```
# -*- coding: utf-8 -*-
"""
Created on Sat Feb 24 22:25:03 2024

@author: cassh
"""

## Libraries
import pandas as pd
import numpy as np
import re
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
plot_confusion_matrix
from sklearn.metrics import (accuracy_score, f1_score, classification_report)
from sklearn.ensemble import RandomForestClassifier
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from PIL import Image
import random as rd
import itertools
import json


## Reading csv files
action = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
action.csv')
adventure = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
adventure.csv')
animation = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
animation.csv')
bio = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
biography.csv')
crime = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
crime.csv')
family = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
family.csv')
fantasy = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
fantasy.csv')
film_noir = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
film-noir.csv')
history = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
history.csv')
horror = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
horror.csv')
mystery = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
mystery.csv')
romance = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
romance.csv')
scifi = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
scifi.csv')
sports = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
```

```
sports.csv')
thriller = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\
thriller.csv')
war = pd.read_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\data\war.csv')


## Adding Genre column
action['genre'] = 'action'
adventure['genre'] = 'adventure'
animation['genre'] = 'animation'
bio['genre'] = 'biography'
crime['genre'] = 'crime'
family['genre'] = 'family'
fantasy['genre'] = 'fantasy'
film_noir['genre'] = 'film-noir'
history['genre'] = 'history'
horror['genre'] = 'horror'
mystery['genre'] = 'mystery'
romance['genre'] = 'romance'
scifi['genre'] = 'sci-fi'
sports['genre'] = 'sports'
thriller['genre'] = 'thriller'
war['genre'] = 'war'

## Concatenate all files into on dataframe
#movies = pd.concat(file_list, ignore_index = True)
movies = pd.concat([action,adventure,animation,bio,crime,family,fantasy,
                    film_noir,history,horror,mystery,romance,scifi,
                    sports,thriller,war])
# Resetting index
movies = movies.reset_index(drop=True)
movies.head()
## Writing the dataframe to a csv file
movies.to_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\new data\
combined_data.csv', index = False)

## Data Cleansing
print(movies.shape) # 368,300 rows (movies), 14 columns (categories)
print('The column names are:\n')
for col in movies.columns:
    print(col)

## Checking for missing values
movies.isnull().sum()

movies['year'].head(10)
movies['year'].tail(10)

movies['certificate'].head(10)
movies['certificate'].tail(10)

### Removing unwanted values in columns
## Year
movies['year'].unique()
unwanted_years = ['I', 'II', 'V', 'III', 'VII', 'IV', 'XXIII', 'IX', 'XV', 'VI',
                  'X', 'XIV', 'XIX', 'XXIX', 'XXI', 'VIII', 'XI', 'XVIII', 'XII',
                  'XIII', 'LXXI', 'XVI', 'XX', 'XXXIII', 'XXXII', 'XXXVI', 'XVII',
                  'LXIV', 'LXII', 'LXVIII', 'XL', 'XXXIV', 'XXXI', 'XLV', 'XLIV',
                  'XXIV', 'XXVII', 'LX', 'XXV', 'XXXIX', '2029', 'XXVIII', 'XXX',
```

```
                  'LXXII', '1909', 'XXXVIII', 'XXII', 'LVI', 'LVII' 'XLI', 'LII',
                  'XXXVII', 'LIX', 'LVIII', 'LXX', 'XLIII', 'XLIX', 'LXXIV',
'XXVI',
                  'C', 'XLI', 'LVII', 'LV','XLVI', 'LXXVII', 'XXXV', 'LIV', 'LI',
                  'LXXXII', 'XCIX', 'LXIII']
movies_clean = movies[~movies['year'].isin(unwanted_years)]
# Filling missing values with 0s
movies_clean['year'] = movies_clean['year'].fillna(0).astype(int)
# Replacing 0s with NaNs
movies_clean['year'] = movies['year'].replace(0, np.nan)
# Dropping rows with missing values
movies_clean = movies_clean.dropna(subset=['year'])
movies_clean['year'].isnull().sum()
