeded_story(seed_word, markov_chain, lstm_model, tokenizer, max_sequence_len, num_sentences=10, next_words=5,
    story = [seed_word]
    initial_sentence = seed_word

    for i in range(num_sentences - 1):
        if i % 2 == 0:
            for _ in range(next_words):
                token_list = tokenizer.texts_to_sequences([initial_sentence])[0]
                token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
                predicted = lstm_model.predict(token_list, verbose=0)
                predicted_word_index = np.argmax(predicted)
                output_word = tokenizer.index_word.get(predicted_word_index, "<OOV>")
                initial_sentence += " " + output_word
            story.append(initial_sentence)
        else:
            tokens = word_tokenize(initial_sentence.lower())
            state = tuple(tokens[-markov_chain.order:])
            next_word = random.choice(markov_chain.transitions.get(state, [""]))
            initial_sentence += " " + next_word
            story.append(initial_sentence)

    return ' '.join(story)

seed_word = "Once"
generated_seeded_story = generate_seeded_story(seed_word, markov, model, tokenizer, max_sequence_len, num_sentences=10, next
print(generated_seeded_story)
```

```
Spot . she had ruined it . lily enjoyed their pot of rice filled with ice cream and yummy food.
```