```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
```

```
data=pd.read_excel("imdb_data.xlsx")
data.head()
```

	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

```
#checking for NAs
data.isnull().sum()
```

```
review       0
sentiment    0
dtype: int64
```

```
#converting to lowercase
data['review'] = data['review'].str.lower()
data.head()
```

	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

```
data['sentiment'].unique()
```

```
array(['positive', 'negative'], dtype=object)
```

```
#converting 'positive' to 1, 'negative' to 0
data['sentiment'].replace({'positive':1, 'negative':0}, inplace=True)
data.head()
```

	review	sentiment
0	one of the other reviewers has mentioned that ...	1
1	a wonderful little production. the...	1
2	i thought this was a wonderful way to spend ti...	1
3	basically there's a family where a little boy ...	0
4	petter mattei's "love in the time of money" is...	1

```
#removing html markups, eg <br>...</br>
import re
def remove_bracket_text(text):
```

```
    return re.sub('\<.*?\>', '', text)

data['review'] = data['review'].apply(remove_bracket_text)
data.head()
```

	review	sentiment
0	one of the other reviewers has mentioned that ...	1
1	a wonderful little production. the filming tec...	1
2	i thought this was a wonderful way to spend ti...	1
3	basically there's a family where a little boy ...	0
4	petter mattei's "love in the time of money" is...	1

```
#removing special characters
def remove_sp_char(text):
    return re.sub(r'[^a-zA-z0-9\s]', '', text)

data['review']=data['review'].apply(remove_sp_char)
data.head(10)
```

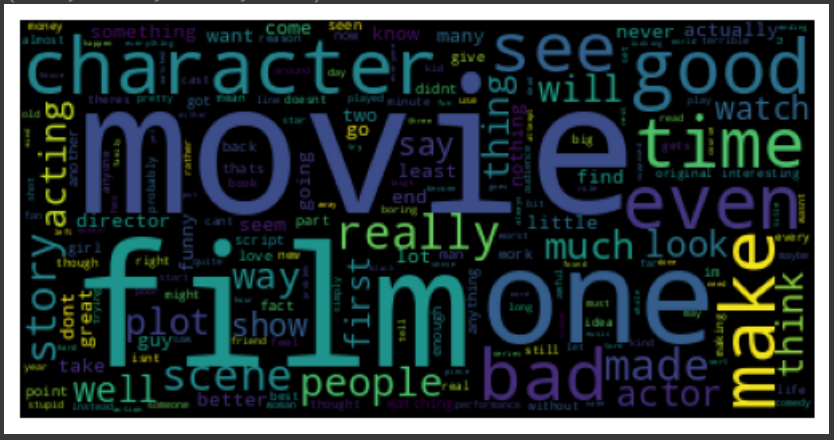
	review	sentiment
0	one of the other reviewers has mentioned that ...	1
1	a wonderful little production the filming tech...	1
2	i thought this was a wonderful way to spend ti...	1
3	basically theres a family where a little boy j...	0
4	petter matteis love in the time of money is a ...	1
5	probably my alltime favorite movie a story of ...	1
6	i sure would like to see a resurrection of a u...	1
7	this show was an amazing fresh innovative ide...	0
8	encouraged by the positive comments about this...	0
9	if you like original gut wrenching laughter yo...	1

```
pos = data[data['sentiment'] == 1]
neg = data[data['sentiment'] == 0]

#generating word cloud for positive reviews
from wordcloud import WordCloud
text = " ".join(pos['review'])
wc = WordCloud().generate(text)
plt.imshow(wc)
plt.axis('off')
```

```
(-0.5, 399.5, 199.5, -0.5)
```

```
#generating word cloud for negative reviews
text = " ".join(neg['review'])
wc = WordCloud().generate(text)
plt.imshow(wc)
plt.axis('off')
```

```
(-0.5, 399.5, 199.5, -0.5)
```




```
#TEXT PREPROCESSING

#tokenization
data['review'] = data['review'].str.split()
data.head(15)
```

	review	sentiment
0	[one, of, the, other, reviewers, has, mentione...	1
1	[a, wonderful, little, production, the, filmin...	1
2	[i, thought, this, was, a, wonderful, way, to,...	1
3	[basically, theres, a, family, where, a, littl...	0
4	[petter, matteis, love, in, the, time, of, mon...	1
5	[probably, my, alltime, favorite, movie, a, st...	1
6	[i, sure, would, like, to, see, a, resurrectio...	1
7	[this, show, was, an, amazing, fresh, innovati...	0
8	[encouraged, by, the, positive, comments, abou...	0
9	[if, you, like, original, gut, wrenching, laug...	1
10	[phil, the, alien, is, one, of, those, quirky,...	0
11	[i, saw, this, movie, when, i, was, about, 12,...	0
12	[so, im, not, a, big, fan, of, bolls, work, bu...	0
13	[the, cast, played, shakespeareshakespeare, lo...	0
14	[this, a, fantastic, movie, of, three, prisone...	1

```
#removing redundant words like 'movie' and 'film'
def rem(text):
  no_use_words = ['movies','films','movie','film']
  return [word for word in text if word not in no_use_words]

data['review'] = data['review'].apply(rem)
data.head(15)
```

