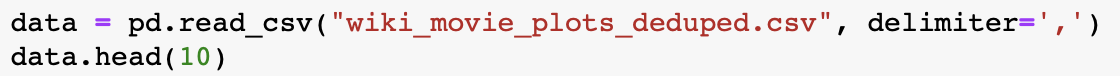
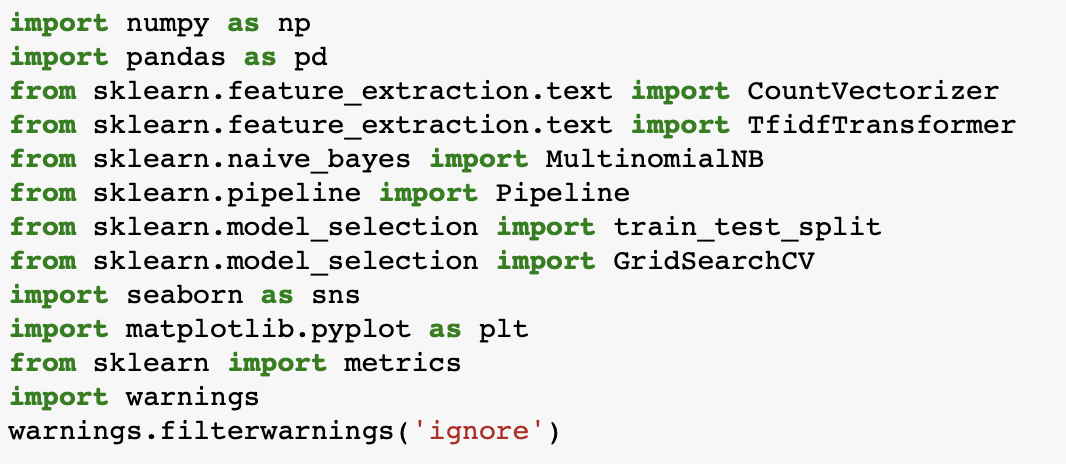
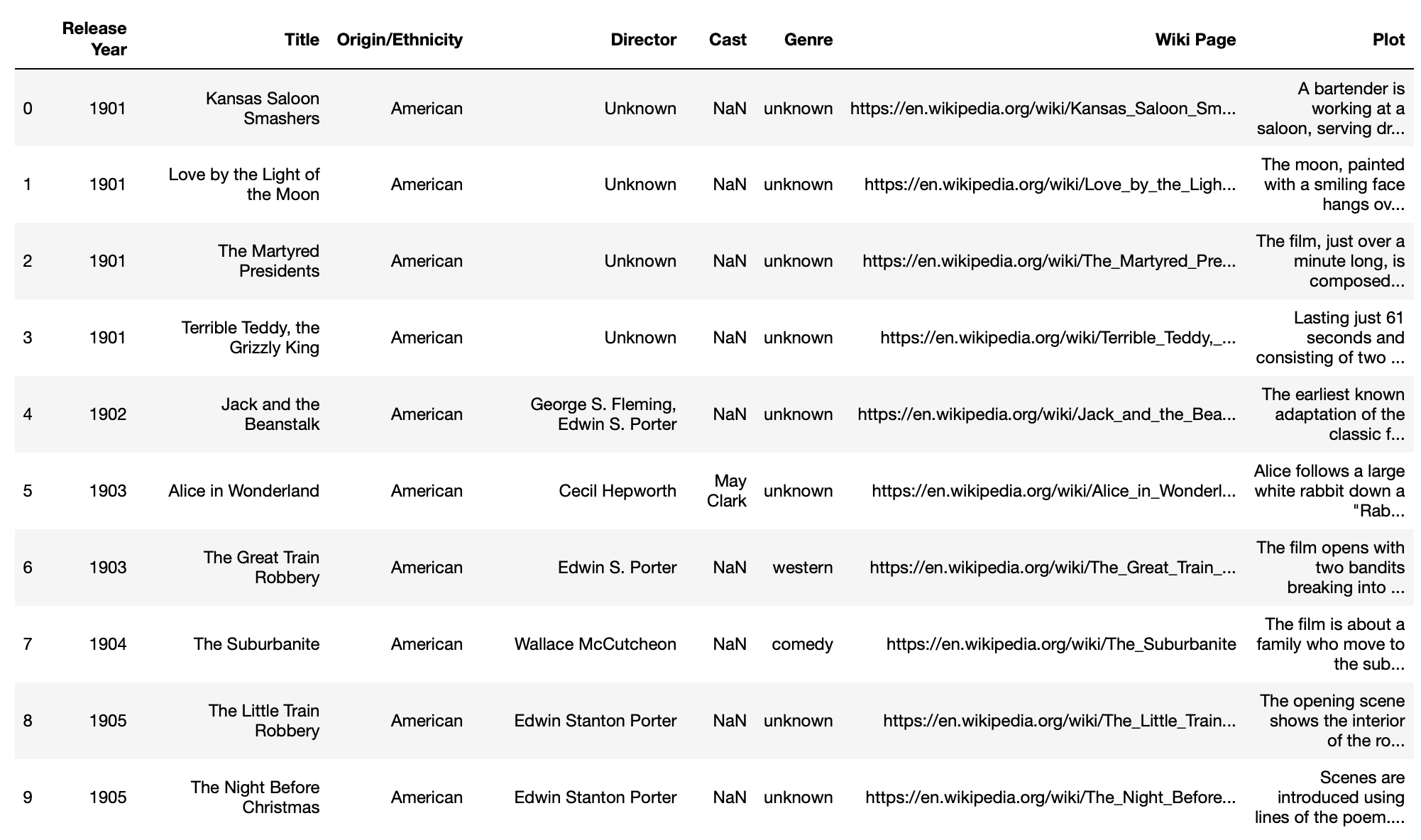
**Baseline Implementation**

