Film Production Clustering for Future Investment Reference

March 15, 2023

0.1 Problem Statement Brief

0.1.1 The Dataset

The dataset is “Top 500 Movies by Production Budget” and was obtained from Kaggle: https://www.kaggle.com/datasets/mitchellharrison/top-500-movies-budget

The dataset contains basic information about top productions. By classifying these top 500 films as indicators, we might be able to find a reference for future movie investment.

Let’s take a look at the data:

[1]:
```
import pandas as pd

# command below ensures matplotlib output can be included in Notebook
%matplotlib inline

df = pd.read_csv('top-500-movies.csv')
df.head()
```

[1]:
```
   rank release_date                                          title  \
0     1   2019-04-23                              Avengers: Endgame
1     2   2011-05-20  Pirates of the Caribbean: On Stranger Tides
2     3   2015-04-22                        Avengers: Age of Ultron
3     4   2015-12-16          Star Wars Ep. VII: The Force Awakens
4     5   2018-04-25                         Avengers: Infinity War

                                                 url  production_cost  \
0         /movie/Avengers-Endgame-(2019)#tab=summary        400000000
1  /movie/Pirates-of-the-Caribbean-On-Stranger-Ti…        379000000
2          /movie/Avengers-Age-of-Ultron#tab=summary        365000000
3  /movie/Star-Wars-Ep-VII-The-Force-Awakens#tab=…        306000000
4           /movie/Avengers-Infinity-War#tab=summary        300000000

   domestic_gross  worldwide_gross  opening_weekend   mpaa      genre  \
0       858373000       2797800564      357115007.0  PG-13     Action
1       241071802       1045713802       90151958.0  PG-13  Adventure
2       459005868       1395316979      191271109.0  PG-13     Action
3       936662225       2064615817      247966675.0  PG-13  Adventure
4       678815482       2048359754      257698183.0  PG-13     Action
```

```
   theaters  runtime    year
0    4662.0    181.0  2019.0
1    4164.0    136.0  2011.0
2    4276.0    141.0  2015.0
3    4134.0    136.0  2015.0
4    4474.0    156.0  2018.0
```

0.2 Data Preprocessing

0.2.1 Feature Modification

Now, let's take a closer look at the dataset to get a sense of how to modify it as needed for classification.

[2]:
```
df.info() # check if null, data type
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   rank             500 non-null    int64
 1   release_date     499 non-null    object
 2   title            500 non-null    object
 3   url              500 non-null    object
 4   production_cost  500 non-null    int64
 5   domestic_gross   500 non-null    int64
 6   worldwide_gross  500 non-null    int64
 7   opening_weekend  479 non-null    float64
 8   mpaa             492 non-null    object
 9   genre            495 non-null    object
 10  theaters         479 non-null    float64
 11  runtime          487 non-null    float64
 12  year             499 non-null    float64
dtypes: float64(4), int64(4), object(5)
memory usage: 50.9+ KB
```

As you can see, some of the fields have null values, and we will adjust them accordingly. We'll start by modifying the feature we require.

