In [2]:

```
#import the necessary libraries

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense , Bidirectional , LSTM , Embedding
from keras.utils.data_utils import pad_sequences
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.text import Tokenizer
import pickle
import re
import string as str1
```

In [3]:

```
#get the data
#Next,let's explore our data.

data = pd.read_csv("medium_data.csv")
data.head()
```

Out[3]:

	id	url	title	subtitle	image	claps	responses	reading_time	publication	date
0	1	https://towardsdatascience.com/a-beginners-gui...	A Beginner's Guide to Word Embedding with Gens...	NaN	1.png	850	8	8	Towards Data Science	2019-05-30
1	2	https://towardsdatascience.com/hands-on-graph-...	Hands-on Graph Neural Networks with PyTorch & ...	NaN	2.png	1100	11	9	Towards Data Science	2019-05-30
2	3	https://towardsdatascience.com/how-to-use-ggpl...	How to Use ggplot2 in Python	A Grammar of Graphics for Python	3.png	767	1	5	Towards Data Science	2019-05-30
3	4	https://towardsdatascience.com/databricks-how-...	Databricks: How to Save Files in CSV on Your L...	When I work on Python projects dealing…	4.jpeg	354	0	4	Towards Data Science	2019-05-30
4	5	https://towardsdatascience.com/a-step-by-step-...	A Step-by-Step Implementation of Gradient Desc...	One example of building neural…	5.jpeg	211	3	4	Towards Data Science	2019-05-30

In [4]:

```
#By analyzing the text data present in data['title'] column , i found that there are some special characters like ['“','”','\xa0', '\u200a

data['title']=data['title'].apply(lambda x : x.replace("“" , "") ).apply(lambda x : x.replace("”" , ""))
data['title'] = data['title'].apply(lambda x: x.replace(u'\xa0',u' '))
data['title'] = data['title'].apply(lambda x: x.replace('\u200a',' '))
```

In [5]:

```
#Let's have a look at our title column again.
data['title']
```

Out[5]:

```
0       A Beginner’s Guide to Word Embedding with Gens...
1       Hands-on Graph Neural Networks with PyTorch & ...
2                            How to Use ggplot2 in Python
3       Databricks: How to Save Files in CSV on Your L...
4       A Step-by-Step Implementation of Gradient Desc...
                              ...
6503    We vs I — How Should You Talk About Yourself o...
6504                  How Donald Trump Markets Himself
6505        Content and Marketing Beyond Mass Consumption
6506    5 Questions All Copywriters Should Ask Clients...
6507             How To Write a Good Business Blog Post
Name: title, Length: 6508, dtype: object
```

In [ ]:

```
#it looks fine
```

In [ ]:

```
#so next we are going to tokenize eacch word, which means we will provide a no. for each unique word.
#this is necessary as machines don't uderstand charcters.
```

In [8]:

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data['title'])
wordindex = tokenizer.word_index
totalwords = len(wordindex)+1
totalwords
```

Out[8]:

```
8099
```

In [ ]:

```
#ok, now we have got total unique words in our title column and we have provided each word with a token which is nothing but integer
```

In [26]:

```
#next we will convert the complete titles in sequences of the tokens which we have given to each word above.
#We will convert the sentences into list of tokens in which each tken represents a word
#for e.g "How to Use ggplot2 in Python " --> [20, 522, 1031, 4432, 12, 2342]. this just an e.g

#then in the 2nd loop we are creating the dataset for our model.
#for e.g x1=[20,522] , y1=[1031] ; x2=[20,522,1031] , y2=[4432] ; x3=[20,522,1031,4432] , y3=[12]
```

In [31]:

```
sequence=[]
input_sequence=[]
for line in data['title']:
    #print([line])
    seq= tokenizer.texts_to_sequences([line])[0]
    sequence.append(seq)
    for i in range(1 , len(seq)):
        x = seq[:i+1]
        input_sequence.append(x)
```

In [27]:

```
#let's have a look what our title coulumn looks when converted into sequence

sequence
```

```
  49,
  829,
  3662,
  3663,
  3664,
  106],
 [121, 4, 3665, 564, 32, 267],
 [104, 4, 3666, 1869, 3667],
 [6, 1, 565, 175],
 [1089, 2466, 11, 206, 193],
 [6, 1, 679, 2467, 566],
 [246, 33, 3668, 11, 16, 1870],
 [3669, 2468, 53, 2469, 2470, 7, 4, 2469, 744],
 [16, 47, 329, 314],
 [830, 461, 11, 831, 1871],
 [3670,
  1872,
  745,
  77,
  680,
```

In [29]:

```
#this is how next word prediction models are trained, now we just need to pad these sequences to make them of same length.
input_sequence
```

Out[29]:

