In [1]:
```
import numpy as np
import pandas as pd
import re
from bs4 import BeautifulSoup
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate,
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import warnings
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")
```

In [2]:
```
data1=pd.read_csv("Reviews.csv",nrows=100000)
```

In [3]:
```
data1
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulne
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	

In [4]:
```
data=data1.head(100)
```

In [5]:
```
data
```

Out[5]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	
...	...	...	...	...	...	
95	96	B0019CW0HE	A1BFNM27629VAV	E. Triebe	0	
96	97	B0019CW0HE	A18AAABCIJKC5Q	Rhiever	0	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
97	98	B0019CW0HE	A3UII2114114PI	FuNky Faja "SiLkk"	0	
98	99	B0019CW0HE	ABZ9F0D94YK45	Amazon-tron 3000	0	
99	100	B0019CW0HE	A2P6ACFZ8FTNVV	Melissa Benjamin	0	

100 rows × 10 columns

In [ ]:

In [6]:
```
data.drop_duplicates(subset=['Text'],inplace=True)  #dropping duplicates
data.dropna(axis=0,inplace=True)
#dropping na
```

In [7]:
```
contraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot",
                       "didn't": "did not", "doesn't": "does not", "don't":
                       "he'd": "he would","he'll": "he will", "he's": "he is
                       "I'd": "I would", "I'd've": "I would have", "I'll": "
                       "i'd've": "i would have", "i'll": "i will",  "i'll've
                       "it'd've": "it would have", "it'll": "it will", "it'l
                       "mayn't": "may not", "might've": "might have","mightn
                       "mustn't": "must not", "mustn't've": "must not have",
                       "oughtn't": "ought not", "oughtn't've": "ought not ha
                       "she'd": "she would", "she'd've": "she would have", "
                       "should've": "should have", "shouldn't": "should not"
                       "this's": "this is","that'd": "that would", "that'd've
                       "there'd've": "there would have", "there's": "there i
                       "they'll": "they will", "they'll've": "they will have
                       "wasn't": "was not", "we'd": "we would", "we'd've": "
                       "we've": "we have", "weren't": "were not", "what'll":
                       "what's": "what is", "what've": "what have", "when's"
                       "where've": "where have", "who'll": "who will", "who'
                       "why's": "why is", "why've": "why have", "will've": "
                       "would've": "would have", "wouldn't": "would not", "w
                       "y'all'd": "you all would","y'all'd've": "you all woul
                       "you'd": "you would", "you'd've": "you would have", "
                       "you're": "you are", "you've": "you have"}
```

In [8]:
```
data['Text'][:10]
```

Out[8]:
```
0    I have bought several of the Vitality canned dog food products and have fo
und them all to be of good quality. The product looks more like a stew than a p
rocessed meat and it smells better. My Labr...
1             Product arrived labeled as Jumbo Salted Peanuts...the peanuts wer
e actually small sized unsalted. Not sure if this was an error or if the vendor
intended to represent the product as "Jumbo".
2    This is a confection that has been around a few centuries.  It is a light,
pillowy citrus gelatin with nuts - in this case Filberts. And it is cut into ti
ny squares and then liberally coated with ...
3    If you are looking for the secret ingredient in Robitussin I believe I hav
e found it.  I got this in addition to the Root Beer Extract I ordered (which w
as good) and made some cherry soda.  The fl...
4                                                                  Great taffy at
a great price.  There was a wide assortment of yummy taffy.  Delivery was very
quick.  If your a taffy lover, this is a deal.
5    I got a wild hair for taffy and ordered this five pound bag. The taffy was
all very enjoyable with many flavors: watermelon, root beer, melon, peppermint,
grape, etc. My only complaint is there wa...
6    This saltwater taffy had great flavors and was very soft and chewy.  Each
candy was individually wrapped well.  None of the candies were stuck together,
which did happen in the expensive version, ...
7                                                                  This taffy is s
o good.  It is very soft and chewy.  The flavors are amazing.  I would definite
ly recommend you buying it.  Very satisfying!!
8                                                                        Right
now I'm mostly just sprouting this so my cats can eat the grass. They love it.
I rotate it around with Wheatgrass and Rye too
9                                                                    This is a ve
ry healthy dog food. Good for their digestion. Also good for small puppies. My
dog eats her required amount at every feeding.
Name: Text, dtype: object
```

In [9]:
```
stop_words = set(stopwords.words('english'))
def text_cleaner(text):
    newString = text.lower()
    newString = BeautifulSoup(newString, "lxml").text
    newString = re.sub(r'\([^)]*\)', '', newString)
    newString = re.sub('"','', newString)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    tokens = [w for w in newString.split() if not w in stop_words]
    long_words=[]
    for i in tokens:
        if len(i)>=3:                  #removing short word
            long_words.append(i)
    return (" ".join(long_words)).strip()