# Make 'year' column int
movies_clean['year'] = movies_clean['year'].astype(int)

## Removing unwanted columns
movies_clean = movies_clean.drop(['movie_id', 'director_id', 'star_id'], axis = 1)

## Director
# Change type
movies_clean['director'] = movies_clean['director'].str.replace('\n', '')
movies_clean['director'] = movies_clean['director'].fillna('none')

## Star
# Change type
movies_clean['star'] = movies_clean['star'].str.replace('\n', '')
movies_clean['star'] = movies_clean['star'].fillna('none')
# Rename column
movies_clean.rename(columns = {'star':'actors'}, inplace = True)

## Runtime
movies_clean = movies_clean.dropna(subset=['runtime'])
#Change type to int
movies_clean['runtime'] = movies_clean['runtime'].str.replace('min', '')
movies_clean['runtime'] = movies_clean['runtime'].str.replace(',', '').astype(int)
# Convert to datetime format
movies_clean['runtime'] = pd.to_timedelta(movies_clean['runtime'], unit='m')

## Certificate
movies_clean['certificate'].unique()
movies_clean['certificate'].value_counts()
movies_clean['certificate'] = movies_clean['certificate'].fillna('NR')
movies_clean['certificate'] = movies_clean['certificate'].replace({'Not
Rated':'NR'}, regex = True)

## Movie Name
movies_clean = movies_clean.dropna(subset=['movie_name'])

## Gross
# Rename column
movies_clean.rename(columns = {'gross(in $)':'revenue'}, inplace = True)
# Normalizing revenue
movies_clean['revenue'] =
movies_clean['revenue'].fillna(movies_clean['revenue'].median())

## Votes
movies_clean['votes'] = movies_clean['votes'].fillna(0).astype(int)
```

```
## Rating
movies_clean['rating'] = movies_clean['rating'].fillna(0).astype(float)

## Final check for missing values
movies_clean.isna().sum()

## Checking for duplicated rows
duplicate_values = movies_clean['movie_name'].duplicated()
print(duplicate_values)
## Removing duplicate rows
movies_clean = movies_clean.drop_duplicates(subset=['movie_name'], keep='first')
print(movies_clean)

## Writing the clean dataframe to a csv file
movies_clean.to_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\new data\
movies_clean.csv', index = False)


### Exploratory Data/Visualizations
movies_clean.describe()
movies_clean.columns.unique()
movies_clean.shape # 142,626 movies, 11 columns
movies_clean.info()

#options for plot styles
print(plt.style.available)

### Top Grossing Movies
revenue_ranked = movies_clean.sort_values(by=['revenue'], ascending = False)
revenue_ranked.reset_index(inplace = True)
revenue_ranked.head()
## Plotting the top 20 grossing movies
plt.figure(figsize=(15,12))
sns.barplot(x=revenue_ranked.loc[0:19, 'movie_name'],
            y=revenue_ranked.loc[0:19, 'revenue'],
            color='mediumspringgreen',
            label='Revenue', ci=None)

plt.xlabel('movie', fontdict = {'fontname': 'Times New Roman', 'color': 'black',
'fontsize' : '15'})
plt.ylabel('revenue (hundreds of millions)',
           fontdict = {'fontname': 'Times New Roman', 'color': 'black',
'fontsize' : '15'})

plt.title("Top 20 Grossing Movies",
          fontdict = {'fontname': 'Times New Roman', 'color': 'black', 'fontsize' :
'25'})
plt.xticks(rotation=35, horizontalalignment='right', fontsize=15)
plt.legend(loc='best', fontsize=15)

## Plotting the top 10 grossing movies
plt.figure(figsize=(15,12))
sns.barplot(x=revenue_ranked.loc[0:9, 'movie_name'],
            y=revenue_ranked.loc[0:9, 'revenue'],
            color='mediumspringgreen',
            label='Revenue', ci=None)

plt.xlabel('movie', fontdict = {'fontname': 'Times New Roman', 'color': 'black',
'fontsize' : '15'})
```

```
plt.ylabel('revenue (hundreds of millions)',
           fontdict = {'fontname': 'Times New Roman', 'color': 'black',
'fontsize' : '15'})

plt.title("Top 10 Grossing Movies",
          fontdict = {'fontname': 'Times New Roman', 'color': 'black', 'fontsize' :
'25'})
plt.xticks(rotation=35, horizontalalignment='right', fontsize=15)
#plt.yticks(np.arange(0,10,2))
plt.legend(loc='best', fontsize=15)


### Revenue by Genre
revenue_genre = movies_clean.groupby('genre', as_index=False)
['revenue'].sum().sort_values(by='revenue', ascending=False)
revenue_genre
## Plotting revenue by genre
plt.figure(figsize=(14,7))
ax = sns.barplot(x=revenue_genre['revenue'],
