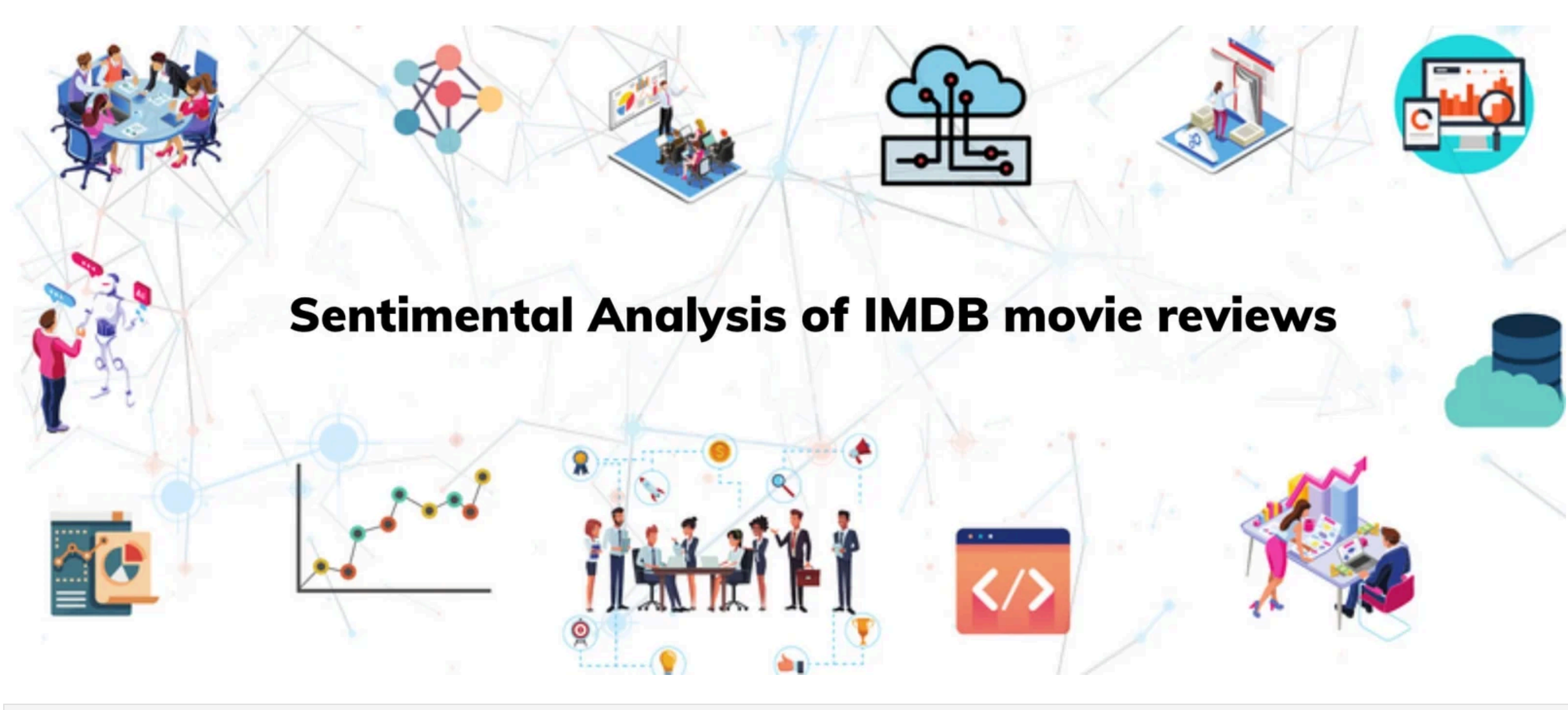


In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from matplotlib import style
style.use('ggplot')
import re
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

In [3]:
```
df = pd.read_csv('IMDB Dataset.csv')
df.head()
```

Out[3]:

	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

In [4]:
```
df.shape
```

Out[4]:
```
(50000, 2)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

In [6]:
```
sns.countplot(x='sentiment', data=df)
plt.title("Sentiment distribution")
```

Out[6]:
```
Text(0.5, 1.0, 'Sentiment distribution')
```

In [7]:
```
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 what happened with me.<br /><br />The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.<br /><br />It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.<br /><br />I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side.

Sentiment:  positive


Review:  [1]
A wonderful little production. <br /><br />The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece. <br /><br />The actors are extremely well chosen- Michael Sheen not only "has got all the polari" but he has all the voices down pat too! You can truly see the seamless editing guided by the references to Williams' diary entries, not only is it well worth the watching but it is a terrificly written and performed piece. A masterful production about one of the great master's of comedy and his life. <br /><br />The realism really comes home with the little things: the fantasy of the guard which, rather than use the traditional 'dream' techniques remains solid then disappears. It plays on our knowledge and our senses, particularly with the scenes concerning Orton and Halliwell and the sets (particularly of their flat with Halliwell's murals decorating every surface) are terribly well done.

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 light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer). While some may be disappointed when they realize this is not Match Point 2: Risk Addiction, I thought it was proof that Woody Allen is still fully in control of the style many of us have grown to love.<br /><br />This was the most I'd laughed at one of Woody's comedies in years (dare I say a decade?). While I've never been impressed with Scarlet Johanson, in this she managed to tone down her "sexy" image and jumped right into a average, but spirited young woman.<br /><br />This may not be the crown jewel of his career, but it was wittier than "Devil Wears Prada" and more interesting than "Superman" a great comedy to go see with friends.

