In [1]:
```python
import csv
from bs4 import BeautifulSoup
import requests
import pandas as pd
import time
import numpy as np
time.sleep(2)
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
urls = []
url1 = 'https://www.imdb.com/title/tt0208092/reviews/?ref_=tt_ql_2'
url2 = 'https://www.imdb.com/title/tt0107207/reviews/?ref_=tt_ql_2'
url3 = 'https://www.imdb.com/title/tt0051792/reviews/?ref_=tt_ql_2'
url4 = 'https://www.imdb.com/title/tt2358592/reviews/?ref_=tt_ql_2'
url5 = 'https://www.imdb.com/title/tt0070579/reviews/?ref_=tt_ql_2'
url6 = 'https://www.imdb.com/title/tt1098327/reviews/?ref_=tt_ql_2'
url7 = 'https://www.imdb.com/title/tt1773764/reviews/?ref_=tt_ql_2'
```

In [3]:
```python
urls.append(url1)
urls.append(url2)
urls.append(url3)
urls.append(url4)
urls.append(url5)
urls.append(url6)
urls.append(url7)
```

In [4]:
```python
content = []
for url in urls:
    page = requests.get(url, timeout=2.50)
    page_content = page.content
    soup = BeautifulSoup(page_content, 'html.parser')
    content.append(soup.find_all('div', class_= 'review-container'))
```

In [5]:
```python
print(content)
```

```
[[<div class="review-container">
<div class="lister-item-content">
<div class="ipl-ratings-bar">
<span class="rating-other-user-rating">
<svg class="ipl-icon ipl-star-icon" fill="#000000" height="24" viewbox="0 0 24 24" width="24" xmlns="http://www.w3.
org/2000/svg">
<path d="M0 0h24v24H0z" fill="none"></path>
<path d="M12 17.27L18.18 21l-1.64-7.03L22 9.24l-7.19-.61L12 2 9.19 8.63 2 9.24l5.46 4.73L5.82 21z"></path>
<path d="M0 0h24v24H0z" fill="none"></path>
</svg>
<span>9</span><span class="point-scale">/10</span>
</span>
</div>
<a class="title" href="/review/rw0663117/"> There are few films that can make me laugh like this one can
</a> <div class="display-name-date">
<span class="display-name-link"><a href="/user/ur2339662/">FilmOtaku</a></span><span class="review-date">24 August
2004</span>
</div>
<div class="content">
```

In [6]:
```python
movie = pd.DataFrame(columns=['Review','Rating'])
```

In [7]:
```
review = []
rating = []
count = 0
for cc in content:
    for c in cc:
        count+= 1

        print('\nMovie review ', count)
        #Get review.
        str = c.find_all('a', attrs={'class':'title'})
        rReview =''
        for s in str:
            #print('Review is: ',s.get_text())
            rReview = s.get_text()

        #Get rating.
        ratings = c.find_all('span', attrs={'class':''})
        rVal = []
        for r in ratings:
            str1 = r.get_text().strip()
            rVal.append(str1)

        val = rVal[0]
        if(len(val) > 2):
            continue
        else:
            review.append(rReview)
            rating.append(val)
            print('Review: ', rReview)
            print('Rating: ',val)

movie['Review'] = review
movie['Rating'] = rating
```

```
Movie review  1
Review:   There are few films that can make me laugh like this one can

Rating:  9

Movie review  2
Review:   Perfect

Rating:  10

Movie review  3
Review:   Lock, Stock, and Many Smoking Barrels

Rating:  8

Movie review  4
Review:   The pinnacle of Guy Ritchie's career
```

In [8]:
```
movie.head()
```

Out[8]:

	Review	Rating
0	There are few films that can make me laugh li...	9
1	Perfect\n	10
2	Lock, Stock, and Many Smoking Barrels\n	8
3	The pinnacle of Guy Ritchie's career\n	10
4	A Comedy Masterpiece\n	10