[3]:
```
df['mpaa'].unique()
```

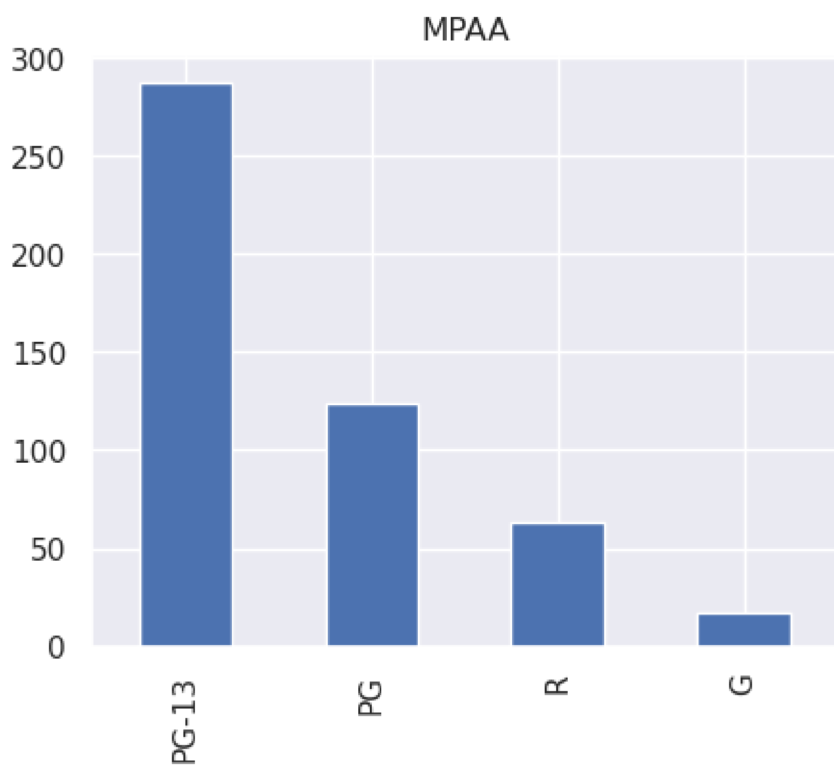
[3]:
```
array(['PG-13', nan, 'PG', 'G', 'R', 'Unrated'], dtype=object)
```

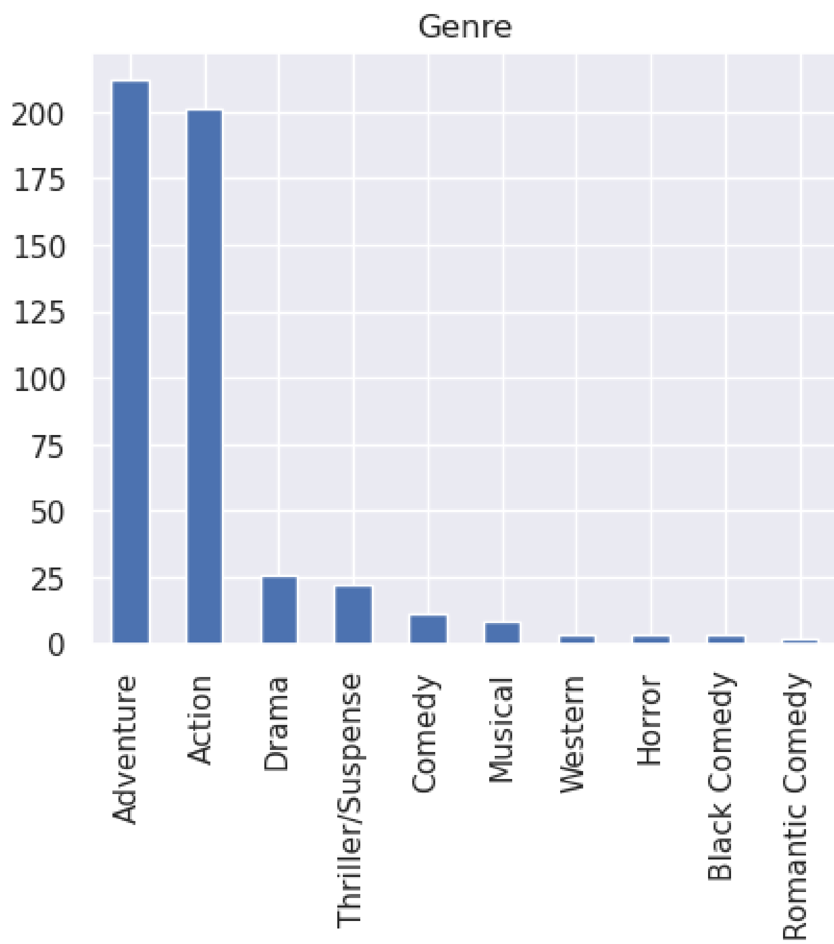
[4]:
```
df['genre'].unique()
```

[4]: array(['Action', 'Adventure', 'Musical', 'Western', 'Drama',
       'Thriller/Suspense', nan, 'Comedy', 'Horror', 'Black Comedy',
       'Romantic Comedy'], dtype=object)

These two features will be required. We can't give them a value for an undefined type, so we'll just drop it. Then, we could create a dataset reference for future use.

