```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import re
import nltk
import warnings
%matplotlib inline

warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('/content/train.csv')
df.head()
```

	id	title	author	text	label
0	0	House Dem Aide: We Didn’t Even See Comey’s Let...	Darrell Lucus	House Dem Aide: We Didn’t Even See Comey’s Let...	1
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1

Next steps: Generate code with df | New interactive sheet

```
df['title'][0]
```

```
'House Dem Aide: We Didn’t Even See Comey’s Letter Until Jason Chaffetz Tweeted It'
```

```
df['text'][0]
```

```
'House Dem Aide: We Didn’t Even See Comey’s Letter Until Jason Chaffetz Tweeted It By Darrell Lucus on October 30, 2016 Su
bscribe Jason Chaffetz on the stump in American Fork, Utah ( image courtesy Michael Jolley, available under a Creative Com
mons-BY license) \nWith apologies to Keith Olbermann, there is no doubt who the Worst Person in The World is this week–FBI
Director James Comey. But according to a House Democratic aide, it looks like we also know who the second-worst person is
as well. It turns out that when Comey sent his now-infamous letter announcing that the FBI was looking into emails that ma
y be related to Hillary Clinton’s email server, the ranking Democrats on the relevant committees didn’t hear about it from
Comey. They found out via a tweet from one of the Republican committee chairmen. \nAs we now know, Comey notified the Repu
blican chairmen and Democratic ranking members of the House Intelligence, Judiciary, and Oversight committees that his age
ncy was reviewing email…'
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20800 entries, 0 to 20799
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      20800 non-null  int64 
 1   title   20242 non-null  object
 2   author  18843 non-null  object
 3   text    20761 non-null  object
 4   label   20800 non-null  int64 
dtypes: int64(2), object(3)
memory usage: 812.6+ KB
```

```
# drop unnecessary columns
df = df.drop(columns=['id', 'title', 'author'], axis=1)
```

```
# drop null values
df = df.dropna(axis=0)
```

```
len(df)
```

```
20761
```

```
# remove special characters and punctuations
```

```
df['clean_news'] = df['text'].str.lower()
df['clean_news']
```

	clean_news
0	house dem aide: we didn’t even see comey’s let...
1	ever get the feeling your life circles the rou...
2	why the truth might get you fired october 29, ...
3	videos 15 civilians killed in single us airstr...
4	print \nan iranian woman has been sentenced to...
...	...
20795	rapper t. i. unloaded on black celebrities who...
20796	when the green bay packers lost to the washing...
20797	the macy’s of today grew from the union of sev...
20798	nato, russia to hold parallel exercises in bal...
20799	david swanson is an author, activist, journa...

20761 rows × 1 columns

dtype: object

```
df['clean_news'] = df['clean_news'].str.replace('[^A-Za-z0-9\s]', '')
df['clean_news'] = df['clean_news'].str.replace('\n', '')
df['clean_news'] = df['clean_news'].str.replace('\s+', ' ')
df['clean_news']
```

	clean_news
0	house dem aide: we didn’t even see comey’s let...
1	ever get the feeling your life circles the rou...
2	why the truth might get you fired october 29, ...
3	videos 15 civilians killed in single us airstr...
4	print an iranian woman has been sentenced to s...
...	...
20795	rapper t. i. unloaded on black celebrities who...
20796	when the green bay packers lost to the washing...
20797	the macy’s of today grew from the union of sev...
20798	nato, russia to hold parallel exercises in bal...
20799	david swanson is an author, activist, journa...

