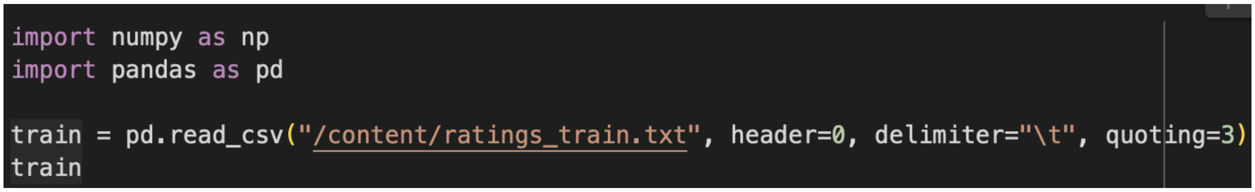
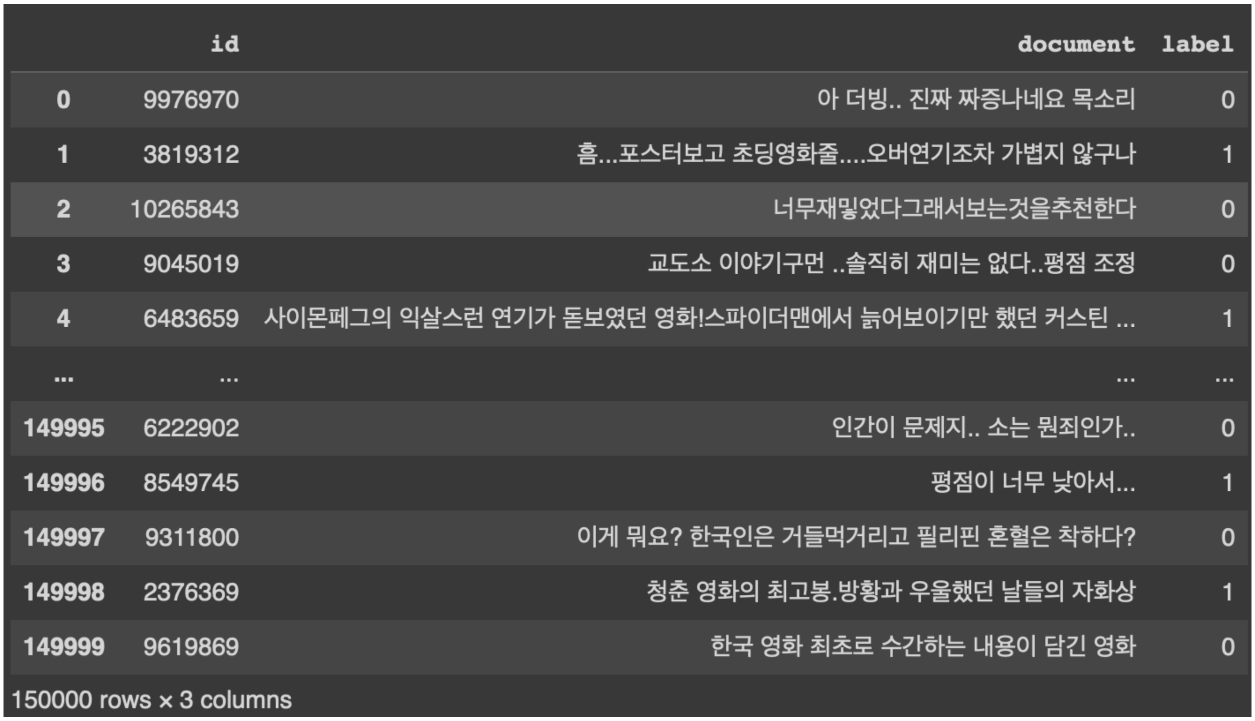
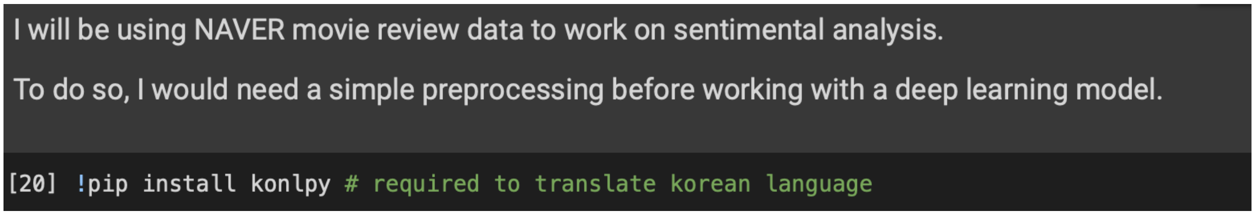


```
import numpy as np
import pandas as pd

train = pd.read_csv("/content/ratings_train.txt", header=0, delimiter="\t", quoting=3)
train
```



