Analysis of movie ratings

Group ID: 14
Group Members: Yujian Chen 522792, Fandi Sun 523645, Patrick Chang 517715, Charles Zhuang 522336, Colin Zhu 519644, Yufan Zhou 519097

1. Executive Summary

The film industry plays an indispensable role in people's entertainment life. After the 21st century, movie scoring platforms have gradually attracted attention, it can not only help the audience understand the movie information and synopsis, but also know whether a movie is worth watching through the platform. Therefore, this sparked our curiosity to understand which factors in movies might influence users' ratings.

2. Introduction

The film industry stands as a cornerstone in shaping the entertainment landscape, playing an indispensable role in the lives of people around the world. As we navigate through the 21st century, the emergence of movie scoring platforms has added a new dimension to the cinematic experience. These platforms not only serve as repositories of movie information and synopses but have also become invaluable tools for audiences, helping them determine the worthiness of a movie before watching.

Before we start our data analysis journey, some key concepts need explanation: 1. In the United States, a movie rating system is in place to regulate and categorize films based on suitability for different age groups. This system ensures that individuals of varying ages have access to movies deemed appropriate and aligned with their maturity levels. The levels are G, PG, PG-13, R, NC-17. 2. Three popular movie communities will be used in our dataset: IMDB, Metacritic, Rotten Tomatoes. All three platforms have user ratings for movies. 3. Some films may have won awards, categorized into three types: nominated, win, nomination. "Nominated" indicates the movie was nominated by a well-known award, "win" signifies the number of awards it received, and "nomination" means the action of being proposed or selected for an award.

In conclusion, as we embark on this cinematic data analysis journey, our exploration into the film industry's multifaceted factors and the role of movie scoring platforms promise to unveil insightful patterns and trends. By delving into the intricacies of the movie rating system, user-contributed platforms, and the diverse recognition garnered by films, we aim to decipher the factors influencing audience perceptions and cinematic success.

3. Description of the Data

Our dataset mainly consists by two sources. For the first part, we absorb the dataset about movies rating information from Kaggle. This dataset contains 5 CSV files, total size is about 6.5GB（first one is 15.5 million records and 13 columns, second one is 227K records and 10 columns）, but we call only the two files that contain the user's ratings for the movie and the files that contain information about the movie, and filtering these two files, which only

include the movies after 2000's information and rating information. Due to the first dataset still not detail enough about movies, we also actively sought for the second dataset. The second dataset was obtained from the omdbapi.com by calling its API. For this dataset, it has more detailed information about movies' directors' and actors' information, awards, and other movie rating platforms (e.g., Rotten Tomato, IMDB) rating information. The size of this dataset is about 100MB.

4. Why is Big Data

Big Data is to discover some insight, relations or fitting models by using unstructured or structured data. These two datasets fully meet the Big Data 3V requirements. Volume, these two files are around 7 GB large with almost 17 million rows, we need spend plenty of time to open that file on our own computers, and even cannot display all records in Excel. Variety, we have columns in int, double, string format, also have string in URL format. Velocity, for data processing operations such as data cleaning, data filtering, file merging, we all couldn't done that in a short time.

5. Problem Statement

We are interested in movies and want to use big data to predict movie ratings, so we asked the following five questions to help us predict movie ratings:

1. **Which country and language has the highest average movie rating?**

We have ranked the movie ratings for countries with more than 1,000 movies in total, as well as for languages with more than 500 movies in total.

2. **Which genre of movie has the highest ratings?**

The type of movie affects the movie rating. Therefore, we have sorted the average ratings of the 20 movie genres on the line.

3. **Which year's movie has the highest average rating?**

We were unsure whether time was related to movie ratings, so we calculated a curve of change in average movie ratings from 2000 to 2021.

4. **Is there a difference in scoring criteria between professional judges and public aesthetics?**

For this question, we are going to filter out movies those ImdbRating and avg_score lower than the average. Then order these movies by Awards with decreasing order. After that, we analysis top 50 outputs to show professional judges and public aesthetics have differences.

5. **What factors influence movie ratings?**

First, we convert any other than string format column to integer or float format, and I dropped the columns that cannot convert. Then, I treated the null values, and select the column which in the Year, MUBI Rating Numbers, Runtime, Win, IMDB Rating Numbers, Rated Category, Language, Quarter, Movie Popularity, Nominated, Nomination, Director Value, Genre, Category, Country. Then, split the dataset to training dataset and testing dataset, transform and create the feature column and prepare for fitting the model. Due to we think the dataset is highly discrete, and we don't want the model overfit the training dataset, so we choose to use

random forest to do the prediction. Finally, we use evaluator.evaluate() to check the test error rate, and use featureImportances to know which variables are important to the model.

6. Code

Code will be shown in the appendix part in the last page.

7. Method

We initially leverage Linux for preliminary data integration and fundamental data cleaning of our raw datasets. Subsequently, we employ PySpark to conduct more in-depth data cleaning and extraction. Utilizing the PySpark SQL package, we aggregate and structure the data to facilitate preparation for subsequent data visualization. Additionally, we leverage the PySpark ML package to implement a random tree model, aiming to predict the impact of correlation coefficients on the ultimate movie scores.

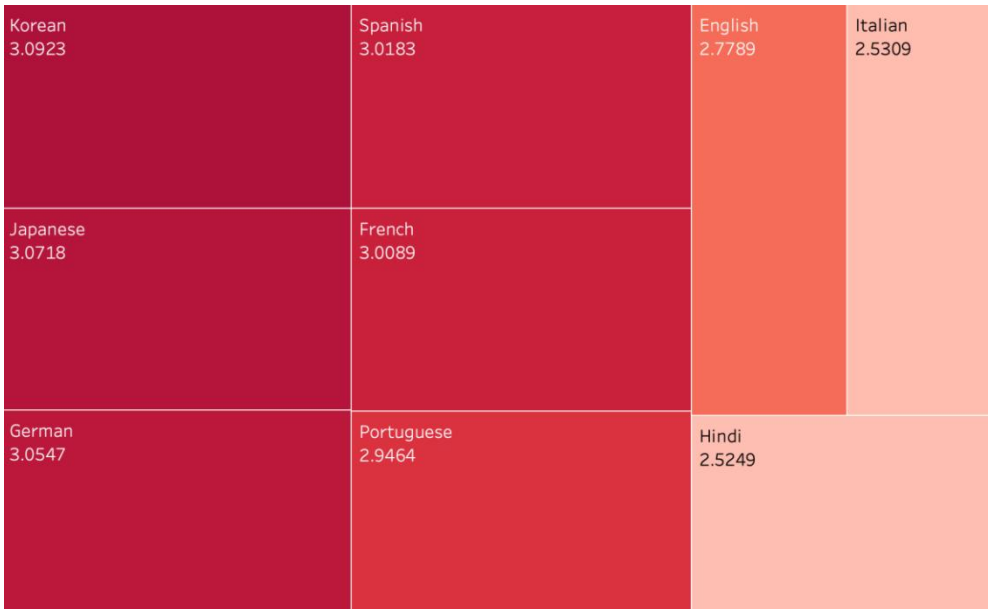
Transitioning to the visualization phase, we export the processed data from PySpark and drop the result into Tableau. In Tableau, we craft visually appealing and informative data graphs to allow audiences to easily understand our conclusions.

