In [169]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
d as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session
```

Importing Libraries

Here we are importing all the important libraries to solve the sherlock text. In this we will create lstm model and that will predict the next word. For doing this we have taken pandas numpy matplot for data visualization & reading, creating dataframe many more.

In [170]:

```
import warnings
warnings.filterwarning('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Nltk specific requirement
import nltk
nltk.download('omw-1.4')
import re #REGEX very important for text processing
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from numpy import unique
# LSTM, and all
from keras.layers import Dense,Dropout, RNN, LSTM, Activation, Embedding
from keras.models import Sequential
```

Read Text

To read this data we will open that file from folder. After that we will be able to read sherlock text

In [171]:

```
# df = open("../input/sherlocktxt/sherlock.txt")
# df.read()
```

In [172]:

```
df = open("../input/sherlocktxt/sherlock.txt")
text = df.read()
print(text[700:1100])
```

In [173]:

```
def sequence_to_sum(seq):
    """
    Params: Seq
    return: seq_ac
    desc: This function shows the sum of vowel sequence.
    """
    seq_ac = []
    xn = 0
    mean = np.mean(seq)

    for xi in seq:
        xn = xi + xn - mean
        seq_ac.append(xn)
    seq_ac = np.array(seq_ac)
    return seq_ac
```

In [174]:

```
def vowel_to_sequence(c1, c2, text):
    """
    Params: c1, c2, text
    return: sequence
    We made this function to convert the vowels into sequence.
    """
    sequence = []

    for i in text:
        if i == c1 or i == c1.upper():
            sequence.append(-1)
        elif i == c2 or i == c2.upper():
            sequence.append(1)
        else:
            pass
    return sequence
```

In [175]:

```
seq = vowel_to_sequence('o', 'u', text)
seq[0:10]
```

In [176]:

```
seq_ac = sequence_to_sum(seq)
seq_ac
```

Data Visualization

This is the data visualization part for vowels seq. Here we have seen how sequenced are those vowels. It helps us to get an idea of the further process.

In [177]:

```
plt.style.use('seaborn-poster')
plt.figure(figsize = (10,8))
plt.xlabel("Elements")
plt.ylabel("Cumulative Sum")
plt.plot(seq_ac[0:1500])
plt.show()
```

Text Preprocessing

It is very imp part of the text analysis. That convert our data into the processed data. We generally not use the whole word for prediction. We have to apply lemmatize and stemmer and split to see the particular word. Those words we will use for next step. The turning point of the data.

In [178]:

```
ps = PorterStemmer()
lm = WordNetLemmatizer()

def word_text_preProcess(text, limit):
    """\
    Params: text , limit
    return: processed data
    desc: This function will take the raw data
          and convert that into capital
          and take that into the base root.
    """
    word_text = []
    for i in text[0:limit].split(" "):
        upper = i.upper()
        special_char = re.sub("[^A-ZS0-9]", "", upper)
        stemmed_txt = ps.stem(special_char)
        lemmed_txt = lm.lemmatize(stemmed_txt)
        word_text.append(lemmed_txt.upper())

    return word_text
```

In [179]:

```
# text

limit = 10000
word_text = word_text_preProcess(text, limit)
# len(word_text)
def lag_of_words(word_text):
    """
    Params: word_text
    return: str_X, str_Y
    desc: It will take the processed text for creating an X and y in string format.
          There we are taking only 3 lags at one go and predicting only one word.
    """
    str_X = []
    str_Y = []
    for i in range(3,len(word_text),1):
        first_word = word_text[i - 3]
        second_word = word_text[i - 2]
        third_word = word_text[i - 1]
        fourth_word = word_text[i]
        Str_x = str(first_word) + " " + str(second_word) + " " + str (third_word)
        Str_y = str(fourth_word)
        str_X.append(Str_x)
        str_Y.append(Str_y)
    return str_X, str_Y
```

In [180]:

```
str_X, str_Y = lag_of_words(word_text)

def show_data_XandY(str_X, str_Y, rows):
    """
    Params: str_X, str_Y, rows
    return: dataframe
    desc: That function shows the data which is going to predict that will matching fine
```