In [9]:
```
movie.shape
```

Out[9]:
```
(136, 2)
```

In [10]:
```
movie.to_csv('DekeAdeleye_9810.csv', index=False)
```

In [11]:
```
import string
import re
#import nltk
#nltk.download()
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
```

In [12]:
```
textFeatures = movie['Review'].copy()
textFeatures.shape
```

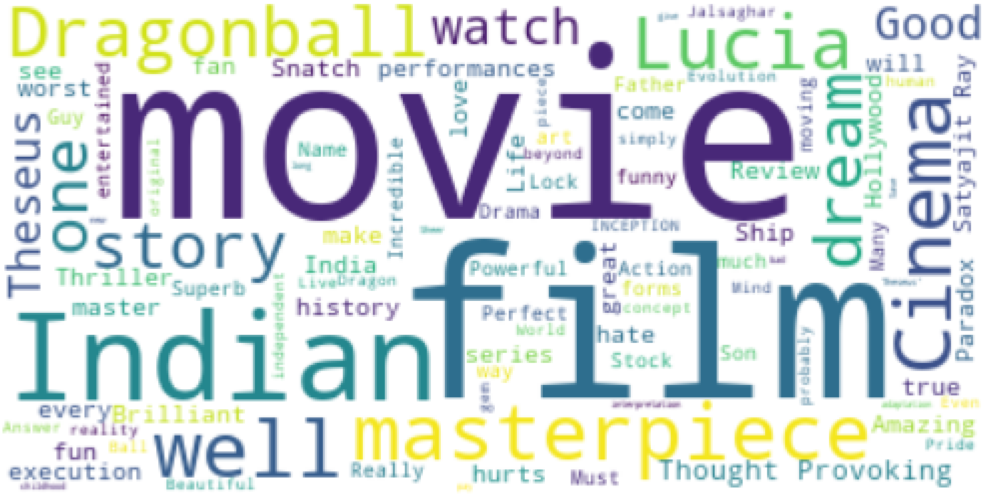
Out[12]:
```
(136,)
```

In [13]:
```
#Preparing text for Wordcloud
text = []
for t in textFeatures:
  text.append(t)
all_text = ', '.join(t for t in text)
#print(all_text)
print(len(all_text))
```

```
6554
```

In [14]:
```
from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [15]:
```python
# Create stopword list
stopwords = set(STOPWORDS)
stopwords.update(["br", "im", "thats"]) #"im","lol","Xa","film"])
# Generate a word cloud image
wordcloud = WordCloud(stopwords=stopwords, background_color="white").generate(all_text)
# Display the image
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
#save the generated image to a file
#wordcloud.to_file("wordcloud_cb_all.png")
```



In [16]:
```python
from textblob import TextBlob

reviews = pd.Series(movie['Review'])

positive_reviews = []
negative_reviews = []

for review in reviews:
    sentiment = TextBlob(review).sentiment.polarity
    if sentiment > 0:
        positive_reviews.append(review)
    elif sentiment < 0:
        negative_reviews.append(review)

print("Positive Reviews:")
for review in positive_reviews:
    print(review)

print("\nNegative Reviews:")
for review in negative_reviews:
    print(review)
```

```
Positive Reviews:
 There are few films that can make me laugh like this one can

 Perfect

 Lock, Stock, and Many Smoking Barrels

 Subtitles A Must To Really Enjoy This

 Just as much fun as Lock, Stock. Snatch is a great and entertaining movie.

 Well edited and darkly funny

 For Every Action, There's A Pikey Reaction!

 Might be my Favorite Movie

 Good, but it's not a film for most folks....
```

In [18]:
```python
positive_reviews_df = pd.DataFrame({'Positive Reviews': positive_reviews})
negative_reviews_df = pd.DataFrame({'Negative Reviews': negative_reviews})

merged_reviews_df = pd.concat([positive_reviews_df, negative_reviews_df], axis=1)