cleaned_text = []
for t in data['Text']:
    cleaned_text.append(text_cleaner(t))
```

In [10]:
```
data['Summary'][:10]
```

Out[10]:
```
0                             Good Quality Dog Food
1                                 Not as Advertised
2                           "Delight" says it all
3                                  Cough Medicine
4                                     Great taffy
5                                      Nice Taffy
6    Great!  Just as good as the expensive brands!
7                          Wonderful, tasty taffy
8                                      Yay Barley
9                                Healthy Dog Food
Name: Summary, dtype: object
```

In [11]:
```
def summary_cleaner(text):
    newString = re.sub('"','', text)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    newString = newString.lower()
    tokens=newString.split()
    newString=''
    for i in tokens:
        if len(i)>1:
            newString=newString+i+' '
    return newString

#Call the above function
cleaned_summary = []
for t in data['Summary']:
    cleaned_summary.append(summary_cleaner(t))

data['cleaned_text']=cleaned_text
data['cleaned_summary']=cleaned_summary
data['cleaned_summary'].replace('', np.nan, inplace=True)
data.dropna(axis=0,inplace=True)
```

In [12]:
```
data['cleaned_summary'] = data['cleaned_summary'].apply(lambda x : '_START_ '+ x
```

In [13]:
```
for i in range(5):
    print("Review:",data['cleaned_text'][i])
    print("Summary:",data['cleaned_summary'][i])
    print("\n")
```

```
Review: bought several vitality canned dog food products found good quality pro
duct looks like stew processed meat smells better labrador finicky appreciates
product better
Summary: _START_ good quality dog food  _END_


Review: product arrived labeled jumbo salted peanuts peanuts actually small siz
ed unsalted sure error vendor intended represent product jumbo
Summary: _START_ not as advertised  _END_


Review: confection around centuries light pillowy citrus gelatin nuts case filb
erts cut tiny squares liberally coated powdered sugar tiny mouthful heaven chew
y flavorful highly recommend yummy treat familiar story lewis lion witch wardro
be treat seduces edmund selling brother sisters witch
Summary: _START_ delight says it all  _END_


Review: looking secret ingredient robitussin believe found got addition root be
er extract ordered made cherry soda flavor medicinal
Summary: _START_ cough medicine  _END_


Review: great taffy great price wide assortment yummy taffy delivery quick taff
y lover deal
Summary: _START_ great taffy  _END_
```

In [14]:
```
import matplotlib.pyplot as plt
text_word_count = []
summary_word_count = []

# populate the lists with sentence lengths
for i in data['cleaned_text']:
      text_word_count.append(len(i.split()))

for i in data['cleaned_summary']:
      summary_word_count.append(len(i.split()))

length_df = pd.DataFrame({'text':text_word_count, 'summary':summary_word_count})
length_df.hist(bins = 30)
plt.show()
```

```
<Figure size 640x480 with 2 Axes>
```

In [15]:
```
max_len_text=80
max_len_summary=10
```

In [16]:
```
from sklearn.model_selection import train_test_split
x_tr,x_val,y_tr,y_val=train_test_split(data['cleaned_text'],data['cleaned_summary
```

In [17]:
```
#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_tr))