[5]:
```
df = df[df.mpaa != 'Unrated']
df = df.dropna(subset=['mpaa','genre'])

OG_dataset = df.iloc[:,:].values # preserve an original version of dataset for␣
↪future reference use

import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
df['mpaa'].value_counts().plot.bar(figsize=(5, 4),title='MPAA')
plt.show()
df['genre'].value_counts().plot.bar(figsize=(5, 4),title='Genre')
plt.show()
```





As the plot shows, we successfully adjust the feature we require. We could expect there will be 14 more columns after one-hot encoding(4 for "mpaa" and 10 for "genre").

We will now remove the remaining unneeded features, and we will be ready to go.

[6]:
```
del df['rank']
del df['release_date']
del df['title']
del df['url']
del df['domestic_gross']  # could be selected as a feature for pure␣
 ↪classification for movies
del df['worldwide_gross'] # could be selected as a feature for pure␣
 ↪classification for movies
del df['opening_weekend'] # could be selected as a feature for gross prediction␣
 ↪after the opening week
```

0.2.2 Adding missing values

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 491 entries, 0 to 498
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   production_cost  491 non-null    int64
 1   mpaa             491 non-null    object
 2   genre            491 non-null    object
 3   theaters         479 non-null    float64
 4   runtime          486 non-null    float64
 5   year             491 non-null    float64
dtypes: float64(3), int64(1), object(2)
memory usage: 26.9+ KB
```

We can see that the columns have been deleted, and we can concentrate on the necessary columns, paying attention to their data types.

[8]:
```
X = df.iloc[:,:].values

# Fill in numeric null data
import numpy as np
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan,strategy="median")
imputer = imputer.fit(X[:,[0,3,4,5]])
X[:,[0,3,4,5]] = imputer.transform(X[:,[0,3,4,5]])

# Be careful with the not-null but 0 values
imputer = SimpleImputer(missing_values=0,strategy="median")
imputer = imputer.fit(X[:,[0,3,4,5]])
X[:,[0,3,4,5]] = imputer.transform(X[:,[0,3,4,5]])



# One-hot encoder for object datatype
# Do it backwards for error-free concatenation
# genre
ary_dummies = pd.get_dummies(X[:,2]).values
ary_dummies = np.concatenate((X[:,:2],ary_dummies),axis=1)
X = np.concatenate((ary_dummies,X[:,3:]),axis=1)
# mpaa
ary_dummies = pd.get_dummies(X[:,1]).values
ary_dummies = np.concatenate((X[:,:1],ary_dummies),axis=1)
X = np.concatenate((ary_dummies,X[:,2:]),axis=1)
```