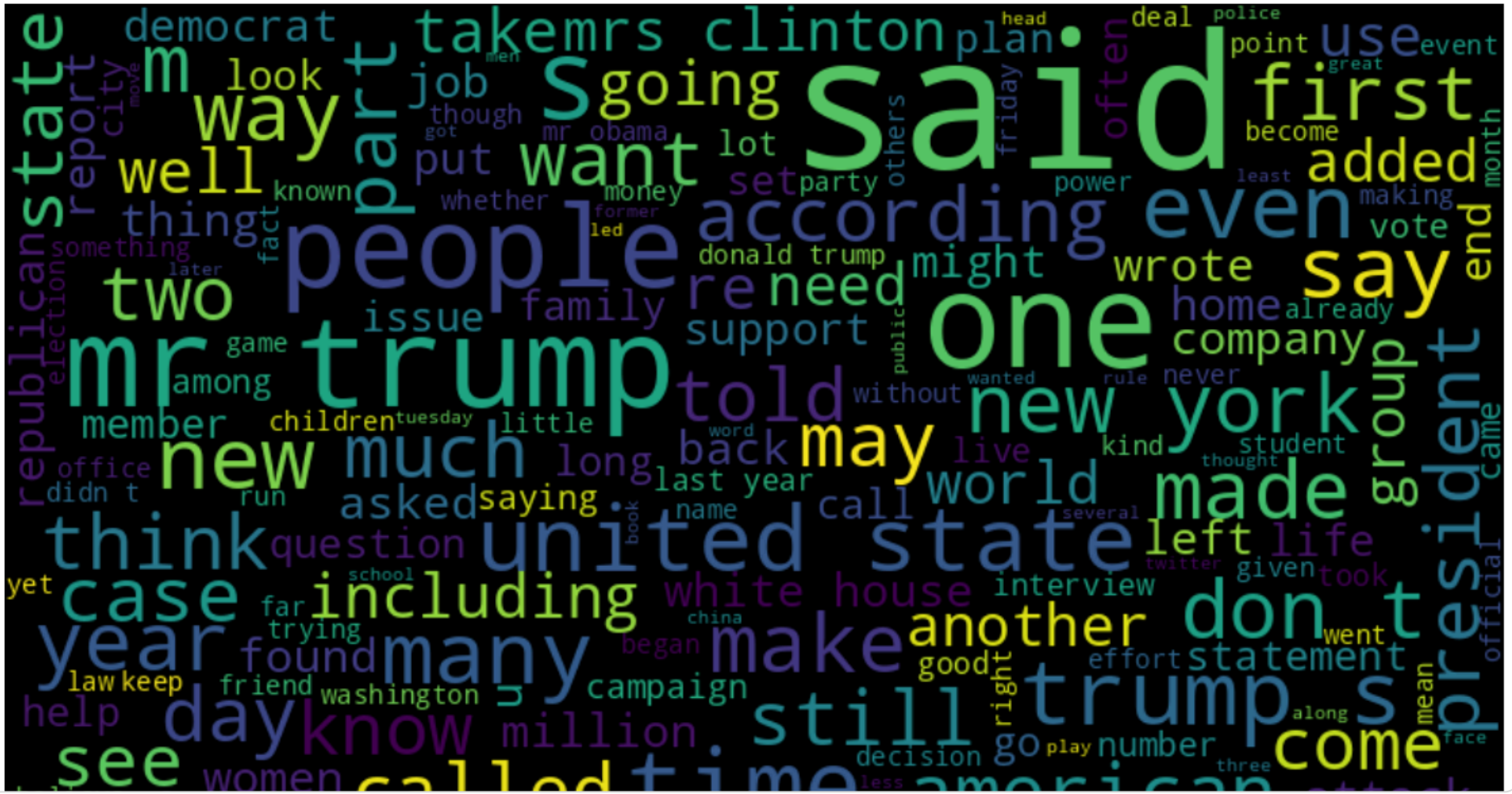
20761 rows × 1 columns

dtype: object

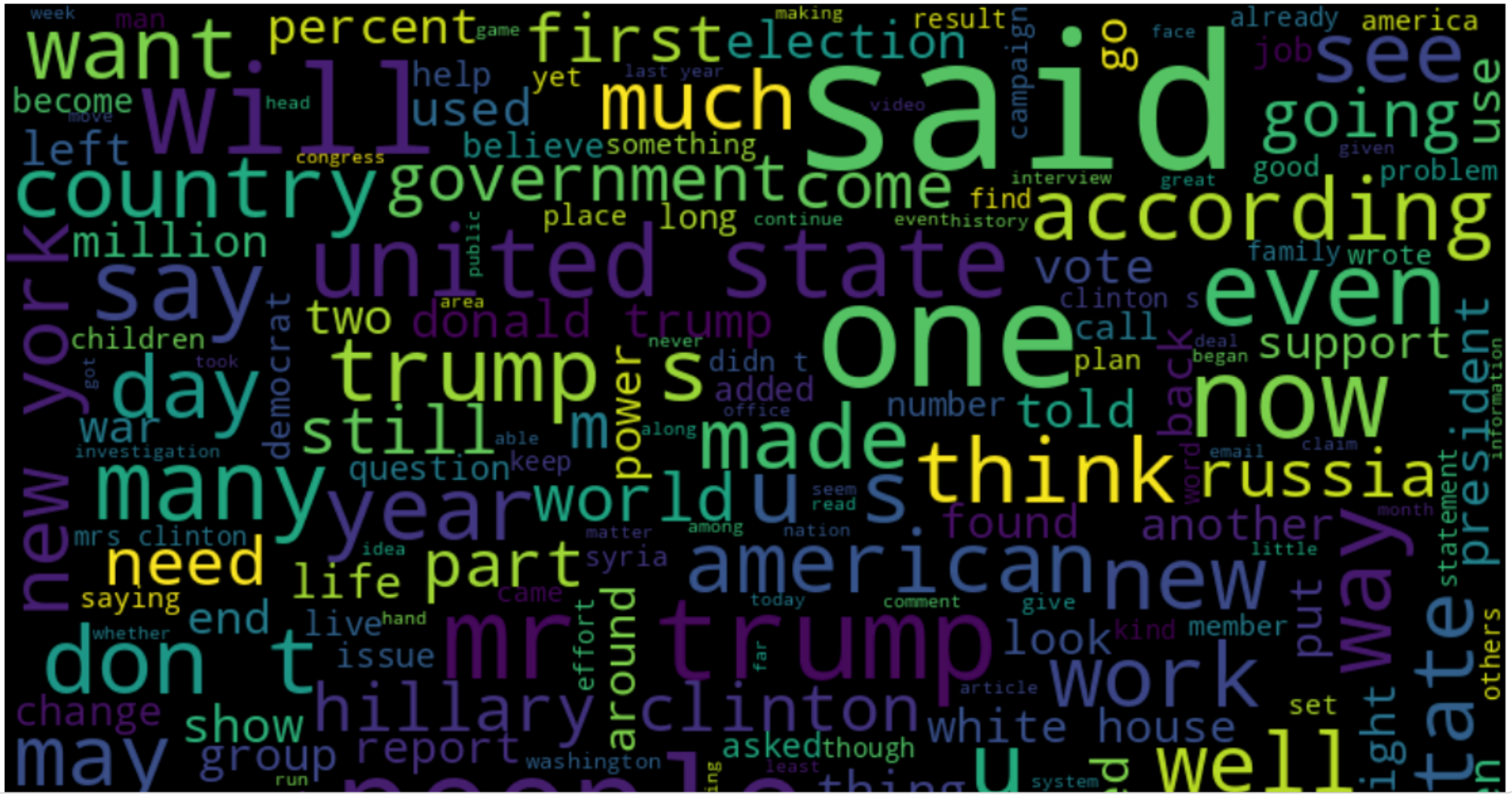
```
# remove stopwords
from nltk.corpus import stopwords
```

```
# visualize the frequent words for genuine news
all_words = " ".join([sentence for sentence in df['clean_news'][df['label']==0]])
```