	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밓었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1
...	...	...	...
149995	6222902	인간이 문제지.. 소는 뭔죄인가..	0
149996	8549745	평점이 너무 낮아서...	1
149997	9311800	이게 뭐요? 한국인은 거들먹거리고 필리핀 혼혈은 착하다?	0
149998	2376369	청춘 영화의 최고봉.방황과 우울했던 날들의 자화상	1
149999	9619869	한국 영화 최초로 수간하는 내용이 담긴 영화	0

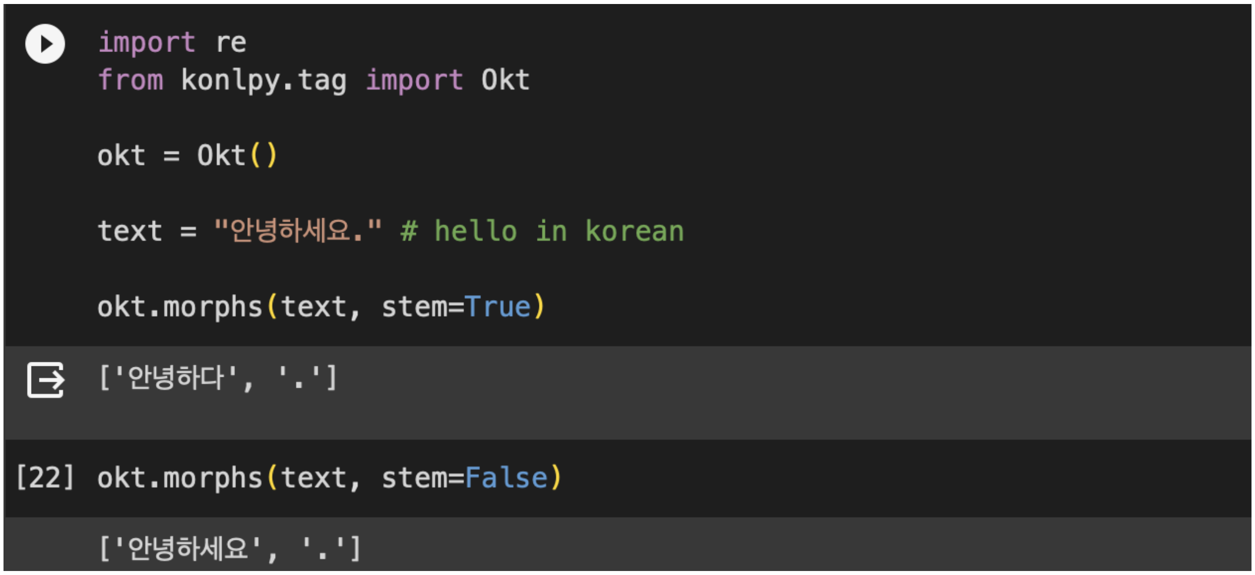
150000 rows × 3 columns



I will be using NAVER movie review data to work on sentimental analysis.

To do so, I would need a simple preprocessing before working with a deep learning model.

```
[20] !pip install konlpy # required to translate korean language
```

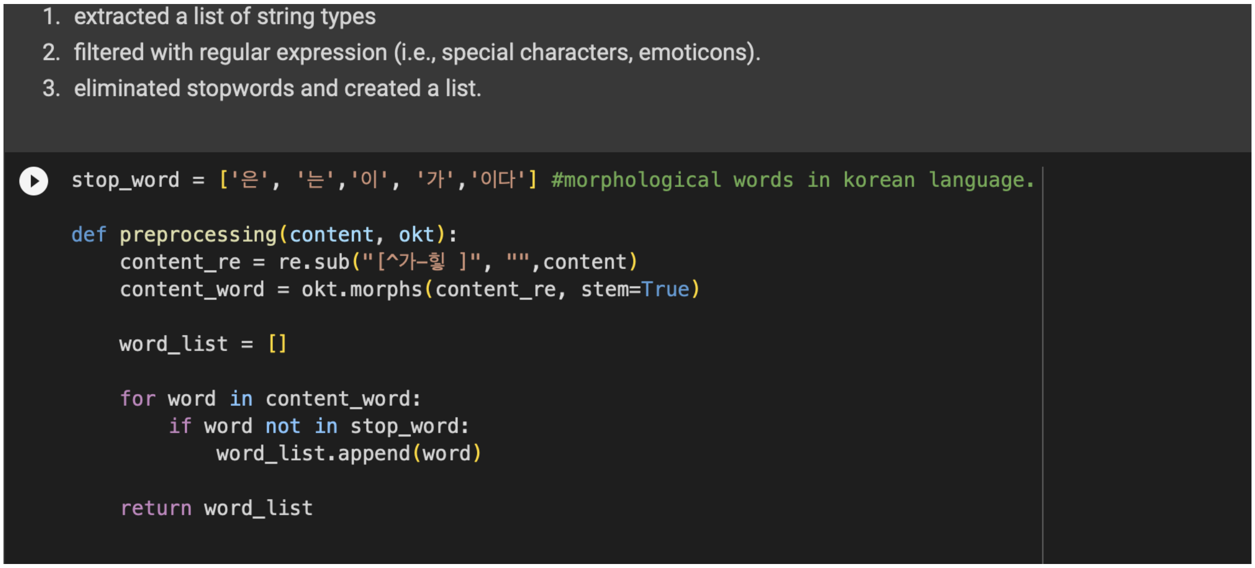


```
import re
from konlpy.tag import Okt

okt = Okt()

text = "안녕하세요." # hello in korean

okt.morphs(text, stem=True)
```

```
['안녕하다', '.']
```

