Predicting IMdb Scores

Phase3 document submission

Name:Dhinisha Mol .T

Reg.no:961521104008

OBJECTIVES:

I can't predict the exact IMDb score for a movie that hasn't been released or rated yet. IMDb scores are 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.

```
Import pandas as pd

matplotlib.pyplot as plt

Import seaborn as sns

Import plotly.express as px
```

Certainly! It looks like you're starting to use some popular Python libraries for data analysis and visualization.

Pandas is a powerful library for data manipulation and analysis.It provides data structures like DataFrames to work with structured data efficiently.matplotlib is a widely used data visualization library in Python.pyplot is a module within matplotlib that provides a simple and high-level interface for creating various types of plots and charts.

```
Df = pd.read_csv("/kaggle/input/netflix-original-films-imdb-scores/NetflixOriginals.csv",encoding = "ISO-8859-1")

Df
```

This line of code uses the pd.read_csv() function from the pandas library to read data from a CSV file.

The file path "/kaggle/input/netflix-original-films-imdb-scores/NetflixOriginals.csv" specifies the location of the CSV file you want to read.

This line assigns the data read from the CSV file to a DataFrame called df.By loading the data into a DataFrame, you can now perform various data analysis and manipulation tasks on it, such as filtering, grouping, visualizing, and more. If you have specific questions or tasks you'd like to perform with this dataset, please let me know, and I'll be happy to assist You further.

```
Df.describe()
```

When you use the df.describe() function on a DataFrame in pandas, it provides a summary of the basic statistics for each numerical column in the DataFrame. These statistics include:

Count: The number of non-null values in the column.

Mean: The average value of the column.

Std: The standard deviation, which measures the dispersion or spread of the data.

Min: The minimum value in the column.

Max: The maximum value in the column.

By using df.describe(), you can quickly get an overview of the central tendency and spread of your numerical data.

Df.isnull().sum()

The df.isnull().sum() command is used to identify and count the number of missing (null or NaN) values in each column of a DataFrame (df). Here's what each part of the command does:

Df: This is the DataFrame containing your data.

isnull(): This is a DataFrame method that returns a DataFrame of the same shape as the original df. It contains Boolean values (True or False) that indicate whether each element in the original DataFrame is null (True) or not (False).

sum(): This function is applied to the Boolean DataFrame created by

The output of df.isnull().sum() is a Series that lists the number of missing values in each column of your DataFrame. This information is valuable for data cleaning and preparation because it helps you identify which columns have missing data and to what extent

```
Df['Premiere'] = pd.to_datetime(df['Premiere'])

#Columns year, month and weekday

Df['year']    = df['Premiere'].dt.year

Df['month']   = df['Premiere'].dt.month_name()

Df['weekday'] = df['Premiere'].dt.day_name()
```

Df.head()

The code you provided is related to working with date and time data within a pandas DataFrame. It appears to be performing the following operations:

Df['Premiere'] = pd.to_datetime(df['Premiere']):

This line converts the 'Premiere' column in the DataFrame df into a datetime data type using the pd.to_datetime() function.

By adding these new columns, you can analyze your data with more granularity, such as examining trends by year, month, or day of the week. This can be particularly useful when working with time series data or any dataset that includes date and time information.

```
Df_temp=df.groupby(['Runtime','Title','Language']).mean().sort_values(by='Runtime', ascending=False).reset_index().iloc[:,:3]
```

The code you provided involves several operations on a DataFrame (df) and appears to be summarizing and reorganizing the data. Df_temp = df.groupby(['Runtime', 'Title', 'Language']).mean():

This line groups the original DataFrame df by the columns 'Runtime', 'Title', and 'Language'.So, the overall result is a DataFrame df_temp that contains data grouped by 'Runtime', 'Title', and 'Language', where the mean values of the numeric columns are calculated for each group. This data is then sorted based on the 'Runtime' column in descending order, and the index is reset to a default integer index. The final DataFrame contains only the first three columns. This can be useful for exploring and summarizing the data based on these specific columns.

```
Fig = px.box(df, x= 'Runtime', hover_data = df[['Title','Language']])

Fig.update_traces(quartilemethod="inclusive")

Fig.show()
```

This line creates a box plot using Plotly Express (px.box) with the data from the DataFrame df.The x argument specifies that the 'Runtime' column from your DataFrame will be represented on the x-axis

The quartilemethod="inclusive" argument sets the quartile calculation method to "inclusive." This means that the whiskers of the box plot will extend to the minimum and maximum values within 1.5 times the interquartile range, which might include potential outliers.Finally, this command displays the box plot that you've created.

This kind of visualization is useful for understanding the distribution of a numerical variable (in this case, 'Runtime') and for identifying any movies with runtimes that are significantly different from the rest, which might be potential outliers.

```
Df_doc = df[ ((df["year"]== 2019) |

        ((df["year"]== 2020) & ((df["month"] ==("January"))| (df["month"] ==("February"))| (df["month"] ==("March"))| (df["month"] ==("April")) | (df["month"] ==("May")) | (df["month"] ==("June")))) )
```

The code you provided filters the DataFrame df to create a new DataFrame df_doc based on specific conditions related to the columns 'year' and 'month.This part of the condition checks if the 'year' column in the DataFrame df is equal to 2019.This part of the condition checks if the 'year' column in the DataFrame df is equal to 2020.The resulting DataFrame df_doc will contain only the rows that meet these conditions, effectively filtering the dataset to include data for movies that meet these year and month criteria.

```
Fig = px.scatter(df_doc, x='year', y='IMDB Score',color="month")

Fig.update_traces(marker_size=10)

Fig.show()
```

The code you provided uses Plotly Express to create a scatter plot for the DataFrame df_doc.Finally, this command displays the scatter plot you've created.

The resulting plot is a scatter plot where the x-axis represents the 'year,' the y-axis represents the 'IMDB Score,' and each data point is color-coded according to the 'month.' This allows you to visualize how IMDb scores for movies in your dataset vary over different years and months. The marker size is also increased to make the data points more visible and distinguishable.

```
top_imdb_english = df[df['Language'] == "English"]
```

```
top_imdb_english =
top_imdb_english.groupby(['Language','Genre','Title']).mean().sort_values(by=["IMDB
Score"],ascending=False)[:10]
```

```
top_imdb_english
```

The code you provided is designed to filter and analyze the DataFrame df to identify the top 10 English-language movies with the highest IMDb scores. Following the group-by and mean calculation, this line sorts the resulting DataFrame by the 'IMDB Score' column in descending order (highest IMDb scores first).

The final DataFrame top_imdb_english contains the top 10 English-language movies with the highest IMDb scores, along with their associated language, genre, and title. This analysis allows you to identify the best-rated English-language films in your

```
dataset.df_hindi = df[df["Language"] == "Hindi"]
```

```
Df_hindi.Runtime.value_counts()
```

```
Df_hindi.Runtime.mean()
```

The code you provided is focused on analyzing a subset of your DataFrame df, specifically for movies with the language "Hindi."

This line filters the original DataFrame df to create a new DataFrame df_hindi that includes only the rows where the 'Language' column is equal to "Hindi." It effectively selects movies with Hindi as the spoken language.This line calculates the count of unique values in the 'Runtime' column within the DataFrame df_hindi. It returns a Series that displays how many movies fall into each unique runtime category. In other words, it shows the frequency of different runtime values for Hindi-language movies.

So, by running these lines of code, you can gather insights about Hindi-language movies, such as how many movies have different runtime values and what the average runtime duration is for these movies. This kind of analysis can be useful for understanding trends and characteristics within a specific subset of your data.

```
Df['Genre'].value_counts()
```

```
Df['Genre'].value_counts().sum()
```

```
Genre =df['Genre'].value_counts()
```

The code you provided is focused on analyzing the 'Genre' column within your DataFrame, dfThis line assigns the Series of genre counts to a variable named genre. This allows you to store and reference these counts for further analysis or visualization.In summary, these lines of code help you understand the distribution of movie genres in your dataset. The value_counts() function provides a count of how many movies belong to each genre, and the

sum() function calculates the total number of movies in your dataset. The genre variable holds these genre counts for potential use in further analysis or reporting.

```
Fig = px.bar(genre, x= genre.index, y=genre.values, labels={'y':'Number of Movies from the Genre', 'index':'Genres'})

Fig.update_layout(xaxis={'categoryorder':'total descending'})

Fig.show()
```

The code you provided is using Plotly Express to create a bar chart to visualize the distribution of movie genres in your dataset. This line creates a bar chart using Plotly Express (px.bar).Genre is the variable that holds a Series with the count of movies for each .This line updates the layout of the chart. Specifically, it sets the category order of the x-axis to 'total descending.' This means that the genres will be sorted in descending order based on the total number of movies, with the genre having the most movies displayed first.Finally, this command displays the bar chart you've created.

```
# printing unique values of Language

Df.Language.unique()

Df.Language.value_counts()
```

The code you provided is used to analyze and display unique values and their counts in the 'Language' column of your DataFrame df.This line returns an array of unique values found in the 'Language' column. It effectively lists all the distinct languages present in your dataset.This line calculates the count of occurrences for each unique language in the 'Language' column.

```
Df_top_lang = df.Language.value_counts().nlargest(3)
```

The code you provided is used to create a new DataFrame df_top_lang that contains the top three languages with the highest number of movies in your dataset. So, the resulting df_top_lang DataFrame will contain the top three languages with the highest number of movies in your dataset. This can be useful for focusing your analysis on the most prevalent languages or for further exploration of movies in these languages.

```
# plotting a bar graph for better visualisation

Plt.figure(figsize=(12,8))

Sns.barplot(x=df_top_lang.index,y=df_top_lang.values,data=df,color='blue')

Plt.xlabel('Top 3 Languages in Movies')

Plt.xticks(rotation=90)

Plt.ylabel('Number')

Plt.show()
```

The code you provided is used to create a bar graph for better visualization of the top three languages with the highest number of movies in your dataset.This line sets up the figure for the plot, specifying the figure size with a width of 12 units and a height of 8 units.The resulting bar graph visually represents the top three languages with the highest number of movies in your

dataset. This visualization provides a clear overview of the distribution of movies among these languages.

```
Df[['IMDB Score','Runtime']].corr()
```

The code you provided calculates the correlation between two numeric columns in your DataFrame, specifically the 'IMDB Score' and 'Runtime' columns. This part of the code selects a subset of your DataFrame df, After selecting these two columns, you use the .corr(). The output of this code allows you to assess the relationship between IMDb scores and movie runtimes. If the correlation coefficient is close to 1, it might suggest that longer movies tend to have higher IMDb scores, or if it's close to -1, it could imply the opposite. A value close to 0 suggests a weak or no linear relationship between the two variables.

```
Fig = px.scatter(df, x='IMDB Score', y='Runtime')

Fig.show()
```

The code you provided is using the plotly.express library (commonly imported as px) in Python to create a scatter plot. Px.scatter(df, x='IMDB Score', y='Runtime'): This line of code creates a scatter plot using the DataFrame 'df' as the data source. It specifies that the 'IMDB Score' column should be used for the x-axis (horizontal) and the 'Runtime' column should be used for the y-axis (vertical). The resulting scatter plot will show individual data points, where each point represents a movie. The 'IMDB Score' will be on the x-axis, and 'Runtime' will be on the y-axis. Scatter plots like this are useful for visualizing the relationship or distribution of data points between two numerical variables

```
Df_temp=df.groupby(['Genre']).mean(['IMDB rating']).sort_values(by='IMDB Score',
ascending=False).reset_index().iloc[:10,:]


Fig, ax = plt.subplots(1,1, figsize = (10, 6), constrained_layout = True)

Ax = sns.barplot(x = 'Genre', y = 'IMDB Score', data = df_temp, color = 'violet')


For i in ax.patches:

  Ax.text(x = i.get_x() + i.get_width()/2, y = i.get_height()/2,

      S = f"{round(i.get_height(),1)}",

      Ha = 'center', size = 14, weight = 'bold', rotation = 0, color = 'white',

      Bbox=dict(boxstyle="round,pad=0.5", fc='pink', ec="pink", lw=2))


  Ax.set_xlabel('Genre', fontsize=14)

Ax.set_ylabel('Average IMDB Score', fontsize=14)

Ax.set_xticklabels([i[:15] for i in df_temp['Genre'].unique()], fontsize=12, rotation = -45 )
```

Plt.title('Top 10 Genre by IMDB Score', fontsize=16);