merged_reviews_df.to_csv('merged_reviews.csv', index=False)
```

In [19]:
```python
positive_reviews_df.shape
```

Out[19]:
```
(63, 1)
```

In [20]:
```python
negative_reviews_df.shape
```

Out[20]:
```
(14, 1)
```

In [21]:
```python
positive_reviews_df.head()
```

Out[21]:

	Positive Reviews
0	There are few films that can make me laugh li...
1	Perfect\n
2	Lock, Stock, and Many Smoking Barrels\n
3	Subtitles A Must To Really Enjoy This\n
4	Just as much fun as Lock, Stock. Snatch is a ...

In [22]:
```python
negative_reviews_df.head()
```

Out[22]:

	Negative Reviews
0	An Involving, Sometimes Brutal Story\n
1	About the dangers of giving into your fear an...
2	A subtle look at ageing + the pre-Independenc...
3	Typical extremely silly Danish 1970's farce..\n
4	Insulting as a Dragon Ball fan, as a movies f...

In [23]:
```
merged_reviews_df.head()
```

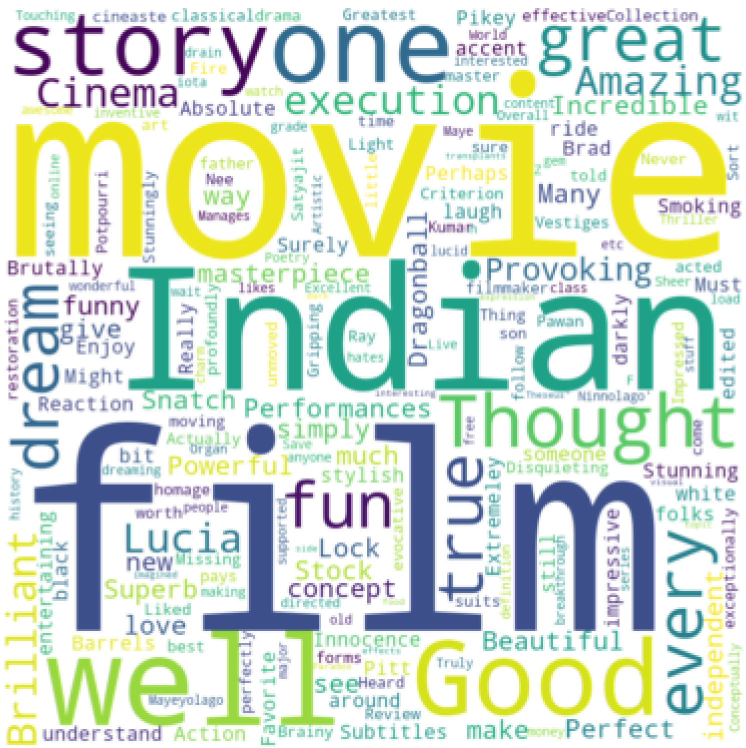
Out[23]:

	Positive Reviews	Negative Reviews
0	There are few films that can make me laugh li...	An Involving, Sometimes Brutal Story\n
1	Perfect\n	About the dangers of giving into your fear an...
2	Lock, Stock, and Many Smoking Barrels\n	A subtle look at ageing + the pre-Independenc...
3	Subtitles A Must To Really Enjoy This\n	Typical extremely silly Danish 1970's farce..\n
4	Just as much fun as Lock, Stock. Snatch is a ...	Insulting as a Dragon Ball fan, as a movies f...

In [24]:

```python
def create_wordcloud(text, title):
    wordcloud = WordCloud(width=800, height=800, background_color='white', min_font_size=10).generate(text)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
positive_text = ' '.join(positive_reviews)

create_wordcloud(positive_text, "Positive Reviews Wordcloud")
```



In [25]:

```
negative_text = ' '.join(negative_reviews)


create_wordcloud(negative_text, "Negative Reviews Wordcloud")
```

In [26]:

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

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

In [27]:
```python
sid = SentimentIntensityAnalyzer()
c = 0
for t in text:
    c+=1
    print(c, t)
    ss = sid.polarity_scores(t)
    print(ss)

    if(ss['compound'] >= 0.05):
        print('positive')

    elif(ss['compound'] <= -0.05):
        print('negative')
    else:
        print('neutral')
    print('\n')
```