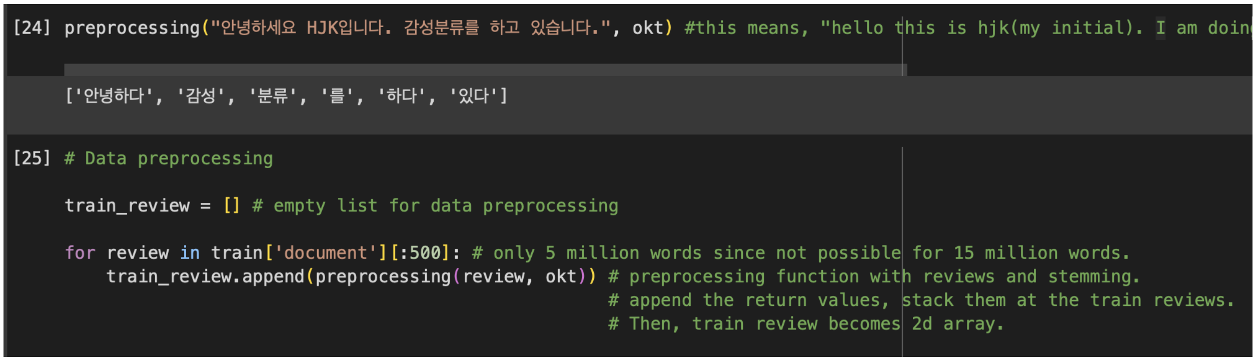
```
[22] okt.morphs(text, stem=False)
```

```
['안녕하세요', '.']
```



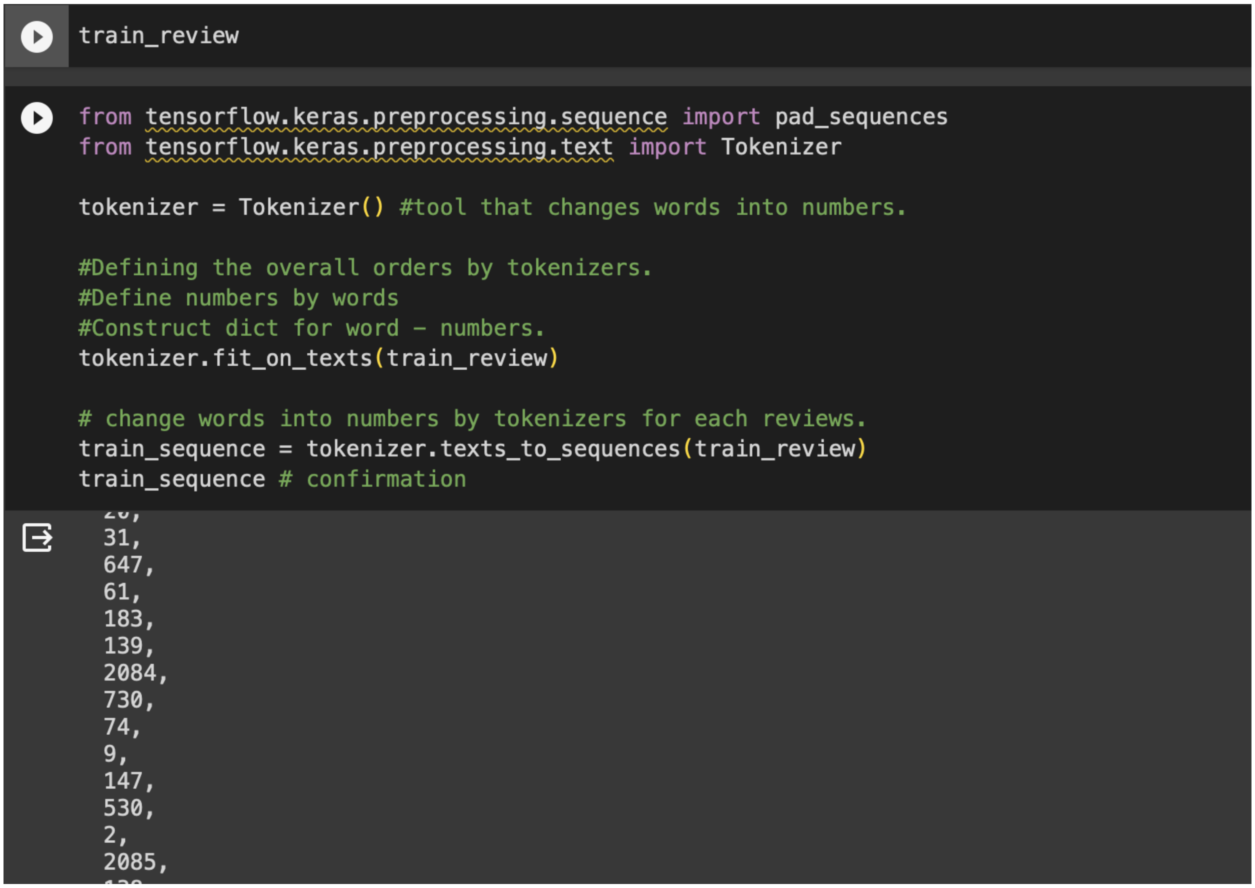
1. extracted a list of string types
2. filtered with regular expression (i.e., special characters, emoticons).
3. eliminated stopwords and created a list.

```
stop_word = ['은', '는','이', '가','이다'] #morphological words in korean language.

def preprocessing(content, okt):
    content_re = re.sub("[^가-힣 ]", "",content)
    content_word = okt.morphs(content_re, stem=True)

    word_list = []

    for word in content_word:
        if word not in stop_word:
            word_list.append(word)

    return word_list
```



```
[24] preprocessing("안녕하세요 HJK입니다. 감성분류를 하고 있습니다.", okt) #this means, "hello this is hjk(my initial). I am doin

     ['안녕하다', '감성', '분류', '를', '하다', '있다']
```