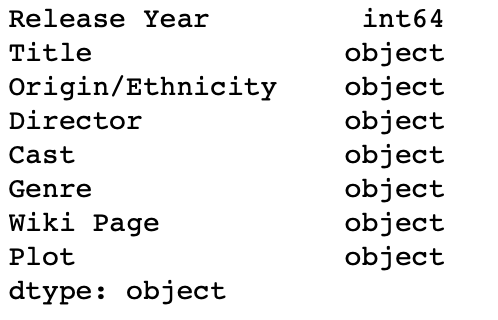
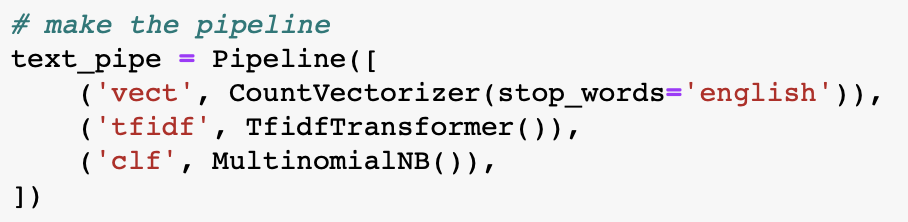


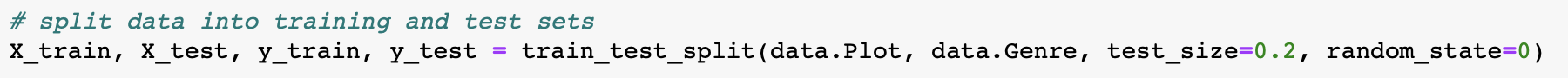
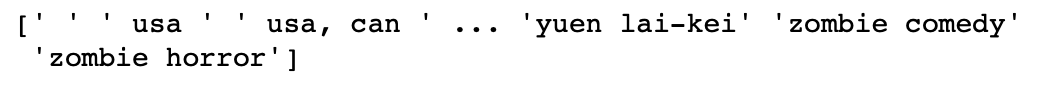
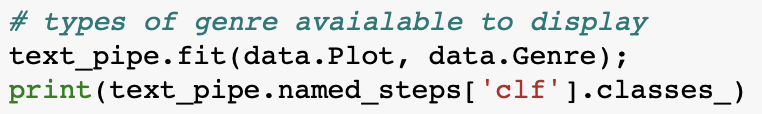