	review	sentiment
0	[one, of, the, other, reviewers, has, mentione...	1
1	[a, wonderful, little, production, the, filmin...	1
2	[i, thought, this, was, a, wonderful, way, to,...	1
3	[basically, theres, a, family, where, a, littl...	0
4	[petter, matteis, love, in, the, time, of, mon...	1
5	[probably, my, alltime, favorite, a, story, of...	1
6	[i, sure, would, like, to, see, a, resurrectio...	1
7	[this, show, was, an, amazing, fresh, innovati...	0
8	[encouraged, by, the, positive, comments, abou...	0
9	[if, you, like, original, gut, wrenching, laug...	1
10	[phil, the, alien, is, one, of, those, quirky,...	0
11	[i, saw, this, when, i, was, about, 12, when, ...	0
12	[so, im, not, a, big, fan, of, bolls, work, bu...	0

```
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words[1:10]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
['me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

```
#removing stopwords
def stop(text):
  return [word for word in text if word not in stop_words]

data['review'] = data['review'].apply(stop)
data.head(10)
```

	review	sentiment
0	[one, reviewers, mentioned, watching, 1, oz, e...	1
1	[wonderful, little, production, filming, techn...	1
2	[thought, wonderful, way, spend, time, hot, su...	1
3	[basically, theres, family, little, boy, jake,...	0
4	[petter, matteis, love, time, money, visually,...	1
5	[probably, alltime, favorite, story, selflessn...	1
6	[sure, would, like, see, resurrection, dated, ...	1
7	[show, amazing, fresh, innovative, idea, 70s, ...	0
8	[encouraged, positive, comments, looking, forw...	0
9	[like, original, gut, wrenching, laughter, lik...	1

```
#lemmatizing
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

def lemmit_it(text):
  return [lemmatizer.lemmatize(word, pos='v') for word in text]
```

```
data['review'] = data['review'].apply(lambda x: ' '.join(lemmit_it(x)))
data.head(15)
```

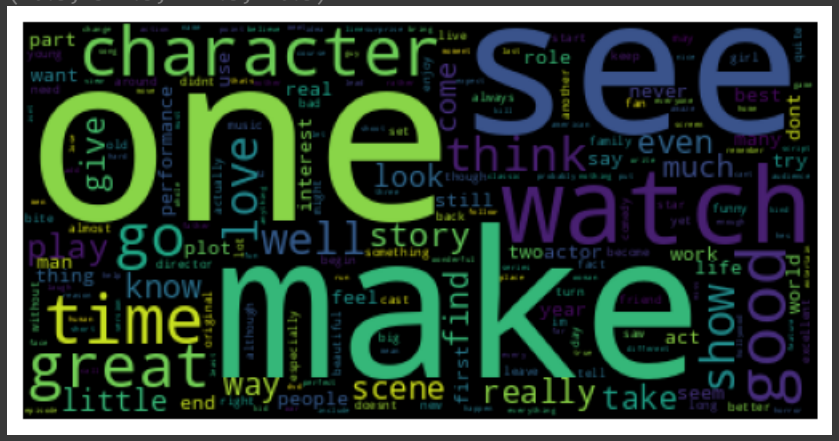
```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

	review	sentiment
0	one reviewers mention watch 1 oz episode youll...	1
1	wonderful little production film technique una...	1
2	think wonderful way spend time hot summer week...	1
3	basically theres family little boy jake think ...	0
4	petter matteis love time money visually stun w...	1
5	probably alltime favorite story selflessness s...	1
6	sure would like see resurrection date seahunt ...	1
7	show amaze fresh innovative idea 70s first air...	0
8	encourage positive comment look forward watch ...	0
9	like original gut wrench laughter like young o...	1
10	phil alien one quirky humour base around oddne...	0
11	saw 12 come recall scariest scene big bird eat...	0
12	im big fan bolls work many enjoy postal maybe ...	0
13	cast play shakespeareshakespeare losti appreci...	0
14	fantastic three prisoners become famous one ac...	1

```
#stemming
#from nltk.stem import PorterStemmer
#ps = PorterStemmer()

#def stemmer(text):
  #return [ps.stem(word) for word in text]
#data['review'] = data['review'].apply(stemmer)
#data.head(15)
```

```
pos = data[data['sentiment'] == 1]
neg = data[data['sentiment'] == 0]

#generating word cloud for positive reviews after text pre-processing
from wordcloud import WordCloud
text = " ".join(pos['review'])
wc = WordCloud().generate(text)
plt.imshow(wc)
plt.axis('off')
```