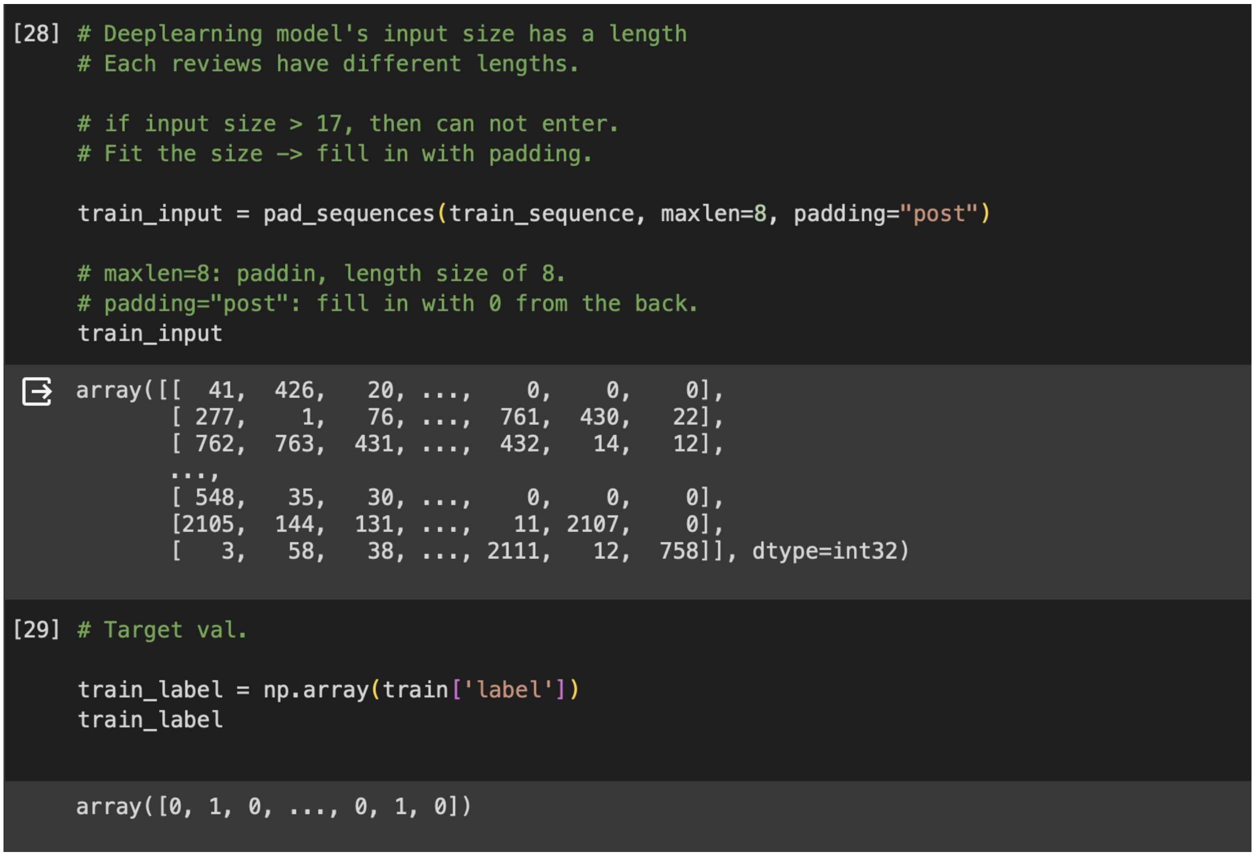
```
[25] # Data preprocessing

     train_review = [] # empty list for data preprocessing

     for review in train['document'][:500]: # only 5 million words since not possible for 15 million words.
         train_review.append(preprocessing(review, okt)) # preprocessing function with reviews and stemming.
                                                         # append the return values, stack them at the train reviews.
                                                         # Then, train review becomes 2d array.
```



```
train_review
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer() #tool that changes words into numbers.

#Defining the overall orders by tokenizers.
#Define numbers by words
#Construct dict for word - numbers.
tokenizer.fit_on_texts(train_review)

# change words into numbers by tokenizers for each reviews.
train_sequence = tokenizer.texts_to_sequences(train_review)
train_sequence # confirmation
```

```
  31,
  647,
  61,
  183,
  139,
  2084,
  730,
  74,
  9,
  147,
  530,
  2,
  2085,
```

