Mellors MSDS 610 Week 5 Assignment

Load Libraries and Data

In [1]:
```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine
from sklearn.ensemble import RandomForestClassifier
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from scipy.sparse import hstack
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
```

In [2]:
```
host = r'127.0.0.1'
db = r'MSDS610'
user = r'postgres'
pw = r'postgres'
port = r'5432'
schema = r'clean'
```

In [3]:
```
db_conn = create_engine("postgresql://{}:{}@{}:{}/{}".format(user, pw, host, port, db))
```

In [4]:
```
table_name = r'movies_cleaned'
schema = r'cleaned'
```

In [5]:
```
df = pd.read_sql_table(table_name, db_conn, schema)
```

In [6]:
```
df.head(3)
```

Out[6]:

	budget	genres	keywords	overview	popularity	revenue	tagline	title	vote_average	vote_count	clean_genres	clea
0	15000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 1463, "name": "culture clash"}, {"id":...	In the 22nd century, a paraplegic Marine is di...	150.437577	19170001	Enter the World of Pandora.	Avatar	7.2	11800	id name action id name adventure id name fanta...	id c fu
1	15000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	Captain Barbossa, long believed to be dead, ha...	139.082615	19170001	At the end of the world, the adventure begins.	Pirates of the Caribbean: At World's End	6.9	4500	id name adventure id name fantasy id name action	id i ak
2	15000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 470, "name": "spy"}, {"id": 818, "name...	A cryptic message from Bond's past sends him o...	107.376788	19170001	A Plan No One Escapes	Spectre	6.3	4466	id name action id name adventure id name crime	id na n

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   budget          4803 non-null   int64
 1   genres          4803 non-null   object
 2   keywords        4803 non-null   object
 3   overview        4803 non-null   object
 4   popularity      4803 non-null   float64
 5   revenue         4803 non-null   int64
 6   tagline         4803 non-null   object
 7   title           4803 non-null   object
 8   vote_average    4803 non-null   float64
 9   vote_count      4803 non-null   int64
 10  clean_genres    4803 non-null   object
 11  clean_keywords  4803 non-null   object
 12  clean_tagline   4803 non-null   object
 13  clean_overview  4803 non-null   object
 14  profitable      4803 non-null   int64
dtypes: float64(2), int64(4), object(9)
memory usage: 563.0+ KB
```

In [8]:
```
df['clean_genres'] = df['clean_genres'].str.replace('id name', '', regex=False).str.strip()
df['clean_keywords'] = df['clean_keywords'].str.replace('id name', '', regex=False).str.strip()
```

In [9]:
```
df.head(1)
```

Out[9]:

	budget	genres	keywords	overview	popularity	revenue	tagline	title	vote_average	vote_count	clean_genres	clean_keywo
0	15000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 1463, "name": "culture clash"}, {"id":...	In the 22nd century, a paraplegic Marine is di...	150.437577	19170001	Enter the World of Pandora.	Avatar	7.2	11800	action adventure fantasy science fiction	culture cl future space space colo

In [10]:
```
df = df.drop(columns = ["genres", "keywords", "overview", "tagline"])
```

In [11]:
```
df.head(1)
```

Out[11]:

	budget	popularity	revenue	title	vote_average	vote_count	clean_genres	clean_keywords	clean_tagline	clean_overview	profital
0	15000000	150.437577	19170001	Avatar	7.2	11800	action adventure fantasy science fiction	culture clash future space war space colony...	enter the world of pandora	in the nd century a paraplegic marine is dispa...	

In [12]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   budget          4803 non-null   int64  
 1   popularity      4803 non-null   float64
 2   revenue         4803 non-null   int64  
 3   title           4803 non-null   object 
 4   vote_average    4803 non-null   float64
 5   vote_count      4803 non-null   int64  
 6   clean_genres    4803 non-null   object 
 7   clean_keywords  4803 non-null   object 
 8   clean_tagline   4803 non-null   object 
 9   clean_overview  4803 non-null   object 
 10  profitable      4803 non-null   int64  
dtypes: float64(2), int64(4), object(5)
memory usage: 412.9+ KB
```

Part 1: Analytical Question

- **Analytical Question 1 (Statistical Determination):** Can I predict whether a movie will be successful based on finances, acclaim, and popularity?
- **Analytical Question 2 (True ML):** Can I predict whether a movie will be worth funding based on its textual information (genres, overview, keywords, tagline)?

