```
import numpy as np
from tqdm import tqdm
import re

# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# TODO: this thing is useless :) Remove it.
END_OF_SONNET = '*' # Special end of sonnet character
```

Load and preprocess

```
# Load Shakespeare.txt
# LSTM is character-based, so we want each sonnet as a single all-lowercase string
# Punctuation included, leading spaces removed (couplet not formatted)
downloaded = drive.CreateFile({'id':'1cwaY0yRvUNgggFytJpca_1Mao85bawQN'})
downloaded.GetContentFile('filename.txt')

data = []
accum = ''

f = open('filename.txt')

for line in f:
  # Case: just a newline
  if len(line) == 1:
    continue
  # Case: line contains only a number--beginning of new sonnet
  elif line.strip(' ').strip('\n').isnumeric():
    if len(accum) != 0:
      accum += END_OF_SONNET
      data.append(accum)
      accum = ''
  # Case: line of sonnet
  else:
    accum += line.strip(' ').lower()

data = np.array(data)
print(f'data.shape: {data.shape}')
print('')
print(f'data[0]: {data[0]}')
```

```
# Create dataset of 40-character strings
# To speed up training, we use semi-redundant sequences, picking 40-character seque
# X1 is array of 40-character strings
# Y1 is array containing the next character

window = 40
n = 5

X1 = []
Y1 = []
for sonnet in data:
  for i in range(0, len(sonnet)-window-1, n):
    X1.append(sonnet[i:i+window])
    Y1.append(sonnet[i+window])

print(f'len(X1): {len(X1)}\t len(Y1): {len(Y1)}')
print('')
print(f'X1[0]: {X1[0]}')
print(f'Y1[0]: {Y1[0]}')
```

```
# RNN expects each character as a boolean array

# rev_mapping: int -> char
rev_mapping = set()
rev_mapping.update(''.join(data))
rev_mapping = np.array(list(rev_mapping))

# mapping: character --> boolean array
I = np.identity(len(s))
mapping = {k:I[v] for v,k in enumerate(rev_mapping)}

X = np.array([[mapping[k] for k in seq] for seq in X1])
Y = np.array([mapping[k] for k in Y1])
```

```
Y = np.array([mapping[x] for x in Y])

print(f'X.shape: {X.shape}\t Y.shape): {Y.shape}')
print('')
print(f'X[0]: {X[0]}')
print('')
print(f'Y[0]: {Y[0]}')
print('')
print(f'rev_mapping: {rev_mapping}')
```

RNN

```
from keras.models import Sequential
from keras.layers import LSTM, Dense, Activation


rnn = Sequential([
                    LSTM(200, input_shape=(window,len(mapping))),
                    Dense(len(mapping), activation='softmax')
])
rnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']
rnn.fit(X, Y, epochs=100)
```

```
# TODO: Um, these are still a bit nonsense. Did I do something else wrong?
def generate(seed, temperature=1):
  x = seed.lower()
  x = np.array([mapping[k] for k in x]).reshape((1,len(seed),len(s)))
  z = np.log(rnn.predict(x).flatten())
  q = np.exp(z / temperature)
  q /= np.sum(q)
  return np.random.choice(rev_mapping, p=q)

def generate_sequence(seq, temperature=1, max_len=4000):
  while seq[-1] != END_OF_SONNET and len(seq) < max_len:
    seq += generate(seq[len(seq)-40:])
  return seq

for temp in [1.5, 0.75, 0.25, 0.01]:
  seq = generate_sequence('shall i compare thee to a summer\'s day?\n',
                          temperature=temp)
  print(f'{temp}: {seq}\n\n')
```

```
    print(f'{temp}: {seq}\n\n')
```