                 y=revenue_genre['genre'],
                 palette='crest')
plt.xlabel('revenue (hundreds of billions)', fontdict = {'fontname': 'Times New
Roman', 'color': 'black', 'fontsize' : '15'})
plt.ylabel('genre',fontdict = {'fontname': 'Times New Roman', 'color': 'black',
'fontsize' : '15'})
plt.title('Revenue by Genre',
          fontdict = {'fontname': 'Times New Roman', 'color': 'black', 'fontsize' :
'25'})
plt.show()


### Top Directors
len(movies_clean['director'].unique())  # 68,696 unique directors
movies_clean['director'].value_counts()
mask = movies_clean['director'] == 'none'
directors = movies_clean[~mask]
directors['director'].value_counts()

## Include directors who have directed at least 5 movies
director_vcounts = directors['director'].value_counts()
director_list = director_vcounts[director_vcounts >= 5].index.tolist()
directors_ranked = directors[directors['director'].isin(director_list)]



### Top Rated Movies
ratings_ranked = movies_clean.sort_values(by=['rating'], ascending = False)
ratings_ranked.reset_index(inplace = True)
ratings_ranked.head(10)
## Plotting the top 10 rated movies
plt.figure(figsize=(15,12))
sns.barplot(x=ratings_ranked.loc[0:9, 'movie_name'],
            y=ratings_ranked.loc[0:9, 'rating'],
            color='mediumspringgreen',
            label='rating', ci=None)

plt.xlabel('movie', fontdict = {'fontname': 'Times New Roman', 'color': 'black',
'fontsize' : '15'})
plt.ylabel('rating (out of 10)',
```

```
           fontdict = {'fontname': 'Times New Roman', 'color': 'black',
'fontsize' : '15'})

plt.title("Top 10 Rated Movies",
          fontdict = {'fontname': 'Times New Roman', 'color': 'black', 'fontsize' :
'25'})
plt.xticks(rotation=35, horizontalalignment='right', fontsize=15)
plt.legend(loc='best', fontsize=15)
plt.show()



### Creating a smaller data frame
## Genres: sci-fi, crime, romance, sports, horror
movies_clean['genre'].unique()
movies_new = movies_clean.loc[movies_clean['genre'].isin(
    ['sci-fi','crime','romance','sports','horror'])]
movies_new['genre'].unique()
movies_new  # 62,010 rows, 11 columns
# Writing the new data set to a csv
movies_new.to_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\new data\
5_genres.csv', index = False)
## DF only containing title and description
movies_new.columns
columns_dropped = ['movie_name','year','certificate','runtime','rating',
                   'director','actors','votes','revenue']
#movie_description = movies_new.drop(columns_dropped, axis = 1, inplace = True)
movies_new.drop(columns_dropped, axis = 1, inplace = True)
movies_new
#print(movie_description)
#print(movies_new)  # 62,010 descriptions (rows), 2 columns
#movie_description = movies_new[['movie_name','description']]
#movie_description  # 62,010 movies, 2 columns
## Reading data as csv, not df
movie_description = r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\new data\
5_genres.csv'

'''
## Removing unwanted characters
def clean_lines(string):
    temp = re.sub(r'\\', '', r'-', string)
    result = temp.strip("'")
    return result

movie_description['description'] = movie_description['description'].apply(lambda x:
clean_lines(x))
movie_description['description']
movie_description.head()
'''


### Writing the DF to csv

### Sampling the data (30%)
myDF = movies_new.sample(frac = 0.3, random_state = 42)
print(myDF.shape)  # 18,603 descriptions
print(myDF)
print(type(myDF))

### Writing the DF to a csv
```

```
myDF.to_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\new data\
descriptions.csv', index = False)
infile = r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\new data\
descriptions.csv'
print(infile)
print(type(infile))


### Tokenize and Vectorize the descriptions
## Create the list of descriptions
## Keep the labels
description_LIST = []
label_LIST = []
with open(infile, 'r', encoding="utf-8") as FILE:
    FILE.readline()
    for row in FILE:
        next_label, next_description = row.split(',', 1)
        label_LIST.append(next_label)
        description_LIST.append(next_description)

    FILE.close()

print('The description list is:\n')
print(description_LIST)
print('The label list is:\n')
print(label_LIST)