Part 2: New Features

New Feature: Financial Success (Profitable)

I had already created this feature during the clean up stage (Week 3), but I am including it here because it is a new feature and it is the target feature of my project.

To create this feature I did research and found that a movie is considered to be profitable once it makes at least 2x the budget. For the purposes of my parameters, I set the threshold as profitable = 2.5x the budget. I had created this buy creating a function that states if the "revenue" was at least 2.5x the "budget", then it is considered profitable. For the purposes of classification, I created the responses in binary, where 1 = Profitable, and 0 = Not Profitable.

Feature Justification: Since my goal is to find out if a movie is successful, I need to create a set of "rules" that define "success". As such, the first most important measurement of a film is was it profitable. As mentioned above, this feature measures if the movie made (revenue) atleast 2.5x the budget.

Below: For consistency purposes, since I decided to create other measurements of success, I decided to rename the feature I created from "profitable" to "financial_success".

In [13]:
```
df.rename(columns={"profitable": "financial_success"}, inplace=True)
```

In [14]:
```
df.financial_success.value_counts()
```

Out[14]:
```
financial_success
0    2563
1    2240
Name: count, dtype: int64
```

New Feature: Cleaned Texts

This was another set of features that I created when I did my clean up, which was to create clean text features. Given that the most important features for accomplished my goal are textbased, it was important to clean these up. I did this by lemmatizing and cleaning the text for features "genres", "keywords". "overview", and "tagline".

Feature Justification: (I created these features during the clean up process). Since my analytical question is to see if predictions of success can be made strictly on the textual information, I had to make sure that I was using clean data, so during the cleaning of the dataset, I performed a text cleaning process.

In [15]:
```
df.clean_keywords.sample(3)
```

Out[15]:
```
1233    london england  england  s  spirit  single mot...
3356             undercover  mafia  mobster  crime family
2186    alabama  martin luther king  president  black ...
Name: clean_keywords, dtype: object
```

In [16]:
```
df.clean_overview.sample(3)
```

Out[16]:
```
919     in the wilderness of british columbia two hunt...
3548    after the death of her father little voice or ...
1670    secondhand lion follows the comedic adventure ...
Name: clean_overview, dtype: object
```

In [17]:
```
df.clean_genres.sample(3)
```

Out[17]:
```
3386                             comedy
2791                     drama  romance
3132    animation  drama  family  music
Name: clean_genres, dtype: object
```

In [18]:
```
df.clean_tagline.sample(3)
```

Out[18]:
```
2997            guess who just made number two
1708    he ha minute to solve a murder his own
558                                  back work
Name: clean_tagline, dtype: object
```

New Feature: Critical Success (Critically Acclaimed)

Note: Realistically, I would set my critical success thresholds higher in real-world application (i.e vote_average = 7.5 & vote_count = 5000), but this gave significant class imbalance (only ~60 movies met that requirement), so to ensure a more balanced class, I opted to lower my critical success thresholds - which now has a class imbalance of 4:1 (not successful : successful).

Feature Justification: While I could judge a movie on its success strictly based off of its financial success, I decided to consider other ways in which I could measure success: so this included 3 types of success: financial success (above), critical success (here), and audience success (below). For critical success, I wanted to focus on my "vote" features. I wanted to specifically identify movies as critical success based on the number of votes and the vote rating. This, along with the other successes, will be used to create a "success score" that will then be used to to make recommender decisions (fund, manually assess, do not fund).

In [19]:
```
df["critical_success"] = ((df["vote_average"] >= 6.0) & (df["vote_count"] > 500)).astype(int)
```

In [20]:
```
df.head(3)
```

Out[20]:

	budget	popularity	revenue	title	vote_average	vote_count	clean_genres	clean_keywords	clean_tagline	clean_overview	fin
0	15000000	150.437577	19170001	Avatar	7.2	11800	action adventure fantasy science fiction	culture clash future space war space colony...	enter the world of pandora	in the nd century a paraplegic marine is dispa...	
1	15000000	139.082615	19170001	Pirates of the Caribbean: At World's End	6.9	4500	adventure fantasy action	ocean drug abuse exotic island east india t...	at the end of the world the adventure begin	captain barbossa long believed to be dead ha c...	
2	15000000	107.376788	19170001	Spectre	6.3	4466	action adventure crime	spy based on novel secret agent sequel mi ...	a plan no one escape	a cryptic message from bond past sends him on ...	