#convert text sequences into integer sequences
x_tr    =   x_tokenizer.texts_to_sequences(x_tr)
x_val   =   x_tokenizer.texts_to_sequences(x_val)

#padding zero upto maximum length
x_tr    =   pad_sequences(x_tr,  maxlen=max_len_text, padding='post')
x_val   =   pad_sequences(x_val, maxlen=max_len_text, padding='post')

x_voc_size   =  len(x_tokenizer.word_index) +1
```

In [18]:
```
#preparing a tokenizer for summary on training data
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_tr))

#convert summary sequences into integer sequences
y_tr    =   y_tokenizer.texts_to_sequences(y_tr)
y_val   =   y_tokenizer.texts_to_sequences(y_val)

#padding zero upto maximum length
y_tr    =   pad_sequences(y_tr, maxlen=max_len_summary, padding='post')
y_val   =   pad_sequences(y_val, maxlen=max_len_summary, padding='post')

y_voc_size  =   len(y_tokenizer.word_index) +1
```

In [19]:
```
from attention import AttentionLayer
```

In [20]:
```
from tensorflow.keras import backend as K
K.clear_session()
latent_dim = 500

# Encoder
encoder_inputs = Input(shape=(max_len_text,))
enc_emb = Embedding(x_voc_size, latent_dim,trainable=True)(encoder_inputs)

#LSTM 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#LSTM 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#LSTM 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# Set up the decoder.
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(y_voc_size, latent_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

#LSTM using encoder_states as initial state
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,ini

#Attention Layer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

# Concat attention output and decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_output

#Dense layer
decoder_dense = TimeDistributed(Dense(y_voc_size, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.summary()
```

```
Model: "model"
__________________________________________________________________________________
_________________
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================
==================
input_1 (InputLayer)         [(None, 80)]              0
__________________________________________________________________________________
_________________
embedding (Embedding)        (None, 80, 500)           624000      input_1[0][0]
__________________________________________________________________________________
_________________
```

```
lstm (LSTM)                     [(None, 80, 500), (N 2002000     embedding[0]
[0]
__________________________________________________________________________________
__________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________
__________________
lstm_1 (LSTM)                   [(None, 80, 500), (N 2002000     lstm[0][0]
__________________________________________________________________________________
__________________
embedding_1 (Embedding)         (None, None, 500)    99500       input_2[0][0]
__________________________________________________________________________________
__________________
lstm_2 (LSTM)                   [(None, 80, 500), (N 2002000     lstm_1[0][0]
__________________________________________________________________________________
__________________
lstm_3 (LSTM)                   [(None, None, 500),  2002000     embedding_1[0]
[0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________
__________________
attention_layer (AttentionLayer ((None, None, 500),  500500      lstm_2[0][0]
                                                                 lstm_3[0][0]
__________________________________________________________________________________
__________________
concat_layer (Concatenate)      (None, None, 1000)   0           lstm_3[0][0]
                                                                 attention_laye
r[0][0]
__________________________________________________________________________________
__________________
time_distributed (TimeDistribut (None, None, 199)    199199      concat_layer
[0][0]
==================================================================================
==================
Total params: 9,431,199
Trainable params: 9,431,199
Non-trainable params: 0
__________________________________________________________________________________
__________________
```

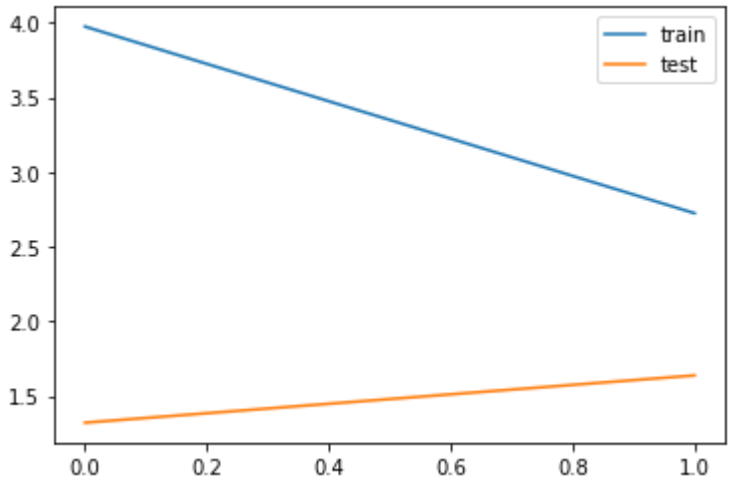
In [21]:
```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')
```

In [22]:
```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)
```

In [23]:
```
history=model.fit([x_tr,y_tr[:,:-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1],
```