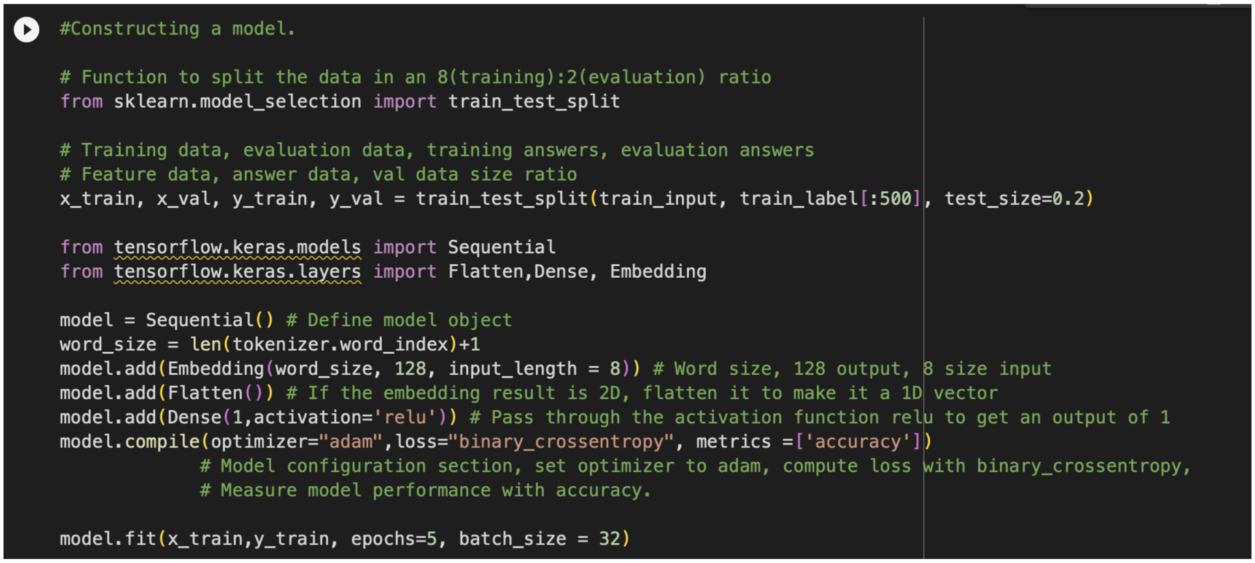


```
[28] # Deeplearning model's input size has a length
     # Each reviews have different lengths.

     # if input size > 17, then can not enter.
     # Fit the size -> fill in with padding.

     train_input = pad_sequences(train_sequence, maxlen=8, padding="post")

     # maxlen=8: paddin, length size of 8.
     # padding="post": fill in with 0 from the back.
     train_input
```

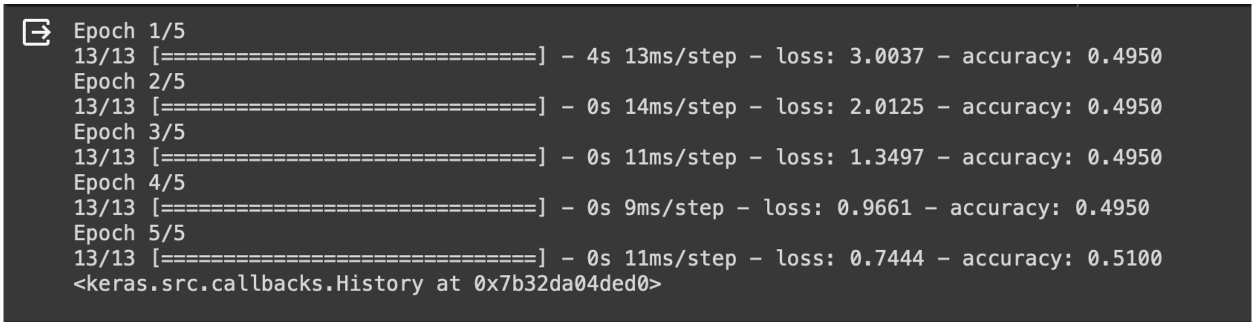
```
array([[  41,  426,   20, ...,    0,    0,    0],
       [ 277,    1,   76, ...,  761,  430,   22],
       [ 762,  763,  431, ...,  432,   14,   12],
       ...,
       [ 548,   35,   30, ...,    0,    0,    0],
       [2105,  144,  131, ...,   11, 2107,    0],
       [   3,   58,   38, ..., 2111,   12,  758]], dtype=int32)
```

```
[29] # Target val.

     train_label = np.array(train['label'])
     train_label
```

