```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from matplotlib import style
style.use('ggplot')
import re
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
#stop_words = set(stopwords.words('english'))
import nltk
nltk.download('stop_words')
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

```
[nltk_data] Error loading stop_words: Package 'stop_words' not found
[nltk_data]     in index
```

```
df = pd.read_csv('IMDB-Dataset.csv',error_bad_lines=False, engine="python")
df.head()
```

```
<ipython-input-17-7490ff12c5c8>:1: FutureWarning: The error_bad_lines argument has be
```

```
  df = pd.read_csv('IMDB-Dataset.csv',error_bad_lines=False, engine="python")
Skipping line 34833: unexpected end of data
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
df.shape
```

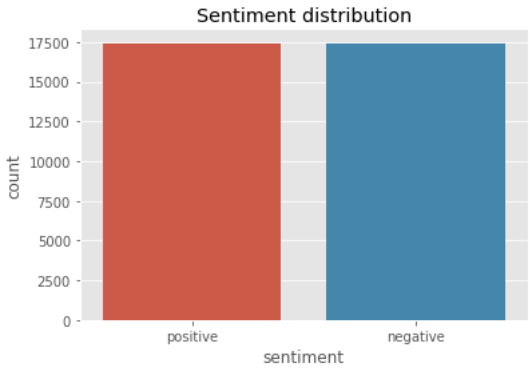
```
(34831, 2)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34831 entries, 0 to 34830
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     34831 non-null  object
 1   sentiment  34831 non-null  object
dtypes: object(2)
memory usage: 544.4+ KB
```

```
sns.countplot(x='sentiment', data=df)
plt.title("Sentiment distribution")
```

```
Text(0.5, 1.0, 'Sentiment distribution')
```



```
for i in range(5):
    print("Review: ", [i])
    print(df['review'].iloc[i], "\n")
    print("Sentiment: ", df['sentiment'].iloc[i], "\n\n")
```

```
Review:  [0]
One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly

Sentiment:  positive


Review:  [1]
A wonderful little production. <br /><br />The filming technique is very unassuming- very old-time-BBC fashion and gives a comforti

Sentiment:  positive


Review:  [2]
I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a

Sentiment:  positive


Review:  [3]
Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time.

Sentiment:  negative


Review:  [4]
Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portrait about human

Sentiment:  positive
```

```
def no_of_words(text):
    words= text.split()
    word_count = len(words)
    return word_count


df['word count'] = df['review'].apply(no_of_words)
df.head()
```

	review	sentiment	word count
0	One of the other reviewers has mentioned that ...	positive	307
1	A wonderful little production. The...	positive	162
2	I thought this was a wonderful way to spend ti...	positive	166
3	Basically there's a family where a little boy ...	negative	138
4	Petter Mattei's "Love in the Time of Money" is...	positive	230

```
df.sentiment.replace("positive", 1, inplace=True)
df.sentiment.replace("negative", 0, inplace=True)
df.head()
```

	review	sentiment	word count
0	One of the other reviewers has mentioned that ...	1	307
1	A wonderful little production. The...	1	162
2	I thought this was a wonderful way to spend ti...	1	166
3	Basically there's a family where a little boy ...	0	138
4	Petter Mattei's "Love in the Time of Money" is...	1	230

```
def data_processing(text):
    text = text.lower()
    text = re.sub('<br />', '', text)
    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)
    text = re.sub(r'\@w+|\#','', text)
    text = re.sub(r'[^\w\s]','', text)
    text_tokens = word_tokenize(text)
    filtered_text = [w for w in text_tokens if not w in stop_words]
    return " ".join(filtered_text)
```

```
duplicated_count = df.duplicated().sum()
print("Number of duplicate entries: ", duplicated_count)
```

```
Number of duplicate entries:  203
```

```
df = df.drop_duplicates('review')
```

```
stemmer = PorterStemmer()
def stemming(data):
    text = [stemmer.stem(word) for word in data]
    return data
```

```
df.review = df['review'].apply(lambda x: stemming(x))
df['word count'] = df['review'].apply(no_of_words)
df.head()
```

	review	sentiment	word count
0	One of the other reviewers has mentioned that ...	1	307
1	A wonderful little production. The...	1	162
2	I thought this was a wonderful way to spend ti...	1	166
3	Basically there's a family where a little boy ...	0	138
4	Petter Mattei's "Love in the Time of Money" is...	1	230

```
pos_reviews = df[df.sentiment ==1]
pos_reviews.head()
```

	review	sentiment	word count
0	One of the other reviewers has mentioned that ...	1	307
1	A wonderful little production. The...	1	162
2	I thought this was a wonderful way to spend ti...	1	166
4	Petter Mattei's "Love in the Time of Money" is...	1	230
5	Probably my all-time favorite movie, a story o...	1	119