8. Conclusion

1. **Question whether country or language is a more important factor in movie ratings:**

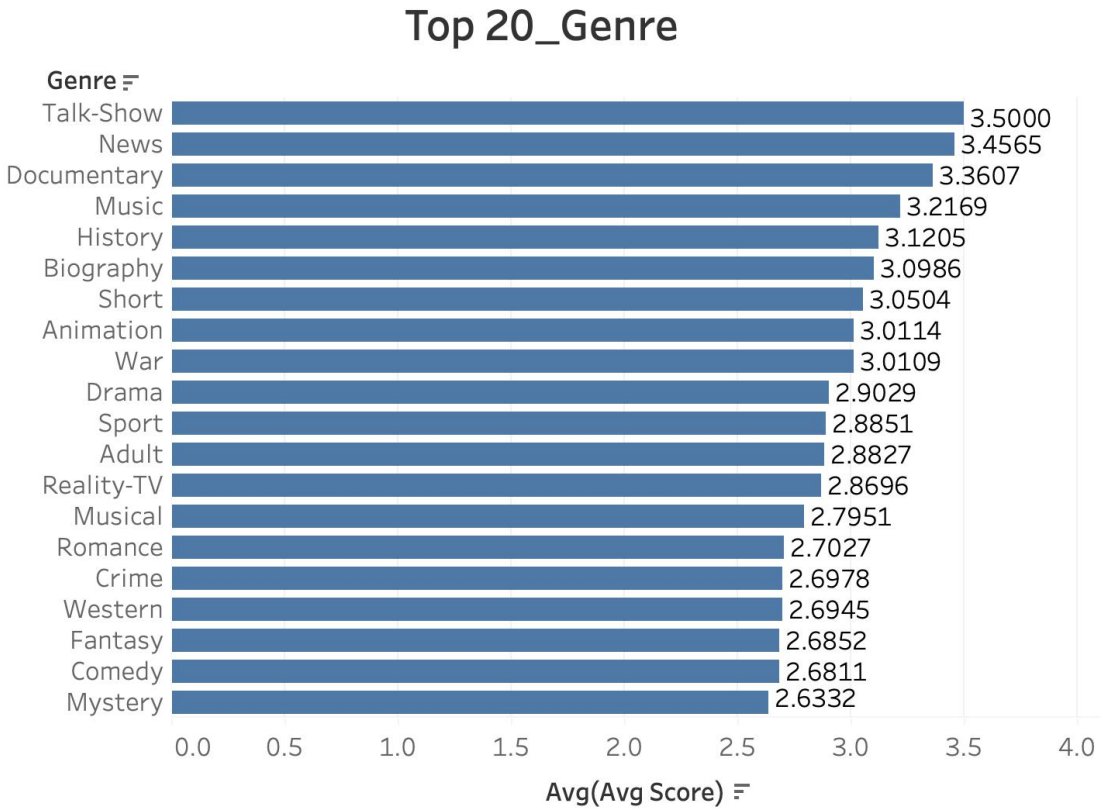
Japan, the United Kingdom, and Franch have the highest average movie ratings, while the well-known movie powerhouse, the United States, surprisingly does not make it into the top ten. We will later develop a more accurate model to analyze whether the United States is truly a movie powerhouse in ML prediction.

In terms of language, Korean, Japanese, and German have the highest ratings, with scores of 3.09, 3.07, and 3.05, respectively. However the main country using Korean, South Korea, is not in the top ten countries, so I believe there are some mistakes in country data. Therefore language is a useful factor in influencing ratings.



2. Top 5 movie genres with the highest average score:

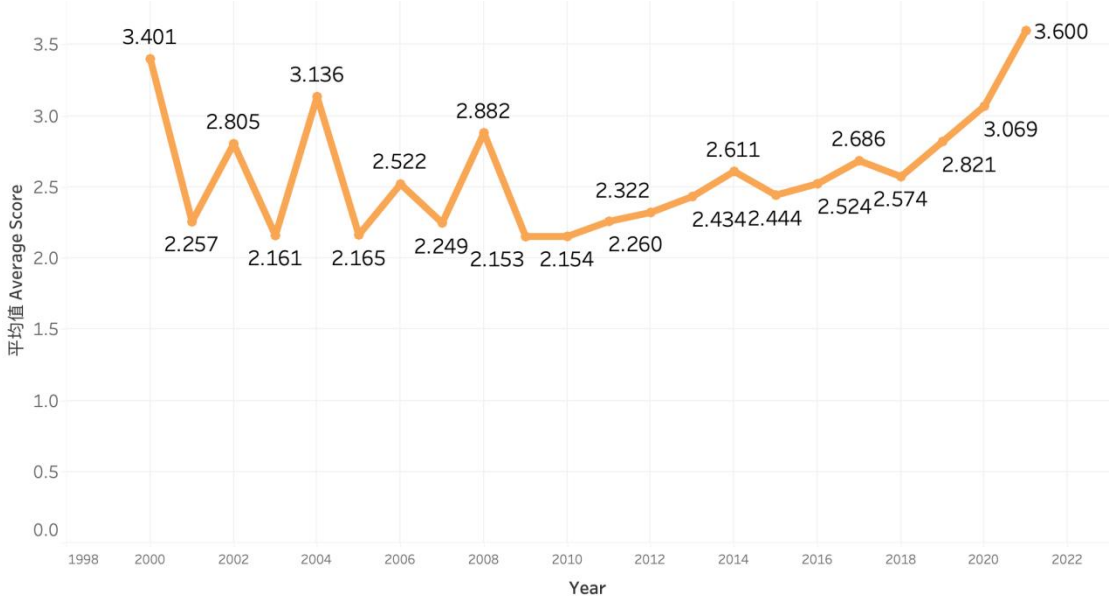
Top 5 movie genres with the highest average score are talk-show, news, documentary, music, and history. Their average scores are around 3.50, 3.46, 3.36, 3.22 and 3.12 respectively.



3. The relationship between the year of publication and the average score of a film:

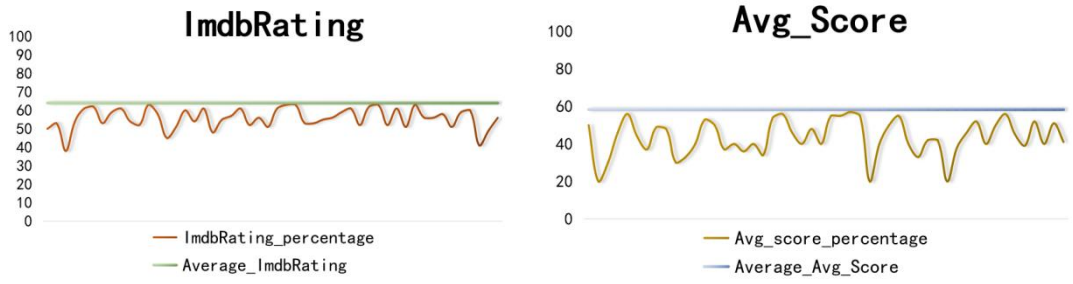
The average score of movies has shown a gradual upward trend since 2009. From this, we can infer that the quality of films has been improving since 2009. For example, Increased

Diversity in Filmmaking: A growing awareness of the importance of diversity in storytelling has led to a broader range of voices and perspectives in the film industry. Therefore, the rating of a film may have a certain relationship with the year in which the film was made.