```
1  There are few films that can make me laugh like this one can

{'neg': 0.0, 'neu': 0.643, 'pos': 0.357, 'compound': 0.7269}
positive


2  Perfect

{'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.5719}
positive


3  Lock, Stock, and Many Smoking Barrels

{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
neutral


4  The pinnacle of Guy Ritchie's career
```

In [28]:
```
label = []
for r in movie['Rating']:
    r = int(r)
    if (r>5):
        label.append('1') #Positive
    elif(r<5):
        label.append('-1') #Negative
    elif(r==5):
        label.append('0') #Netural
movie['class-label'] = label
```

In [29]:
```
movie['class-label'].value_counts()
```

Out[29]:
```
1     105
-1     30
0       1
Name: class-label, dtype: int64
```

In [30]:
```
movie = movie[movie['class-label']!='0']
```

In [31]:
```
movie['class-label'].value_counts()
```

Out[31]:
```
1     105
-1     30
Name: class-label, dtype: int64
```

In [32]:
```
textFeatures = movie['Review'].copy()
textFeatures.shape
```

Out[32]:
```
(135,)
```

In [33]:
```
import nltk
nltk.download('punkt')
# Stemming using TextBlob library for stemming
from textblob import TextBlob
```

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

In [34]:
```python
def textblob_tokenizer(input_str):
    blob = TextBlob(input_str.lower())
    tokens = blob.words
    words = [token.stem() for token in tokens]
    return words
```

In [35]:
```python
#Toy example:
print(textblob_tokenizer('Q: studed studing!!! I miss uuuu! It&#039;s'))
```

```
['q', 'stude', 'stude', 'i', 'miss', 'uuuu', 'it', '039', 's']
```

In [36]:
```python
input_str = 'Q: studed studing!!! I miss uuuu! It&#039;s'
token = textblob_tokenizer(input_str)
print(token)
```

```
['q', 'stude', 'stude', 'i', 'miss', 'uuuu', 'it', '039', 's']
```

In [37]:
```python
missing_chars = set(input_str) - set(''.join(token))
print("Missing Characters :",missing_chars)
```

```
Missing Characters : {' ', '#', '&', 'g', ':', 'n', 'I', ';', 'Q', '!'}
```

In [38]:
```python
num_tokens = len(token)
print("Number of tokens :", num_tokens)
```

```
Number of tokens : 9
```

In [39]:
```python
num_missing_chars = len(missing_chars)
print("Number of missing characters : ", num_missing_chars)
```

```
Number of missing characters :  10
```

In [40]:
```
#Toy example:
print(textblob_tokenizer(textFeatures.iloc[0]))
```

```
['there', 'are', 'few', 'film', 'that', 'can', 'make', 'me', 'laugh', 'like', 'thi', 'one', 'can']
```

In [41]:
```
#countvectorizer convers each review into a vector based on the word count.
countvectorizer = CountVectorizer(analyzer= 'word', stop_words= 'english',
                                  tokenizer=textblob_tokenizer)
#convers text into a vector based on tf-idf weighting scheme.
tfidfvectorizer = TfidfVectorizer(analyzer= 'word', stop_words= 'english',
                                  tokenizer=textblob_tokenizer)
```

In [42]:
```
textFeatures
```

Out[42]:
```
0       There are few films that can make me laugh li...
1                                              Perfect\n
2                   Lock, Stock, and Many Smoking Barrels\n
3                    The pinnacle of Guy Ritchie's career\n
4                                   A Comedy Masterpiece\n
                             ...                        
131        Humane, Thought-Provoking and Philosophical\n
132     The Ship of Theseus is a painstakingly dialec...
133       long. very long. why did I sit this one out?\n
134                                    One of its kind.\n
135     Brilliance in concept but faltering at other ...
Name: Review, Length: 135, dtype: object
```

In [43]:
```
count_matrix = countvectorizer.fit_transform(textFeatures)
tfidf_matrix = tfidfvectorizer.fit_transform(textFeatures)
```

In [44]:
```
print(tfidf_matrix.shape)
print(count_matrix.shape)
```