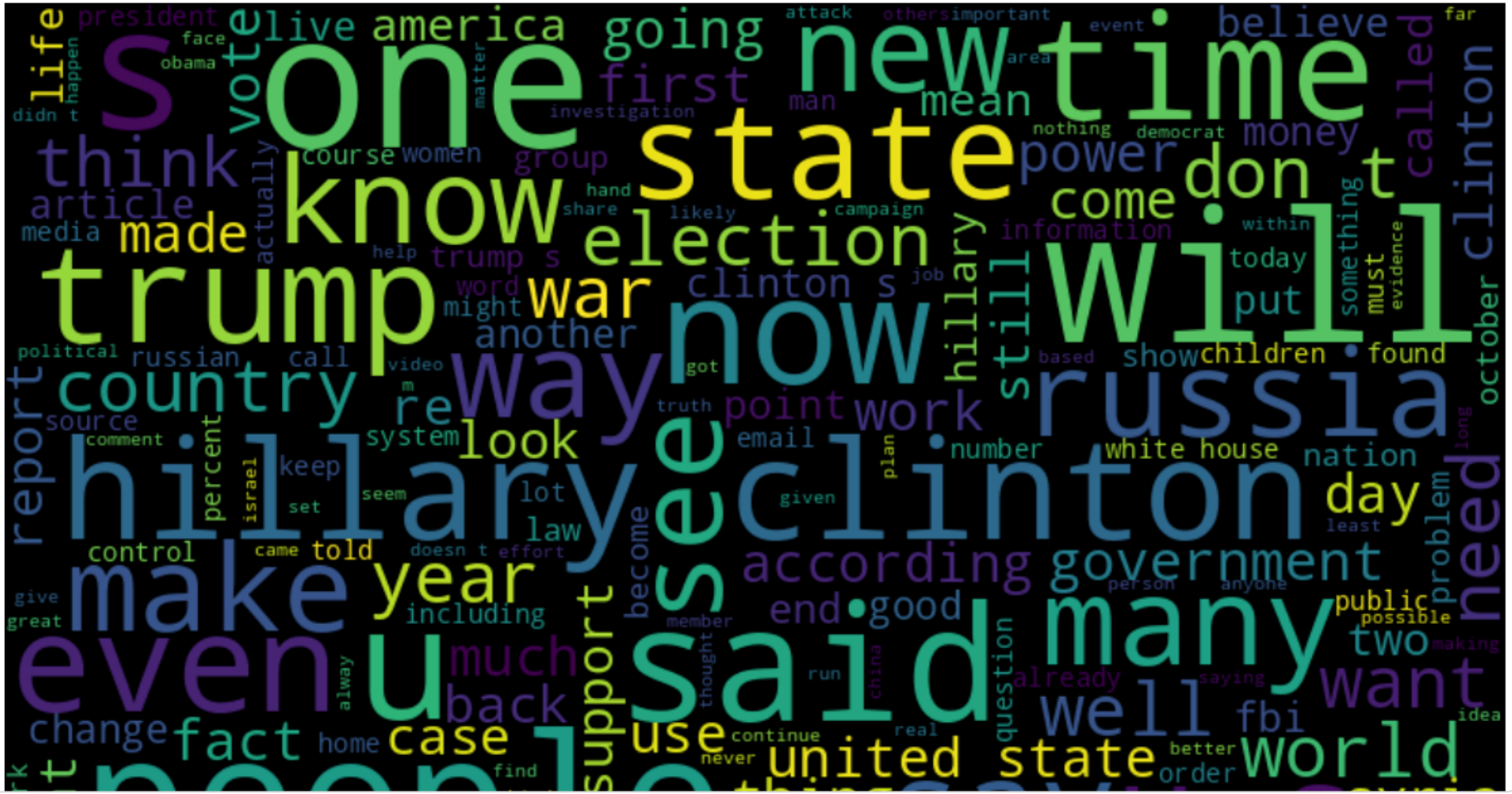
```
wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)

# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



```
# visualize the frequent words
all_words = " ".join([sentence for sentence in df['clean_news']])

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)

# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



```
# visualize the frequent words for fake news
all_words = " ".join([sentence for sentence in df['clean_news'][df['label']==1]])

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_words)

# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



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

```
# tokenize text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['clean_news'])
word_index = tokenizer.word_index
vocab_size = len(word_index)
vocab_size
```

```
239556
```

```
# padding data
sequences = tokenizer.texts_to_sequences(df['clean_news'])
padded_seq = pad_sequences(sequences, maxlen=500, padding='post', truncating='post')
```

```
# download GloVe embeddings
!wget http://nlp.stanford.edu/data/glove.6B.zip

# unzip the downloaded file
!unzip glove.6B.zip
```

```
--2025-10-05 11:40:10--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2025-10-05 11:40:10--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-10-05 11:40:10--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
```