```
X_tree = X # preserve an original version X for Dicision Tree
```

0.3 Learning Model 1 - K-Means

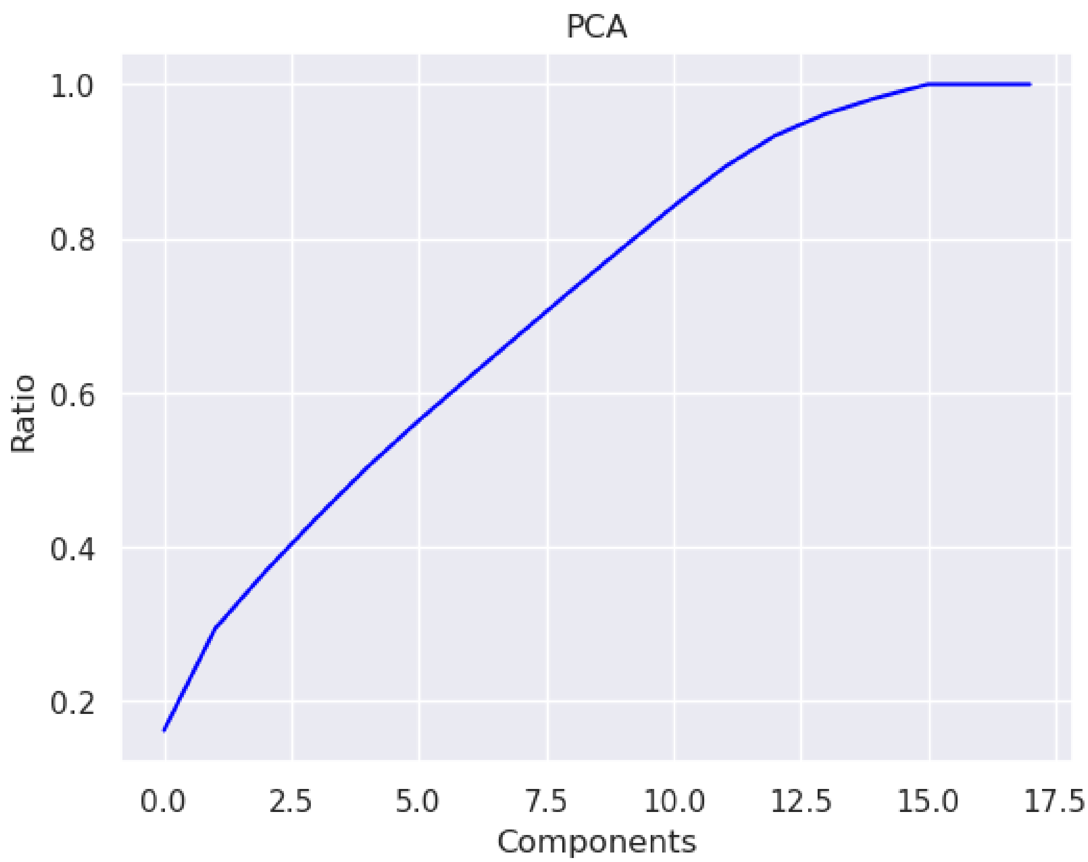
For categorising these unlabeled data. We will experiment with the unsupervised learning method K-Means. We're hoping it will provide us with clusters for future prediction.

0.3.1 Feature Scaling

[9]:
```
from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler().fit(X)
X = sc_X.transform(X)
```

0.3.2 PCA

[10]:
```
from sklearn.decomposition import PCA

# Test the number of PCA components
pca = PCA(n_components=None)
pca.fit(X)
info_covered = pca.explained_variance_ratio_
cumulated_sum = np.cumsum(info_covered)
plt.plot(cumulated_sum, color="blue")
plt.title("PCA")
plt.xlabel("Components")
plt.ylabel("Ratio")
plt.show()
```

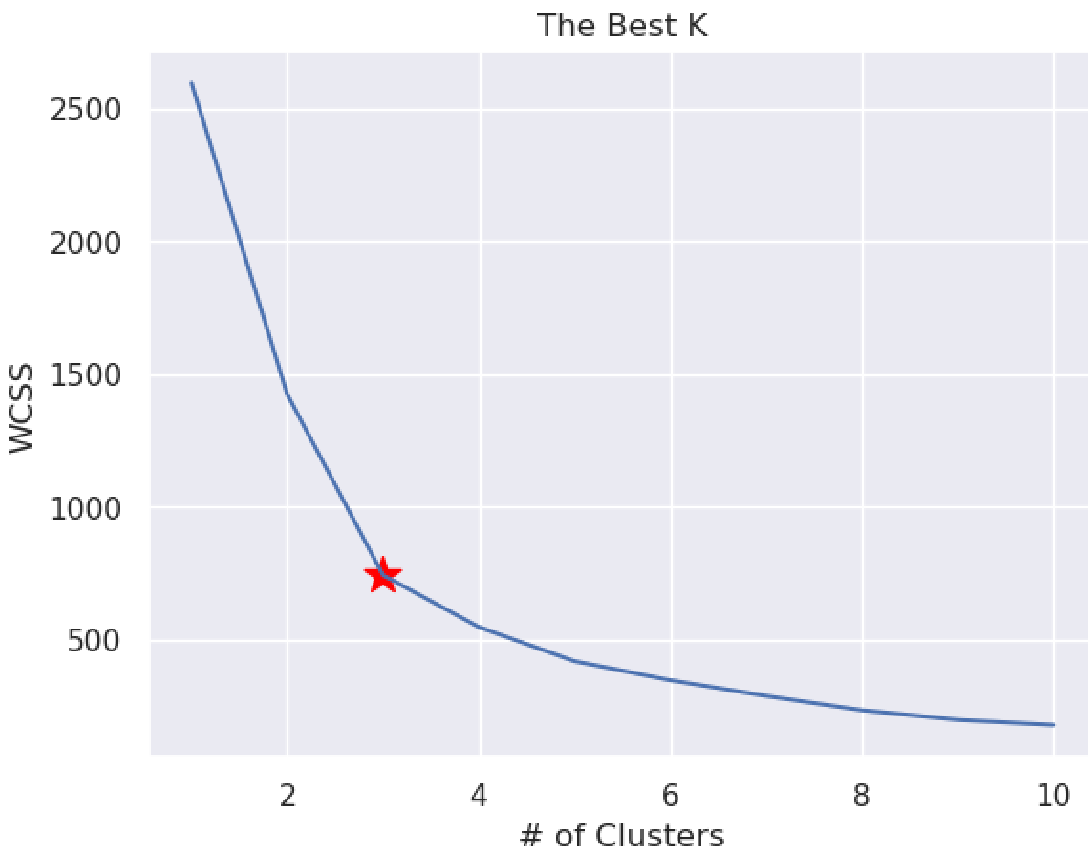


[11]:
```
# Set components to 2 for k-means to find best K
pca = PCA(n_components=2)
X = pd.DataFrame(pca.fit_transform(X))
```

0.3.3 K-Means

[12]:
```
from sklearn.cluster import KMeans