```
(135, 383)
(135, 383)
```

In [45]:
```
countvectorizer = CountVectorizer()
tfidfvectorizer = TfidfVectorizer()
```

In [46]:
```
count_matrix = countvectorizer.fit_transform(textFeatures)
tfidf_matrix = tfidfvectorizer.fit_transform(textFeatures)
```

In [47]:

```
#print elements of the matrix.
print(tfidf_matrix)
```

```
(0, 308)	0.23467880776929687
(0, 438)	0.18642646656583037
(0, 244)	0.2741524991643335
(0, 239)	0.2972431283954693
(0, 273)	0.2972431283954693
(0, 262)	0.2741524991643335
(0, 66)	0.46935761553859373
(0, 432)	0.2577694370004327
(0, 149)	0.2577694370004327
(0, 146)	0.2972431283954693
(0, 28)	0.2577694370004327
(0, 435)	0.2741524991643335
(1, 328)	1.0
(2, 40)	0.4562756431792466
(2, 395)	0.4562756431792466
(2, 266)	0.42083094926580916
(2, 22)	0.2287553525118992
(2, 409)	0.42083094926580916
(2, 253)	0.42083094926580916
(3, 67)	0.48279210572097436
(3, 366)	0.48279210572097436
(3, 182)	0.44528754314588787
(3, 304)	0.2263342129237444
(3, 337)	0.48279210572097436
(3, 433)	0.2263342129237444
:	:
(132, 97)	0.30491219187087765
(132, 480)	0.30491219187087765
(132, 465)	0.27255748011548886
(132, 317)	0.2866909631416897
(132, 308)	0.26100958364668436
(132, 438)	0.20734336807652726
(133, 234)	0.5930657108383801
(133, 225)	0.5930657108383801
(133, 304)	0.2780307699402265
(133, 308)	0.46823606890326025
(134, 318)	0.2862822034771439
(134, 388)	0.2862822034771439
(134, 136)	0.2862822034771439
(134, 263)	0.2862822034771439
(134, 406)	0.2862822034771439
```

```
  (134, 316)	0.2862822034771439
  (134, 137)	0.2862822034771439
  (134, 59)	0.2862822034771439
  (134, 198)	0.2640430477676521
  (134, 81)	0.2640430477676521
  (134, 74)	0.21024602788614508
  (134, 35)	0.24826411568164447
  (134, 64)	0.21757002537272987
  (134, 204)	0.18361664918640216
  (134, 304)	0.13420985229514626
```

In [48]:

```
print(tfidf_matrix.shape)
print(count_matrix.shape)
```

```
(135, 493)
(135, 493)
```

In [49]:

```
features_train, features_test, labels_train, labels_test = train_test_split(
    tfidf_matrix, movie['class-label'], test_size=0.3,random_state=5)
print(features_train.shape, features_test.shape, labels_train.shape, labels_test.shape)
```

```
(94, 493) (41, 493) (94,) (41,)
```

In [50]:

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
```

In [51]:
```
#SVM classifier
from sklearn.svm import SVC
print("\nEvaluation for SVM \n")
svc = SVC(kernel='sigmoid', gamma=1.0)
svc.fit(features_train, labels_train)
prediction = svc.predict(features_test)
acc = accuracy_score(labels_test,prediction)
print('Accuracy:', acc)
from sklearn.metrics import precision_score
prec = precision_score(labels_test,prediction, average='weighted')
print('Precision:', prec)
from sklearn.metrics import recall_score
recall = recall_score(labels_test,prediction, average='weighted')
print('Recall:', recall)
from sklearn.metrics import f1_score
f1 = f1_score(labels_test,prediction, average='weighted')
print('F-1 measure: ', f1)
print('\nConfusion Matrix:\n')
print(confusion_matrix(labels_test, prediction))
print(classification_report(labels_test, prediction))
#print(prediction)
```

```
Evaluation for SVM

Accuracy: 0.7073170731707317
Precision: 0.5002974419988102
Recall: 0.7073170731707317
F-1 measure:  0.5860627177700348

Confusion Matrix:

[[ 0 12]
 [ 0 29]]
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00        12
           1       0.71      1.00      0.83        29

    accuracy                           0.71        41
   macro avg       0.35      0.50      0.41        41
weighted avg       0.50      0.71      0.59        41
```