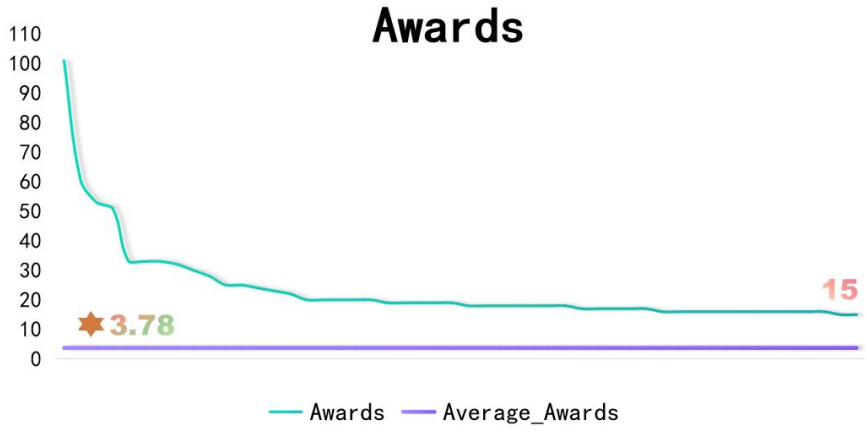


4. The difference between professional judges and public aesthetics:

The average number of Awards is 3.78, the average ImdbRating is 64.02, and the average Avg_Score is 58.4. Subsequently, we filter out movies with ImdbRating and Avg_Score below the dataset's average. Following this, we sort the remaining movies in descending order based on their Awards count. Finally, we select the top 50 movies from this sorted list to create line charts. These charts will illustrate the variance between each movie's data and the overall dataset average.



Our analysis reveals that the ImdbRating and Avg_Score of certain movies are significantly below the average. However, the number of Awards for all the filtered movies exceeds the average. Notably, the lowest Awards count among these movies is 15, roughly four times higher than the dataset's average. This observation leads us to conclude that a high number of awards for a movie does not guarantee its appeal to the general audience. This discrepancy suggests that the criteria used by professional judges and the general public's tastes differ.



5. Prediction:

From the cleaned dataset, we select 14 variables to fitting in random forest model to predict a movie's score. The top 5 most related variable with score are movie popularity, MUBI rating record numbers, time of wins awards, movie language, and time of awards nomination. We can get a conclusion that popular movies are more likely to get a high score. Meanwhile, the test error rate of the model is around 0.3.

```
Sorted Feature Importances:
movie_popularity: 0.41498796858571463
MUBI_rating_count: 0.13505116571283118
Win: 0.11037659874870955
Language_index: 0.09357271086859861
Nomination: 0.06504721712095739
imdbVotes: 0.056718842597969485
Country_index: 0.04266785718869091
Genre_index: 0.027854451626261423
Rated_index: 0.027093939510760983
Runtime: 0.01039591589436673
actor_value: 0.009031404355752325
director_value: 0.0030168060536642856
Year: 0.002402808003279697
Nominated: 0.001256642913410115
Quarter: 0.000525670819032796
```

API Data Get Code

```
import requests
import os
import json
import pandas as pd

def api_get(title,year=None):
    key = '4567a9d4'
    url = f"http://www.omdbapi.com/"


    param = {
        "apikey":key,
        "T": title,
        'y': year,
        'plot':'full',
        'type':'movie'
    }

    response = requests.get(url, params=param)
    report = response.json()
    return report

def append_record(record):
    with open('movie_info_stable.json', 'a') as f:
        json.dump(record, f)
        f.write(os.linesep)
```

```
movie_file = pd.read_csv('/Users/sunfandi/Desktop/WUSTL/DAT 560M
BD/archive/mubi_movie_data_new.csv')
# print(movie_file.columns)
title_list = list(movie_file.movie_title)
year_list = list(movie_file.movie_release_year)

for i in range(123998,len(title_list)):
    append_record(api_get(str(title_list[i]),int(year_list[i])))


print(str(title_list[0]))
print(int(year_list[0]))
```

```
# with open("sample.json", "wr") as outfile:
#     text = json.loads(outfile.read())
#     # json_object = json.dumps(api_get('Titanic', 1999))
#     # json_object = json.dumps(api_get('Titanic', 1998))
#     # outfile.write(json_object)
#     # outfile.write('\n')
#     # outfile.close()
# print(text)
```

dat560m final

December 7, 2023

[ ]:
```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col,row_number

spark = (SparkSession.builder.master("local").appName("final").getOrCreate())
sc = spark.sparkContext
```

[ ]:
```
df1 = spark.read.option('header','true').csv('df1.csv',inferSchema=True)
```

[ ]:
```
df2=spark.read.option('header','true').csv('df2.csv',inferSchema=True)
df3=spark.read.option('header','true').csv('df3.csv',inferSchema=True)
```

[ ]:
```
df1.printSchema()
df2.printSchema()
df3.printSchema()
```

```
root
 |-- movie_id: integer (nullable = true)
 |-- movie_release_year: double (nullable = true)
 |-- rating_count: integer (nullable = true)
 |-- avg_score: double (nullable = true)
 |-- movie_title: string (nullable = true)
 |-- movie_popularity: integer (nullable = true)
 |-- director_name: string (nullable = true)

root
 |-- movie_id: integer (nullable = true)
 |-- movie_release_year: double (nullable = true)
 |-- rating_count: integer (nullable = true)
 |-- avg_score: double (nullable = true)
 |-- movie_title: string (nullable = true)
 |-- movie_popularity: integer (nullable = true)
 |-- director_name: string (nullable = true)
 |-- Title: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Rated: string (nullable = true)
 |-- Released: string (nullable = true)
 |-- Runtime: string (nullable = true)
```

```
 |-- Genre: string (nullable = true)
 |-- Director: string (nullable = true)
 |-- Writer: string (nullable = true)
 |-- Actors: string (nullable = true)
 |-- Plot: string (nullable = true)
 |-- Language: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Awards: string (nullable = true)
 |-- Poster: string (nullable = true)
 |-- Ratings: string (nullable = true)
 |-- Metascore: string (nullable = true)
 |-- imdbRating: string (nullable = true)
 |-- imdbVotes: string (nullable = true)
 |-- imdbID: string (nullable = true)
 |-- Type: string (nullable = true)
 |-- DVD: string (nullable = true)
 |-- BoxOffice: string (nullable = true)
 |-- Production: string (nullable = true)
 |-- Website: string (nullable = true)
 |-- Response: string (nullable = true)
 |-- Error: string (nullable = true)

root
 |-- movie_id: integer (nullable = true)
 |-- movie_release_year: double (nullable = true)
 |-- rating_count: integer (nullable = true)
 |-- avg_score: double (nullable = true)
 |-- movie_title: string (nullable = true)
 |-- movie_popularity: integer (nullable = true)
 |-- director_name: string (nullable = true)
 |-- Title: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Rated: string (nullable = true)
 |-- Released: string (nullable = true)
 |-- Runtime: string (nullable = true)
 |-- Genre: string (nullable = true)
 |-- Director: string (nullable = true)
 |-- Writer: string (nullable = true)
 |-- Actors: string (nullable = true)
 |-- Plot: string (nullable = true)
 |-- Language: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Awards: string (nullable = true)
 |-- Poster: string (nullable = true)
 |-- Ratings: string (nullable = true)
 |-- Metascore: string (nullable = true)
 |-- imdbRating: string (nullable = true)
 |-- imdbVotes: string (nullable = true)
```

```
 |-- imdbID: string (nullable = true)
 |-- Type: string (nullable = true)
 |-- DVD: string (nullable = true)
 |-- BoxOffice: string (nullable = true)
 |-- Production: string (nullable = true)
 |-- Website: string (nullable = true)
 |-- Response: string (nullable = true)
 |-- Error: string (nullable = true)
```