In [21]:
```
df.critical_success.value_counts()
```

Out[21]:
```
critical_success
0    3528
1    1275
Name: count, dtype: int64
```

New Feature: Audience Success (Popularity)

Note: Just like my "critical_success" feature, I would set my critical success thresholds higher in real-world application (i.e audience_success > 75), but this gave significant class imbalance, so to ensure a more balanced class, I opted to lower my critical success thresholds, so now it was closer to the "critical_success" with a 4:1 ratio

Feature Justification: This is the third "success" feature that I am going to use to create a "success score". For this feature, I wanted to focus on the already existing "popularity" feature. I had decided to set the success as having a popularity score above "35" (see above on my justification on this parameter).

So, all-in-all, I created 3 success measurements that I can use to create a "success score". All three of success measurements are in binary form, to make it easily help create the success score. "financial success" is a yes/no (1/0) on if the movie made at least 2.5x the budget, "critical success" is a yes/no (1/0) on if the movie scored atleast a 6.0 with over 500 votes, and the "audience success" is a yes/no (1/0) on if the popularity was above a 35. In real world application, I would hope for a much larger dataset that would allow me to create stricter parameters, but for this project, I just wanted to showcase that I knew how to set the parameters and to clarify that I understand how I would tune them in the real world.

In [22]:
```
df["audience_success"] = (df["popularity"] > 35).astype(int)
```

In [23]:
```
df.audience_success.value_counts()
```

Out[23]:
```
audience_success
0    3891
1     912
Name: count, dtype: int64
```

New Feature: Budget Score (Weighted)

Feature Justification: This feature I actually added after I had already attempted to create my "success score" and this is because when I was considering my successes and how they would effect the score - through weight - I realized that for the purpose of my final project goal, that profitibility (financial success) would be the most important (and had to have more weight than the other successes). As such, I found that there was an extremely high correlation between my success score and my financial success, to the point where the other successes didn't really matter at all. I made a number of adjustments, but then realized that unless I treated all successes equally, it would be hard to create a broader correlation among all successes. To address this, I considered additional features, and I opted to create a weighted addition to the score, which is this new feature "budget score".

This feature helps with distribution of successes to determine overall success ("worth funding") by creating a fourth and independent feature. This feature adds another element to the "success score" by praising lower budget movies - by adding a higher number to the success score - for being a lower risk (less loss for the production company if the movie fails, but easier to hit the ROI) and penalizing higher budget films for being higher risk and likely harder to achieve ROI. In this sense, profitability is still the most important, but now the other successes and the budget of the film have a better correlation to success.

Note: Much like the rest of this project, I did a lot of tuning and retuning. I opted to consider low budget films as under 10 million dollars and high budget films as over 50 million dollars. Additionally, I had to adjust and readjust the scores to try and find a good range to have films in all values.

In [24]:
```
def budget_score(budget):
    if budget < 10000000:
        return 1.00
    elif budget < 50000000:
        return 0.50
    else:
        return 0.25
df["budget_score"] = df.budget.apply(budget_score)
```

Score Breakdown:

- If Budget is < 10,000,000 dollars: Low Budget (Best ROI, Lowest Risk)
- If Budget is between 10,000,001 - 49,999,999: Medium Budget (Balanced Risk)
- If budget is >= 50,000,000: High Budget (High Risk)

In [25]:
```
df.budget_score.value_counts()
```

Out[25]:
```
budget_score
0.50    2084
1.00    2049
0.25     670
Name: count, dtype: int64
```

New Feature: Success Score

This score uses a weighted system to generate a score (based off the binary scores of the 3 success types: financial, critical, and audience). However, since I am a film production company, and I am most interested in ROI, the financial success is the most important (meaning it needs to be weighted heavier than the other two) and then voting averages (critical success) and then audience success (popularity) would be the least, with the "budget success" helping to balance the utility of the three successes, so that profitability can still be the primary measurement without neglecting the rest of the important features.

Feature Justification: This feature is very important for my goal to finding is a movie is successful (worth funding), it takes into consideration 4 features that can be used to intepret success: the financial return on the film, how many people voted and how the voted the film, how popular the film is online (how much engagement the film has), and the risk of ROI based on the budget. This is not the final step in determining success - nor is it my target feature, but it has direct correlation to my target feature "worth funding". The project up to this point has just been creating features and parameters to measure success, but doesn't tell us if the movie is worth funding, yet.