```
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.01MB/s    in 2m 39s

2025-10-05 11:42:49 (5.19 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

```
# create embedding index
embedding_index = {}
with open('glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefs
```

```
# create embedding matrix
embedding_matrix = np.zeros((vocab_size+1, 100))
for word, i in word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
embedding_matrix[1]
```

```
array([-0.038194  , -0.24487001,  0.72812003, -0.39961001,  0.083172  ,
        0.043953  , -0.39140999,  0.3344    , -0.57545   ,  0.087459  ,
        0.28786999, -0.06731   ,  0.30906001, -0.26383999, -0.13231   ,
       -0.20757   ,  0.33395001, -0.33848   , -0.31742999, -0.48335999,
```

```
 0.1464    , -0.37303999,  0.34577   ,  0.052041  ,  0.44946   ,
-0.46970999,  0.02628   , -0.54154998, -0.15518001, -0.14106999,
-0.039722  ,  0.28277001,  0.14393   ,  0.23464   , -0.31020999,
 0.086173  ,  0.20397   ,  0.52623999,  0.17163999, -0.082378  ,
-0.71787   , -0.41531   ,  0.20334999, -0.12763   ,  0.41367   ,
 0.55186999,  0.57907999, -0.33476999, -0.36559001, -0.54856998,
-0.062892  ,  0.26583999,  0.30204999,  0.99774998, -0.80480999,
-3.0243001 ,  0.01254   , -0.36941999,  2.21670008,  0.72201002,
-0.24978   ,  0.92136002,  0.034514  ,  0.46744999,  1.10790002,
-0.19358   , -0.074575  ,  0.23353   , -0.052062  , -0.22044   ,
 0.057162  , -0.15806   , -0.30798   , -0.41624999,  0.37972   ,
 0.15006   , -0.53211999, -0.20550001, -1.25259995,  0.071624  ,
 0.70564997,  0.49744001, -0.42063001,  0.26148   , -1.53799999,
-0.30223   , -0.073438  , -0.28312001,  0.37103999, -0.25217   ,
 0.016215  , -0.017099  , -0.38984001,  0.87423998, -0.72569001,
-0.51058   , -0.52028   , -0.1459    ,  0.82779998,  0.27061999])
```

```
padded_seq[1]
```

```
array([   359,    114,      1,   1739,     89,    193,   5191,      1,
        29168,    544,     67,   2463,      6,      5,   2201,    491,
          675,      1,   1724,   5627,    115,     74,    935,      1,
          291,    492,     10,   2089,      6,  18251,    945,  21175,
         3179,   2845,    782,    880,     49,     90,    370,      2,
         5460,     49,   2171,   3352,      9,      1,  11444,      3,
           49,    237,      1,    124,   2345,  92527,   3298,   3833,
            7,   3757,   3352,    942,      3,    270,   3644,    115,
        10185,  58259,     12,     72,   3423,     87,    329,     56,
           48,     94,   3399,      5,  21175,  10441,    708,      1,