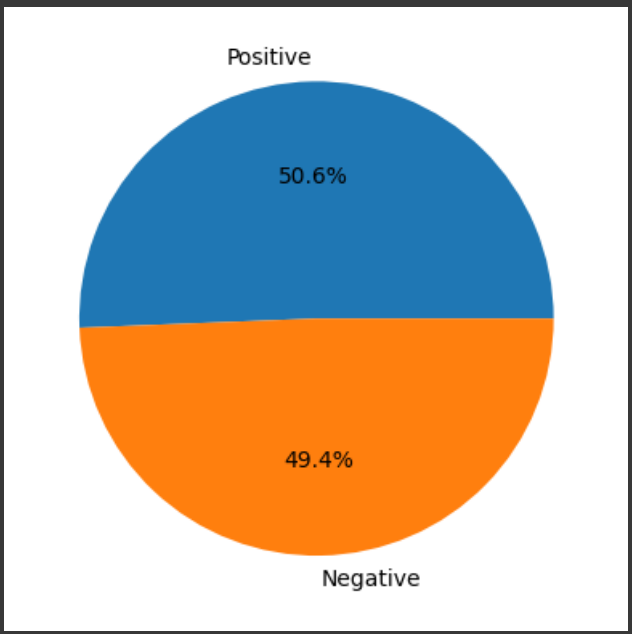
```
(-0.5, 399.5, 199.5, -0.5)
```



```
#generating word cloud for negative reviews after text pre-processing
text = " ".join(neg['review'])
wc = WordCloud().generate(text)
plt.imshow(wc)
plt.axis('off')
```

```
(-0.5, 399.5, 199.5, -0.5)
```


```
#count of positive reviews vs negative reviews
x=[pos['sentiment'].count(), neg['sentiment'].count()]
pie = plt.pie(x, labels = ["Positive", "Negative"], autopct ='%1.1f%%')
```




```
#creating test and train datasets
nrows,ncols = data.shape
train_data = data.head(int(nrows*0.75))
test_data = data.tail(int(nrows*0.25))
```

```
from sklearn.metrics import accuracy_score, confusion_matrix

#to print accuracy and plot confusion matrix
def evaluate(y_true, y_pred):
  accuracy = round(accuracy_score(y_true, y_pred)*100, 2)
  cm = confusion_matrix(y_true, y_pred)
  print("Accuracy: ",accuracy,"%")
  print("Confusion Matrix:\n",cm)
  sns.heatmap(cm, annot=True, fmt='g')
```

```
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
True
```

```
#SIA uses bag-of-words approach
sia = SentimentIntensityAnalyzer()
print(sia.polarity_scores('I am so happy!'))
print(sia.polarity_scores('This is the worst thing ever.'))

res=[]
for text in data['review']:
  res.append(sia.polarity_scores(text))
```

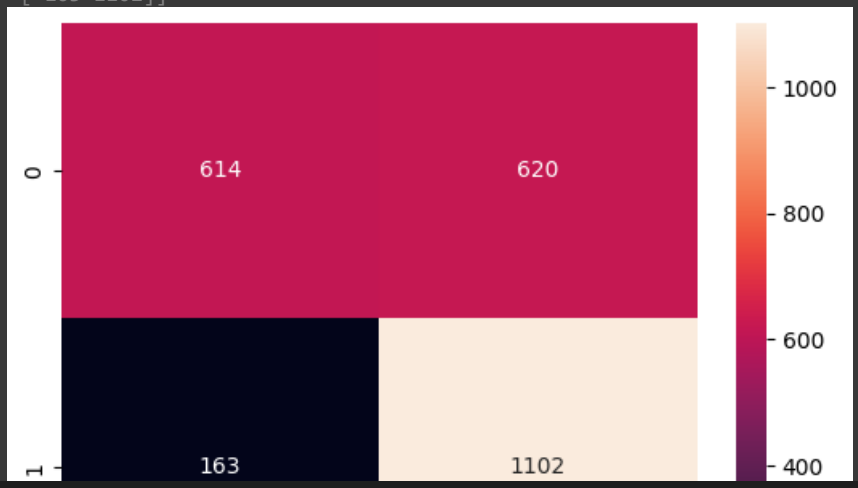
```
{'neg': 0.0, 'neu': 0.318, 'pos': 0.682, 'compound': 0.6468}
{'neg': 0.451, 'neu': 0.549, 'pos': 0.0, 'compound': -0.6249}
```

```
rating=[]
for x in res:
  rating.append(1) if x['compound'] >= 0 else rating.append(0)

data["score"] = rating
data.head(20)
```