```
text = ' '.join([word for word in pos_reviews['review']])
plt.figure(figsize=(20,15), facecolor='None')
wordcloud = WordCloud(max_words=500, width=1600, height=800).generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title('Most frequent words in positive reviews', fontsize = 19)
plt.show()
```

Most frequent words in positive reviews

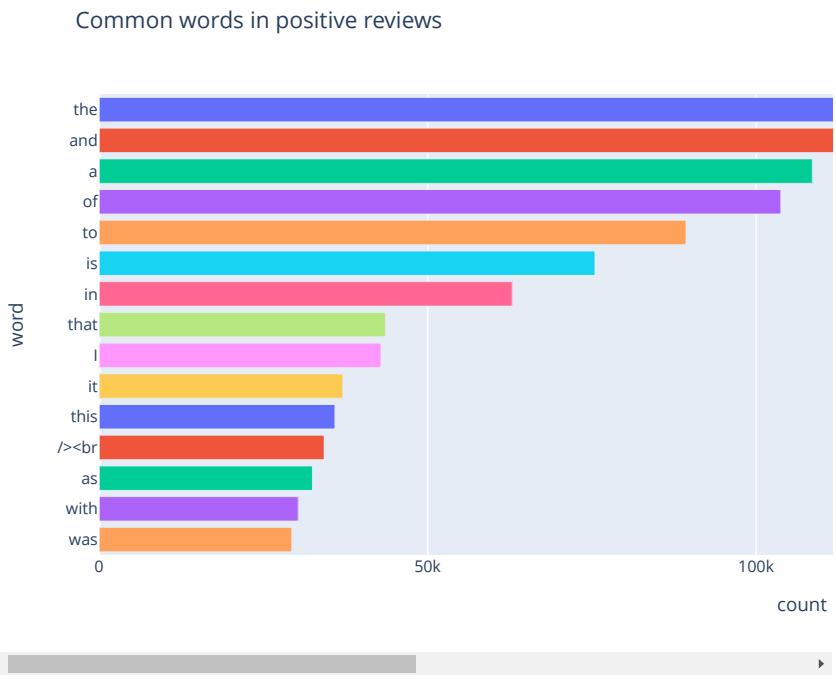
```
from collections import Counter
count = Counter()
for text in pos_reviews['review'].values:
    for word in text.split():
        count[word] +=1
count.most_common(15)
```

```
[('the', 203274),
 ('and', 115479),
 ('a', 108542),
 ('of', 103760),
 ('to', 89330),
 ('is', 75470),
 ('in', 62866),
 ('that', 43585),
 ('I', 42906),
 ('it', 37090),
 ('this', 35883),
 ('/><br', 34244),
 ('as', 32452),
 ('with', 30292),
 ('was', 29322)]
```

```
pos_words = pd.DataFrame(count.most_common(15))
pos_words.columns = ['word', 'count']
pos_words.head()
```

	word	count
0	the	203274
1	and	115479
2	a	108542
3	of	103760
4	to	89330

```
px.bar(pos_words, x='count', y='word', title='Common words in positive reviews', color='word')
```




```
neg_reviews = df[df.sentiment == 0]
neg_reviews.head()
```

	review	sentiment	word count
3	Basically there's a family where a little boy ...	0	138
7	This show was an amazing, fresh & innovative i...	0	174

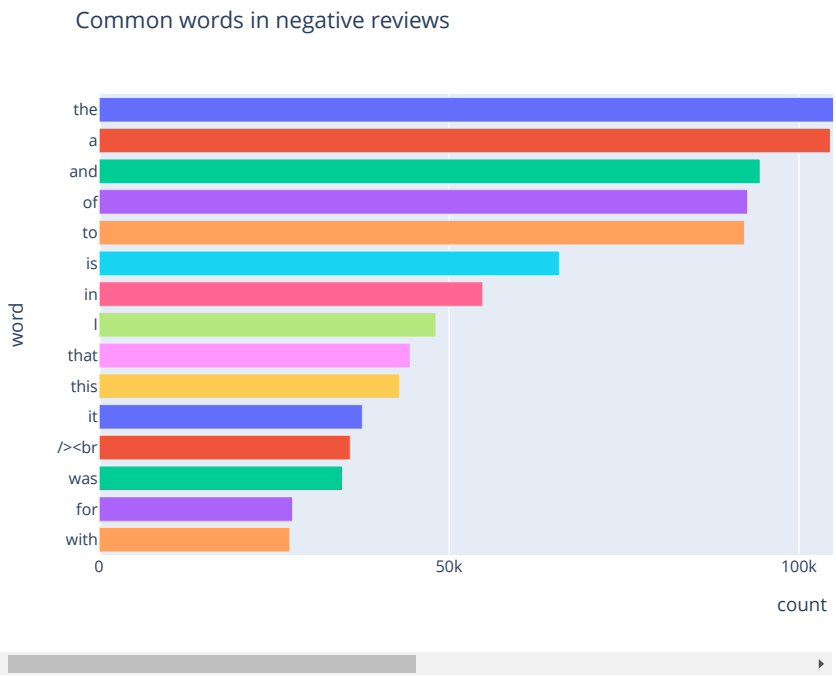
```
count = Counter()
for text in neg_reviews['review'].values:
    for word in text.split():
        count[word] += 1
count.most_common(15)
```