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.<br /><br />This movie is slower than a soap opera... and suddenly, Jake decides to become Rambo and kill the zombie.<br /><br />OK, first of all when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is watchable. Parents are divorcing & arguing like in real life. And then we have Jake with his closet which totally ruins all the film! I expected to see a BOOGEYMAN similar movie, and instead i watched a drama with some meaningless thriller spots.<br /><br />3 out of 10 just for the well playing parents & descent dialogs. As for the shots with Jake: just ignore them.

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 relations. This is a movie that seems to be telling us what money, power and success do to people in the different situations we encounter. <br /><br />This being a variation on the Arthur Schnitzler's play about the same theme, the director transfers the action to the present time New York where all these different characters meet and connect. Each one is connected in one way, or another to the next person, but no one seems to know the previous point of contact. Stylishly, the film has a sophisticated luxurious look. We are taken to see how these people live and the world they live in their own habitat.<br /><br />The only thing one gets out of all these souls in the picture is the different stages of loneliness each one inhabits. A big city is not exactly the best place in which human relations find sincere fulfillment, as one discerns is the case with most of the people we encounter.<br /><br />The acting is good under Mr. Mattei's direction. Steve Buscemi, Rosario Dawson, Carol Kane, Michael Imperioli, Adrian Grenier, and the rest of the talented cast, make these characters come alive.<br /><br />We wish Mr. Mattei good luck and await anxiously for his next work.