	review	sentiment	score
0	one reviewers mention watch 1 oz episode youll...	1	0
1	wonderful little production film technique una...	1	1
2	think wonderful way spend time hot summer week...	1	1
3	basically theres family little boy jake think ...	0	0
4	petter matteis love time money visually stun w...	1	1
5	probably alltime favorite story selflessness s...	1	1
6	sure would like see resurrection date seahunt ...	1	1
7	show amaze fresh innovative idea 70s first air...	0	1
8	encourage positive comment look forward watch ...	0	0
9	like original gut wrench laughter like young o...	1	1
10	phil alien one quirky humour base around oddne...	0	1
11	saw 12 come recall scariest scene big bird eat...	0	1
12	im big fan bolls work many enjoy postal maybe ...	0	1
13	cast play shakespeareshakespeare losti appreci...	0	1
14	fantastic three prisoners become famous one ac...	1	0
15	kind draw erotic scenes realize one amateurish...	0	1
16	simply remake one bad fail capture flavor terr...	1	0
17	make one top 10 awful horrible wasnt continuou...	0	0
18	remember filmit first watch cinema picture dar...	1	0
19	awful must real stinkers nominate golden globe...	0	1

```
evaluate(data['sentiment'], data['score'])
```

```
Accuracy:  68.67 %
Confusion Matrix:
 [[ 614  620]
 [ 163 1102]]
```

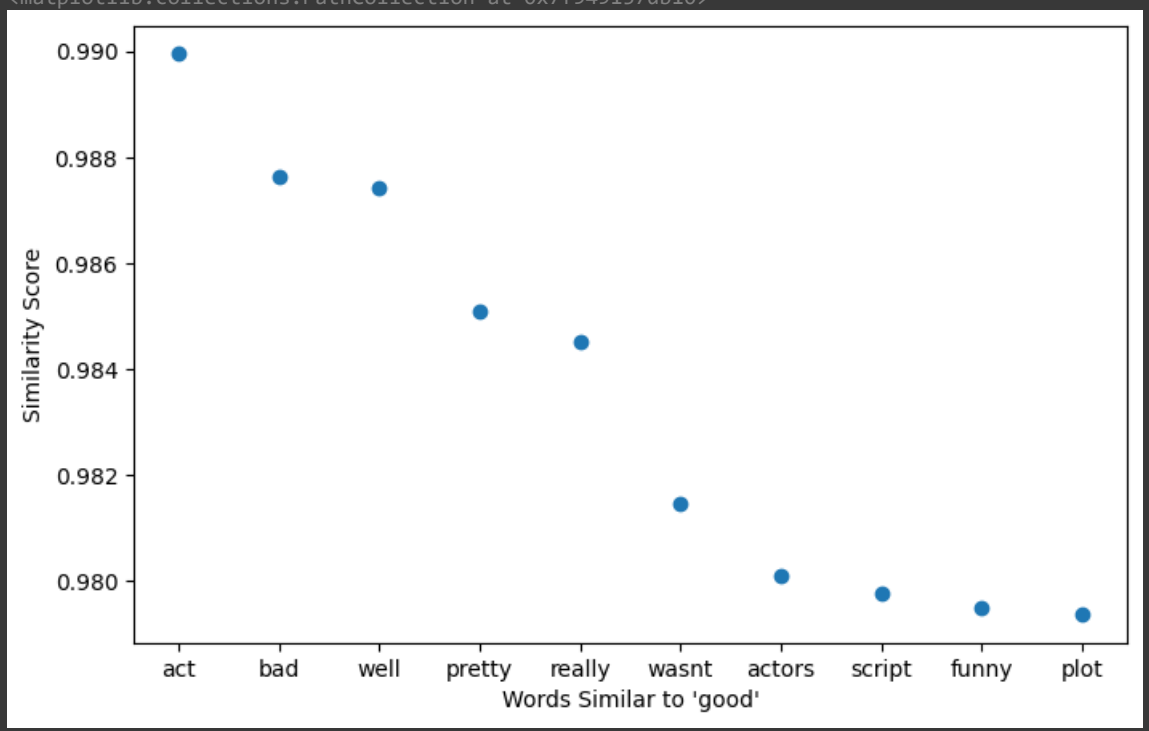



```
#FEATURE EXTRACTION

from gensim.test.utils import common_texts
from gensim.models import Word2Vec

#text vectorization using Word2Vec
x=data['review'].str.split()
w2v = Word2Vec(x, min_count=1,vector_size = 15, sg=0)
print(w2v.wv['good'])   #vector representation of any word
```

```
[-1.9865783   2.0174809   2.9513755  -3.8298001   1.3652021  -0.4953779
  0.78928983  1.3122038   0.35300064 -0.13436587  3.6621485   1.8769405
  2.207517   -0.9213547  -3.371813  ]
```

```
print(w2v.wv.similarity('good', 'like'))   #similarity between two words
print(w2v.wv.most_similar('good', topn=10))   #get other similar words
plt.figure(figsize=(8,5))
plt.xlabel("Words Similar to 'good'")
plt.ylabel("Similarity Score")
plt.scatter(*zip(*w2v.wv.most_similar('good', topn=10)))
```

```
0.97591144
[('act', 0.989953339099884), ('bad', 0.9876246452331543), ('well', 0.9874046444892883), ('pretty', 0.9850807189941406), ('r
<matplotlib.collections.PathCollection at 0x7f949157db10>
```



```
#text vectorization using Count Vectorizer (bag-of-words)
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
vectorizer = CountVectorizer()

cv_train = vectorizer.fit_transform(train_data['review'])
print(vectorizer.get_feature_names_out())
print('CV_train:',cv_train.shape)

df = pd.DataFrame(data=cv_train.toarray(),columns = vectorizer.get_feature_names_out())
print(df)
```

```
['007' '02' '0510' ... 'zwart' 'zwick' 'zzzzzzzzzzzzzzzzzz']
CV_train: (1874, 26245)
      007  02  0510  06  0clock  0s  10  100  1000  10000  ...  zu  zuber  \
0       0   0     0   0       0   0   0    0     0      0  ...   0      0
1       0   0     0   0       0   0   0    0     0      0  ...   0      0
2       0   0     0   0       0   0   0    0     0      0  ...   0      0
3       0   0     0   0       0   0   1    0     0      0  ...   0      0
4       0   0     0   0       0   0   0    0     0      0  ...   0      0
...   ...  ..   ...  ..     ...  ..  ..  ...   ...    ...  ...  ..    ...
1869    0   0     0   0       0   0   0    0     0      0  ...   0      0
1870    0   0     0   0       0   0   0    0     0      0  ...   0      0
1871    0   0     0   0       0   0   0    0     0      0  ...   0      0
1872    0   0     0   0       0   0   0    0     0      0  ...   0      0
1873    0   0     0   0       0   0   0    0     0      0  ...   0      0

      zucker  zues  zulu  zuthe  zuwarriors  zwart  zwick  zzzzzzzzzzzzzzzzzz
0          0     0     0      0           0      0      0                   0
1          0     0     0      0           0      0      0                   0
2          0     0     0      0           0      0      0                   0
3          0     0     0      0           0      0      0                   0
4          0     0     0      0           0      0      0                   0
...      ...   ...   ...    ...         ...    ...    ...                 ...
1869       0     0     0      0           0      0      0                   0
1870       0     0     0      0           0      0      0                   0
1871       0     0     0      0           0      0      0                   0
1872       0     0     0      0           0      0      0                   0
1873       0     0     0      0           0      0      0                   0

[1874 rows x 26245 columns]
```

```
cv_test = vectorizer.transform(test_data['review'])
print('CV_test:',cv_test.shape)
df = pd.DataFrame(data=cv_test.toarray(),columns = vectorizer.get_feature_names_out())
print(df[["like","love","good","best","bad","dumb","hate","disappoint"]])
```