# Find Best K
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init="k-means++", random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Draw WCSS for each K
import matplotlib.pyplot as plt

plt.plot(range(1, 11), wcss)
```

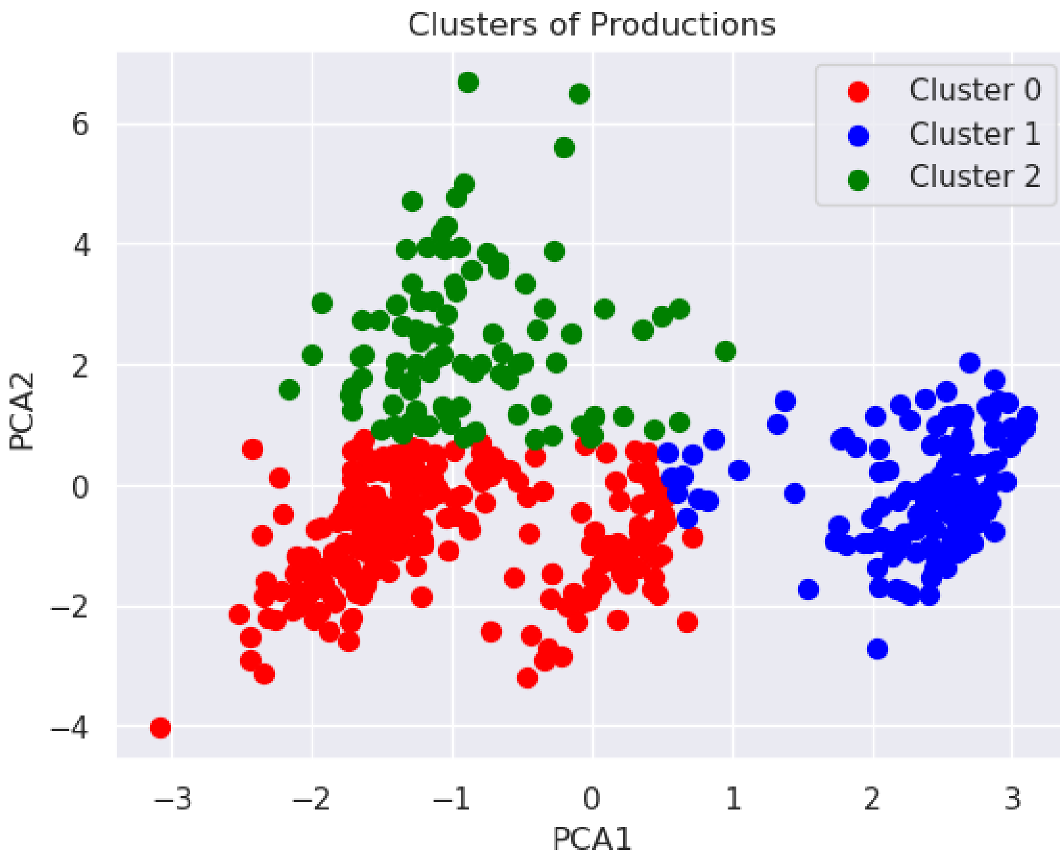
```
plt.scatter(3, wcss[2], s = 200, c = 'red', marker='*')
plt.title("The Best K")
plt.xlabel("# of Clusters")
plt.ylabel("WCSS")
plt.show()
```



0.3.4 Visualisation

[13]:
```
# Draw clusters with the value of 3
kmeans = KMeans(n_clusters=3, init="k-means++", random_state=0)
Y = kmeans.fit_predict(X)

