Importing dependencies

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import re,string,unicodedata
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,f1_score
from sklearn.model_selection import train_test_split
from string import punctuation
from nltk import pos_tag
from nltk.corpus import wordnet
import re
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

Loading data

```
df=pd.read_csv('/content/drive/MyDrive/Sentiment Analysis/Data/IMDB-Dataset.csv', encoding='latin-1')
```

Cleaning data

```
#Customize stopword as per data
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
new_stopwords = ["would","shall","could","might"]
stop_words.extend(new_stopwords)
stop_words.remove("not")
stop_words=set(stop_words)
print(stop_words)
```

```
{'are', 'against', "don't", 'have', 'don', 'does', 'under', 'out', 'so', "didn't", 'itself', "isn't", 'by', 'above', 'all', 'can',
```

```
#Removing special character
def remove_special_character(content):
    return re.sub('\W+',' ', content )#re.sub('\[[^&@#!]]*\]', '', content)

# Removing URL's
def remove_url(content):
    return re.sub(r'http\S+', '', content)

#Removing the stopwords from text
def remove_stopwords(content):
    clean_data = []
    for i in content.split():
        if i.strip().lower() not in stop_words and i.strip().lower().isalpha():
            clean_data.append(i.strip().lower())
    return " ".join(clean_data)

# Expansion of english contractions
def contraction_expansion(content):
    content = re.sub(r"won\'t", "would not", content)
    content = re.sub(r"can\'t", "can not", content)
    content = re.sub(r"don\'t", "do not", content)
    content = re.sub(r"shouldn\'t", "should not", content)
    content = re.sub(r"needn\'t", "need not", content)
    content = re.sub(r"hasn\'t", "has not", content)
```

```
    content = re.sub(r"haven\'t", "have not", content)
    content = re.sub(r"weren\'t", "were not", content)
    content = re.sub(r"mightn\'t", "might not", content)
    content = re.sub(r"didn\'t", "did not", content)
    content = re.sub(r"n\'t", " not", content)
    '''content = re.sub(r"\'re", " are", content)
    content = re.sub(r"\'s", " is", content)
    content = re.sub(r"\'d", " would", content)
    content = re.sub(r"\'ll", " will", content)
    content = re.sub(r"\'t", " not", content)
    content = re.sub(r"\'ve", " have", content)
    content = re.sub(r"\'m", " am", content)'''
    return content

#Data preprocessing
def data_cleaning(content):
    content = contraction_expansion(content)
    content = remove_special_character(content)
    content = remove_url(content)

    content = remove_stopwords(content)
    return content
```

```
pd.options.display.max_colwidth = 1000
#Data cleaning
df['Reviews_clean']=df['Reviews'].apply(data_cleaning)
df.head(5)
```