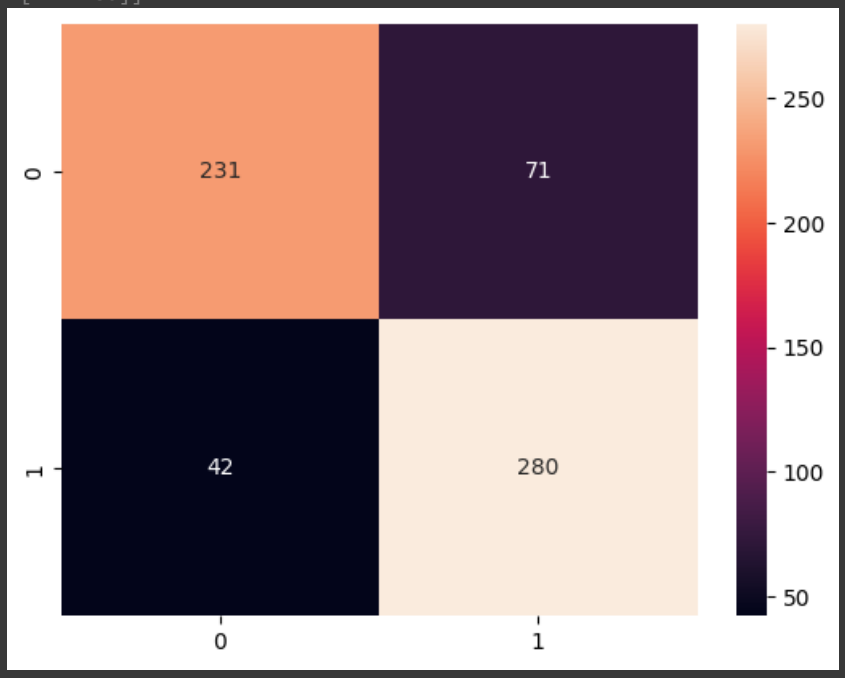
```
CV_test: (624, 26245)
     like  love  good  best  bad  dumb  hate  disappoint
0       1     0     0     0    0     0     0           0
1       0     0     0     0    0     1     0           0
2       0     1     0     0    1     0     0           0
3       1     1     0     1    0     0     0           0
4       0     0     0     0    1     0     0           0
..    ...   ...   ...   ...  ...   ...   ...         ...
619     1     0     0     1    0     0     0           0
620     1     0     0     0    0     0     0           0
621     0     0     1     0    0     0     0           0
622     0     2     0     0    0     0     0           0
623     3     0     1     0    0     0     0           0

[624 rows x 8 columns]
```

```
#text vectorization using TF-IDF

tv=TfidfVectorizer(min_df=0,max_df=1, use_idf=True, ngram_range=(1,3))
tv_train=tv.fit_transform(train_data['review'])
tv_test=tv.transform(test_data['review'])
```

```
#modelling CV using Logistic Regression
from sklearn.linear_model import LogisticRegression
```