```
[('the', 190821),
 ('a', 104484),
 ('and', 94449),
 ('of', 92649),
 ('to', 92214),
 ('is', 65747),
 ('in', 54817),
 ('I', 48116),
 ('that', 44438),
 ('this', 42889),
 ('it', 37594),
 ('/><br', 35873),
 ('was', 34772),
 ('for', 27641),
 ('with', 27224)]
```

```
neg_words = pd.DataFrame(count.most_common(15))
neg_words.columns = ['word', 'count']
neg_words.head()
```

	word	count
0	the	190821
1	a	104484
2	and	94449
3	of	92649
4	to	92214

```
px.bar(neg_words, x='count', y='word', title='Common words in negative reviews', color='word')
```




```
X = df['review']
Y = df['sentiment']
vect = TfidfVectorizer()
X = vect.fit_transform(df['review'])
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
```

```
print("Size of x_train: ", (x_train.shape))
print("Size of y_train: ", (y_train.shape))
print("Size of x_test: ", (x_test.shape))
print("Size of y_test: ", (y_test.shape))
```

```
Size of x_train:  (24239, 87790)
Size of y_train:  (24239,)
Size of x_test:  (10389, 87790)
Size of y_test:  (10389,)
```

```
x_train = x_train[:2000]
y_train = y_train[:2000]
x_test = x_test[:500]
y_test = y_test[:500]
print("Size of x_train: ", (x_train.shape))
print("Size of y_train: ", (y_train.shape))
print("Size of x_test: ", (x_test.shape))
print("Size of y_test: ", (y_test.shape))
```

```
Size of x_train:  (2000, 87790)
Size of y_train:  (2000,)
Size of x_test:  (500, 87790)
Size of y_test:  (500,)
```

```
x_train = x_train.toarray()
x_test = x_test.toarray()
```

```
from keras.models import Sequential
from keras.layers import Dense
```

```
model = Sequential()
model.add(Dense(units=16, activation='relu', input_dim=x_train.shape[1]))
model.add(Dense(units=8, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=10, epochs=15)
```

```
Epoch 1/15
200/200 [==============================] - 4s 17ms/step - loss: 0.6581 - accuracy: 0.5995
Epoch 2/15
200/200 [==============================] - 3s 17ms/step - loss: 0.5186 - accuracy: 0.8070
Epoch 3/15
200/200 [==============================] - 6s 28ms/step - loss: 0.3874 - accuracy: 0.9245
Epoch 4/15
200/200 [==============================] - 4s 22ms/step - loss: 0.3006 - accuracy: 0.9695
Epoch 5/15
200/200 [==============================] - 4s 18ms/step - loss: 0.2424 - accuracy: 0.9895
Epoch 6/15
200/200 [==============================] - 4s 22ms/step - loss: 0.2009 - accuracy: 0.9935
Epoch 7/15
200/200 [==============================] - 4s 18ms/step - loss: 0.1682 - accuracy: 0.9955
Epoch 8/15
200/200 [==============================] - 3s 17ms/step - loss: 0.1422 - accuracy: 0.9960
Epoch 9/15
200/200 [==============================] - 4s 21ms/step - loss: 0.1209 - accuracy: 0.9965
Epoch 10/15
200/200 [==============================] - 4s 18ms/step - loss: 0.1030 - accuracy: 0.9965
Epoch 11/15
200/200 [==============================] - 4s 18ms/step - loss: 0.0880 - accuracy: 0.9965
Epoch 12/15
200/200 [==============================] - 4s 19ms/step - loss: 0.0755 - accuracy: 0.9965
Epoch 13/15
200/200 [==============================] - 4s 20ms/step - loss: 0.0653 - accuracy: 0.9965
Epoch 14/15
200/200 [==============================] - 4s 18ms/step - loss: 0.0565 - accuracy: 0.9965
Epoch 15/15
200/200 [==============================] - 3s 17ms/step - loss: 0.0494 - accuracy: 0.9965
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                1404656
```

```
 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 1,404,801
Trainable params: 1,404,801
Non-trainable params: 0
_________________________________________________________________
```

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
16/16 [==============================] - 0s 13ms/step - loss: 0.6128 - accuracy: 0.8120
Test loss: 0.6127944588661194
Test accuracy: 0.8119999766349792
```

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.plot(history.history['loss'], color='r', label='loss')
plt.title('Training Loss')
plt.xlabel("Number of epochs")
plt.ylabel("Loss")
plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], color='b', label='accuracy')
plt.title('Training Accuracy')
plt.xlabel("Number of epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```