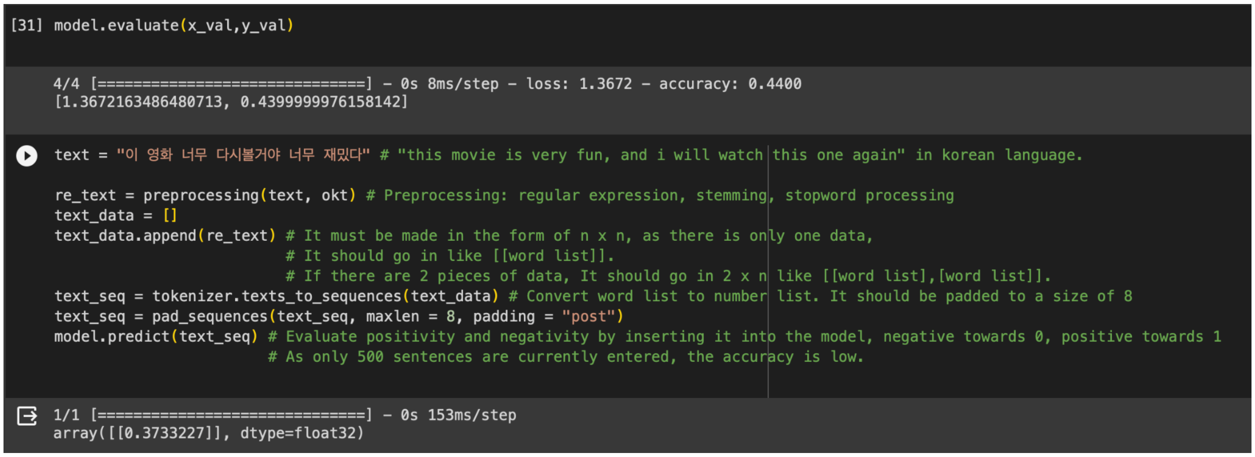
```
array([0, 1, 0, ..., 0, 1, 0])
```



```
#Constructing a model.

# Function to split the data in an 8(training):2(evaluation) ratio
from sklearn.model_selection import train_test_split

# Training data, evaluation data, training answers, evaluation answers
# Feature data, answer data, val data size ratio
x_train, x_val, y_train, y_val = train_test_split(train_input, train_label[:500], test_size=0.2)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten,Dense, Embedding

model = Sequential() # Define model object
word_size = len(tokenizer.word_index)+1
model.add(Embedding(word_size, 128, input_length = 8)) # Word size, 128 output, 8 size input
model.add(Flatten()) # If the embedding result is 2D, flatten it to make it a 1D vector
model.add(Dense(1,activation='relu')) # Pass through the activation function relu to get an output of 1
model.compile(optimizer="adam",loss="binary_crossentropy", metrics =['accuracy'])
             # Model configuration section, set optimizer to adam, compute loss with binary_crossentropy,
             # Measure model performance with accuracy.

model.fit(x_train,y_train, epochs=5, batch_size = 32)
```



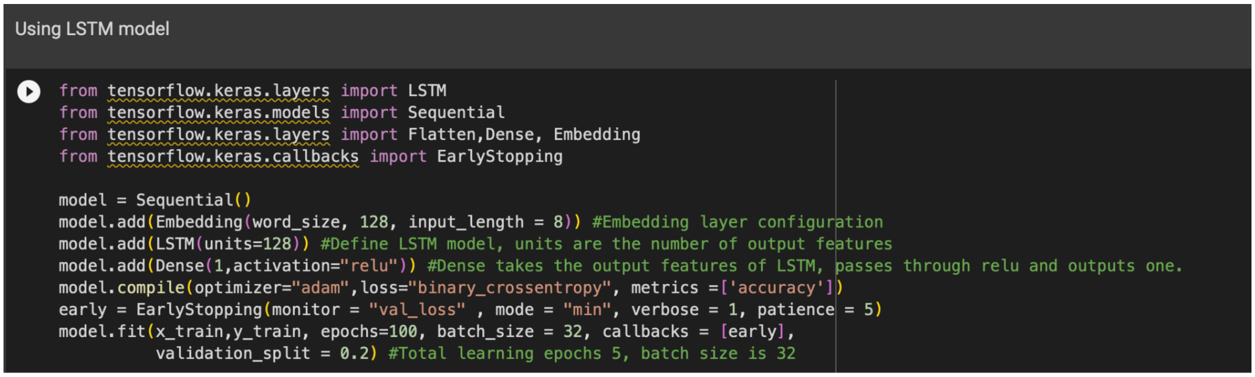
```
Epoch 1/5
13/13 [==============================] - 4s 13ms/step - loss: 3.0037 - accuracy: 0.4950
Epoch 2/5
13/13 [==============================] - 0s 14ms/step - loss: 2.0125 - accuracy: 0.4950
Epoch 3/5
13/13 [==============================] - 0s 11ms/step - loss: 1.3497 - accuracy: 0.4950
Epoch 4/5
13/13 [==============================] - 0s 9ms/step - loss: 0.9661 - accuracy: 0.4950
Epoch 5/5
13/13 [==============================] - 0s 11ms/step - loss: 0.7444 - accuracy: 0.5100
<keras.src.callbacks.History at 0x7b32da04ded0>
```



```
[31] model.evaluate(x_val,y_val)

     4/4 [==============================] - 0s 8ms/step - loss: 1.3672 - accuracy: 0.4400
     [1.3672163486480713, 0.4399999976158142]
```