In [52]:
```python
#Decision Tree
print("\nEvaluation for Decision Tree \n")
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(features_train, labels_train)
prediction = dtree.predict(features_test)
acc = accuracy_score(labels_test,prediction)
print('Accuracy: ', acc)
prec = precision_score(labels_test,prediction, average='weighted')
print('Precision: ', prec)
recall = recall_score(labels_test,prediction, average='weighted')
print('Recall: ', recall)
f1 = f1_score(labels_test,prediction, average='weighted')
print('F-1 measure: ',f1)
print('\nConfusion Matrix:\n')
print(confusion_matrix(labels_test, prediction))
print(classification_report(labels_test, prediction))
```

```
Evaluation for Decision Tree

Accuracy:  0.6585365853658537
Precision:  0.737979094076655
Recall:  0.6585365853658537
F-1 measure:  0.6739024390243903

Confusion Matrix:

[[ 9  3]
 [11 18]]
              precision    recall  f1-score   support

          -1       0.45      0.75      0.56        12
           1       0.86      0.62      0.72        29

    accuracy                           0.66        41
   macro avg       0.65      0.69      0.64        41
weighted avg       0.74      0.66      0.67        41
```

In [53]:
```
#Building a model using the count matrix
features_train, features_test, labels_train, labels_test = train_test_split(
    count_matrix, movie['class-label'], test_size=0.3,random_state=5)
print(features_train.shape, features_test.shape, labels_train.shape, labels_test.shape)
```

```
(94, 493) (41, 493) (94,) (41,)
```

In [54]:

```
from sklearn.svm import SVC
print("\nEvaluation for SVM \n")
svc = SVC(kernel='sigmoid', gamma=1.0)
svc.fit(features_train, labels_train)
prediction = svc.predict(features_test)
acc = accuracy_score(labels_test,prediction)
print('Accuracy:', acc)
from sklearn.metrics import precision_score
prec = precision_score(labels_test,prediction, average='weighted')
print('Precision:', prec)
from sklearn.metrics import recall_score
recall = recall_score(labels_test,prediction, average='weighted')
print('Recall:', recall)
from sklearn.metrics import f1_score
f1 = f1_score(labels_test,prediction, average='weighted')
print('F-1 measure: ', f1)
print('\nConfusion Matrix:\n')
print(confusion_matrix(labels_test, prediction))
print(classification_report(labels_test, prediction))
#print(prediction)
```

```
Evaluation for SVM

Accuracy: 0.6829268292682927
Precision: 0.49512195121951214
Recall: 0.6829268292682927
F-1 measure:  0.574054436196536

Confusion Matrix:

[[ 0 12]
 [ 1 28]]
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00        12
           1       0.70      0.97      0.81        29

    accuracy                           0.68        41
   macro avg       0.35      0.48      0.41        41
weighted avg       0.50      0.68      0.57        41
```

In [55]:
```python
#Decision Tree
print("\nEvaluation for Decision Tree \n")
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(features_train, labels_train)
prediction = dtree.predict(features_test)
acc = accuracy_score(labels_test,prediction)
print('Accuracy: ', acc)
prec = precision_score(labels_test,prediction, average='weighted')
print('Precision: ', prec)
recall = recall_score(labels_test,prediction, average='weighted')
print('Recall: ', recall)
f1 = f1_score(labels_test,prediction, average='weighted')
print('F-1 measure: ',f1)
print('\nConfusion Matrix:\n')
print(confusion_matrix(labels_test, prediction))
print(classification_report(labels_test, prediction))
```

```
Evaluation for Decision Tree

Accuracy:  0.6585365853658537
Precision:  0.6755710414247
Recall:  0.6585365853658537
F-1 measure:  0.6655722326454034

Confusion Matrix:

[[ 6  6]
 [ 8 21]]
              precision    recall  f1-score   support

          -1       0.43      0.50      0.46        12
           1       0.78      0.72      0.75        29

    accuracy                           0.66        41
   macro avg       0.60      0.61      0.61        41
weighted avg       0.68      0.66      0.67        41
```