           68,      3,      1,    616,   2597,     98,   2612,      6,
         8398,      7,      1,    602,    830,   1616,    758,     58,
          253,     15,     32,     82,  23502,     15,      2,     33,
           38,   1039,     13,      2,  25181,    447,      3,      1,
           63,      1,    274,   3877,      6,    145,  11889,      1,
         1016,   2881,    112,     85,     20,      5,    975,   5180,
          287,    136,    431,    132,   5645,      7,   2345,   4700,
          128,     14,    371,      7,     53,     32,    431,    942,
            9,    320,      8,     52,      3,     79,     35,      1,
         2333,      3,     79,     11,   2345,  10185,    155,     49,
         8880,     64,  11605,   1454, 125513,  21610,      2,      1,
```

```
 3949, 125514,      4,  92528, 125515,      6,   5724,      1,
33554,      6,  28306,   8367,  19718,     49,  75656,   4639,
   35,     37,     20,    272,    374,  24594,   4639,    273,
  480,     12,      1,   1041,   1766,    550,   1461,      1,
   94,      2,    464,    121,      2,      5,   2111,      6,
    1,    127,    117,    518,    156,   8787,   3812,  19383,
  288,      6,      9,   1517,      9,    888,      9,  58260,
    9,      1,   2575,      3,   8154,     15,     16,   1763,
 2719,    261,     74,      6,     49,    599,  10587,     10,
12826,  18517,   9040,   6804,    208,  92529,      4,  58261,
10664,      4,  14257,  75657,    436,      1,     55,    233,
  768,    629,    493,     57,      3,      5,    202,  58262,
    5,   2642,  20389,  32303,     11,  10185,   7392,   3784,
    6,   8398,    592,      8,     43,    962,      7,     56,
   32,    281,      1,    708,     20,     70,  14712,     53,
   14,     13,     31,    546,    126,      2,    141,      9,
   72,    653,    288,     61,      2,    126,      4,     17,
92530,     40,   1301,    583,     40,      1,   1243,      3,
 1301,    184,      9,     79,      4,      1,    124,     41,
  495,     40,    306,    583,     40,      1,   1301,   8239,
  570,     53,     62,     34,    141,     40,     14,     53,
   62,     34,    141,     40,      5,   1739,      7,  32304,
    5,   1887,      4,      7,    664,      8,     24,     85,
 2830,     18,     98,     33,     19,  21611,     11,      1,
32303,  65335,   1136,  12411,    395,    145,    650,     48,
 1136,    110,     24,     85,    726,      7,     50,  21611,
   49,     64,  14113,  38331,      1,   1841,   2041,   3870,
    4,      1,   2361,    387,     74,    212,   1454,      1,