[ ]:

[ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, row_number
from pyspark.sql.window import Window

df=spark.read.option('header','true').csv('df2.csv',inferSchema=True)
interested_countries = ['UK', 'France', 'Japan']
filtered_df = df.filter(df['Country'].isin(interested_countries))

grouped_df = filtered_df.groupBy('Country', 'year').avg('avg_score')

windowSpec = Window.partitionBy('Country').orderBy(grouped_df['avg(avg_score)'].
 ↪desc())

ranked_df = grouped_df.withColumn("rank", row_number().over(windowSpec))

top_rated_year_df = ranked_df.filter(col("rank") == 1).drop("rank")

windowSpec2 = Window.partitionBy('Country', 'year').orderBy(df['avg_score'].
 ↪desc())

top_movie_df = filtered_df.withColumn("rank", row_number().over(windowSpec2)) \
                          .filter(col("rank") == 1).select('Country', 'Year',␣
 ↪'movie_title', 'avg_score')
top_rated_year_df.show()
```

```
+-------+----+------------------+
|Country|year|    avg(avg_score)|
+-------+----+------------------+
| France|2018|3.1756730964467006|
|  Japan|2008|3.3823926829268287|
|     UK|2006|3.7333384615384615|
+-------+----+------------------+
```

[ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
```

```
top_france_2018 = df.filter((col('Country') == 'France') & (col('Year') ==␣
↪2018)) \
                    .orderBy(col('avg_score').desc()).limit(1) \
                    .select('movie_title', 'genre', 'avg_score')

top_japan_2008 = df.filter((col('Country') == 'Japan') & (col('Year') == 2008))␣
↪\
                    .orderBy(col('avg_score').desc()).limit(1) \
                    .select('movie_title', 'genre', 'avg_score')

top_uk_2006 = df.filter((col('Country') == 'UK') & (col('Year') == 2006)) \
                .orderBy(col('avg_score').desc()).limit(1) \
                .select('movie_title', 'genre', 'avg_score')


print("2018 Franch ")
top_france_2018.show()

print("2008 Japan ")
top_japan_2008.show()

print("2006 UK ")
top_uk_2006.show()
```

```
2018 Franch
+--------------+-----------+---------+
|   movie_title|      genre|avg_score|
+--------------+-----------+---------+
|The Real Thing|Documentary|      5.0|
+--------------+-----------+---------+

2008 Japan
+--------------------+-----+---------+
|         movie_title|genre|avg_score|
+--------------------+-----+---------+
|Shinonome Omogo I…|Short|      4.5|
+--------------------+-----+---------+

2006 UK
+--------------------+-----------+---------+
|         movie_title|      genre|avg_score|
+--------------------+-----------+---------+
|Ban the Sadist Vi…|Documentary|      5.0|
+--------------------+-----------+---------+
```

dat560m final

December 7, 2023

[ ]:
```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col,row_number

spark = (SparkSession.builder.master("local").appName("final").getOrCreate())
sc = spark.sparkContext
```

[ ]:
```
df1=spark.read.option('header','true').csv('movie.csv',inferSchema=True)
```

[ ]:
```
df1.printSchema()
```

```
root
 |-- movie_id: integer (nullable = true)
 |-- movie_release_year: double (nullable = true)
 |-- rating_count: integer (nullable = true)
 |-- avg_score: double (nullable = true)
 |-- movie_title: string (nullable = true)
 |-- movie_popularity: integer (nullable = true)
 |-- director_name: string (nullable = true)
 |-- Title: string (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Rated: string (nullable = true)
 |-- Released: string (nullable = true)
 |-- Runtime: string (nullable = true)
 |-- Genre: string (nullable = true)
 |-- Director: string (nullable = true)
 |-- Writer: string (nullable = true)
 |-- Actors: string (nullable = true)
 |-- Plot: string (nullable = true)
 |-- Language: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Awards: string (nullable = true)
 |-- Poster: string (nullable = true)
 |-- Ratings: string (nullable = true)
 |-- Metascore: string (nullable = true)
 |-- imdbRating: string (nullable = true)
 |-- imdbVotes: string (nullable = true)
 |-- imdbID: string (nullable = true)
```

```
 |-- Type: string (nullable = true)
 |-- DVD: string (nullable = true)
 |-- BoxOffice: string (nullable = true)
 |-- Production: string (nullable = true)
 |-- Website: string (nullable = true)
 |-- Response: string (nullable = true)
 |-- Error: string (nullable = true)
```

[ ]:
```
df1.select(['movie_id','avg_score','Genre']).show()
```

```
+--------+---------+--------------------+
|movie_id|avg_score|               Genre|
+--------+---------+--------------------+
|    3235|   2.6607|Comedy, Crime, My…|
|    3573|   1.7831|Crime, Horror, My…|
|    3813|   3.6356|               Drama|
|    4089|   2.6667|               Short|
|   11246|   3.0423|              Comedy|
|   13437|   2.4941|Action, Horror, T…|
|   13621|    3.878|  Documentary, Music|
|   16039|   3.3536|Crime, Drama, Mys…|
|   16405|   3.5568|Crime, Drama, Rom…|
|   20948|      3.8|       Short, Comedy|
|   21628|   3.2688|               Drama|
|   23538|   2.9309|               Drama|
|   24799|   3.9661|               Short|
|   25118|   3.6667|     Drama, Thriller|
|   26362|   4.0583|Documentary, Biog…|
|   26940|   3.2093|Drama, Romance, S…|
|   28626|   2.6667|Short, Action, Drama|
|   29389|   2.7586|Comedy, Crime, My…|
|   30144|   3.7273|               Drama|
|   30488|      3.0|Documentary, Hist…|
+--------+---------+--------------------+
only showing top 20 rows
```

[ ]:
```
from pyspark.sql.functions import explode, split, avg


df_exploded = df1.withColumn("Genre", explode(split(col("Genre"), ",\s*")))

df_filtered = df_exploded.filter(df_exploded["Genre"] != "N/A")

genre_avg_score = df_filtered.groupBy("Genre").agg(avg("avg_score"))
genre_avg_score.show()
```

```
+-----------+-----------------+
```

```
|      Genre|    avg(avg_score)|
+-----------+------------------+
|      Crime|2.6978253311258302|
|    Romance| 2.702735069955815|
|   Thriller| 2.494175249615972|
|  Adventure| 2.579166352098924|
|      Drama| 2.902885633470829|
|        War| 3.010854418604651|
|Documentary| 3.360701692708327|
| Reality-TV|2.8696437500000003|
|     Family| 2.611217677286743|
|    Fantasy|2.6851956579530354|
|      Adult| 2.882682417582418|
|    History| 3.120451739405441|
|    Mystery| 2.633249634018825|
|    Musical|2.7951456289978682|
|  Animation|  3.01137857142857|
|      Music|3.2169276051188294|
|     Horror| 2.363022076243805|
|      Short| 3.050448323571864|
|    Western|2.6944866242038215|
|  Biography| 3.098555795677802|
+-----------+------------------+
only showing top 20 rows
```