In this workbook, we performed the following functions : generated the data from the website (lines #2-10); prepared text and generated word clouds (Lines #12-15).In lines #16-25, the reviews were seperated into positive and negative reviews and word cloud was built for each of the positive and negative reviews. performed sentiment analysis using Vader (lines #26-27), performed sentiment classification using machine learning. A truth set was first prepared by categorising the reviews into positie, negative and neutral reviews. (line #28-32). Stemming was done using the Textblob function (lines #33-40), and then the text dataset was transformed into 2 matrix repreentations - the count matrix and the tdif matrix(lines #41-48). MI models were built using the tdif matrix in line #48-52., while another model was built using the count matrix in line #53- 55.

Evaluating the results of the SVM Classifier for the dataset transformed into a vector using tdidf vectoriser, using the evalution metrics and the confusion matrix; we have the following results.
The accuracy of the model is 0.7073, which means that the model correctly predicted the class label for 70.73% of the instances in the test dataset. with a precision of 0.5003, when the model predicts a positive class, it is correct 50.03% of the time.
The recall of the model is 0.7073, which means that the model correctly identifies 70.73% of the positive class instances.
The F1-measure of the model is 0.5861, which is an overall measure of the model's performance.
The confusion matrix shows that the model did not correctly predict any of the negative reviews (all 12 were predicted as positive).
In the decision tree, accuracy is 0.6585 which means that the model correctly predicted the class label for 65.85% of the instances
Precision is 0.7380 which means that the model correctly predicted the class label for 73.80% of the instances
Recall is 0.6585 which means that the model correctly identifies 65.85% of the positive class instances
F-1 measure is 0.6739.
The confusion matrix shows that the model correctly identified 9 negatives (true negative) and 18 positive (true positive), but misclassified 3 negative samples as positive(false positive) and 11 positive samples as negative(false negative). This suggests that the model may need to be optimized for the given dataset.
Based on the result of the confusion matrix on the SVM classifier, where it did not correctly identify any negatives its performance also needs to be improved, even though its accuracy is slightly better than the decision tree. Both models need to be optimised for the given datas set.

Evaluating the results of the SVM Classifier for the dataset transformed into a vector using count matrix vectoriser, using the evalution metrics and the confusion matrix; we have the following results.
The accuracy of the model is 0.6829, which means that the model correctly predicted the class label for 68.29% of the instances in the test dataset. with a precision of 0.4951, when the model predicts a positive class, it is correct 49.51% of the time.
The recall of the model is 0.6829, which means that the model correctly identifies 68.29% of the positive class instances.
The F1-measure of the model is 0.5741, which is an overall measure of the model's performance.
The confusion matrix shows that the model did correctly predicted only 1 negative reviews, correctly identified 28 positives, but misclassified 12 negative samples as positive,

Based on the evaluation metrics and confusion matrix using count matrix, performance of the Decision Tree model is evaluated.
Accuracy: The accuracy of the model is 0.6585, which means that the model correctly predicted the class label for only 65.85% of the instances in the test dataset.Precision: The precision of the model is 0.6756, which means that when the model predicts a positive (1) class, it is correct 67.56% of the time.Recall: The recall of the model is 0.6585, which means that the model correctly identifies 65.85% of the positive class instances.F1-measure: The F1-measure of the model is 0.6656
Confusion matrix: the model correctly predicted 6 true negatives, and incorrectly predicted 6 false positives. the model correctly predicted 21 true positives, and incorrectly predicted 8 false negatives.

There isn't any significant difference in the evaluation of the count matrix and dfidf mtrix using the svm clasifier and the decision tree.
The models need to be improved upon.