In [1]:
```
import pandas as pd
```

EXERCISE-1

1. Open the file, 'rotten_tomato_train.tsv' and read into a DataFrame

In [2]:
```
rotten_tomato_train = pd.read_csv('rotten_tomato_train.tsv', sep='\t')
```

In [3]:
```
rotten_tomato_train.head()
```

Out[3]:

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

2. Print the basic statistics such as head, shape, describe, and columns

In [4]:
```
rotten_tomato_train.tail()
```

Out[4]:

	PhraseId	SentenceId	Phrase	Sentiment
156055	156056	8544	Hearst 's	2
156056	156057	8544	forced avuncular chortles	1
156057	156058	8544	avuncular chortles	3
156058	156059	8544	avuncular	2
156059	156060	8544	chortles	2

In [5]:
```
rotten_tomato_train.shape
```

Out[5]:
```
(156060, 4)
```

In [6]:
```
rotten_tomato_train.describe
```

Out[6]:
```
<bound method NDFrame.describe of        PhraseId  SentenceId  \
0             1           1
1             2           1
2             3           1
3             4           1
4             5           1
...         ...         ...
156055    156056        8544
156056    156057        8544
156057    156058        8544
156058    156059        8544
156059    156060        8544

                                                   Phrase  Sentiment
0       A series of escapades demonstrating the adage ...          1
1       A series of escapades demonstrating the adage ...          2
2                                               A series          2
3                                                      A          2
4                                                 series          2
...                                                  ...        ...
156055                                         Hearst 's          2
156056                          forced avuncular chortles          1
156057                                 avuncular chortles          3
156058                                          avuncular          2
156059                                           chortles          2

[156060 rows x 4 columns]>
```

In [7]:
```
rotten_tomato_train.columns
```

Out[7]:
```
Index(['PhraseId', 'SentenceId', 'Phrase', 'Sentiment'], dtype='object')
```

3. How many reviews exist for each sentiment?

In [8]:
```
review=rotten_tomato_train.groupby('Sentiment').count()
review.Phrase
```

Out[8]:
```
Sentiment
0     7072
1    27273
2    79582
3    32927
4     9206
Name: Phrase, dtype: int64
```

EXERCISE-2

1. Extract 200 reviews for each sentiment, store them into a new dataframe and create a smaller dataset. Save this dataframe in a new

file, say, “small_rotten_train.csv”.

In [9]:
```
a=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 0]
b=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 1]
c=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 2]
d=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 3]
e=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 4]
```

In [10]:
```
small_rotten_train=pd.concat([a[:200],b[:200],c[:200],d[:200],e[:200]])
```

EXERCISE-3

1. Open the file, “small_rotten_train.csv”.

In [11]:
```
small_rotten_train
```

Out[11]:

	PhraseId	SentenceId	Phrase	Sentiment
101	102	3	would have a hard time sitting through this one	0
103	104	3	have a hard time sitting through this one	0
157	158	5	Aggressive self-glorification and a manipulati...	0
159	160	5	self-glorification and a manipulative whitewash	0
201	202	7	Trouble Every Day is a plodding mess .	0
...	...	...	...	...
3744	3745	142	amazing slapstick	4
3745	3746	142	amazing	4
3847	3848	147	When cowering and begging at the feet a scruff...	4
3866	3867	147	gives her best performance since Abel Ferrara ...	4
3993	3994	151	Spielberg 's realization of a near-future Amer...	4

1000 rows × 4 columns

2. The review text are stored in “Phrase” column. Extract that into a separate DataFrame, say “X”.

In [12]:
```
X = small_rotten_train.Phrase
```

3. The “sentiment” column is your target, say “y”.

In [13]:
```
y = small_rotten_train.Sentiment
```

4. Perform pre-processing: convert into lower case, remove stop words and lemmatize. The following function will help.

In [14]:
```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

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

In [15]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

In [16]:
```
def clean_review(review):
    tokens = review.lower().split()
    filtered_tokens = [lemmatizer.lemmatize(w) for w in tokens if w not in stop_v
    return " ".join(filtered_tokens)
```

5. Apply the above function to X

In [17]:
```
temp=X.tolist()
fax=[]
for i in temp:
    fax.append(clean_review(i))
n_X=pd.Series(fax)
```

6. Split X and y for training and testing (Use 20% for testing)

In [18]:
```
from sklearn.model_selection import train_test_split
```

In [19]:
```
X_train,X_test,y_train,y_test = train_test_split(n_X,y,train_size=0.8,test_size=0
```

7. Create TfidfVectorizer as below and perfrom vectorization on X_train using fit_perform() method.

In [20]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [21]:
```
tf=TfidfVectorizer(min_df=3, max_features=None,ngram_range=(1, 2), use_idf=1)
tf
```

Out[21]:
```
TfidfVectorizer(min_df=3, ngram_range=(1, 2), use_idf=1)
```

In [22]:
```
m=tf.fit_transform(X_train)
m.shape
```

Out[22]:
```
(800, 874)
```

8. Create MultinomialNB model and perform training using X_train_lemmartized and y_train.

In [23]:
```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