21611,   1266,     10,    410,     60,     13,      5,   8295,
    3,  13703,     10,      1,   1332,      2,      1,  20044,
    3,     49,    142,  15530,      4,   8545,     25,      1,
   95,     50,     35,     39,  20764,     49,   3423,    168,
    2,   3211,    168,     10,      1,    415,   1053,   2212,
    2,  20045,    121,     35,   1776,     15,      5,    362,
  582,      9,      1,   3023,      3,     98,  15530,     74,
  694,     49, 125516,      4,     23,    615,     15,   7647,
    5,  75658,     11,      5,  40306,     11,  22986,   1737,
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(padded_seq, df['label'], test_size=0.20, random_state=42, stratify=df
```

```
from keras.layers import LSTM, Dropout, Dense, Embedding
from keras import Sequential

# model = Sequential([
#     Embedding(vocab_size+1, 100, weights=[embedding_matrix], trainable=False),
#     Dropout(0.2),
#     LSTM(128, return_sequences=True),
#     LSTM(128),
#     Dropout(0.2),
#     Dense(512),
#     Dropout(0.2),
#     Dense(256),
#     Dense(1, activation='sigmoid')
# ])

model = Sequential([
    Embedding(vocab_size+1, 100, weights=[embedding_matrix], trainable=False),
    Dropout(0.2),
    LSTM(128),
    Dropout(0.2),
    Dense(256),
    Dense(1, activation='sigmoid')
])
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	23,955,700
dropout (Dropout)	?	0
lstm (LSTM)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0
dense (Dense)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

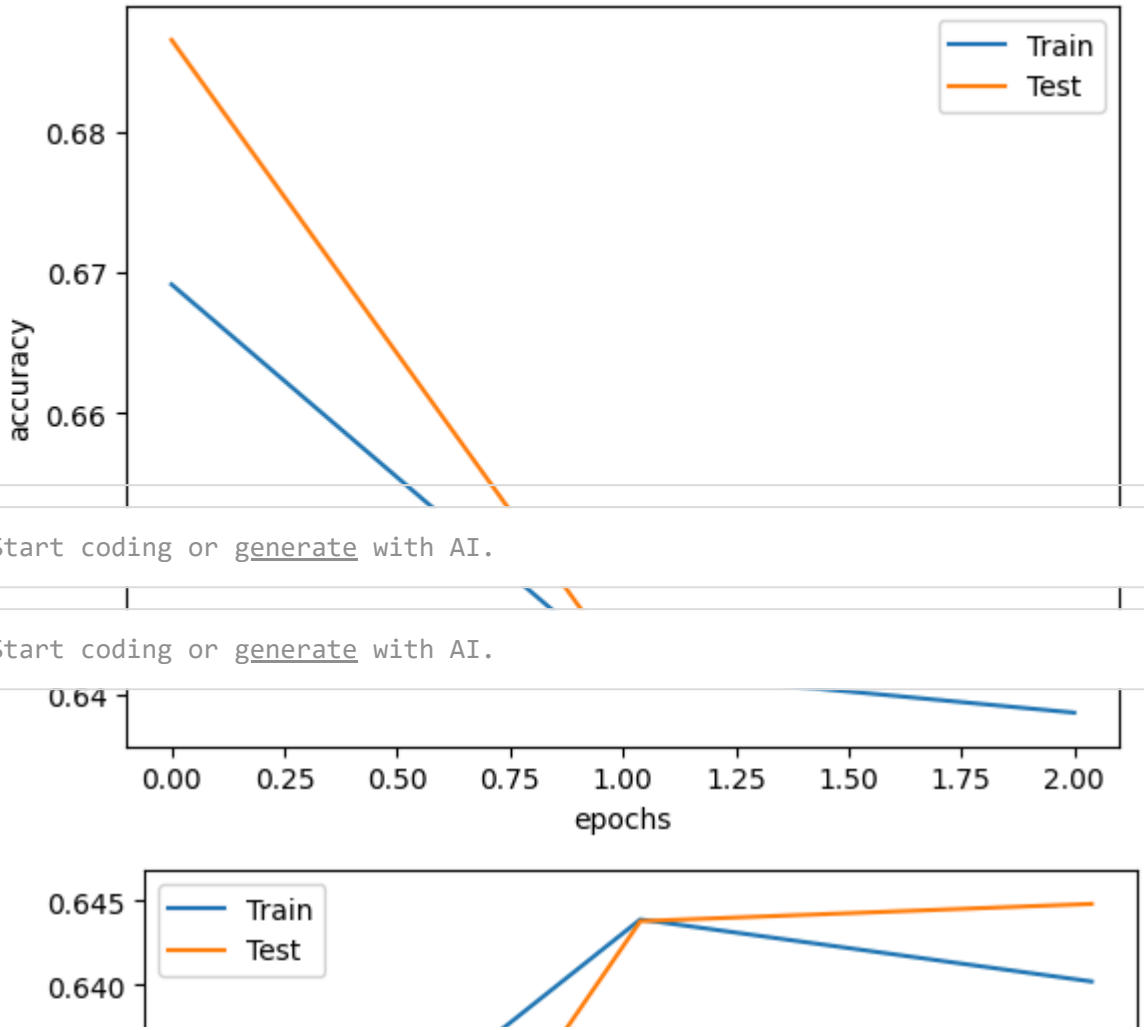
Total params: 23,955,700 (91.38 MB)
Trainable params: 0 (0.00 B)
Non-trainable params: 23,955,700 (91.38 MB)

```
# train the model
history = model.fit(x_train, y_train, epochs=3, batch_size=256, validation_data=(x_test, y_test))
```

```
Epoch 1/3
65/65 ━━━━━━━━━━━━━━━━━━━━ 200s 3s/step - accuracy: 0.6572 - loss: 0.6330 - val_accuracy: 0.6865 - val_loss: 0.6045
Epoch 2/3
65/65 ━━━━━━━━━━━━━━━━━━━━ 195s 3s/step - accuracy: 0.6561 - loss: 0.6290 - val_accuracy: 0.6419 - val_loss: 0.6438
Epoch 3/3
65/65 ━━━━━━━━━━━━━━━━━━━━ 199s 3s/step - accuracy: 0.6563 - loss: 0.6294 - val_accuracy: 0.6439 - val_loss: 0.6448
```

```
# visualize the results
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['Train', 'Test'])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
plt.plot(history.history['val_loss'])
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['Train', 'Test'])
plt.show()
```



Start coding or generate with AI.

Start coding or generate with AI.