[ ]:
```
Top20_genre=genre_avg_score.orderBy(col("avg(avg_score)").desc()).limit(20)
Top20_genre.show()
```

```
+-----------+------------------+
|      Genre|    avg(avg_score)|
+-----------+------------------+
|  Talk-Show|               3.5|
|       News|3.4564978260869577|
|Documentary| 3.360701692708327|
|      Music|3.2169276051188294|
|    History| 3.120451739405441|
|  Biography| 3.098555795677802|
|      Short| 3.050448323571864|
|  Animation|  3.01137857142857|
|        War| 3.010854418604651|
|      Drama| 2.902885633470829|
|      Sport|2.8850673387096792|
|      Adult| 2.882682417582418|
| Reality-TV|2.8696437500000003|
|    Musical|2.7951456289978682|
|    Romance| 2.702735069955815|
|      Crime|2.6978253311258302|
```

```
|    Western|2.6944866242038215|
|    Fantasy|2.6851956579530354|
|     Comedy|  2.68111830103575|
|    Mystery| 2.633249634018825|
+-----------+------------------+
```

[ ]:
```
Top20_genre.write.csv('genre_movie',header=True)
```

[ ]:
```

```

Final_V7

December 7, 2023

[ ]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

spark=SparkSession.builder.appName("MovieAnalysis").getOrCreate()
df=spark.read.option('header','true').csv('movie.csv', inferSchema=True)
#Removing duplicates
df=df.dropDuplicates()
df=df.dropDuplicates(['movie_title'])
#Data Cleaning(Awards)
df=df.withColumn("Awards",expr("CASE WHEN Awards = 'N/A' THEN 0 ELSE␣
 ↪CAST(SUBSTRING_INDEX(Awards, ' ', 1) AS INT) END"))
df=df.withColumn("avg_score",col("avg_score").cast("double"))
df=df.withColumn("imdbRating",col("imdbRating").cast("double"))
df=df.filter((col("imdbRating").isNotNull())&(col("imdbRating")!= 0)&
            (col("avg_score").isNotNull())&(col("avg_score")!= 0)&
            (col("Awards").isNotNull())&(col("Awards")!= 0))
#Converting imdbRating and avg_score to percentage
df=df.withColumn("imdbRating_percentage",(col("imdbRating")*10).cast("int"))
df=df.withColumn("avg_score_percentage",(col("avg_score")*20).cast("int"))
#Calculate the average of imdbRating_percentage
average_imdbRating=df.agg(avg("imdbRating_percentage").
 ↪alias("average_imdbRating_percentage"))
average_avg_score=df.agg(avg("avg_score_percentage").
 ↪alias("average_avg_score_percentage"))
average_Awards=df.agg(avg("Awards").alias("average_Awards"))
average_imdbRating.show()
average_avg_score.show()
average_Awards.show()
```

```
+-----------------------------+
|average_imdbRating_percentage|
+-----------------------------+
|            64.02234839881899|
+-----------------------------+

+---------------------------+
|average_avg_score_percentage|
+---------------------------+
```

```
|           58.40572337042925|
+----------------------------+

+------------------+
|    average_Awards|
+------------------+
|3.7763797410856235|
+------------------+
```

[ ]:
```
#Add the filter to data frame
df=df.filter(col("imdbRating_percentage")<64)
df=df.filter(col("avg_score_percentage")<58.4)
#Selecting and Showing Results Sorted by Awards in Descending Order
result=df.
 ↪select("movie_title","Awards","imdbRating_percentage","avg_score_percentage")
result=result.orderBy(col("Awards").desc())
result_top_50 = result.limit(50)
result_top_50.show(50)
```

```
+--------------------+------+--------------------+--------------------+
|         movie_title|Awards|imdbRating_percentage|avg_score_percentage|
+--------------------+------+--------------------+--------------------+
|           Agnus Dei|   101|                  50|                  50|
|       Love Possibly|    61|                  53|                  20|
|            D-Railed|    53|                  38|                  30|
|             Capsule|    51|                  54|                  46|
|     Despicable Me 3|    33|                  62|                  44|
|     Chennai Express|    33|                  61|                  56|
|            Twilight|    33|                  53|                  37|
|      Wolf Warrior 2|    32|                  59|                  49|
|               Gypsy|    30|                  61|                  48|
|   Wonder Woman 1984|    28|                  54|                  30|
|                 F20|    25|                  52|                  33|
|         Omar and Us|    25|                  63|                  40|
|  Ae Dil Hai Mushkil|    24|                  58|                  53|
|            Mindanao|    23|                  45|                  50|
|The Twilight Saga…|    22|                  51|                  37|
|             Hostile|    20|                  54|                  36|
|The Twilight Saga…|    20|                  48|                  34|
|  Notes from Melanie|    20|                  61|                  40|
|The Trouble with …|    20|                  55|                  54|
|      Victim of Love|    20|                  60|                  40|
| Motivational Growth|    19|                  57|                  56|
|          Dark Waves|    19|                  52|                  40|
|            The Knot|    19|                  61|                  46|
|        Ek Tha Tiger|    19|                  56|                  48|
|           Long Lost|    19|                  51|                  40|
```

```
|Thesis on a Homicide|    18|                   63|                   55|
|    Dave Made a Maze|    18|                   63|                   57|
|             Embers|    18|                   53|                   55|
|        Finding Home|    18|                   53|                   20|
|       Birds of Prey|    18|                   61|                   55|
|              Money|    18|                   55|                   40|
|             Wished|    18|                   56|                   50|
|             Ainhoa|    17|                   61|                   40|
|           Heropanti|    17|                   52|                   33|
|         The Gymnast|    17|                   63|                   42|
|    Lost in Thailand|    17|                   62|                   42|
|     Before the Fall|    17|                   59|                   55|
|             Invader|    16|                   56|                   50|
|       Happy Hunting|    16|                   51|                   45|
|            Ravenous|    16|                   58|                   56|
|           Guess Who|    16|                   59|                   39|
|           Tru Loved|    16|                   56|                   40|
|Kabhi Alvida Naa …|    16|                   60|                   52|
|Oy Vey! My Son Is…|    16|                   51|                   46|
|             Peelers|    16|                   41|                   40|
|             Refugee|    16|                   52|                   20|
|The Huntsman: Win…|    16|                   61|                   38|
|     The Smiling Man|    16|                   63|                   52|
|5 Is the Perfect …|    15|                   59|                   47|
|         Housefull 2|    15|                   53|                   20|
+--------------------+------+---------------------+---------------------+
```

[ ]:
```
result_top_50.write.csv('awards_movie',header=True)
```

rating

December 5, 2023

[ ]:
```
import findspark

findspark.init()
```

[ ]:
```
import pyspark
import pandas as pd
from pyspark.sql import SparkSession
spark=SparkSession.builder.master("local").appName('Final').getOrCreate()
sc=spark.sparkContext
```

[ ]:
```
api_movie = spark.read.option('header','true').csv('./clean_api_movie.
 ↪csv',inferSchema=True)
api_movie.dtypes
api_movie.show(2)
```