# Draw Samples
Y_array = Y.ravel()
plt.scatter(X.iloc[Y_array==0, 0], X.iloc[Y_array==0, 1], s=50, c="red",␣
 ↪label="Cluster 0")
plt.scatter(X.iloc[Y_array==1, 0], X.iloc[Y_array==1, 1], s=50, c="blue",␣
 ↪label="Cluster 1")
```

```
plt.scatter(X.iloc[Y_array==2, 0], X.iloc[Y_array==2, 1], s=50, c="green",␣
    ↪label="Cluster 2")

# Labels & Legends
plt.title("Clusters of Productions")
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.legend(loc="best")
plt.show()
```



0.4 Learning Method 2 - Decision Tree

Finally, we could add the clustering results to the original data and determine which features lead a production to a specific group.

0.4.1 Split training and testing set

[14]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,train_size=0.
↪8,random_state=0)
```

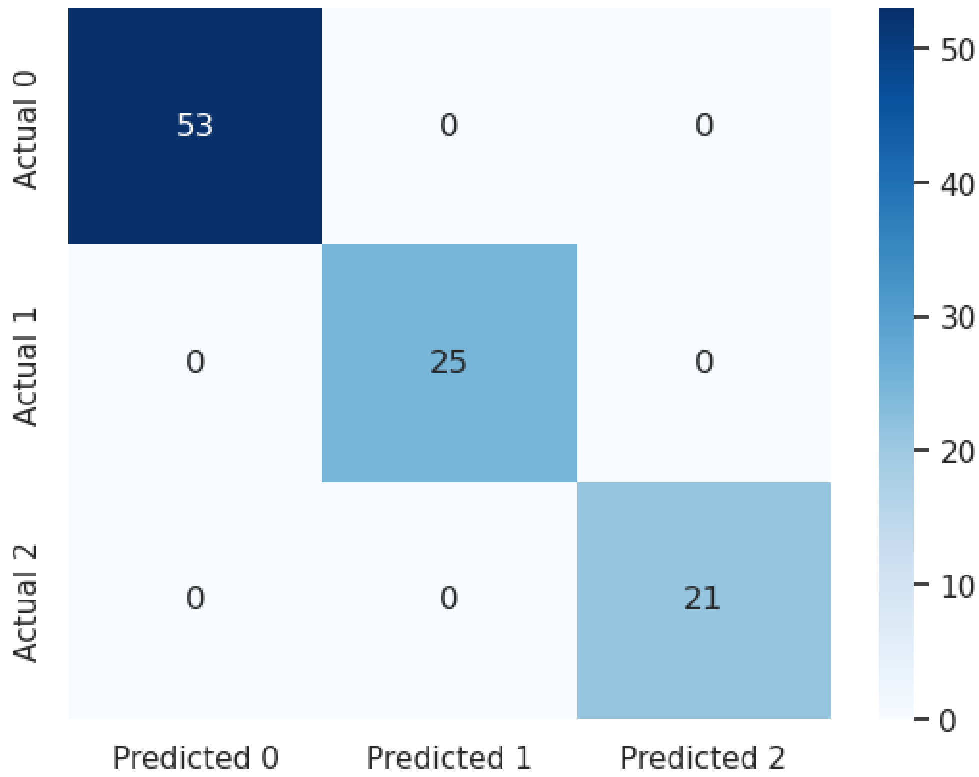
0.4.2 Decision Tree

[15]:
```
from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion="entropy",random_state=0)
classifier.fit(X_train, Y_train)
Y_pred = classifier.predict(X_test)
```