Sentiment:  positive
```

In [8]:
```
def no_of_words(text):
    words= text.split()
    word_count = len(words)
    return word_count
```

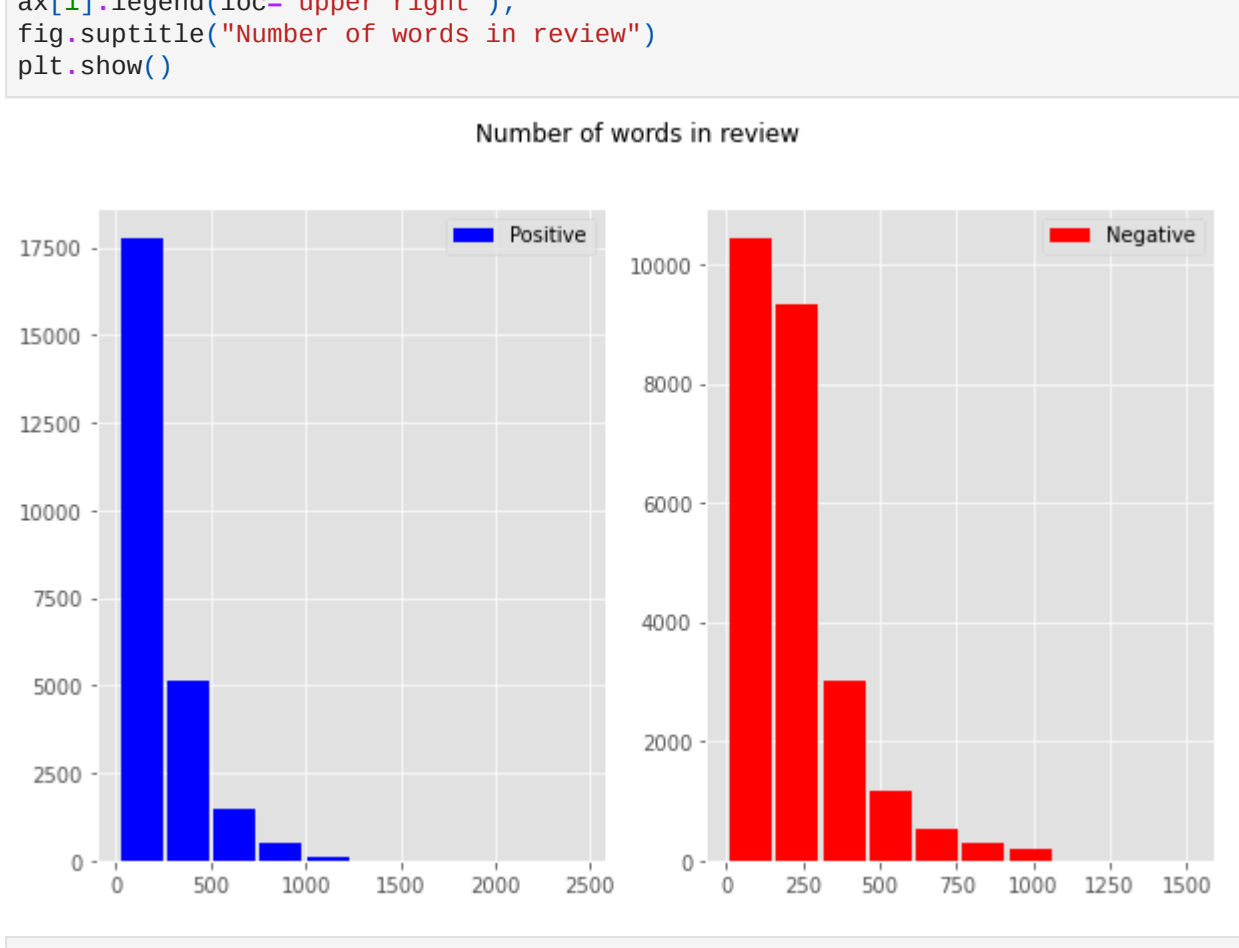
In [9]:
```
df['word count'] = df['review'].apply(no_of_words)
```

In [10]:
```
df.head()
```

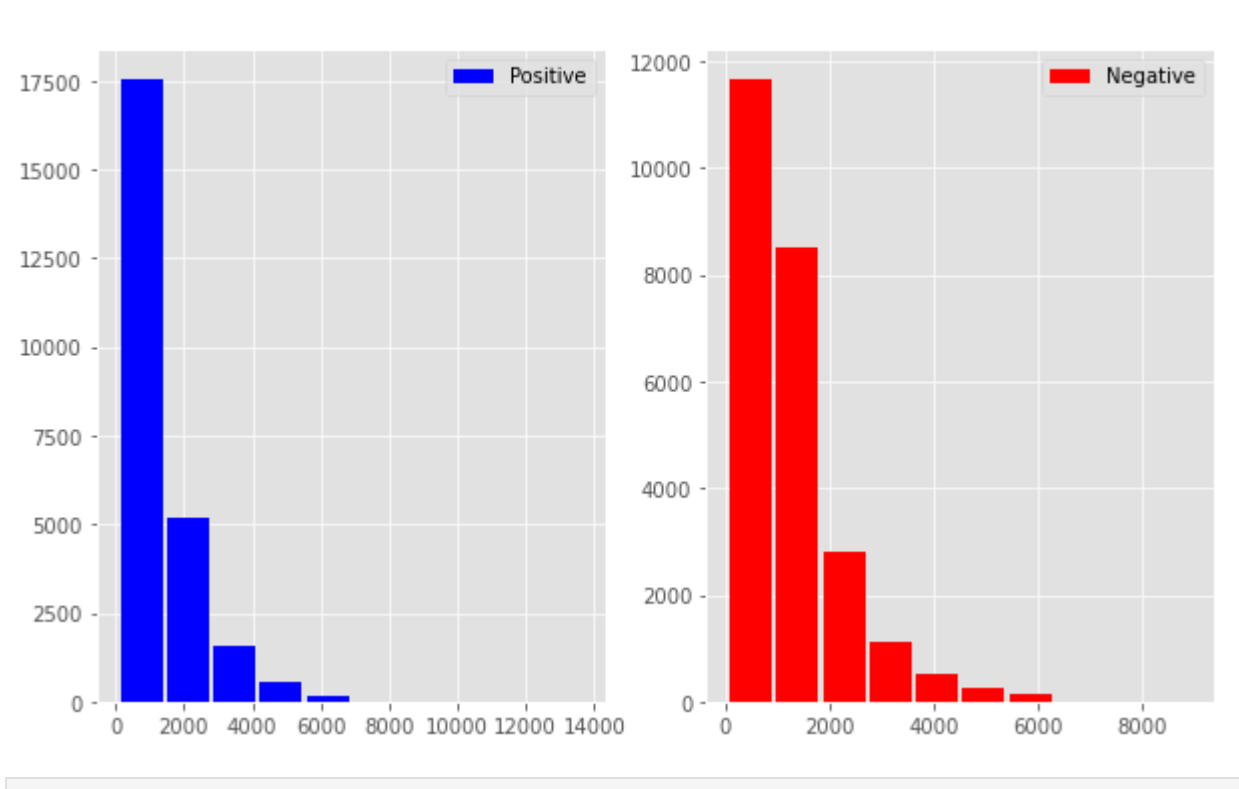
Out[10]:

	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

In [11]:
```
fig, ax = plt.subplots(1,2, figsize=(10,6))
ax[0].hist(df[df['sentiment'] == 'positive']['word count'], label='Positive', color='blue', rwidth=0.9);
ax[0].legend(loc='upper right');
ax[1].hist(df[df['sentiment'] == 'negative']['word count'], label='Negative', color='red', rwidth=0.9);
ax[1].legend(loc='upper right');
fig.suptitle("Number of words in review")
plt.show()
```



In [12]:
```
fig, ax = plt.subplots(1,2, figsize=(10,6))
ax[0].hist(df[df['sentiment'] == 'positive']['review'].str.len(), label='Positive', color='blue', rwidth=0.9);
ax[0].legend(loc='upper right');
ax[1].hist(df[df['sentiment'] == 'negative']['review'].str.len(), label='Negative', color='red', rwidth=0.9);
ax[1].legend(loc='upper right');
fig.suptitle("Number of words in review")
plt.show()
```



In [13]:
```
df.sentiment.replace("positive", 1, inplace=True)
df.sentiment.replace("negative", 0, inplace=True)
```

In [14]:
```
df.head()
```

Out[14]:

	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

In [15]:
```
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)
```

In [16]:
```
df.review = df['review'].apply(data_processing)
```

In [17]:
```
duplicated_count = df.duplicated().sum()
print("Number of duplicate entries: ", duplicated_count)
```