## Remove all words in desciptions that match the genres.
new_description_LIST = []

for element in description_LIST:
    print(element)
    print(type(element))
    ## make into list
    all_words = element.split(" ")
    print(all_words)
    ## Now remove words that are in your topics
    new_words_LIST = []
    for word in all_words:
        print(word)
        word = word.lower()
        if word in infile['description']:
            print(word)
        else:
            new_words_LIST.append(word)
    ##turn back to string
    new_words = " ".join(new_words_LIST)
    ## Place into new_headline_LIST
    new_description_LIST.append(new_words)

## Seting the old description list to the new one
description_LIST = new_description_LIST
print(description_LIST)

### Lemmatizer
lemmer = WordNetLemmatizer()
def my_lemmer(str_input):
    words = re.sub(r"[^A-Za-z\-]", " ", str_input).lower().split()
```

```
    words = [lemmer.lemmatize(word) for word in words]
    return words

### Instantiating vectorizers
## CountVectorizer
myCV = CountVectorizer(input = 'content',
                       lowercase = True,
                       stop_words = 'english',
                       max_features = 1000,
                       tokenizer = my_lemmer)
## Bernoulli
myCV2 = TfidfVectorizer(input = 'content',
                           stop_words = 'english',
                           lowercase = True,
                           max_features = 1000,
                           binary = True,
                           tokenizer = my_lemmer)
myCV3 = CountVectorizer(input = 'content',
                           stop_words = 'english',
                           lowercase = True,
                           max_features = 1000,
                           binary = True,
                           tokenizer = my_lemmer)
## Applying vectorizers
matrix_CV = myCV.fit_transform(description_LIST)
matrix_CV2 = myCV2.fit_transform(description_LIST)
matrix_CV3 = myCV3.fit_transform(description_LIST)
print(type(matrix_CV))
print(type(matrix_CV2))
print(type(matrix_CV3))
## Retrieving column names
cols1 = myCV.get_feature_names_out()
cols2 = myCV2.get_feature_names_out()
cols3 = myCV3.get_feature_names_out()
print(cols1)
print(cols2)
print(cols3)
## Creating the dataframes
DF1 = pd.DataFrame(matrix_CV.toarray(), columns = cols1)
DF2 = pd.DataFrame(matrix_CV2.toarray(), columns = cols2)
DF3 = pd.DataFrame(matrix_CV3.toarray(), columns = cols3)
## Checking columns
DF1.columns
DF2.columns
DF3.columns
## Remove columns containing numbers
def remove_cols(myStrings):
    return any(char.isdigit() for char in myStrings)
for next_col in DF1.columns:
    logical = remove_cols(next_col)
    if(logical==True):
        DF1 = DF1.drop([next_col], axis = 1)
        DF2 = DF2.drop([next_col], axis = 1)
        DF3 = DF3.drop([next_col], axis = 1)
    ## Remove columns containing 2 words or less
    elif(len(str(next_col))<=2):
        print(next_col)
        DF1 = DF1.drop([next_col], axis = 1)
        DF2 = DF2.drop([next_col], axis = 1)
```

```
        DF3 = DF3.drop([next_col], axis = 1)

print(DF1)
print(DF2)
print(DF3)

## Removing
DF1 = DF1.drop('-year-old', axis = 1)
DF2 = DF2.drop('-year-old', axis = 1)
DF2 = DF2.drop('-', axis = 1)
DF3 = DF3.drop('-year-old', axis = 1)
DF3 = DF3.drop('-', axis = 1)
print(DF1)
print(DF2)
print(DF3)

## Adding labels to the dfs
DF1.insert(loc = 0, column = 'GENRE', value = label_LIST)
DF2.insert(loc = 0, column = 'GENRE', value = label_LIST)
DF3.insert(loc = 0, column = 'GENRE', value = label_LIST)
print(DF1) # 18,603 rows, 998 unique words
print(DF2) # 18,603 rows, 999 unique words
print(DF3)