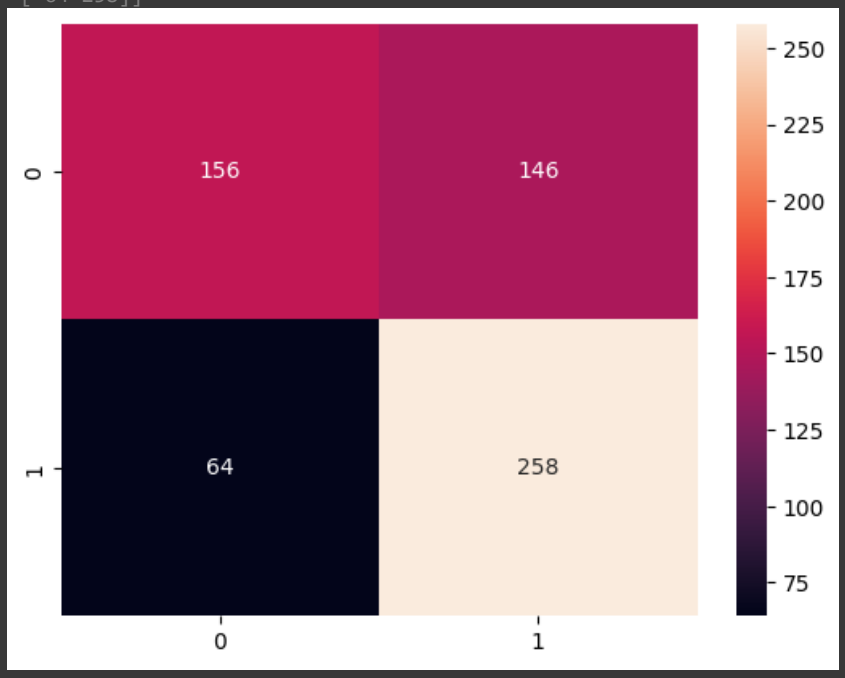
```
lr = LogisticRegression()
lr.fit(cv_train,train_data['sentiment'])
pred = lr.predict(cv_test)
print(pred[1:20])

evaluate(test_data['sentiment'], pred)
```

```
[0 0 1 0 0 0 0 0 1 1 1 1 0 1 1 0 1 1 1]
Accuracy:  81.89 %
Confusion Matrix:
 [[231  71]
 [ 42 280]]
```



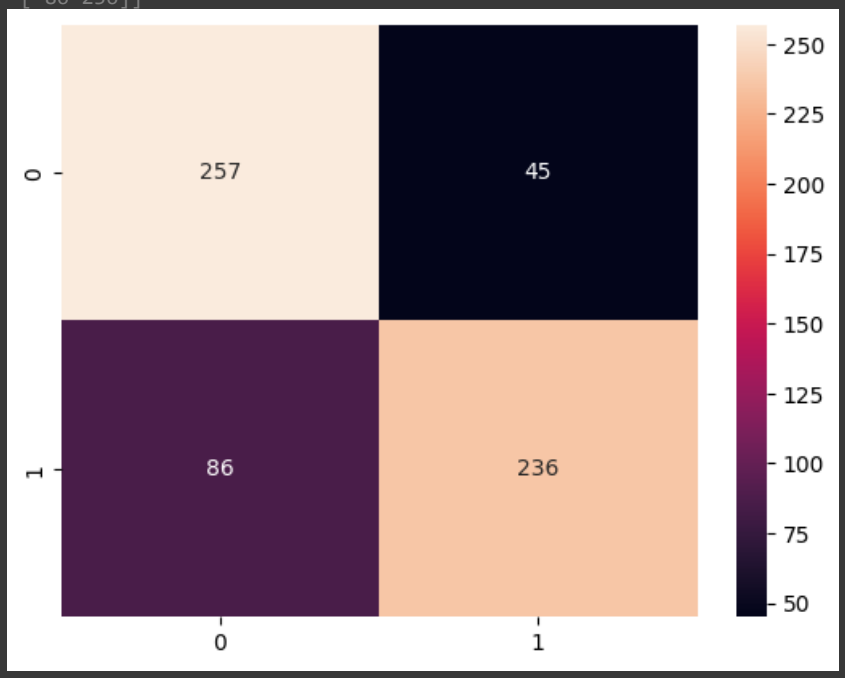

```
#modelling TF-IDF using Logistic Regression
lr.fit(tv_train,train_data['sentiment'])
pred = lr.predict(tv_test)

evaluate(test_data['sentiment'], pred)
```

```
Accuracy:  66.35 %
Confusion Matrix:
 [[156 146]
 [ 64 258]]
```