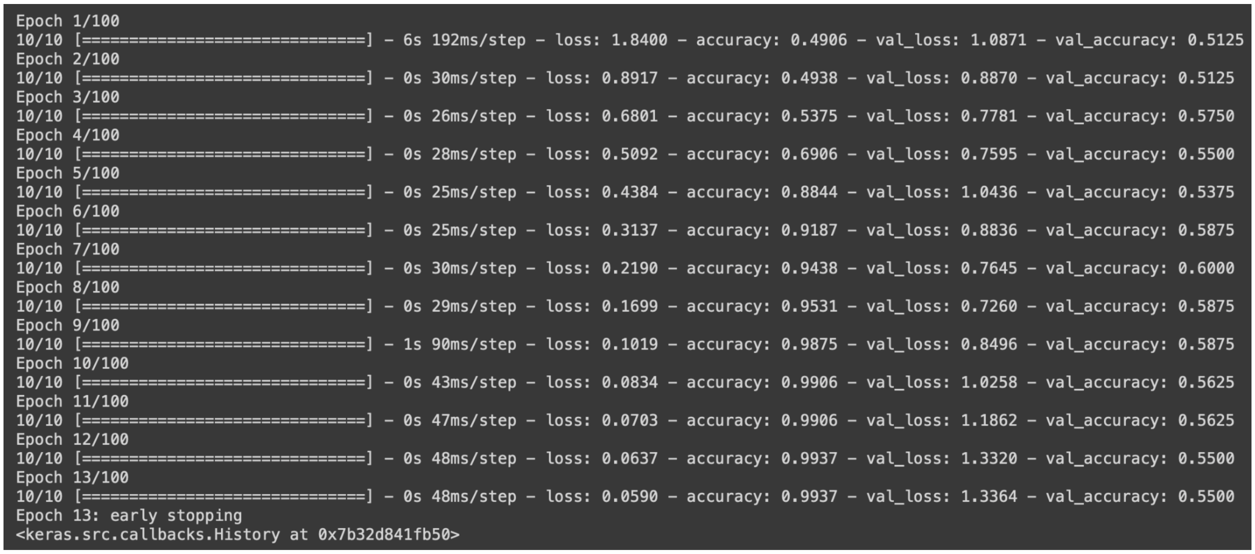
```
text = "이 영화 너무 다시볼거야 너무 재밌다" # "this movie is very fun, and i will watch this one again" in korean language.

re_text = preprocessing(text, okt) # Preprocessing: regular expression, stemming, stopword processing
text_data = []
text_data.append(re_text) # It must be made in the form of n x n, as there is only one data,
                          # It should go in like [[word list]].
                          # If there are 2 pieces of data, It should go in 2 x n like [[word list],[word list]].
text_seq = tokenizer.texts_to_sequences(text_data) # Convert word list to number list. It should be padded to a size of 8
text_seq = pad_sequences(text_seq, maxlen = 8, padding = "post")
model.predict(text_seq) # Evaluate positivity and negativity by inserting it into the model, negative towards 0, positive towards 1
                        # As only 500 sentences are currently entered, the accuracy is low.
```

```
1/1 [==============================] - 0s 153ms/step
array([[0.3733227]], dtype=float32)
```