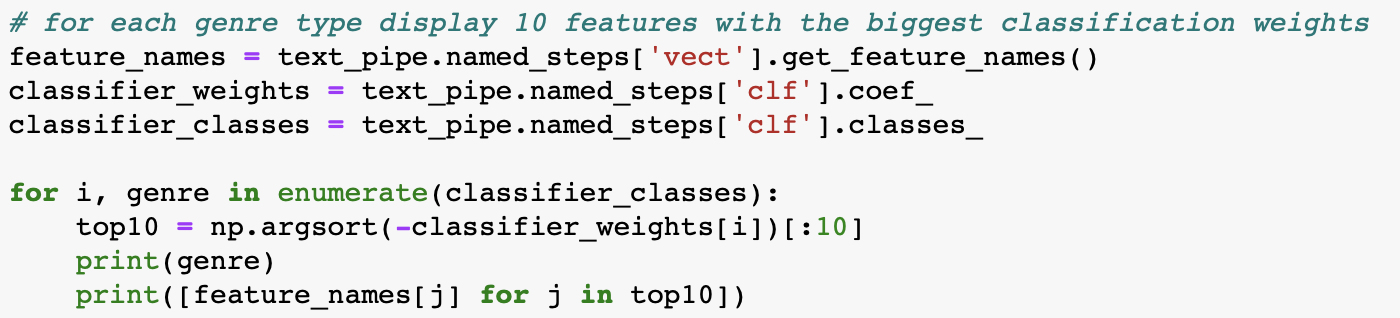
Output:

Output:

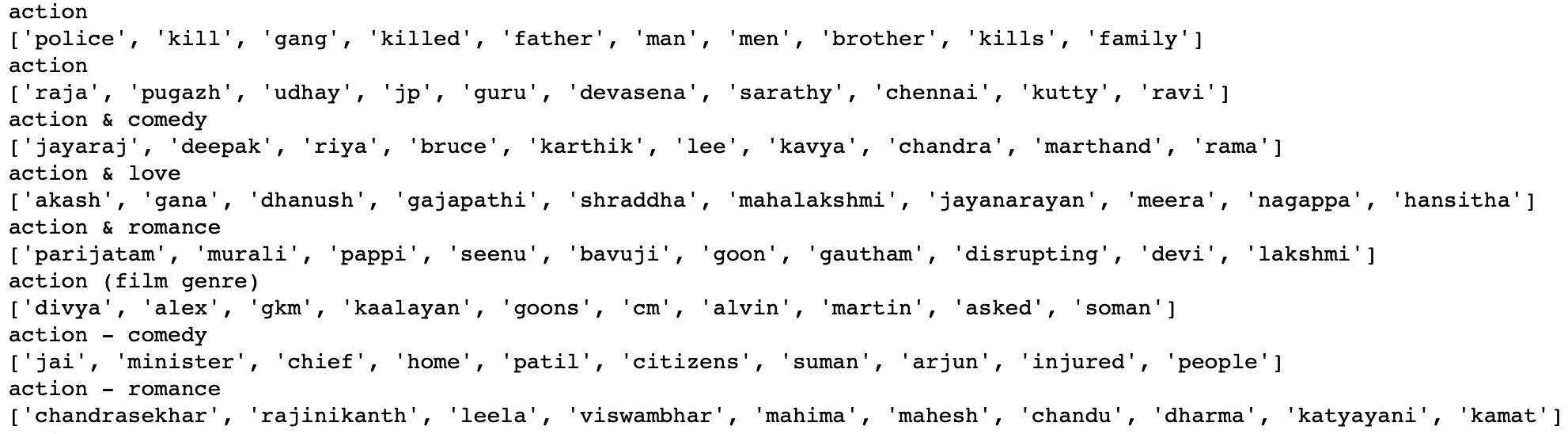
Output:

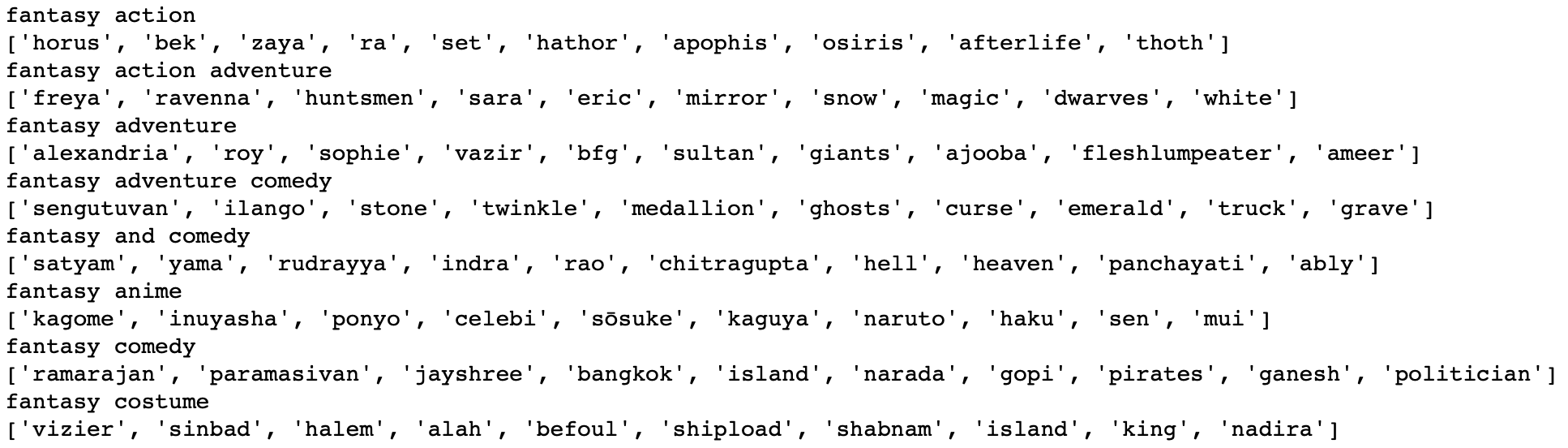


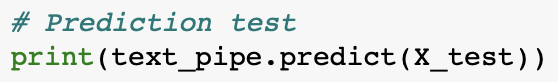
Output:



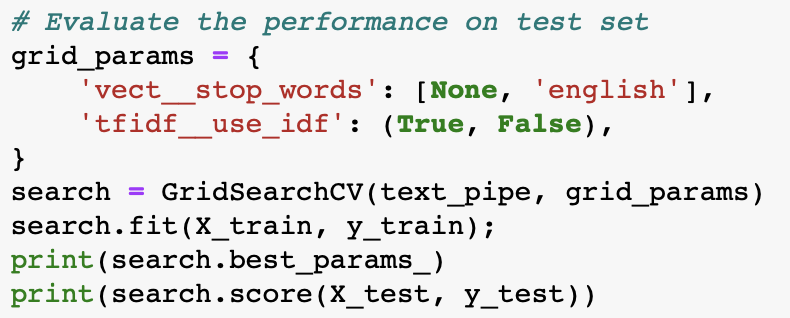
Output examples:

 ...