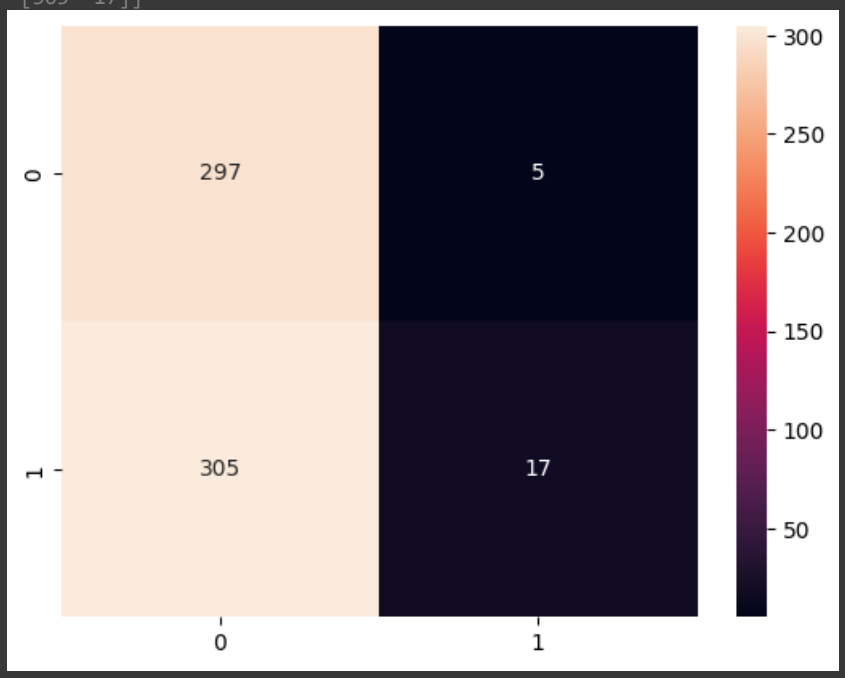
```
#modelling CV using Bernoulli's Naive Bayes
from sklearn.naive_bayes import BernoulliNB
bnb = BernoulliNB(alpha = 2)
bnb.fit(cv_train,train_data['sentiment'])
pred = bnb.predict(cv_test)

evaluate(test_data['sentiment'], pred)
```

```
Accuracy:  79.01 %
Confusion Matrix:
 [[257  45]
 [ 86 236]]
```



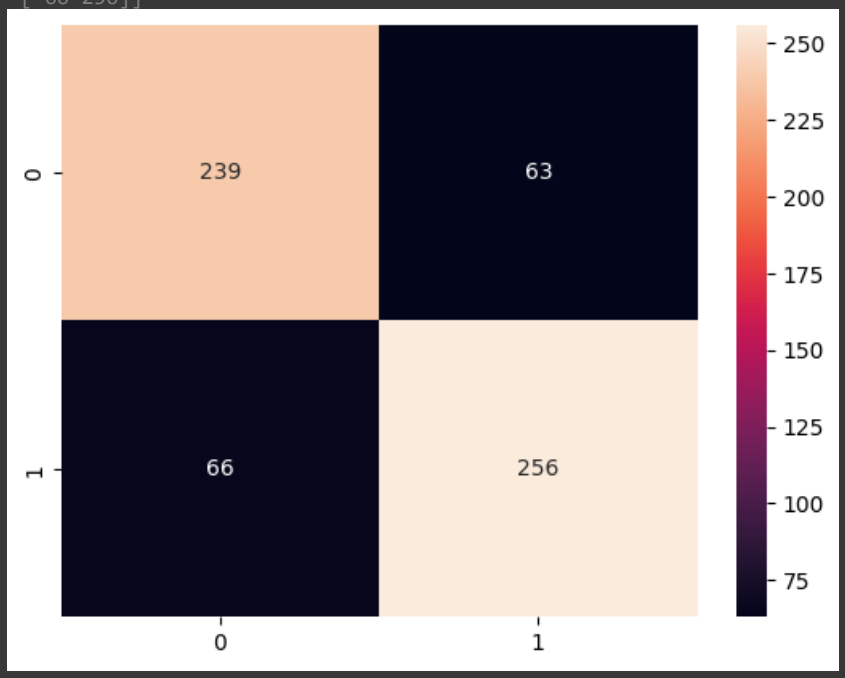

```
#modelling TF-IDF using Bernoulli's Naive Bayes
bnb.fit(tv_train,train_data['sentiment'])
pred = bnb.predict(tv_test)

evaluate(test_data['sentiment'], pred)
```

```
Accuracy:  50.32 %
Confusion Matrix:
 [[297   5]
 [305  17]]
```