Like the other aspects of this project, I had to adjust and readjust the weighted scores until I was able to find a happy "medium" that would give me a good distribution (note: even after a bunch of adjustments, I still had one dominating score: 2.25 w/ 1600). I don't have a particular justification for the exact numbers I used, aside from the fact that I did want financial success to be weighted highest, with critical success (votes averages and scores) being second, and audience success (hype) to be third. I kept them all withing .25 of eachother so that there wasn't a huge leap in weights, since I want them all to be taken into consideration.

In [26]:
```python
df["success_score"] = (
    (1.25 * df["financial_success"])+
    (1.0 * df["critical_success"]) +
    (0.75 * df["audience_success"])+
    df["budget_score"])
```

In [27]:
```python
df.head(3)
```

Out[27]:

	budget	popularity	revenue	title	vote_average	vote_count	clean_genres	clean_keywords	clean_tagline	clean_overview	fin
0	15000000	150.437577	19170001	Avatar	7.2	11800	action adventure fantasy science fiction	culture clash future space war space colony...	enter the world of pandora	in the nd century a paraplegic marine is dispa...	
1	15000000	139.082615	19170001	Pirates of the Caribbean: At World's End	6.9	4500	adventure fantasy action	ocean drug abuse exotic island east india t...	at the end of the world the adventure begin	captain barbossa long believed to be dead ha c...	
2	15000000	107.376788	19170001	Spectre	6.3	4466	action adventure crime	spy based on novel secret agent sequel mi ...	a plan no one escape	a cryptic message from bond past sends him on ...	

In [28]:

```
df.success_score.value_counts().sort_index(ascending=False)
```

Out[28]:

```
success_score
4.00      79
3.50     180
3.25     124
3.00       8
2.75     173
2.50      60
2.25    1600
2.00     171
1.75     288
1.50     189
1.25     137
1.00     587
0.50     917
0.25     290
Name: count, dtype: int64
```

New Feature 6: Worth Funding (Target)

Note: I recognize that I could have just created a "fund" if "profitable" recommender system, but since I needed to create more features, and I wanted to include other aspects in the data frame in the recommender system, I have generated a weighted score that takes into consideration the budget, the profitability, the critical success, and the popularity into the recommendation system.

Feature Justification: This feature predicts, based on my parameters, if a film would have been worth funding, based on the success scores. This is the most important feature for my recommender system, because it is my target feature. I want to be able to determine if a movie is worth funding (by "success score" thresholds) based on provided textual data about a film that wants to be produced. This is different than recommending a new proposal, because that would rely on textual information, but before I can make the textual recommender, I need to identify which types of movies that investing in would have been worthwhile. Once I know what movies were worth funding, I can then build a recommender system that compares the textual data of successful movies to the textual data of the known movies to make a funding recommendation.

Worth Funding Scoring

- 2: Worth Funding - "Success Score >= 2.5"
- 1: Consider Funding (Manual Decision) - "Success Score >= 2.0"
- 0: Not Worth Funding - "Success Score < 2.0"

Note: Again, I adjusted the thresholds for this feature to get a more even class balance. In the real world, I would fund way fewer movies than I would reject, but I wanted to address class imbalance, given the size of my dataset (~4,800 entries). Additionally, in the real world I would want a much larger dataset: like in the 100s of thousands.

Below: you can see the pearson correlation of the four variables to determine the "success score". As anticipated, with each lower level feature there is less, but still relevant, positive correlation. The financial success is still the highest determinate, as I wanted it to be. I decided to split the score as evenly as possible among the three "worth funding" recommender responses, to attempt to address class imbalance. The best I could come up with still had imbalance, but each category had an acceptable level of responses.

Personal Insight: As I consider the success of my model and what I want it to accomplish, I am on the fence on if it would have been better to split the the recommendation into 2: fund or do not fund. I am opting to stick to the 3 responses - even though I would have a more even distribution of two - because I would imagine in the real world that there would be instances where the model wouldn't have enough information or have enough prediction power to address the ones that toe-the-line on whether or not to fund.