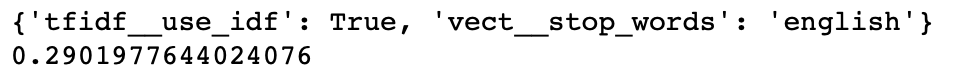


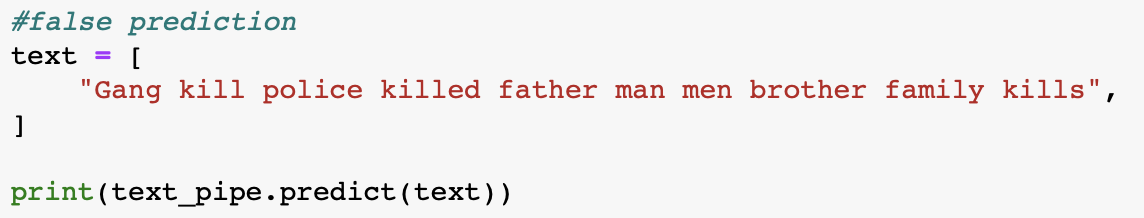


Output:



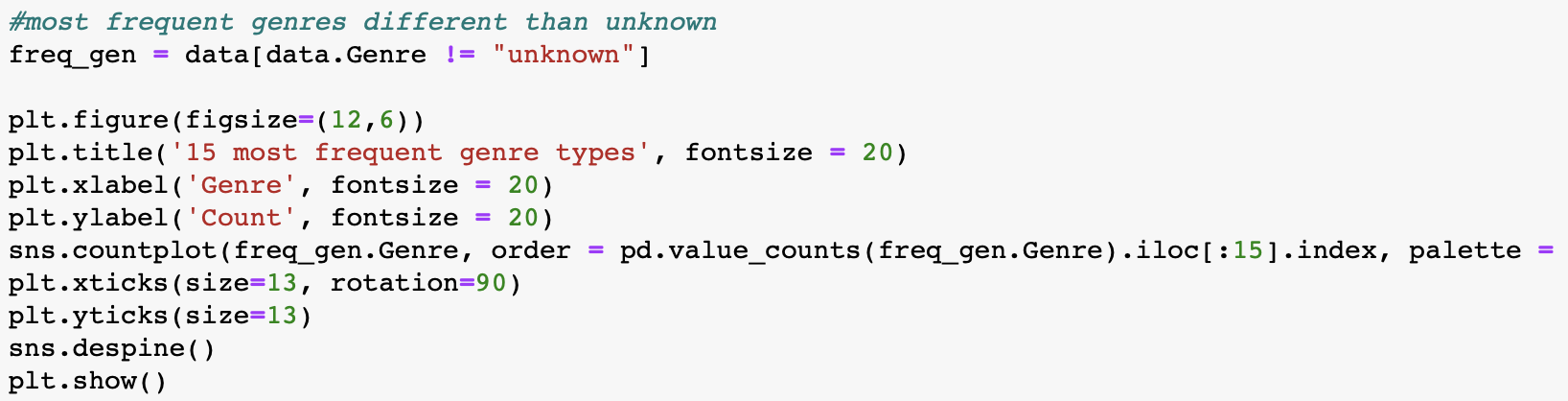
Output:

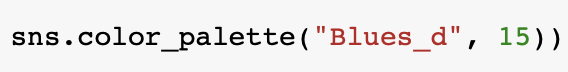




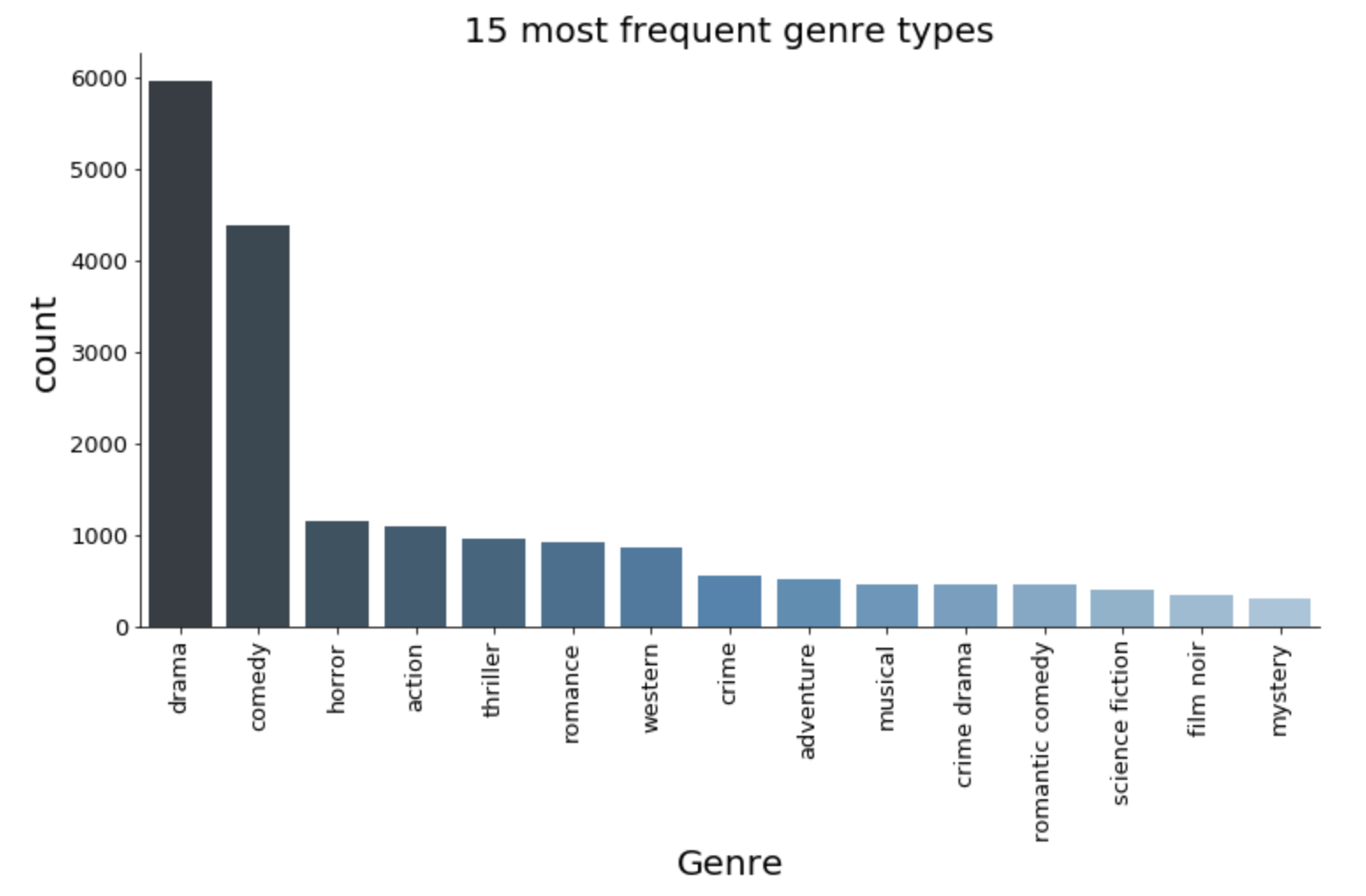
Output:

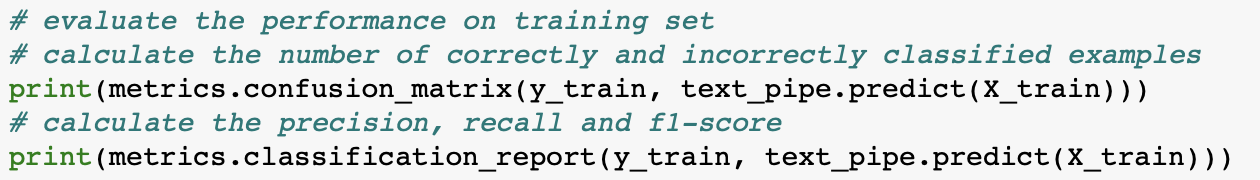


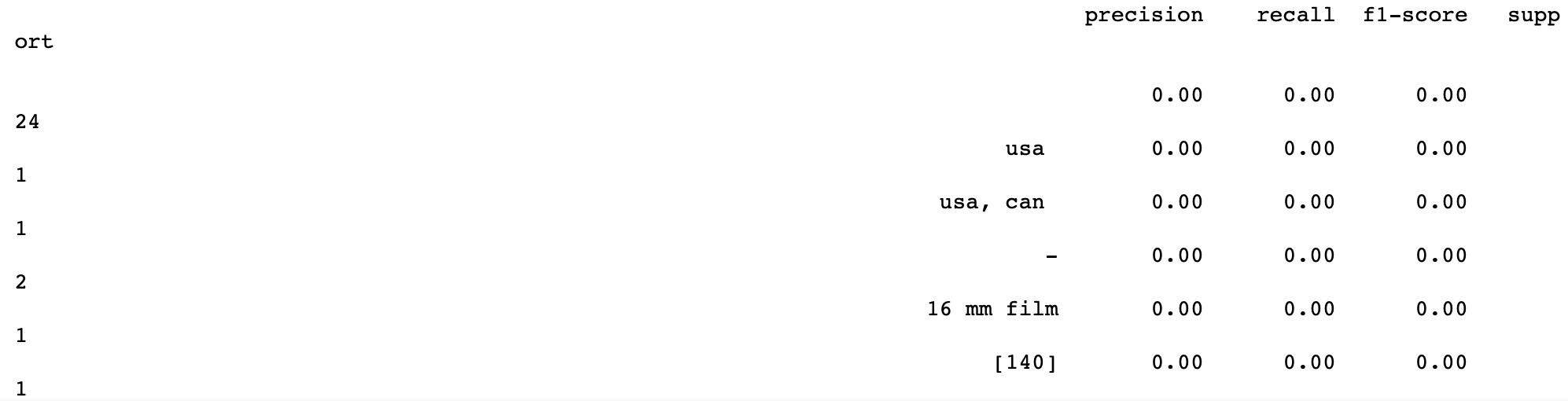


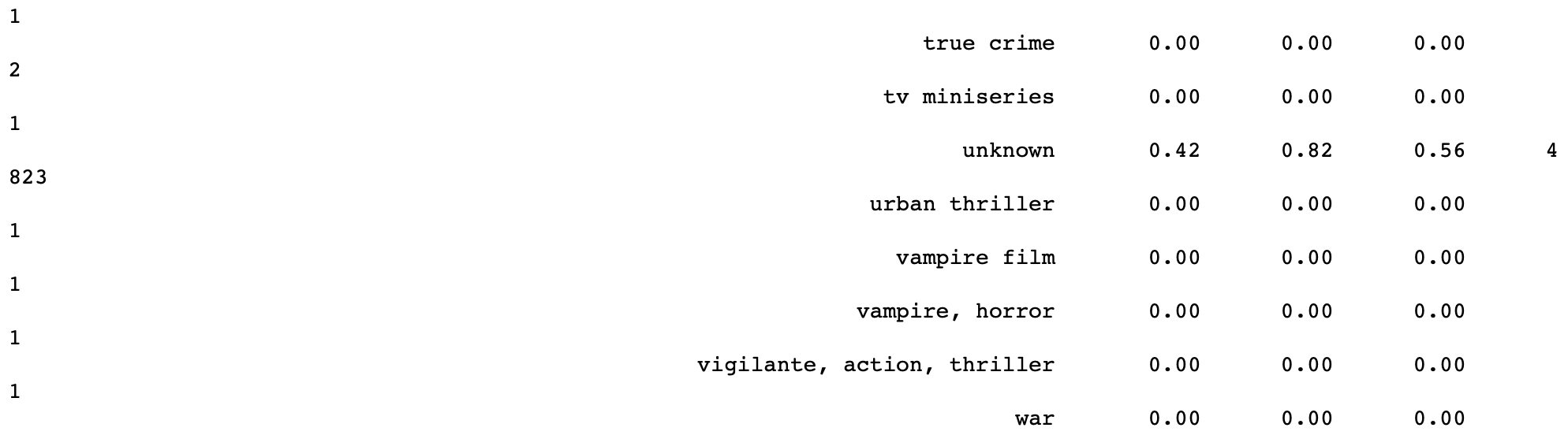


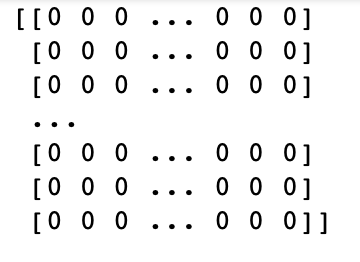
Output:





Outputs:





**BASELINE CODE:**

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import Pipeline

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn import metrics

import warnings

warnings.filterwarnings('ignore')

data = pd.read\_csv("wiki\_movie\_plots\_deduped.csv", delimiter=',')

data.head(10)

data.shape

data.dtypes

# split data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.Plot, data.Genre, test\_size=0.2, random\_state=0)

# make the pipeline

text\_pipe = Pipeline([

('vect', CountVectorizer(stop\_words='english')),

('tfidf', TfidfTransformer()),

('clf', MultinomialNB()),

])

# types of genre avaialable to display

text\_pipe.fit(data.Plot, data.Genre);

print(text\_pipe.named\_steps[‚clf'].classes\_)

# for each genre type display 10 features with the biggest classification weights

feature\_names = text\_pipe.named\_steps['vect'].get\_feature\_names()

classifier\_weights = text\_pipe.named\_steps['clf'].coef\_

classifier\_classes = text\_pipe.named\_steps['clf'].classes\_

for i, genre in enumerate(classifier\_classes):

top10 = np.argsort(-classifier\_weights[i])[:10]

print(genre)