In [24]:
```
X_train_dtm = cv.fit_transform(X_train)
X_test_dtm = cv.transform(X_test)
```

In [25]:
```
from sklearn.naive_bayes import MultinomialNB
```

In [26]:
```
clf = MultinomialNB()
```

In [27]:
```
clf.fit(X_train_dtm,y_train)
```

Out[27]:
```
MultinomialNB()
```

9. Perform validation on X_test lemmatized and predict output

In [28]:
```
y_real_pred = clf.predict(X_test_dtm)
y_real_pred
```

Out[28]:
```
array([2, 4, 4, 0, 0, 4, 2, 2, 3, 3, 1, 3, 0, 3, 2, 4, 1, 1, 1, 2, 3, 4,
       3, 4, 3, 3, 4, 3, 1, 3, 1, 3, 4, 0, 3, 2, 0, 2, 1, 4, 1, 0, 3, 1,
       1, 1, 3, 3, 1, 0, 3, 1, 3, 3, 0, 0, 2, 1, 2, 2, 2, 4, 3, 3, 3, 3,
       0, 2, 1, 1, 4, 2, 3, 2, 3, 2, 4, 3, 1, 2, 4, 4, 3, 3, 1, 3, 4, 1,
       2, 3, 1, 3, 3, 3, 2, 3, 4, 4, 3, 2, 2, 1, 3, 3, 1, 2, 3, 2, 3, 1,
       2, 1, 3, 4, 2, 0, 4, 0, 4, 0, 4, 1, 0, 3, 1, 3, 1, 0, 3, 3, 0, 3,
       1, 0, 2, 0, 4, 3, 0, 3, 0, 2, 0, 3, 2, 0, 4, 1, 4, 2, 0, 1, 4, 3,
       2, 2, 2, 2, 0, 0, 3, 3, 2, 0, 3, 1, 4, 2, 1, 1, 0, 3, 1, 4, 0, 4,
       4, 2, 1, 2, 0, 0, 2, 1, 4, 3, 1, 4, 1, 3, 4, 1, 0, 3, 0, 1, 4, 3,
       2, 3], dtype=int64)
```

10. Print classification_report and accuracy score.

In [29]:
```
from sklearn.metrics import classification_report
```

In [30]:
```
print(classification_report(y_test,y_real_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.73      0.78        37
           1       0.65      0.59      0.62        44
           2       0.66      0.54      0.60        46
           3       0.44      0.76      0.56        33
           4       0.73      0.60      0.66        40

    accuracy                           0.64       200
   macro avg       0.66      0.64      0.64       200
weighted avg       0.67      0.64      0.64       200
```

In [31]:
```
from sklearn.metrics import accuracy_score
```

In [32]:
```
accuracy_score(y_test,y_real_pred)
```

Out[32]: 0.635

EXERCISE-4

In [33]:
```
rotten_tomato_test = pd.read_csv('rotten_tomato_test.tsv', sep='\t')
```

In [34]:
```
rotten_tomato_test.shape
```

Out[34]: (66292, 3)

In [35]:
```
X_ = rotten_tomato_test.Phrase
```

In [36]:
```
t_temp=X_.tolist()
t_fax=[]
for i in t_temp:
    t_fax.append(clean_review(i))
nt_X=pd.Series(t_fax)
```

In [37]:
```
nt_X
```

Out[37]:
```
0        intermittently pleasing mostly routine effort .
1          intermittently pleasing mostly routine effort
2                                                       
3          intermittently pleasing mostly routine effort
4                 intermittently pleasing mostly routine
                              ...
66287               long-winded , predictable scenario .
66288                 long-winded , predictable scenario
66289                                      long-winded ,
66290                                        long-winded
66291                                 predictable scenario
Length: 66292, dtype: object
```

In [38]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [39]:
```
tf2=TfidfVectorizer(use_idf=True,ngram_range=(1,3),min_df = 1)
tf2
```

Out[39]:
```
TfidfVectorizer(ngram_range=(1, 3))
```

In [41]:
```
my2=tf2.fit_transform(nt_X)
my2
```

Out[41]:
```
<66292x61283 sparse matrix of type '<class 'numpy.float64'>'
        with 571899 stored elements in Compressed Sparse Row format>
```

In [ ]:
```

```