```
+--------------------+----+-----+---------+-------+-----------------+----------
----------+--------------------+--------------------+--------------------+------
--+------------+--------------------+--------------------+--------------------+-
--------+----------+---------+---------+-----+----+---------+----------+-------
+--------+-----+----------------------+----------+---------------+
|               Title|Year|Rated| Released|Runtime|             Genre|
Director|              Writer|              Actors|
Plot|Language|     Country|              Awards|              Poster|
Ratings|Metascore|imdbRating|imdbVotes|   imdbID| Type|
DVD|BoxOffice|Production|Website|Response|Error|Internet Movie
Database|Metacritic|Rotten Tomatoes|
+--------------------+----+-----+---------+-------+-----------------+----------
----------+--------------------+--------------------+--------------------+------
--+------------+--------------------+--------------------+--------------------+-
--------+----------+---------+---------+-----+----+---------+----------+-------
+--------+-----+----------------------+----------+---------------+
|Cómo se hizo: La …|2007| null|     null| 25 min|Documentary; Short|Lisandro
Grané; M…|                null|Gonzalo Agulla; J…|                null|
Spanish|   Argentina|                null|                null|
[]|     null|      null|     null|tt1941495|movie|null|     null|      null|
null|    true| null|                   null|      null|           null|
|         It's Winter|2006| null|11-Jan-07| 86 min|             Drama|
```

```
Rafi Pitts|Mahmoud Dowlataba…|Mitra Hajjar; Ali…|A man is fired fr…|
Persian|Iran; France|8 wins & 2 nomina…|https://m.media-a…|[{'Source':
'Inte…|     null|       6.7|      643|tt0499166|movie|null|     null|
null|   null|    true| null|                 6.7/10|      null|           null|
+--------------------+----+-----+---------+-------+------------------+----------
----------+--------------------+--------------------+--------------------+------
--+------------+--------------------+--------------------+--------------------+-
--------+----------+---------+---------+-----+----+---------+----------+-------
+--------+-----+----------------------+----------+---------------+
only showing top 2 rows
```

[ ]:
```
movie_rating = spark.read.option('header','true').csv('./mubi_ratings_data.
  ↪csv',inferSchema=True)
```

[ ]:
```
from pyspark.sql import functions as F
result = movie_rating.groupBy('movie_id').agg(F.count('movie_id').
  ↪alias('rating_count'),F.round(F.mean('rating_score'),4).alias('avg_score'))
result.dtypes
```

[ ]:
```
[('movie_id', 'int'), ('rating_count', 'bigint'), ('avg_score', 'double')]
```

[ ]:
```
movie_list = spark.read.option('header','true').csv('./mubi_movie_data_new.
  ↪csv',inferSchema=True)
```

[ ]:
```
movie_list.dtypes
# movie_list.show(3)
```

[ ]:
```
[('movie_id', 'int'),
 ('movie_title', 'string'),
 ('movie_release_year', 'double'),
 ('movie_url', 'string'),
 ('movie_title_language', 'string'),
 ('movie_popularity', 'int'),
 ('movie_image_url', 'string'),
 ('director_id', 'string'),
 ('director_name', 'string'),
 ('director_url', 'string')]
```

[ ]:
```
merged_df = result.join(movie_list, "movie_id", "left")
# merged_df.show(3)
```

[ ]:
```
merged_df1 = merged_df.
  ↪select(['movie_id','movie_release_year','rating_count','avg_score','movie_title','movie_pop
merged_df3 = merged_df1.join(api_movie, (merged_df1.movie_title == api_movie.
  ↪Title) & (merged_df1.movie_release_year == api_movie.Year), how='inner')
```

```
merged_df4 = merged_df3.
 ↪select(['movie_title','Year','rating_count','avg_score','movie_popularity','director_name',
 ↪Movie Database','Metacritic','Rotten Tomatoes'])
# merged_df4.show(3)
```

[ ]:
```
movie = merged_df4
```

[ ]:
```
from pyspark.sql.types import FloatType

@F.udf(FloatType())
def score_convert(score):
    if score and isinstance(score, str):
        if '/' in score:
            u, d = score.split("/")
            if int(d) == 100:
                return float(u) / 20
            elif int(d) == 10:
                return float(u) / 2
        elif '%' in score:
            score = score.strip("%")
            return float(score) / 20


movie = movie.withColumn('Internet Movie Database', score_convert('Internet␣
 ↪Movie Database'))
movie = movie.withColumn('Metacritic', score_convert('Metacritic'))
movie = movie.withColumn('Rotten Tomatoes', score_convert('Rotten Tomatoes'))
```

[ ]:
```
movie = movie.withColumnRenamed('Internet Movie Database','IMDB')
movie = movie.withColumnRenamed('Rotten Tomatoes','Rotten_Tomatoes')
movie = movie.withColumnRenamed('rating_count','MUBI_rating_count')
movie = movie.withColumnRenamed('avg_score','MUBI_Score')
```

[ ]:
```
converted_scores = (
    movie
    .withColumn('num_scores', F.lit(0))
    .withColumn('score_total', F.lit(0))
    .withColumn('num_scores', F.when(F.col('IMDB').isNotNull(), F.
 ↪col('num_scores') + 1).otherwise(F.col('num_scores')))
    .withColumn('num_scores', F.when(F.col('Metacritic').isNotNull(), F.
 ↪col('num_scores') + 1).otherwise(F.col('num_scores')))
    .withColumn('num_scores', F.when(F.col('Rotten_Tomatoes').isNotNull(), F.
 ↪col('num_scores') + 1).otherwise(F.col('num_scores')))
    .withColumn('score_total', F.when(F.col('IMDB').isNotNull(), F.
 ↪col('score_total') + F.col('IMDB')).otherwise(F.col('score_total')))
    .withColumn('score_total', F.when(F.col('Metacritic').isNotNull(), F.
 ↪col('score_total') + F.col('Metacritic')).otherwise(F.col('score_total')))
```

```
    .withColumn('score_total', F.when(F.col('Rotten_Tomatoes').isNotNull(), F.
 ↪col('score_total') + F.col('Rotten_Tomatoes')).otherwise(F.
 ↪col('score_total')))
    .withColumn(
        'average_score',
        F.when(F.col('num_scores') != 0, F.col('score_total') / F.
 ↪col('num_scores')).otherwise(None)
    )
)

# Show the result
# converted_scores.show(3)
```

[ ]:
```
converted_scores.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in␣
 ↪converted_scores.columns]).show()
```

```
[Stage 521:>                                                        (0 + 1) / 1]
+-----------+----+----------------+---------+----------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
|movie_title|Year|MUBI_rating_count|MUBI_Score|movie_popularity|director_name|Ra
ted|Released|Runtime|Genre|Director|Writer|Actors|Plot|Language|Country|Awards|P
oster|Metascore|imdbRating|imdbVotes|Type|  DVD|BoxOffice|IMDB|Metacritic|Rotten
_Tomatoes|num_scores|score_total|average_score|
+-----------+----+----------------+---------+----------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
|          0|   0|               0|       29|               0|
1|28788|    3992|   1422|  296|     237|  6268|  5025|3300|    2599|    351|
19612|  4390|    40467|      6188|     4657|   4|28843|    42197|6188|
40303|          36354|         0|          0|         6106|
+-----------+----+----------------+---------+----------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
```

[ ]:
```
from pyspark.sql.types import StringType
@F.udf(StringType())
def filter_1(genre):
    if ';' in genre and isinstance(genre, str):
        return genre.split(';')[0]
    elif ';' not in genre and isinstance(genre, str):
```