In [29]:
```
print(df["financial_success"].corr(df["success_score"], method="pearson"))
print(df["critical_success"].corr(df["success_score"], method="pearson"))
print(df["audience_success"].corr(df["success_score"], method="pearson"))
print(df["budget_score"].corr(df["success_score"], method="pearson"))
```

```
0.7585633201888466
0.5378743133806207
0.45572471909377277
0.34784561595560654
```

In [30]:
```
def worth_funding(success_score):
    if success_score >= 2.50:
        return 2
    elif success_score >= 2.00:
        return 1
    else:
        return 0
df["worth_funding"] = df.success_score.apply(worth_funding)
```

In [31]:
```
df.sample(3)
```

Out[31]:

	budget	popularity	revenue	title	vote_average	vote_count	clean_genres	clean_keywords	clean_tagline	clean_overview
527	0	20.632673	43312294	The Edge	6.7	349	action adventure drama	photographer grizzly bear wilderness airpla...	they were fighting over a woman when the plane...	the plane carrying wealthy charles morse crash...
3623	5000000	20.326337	142817992	Paranormal Activity 4	5.2	563	horror	garage poltergeist webcam imaginary friend ...	it closer than you think	it ha been five year since the disappearance o...
517	80000000	25.978555	123729176	The Time Machine	5.8	631	science fiction adventure action	future time machine	the greatest adventure through all time	hoping to alter the event of the past a th cen...

In [32]:
```
df.worth_funding.value_counts().sort_index(ascending=False)
```

Out[32]:
```
worth_funding
2     624
1    1771
0    2408
Name: count, dtype: int64
```

In [33]:
```
print(df["worth_funding"].corr(df["success_score"], method="pearson"))
```

```
0.9095587442595163
```

Part 3: Checking For Multicollinearity on "Successes": VIF

Task Justification: Here I want to check the correlation, through collinearity, of the relationships between my important features and target. I want to check for multicollinearity, because I want to make sure that the features are independent of each other, there is the possibility that there can be too high of a correlation between the features that I would need to drop one so that it doesn't effect the determination of success. I thought that this was important to do because there is a reasonable possibility that there can be a correlation between critical success and audience success (more hype might mean more voting and scoring).

Results: All of my features have VIF scores between (rounded) 2.0 to 2.4. Given that the closer to 1 the score is the least amount of collinearity exists, and that the higher it is (> 5.0) the more likely multicollinearity exists. These scores make me feel good, since they justify that the features are working independently of each other.

In [34]:
```
X_corr = df[["financial_success", "critical_success", "audience_success", "budget_score"]]
```

In [35]:
```
vif_data = pd.DataFrame()
vif_data["Feature"] = X_corr.columns
vif_data["VIF"] = [variance_inflation_factor(X_corr.values, i) for i in range(X_corr.shape[1])]
```

In [36]:
```
print(vif_data)
```

```
             Feature       VIF
0  financial_success  2.351302
1   critical_success  2.090426
2   audience_success  1.967599
3       budget_score  2.416434
```

Part 4: Random Forest ML: Predicting "Success" (Statistical)

Task Justification: I am aware that this task is likely not one where ML is necessary, because all predictions can be made using statistical analysis (which, from the lecture, is not the type of analytical question we want to ask). But, because I built the features and I do have a target for them - and so I can practice my ML models - I opted to include this model and its analytical question: "Can I predict whether a movie will be successful based on finances, acclaim, and popularity?".

Additionally, this is still an important model when you consider that part of my scenario is that new films will continue to be added to the dataset, where there success_scores will need to be determined so that they can help grow the dataset and provide additional data for future fund proposal requests.

Results: As expected, the model performed with 100% accuracy, as it was able to determine the formulas used to predict the the "success_score". This ML model is likely unnecessary, but is a good example of why not every task needs to be done through ML. But, this would be a good AutoML to use to generate the "success_scores" for newly added films to the dataset.

Important Note: In the next section, I do have a true ML scenario, where I test if a film is worth funding based on the textual information on the film (genres, overview, tagline, keywords).

In [37]:
```
X_stat = df[["financial_success", "critical_success", "audience_success", "budget_score"]]
y_stat = df["success_score"].astype(int)
```

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(X_stat, y_stat, test_size=0.2, random_state=4, stratify=y_stat)
```

In [39]:
```
rf_stat = RandomForestClassifier(n_estimators=500, random_state=4, class_weight="balanced")
rf_stat.fit(X_train, y_train)
```

Out[39]:
```
▾ RandomForestClassifier                    i ?