```
Number of duplicate entries:  421
```

In [18]:
```
df = df.drop_duplicates('review')
```

In [19]:
```
stemmer = PorterStemmer()
def stemming(data):
    text = [stemmer.stem(word) for word in data]
    return data
```

In [20]:
```
df.review = df['review'].apply(lambda x: stemming(x))
```

In [21]:
```
df['word count'] = df['review'].apply(no_of_words)
df.head()
```

Out[21]:

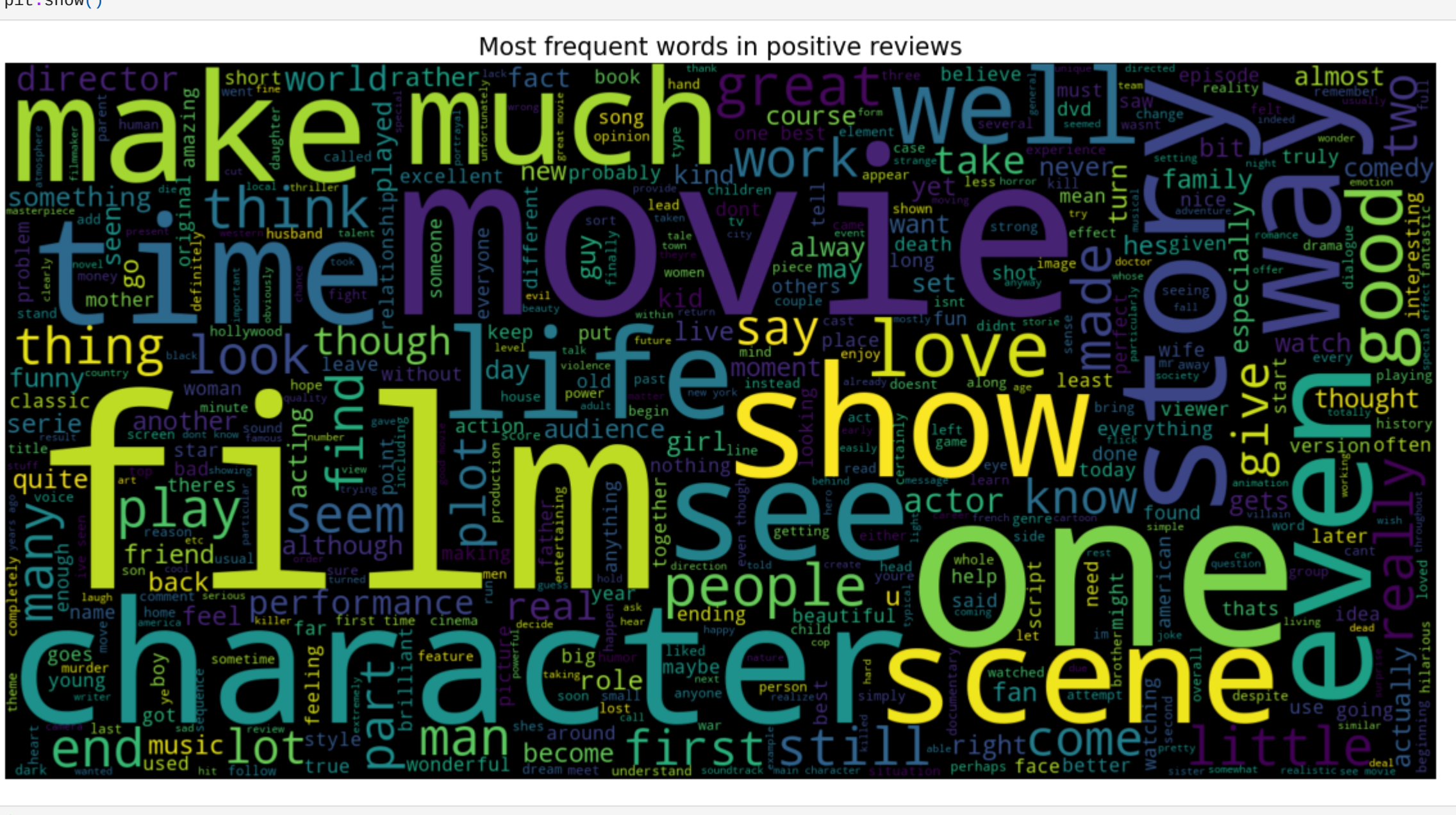
	review	sentiment	word count
0	one reviewers mentioned watching 1 oz episode ...	1	168
1	wonderful little production filming technique ...	1	84
2	thought wonderful way spend time hot summer we...	1	86
3	basically theres family little boy jake thinks...	0	67
4	petter matteis love time money visually stunni...	1	125

In [22]:
```
pos_reviews = df[df.sentiment ==1]
pos_reviews.head()
```

Out[22]:

	review	sentiment	word count
0	one reviewers mentioned watching 1 oz episode ...	1	168
1	wonderful little production filming technique ...	1	84
2	thought wonderful way spend time hot summer we...	1	86
4	petter matteis love time money visually stunni...	1	125
5	probably alltime favorite movie story selfless...	1	58

In [23]:
```
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()
```



In [24]:
```
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)
```

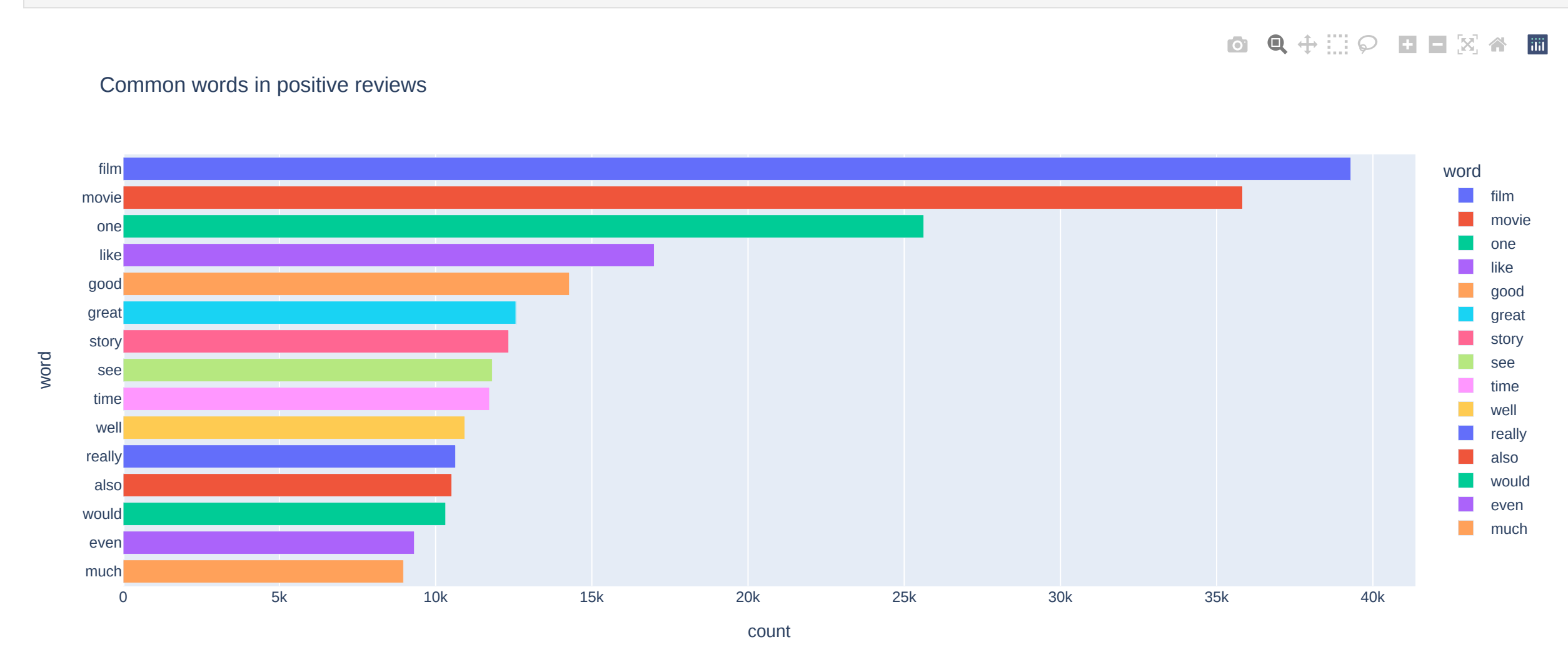
Out[24]:
```
[('film', 39285),
 ('movie', 35830),
 ('one', 25621),
 ('like', 16998),
 ('good', 14281),
 ('great', 12568),
 ('story', 12338),
 ('see', 11814),
 ('time', 11724),
 ('well', 10930),
 ('really', 10638),
 ('also', 10516),
 ('would', 10320),
 ('even', 9318),
 ('much', 8971)]
```

In [25]:
```
pos_words = pd.DataFrame(count.most_common(15))
pos_words.columns = ['word', 'count']
pos_words.head()
```

Out[25]:

	word	count
0	film	39285
1	movie	35830
2	one	25621
3	like	16998
4	good	14281

In [26]:
```
px.bar(pos_words, x='count', y='word', title='Common words in positive reviews', color='word')
```

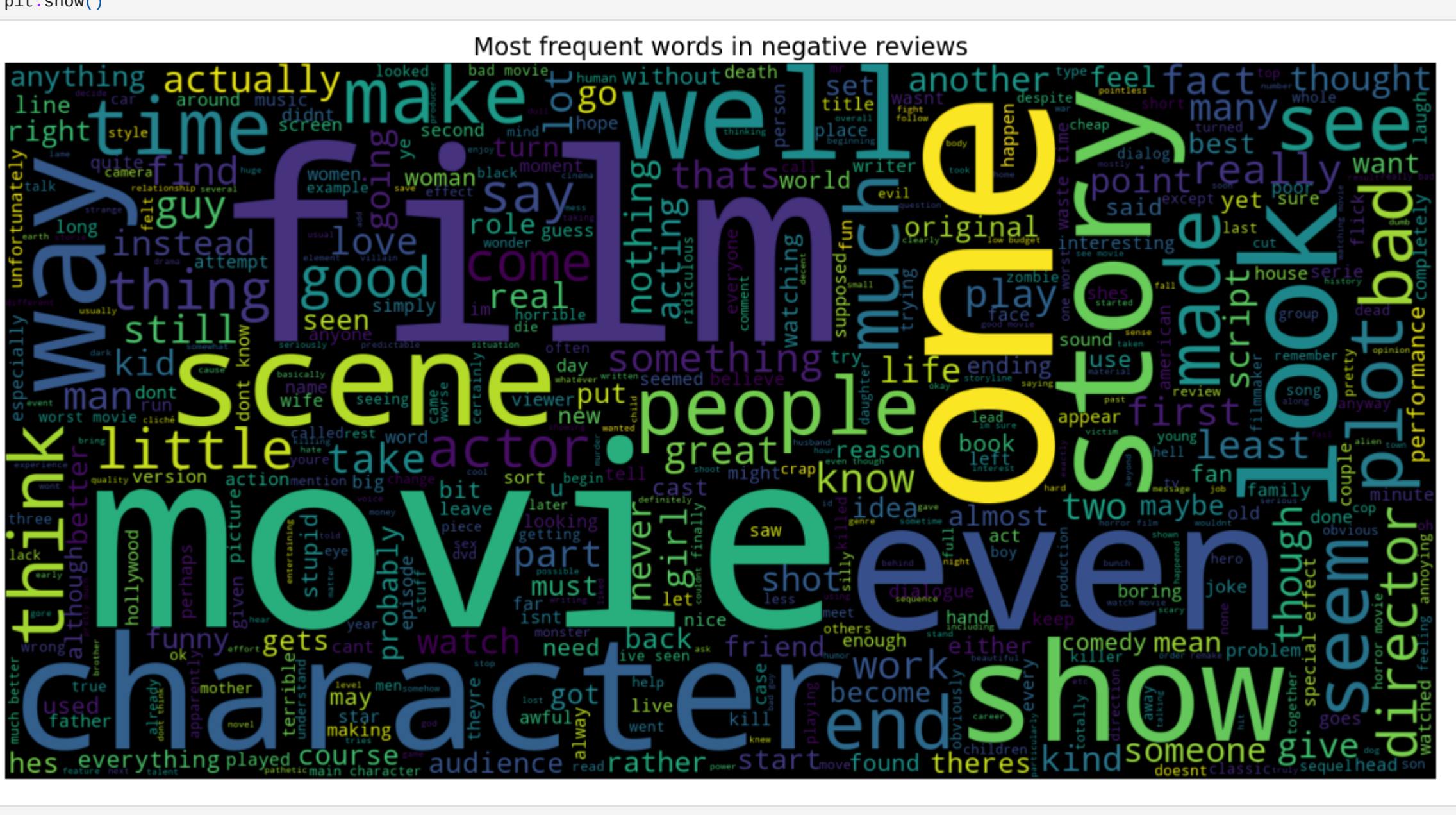


In [27]:
```
neg_reviews = df[df.sentiment == 0]
neg_reviews.head()
```

Out[27]:

	review	sentiment	word count
3	basically theres family little boy jake thinks...	0	67
7	show amazing fresh innovative idea 70s first a...	0	83
8	encouraged positive comments film looking forw...	0	64
10	phil alien one quirky films humour based aroun...	0	51
11	saw movie 12 came recall scariest scene big bi...	0	84