	Ratings	Reviews	Movies	Resenhas	Reviews_clean
0	1	*Disclaimer: I only watched this movie as a conditional agreement. And I see films for free. I wouldn't be caught dead giving my hard earned money to these idiots.Well, to explain the depth of this 'film', I could write my shortest review, ever. Don't see this movie. It is by far the stupidest, lamest, most lazy, and unbelievably UNFUNNY movie I have ever seen. It is a total disaster. But since my hatred for this movie, and the others like it, extends far beyond one viewing, I think I'll go on for a bit.I don't know any of the people in the movie besides Carmen Electra, Vanessa Minnillo, and Kim Kardashian, but it doesn't matter. They're all horrible, though I think that was the point. The editing is flat out horrible, and possibly blatant continuity errors make this crapfast even crappier than I thought it would be. Now I know that these films are not supposed to be serious at all, but come on, it's film-making 101 that if someone gets a minor facial cut, it should be there in the...	Disaster Movie	* IsenÃ§Ã£o de responsabilidade: eu sÃ³ assisti esse filme como um acordo condicional. E eu vejo filmes de graÃ§a. Eu nÃ£o seria pego morto dando meu dinheiro suado a esses idiotas. Bem, para explicar a profundidade desse 'filme', eu poderia escrever minha crÃtica mais curta de todos os tempos. NÃ£o vÃª este filme. Ã de longe o filme mais estÃºpido, lamenta, preguiÃ§oso e inacreditavelmente UNFUNNY que eu jÃ¡ vi. Ã um desastre total. Mas como o meu Ã³dio por este filme e por outros, se estende muito alÃ©m de uma exibiÃ§Ã£o, acho que vou continuar um pouco. NÃ£o conheÃ§o nenhuma das pessoas do filme alÃ©m de Carmen Electra, Vanessa Minnillo, e Kim Kardashian, mas isso nÃ£o importa. Eles sÃ£o todos horrÃveis, embora eu ache que esse seja o ponto. A ediÃ§Ã£o Ã© horrÃvel e, possivelmente, erros de continuidade flagrantes tornam essa porcaria ainda mais horrÃvel do que eu pensava. Agora eu sei que esses filmes nÃ£o devem ser sÃ©rios, mas vamos lÃ¡, Ã© o cinema 101 que se alguÃ©m f...	disclaimer watched movie conditional agreement see films free not caught dead giving hard earned money idiots well explain depth film write shortest review ever not see movie far stupidest lamest lazy unbelievably unfunny movie ever seen total disaster since hatred movie others like extends far beyond one viewing think go bit not know people movie besides carmen electra vanessa minnillo kim kardashian not matter horrible though think point editing flat horrible possibly blatant continuity errors make crapfast even crappier thought know films not supposed serious come film making someone gets minor facial cut next shot someone gets cut sword blood least cut though since narnia films get away give disaster movie pass jokes thoughtless mindless physical gags obviously take popular movies last year late well including best picture nominees know saddest thing stupid movies not care much money make many cameos sorry ass excuses films taking away jobs actors writers directors truly deserv...

```
#Mapping rating data to Binary label 1 (+ve) if rating >=7 and 0 (-ve) if rating <=4 and 2 (neutral) if ra
df['Label'] = df['Ratings'].apply(lambda x: '1' if x >= 7 else ('0' if x<=4 else '2'))
#Removing
df=df[df.Label<'2']
data=df[['Reviews_clean','Reviews','Ratings','Label']]
print(data['Label'].value_counts())
```

```
0    60000
1    60000
Name: Label, dtype: int64
```

being "Twister". How does Juno, conte-os, 1) filme de desastre shown lack sort writing skill

```
#Importing dependencies for feature engineering
import sys
import os
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from prettytable import PrettyTable
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
```

that because some of these outros filmes. AlguÃ©m deveria negatives rate deserves top

Lemmatization

clever or witty or re... copiar e colar de um filme para movie meet spartans rather

```
# lemmatization of word
class LemmaTokenizer(object):
    def __init__(self):
        self.wordnetlemma = WordNetLemmatizer()
    def __call__(self, reviews):
        return [self.wordnetlemma.lemmatize(word) for word in word_tokenize(reviews)]
```

no point in watching these mais sentido assistir a esses making observations points

Vectorization with Count Vectorizer and TDIDF Vectorizer with unigram, bigram and trigram

Movie? Remember how it was Lembra como era original, com remember scary movie

```
train,test=train_test_split(data,test_size=.3,random_state=42, shuffle=True)
#countvect = CountVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(), ngram_range=(1,3), min_df=10
tfidfvect = TfidfVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(), ngram_range=(1,3),min_df=10,m
#x_train_count = countvect.fit_transform(train['Reviews_clean']).toarray()
#x_test_count = countvect.transform(test['Reviews_clean']).toarray()
x_train_tfidf = tfidfvect.fit_transform(train['Reviews_clean']).toarray()
x_test_tfidf = tfidfvect.transform(test['Reviews_clean']).toarray()

y_train = train['Label']
y_test = test['Label']
```