RandomForestClassifier(class_weight='balanced', n_estimators=500,
                       random_state=4)
```

In [40]:
```
y_pred = rf_stat.predict(X_test)
```

In [41]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
```

```
Model Accuracy: 1.0
```

In [42]:
```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       242
           1       1.00      1.00      1.00       240
           2       1.00      1.00      1.00       401
           3       1.00      1.00      1.00        62
           4       1.00      1.00      1.00        16

    accuracy                           1.00       961
   macro avg       1.00      1.00      1.00       961
weighted avg       1.00      1.00      1.00       961
```

Part 5: Random Forest ML: Predict Worth Funding Based on Text (True ML)

Task Justification: This is the process where I answer my analytical question 2 (my end-goal): Can I predict whether a movie will be worth funding based on its textual information (genres, overview, keywords, tagline)?

To accomplish this task, I need to focus on the textual data and make sure I take the necessary steps to ensure that it is prepped for ML modelling and learning.

Vectorize The Text

Task Justification: This is an important - and necessary - step in utilizing ML for text-based data. A Random Forest cannot process raw text, but requires it to be numerical. A vectorizer converts text into a numerical format. Additionally, it captures the importance of words.

Below: Since I have 4 features that are text based, I had to vectorize each one. As a side note as I am writing this, I am now realizing that I shaould have propably combined all these into one document for each movie. Since I already wrote the code like this, I will combine them later.

In [43]:
```
vectorizer_genres = TfidfVectorizer(stop_words="english", max_features=6000)
X_genres = vectorizer_genres.fit_transform(df["clean_genres"])

vectorizer_keywords = TfidfVectorizer(stop_words="english", max_features=6000)
X_keywords = vectorizer_keywords.fit_transform(df["clean_keywords"])

vectorizer_tagline = TfidfVectorizer(stop_words="english", max_features=6000)
X_tagline = vectorizer_tagline.fit_transform(df["clean_tagline"])

vectorizer_overview = TfidfVectorizer(stop_words="english", max_features=20000)
X_overview = vectorizer_overview.fit_transform(df["clean_overview"])
```

ML - SMOTE and Random Forest Classifier

Task Justification: SMOTE: This was actually an addition after I ran my model, because my model had class disparity issues and was not at all predicting my minor class (2, do fund). To address this I had to apply a sampling method, since using the "class_weight="balanced"" function in the RF also did not result in any predictions for the minor class. So, I decided to apply this SMOTE sampling technique to even the number of samples for each variable. To make sure that I wasn't causing overfitting issues due to the synthetic data, I made sure to only apply SMOTE to the training data, so that the testing data maintains all real-world data.

Task Justification: RF ML: This is the ML model that does the prediction for my second Analytic Question: "Can I predict whether a movie will be worth funding based on its textual information (genres, overview, keywords, tagline)?", this is the "meat and potatoes" of the project. Since the predictions fall into 1 of 3 categories, I am using the Random Forest Classifier to make the predictions. I am making sure that I am using my resampled data - from SMOTE - to train on.

In [44]:
```
X_text = hstack([X_genres, X_keywords, X_tagline, X_overview])
y = df["worth_funding"]
```

In [45]:
```
X_dense = X_text.toarray()
```

In [46]:
```
X_train, X_test, y_train, y_test = train_test_split(X_dense, y, test_size=0.2, random_state=4, stratify=y)
```

In [47]:
```
sm = SMOTE(random_state=4)
X_train_resampled, y_train_resampled = sm.fit_resample(X_train, y_train)
```

In [48]:
```
rf = RandomForestClassifier(n_estimators=2000, random_state=4,class_weight="balanced")
rf.fit(X_train_resampled, y_train_resampled)
```

Out[48]:
```
▾ RandomForestClassifier                                  i ?

RandomForestClassifier(class_weight='balanced', n_estimators=2000,
                       random_state=4)
```

In [49]:
```
y_pred = rf.predict(X_test)
```

In [50]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Model Accuracy: 0.5650364203954215

Classification Report:
               precision    recall  f1-score   support

           0       0.55      0.86      0.67       482
           1       0.61      0.36      0.45       354
           2       0.50      0.02      0.03       125

    accuracy                           0.57       961
   macro avg       0.55      0.41      0.39       961
weighted avg       0.57      0.57      0.51       961
```

RF Classification Report at Time of Reporting:

Model Accuracy: 0.5650364203954215

Classification Report: precision recall f1-score support