## Writing the new dfs to their own csv files
DF1.to_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\new data\df_CV.csv',
index = False)
DF2.to_csv(r'C:\Users\cassh\OneDrive\Desktop\IST-736\Project\new data\df_TFID.csv',
index = False)

### Prepping the data for modeling
## Splitting the data into training and testing sets
train1, test1 = train_test_split(DF1, test_size = 0.3, random_state = 42)
train2, test2 = train_test_split(DF2, test_size = 0.3, random_state = 42)
train3, test3 = train_test_split(DF3, test_size = 0.3, random_state = 42)
print(train1) # 13,022 rows
print(test1)  # 5,581 rows
print(train2)
print(test2)

## Saving and removing labels DF1
train1_LABEL = train1['GENRE']
train1_DATA = train1.drop(columns='GENRE')
test1_LABEL = test1['GENRE']
test1_DATA = test1.drop(columns='GENRE')
print(train1_LABEL)
print(train1_DATA)
print(test1_LABEL)
print(test1_DATA)
## Saving and removing labels DF2
train2_LABEL = train2['GENRE']
train2_DATA = train2.drop(columns='GENRE')
test2_LABEL = test2['GENRE']
test2_DATA = test2.drop(columns='GENRE')
print(train2_LABEL)
print(train2_DATA)
print(test2_LABEL)
print(test2_DATA)
## Saving and removing labels DF3
train3_LABEL = train3['GENRE']
```

```
train3_DATA = train3.drop(columns='GENRE')
test3_LABEL = test3['GENRE']
test3_DATA = test3.drop(columns='GENRE')
print(train3_LABEL)
print(train3_DATA)
print(test3_LABEL)
print(test3_DATA)


### Fitting the Multinomial NB model
## Instantiate
NB = MultinomialNB()
## Model Fitting
NB_model1 = NB.fit(train1_DATA, train1_LABEL)
print('\nThe classes are:')
print(NB_model1.classes_)  # crime, horror, romance, sci-fi, sports
print('\nThe class counts are:')
print(NB_model1.class_count_)  # crime:3488, horror:3128, romance:5457, scifi:526,
sports:423
print('\nThe feature log probabilities are:')
print(NB_model1.feature_log_prob_)
## Prediction
prediction1 = NB_model1.predict(test1_DATA)
print(np.round(NB_model1.predict_proba(test1_DATA), 2))
print('\nThe prediction from NB is:')
print(prediction1)
print('\nThe actual labels are:')
print(test1_LABEL)
print('\nThe value counts of the actual labels are:')
print(test1_LABEL.value_counts())  # crime:2303, horror:1537, romance:1330,
scifi:219, sports:192
## Model Accuracy
accuracy1 = accuracy_score(test1_LABEL, prediction1)
print('Accuracy of MultinomialNB: {}%'.format(round(accuracy1 * 100, 2))) #
accuracy of 66.91%
print(classification_report(test1_LABEL, prediction1))
## Confusion Matrix
cm_NB1 = confusion_matrix(test1_LABEL, prediction1)
print(cm_NB1)
classes = ['crime','horror','romance','sci-fi','sports']
sns.heatmap(cm_NB1, annot = True, fmt = 'd', cmap = 'flare',
            xticklabels = classes, yticklabels = classes)
plt.title('Confusion Matrix for MultinomialNB (CV)')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()



### Fitting the Bernoulli NB model (TFID)
## Instantiate
BNB = BernoulliNB()
## Model Fitting
BNB_model1 = BNB.fit(train2_DATA, train2_LABEL)
print('\nThe classes are:')
print(BNB_model1.classes_)  # crime, horror, romance, sci-fi, sports
print('\nThe class counts are:')
print(BNB_model1.class_count_)  # crime:3488, horror:3128, romance:5457, scifi:526,
sports:423
```

```
print('\nThe feature log probabilities are:')
print(BNB_model1.feature_log_prob_)
## Prediction
prediction2 = BNB_model1.predict(test2_DATA)
print(np.round(BNB_model1.predict_proba(test2_DATA), 2))
print('\nThe prediction from NB is:')
print(prediction2)
print('\nThe actual labels are:')
print(test2_LABEL)
print('\nThe value counts of the actual labels are:')
print(test2_LABEL.value_counts())  # crime:2303, horror:1537, romance:1330,
scifi:219, sports:192