```
Train on 89 samples, validate on 10 samples
Epoch 1/5
89/89 [==============================] - 179s 2s/sample - loss: 3.9744 - val_lo
ss: 1.3198
Epoch 2/5
89/89 [==============================] - 320s 4s/sample - loss: 2.7234 - val_lo
ss: 1.6364
Epoch 00002: early stopping
```

In [24]:
```
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



In [25]:
```
reverse_target_word_index=y_tokenizer.index_word
reverse_source_word_index=x_tokenizer.index_word
target_word_index=y_tokenizer.word_index
```

In [26]:
```
# encoder inference
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, s

# decoder inference
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_len_text,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)

# To predict the next word in the sequence, set the initial states to the states
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[dec

#attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input, decoder_
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
[decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_st
[decoder_outputs2] + [state_h2, state_c2])
```

In [27]:
```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Chose the 'start' word as the first word of the target sequence
    target_seq[0, 0] = target_word_index['start']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, c

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='end'):
            decoded_sentence += ' '+sampled_token

            # Exit condition: either hit max length or find stop word.
            if (sampled_token == 'end' or len(decoded_sentence.split()) >= (max_

                stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

In [28]:
```
def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if((i!=0 and i!=target_word_index['start']) and i!=target_word_index['end
            newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=''
    for i in input_seq:
        if(i!=0):
            newString=newString+reverse_source_word_index[i]+' '
    return newString
```

In [29]:
```
for i in range(len(x_val)):
    print("Review:",seq2text(x_val[i]))
    print("Original summary:",seq2summary(y_val[i]))
    #print("Predicted summary:",decode_sequence(x_val[i].reshape(1,max_len_text),
    print("\n")
```

```
Review: candy red flavor chewy would never buy
Original summary: no flavor


Review: mix vet recommended limited ingredient food really helped symptoms like
s always buy amazon cheaper free shipping
Original summary: great for stomach


Review: around nuts case cut tiny powdered sugar tiny chewy highly recommend yu
mmy treat familiar treat selling
Original summary: it


Review: deal awesome arrived halloween enough love quality product much less ex
pensive local store candy
Original summary: deal


Review: looked like perfect mix unfortunately arrived melted chocolate days roo
m still fridge breaking ever since taste good chocolate order online see store
would
Original summary: good


Review: golden retriever one dogs ever various food found loves natural balance
really like natural balance fact flavors dry wet varieties mix dry food little
wet food golden loves like mixing flavors time think meal day day might get lit
tle away type though smells started purchasing amazon wet food box couple came
home realized could save time bought dog food buy amazon definitely recommend g
ive natural balance dog food never eaten dog seems love
Original summary: great dog food


Review: love eating good looking sweet like fresh take time eating
Original summary: taste


Review: would way buy
Original summary:


Review: roast wonderful roasted beans taste delicious coffee smooth aftertaste
roasted beans home friends like much
Original summary: our love it


Review: natural balance dog food dogs dog foods past someone recommend natural
balance free since allergic since also different size sized dogs
```

```
Original summary: good healthy dog food
```

In [ ]: