Dataset

https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

In [1]:
```
import numpy as np
import pandas as pd
import re
import nltk
import seaborn as sns
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,classification_report as cr
```

In [2]:
```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[2]:
```
True
```

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

In [4]:
```
movie_reviews
```

Out[4]:

	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
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

In [5]:
```
# Looking for one review example
movie_reviews['review'].loc[49995]
```

Out[5]:
```
"I thought this movie did a down right good job. It wasn't as creative or original as th
e first, but who was expecting it to be. It was a whole lotta fun. the more i think abou
t it the more i like it, and when it comes out on DVD I'm going to pay the money for it
very proudly, every last cent. Sharon Stone is great, she always is, even if her movie i
s horrible(Catwoman), but this movie isn't, this is one of those movies that will be und
errated for its lifetime, and it will probably become a classic in like 20 yrs. Don't wa
it for it to be a classic, watch it now and enjoy it. Don't expect a masterpiece, or som
ething thats gripping and soul touching, just allow yourself to get out of your life and
get yourself involved in theirs.<br /><br />All in all, this movie is entertaining and i
recommend people who haven't seen it see it, because what the critics and box office say
doesn't always count, see it for yourself, you never know, you might just enjoy it. I ti
p my hat to this movie<br /><br />8/10"
```

In [ ]:
```
import nltk
import ssl

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

nltk.download()
```

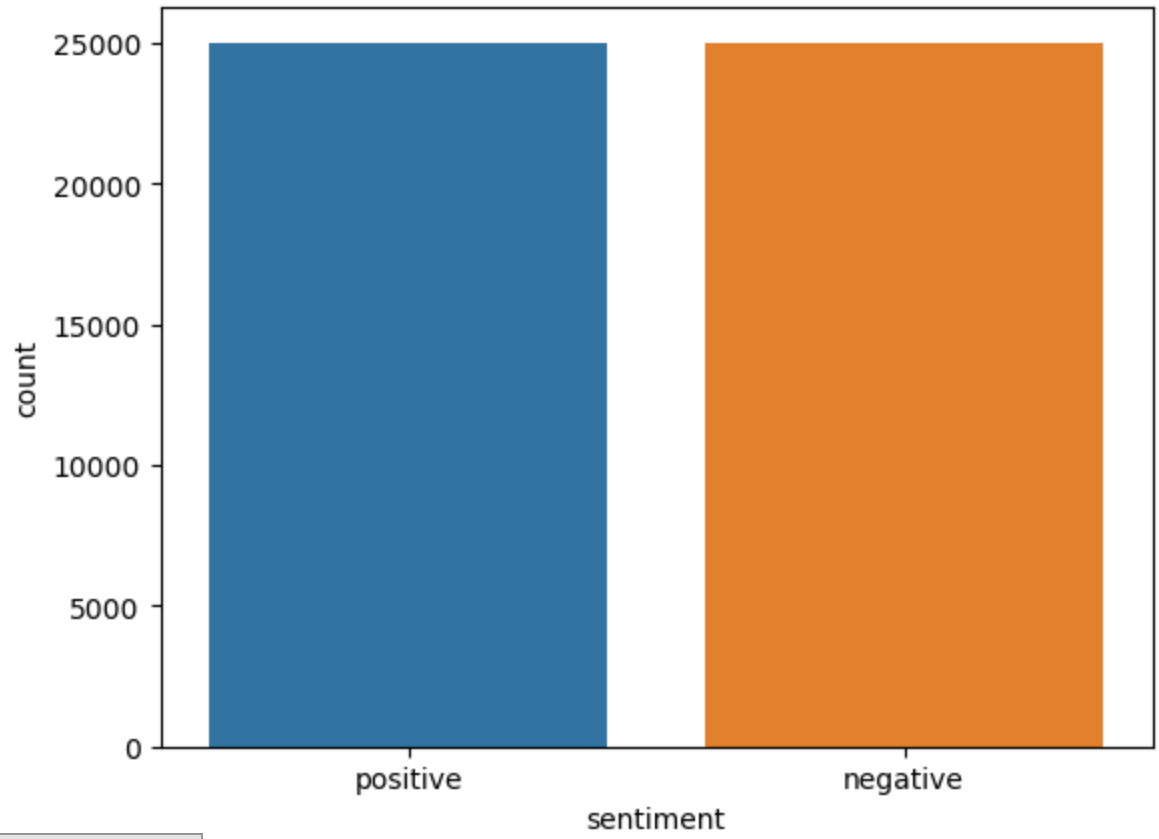
In [6]:
```
# Downloading Stop Words
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[6]:
```
True
```

In [7]:
```
sns.countplot(x='sentiment',data=movie_reviews)
```

Out[7]:
```
<Axes: xlabel='sentiment', ylabel='count'>
```



Implementing Stemming and removing stop words in the reviews

In [8]:
```
# Calling portstemmer for the purpose of stemming the words in review
ps = PorterStemmer()
```

In [9]:
```
corpus = []
for i in range(0,len(movie_reviews)):
    review = re.sub('[^a-zA-Z]',' ',movie_reviews['review'][i])
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    corpus.append(review)
```

Applying BagOfWords (BOW) Method

In [10]:
```
# Creating a bag of words model
cv = CountVectorizer(max_features=2500)
x = cv.fit_transform(corpus).toarray()
```

In [11]:
```
# Converting the categorical values into dummy variables
y = pd.get_dummies(movie_reviews['sentiment'])
y = y.iloc[:,1].values
```

In [12]:
```
# spliting the data for traing and testing
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=0)
```

Navie Bayes

In [13]:
```
#Creating navie bayes model
text_analysis_model = MultinomialNB().fit(X_train,y_train)
```

In [14]:
```
# prediction
y_pred = text_analysis_model.predict(X_test)
```

In [15]:
```
# getting accuracy score for test data and predicted data
score = accuracy_score(y_test,y_pred)
print(score)
print(cr(y_test,y_pred))
```

```
0.83872
              precision    recall  f1-score   support

           0       0.84      0.84      0.84      6291
           1       0.84      0.84      0.84      6209

    accuracy                           0.84     12500
   macro avg       0.84      0.84      0.84     12500
weighted avg       0.84      0.84      0.84     12500
```

Implementation of Lemmatization and removing stop words in the reviews

In [16]:
```
lemmatizer=WordNetLemmatizer()
```

In [17]:
```
corpus = []
for i in range(0, len(movie_reviews)):
    review = re.sub('[^a-zA-Z]', ' ', movie_reviews['review'][i])
    review = review.lower()
    review = review.split()

    review = [lemmatizer.lemmatize(word) for word in review if not word in stopwords.wor
    review = ' '.join(review)
    corpus.append(review)
```

Applying BagOfWords (BOW)

In [18]:
```
# Creating Bag of Words model
cv = CountVectorizer(max_features=2500)
x = cv.fit_transform(corpus).toarray()
```

In [19]:
```
# Converting the categorical values into dummy variables
y = pd.get_dummies(movie_reviews['sentiment'])
y = y.iloc[:,1].values
```

In [20]:
```
# spliting the data for traing and testing
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=0)
```

Naive Bayes

In [21]:
```
text_analysis_model2 = MultinomialNB().fit(X_train,y_train)
```

In [22]:
```
y_prediction = text_analysis_model2.predict(X_test)
```

In [23]:
```
score = accuracy_score(y_test,y_prediction)
print(score)
print(cr(y_test,y_prediction))
```

```
0.8404
              precision    recall  f1-score   support

           0       0.84      0.84      0.84      6291
           1       0.84      0.84      0.84      6209

    accuracy                           0.84     12500
   macro avg       0.84      0.84      0.84     12500
weighted avg       0.84      0.84      0.84     12500
```

Implementing Tf-Idf for Lemmatization

In [24]:
```
# Creating a TFIDF model
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer(max_features=2500)
X = tv.fit_transform(corpus).toarray()
```

In [25]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state
```

Naive bayes for Tf-Idf

In [26]:
```
text_analysis_model2 = MultinomialNB().fit(X_train,y_train)
```

In [27]:
```
y_predtf = text_analysis_model2.predict(X_test)
```

In [28]:
```
score=accuracy_score(y_test,y_predtf)
print(score)
print(cr(y_test,y_predtf))
```

```
0.8471
              precision    recall  f1-score   support

           0       0.86      0.84      0.85      5035
           1       0.84      0.86      0.85      4965

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000
```

Results of Bag Of Words(BOW)

Accuracy

Stemming Naive-Bayes - 83.87% ~ (84%)

Lemmatization Naive-Bayes - 84.04% ~ (84%)

Results of TF-IDF

Lemmatization Naive-Bayes - 84.71% ~ (85%)