```
           0       0.55      0.86      0.67       482
           1       0.61      0.36      0.45       354
           2       0.50      0.02      0.03       125

    accuracy                           0.57       961
```

macro avg 0.55 0.41 0.39 961 weighted avg 0.57 0.57 0.51 961

Summary: I developed an RF Classifier to see if a film could be determined to be worth funding based on text data alone. While I did build the model, addressed sampling issues, and raised my ML parameters to reasonably high levels, my model - after many, many runs - was barely able to perform better than a 3-sided cointoss, with a 57% accuracy rating. It performed the best at predicting "do not fund" (f1 = 0.67), and the worst at predicting "do fund" (0.03). This model did terrible at the specific task I was hoping to achieve, which - at the least - identify films worth investing in.

At this point, I am not sure what to do. If I raise my parameters, I run the risk of running out of computational capacity or time commitment. I think that deep learning, like neural networks, would be better suited for this task.

Feature Importance 1 - Words

Task Justification: I was interested to see what - if any - particular words had an effect on the classification decisions. To do this, I looked at all words, across all features, to find the top 10 most important words.

In [51]:
```
feature_importances = rf.feature_importances_
```

In [52]:
```
top_features_idx = np.argsort(feature_importances)[-10:][::-1]
```

In [53]:
```
feature_names = (
    list(vectorizer_genres.get_feature_names_out()) +
    list(vectorizer_keywords.get_feature_names_out()) +
    list(vectorizer_tagline.get_feature_names_out()) +
    list(vectorizer_overview.get_feature_names_out()))
```

In [54]:
```
top_feature_names = [feature_names[i] for i in top_features_idx]
```

In [55]:
```
top_feature_importance = pd.DataFrame({
    "Feature": top_feature_names,
    "Importance": feature_importances[top_features_idx]})
```

In [56]:
```
print(top_feature_importance)
```

```
    Feature  Importance
0  included    0.014257
1      life    0.003472
2     drama    0.003207
3    comedy    0.003057
4     based    0.002796
5    action    0.002638
6  thriller    0.002596
7       new    0.002558
8        ha    0.002417
9     young    0.002356
```

Feature Importance 2 - Features

Task Justification: Just like I was interested in seeing if any individual words had high impact on the predictive decision-making, I decided to see if the features themselves played any significant role in the decision making.

In [57]:
```
num_features_genres = len(vectorizer_genres.get_feature_names_out())
num_features_keywords = len(vectorizer_keywords.get_feature_names_out())
num_features_tagline = len(vectorizer_tagline.get_feature_names_out())
num_features_overview = len(vectorizer_overview.get_feature_names_out())
```

In [58]:
```
importance_genres = np.sum(feature_importances[:num_features_genres])
importance_keywords = np.sum(feature_importances[num_features_genres:num_features_genres + num_features_keywords])
importance_tagline = np.sum(feature_importances[num_features_genres + num_features_keywords:num_features_genres + nur
importance_overview = np.sum(feature_importances[num_features_genres + num_features_keywords + num_features_tagline:]
```

In [59]:
```
feature_importance_summary = pd.DataFrame({
    "Feature Category": ["Genres", "Keywords", "Tagline", "Overview"],
    "Total Importance": [importance_genres, importance_keywords, importance_tagline, importance_overview]})
```

In [60]:
```
print(feature_importance_summary)
```