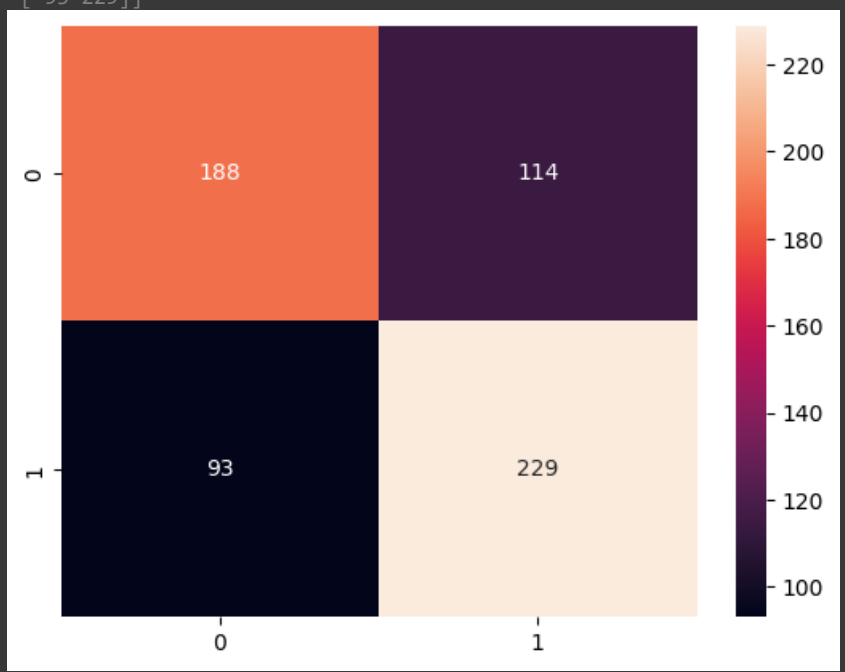
```
#modelling CV using Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()
mnb.fit(cv_train,train_data['sentiment'])
pred = mnb.predict(cv_test)

evaluate(test_data['sentiment'], pred)
```

```
Accuracy:  79.33 %
Confusion Matrix:
 [[239  63]
 [ 66 256]]
```



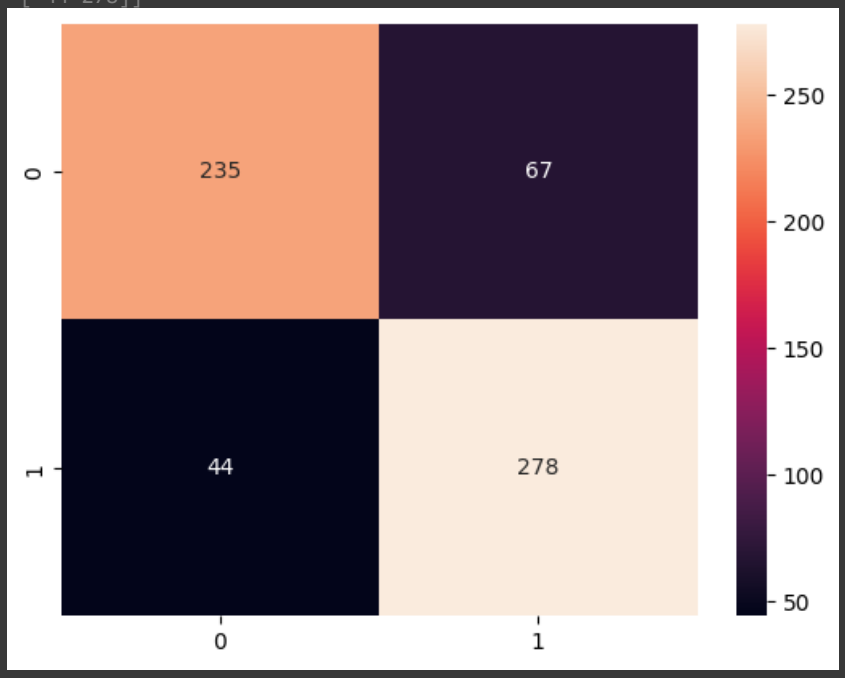

```
#modelling TF-IDF using Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()
mnb.fit(tv_train,train_data['sentiment'])
pred = mnb.predict(tv_test)

evaluate(test_data['sentiment'], pred)
```

```
Accuracy:  66.83 %
Confusion Matrix:
 [[188 114]
 [ 93 229]]
```



```
#modelling CV using Linear SVM
from sklearn.svm import SVC
svc = SVC(kernel='linear')
svc.fit(cv_train,train_data['sentiment'])
pred = svc.predict(cv_test)

evaluate(test_data['sentiment'], pred)
```

```
Accuracy:  82.21 %
Confusion Matrix:
 [[235  67]
 [ 44 278]]
```




```
#modelling TF-IDF using Linear SVM
svc.fit(tv_train,train_data['sentiment'])
pred = svc.predict(tv_test)

evaluate(test_data['sentiment'], pred)
```

```
Accuracy:  66.35 %
Confusion Matrix:
 [[184 118]
 [ 92 230]]
```