0.4.3 Validation

[16]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils.multiclass import unique_labels

print(classification_report(Y_test, Y_pred))
labels = unique_labels(Y_test)
col = [f'Predicted {label}' for label in labels]
ind = [f'Actual {label}' for label in labels]
table = pd.DataFrame(confusion_matrix(Y_test, Y_pred), columns=col, index=ind)
sns.heatmap(table, annot=True, fmt='d', cmap='Blues')
plt.show()

# K-Fold Validation
from sklearn.model_selection import cross_val_score

k_fold = 10
accuracies = cross_val_score(estimator=classifier, X=X, y=Y.ravel(),
                             scoring="accuracy", cv=k_fold, n_jobs=-1)
print("{} Folds Mean Accuracy: {:.2%}".format(k_fold, accuracies.mean()))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       1.00      1.00      1.00        25
           2       1.00      1.00      1.00        21

    accuracy                           1.00        99
   macro avg       1.00      1.00      1.00        99
weighted avg       1.00      1.00      1.00        99
```

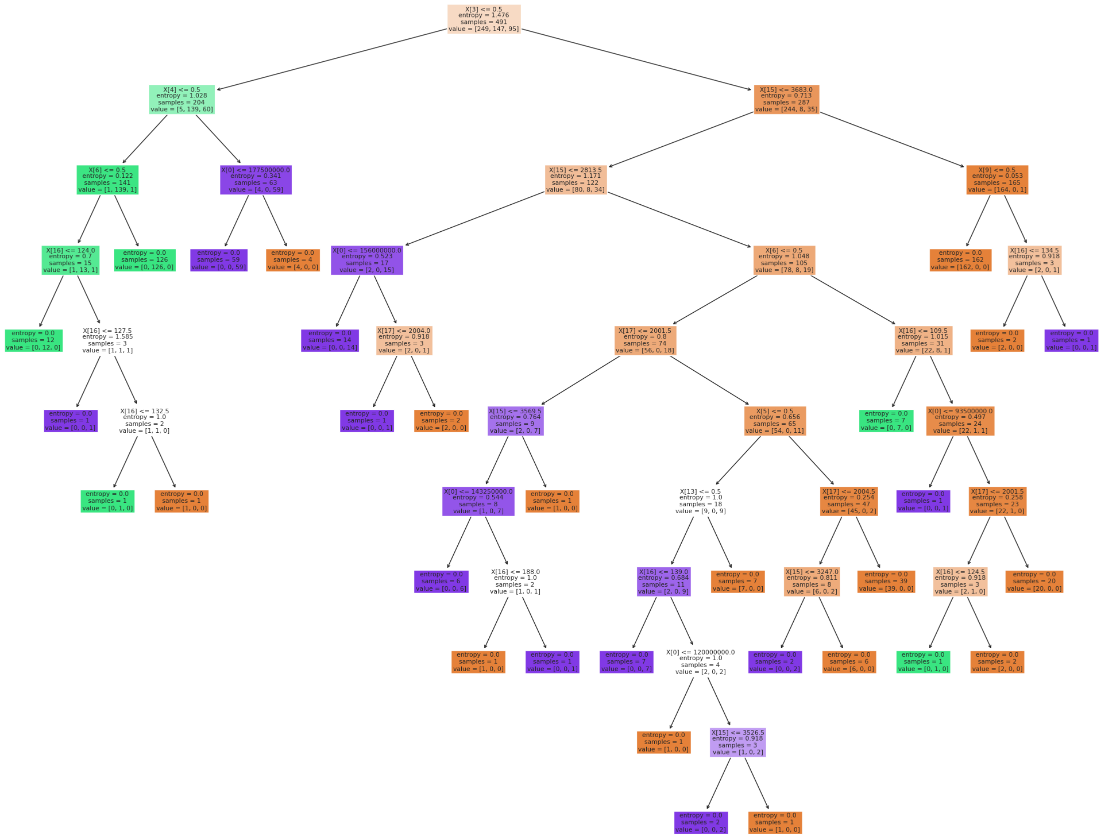