Feature Selection with Chi squared

```
from sklearn.feature_selection import chi2
import numpy as np
N = 500
Number = 1
featureselection = PrettyTable(["Unigram", "Bigram","Trigram"])
for category in train['Label'].unique():
    features_chi2 = chi2(x_train_tfidf, train['Label'] == category)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidfvect.get_feature_names_out())[indices]
    unigrams = [x for x in feature_names if len(x.split(' ')) == 1]
    bigrams = [x for x in feature_names if len(x.split(' ')) == 2]
    trigrams = [x for x in feature_names if len(x.split(' ')) == 3]
    print("%s. %s :" % (Number,category))
    print("\t# Unigrams :\n\t. %s" %('\n\t. '.join(unigrams[-N:])))
    print("\t# Bigrams :\n\t. %s" %('\n\t. '.join(bigrams[-N:])))
    print("\t# Trigrams :\n\t. %s" %('\n\t. '.join(trigrams[-N:])))
    Number += 1
```

```
1. 1 :
        # Unigrams :
        . due
        . tell
        . huge
        . turn
        . took
        . person
        . one
        . behind
        . second
        . came
        . scary
        . main
        . dvd
        . voice
        . despite
        . extremely
        . evil
        . blood
        . later
        . â
        . number
        . however
        . close
        . playing
        . face
        . black
        . say
        . lead
        . away
        . know
        . matter
        . need
        . two
        . going
        . fi
        . remember
        . sci
        . happen
        . want
        . watch
        . kind
        . add
        . set
        . stop
        . fight
        . credit
```

```
. group
. flick
. death
. perhaps
. much
. becomes
. sequence
. version
. help
. soon
```

Model selection

```
# Import prerequisite libraries
import sys
import numpy as np
import scipy as sp
import sklearn as sk
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, roc_auc_score, precision_score, recall_score
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
```

Logistic regression model

```
model_1=LogisticRegression()
```

Training of Logistic Regression Model

```
model_1.fit(x_train_tfidf,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Evaluation on Test and Train dataset

```
%%time
print("Precision Score on training dateset for Logistic Regression: %s" % precision_score(y_train,model_1.
print("AUC Score on training dateset for Logistic Regression: %s" % roc_auc_score(y_train,model_1.predict_
f1_score_train_1 =f1_score(y_train,model_1.predict(x_train_tfidf),average="weighted")
print("F1 Score ftraining dateset for Logistic Regression: %s" % f1_score_train_1)
print("Precision Score on test for Logistic Regression: %s" % precision_score(y_test,model_1.predict(x_tes
print("AUC Score on test for Logistic Regression: %s" % roc_auc_score(y_test,model_1.predict_proba(x_test_
f1_score_1 =f1_score(y_test,model_1.predict(x_test_tfidf),average="weighted")
print("F1 Score for Logistic Regression: %s" % f1_score_1)
```

```
Precision Score on training dateset for Logistic Regression: 0.8448452380952381
AUC Score on training dateset for Logistic Regression: 0.9229142463745506
F1 Score ftraining dateset for Logistic Regression: 0.8448283416780853
Precision Score on test for Logistic Regression: 0.8379166666666666
AUC Score on test for Logistic Regression: 0.9173375519794157
F1 Score for Logistic Regression: 0.8379167065621785
CPU times: user 3.45 s, sys: 562 ms, total: 4.01 s
Wall time: 4.32 s
```

Decision Tree Classifier

```
model_2 = Pipeline(
    steps=[
    ("classifier", DecisionTreeClassifier())
    ]
)
```

Training of Decision Tree Classifier

```
model_2.fit(x_train_tfidf,y_train)
```

```
▸ Pipeline
▸ DecisionTreeClassifier
```

Evaluation on test data and training data of Decision Tree Classifier

```
%%time
print("Precision Score on training dateset for Decision Tree Classifier: %s" % precision_score(y_train,mode
print("AUC Score on training dateset for Decision Tree Classifier: %s" % roc_auc_score(y_train,model_2.pre
f1_score_train_2 =f1_score(y_train,model_2.predict(x_train_tfidf),average="weighted")
print("F1 Score training dateset for Decision Tree Classifier: %s" % f1_score_train_2)
print("Precision Score on test for Decision Tree Classifier: %s" % precision_score(y_test,model_2.predict(
print("AUC Score on test for Decision Tree Classifier: %s" % roc_auc_score(y_test,model_2.predict_proba(x_
f1_score_2 =f1_score(y_test,model_2.predict(x_test_tfidf),average="weighted")
print("F1 Score for Decision Tree Classifier: %s" % f1_score_2)
```