```
[[4, 678],
 [4, 678, 67],
 [4, 678, 67, 1],
 [4, 678, 67, 1, 460],
 [4, 678, 67, 1, 460, 1522],
 [4, 678, 67, 1, 460, 1522, 13],
 [4, 678, 67, 1, 460, 1522, 13, 2463],
 [4, 678, 67, 1, 460, 1522, 13, 2463, 3657],
 [4, 678, 67, 1, 460, 1522, 13, 2463, 3657, 98],
 [1865, 22],
 [1865, 22, 742],
 [1865, 22, 742, 81],
 [1865, 22, 742, 81, 102],
 [1865, 22, 742, 81, 102, 13],
 [1865, 22, 742, 81, 102, 13, 345],
 [1865, 22, 742, 81, 102, 13, 345, 345],
 [1865, 22, 742, 81, 102, 13, 345, 345, 1866],
 [6, 1],
```

In [12]:

```
#As the Dense Layer accepts inputs of fixed length we have to convert all the title sequences to same length.
#We will do this by padding all the titles with zeros to make them of same length.
#for this we need to know the what is the max length of title in data['title'] column
#Bcz we are going to make all sentences of that length only , as we can increase the lengths of other titles but can't reduce them.

len(input_sequence)
maxlen = max(len(i) for i in input_sequence)
```

In [13]:

```
#we are using the pad_sequences library of keras in which we have provided padding = 'pre'.
#which means that we are adding zeros in left of the sequences and padding='post' means adding zeros to the right side.

X = np.array(pad_sequences(input_sequence , maxlen , padding='pre' ))
```

In [14]:

```
#Let's have a look at our padded sequnces

X
```

Out[14]:

```
array([[  0,   0,   0, ...,   0,   4, 678],
       [  0,   0,   0, ...,   4, 678,  67],
       [  0,   0,   0, ..., 678,  67,   1],
       ...,
       [  0,   0,   0, ...,   4,  79,  55],
       [  0,   0,   0, ...,  79,  55, 731],
       [  0,   0,   0, ...,  55, 731, 554]])
```

In [17]:

```
#you know what we are going to do now?
#we will extract input and output arrays from X.
#our input will be X[:,:-1] which means all the rows but just excluding the last column and we will tae that last column as output.

inputseq=np.array(input_sequence)
xs = X[:,:-1]
labels = X[:,-1]
```

```
C:\Users\Ankita Sharma\AppData\Local\Temp\ipykernel_5576\3831159186.py:1: VisibleDeprecationWarning: Creating an ndarray fr
om ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is de
precated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  inputseq=np.array(input_sequence)
```

In [18]:

```
#Let's have a look at our input array

xs
```

Out[18]:

```
array([[  0,   0,   0, ...,   0,   0,   4],
       [  0,   0,   0, ...,   0,   4, 678],
       [  0,   0,   0, ...,   4, 678,  67],
       ...,
       [  0,   0,   0, ...,  65,   4,  79],
       [  0,   0,   0, ...,   4,  79,  55],
       [  0,   0,   0, ...,  79,  55, 731]])
```

In [19]:

```
#now look at our labels.

labels
```

Out[19]:

```
array([678,  67,   1, ...,  55, 731, 554])
```

In [ ]:

```
#wait.
#don't you think there is something incomplete.
#oh yess , we haven't converted our labels into one hot encodings.
#so , now we will convert our labels into one hot encodings using the to_categoical function of keras.
```

In [20]:

```
ys = to_categorical(labels)
```

In [21]:

```
ys.shape
```

Out[21]:

```
(48448, 8099)
```

In [22]:

```
xs.shape
```

Out[22]:

```
(48448, 39)
```

In [23]:

```
totalwords
```

Out[23]:

```
8099
```

In [24]:

```
#let's prepare the model architecture.
#we will use embedding layer for dimensionality reduction and text undersanding.
#we will convert each word into vector of 100 dimension using embedding layer. this will provide the model an understaniding of the contex
#Then we arre using Bidirectional LSTM Layer which is followed by Dense Layer which comprises of Softmax Activation.


model = Sequential()
model.add(Embedding(input_dim = totalwords , output_dim=100 , input_length=maxlen-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(totalwords , activation='Softmax'))
model.compile(optimizer='adam' , loss='categorical_crossentropy' , metrics=['accuracy'])
```

In [25]:

```
#finally fitting the data in our model and training it.
#Let's go.