print([feature\_names[j] for j in top10])

# Prediction test

print(text\_pipe.predict(X\_test))

# Evaluate the performance on test set

grid\_params = {

'vect\_\_stop\_words': [None, 'english'],

'tfidf\_\_use\_idf': (True, False),

}

search = GridSearchCV(text\_pipe, grid\_params)

search.fit(X\_train, y\_train);

print(search.best\_params\_)

print(search.score(X\_test, y\_test))

#false prediction - should be „action”

text = [

"Gang kill police killed father man men brother family kills",

]

print(text\_pipe.predict(text))

#diagram with most frequent genres, different than unknown

freq\_gen = data[data.Genre != "unknown"]

plt.figure(figsize=(12,6))

plt.title('15 most frequent genre types', fontsize = 20)

plt.xlabel('Genre', fontsize = 20)

plt.ylabel('Count', fontsize = 20)

sns.countplot(freq\_gen.Genre, order = pd.value\_counts(freq\_gen.Genre).iloc[:15].index, palette = sns.color\_palette("Blues\_d", 15))

plt.xticks(size=13, rotation=90)

plt.yticks(size=13)

sns.despine()

plt.show()

# evaluate the performance on training set

# calculate the number of correctly and incorrectly classified examples

print(metrics.confusion\_matrix(y\_train, text\_pipe.predict(X\_train)))

# calculate the precision, recall and f1-score

print(metrics.classification\_report(y\_train, text\_pipe.predict(X\_train)))