```
Precision Score on training dateset for Decision Tree Classifier: 0.9998333333333334
AUC Score on training dateset for Decision Tree Classifier: 0.9999998883203747
F1 Score training dateset for Decision Tree Classifier: 0.999833333420636
Precision Score on test for Decision Tree Classifier: 0.7031388888888889
AUC Score on test for Decision Tree Classifier: 0.7033137849780722
F1 Score for Decision Tree Classifier: 0.7031430192551575
CPU times: user 2.76 s, sys: 140 ms, total: 2.9 s
Wall time: 2.9 s
```

Random Forest Classifier

```
model_3 = Pipeline(
    steps=[
    ("classifier", RandomForestClassifier())]
)
```

Training of Random Forest Classifier

```
model_3.fit(x_train_tfidf,y_train)
```

```
▸ Pipeline
▸ RandomForestClassifier
```

Evaluation on test data and training data of Random Forest Classifier

```
%%time
print("Precision Score on training dateset for Random Forest Classifier: %s" % precision_score(y_train,mode
print("AUC Score on training dateset for Random Forest Classifier: %s" % roc_auc_score(y_train,model_3.pre
f1_score_train_4 =f1_score(y_train,model_3.predict(x_train_tfidf),average="weighted")
print("F1 Score training dateset for Random Forest Classifier: %s" % f1_score_train_4)
print("Precision Score on test for Random Forest Classifier: %s" % precision_score(y_test,model_3.predict(
print("AUC Score on test for Random Forest Classifier: %s" % roc_auc_score(y_test,model_3.predict_proba(x_
f1_score_4 =f1_score(y_test,model_3.predict(x_test_tfidf),average="weighted")
print("F1 Score for Random Forest Classifier: %s" % f1_score_4)
```

```
Precision Score on training dateset for Random Forest Classifier: 0.9998333333333334
AUC Score on training dateset for Random Forest Classifier: 0.9999654785239513
F1 Score training dateset for Random Forest Classifier: 0.9998333334065579
Precision Score on test for Random Forest Classifier: 0.8111111111111111
AUC Score on test for Random Forest Classifier: 0.8924307229213173
F1 Score for Random Forest Classifier: 0.8111148330028652
CPU times: user 19.5 s, sys: 182 ms, total: 19.7 s
Wall time: 21.2 s
```

Ada Boost Classifier

```
model_5 = Pipeline(
    steps=[
        ("classifier", AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=4),
    n_estimators=100,
    learning_rate=.8)),
    ]
)
```

Training of Ada boost Classifier

```
model_5.fit(x_train_tfidf,y_train)
```

```
▸ Pipeline
  ▸ classifier: AdaBoostClassifier
  ▸ base_estimator: DecisionTreeClassifier
      ▸ DecisionTreeClassifier
```

Evaluation on test data and training data of Ada boost Classifier

```
%%time
print("Precision Score on training dateset for Ada Boost Classifier: %s" % precision_score(y_train,model_5
print("AUC Score on training dateset for Ada Boost Classifier: %s" % roc_auc_score(y_train,model_5.predict_
f1_score_train_5 =f1_score(y_train,model_5.predict(x_train_tfidf),average="weighted")
print("F1 Score training dateset for Ada Boost Classifier: %s" % f1_score_train_5)
print("Precision Score on test for Ada Boost Classifier: %s" % precision_score(y_test,model_5.predict(x_te
print("AUC Score on test for Ada Boost Classifier: %s" % roc_auc_score(y_test,model_5.predict_proba(x_test_
f1_score_5 =f1_score(y_test,model_5.predict(x_test_tfidf),average="weighted")
print("F1 Score for Random Forest Classifier: %s" % f1_score_5)
```

```
Precision Score on training dateset for Ada Boost Classifier: 0.871
AUC Score on training dateset for Ada Boost Classifier: 0.9499708433977274
F1 Score training dateset for Ada Boost Classifier: 0.8709953709003464
Precision Score on test for Ada Boost Classifier: 0.8114444444444444
AUC Score on test for Ada Boost Classifier: 0.8894715693585666
F1 Score for Random Forest Classifier: 0.8114480381041321
CPU times: user 36.4 s, sys: 14.1 s, total: 50.5 s
Wall time: 52.3 s
```

✓ 52s completed at 12:04 PM