model.fit(xs , ys , epochs=40)
```

```
Epoch 1/40
1514/1514 [==============================] - 145s 91ms/step - loss: 6.9837 - accuracy: 0.0836
Epoch 2/40
1514/1514 [==============================] - 126s 83ms/step - loss: 6.1442 - accuracy: 0.1404
Epoch 3/40
1514/1514 [==============================] - 124s 82ms/step - loss: 5.6337 - accuracy: 0.1659
Epoch 4/40
1514/1514 [==============================] - 118s 78ms/step - loss: 5.1621 - accuracy: 0.1906
Epoch 5/40
1514/1514 [==============================] - 125s 83ms/step - loss: 4.7118 - accuracy: 0.2166
Epoch 6/40
1514/1514 [==============================] - 117s 77ms/step - loss: 4.2854 - accuracy: 0.2469
Epoch 7/40
1514/1514 [==============================] - 121s 80ms/step - loss: 3.8881 - accuracy: 0.2877
Epoch 8/40
1514/1514 [==============================] - 119s 79ms/step - loss: 3.5142 - accuracy: 0.3392
Epoch 9/40
1514/1514 [==============================] - 118s 78ms/step - loss: 3.1706 - accuracy: 0.3921
Epoch 10/40
1514/1514 [==============================] - 121s 80ms/step - loss: 2.8560 - accuracy: 0.4436
Epoch 11/40
1514/1514 [==============================] - 124s 82ms/step - loss: 2.5738 - accuracy: 0.4923
Epoch 12/40
1514/1514 [==============================] - 123s 81ms/step - loss: 2.3247 - accuracy: 0.5384
Epoch 13/40
1514/1514 [==============================] - 126s 83ms/step - loss: 2.1063 - accuracy: 0.5787
Epoch 14/40
1514/1514 [==============================] - 120s 79ms/step - loss: 1.9094 - accuracy: 0.6170
Epoch 15/40
1514/1514 [==============================] - 126s 83ms/step - loss: 1.7339 - accuracy: 0.6527
Epoch 16/40
1514/1514 [==============================] - 123s 81ms/step - loss: 1.5800 - accuracy: 0.6844
Epoch 17/40
1514/1514 [==============================] - 117s 77ms/step - loss: 1.4456 - accuracy: 0.7096
Epoch 18/40
1514/1514 [==============================] - 117s 77ms/step - loss: 1.3264 - accuracy: 0.7338
Epoch 19/40
1514/1514 [==============================] - 123s 82ms/step - loss: 1.2230 - accuracy: 0.7544
Epoch 20/40
1514/1514 [==============================] - 143s 94ms/step - loss: 1.1323 - accuracy: 0.7712
Epoch 21/40
1514/1514 [==============================] - 140s 93ms/step - loss: 1.0527 - accuracy: 0.7869
Epoch 22/40
1514/1514 [==============================] - 124s 82ms/step - loss: 0.9846 - accuracy: 0.8002
Epoch 23/40
1514/1514 [==============================] - 120s 79ms/step - loss: 0.9269 - accuracy: 0.8102
Epoch 24/40
1514/1514 [==============================] - 119s 79ms/step - loss: 0.8738 - accuracy: 0.8205
Epoch 25/40
1514/1514 [==============================] - 115s 76ms/step - loss: 0.8334 - accuracy: 0.8255
Epoch 26/40
1514/1514 [==============================] - 119s 79ms/step - loss: 0.7970 - accuracy: 0.8315
Epoch 27/40
1514/1514 [==============================] - 119s 79ms/step - loss: 0.7630 - accuracy: 0.8369
Epoch 28/40
1514/1514 [==============================] - 120s 79ms/step - loss: 0.7350 - accuracy: 0.8421
Epoch 29/40
1514/1514 [==============================] - 119s 79ms/step - loss: 0.7124 - accuracy: 0.8459
Epoch 30/40
1514/1514 [==============================] - 124s 82ms/step - loss: 0.6925 - accuracy: 0.8470
Epoch 31/40
1514/1514 [==============================] - 125s 83ms/step - loss: 0.6752 - accuracy: 0.8489
Epoch 32/40
1514/1514 [==============================] - 124s 82ms/step - loss: 0.6617 - accuracy: 0.8506
Epoch 33/40
1514/1514 [==============================] - 122s 81ms/step - loss: 0.6497 - accuracy: 0.8516
Epoch 34/40
1514/1514 [==============================] - 123s 81ms/step - loss: 0.6390 - accuracy: 0.8524
Epoch 35/40
1514/1514 [==============================] - 123s 81ms/step - loss: 0.6301 - accuracy: 0.8528
Epoch 36/40
1514/1514 [==============================] - 124s 82ms/step - loss: 0.6209 - accuracy: 0.8532
Epoch 37/40
1514/1514 [==============================] - 663s 438ms/step - loss: 0.6157 - accuracy: 0.8542
Epoch 38/40
1514/1514 [==============================] - 117s 77ms/step - loss: 0.6103 - accuracy: 0.8537
Epoch 39/40
1514/1514 [==============================] - 118s 78ms/step - loss: 0.6042 - accuracy: 0.8543
Epoch 40/40
1514/1514 [==============================] - 117s 77ms/step - loss: 0.5995 - accuracy: 0.8550
```

Out[25]:

```
<keras.callbacks.History at 0x28799736be0>
```

In [ ]:

In [ ]:

In [ ]: