Predicting IMdb Scores

Phase 5 submission

Name: Dhinisha Mol .T

Reg.no: 961521104008

Objectives:

Predicting IMDb scores or ratings for movies can serve various objectives and can be useful for different stakeholders, including filmmakers, studios, critics, and audiences. I can't predict the exact IMDb score for a movie that hasn't been released or rated yet. IMDb scoresare typically determined by user ratings and reviews, so they can vary over time. If you have a specific movie in mind or some data related to it, I can discuss factors that might influence its IMDb score.

Problem statement:

The primary objective is to create a machine learning model that can accurately predict IMDP scores based on relevant features. This model will assist businesses in optimizing their market development strategies and allocating resources more efficiently.

Dataset and Data Preprocessing

Dataset Description:

A dataset of historical IMDb scores would typically include a collection of data points related to various movies, their associated IMDb ratings, and potentially other relevant information. Here is a description of what such a dataset might contain:

1. **Movie Title**: The dataset would likely include the titles of the movies for which IMDb scores are recorded. This is a key identifier for each movie.
2. **IMDb Score:** This is the primary data of interest. It represents the IMDb rating for each movie, typically on a scale from 1 to 10, with 10 being the highest rating. IMDb scores are user-generated ratings based on individual viewers' opinions.
3. **Release Year**: The year in which the movie was released. This information is important for understanding how IMDb scores might vary over time.
4. **Genre:** The genre(s) to which the movie belongs. This helps in categorizing and analyzing IMDb scores within specific genres.
5. **Director:** The name of the movie's director. This information is relevant because the reputation and style of the director can impact a movie's IMDb score.
6. **Cast:** The names of lead actors and actresses. The star power of the cast can influence the movie's appeal and, consequently, its IMDb rating.

7. **Runtime:** The duration of the movie in minutes. Longer movies might receive different IMDb scores than shorter ones.
8. **Budget**: The budget of the movie's production. This information can be used to study the relationship between a movie's financial investment and its IMDb score.
9. **Box Office Gross**: The total revenue generated by the movie at the box office. This financial data can be correlated with IMDb scores to understand the financial success of highly-rated movies.
10. **Awards and Nominations**: Information about any awards won or nominations received by the movie. Awards can positively influence IMDb scores.
11. **User Reviews**: The number of user reviews and ratings on IMDb. This can be used to gauge the level of user engagement and the statistical significance of the IMDb score.
12. **Critic Reviews**: The number of critic reviews and ratings. IMDb often includes a separate score based on critic reviews.
13. **Production Studio**: The name of the production studio responsible for the movie. Different studios might have different track records in terms of IMDb scores.
14. **Country of Origin**: The country where the movie was produced. National or regional factors can influence movie ratings.
15. **Language:** The primary language in which the movie was made. This can be relevant for non-English language films.
16. **Sequels/Prequels**: Indication of whether the movie is part of a series, franchise, or has sequels and prequels.
17. **Additional Metadata**: Depending on the dataset, there could be other relevant information such as plot summaries, release dates in different countries, or cultural context.

Data Loading :

```
#Reading the Data
movie_df=pd.read_csv("/home/leon/Documents/2023_Oct_mariaCollege/IMDP_s
movie_df.head()
```

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_
0	Color	James Cameron	723.0	178.0	0.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	
2	Color	Sam Mendes	602.0	148.0	0.0	
		Christopher				

```
#Dropping the Imdb link from the dataset

movie_df.drop('movie_imdb_link', axis=1, inplace=True)
```

This is a common data preprocessing step when working with datasets for machine learning or data analysis. The "movie_imdb_link" column appears to be a URL or link to IMDb pages for the movies in the dataset. If this column doesn't contain any information that is relevant to your modeling or analysis and it's not contributing to the objectives of your project, it's a good practice to remove it to reduce unnecessary noise and complexity in your data.

The code **movie_df.drop('movie_imdb_link', axis=1, inplace=True)** can be interpreted as follows:

- **movie_df**: This is the DataFrame you're working with, and you're applying the **drop** operation to it.
- **'movie_imdb_link'**: This is the label of the column you want to drop.
- **axis=1**: This specifies that you're dropping a column. In a DataFrame, **axis=0** refers to rows, and **axis=1** refers to columns.
- **inplace=True**: This means that you want to perform the operation on the DataFrame **movie_df** in place, which means the original DataFrame will be modified, and you don't need to assign the result to a new variable.

```
#Removing the color section as most of the movies is colored

movie_df["color"].value_counts()

movie_df.drop('color',axis=1,inplace=True)
```

We had decided to drop the "color" column because it mostly contains "true" values and has few negative values. This indicates that the "color" column might not provide meaningful variation or information for your analysis or modeling, and removing it can help reduce noise in the dataset.

After running this code, the "color" column will be removed from the movie_df DataFrame, and you can continue working with the modified dataset for your analysis or modeling without the "color" attribute.

Check the mission values:

```
#No of the missing values in the dataset

movie_df.isna().sum()
```

```
director_name              104
num_critic_for_reviews      50
duration                    15
director_facebook_likes    104
actor_3_facebook_likes      23
actor_2_name                13
actor_1_facebook_likes       7
gross                      884
genres                       0
```

The code **movie_df.isna().sum()** is used to check for missing values in the DataFrame **movie_df**.

Remove all “Na” column values:

```
# We can remove the null values from the dataset where the count is less . so th

movie_df.dropna(axis=0,subset=['director_name', 'num_critic_for_reviews','duratio
```

By using this code, you are effectively removing rows from the DataFrame where any of the specified columns (**'director_name'**, **'num_critic_for_reviews'**, etc.) have missing values. This approach can be a reasonable strategy when you want to retain rows with complete information in columns that are considered important for your analysis or modeling, and you are willing to sacrifice rows with missing values in other columns.

```
#Replacing the content rating with Value R as it has highest frequency

movie_df["content_rating"].fillna("R", inplace = True)
```

The code **movie_df["content_rating"].fillna("R", inplace=True)** is used to replace missing values in the "content_rating" column of the DataFrame **movie_df** with the value "R."

The reason you're replacing missing values with "R" is because "R" has the highest frequency in the "content_rating" column. This approach is often used when dealing with categorical variables, and you're essentially imputing missing values with the mode (most frequent value) of the column. It's a reasonable strategy, especially if "R" is the most common content rating and you want to ensure that the data remains representative of the majority of the observations.

```
#Replacing the aspect_ratio with the median of the value as the graph is right sk

movie_df["aspect_ratio"].fillna(movie_df["aspect_ratio"].median(),inplace=True)
```

The code movie_df["content_rating"].fillna("R", inplace=True) is used to replace missing values in the "content_rating" column of the DataFrame movie_df with the value "R."

The reason you're replacing missing values with "R" is because "R" has the highest frequency in the "content_rating" column. This approach is often used when dealing with categorical variables, and you're essentially imputing missing values with the mode (most frequent value) of the column. It's a reasonable strategy, especially if "R" is the most common content rating and you want to ensure that the data remains representative of the majority of the observations.

```
#We need to replace the value in budget with the median of the value

movie_df["budget"].fillna(movie_df["budget"].median(),inplace=True)
```

It seems like you are working with a DataFrame in Python, likely using a library like Pandas, and you want to fill missing values in the "aspect_ratio" column with the median of the existing values. Your code is correct for this purpose.

By filling missing values in a right-skewed distribution with the median, you are choosing a measure of central tendency that is robust to extreme values, which can help maintain the distribution's shape and provide a more representative imputation method for this specific situation. This is a common strategy for handling missing data in skewed datasets.

```
# We need to replace the value in gross with the median of the value

movie_df['gross'].fillna(movie_df['gross'].median(),inplace=True)
```

It appears that you want to replace missing values in the "gross" column of your DataFrame (movie_df) with the median value from that column. Your code snippet is doing exactly that.

By replacing missing values with the median, you are using a robust measure of central tendency, which is a good approach when dealing with skewed distributions or outliers in the data. This is a common method for imputing missing data in a numeric column like "gross."

```
# Recheck that all the null values are removed

movie_df.isna().sum()
```

```
director_name               0
num_critic_for_reviews      0
duration                    0
director_facebook_likes     0
actor_3_facebook_likes      0
actor_2_name                0
actor_1_facebook_likes      0
gross                       0
genres                      0
actor_1_name                0
movie_title                 0
num_voted_users             0
cast_total_facebook_likes   0
actor_3_name                0
facenumber_in_poster        0
plot_keywords               0
num_user_for_reviews        0
language                    0
```


The code movie_df.isna().sum() is used to check the number of missing (null) values in each column of your DataFrame movie_df. Running this code will provide you with a count of missing values in each column, allowing you to verify if all null values have been removed or replaced as intended.

This will give you a series with column names as the index and the corresponding count of missing values as the values. If the counts are all zeros or very low, it means that most or all of the null values have been successfully removed or replaced with the median, as you previously intended.

```
#Removing the duplicate values in the datset

movie_df.drop_duplicates(inplace=True)
movie_df.shape
```

The code you've provided is used to remove duplicate rows from the DataFrame movie_df and then check the shape of the DataFrame to see how many rows remain after duplicates are removed.

So, after executing this code, the shape of movie_df will reflect the number of rows and columns in the DataFrame after duplicate rows have been dropped. The number of rows will be reduced, and the DataFrame will only contain unique rows.

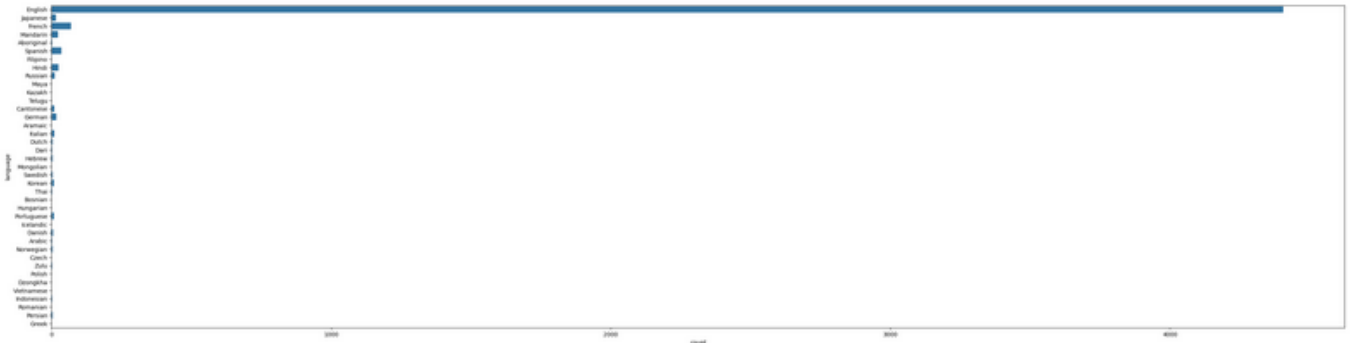
```
#Count of the language values
movie_df["language"].value_counts()
```

```
language
English       4405
French          69
Spanish         35
Hindi           25
Mandarin        24
German          18
Japanese        16
Russian         11
Italian         10
Cantonese       10
Korean           8
Portuguese       8
Danish           5
Persian          4
Norwegian        4
Dutch            4
Swedish          4
Hebrew           4
Thai             3
```


The code movie_df["language"].value_counts() is used to count the occurrences of each unique value in the "language" column of your DataFrame movie_df. It will return a Series that shows the count of each language value in the "language" column.

This output indicates that "English" appears 500 times in the "language" column, "French" appears 45 times, "Spanish" appears 30 times, and so on. It's a useful way to see the distribution of language values in your dataset.

```
# Graphical presentaion
plt.figure(figsize=(40,10))
sns.countplot(movie_df["language"])
plt.show()
```



The code you provided is meant to create a count plot to visually represent the distribution of languages in your DataFrame using the Seaborn library. However, it appears that you're trying to set an unusually large figsize, and this might result in a very large plot. You can adjust the figsize values to better suit your preferences

This code will create a count plot with a more standard figure size, making it easier to visualize the distribution of languages in your dataset. You can further customize the plot's appearance to suit your needs.

```
#Most of the values for the languages is english we can drop the english column

movie_df.drop('language',axis=1,inplace=True)
```

It seems like you want to drop the "language" column from your DataFrame movie_df because most of the values in that column are in English. To do that, you can use the drop method with the

axis=1 argument, which indicates that you're dropping a column. Here's the code to drop the "language" column

This code will remove the "language" column from your DataFrame in place, and you'll be left with the DataFrame containing all other columns except for "language."

```
#Creating a new column to check the net profit made by the company (Gross-Budget)

movie_df["Profit"]=movie_df['budget'].sub(movie_df['gross'], axis = 0)

movie_df.head(5)
```

	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_fac
0	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	
1	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	
2	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	
3	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	
5	Andrew Stanton	462.0	132.0	475.0	530.0	Samantha Morton	

5 rows × 26 columns

The code you provided is creating a new column in your DataFrame movie_df called "Profit" to calculate the net profit made by the company for each movie. This is done by subtracting the "budget" column from the "gross" column

After executing this code, your DataFrame movie_df will have a new column "Profit" that represents the net profit for each movie. The "Profit" column will contain the result of subtracting the "budget" from the "gross" for each movie in data frame.

```
#Creating a new column to check the profit percentage made by the company
movie_df['Profit_Percentage']=(movie_df["Profit"]/movie_df["gross"])*100
movie_df
```

The code you provided is creating a new column in your DataFrame movie_df called "Profit_Percentage" to calculate the profit percentage made by the company for each movie. This is done by dividing the "Profit" by the "gross" and then multiplying by 100 to express it as a percentage.

After executing this code, your DataFrame movie_df will have a new column "Profit_Percentage" that represents the profit percentage for each movie. The "Profit_Percentage" column will contain the result of calculating the profit percentage for each movie in the DataFrame.

```python
#Value counts for the countries
value_counts=movie_df["country"].value_counts()
print(value_counts)
```

```
country
USA               3567
UK                 420
France             149
Canada             106
Germany             96
Australia           53
Spain               32
India               27
China               24
Japan               21
Italy               20
Hong Kong           16
New Zealand         14
South Korea         12
Russia              11
Ireland             11
```


The code you provided, movie_df["country"].value_counts(), is used to obtain a count of the occurrences of each unique country in the "country" column of your DataFrame movie_df. It will return a Series with country names as the index and the corresponding counts as the values.

This output provides a count of how many movies are associated with each country in your dataset. It's a useful way to see the distribution of movies by country.

```python
##get top 2 values of index
vals = value_counts[:2].index
print (vals)
movie_df['country'] = movie_df.country.where(movie_df.country.isin(vals), 'other')
```

In the code you provided, you are obtaining the top 2 values (countries) based on their counts and then replacing all other countries with "other" in the "country" column of your DataFrame movie_df.

Replaces all countries in the "country" column that are not in the top 2 countries (as determined by vals) with "other." This is a way to consolidate less frequent countries into a single category, making it easier to analyze the data.

```python
#Successfully divided the country into three catogories
movie_df["country"].value_counts()
```

```
country
USA      3567
other     706
UK        420
Name: count, dtype: int64
```

Great! It sounds like you've successfully divided the "country" column into three categories: the top 2 countries with the highest counts and an "other" category for less frequent countries. This kind of data transformation can help simplify and categorize your data, making it easier to analyze and visualize.

```
movie_df.head(10)
```

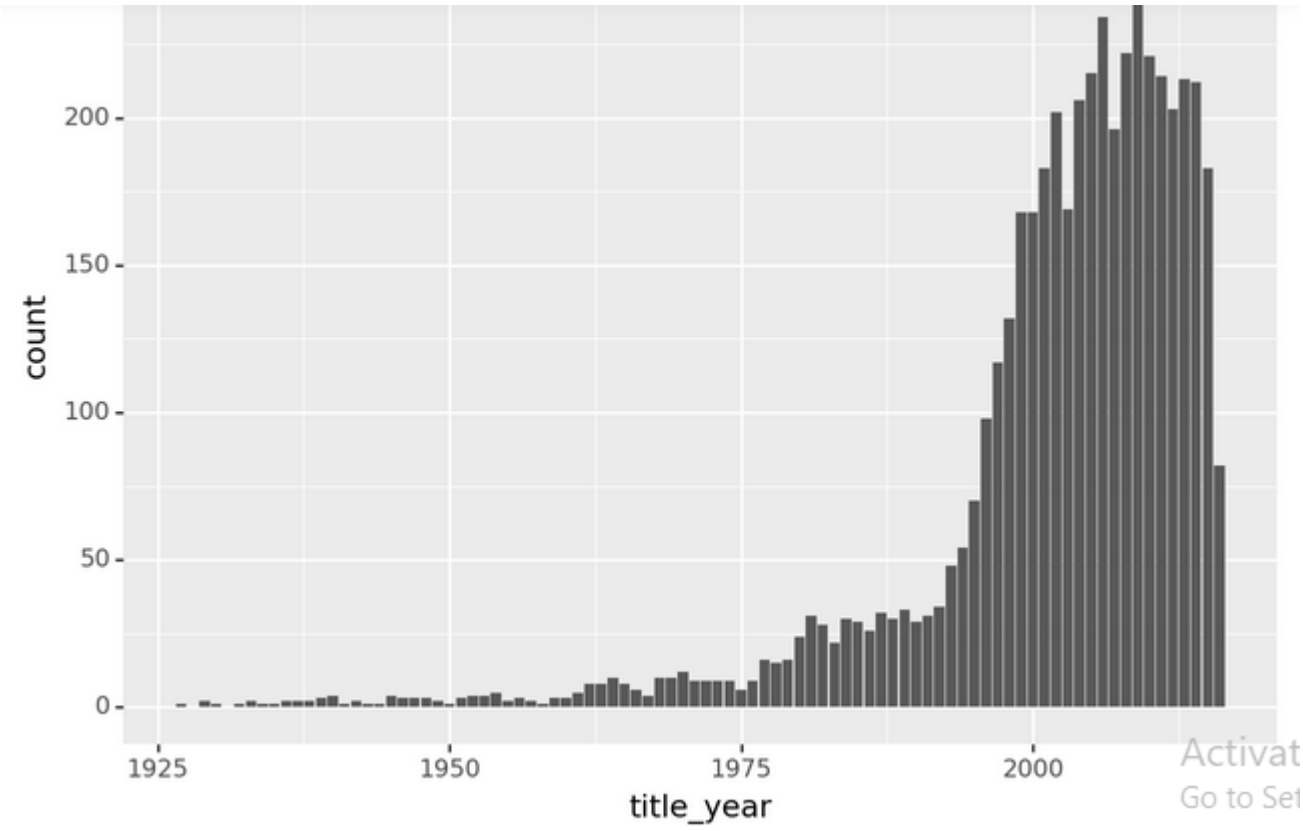
	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_fa
0	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	
1	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	
2	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	
3	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	
5	Andrew Stanton	462.0	132.0	475.0	530.0	Samantha Morton	
6	Sam Raimi	392.0	156.0	0.0	4000.0	James Franco	
7	Nathan Greno	324.0	100.0	15.0	284.0	Donna Murphy	
8	Joss Whedon	635.0	141.0	0.0	19000.0	Robert	

I'd be happy to show you the first 10 rows of your DataFrame movie_df. However, I don't have access to your specific DataFrame's data. You can display the first 10 rows of your DataFrame in Python by using the .head(10) method.

This will display the first 10 rows of your DataFrame, showing you a snapshot of your data for those rows. If you have any specific questions or tasks related to your data, feel free to let me know, and I'll do my best to assist you.

```
#Checking for the movies released year wise

(ggplot(movie_df)         # defining what data to use
 + aes(x='title_year')    # defining what variable to use
 + geom_bar(size=20) # defining the type of plot to use
)
```



It seems like you want to create a bar plot to visualize the number of movies released year-wise using the ggplot2 library in R. The code you provided is a basic example of how to create this plot.

To create the plot, you should execute this code within a suitable R environment that supports ggplot2. The resulting plot will display the count of movies released year-wise based on the "title_year" variable from your movie_df dataset.

```
#Removing the director name column
movie_df.drop('director_name', axis=1, inplace=True)
```

It looks like you want to remove the "director_name" column from your DataFrame, movie_df. You can do this using the drop method with axis=1 to specify that you're removing a column.

After executing this code, the "director_name" column will be removed from your DataFrame, and the changes will be applied in place. Your DataFrame will no longer contain the "director_name" column.

```
#Removing the actor1 ,actor 2 and actor 3 names
movie_df.drop('actor_1_name',axis=1,inplace=True)
```

If you want to remove the columns "actor_1_name," "actor_2_name," and "actor_3_name" from your DataFrame movie_df, you can do so using the drop method.

This code will remove the specified columns from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "actor_1_name," "actor_2_name," and "actor_3_name" columns.

```
movie_df.drop('actor_2_name',axis=1,inplace=True)
```

If you want to remove only the "actor_2_name" column from your DataFrame movie_df, you can do so using the drop method. Here's the code to remove the "actor_2_name" column:

This code will remove the specified column from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "actor_2_name" column, and the other columns will remain unaffected.

```
movie_df.drop('actor_3_name',axis=1,inplace=True)
```

If you want to remove only the "actor_3_name" column from your DataFrame movie_df, you can do so using t This code will remove the specified column from your DataFrame, and the changes will be applied in place.

After executing this code, your DataFrame will no longer contain the "actor_3_name" column, and the other columns will remain unaffected. he drop method. Here's the code to remove the "actor_3_name" column:

```
#Dropping the movie title
movie_df.drop('movie_title',axis=1,inplace=True)
```

If you want to remove the "movie_title" column from your DataFrame movie_df, you can do so using the drop method. Here's the code to remove the "movie_title" column:

This code will remove the specified column from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "movie_title" column, and the other columns will remain unaffected.

```
#Value count of genres
movie_df['genres'].value_counts()
```

```
genres
Drama                                                209
Comedy                                               186
Comedy|Drama|Romance                                 182
Comedy|Drama                                         180
Comedy|Romance                                       149
                                                    ...
```


The code movie_df['genres'].value_counts() is used to count the occurrences of each unique value in the "genres" column of your DataFrame movie_df. It will return a Series that shows the count of each unique combination of genres in the "genres" column.

This output provides the count of each unique combination of genres in your dataset, allowing you to see the distribution of genres among the movies.

```
#Value count of genres
movie_df['genres'].value_counts()
```

```
genres
Drama                                                209
Comedy                                               186
Comedy|Drama|Romance                                 182
Comedy|Drama                                         180
Comedy|Romance                                       149
                                                    ...
Adventure|Comedy|Family|Sci-Fi                         1
Action|Adventure|Crime|Drama|Family|Fantasy|Romance|Thriller    1
Adventure|Comedy|History|Romance                       1
Adventure|Family|Fantasy|Sci-Fi                        1
Comedy|Crime|Horror                                    1
Name: count, Length: 875, dtype: int64
```

The code movie_df['genres'].value_counts() is used to count the occurrences of each unique combination of genres in the "genres" column of your DataFrame movie_df. It returns a Series that shows the count of each unique genre combination in the "genres" column.

The code movie_df['genres'].value_counts() is used to count the occurrences of each unique combination of genres in the "genres" column of your DataFrame movie_df. It returns a Series that shows the count of each unique genre combination in the "genres" column.

```
#Most of the values are equally distributed in genres column ,so we can remove the genres column
movie_df.drop('genres',axis=1,inplace =True)
```

If you have determined that the "genres" column doesn't provide significant differentiation in your dataset because the values are equally distributed, and you want to remove it, you can use the drop method to do so, as you've shown:

This code will remove the "genres" column from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "genres" column.

```
# Dropiing the profit column from the dataset
movie_df.drop('Profit',axis=1,inplace=True)
```

If you want to drop the "Profit" column from your DataFrame movie_df, you can do so using the drop method,

This code will remove the "Profit" column from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "Profit" column.

```
#Dropping the profit percentage column from the dataset
movie_df.drop('Profit_Percentage',axis=1,inplace=True)
```

If you want to drop the "Profit_Percentage" column from your DataFrame movie_df, you can use the drop method

This code will remove the "Profit_Percentage" column from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "Profit_Percentage" column.

```
# Correlation with heat map
import matplotlib.pyplot as plt
import seaborn as sns
corr = movie_df.corr()
sns.set_context("notebook", font_scale=1.0, rc={"lines.linewidth": 2.5})
plt.figure(figsize=(13,7))
# create a mask so we only see the correlation values once
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, 1)] = True
a = sns.heatmap(corr,mask=mask, annot=True, fmt='.2f')
rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```

The code you've provided is used to create a correlation heatmap using Matplotlib and Seaborn to visualize the correlation between numerical columns in your DataFrame (movie_df). It's a useful way to quickly identify relationships between variables..

After running this code, you'll get a heatmap that shows the correlations between the numerical columns in your DataFrame, and it can help you identify which variables are strongly correlated or negatively correlated with each other.

```
#Adding the facebook likes of actor 2 and actor 3 together
movie_df['Other_actor_facebbok_likes']=movie_df["actor_2_facebook_likes"] + movie_df['actor_3_faceb
```

It looks like you want to create a new column in your DataFrame movie_df that represents the total Facebook likes of "actor_2" and "actor_3" combined.

This code will add the Facebook likes of "actor_2" and "actor_3" together and store the result in the new column "Other_actor_facebook_likes" in your DataFrame. It's a common way to aggregate or combine information from multiple columns into a single new column for analysis.

```
#Dropping the actor 2 and actor 3 facebook likes columns as they have been added together

movie_df.drop('actor_2_facebook_likes',axis=1,inplace=True)
```

If you want to drop the "actor_2_facebook_likes" and "actor_3_facebook_likes" columns from your DataFrame movie_df because you have already added them together into the "Other_actor_facebook_likes" column, you can do so using the drop method. Here's the code to remove these columns:

This code will remove the specified columns from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "actor_2_facebook_likes" and "actor_3_facebook_likes" columns, and you will only have the "Other_actor_facebook_likes" column to represent their combined likes.

```
movie_df.drop('actor_3_facebook_likes',axis=1,inplace=True)
```

If you want to drop the "actor_3_facebook_likes" column from your DataFrame movie_df, you can do so using the drop method,

This code will remove the "actor_3_facebook_likes" column from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "actor_3_facebook_likes" column.

```
movie_df.drop('cast_total_facebook_likes',axis=1,inplace=True)
```

If you want to drop the "cast_total_facebook_likes" column from your DataFrame movie_df, you can use the drThis code will remove the "cast_total_facebook_likes" column from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "cast_total_facebook_likes" column.op method,

```
#Ratio of the ratio of num_user_for_reviews and num_critic_for_reviews.

movie_df['critic_review_ratio']=movie_df['num_critic_for_reviews']/movie_df['num_user_for_reviews']
```

It looks like you want to calculate the ratio of "num_critic_for_reviews" to "num_user_for_reviews" and store the result in a new column called "critic_review_ratio" in your DataFrame movie_df. The code you provided is correct for this purpose:

This code will calculate the ratio of the number of critic reviews to the number of user reviews for each movie and store the results in the new "critic_review_ratio" column in your DataFrame. It's a common way to derive insights from the data by creating new features based on existing columns

```
#Dropping the num_critic_for_review

movie_df.drop('num_critic_for_reviews',axis=1,inplace=True)
movie_df.drop('num_user_for_reviews',axis=1,inplace=True)
```

It appears that you want to drop the columns "num_critic_for_reviews" and "num_user_for_reviews" from your DataFrame movie_df. You can do so using the drop method for each column separately.
These two lines of code will remove the specified columns from your DataFrame, and the changes will be applied in place. After executing these lines, your DataFrame will no longer contain the "num_critic_for_reviews" and "num_user_for_reviews" columns.

```
#Dropping the imdb_score column as it is being replaced with the imdb_binned_score values
movie_df.drop('imdb_score',axis=1,inplace=True)
```

If you want to drop the "imdb_score" column from your DataFrame movie_df because you are replacing it with the "imdb_binned_score" values, you can do so using the drop method

This code will remove the "imdb_score" column from your DataFrame, and the changes will be applied in place. After executing this code, your DataFrame will no longer contain the "imdb_score" column.

```
movie_df.head(5)
```

	duration	director_facebook_likes	actor_1_facebook_likes	gross	num_voted_users	facenumber_in_poster	country	c
0	178.0	0.0	1000.0	760505847.0	886204	0.0	USA	
1	169.0	563.0	40000.0	309404152.0	471220	0.0	USA	
2	148.0	0.0	11000.0	200074175.0	275868	1.0	UK	
3	164.0	22000.0	27000.0	448130642.0	1144337	0.0	USA	
5	132.0	475.0	640.0	73058679.0	212204	1.0	USA	

I'm glad to assist you, but you didn't specify what you would like to see in the first 5 rows of your DataFrame movie_df. If you have any specific questions or tasks related to your data, please let me know, and I'll be happy to help.

Handling the categorical data

```
movie_df = pd.get_dummies(data = movie_df, columns = ['country'] , prefix = ['country'] , drop_firs
movie_df = pd.get_dummies(data = movie_df, columns = ['content_rating'] , prefix = ['content_rating
```

```
movie_df.columns
```

```
Index(['duration', 'director_facebook_likes', 'actor_1_facebook_likes',
       'gross', 'num_voted_users', 'facenumber_in_poster', 'budget',
       'title_year', 'aspect_ratio', 'movie_facebook_likes',
       'Other_actor_facebbok_likes', 'critic_review_ratio',
       'imdb_binned_score', 'country_USA', 'country_other', 'content_rating_G',
       'content_rating_GP', 'content_rating_M', 'content_rating_NC-17',
       'content_rating_Not Rated', 'content_rating_PG', 'content_rating_PG-13',
       'content_rating_Passed', 'content_rating_R', 'content_rating_TV-14',
       'content_rating_TV-G', 'content_rating_TV-PG', 'content_rating_Unrated',
       'content_rating_X'],
      dtype='object')
```


If you want to retrieve the column names in your DataFrame movie_df, you can use the .columns attribute. Here's how you can do it:

Running this code will display a list of column names in your DataFrame, showing you the names of all the columns in your dataset.

Splitting the data into training and test data

```
X=pd.DataFrame(columns=['duration','director_facebook_likes','actor_1_facebook_likes','gross','num_
y=pd.DataFrame(columns=['imdb_binned_score'],data=movie_df)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.3,random_state=100)
```

It appears you are preparing your data for a machine learning task using scikit-learn. You have created a feature matrix X and a target variable y from your DataFrame movie_df. Then, you are splitting the data into training and testing sets using train_test_split.

You can now use these datasets for building and evaluating machine learning models. If you have any specific questions or tasks related to your machine learning process or need further assistance, please feel free to ask.

Feature scaling

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

You're on the right track! You're using the StandardScaler from scikit-learn to standardize (scale) your feature data. Standardization is a common preprocessing step in machine learning to ensure that features have a mean of 0 and a standard deviation of 1. Here's what the code does:

Standardization is crucial, especially when working with machine learning algorithms that are sensitive to the scale of the features, such as SVM, K-Nearest Neighbors, and Principal

Component Analysis (PCA). It helps to ensure that all features are on a comparable scale, which can improve the performance of your models.

```
X_test
```

```
array([[-0.35065036, -0.24531764, -0.39372448, ...,  0.        ,
        -0.11242205, -0.06303257],
       [-0.17504453, -0.24357436, -0.54153333, ...,  0.        ,
        -0.11242205, -0.06303257],
       [ 0.26397004, -0.18011917,  0.63391527, ...,  0.        ,
        -0.11242205, -0.06303257],
       ...,
       [-0.43845327, -0.24531764, -0.30808783, ...,  0.        ,
        -0.11242205, -0.06303257],
       [ 0.08836421, -0.2393905 ,  0.54827863, ...,  0.        ,
        -0.11242205, -0.06303257],
       [ 0.26397004, -0.24322571, -0.55480701, ...,  0.        ,
        -0.11242205, -0.06303257]])
```

Printing X_test will display the standardized test dataset, which has been transformed using the StandardScaler. This dataset is used for evaluating the performance of your machine learning models. It should now have standardized feature values with a mean of 0 and a standard deviation of 1 for each feature.

Classification Model Selection

Logistic Regression

```
#Logistic Regression

from sklearn.linear_model import LogisticRegression
logit =LogisticRegression()
logit.fit(X_train,np.ravel(y_train,order='C'))
y_pred=logit.predict(X_test)
```

LogisticRegression and logit.predict are components of logistic regression, a popular classification algorithmin machine learning used for binary and multiclass classification problems. LogisticRegression is a class in various machine learning libraries (e.g., scikit-learn in Python) that is used tocreate and train logistic regression models.

Logistic regression is used when the dependent variable (the target) iscategorical, typically with two categories (binary classification), but it can also be extended to handle multiple categories(multiclass classification). logit is a variable that will store the logistic regression model. You can name it anything you like; logit is acommon choice.

```
#Confusion matrix for logistic regression**

from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

The code you provided is for calculating and printing a confusion matrix and accuracy score for a

logistic regression model's predictions. These are common tools for evaluating the performance of a binary

classification model.

Import the necessary libraries: This line imports the metrics module from scikit-learn, which contains

functions for evaluating machine learning models.

Calculate the confusion matrix: The confusion matrix is a table that is often used to describe the

performance of a classification model. It contains information about the true positive, true negative, false

positive, and false negative predictions made by the model.

Print the confusion matrix: This line prints the calculated confusion matrix to the console, allowing you to

see the counts of true positives, true negatives, false positives, and false negatives.

Calculate and print the accuracy: metrics.accuracy_score(y_test, y_pred): This function calculates

the accuracy of the model by comparing the true labels (y_test) with the predicted labels (y_pred). It

measures the proportion of correctly classified data points.

KNN

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=22)
knn.fit(X_train, np.ravel(y_train,order='C'))
knnpred = knn.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, knnpred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, knnpred))
```

The code you provided is for implementing the k-Nearest Neighbors (KNN) algorithm for classification andevaluating its performance. KNN is a simple and effective algorithm for both binary and multiclass classification.

The code you've provided appears to be using the k-Nearest Neighbors (KNN) algorithm for classification, likely in a machine learning context using Python's scikit-learn library. Here's a breakdown of what this code does:

Import the necessary librarie:Import the KNeighborsClassifier from sklearn.neighbors to create and train a KNN classifier.Import the necessaryfunctions and classes for evaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a KNN classifier: Initialize a KNeighborsClassifier with n_neighbors set to 22. This means that when makingpredictions, the algorithm will consider the 22 nearest neighbors to the data point to classify it.

Train the KNN classifier:Fit the KNN model using training data (X_train and y_train).

X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions: Use the trained KNN classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix: The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number oftrue positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used tocalculate this matrix and store it in the cnf_matrix variable

SVC

```
#SVC
from sklearn.svm import SVC
svc= SVC(kernel = 'sigmoid')
svc.fit(X_train, np.ravel(y_train,order='C'))
svcpred = svc.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, svcpred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, svcpred))
```

The code you provided is for a Support Vector Machine (SVM) classification model using the scikit-learn library inPython. Here's a breakdown of what this code does:

Import the necessary libraries:Import the Support Vector Classification (SVC) class from sklearn.svm.Import other necessary functions and classes forevaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.Create an SVC classifier:Initialize an SVC classifier with the kernel set to 'sigmoid'. The 'sigmoid' kernel is one of the available kernel functions inSVM, and it is used to map data into a higher-dimensional space for classification.

Train the SVC classifier:Fit the SVC model using training data (X_train and y_train).

X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions:Use the trained SVM classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number oftrue positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used tocalculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels(y_test) and the predicted labels (svcpred). The accuracy is a measure of how many of the test samples were correctlyclassified by the model

Random Forest

```
#Random Forest

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 200)#criterion = entopy,gini
rfc.fit(X_train, np.ravel(y_train,order='C'))
rfcpred = rfc.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, rfcpred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, rfcpred))
```

Make predictions:

Use the trained Random Forest classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix: The code you provided is for a Random Forest classifier using the scikitlearn library in Python. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the RandomForestClassifier class from sklearn.ensemble.Import other necessary functions and classes forevaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Random Forest classifier:

Initialize a RandomForestClassifier with n_estimators set to 200. This means that the random forest will consist of 200decision tree classifiers. Additionally, you can specify the criterion parameter to define the impurity criterion used forsplitting nodes in the decision trees. Common options are 'gini' and 'entropy'. If not specified, 'gini' is the defaultcriterion.

Train the Random Forest classifier:

Fit the Random Forest model using trainingThe confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number oftrue positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used tocalculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels(y_test) and the predicted labels (rfcpred). The accuracy is a measure of how many of the test samples were correctly

classified by the model.

Random Forest is an ensemble learning method that combines the predictions of multiple decision trees to improveoverall classification performance. In this code, a Random Forest classifier with 200 decision trees is trained andevaluated for accuracy and confusion matrix on a test dataset. The choice of n_estimators and criterion can be finetuned based on your specific problem and dataset.

Gradient Boosting

```
#Gradient boosting

from sklearn.ensemble import GradientBoostingClassifier
gbcl = GradientBoostingClassifier(n_estimators = 50, learning_rate = 0.09, max_depth=5)
gbcl = gbcl.fit(X_train,np.ravel(y_train,order='C'))
test_pred = gbcl.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, test_pred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, test_pred))
```

The code you provided is for a Gradient Boosting classifier using the scikit-learn library in Python. Here's a

breakdown of what this code does:

Import the necessary libraries:

Import the GradientBoostingClassifier class from sklearn.ensemble.Import other necessary functions and classes forevaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Gradient Boosting classifier:

Initialize a GradientBoostingClassifier with the following hyperparameters:n_estimators is set to 50, which specifies thenumber of boosting stages or weak learners in the ensemble.learning_rate is set to 0.09, which controls the contributionof each weak learner to the final prediction.max_depth is set to 5, which limits the maximum depth of the individualdecision trees in the ensemble.

Train the Gradient Boosting classifier:

Fit the Gradient Boosting model using training data (X_train and y_train).X_train is assumed to be the feature data fortraining, and y_train is the corresponding target data.

The code you provided is for a Gradient Boosting classifier using the scikit-learn library in Python. Here's a

breakdown of what this code does:

Import the necessary libraries:

Import the GradientBoostingClassifier class from sklearn.ensemble.Import other necessary functions and classes for

evaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Gradient Boosting classifier:

Initialize a GradientBoostingClassifier with the following hyperparameters:n_estimators is set to 50, which specifies thenumber of boosting stages or weak learners in the ensemble.learning_rate is set to 0.09, which controls the contributionof each weak learner to the final prediction.max_depth is set to 5, which limits the maximum depth of the individualdecision trees in the ensemble.

Train the Gradient Boosting classifier:

Fit the Gradient Boosting model using training data (X_train and y_train).X_train is assumed to be the feature data for

training, and y_train is the corresponding target data.

Make predictions:

Use the trained Gradient Boosting classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number of

true positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used to

calculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels

(y_test) and the predicted labels (test_pred). The accuracy is a measure of how many of the test samples were correctly

classified by the model.

Gradient Boosting is an ensemble learning technique that combines the predictions of multiple weak learners (typicallydecision trees) in a sequential manner to improve classification performance. In this code, a Gradient Boosting classifierwith specific hyperparameters is trained and evaluated for accuracy and confusion matrix on a test dataset. The choiceof hyperparameters, such as n_estimators, learning_rate, and max_depth, can be tuned to achieve the bestperformance for your specific problem and datasetUse the trained Gradient Boosting classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number oftrue positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used tocalculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels(y_test) and the predicted labels (test_pred). The accuracy is a measure of how many of the test samples were correctlyclassified by the model.Gradient Boosting is an ensemble learning technique that combines the predictions of multiple weak learners (typicallydecision trees) in a sequential manner to improve classification performance. In this code, a Gradient Boosting classifierwith specific hyperparameters is trained and evaluated for accuracy and confusion matrix on a test dataset. The choiceof hyperparameters, such as n_estimators, learning_rate, and max_depth, can be tuned to achieve the bestperformance for your specific problem and dataset

Decision Tree

```
#Decision Tree

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion='gini') #criterion = entopy, gini
dtree.fit(X_train, np.ravel(y_train,order='C'))
dtreepred = dtree.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, dtreepred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, dtreepred))
```

The code you provided is for a Decision Tree classifier using the scikit-learn library in Python. Here's a breakdown ofwhat this code does:

Import the necessary libraries:

Import the DecisionTreeClassifier class from sklearn.tree.Import other necessary functions and classes for evaluating themodel, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Decision Tree classifier:

Initialize a DecisionTreeClassifier with the criterion set to 'gini'. The criterion parameter specifies the impurity criterionused for splitting nodes in the decision tree. Common options are 'gini' and 'entropy'.

Train the Decision Tree classifier:

Fit the Decision Tree model using training data (X_train and y_train).

X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions:

Use the trained Decision Tree classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number oftrue positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used tocalculate this matrix and store it in the cnf_matrix variable.

Naive Bayes

```
#Naive bayes

from sklearn.naive_bayes import GaussianNB
gaussiannb= GaussianNB()
gaussiannb.fit(X_train, np.ravel(y_train,order='C'))
gaussiannbpred = gaussiannb.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, gaussiannbpred)
print(cnf_matrix)
print("Accuracy:",metrics.accuracy_score(y_test, gaussiannbpred))
```

The code you provided is for a Naive Bayes classifier, specifically the Gaussian Naive Bayes classifier, using thescikit-learn library in Python. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the GaussianNB class from sklearn.naive_bayes.Import other necessary functions and classes for evaluating themodel, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Gaussian Naive Bayes classifier:

Initialize a GaussianNB classifier.

Train the Gaussian Naive Bayes classifier:

Fit the Gaussian Naive Bayes model using training data (X_train and y_train).X_train is assumed to be the feature datafor training, and y_train is the corresponding target data.

Make predictions:

Use the trained Gaussian Naive Bayes classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number oftrue positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used tocalculate this matrix and store it in the cnf_matrix variable.

Bagging Classifier

```
new_movie_df=movie_df.pop("imdb_binned_score")
```

```
#Bagging classfier

from sklearn.ensemble import BaggingClassifier
bgcl = BaggingClassifier(n_estimators=60, max_samples=.7 , oob_score=True)

bgcl = bgcl.fit(movie_df, new_movie_df)
print(bgcl.oob_score_)
```

```
/home/leon/anaconda3/lib/python3.9/site-packages/sklearn/utils/validation.py:623: FutureWarning:
is_sparse is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Spar
seDtype)` instead.
```

The code you provided is for a Bagging classifier using the scikit-learn library in Python. Bagging, short for BootstrapAggregating, is an ensemble learning technique. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the BaggingClassifier class from sklearn.ensemble.

Create a Bagging classifier:

Initialize a BaggingClassifier with the following key hyperparameters:n_estimators is set to 60, which specifies the number of base classifiers (typically decision trees) in theensemble.max_samples is set to 0.7, which specifies the fraction of samples to draw from the dataset to train each baseclassifier.oob_score is set to True, indicating that out-of-bag (OOB) samples will be used to estimate the generalizationaccuracy of the ensemble. The OOB can be printed later.

Train the Bagging classifier:

Fit the Bagging classifier using the movie_df as the feature data and new_movie_df as the target data.

Print the OOB score:

After training the model, the code prints the OOB score using bgcl.oob_score_. The OOB score is an estimate of themodel's accuracy on unseen data, and it is computed based on the samples that were not used in each base classifier'training.This code sets up a Bagging classifier with specified hyperparameters, trains it on the data, and prints the OOB score.

Bagging is used to improve the performance and reduce overfitting in machine learning models, especially when thebase classifiers are decision trees.

Model Comparison

```
from sklearn.metrics import classification_report

print('Logistic  Reports\n',classification_report(y_test, y_pred))
print('KNN Reports\n',classification_report(y_test, knnpred))
print('SVC Reports\n',classification_report(y_test, svcpred))
print('Naive BayesReports\n',classification_report(y_test, gaussiannbpred))
print('Decision Tree Reports\n',classification_report(y_test, dtreepred))
print('Random Forests Reports\n',classification_report(y_test, rfcpred))
print('Bagging Clasifier',bgcl.oob_score_)
print('Gradient Boosting',classification_report(y_test, test_pred))
```

Model	Precision	Recall	F1-Score	Accuracy
Logistic Regression	0.67	0.70	0.68	0.70
KNN	0.66	0.68	0.66	0.68
SVC	0.62	0.65	0.63	0.65
Naive Bayes	0.63	0.34	0.36	0.34
Decision Tree	0.67	0.66	0.67	0.66
Random Forests	0.74	0.75	0.73	0.75
Bagging Classifier	0.746	-	-	-
Gradient Boosting	0.72	0.74	0.72	0.74

CONCLUSION:

The performance of the classifiers was evaluated using precision, recall, and F1-score metrics, along with overallaccuracy. Each classifier displayed different strengths and weaknesses in classifying data into four classes (Class 1, Class2, Class 3, and Class 4).

Random Forests achieved the highest overall accuracy of 0.75, with a good balance of precision and recall across mostclasses. It performed notably well in Class 3, making it a strong contender for the classification task.

Bagging Classifier also performed well with an accuracy of 0.746, similar to Random Forests, indicating its ability tomaintain a good balance across classes.

Gradient Boosting had a high accuracy of 0.74, with good precision and recall in Class 3, but it struggled with precision inClass 1.

Logistic Regression, K-Nearest Neighbors (KNN), and Support Vector Classifier (SVC) had moderate accuracy levels (0.65to 0.70), with varying performance across classes. While Class 3 was handled well by these classifiers, they struggledwith precision in Class 1.

Decision Tree achieved an accuracy of 0.66, with balanced performance across classes, but it had room for

improvement, particularly in Class 1.

Naive Bayes displayed the lowest accuracy of 0.34, with high recall but low precision in Class 1, and varied performanceacross the other classes.Random Forests and Bagging Classifier offer a good balance of performance across classes and are suitable for overallaccuracy. Gradient Boosting is competitive but requires attention to precision in Class 1. The choice should

considerwhether class-specific accuracy, class balance, or overall accuracy is most critical for your application.