In [28]:
```
text = ' '.join([word for word in neg_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 negative reviews', fontsize = 19)
plt.show()
```



In [29]:
```
count = Counter()
for text in neg_reviews['review'].values:
    for word in text.split():
        count[word] += 1
count.most_common(15)
```

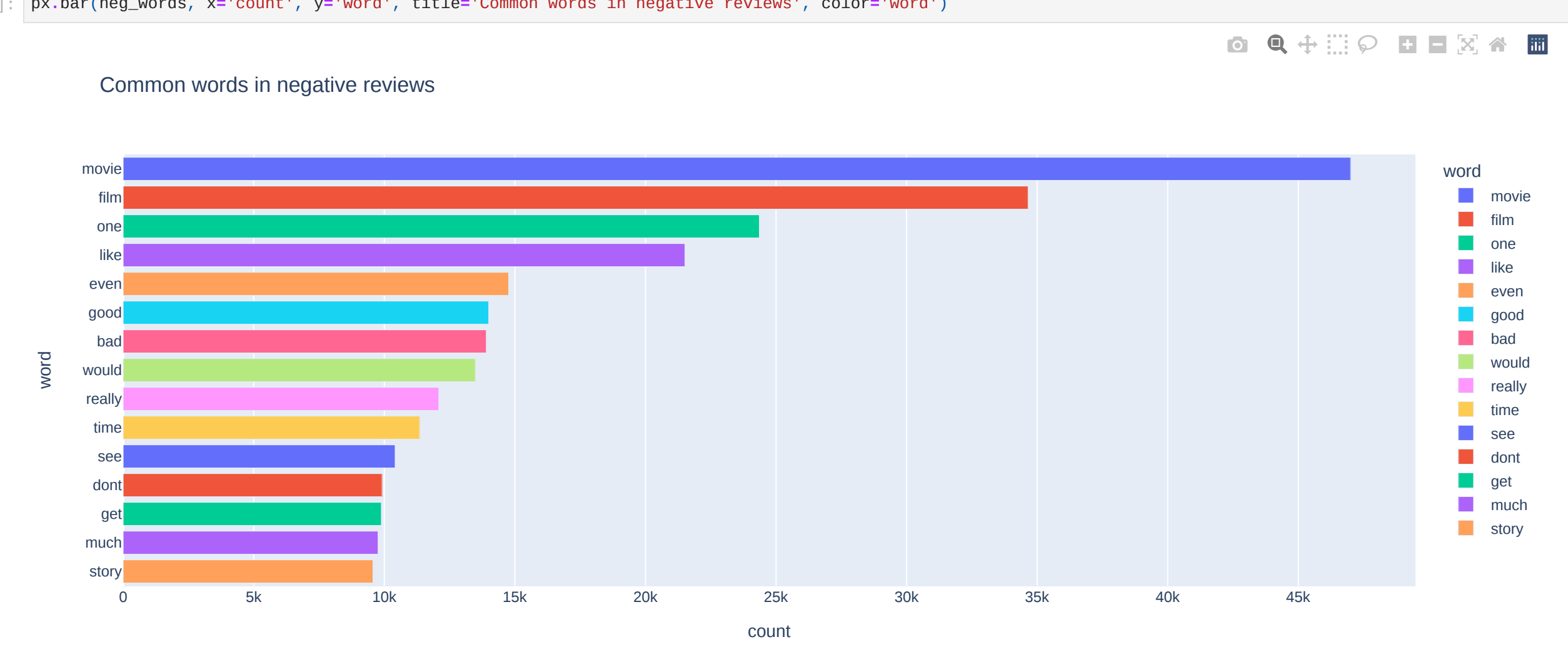
Out[29]:
```
[('movie', 47001),
 ('film', 34651),
 ('one', 24361),
 ('like', 21508),
 ('even', 14759),
 ('good', 13995),
 ('bad', 13903),
 ('would', 13482),
 ('really', 12084),
 ('time', 11349),
 ('see', 10412),
 ('dont', 9912),
 ('get', 9884),
 ('much', 9758),
 ('story', 9563)]
```

In [30]:
```
neg_words = pd.DataFrame(count.most_common(15))
neg_words.columns = ['word', 'count']
neg_words.head()
```

Out[30]:

	word	count
0	movie	47001
1	film	34651
2	one	24361
3	like	21508
4	even	14759

In [31]:
```
px.bar(neg_words, x='count', y='word', title='Common words in negative reviews', color='word')
```



In [32]:
```
X = df['review']
Y = df['sentiment']
```

In [33]:
```
vect = TfidfVectorizer()
X = vect.fit_transform(df['review'])
```

In [34]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
```