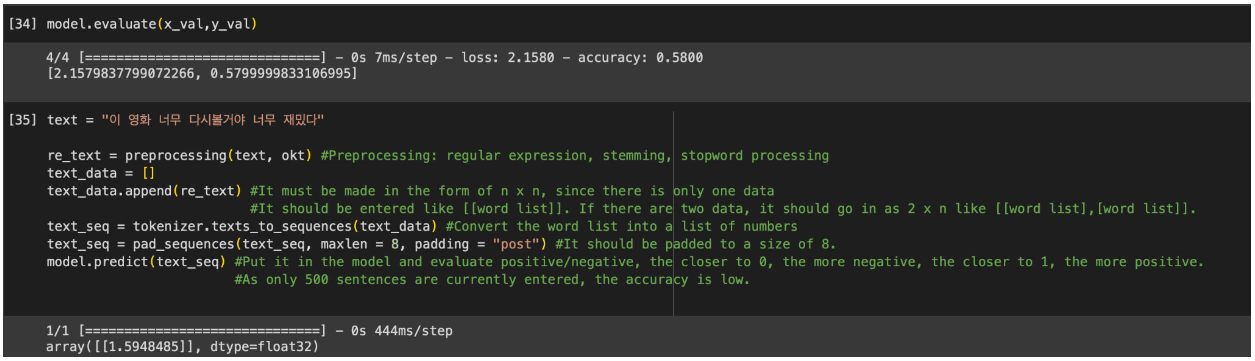


Using LSTM model

```
from tensorflow.keras.layers import LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten,Dense, Embedding
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()
model.add(Embedding(word_size, 128, input_length = 8)) #Embedding layer configuration
model.add(LSTM(units=128)) #Define LSTM model, units are the number of output features
model.add(Dense(1,activation="relu")) #Dense takes the output features of LSTM, passes through relu and outputs one.
model.compile(optimizer="adam",loss="binary_crossentropy", metrics =['accuracy'])
early = EarlyStopping(monitor = "val_loss" , mode = "min", verbose = 1, patience = 5)
model.fit(x_train,y_train, epochs=100, batch_size = 32, callbacks = [early],
          validation_split = 0.2) #Total learning epochs 5, batch size is 32
```



```
Epoch 1/100
10/10 [==============================] - 6s 192ms/step - loss: 1.8400 - accuracy: 0.4906 - val_loss: 1.0871 - val_accuracy: 0.5125
Epoch 2/100
10/10 [==============================] - 0s 30ms/step - loss: 0.8917 - accuracy: 0.4938 - val_loss: 0.8870 - val_accuracy: 0.5125
Epoch 3/100
10/10 [==============================] - 0s 26ms/step - loss: 0.6801 - accuracy: 0.5375 - val_loss: 0.7781 - val_accuracy: 0.5750
Epoch 4/100
10/10 [==============================] - 0s 28ms/step - loss: 0.5092 - accuracy: 0.6906 - val_loss: 0.7595 - val_accuracy: 0.5500
Epoch 5/100
10/10 [==============================] - 0s 25ms/step - loss: 0.4384 - accuracy: 0.8844 - val_loss: 1.0436 - val_accuracy: 0.5375
Epoch 6/100
10/10 [==============================] - 0s 25ms/step - loss: 0.3137 - accuracy: 0.9187 - val_loss: 0.8836 - val_accuracy: 0.5875
Epoch 7/100
10/10 [==============================] - 0s 30ms/step - loss: 0.2190 - accuracy: 0.9438 - val_loss: 0.7645 - val_accuracy: 0.6000
Epoch 8/100
10/10 [==============================] - 0s 29ms/step - loss: 0.1699 - accuracy: 0.9531 - val_loss: 0.7260 - val_accuracy: 0.5875
Epoch 9/100
10/10 [==============================] - 1s 90ms/step - loss: 0.1019 - accuracy: 0.9875 - val_loss: 0.8496 - val_accuracy: 0.5875
Epoch 10/100
10/10 [==============================] - 0s 43ms/step - loss: 0.0834 - accuracy: 0.9906 - val_loss: 1.0258 - val_accuracy: 0.5625
Epoch 11/100
10/10 [==============================] - 0s 47ms/step - loss: 0.0703 - accuracy: 0.9906 - val_loss: 1.1862 - val_accuracy: 0.5625
Epoch 12/100
10/10 [==============================] - 0s 48ms/step - loss: 0.0637 - accuracy: 0.9937 - val_loss: 1.3320 - val_accuracy: 0.5500
Epoch 13/100
10/10 [==============================] - 0s 48ms/step - loss: 0.0590 - accuracy: 0.9937 - val_loss: 1.3364 - val_accuracy: 0.5500
Epoch 13: early stopping
<keras.src.callbacks.History at 0x7b32d841fb50>
```



```
[34] model.evaluate(x_val,y_val)

     4/4 [==============================] - 0s 7ms/step - loss: 2.1580 - accuracy: 0.5800
     [2.1579837799072266, 0.5799999833106995]

[35] text = "이 영화 너무 다시볼거야 너무 재밌다"

     re_text = preprocessing(text, okt) #Preprocessing: regular expression, stemming, stopword processing
     text_data = []
     text_data.append(re_text) #It must be made in the form of n x n, since there is only one data
                               #It should be entered like [[word list]]. If there are two data, it should go in as 2 x n like [[word list],[word list]].
     text_seq = tokenizer.texts_to_sequences(text_data) #Convert the word list into a list of numbers
     text_seq = pad_sequences(text_seq, maxlen = 8, padding = "post") #It should be padded to a size of 8.
     model.predict(text_seq) #Put it in the model and evaluate positive/negative, the closer to 0, the more negative, the closer to 1, the more positive.
                             #As only 500 sentences are currently entered, the accuracy is low.

     1/1 [==============================] - 0s 444ms/step
     array([[1.5948485]], dtype=float32)
```