```
        return genre
    else:
        return 'null'


converted_genre = converted_scores.withColumn('Genre',filter_1('Genre'))
```

[ ]:
```
movie_with_score = converted_genre.filter(converted_scores.average_score.
 ↪isNotNull())
```

[ ]:
```
movie_with_genre = movie_with_score.filter(converted_scores.Genre.isNotNull())
movie_with_genre = movie_with_genre.filter(converted_scores.Plot.isNotNull())
```

[ ]:
```
movie_with_genre.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in␣
 ↪converted_scores.columns]).show()
# movie_with_genre.write.csv('movie_with_genre', header=True)
```

```
+-----------+----+-----------------+----------+-----------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
|movie_title|Year|MUBI_rating_count|MUBI_Score|movie_popularity|director_name|Ra
ted|Released|Runtime|Genre|Director|Writer|Actors|Plot|Language|Country|Awards|P
oster|Metascore|imdbRating|imdbVotes|Type|  DVD|BoxOffice|IMDB|Metacritic|Rotten
_Tomatoes|num_scores|score_total|average_score|
+-----------+----+-----------------+----------+-----------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
|          0|   0|                0|        23|                0|
1|21431|    1642|    392|    0|      79|  3932|  2493|   0|    1252|    119|
14106|  1105|    32915|        59|       43|   0|21489|    34643|  59|
32750|          28833|         0|          0|            0|
+-----------+----+-----------------+----------+-----------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
```

[ ]:
```
# movie_with_genre = spark.read.option('header','true').csv('./
 ↪movie_with_genre',inferSchema=True)
```

[ ]:
```
from pyspark.sql import functions as F
from pyspark.sql.types import StringType
@F.udf(StringType())
def filter_1(genre):
```

```
    if genre and ';' in genre and isinstance(genre, str):
        return genre.split(';')[0]
    elif genre and ';' not in genre and isinstance(genre, str):
        return genre
    else:
        return 'null'
```

[ ]:
```
from pyspark.sql.types import ArrayType, IntegerType
from pyspark.sql.functions import udf
import re

converted_direct = movie_with_genre.withColumn('director_name',␣
 ↪filter_1('director_name'))
converted_direct = converted_direct.filter(converted_direct.director_name.
 ↪isNotNull())


converted_ct = converted_direct.withColumn('Country', filter_1('Country'))
converted_ct = converted_ct.withColumn('Country', F.when(F.
 ↪col('Country')=='USA', 'United States').otherwise(F.col('Country')))
converted_ct = converted_ct.filter(converted_ct.Country.isNotNull())

converted_ln = converted_ct.withColumn('Language', filter_1('Language'))
converted_ln = converted_ln.filter(converted_ln.Language.isNotNull())

converted_ln = converted_ln.withColumn('Actors', filter_1('Actors'))

# converted_ln.write.csv('single_value', header=True)
```

[ ]:
```
converted_ln.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in␣
 ↪converted_ln.columns]).show()
```

```
+----------+----+----------------+----------+---------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
|movie_title|Year|MUBI_rating_count|MUBI_Score|movie_popularity|director_name|Ra
ted|Released|Runtime|Genre|Director|Writer|Actors|Plot|Language|Country|Awards|P
oster|Metascore|imdbRating|imdbVotes|Type|  DVD|BoxOffice|IMDB|Metacritic|Rotten
_Tomatoes|num_scores|score_total|average_score|
+----------+----+----------------+----------+---------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
|          0|   0|               0|        23|              0|
0|21431|    1642|    392|    0|      79|  3932|     0|   0|       0|      0|
```

```
14106|  1105|    32915|        59|       43|   0|21489|    34643|  59|
32750|          28833|         0|          0|           0|
+----------+----+----------------+---------+---------------+-------------+--
---+--------+-------+-----+--------+------+------+----+--------+-------+------+-
-----+---------+----------+---------+----+-----+---------+----+----------+------
---------+----------+-----------+-------------+
```

[ ]:
```
from pyspark.sql.functions import udf, col
from pyspark.sql.types import StringType, ArrayType, IntegerType
import re

@udf(StringType())
def format_award(awards):
    if awards and isinstance(awards, str):
        awards = awards.lower()
        if 'win' in awards and 'nomination' not in awards and 'nominated' not␣
 ↪in awards:
            return '0,' + awards + ',0'
        elif 'win' in awards and 'nomination' in awards and 'nominated' not in␣
 ↪awards:
            return '0,' + awards
        elif 'win' in awards and 'nomination' in awards and 'nominated' in␣
 ↪awards:
            return awards
        elif 'win' not in awards and 'nomination' in awards and 'nominated' not␣
 ↪in awards:
            return '0,0,' + awards
        elif 'win' not in awards and 'nomination' in awards and 'nominated' in␣
 ↪awards:
            return awards[:int(len(awards)/2)] + ',0,' + awards[int(len(awards)/
 ↪2):]
        elif 'win' not in awards and 'nomination' not in awards and 'nominated'␣
 ↪in awards:
            return awards + ',0,0'
    else:
        return '0,0,0'


@udf(ArrayType(IntegerType()))
def award(awards):
    if awards and isinstance(awards, str):
        return list(map(int, re.findall(r'\d+', awards)))
    else:
        return [0, 0, 0]

df = converted_ln.withColumn('Awards', format_award(col('Awards')))
```

```
award_clean = df.withColumn('Nominated', award(col('Awards'))[0])
award_clean = award_clean.withColumn('DVD_have', F.when(F.col('DVD').
 ↪isNotNull(), 1).otherwise(0))
award_clean = award_clean.withColumn('Win', award(col('Awards'))[1])
award_clean = award_clean.withColumn('Nomination', award(col('Awards'))[2])
# award_clean.show(3)
```

[ ]:
```
pre_analy = award_clean.
 ↪select(['Year','MUBI_rating_count','movie_popularity','director_name','Rated','Released','R
```

[ ]:
```
pre_analy.dtypes
```

[ ]:
```
[('Year', 'string'),
 ('MUBI_rating_count', 'bigint'),
 ('movie_popularity', 'int'),
 ('director_name', 'string'),
 ('Rated', 'string'),
 ('Released', 'string'),
 ('Runtime', 'string'),
 ('Genre', 'string'),
 ('Actors', 'string'),
 ('Language', 'string'),
 ('Country', 'string'),
 ('average_score', 'double'),
 ('MUBI_Score', 'double'),
 ('Nominated', 'int'),
 ('Win', 'int'),
 ('Nomination', 'int'),
 ('imdbVotes', 'string'),
 ('DVD_have', 'int')]
```

[ ]:
```
pre_analy = pre_analy.withColumn(
    "Runtime",
    F.regexp_replace(col("Runtime"), "[^0-9]", ""),
)
```

[ ]:
```
pre_analy = pre_analy.na.fill("Not Rated", subset=["Rated"])
# pre_analy.show(5)
```

[ ]:
```
pre_analy.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in pre_analy.
 ↪columns]).show()
```

```
+----+----------------+---------------+-------------+-----+--------+-------+--
---+------+--------+-------+-------------+----------+---------+---+----------+--
-------+--------+
|Year|MUBI_rating_count|movie_popularity|director_name|Rated|Released|Runtime|Ge
```