In [35]:
```
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:  (34704, 221707)
Size of y_train:  (34704,)
Size of x_test:  (14874, 221707)
Size of y_test:  (14874,)
```

In [36]:
```
x_train = x_train[:2000]
y_train = y_train[:2000]
x_test = x_test[:500]
y_test = y_test[:500]
```

In [37]:
```
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, 221707)
Size of y_train:  (2000,)
Size of x_test:  (500, 221707)
Size of y_test:  (500,)
```

In [38]:
```
x_train = x_train.toarray()
x_test = x_test.toarray()
```

In [39]:
```
from keras.models import Sequential
from keras.layers import Dense
```

In [40]:
```
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'))
```

In [41]:
```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

In [42]:
```
history = model.fit(x_train, y_train, batch_size=10, epochs=15)
```

```
Epoch 1/15
200/200 [==============================] - 6s 28ms/step - loss: 0.6776 - accuracy: 0.7190
Epoch 2/15
200/200 [==============================] - 5s 25ms/step - loss: 0.5571 - accuracy: 0.9160
Epoch 3/15
200/200 [==============================] - 5s 25ms/step - loss: 0.3565 - accuracy: 0.9535
Epoch 4/15
200/200 [==============================] - 5s 26ms/step - loss: 0.1900 - accuracy: 0.9715
Epoch 5/15
200/200 [==============================] - 5s 25ms/step - loss: 0.0982 - accuracy: 0.9795
Epoch 6/15
200/200 [==============================] - 5s 25ms/step - loss: 0.0540 - accuracy: 0.9870
Epoch 7/15
200/200 [==============================] - 5s 26ms/step - loss: 0.0311 - accuracy: 0.9915
Epoch 8/15
200/200 [==============================] - 5s 25ms/step - loss: 0.0184 - accuracy: 0.9960
Epoch 9/15
200/200 [==============================] - 5s 25ms/step - loss: 0.0116 - accuracy: 0.9980
Epoch 10/15
200/200 [==============================] - 5s 26ms/step - loss: 0.0076 - accuracy: 0.9985
Epoch 11/15
200/200 [==============================] - 5s 25ms/step - loss: 0.0055 - accuracy: 0.9990
Epoch 12/15
200/200 [==============================] - 5s 25ms/step - loss: 0.0038 - accuracy: 0.9995
Epoch 13/15
200/200 [==============================] - 5s 27ms/step - loss: 0.0030 - accuracy: 0.9995
Epoch 14/15
200/200 [==============================] - 6s 28ms/step - loss: 0.0024 - accuracy: 0.9995
Epoch 15/15
200/200 [==============================] - 5s 25ms/step - loss: 0.0022 - accuracy: 0.9995
```

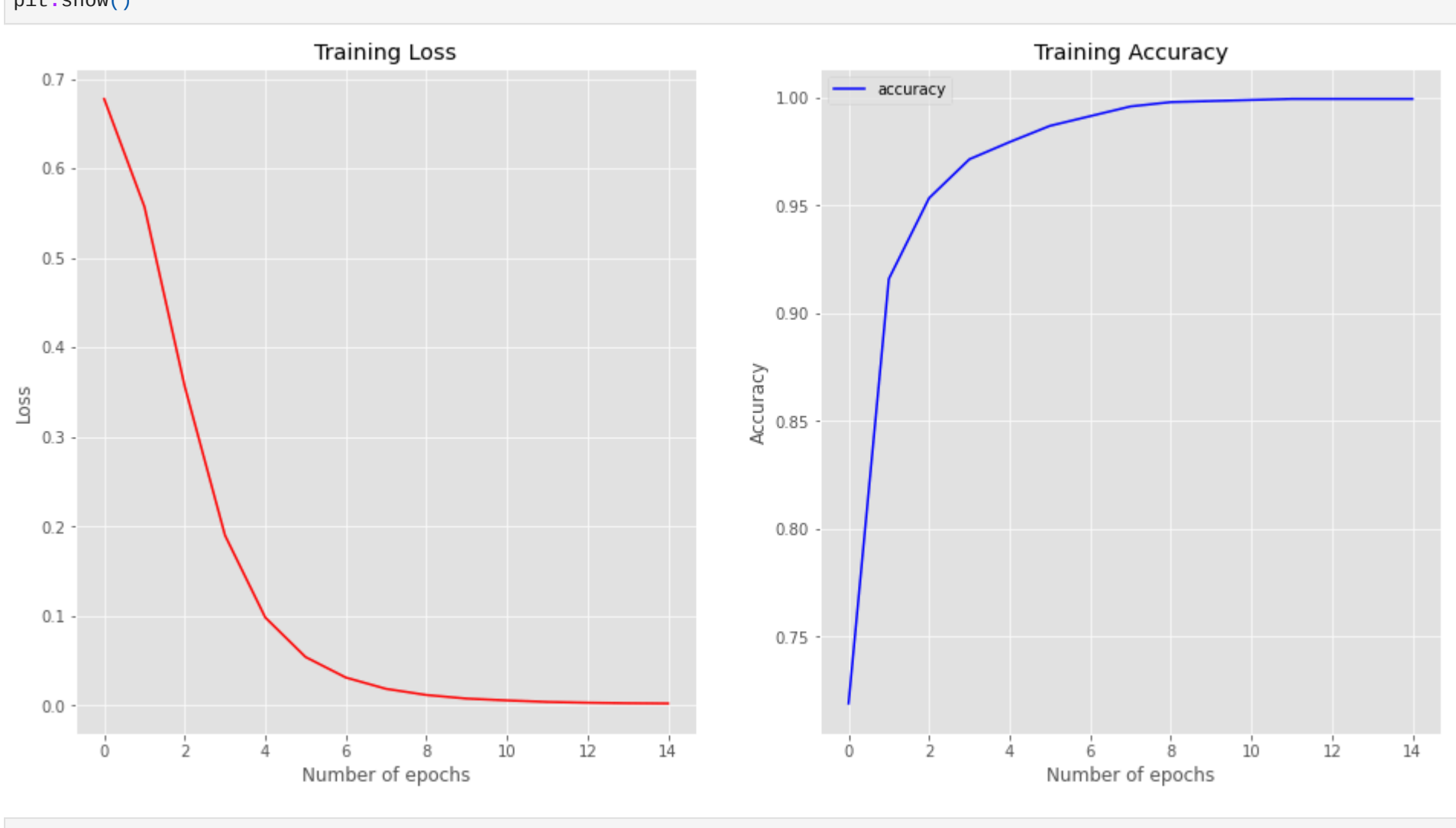
In [43]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                3547328

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 3,547,473
Trainable params: 3,547,473
Non-trainable params: 0
_________________________________________________________________
```