```
  Feature Category  Total Importance
0           Genres          0.031981
1         Keywords          0.294262
2          Tagline          0.088429
3         Overview          0.585329
```

Part 5: Summary

Important Note: I have included Markdown text in each section that provides more indepth information.

For this assignment, I looked to answer 2 analytical questions: one of them was a statistics-based question (not the end goal) and the other was a Machine Learning analytics question. The questions were:

- **Analytical Question 1 (Statistical Determination):** Can I predict whether a movie will be successful based on finances, acclaim, and popularity?
- **Analytical Question 2 (True ML):** Can I predict whether a movie will be worth funding based on its textual information (genres, overview, keywords, tagline)?

To answer both of these questions, I had to develop new features, where both questions relied on the other. To answer the first question, I had to develop a feature that measures the success of a film (which I called "success_score"). To be able to answer this first question, I had to develop "rules" that dictate how to measure a films success, and this is what produced my new features. To measure success, given the data available to me, I decided to measure success based on 4 factors (disregarding my "clean text" features, which was necessary text clean up to answer my second question). My four new features that would be necessary to measure my final two features ("success_score" and "worth_funding") and the "rules" I set for them are:

- "financial_success": Profitability: Yes/No (1/0). 1 = the "revenue" of the movie was at least 2.5x the "budget" of the movie.
- "critical_success": Voting Scores: Yes/No (1/0). 1 = the movie was rated at least "6.0" with over "500" votes.
- "audience_success": Popularity: Yes/No (1/0). 1 = The movie's popularity is at least "35".
- "budget_score": Reward for lower budget, penalty for higher budgets: Low/Medium/High (1.0/0.75/0.25). 1.0 = low budget, 0.25 = high budget

Note: I explain in each section why I chose those parameters. In short, I had to find a happy medium given the small size of my data and a goal of finding as even of a class balance as I can. In real-world application I would hope for a much bigger dataset and would set tighter parameters.

With the 4 new features to measure the success of a film, I had to decide how I would answer that question. Since I used binary for my "successes" and a float for my "budget_score", I thought that I would create a scoring equation that would measure the score, where the higher the score the more "successes" the film meets: with the highest score of "4" being that it met the qualifications for all 3 "successes" (1,1,1) and had a low budget (1); and the lowest score of "0.25" being that it did not meet any of the 3 "successes" (0,0,0) and had a high budget (0.25), with a mix of other scores in between. This "success_score" will be used to answer my 2nd analytical question. I also provided "weighted" values for the three "successes", because I wanted to make sure that "financial_success" was the highest considered "success", without it having a direct correlation between "financial_success" and "success_score".

After I created my "success" features and "success_score" I created a RF model to test if it could predict the "success_score", since the success score is just an statistical equation, I wanted to see if the RF model would be able to interpret the equation and show that it could implement it successfully. It did, as anticipated, score a perfect 100% accuracy -- as expected because this is a statistical analytical question.

Now that I have created my "success_score" I can now answer my second analytical question (the end-goal) that requires ML intervention. To do this, I created another feature called "worth_funding" that divided up the "success_scores" (from high to low) into 3 sections: "fund" (2), "manual decision" (1), "do not fund"(0). With this new "worth_funding" I wanted to test to see if a movie is worth funding (successful) based off the text-based information (overview, tagline, keywords, genres). I had to address class imbalance, due to a very low number of films listed as "fund", so I used SMOTE. I also continually raised my parameters to allow for more computation, leading to longer processing times, and unfortunately, the model barely performed better than a guess, with a 56% accuracy, and did terrible at predicting "fund" (2), which is not acceptable, because, at a minimum, I would want the model to predict movies worth funding the best. I would not recommend this model. And, I am not sure what I could do to improve it - other than what I did to address sampling and parameter tuning. There is a possibility that I messed something up or that this kind of recommender system is better suited for a different model, like deep learning.

As a side note: I created a copy of this and notebook and decided to run the RF model, but this time with only 2 balanced "worth_funding" variables" "fund", "do not fund" and ran the model again, and while it did perform better than this model, it still only did 60% accuracy with f-scores of 0.64 and 0.55. So, the issue is not the likely class distribution, but rather something else either related to how I processed the data or this is not a preferred model for this analytical question.

To evaluate the performance of my model I did run a Classification Report to look at the statistics (discussed above) and looked at Most Important Feature: Words and Most Important Feature: Features. Noting that my model barely did better than guessing, I still wanted to showcase my ability to look at these performance matrices. For the most important words I looked at the top 10 words that had the highest correlation to "worth_funding" decisions, and found that all the words had weak correlation, with the top word being "included" with a score of "0.014" (closer to 1 the more important it is). The most important words had very little impact on decisionmaking. After that I did the most important features, which had these scores:

- Genres: 0.031981
- Keywords: 0.294262
- Tagline: 0.088429
- Overview: 0.585329

This tells us that "Overview" (58.5%) was the highest predicator for prediction and "Keywords" was second with 29.4%. "Genre" (3.2%) and "Tagline" (8.8%) had very little effect on successful predictions.

Verdict: I would not recommend this system for practical use, while the statistics and feature engineering went well (and I actually enjoyed), the model performed poorly (barely above a guess) for actually determining if a film is worth funding based on text data. This might be better suited for a different type of model.

In [ ]:

```

```