## Model Accuracy
accuracy2 = accuracy_score(test2_LABEL, prediction2)
print('Accuracy of MultinomialNB: {}%'.format(round(accuracy2 * 100, 2))) #
accuracy of 66.82%
print(classification_report(test2_LABEL, prediction2))
## Confusion Matrix
cm_BNB1 = confusion_matrix(test2_LABEL, prediction2)
print(cm_BNB1)
classes = ['crime','horror','romance','sci-fi','sports']
sns.heatmap(cm_BNB1, annot = True, fmt = 'd', cmap = 'flare_r',
            xticklabels = classes, yticklabels = classes)
plt.title('Confusion Matrix for BernoulliNB (TFID)')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()


### Fitting the Bernoulli NB model (CV)
## Instantiate
BNB = BernoulliNB()
## Model Fitting
BNB_model2 = BNB.fit(train3_DATA, train3_LABEL)
print('\nThe classes are:')
print(BNB_model2.classes_)  # crime, horror, romance, sci-fi, sports
print('\nThe class counts are:')
print(BNB_model2.class_count_)  # crime:3488, horror:3128, romance:5457, scifi:526,
sports:423
print('\nThe feature log probabilities are:')
print(BNB_model2.feature_log_prob_)
## Prediction
prediction4 = BNB_model2.predict(test3_DATA)
print(np.round(BNB_model2.predict_proba(test3_DATA), 2))
print('\nThe prediction from NB is:')
print(prediction4)
print('\nThe actual labels are:')
print(test3_LABEL)
print('\nThe value counts of the actual labels are:')
print(test3_LABEL.value_counts())  # crime:2303, horror:1537, romance:1330,
scifi:219, sports:192
## Model Accuracy
accuracy4 = accuracy_score(test3_LABEL, prediction4)
print('Accuracy of MultinomialNB: {}%'.format(round(accuracy4 * 100, 2))) #
accuracy of 66.87%
print(classification_report(test3_LABEL, prediction4))
## Confusion Matrix
cm_BNB2 = confusion_matrix(test3_LABEL, prediction4)
print(cm_BNB2)
```

```
classes = ['crime','horror','romance','sci-fi','sports']
sns.heatmap(cm_BNB2, annot = True, fmt = 'd', cmap = 'flare_r',
            xticklabels = classes, yticklabels = classes)
plt.title('Confusion Matrix for BernoulliNB (CV)')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()


### Fitting the 1st SVM model (Linear kernel)
## Instantiate
#SVM = LinearSVC(C=0.001) # accuracy score of 62.8%
SVM = LinearSVC(C=0.01) # accuracy score of 66.48
#SVM = LinearSVC(C=1) # accuracy score of 65.6%
#SVM = LinearSVC(C=8) # accuracy score of 64.65%
#SVM = LinearSVC(C=10) # accuracy score of 64.61%
#SVM = LinearSVC(C=20) # accuracy score of 64.38%
#SVM = sklearn.svm.SVC(C=10, kernel = 'linear', verbose=True, gamma='auto')
## Model fitting
SVM_model1 = SVM.fit(train1_DATA, train1_LABEL)
## Prediction
prediction_3 = SVM_model1.predict(test1_DATA)
print('\nThe prediction for SVM is:')
print(prediction_3)
## Model accuracy
accuracy_3 = accuracy_score(test1_LABEL, prediction_3)
print('Accuracy of SVM: {}%'.format(round(accuracy_3 * 100, 2))) # accuracy of
66.48%
print(classification_report(test1_LABEL, prediction_3))
## Confusion matrix
cm_SVM1 = confusion_matrix(test1_LABEL, prediction_3)
print(cm_SVM1)
sns.heatmap(cm_SVM1, annot = True, fmt = 'd', cmap = 'YlGn',
            xticklabels = classes, yticklabels = classes)
plt.title('Confusion Matrix for Linear SVM')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()


### Plotting support values
supp = {'Genre': ['crime', 'horror', 'romance', 'sci-fi', 'sports'],
        'Support': [1537, 1330, 2303, 219, 192]}
df_supp = pd.DataFrame(supp)
print(df_supp)
supp_ranked = df_supp.sort_values(by=['Support'], ascending=False)
print(supp_ranked)


supp_ranked.plot(kind = 'bar',
                 x = 'Genre',
                 y = 'Support',
                 color = 'navajowhite')
plt.title('Distribution of Support')
plt.xticks(rotation = 35, horizontalalignment = 'right')
plt.show()
```