In [44]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
16/16 [==============================] - 0s 5ms/step - loss: 0.5546 - accuracy: 0.8320
Test loss: 0.5545622706413269
Test accuracy: 0.8320000171661377
```

In [45]:
```
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()
```



In [46]:
```
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'))
```

In [47]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [48]:
```
history = model.fit(x_train, y_train, batch_size=10, epochs=15)
```

```
Epoch 1/15
200/200 [==============================] - 2s 9ms/step - loss: 0.6563 - accuracy: 0.6155
Epoch 2/15
200/200 [==============================] - 2s 8ms/step - loss: 0.2878 - accuracy: 0.9540
Epoch 3/15
200/200 [==============================] - 2s 12ms/step - loss: 0.0427 - accuracy: 0.9970
Epoch 4/15
200/200 [==============================] - 3s 13ms/step - loss: 0.0112 - accuracy: 0.9995
Epoch 5/15
200/200 [==============================] - 3s 13ms/step - loss: 0.0044 - accuracy: 1.0000
Epoch 6/15
200/200 [==============================] - 2s 12ms/step - loss: 0.0021 - accuracy: 1.0000
Epoch 7/15
200/200 [==============================] - 3s 14ms/step - loss: 0.0012 - accuracy: 1.0000
Epoch 8/15
200/200 [==============================] - 2s 12ms/step - loss: 7.7711e-04 - accuracy: 1.0000
Epoch 9/15
200/200 [==============================] - 2s 12ms/step - loss: 5.2701e-04 - accuracy: 1.0000
Epoch 10/15
200/200 [==============================] - 2s 12ms/step - loss: 3.7625e-04 - accuracy: 1.0000
Epoch 11/15
200/200 [==============================] - 3s 14ms/step - loss: 2.7819e-04 - accuracy: 1.0000
Epoch 12/15
200/200 [==============================] - 3s 14ms/step - loss: 2.1216e-04 - accuracy: 1.0000
Epoch 13/15
200/200 [==============================] - 3s 15ms/step - loss: 1.6541e-04 - accuracy: 1.0000
Epoch 14/15
200/200 [==============================] - 3s 13ms/step - loss: 1.3109e-04 - accuracy: 1.0000
Epoch 15/15
200/200 [==============================] - 3s 14ms/step - loss: 1.0546e-04 - accuracy: 1.0000
```

In [49]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 16)                3547328

 dense_4 (Dense)             (None, 8)                 136

 dense_5 (Dense)             (None, 1)                 9

=================================================================
Total params: 3,547,473
Trainable params: 3,547,473
Non-trainable params: 0
_________________________________________________________________
```

In [50]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
16/16 [==============================] - 0s 7ms/step - loss: 0.4168 - accuracy: 0.8460
Test loss: 0.41683560609817505
Test accuracy: 0.8460000157356262
```

In [51]:
```
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()
```