```
or not.
        You can do coparision through this DF.
    """
    Q = pd.DataFrame([str_X, str_Y]).T
    Q. columns = ['Xdata', "Ypred"]
    return Q.head(rows)
```

DataFrame Contains X and Y

Here you will see DataFrame of X and Y. This will help us to compare our end prediction. To get the end prediction it will show the first three words in X data and fourth is predicted word contains Ypredcol.

In [181]:

```
show_data_XandY(str_X, str_Y, 20)
```

We are looking for unique words of text. That text will help you to decide your input shape.

In [182]:

```
unique_words = unique(word_text)
len(unique_words)
```

In [183]:

```
len(str_X)
```

Text to int or bool form

Now in this step our data will convert into an array or bool. For example where your desired word is present there it will show true or 1 and in rest of the places will be false or 0.

In [184]:

```
X_arr = np.zeros((len(str_X), len(word_text), 3))
# X_arr = np.zeros((len(str_X), len(word_text), 3), dtypes = bool)
Y_arr = np.zeros((len(str_X), len(word_text)))
# Y_arr = np.zeros((len(str_X), len(word_text)), dtypes = bool)
```

In [185]:

```
def positionOfWordFinder(unique_words,X_arr):
    """
    Params: unique_words, X_arr
    return: word position
    desc: There you will see when you want to know about the
          position of the word in lakhs of text you can use
          this function.
    """
    word_pos_finder = {}
    for i in range(0, len(unique_words), 1):
        word_pos_finder[unique_words[i]] = i
    for i, j in enumerate(str_X):
    #     if i == 5:
    #         print("index", i,'line', j)
    #         print("index", i,'line', j.split(" "))
    #         print("index", i,'line', list(enumerate(j.split(" "))))
    #           break
        for j, k in enumerate(j.split(" ")):
    #           if i == 5:
    #               print(i, j, k)
    #               break
            pos = word_pos_finder[k]
            X_arr[i,pos, j]=  1

    return word_pos_finder
```

In [186]:

```
word_pos_finder = positionOfWordFinder(unique_words, X_arr)

word_pos_finder['GUTENBERG']
```

In [187]:

```
# X_arr
```

> ## Processing TExt of Y

In [188]:

```
for i, j in enumerate(str_Y):
    pos = word_pos_finder[j]
    Y_arr[i, pos] = 1
```

In [189]:

```
Y_arr
```

In [190]:

```
len(word_text)
```

In [191]:

```
len(unique_words)
```

> ## LSTM Model

Using the lstm model and also have used the embedding but didnt not get very much difference. So if it works for you. You definitely can add.

In [192]:

```
model = Sequential()
# model.add(Embedding(len(word_text), 50, input_length=50))

# nn.add(LSTM(100, return_sequences = True, input_shape =(len(word_text), len(unique_wor
ds))))
model.add(LSTM(100, return_sequences = True, input_shape = (len(unique_words), 3)))
model.add(LSTM(100))
# model.add(LSTM(300))
model.add(Dropout(0.15))
model.add(Dense(len(unique_words)))
model.add(Dense(len(unique_words)))
model.add(Activation('relu'))
model.add(Dense(len(word_text), activation = "softmax"))
```

In [193]:

```
model.summary()
```

In [194]:

```
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ['accurac
y'])
model.fit(X_arr, Y_arr, batch_size=100, epochs = 50)
```

Prediction

It takes the 3 word bcz we have defined that into the lag function. So here y prediction will be after 3 word.

In [195]:

```
input_X_arr = np.zeros((1,len(unique_words),3), dtype = int)
sent = input("Enter a a sentence of [3 words]")

sent = sent.upper()
try:
    for i,j in enumerate(sent.split(" ")):
        pos = word_pos_finder[j]
        input_X_arr[i, pos, j] = 1
except IndexError:
#     print("IndexError But that value had been saved in input array")
    print(" ")
R = pd.DataFrame([unique_words, model.predict([input_X_arr])[0]]).T
R.columns = ["Word", "Pred"]
print(sent,R.sort_values(by= "Pred", ascending = False)['Word'].values[0])
```