```
nre|Actors|Language|Country|average_score|MUBI_Score|Nominated|Win|Nomination|im
dbVotes|DVD_have|
+----+----------------+---------------+-------------+-----+--------+-------+--
---+------+--------+-------+-------------+----------+---------+---+----------+--
-------+--------+
|   0|               0|              0|            0|    0|    1642|    392|
0|     0|       0|      0|            0|        23|        0|  0|         0|
43|       0|
+----+----------------+---------------+-------------+-----+--------+-------+--
---+------+--------+-------+-------------+----------+---------+---+----------+--
-------+--------+
```

[ ]:
```
from pyspark.sql.types import DateType

pre_analy = pre_analy.withColumn("Runtime", col("Runtime").cast("int"))
pre_analy = pre_analy.withColumn("Year", col("Year").cast("int"))
pre_analy = pre_analy.withColumn("Released", F.to_date(col("Released"),␣
 ↪"d-MMM-yy").cast(DateType()))


pre_analy = pre_analy.withColumn("Quarter", F.quarter(col("Released")))

pre_analy = pre_analy.withColumn("imdbVotes",F.
 ↪regexp_replace(col("imdbVotes"),',','').cast(IntegerType()))
```

[ ]:
```
pre_analy = pre_analy.
 ↪select(['Year','Quarter','MUBI_rating_count','movie_popularity','director_name','Rated','Ru
```

[ ]:
```
# pre_analy.show()

pre_analy = pre_analy.withColumn("average_score", F.
 ↪round(pre_analy["average_score"], 0).cast("integer"))
pre_analy = pre_analy.withColumn("MUBI_Score", F.round(pre_analy["MUBI_Score"],␣
 ↪0).cast("integer"))

pre_analy = pre_analy.dropna()
pre_analy = pre_analy.filter(pre_analy.imdbVotes > 4000)
# pre_analy.show()
pre_analy.count()
# pre_analy.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in pre_analy.
 ↪columns]).show()
```

[ ]:
```
10146
```

[ ]:
```
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml import Pipeline
indexers = [StringIndexer(inputCol=column, outputCol=column+"_index")
            .fit(pre_analy) for column in␣
 ↪["director_name","Rated",'Genre','Actors','Language','Country']]
pipeline = Pipeline(stages=indexers)
convert_df = pipeline.fit(pre_analy).transform(pre_analy)

convert_df = convert_df.
 ↪drop("director_name","Rated",'Genre','Actors','Language','Country')
convert_df = convert_df.drop('director_name_index','Actors_index')
convert_df.dtypes
```

[ ]:
```
[('Year', 'int'),
 ('Quarter', 'int'),
 ('MUBI_rating_count', 'bigint'),
 ('movie_popularity', 'int'),
 ('Runtime', 'int'),
 ('average_score', 'int'),
 ('MUBI_Score', 'int'),
 ('Nominated', 'int'),
 ('Win', 'int'),
 ('Nomination', 'int'),
 ('imdbVotes', 'int'),
 ('DVD_have', 'int'),
 ('Rated_index', 'double'),
 ('Genre_index', 'double'),
 ('Language_index', 'double'),
 ('Country_index', 'double')]
```

[ ]:
```
selected_columns = [col for i, col in enumerate(convert_df.columns[:]) if i not␣
 ↪in [5,6]]
print(selected_columns)

feature = VectorAssembler(inputCols=selected_columns,outputCol="features")
feature_vector= feature.transform(convert_df)
# feature_vector.show(5)
(trainingData, testData) = feature_vector.randomSplit([0.8, 0.2],seed = 21)
```

```
['Year', 'Quarter', 'MUBI_rating_count', 'movie_popularity', 'Runtime',
'Nominated', 'Win', 'Nomination', 'imdbVotes', 'DVD_have', 'Rated_index',
'Genre_index', 'Language_index', 'Country_index']
```

[ ]:
```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="MUBI_Score",␣
 ↪featuresCol="features",maxBins=1000)
```

```
rf_model = rf.fit(trainingData)
rf_prediction = rf_model.transform(testData)
rf_prediction.select("prediction", "MUBI_Score", "features").show(5)

# rf_prediction.write.parquet('predict_result_parquet', mode='overwrite')
```

```
+----------+----------+--------------------+
|prediction|MUBI_Score|            features|
+----------+----------+--------------------+
|       2.0|         2|(14,[0,1,2,4,7,8,…|
|       2.0|         2|[2000.0,1.0,49.0,…|
|       3.0|         3|[2000.0,1.0,108.0…|
|       3.0|         3|[2000.0,1.0,918.0…|
|       3.0|         2|[2000.0,1.0,1182…|
+----------+----------+--------------------+
only showing top 5 rows
```

[ ]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="MUBI_Score",␣
 ↪predictionCol="prediction", metricName="accuracy")
rf_accuracy = evaluator.evaluate(rf_prediction)
print("Accuracy of RandomForestClassifier is = %g"% (rf_accuracy))
print("Test Error of RandomForestClassifier  = %g " % (1.0 - rf_accuracy))
```

```
Accuracy of RandomForestClassifier is = 0.683708
Test Error of RandomForestClassifier  = 0.316292
```

[ ]:
```
importances = rf_model.featureImportances

# Display feature importances
print("Feature Importances:")
for i, imp in enumerate(importances):
    print(f"Feature {selected_columns[i]}: {imp}")
```

```
Feature Importances:
Feature Year: 0.0025430551357666416
Feature Quarter: 0.0001441393101566302
Feature MUBI_rating_count: 0.11338585963458732
Feature movie_popularity: 0.4581810576711831
Feature Runtime: 0.01630695845769419
Feature Nominated: 0.0062915946852650975
Feature Win: 0.09820855518576371
Feature Nomination: 0.07124400136683297
Feature imdbVotes: 0.05728860036687387
```

```
Feature DVD_have: 0.00041536836329448136
Feature Rated_index: 0.04792604765943338
Feature Genre_index: 0.02843910470970159
Feature Language_index: 0.0622623007599792
Feature Country_index: 0.03736335669346782
```

[ ]:
```
evaluator = MulticlassClassificationEvaluator(labelCol="average_score",␣
 ↪predictionCol="prediction", metricName="accuracy")
lr_accuracy = evaluator.evaluate(rf_prediction)
print("Accuracy of LogisticRegression is = %g"% (lr_accuracy))
print("Test Error of LogisticRegression = %g " % (1.0 - lr_accuracy))
```

```
Accuracy of LogisticRegression is = 0.448768
Test Error of LogisticRegression = 0.551232
```

[ ]:
```
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(labelCol="MUBI_Score", featuresCol="features",␣
 ↪maxBins=1000)
dt_model = dt.fit(trainingData)
dt_prediction = dt_model.transform(testData)
dt_prediction.select("prediction", "MUBI_Score", "features").show(5)
```

```
+----------+----------+--------------------+
|prediction|MUBI_Score|            features|
+----------+----------+--------------------+
|       3.0|         1|[2000.0,1.0,1.0,0…|
|       3.0|         3|[2000.0,1.0,95.0,…|
|       2.0|         2|[2000.0,1.0,152.0…|
|       4.0|         4|[2000.0,1.0,208.0…|
|       3.0|         3|[2000.0,1.0,223.0…|
+----------+----------+--------------------+
only showing top 5 rows
```

[ ]:
```
dt_accuracy = evaluator.evaluate(dt_prediction)
print("Accuracy of DecisionTreeClassifier is = %g"% (dt_accuracy))
print("Test Error of DecisionTreeClassifier = %g " % (1.0 - dt_accuracy))
```

```
Accuracy of DecisionTreeClassifier is = 0.434975
Test Error of DecisionTreeClassifier = 0.565025
```