```
10 Folds Mean Accuracy: 98.78%
```

With this model, we were able to achieve a mean accuracy of 98%!

0.4.4 Visualisation

[17]:
```python
from sklearn import tree

model = classifier.fit(X_tree, Y_array)
fig = plt.figure(figsize=(25,20))
graph = tree.plot_tree(classifier, filled=True)
plt.show()

# clearer graph in text
text_representation = tree.export_text(classifier)
print(text_representation)
```



```
|--- feature_3 <= 0.50
|   |--- feature_4 <= 0.50
|   |   |--- feature_6 <= 0.50
|   |   |   |--- feature_16 <= 124.00
|   |   |   |   |--- class: 1
|   |   |   |--- feature_16 >  124.00
|   |   |   |   |--- feature_16 <= 127.50
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_16 >  127.50
|   |   |   |   |   |--- feature_16 <= 132.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_16 >  132.50
|   |   |   |   |   |   |--- class: 0
|   |   |--- feature_6 >  0.50
|   |   |   |--- class: 1
|   |--- feature_4 >  0.50
|   |   |--- feature_0 <= 177500000.00
|   |   |   |--- class: 2
|   |   |--- feature_0 >  177500000.00
|   |   |   |--- class: 0
```

```
|--- feature_3 >  0.50
|   |--- feature_15 <= 3683.00
|   |   |--- feature_15 <= 2813.50
|   |   |   |--- feature_0 <= 156000000.00
|   |   |   |   |--- class: 2
|   |   |   |--- feature_0 >  156000000.00
|   |   |   |   |--- feature_17 <= 2004.00
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_17 >  2004.00
|   |   |   |   |   |--- class: 0
|   |   |--- feature_15 >  2813.50
|   |   |   |--- feature_6 <= 0.50
|   |   |   |   |--- feature_17 <= 2001.50
|   |   |   |   |   |--- feature_15 <= 3569.50
|   |   |   |   |   |   |--- feature_0 <= 143250000.00
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_0 >  143250000.00
|   |   |   |   |   |   |   |--- feature_16 <= 188.00
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- feature_16 >  188.00
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_15 >  3569.50
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_17 >  2001.50
|   |   |   |   |   |--- feature_5 <= 0.50
|   |   |   |   |   |   |--- feature_13 <= 0.50
|   |   |   |   |   |   |   |--- feature_16 <= 139.00
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |--- feature_16 >  139.00
|   |   |   |   |   |   |   |   |--- feature_0 <= 120000000.00
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |--- feature_0 >  120000000.00
|   |   |   |   |   |   |   |   |   |--- feature_15 <= 3526.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |   |   |--- feature_15 >  3526.50
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_13 >  0.50
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_5 >  0.50
|   |   |   |   |   |   |--- feature_17 <= 2004.50
|   |   |   |   |   |   |   |--- feature_15 <= 3247.00
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |--- feature_15 >  3247.00
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_17 >  2004.50
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |--- feature_6 >  0.50
|   |   |   |   |--- feature_16 <= 109.50
```

```
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_16 >  109.50
|   |   |   |   |   |--- feature_0 <= 93500000.00
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_0 >  93500000.00
|   |   |   |   |   |   |--- feature_17 <= 2001.50
|   |   |   |   |   |   |   |--- feature_16 <= 124.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_16 >  124.50
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_17 >  2001.50
|   |   |   |   |   |   |   |--- class: 0
|   |--- feature_15 >  3683.00
|   |   |--- feature_9 <= 0.50
|   |   |   |--- class: 0
|   |   |--- feature_9 >  0.50
|   |   |   |--- feature_16 <= 134.50
|   |   |   |   |--- class: 0
|   |   |   |--- feature_16 >  134.50
|   |   |   |   |--- class: 2
```

0.5 Conclusion Brief

According to the tree and by tracing back the features, we could discover that the main clustering factors are the mpaa rating and the movie genre, followed by the production budget. Though it is too early and too little information about productions to predict the true cause that will affect the outcome of a film, the prediction states that there is a chance to find an accurate production cluster for reference.