This code is designed to create a bar plot using the matplotlib and seaborn libraries in Python. It appears to visualize the top 10 movie genres based on their average IMDb scores. This line creates a new DataFrame df_temp by grouping the original DataFrame df by the 'Genre' column and calculating the mean IMDb score for each genre. Fig, ax = plt.subplots(1, 1, figsize=(10, 6), constrained_layout=True): plt.title('Top 10 Genre by IMDb Score', fontsize=16): This line sets the title for the plot as 'Top 10 Genre by IMDb Score' and adjusts its font size.This line initializes a figure and axis for your plot. It sets the size of the figure to 10x6 inches.Overall, this code generates a visually appealing bar plot that displays the top 10 movie genres with the highest average IMDb scores, making it easier to compare and understand the IMDb scores of different genres.

Df_temp=df.groupby(['Title']).mean(['Runtime rating']).sort_values(by='Runtime', ascending=False).reset_index().iloc[:10,:2]

plotting a bar graph for better visualisation

Plt.figure(figsize=(12,8))

Sns.barplot(x=df_temp["Title"],y=df_temp["Runtime"],data=df,palette='rainbow')

Plt.xlabel('Top 10 Highest Runtime Movies')

Plt.xticks(rotation=90)

Plt.ylabel('Number')

Plt.show()

This code is designed to create a bar plot to visualize the top 10 movies with the highest runtime based on the 'Runtime' column in the DataFrame 'df'. Df_temp: This line creates a new DataFrame df_temp by grouping the original DataFrame df by the 'Title' column and calculating the mean 'Runtime' for each movie. Plt.figure(figsize=(12,8)): This line initializes a figure for the plot and sets its size to 12x8 inches.plt.ylabel('Number'): This line sets the label for the y-axis as 'Number'.Plt.show(): This line displays the bar plot.

The resulting bar plot will show the top 10 movies with the highest runtime, and each bar's height represents the runtime of a particular movie.

Top_genres = df.loc[df['Genre'].isin(df.groupby('Genre').sum().sort_values(by='IMDB Score', ascending=False).reset_index()['Genre'][:10])].groupby('Genre').mean().sort_values(by='IMDB Score', ascending=False).reset_index()['Genre']

Plt.figure(figsize= (10, 6))

Sns.countplot(x = df.loc[df['Genre'].isin(top_genres)]['year'],

Hue= df.loc[df['Genre'].isin(top_genres)]['Genre'])

Plt.title('Released Genre per Year', size= 25)

```
Plt.xlabel(None)

Plt.xticks(size= 16)

Plt.show()
```

This code is designed to create a countplot to visualize the distribution of the top 10 movie genres, ranked by their average IMDb scores, over the years. Top_genres: This line of code filters the DataFrame 'df' to include only the top 10 movie genres with the highest average IMDb scores. Plt.figure(figsize=(10, 6)): This line initializes a figure for the plot and sets its size to 10x6 inches.sns.countplot(x=df.loc[df['Genre'].isin(top_genres)]['year'], hue=df.loc[df['Genre'].isin(top_genres)]['Genre']): This line creates a countplot using the seaborn library. It specifies that the x-axis should represent the 'year' column, and the hue (color) should be based on the 'Genre' column

The resulting countplot will show how many movies from the top 10 genres with the highest IMDb scores were released each year. Each bar in the plot represents a year, and the bars are divided by color to show the distribution of movies from different genres. This visualization allows you to see how the popularity of these top genres has changed over the years.

```
Df_temp=df.groupby(['year']).mean(['Runtime rating']).sort_values(by='Runtime', ascending=False).reset_index().iloc[:10,:2]


# plotting a bar graph for better visualisation

Plt.figure(figsize=(12,8))

Sns.barplot(x=df_temp["year"],y=df_temp["Runtime"],data=df,palette='rainbow')

Plt.xlabel('Top 10 Year – Runtime Movies')

Plt.xticks(rotation=90)

Plt.ylabel('Runtime')

Plt.show()
```

This code is designed to create a bar plot to visualize the top 10 years with the highest average movie runtime based on the 'Runtime' column in the DataFrame 'df'. Df_temp: This line creates a new DataFrame df_temp by grouping the original DataFrame 'df' by the 'year' column and calculating the mean 'Runtime' for each year.plt.figure(figsize=(12, 8)): This line initializes a figure for the plot and sets its size to 12x8 inchesplt.ylabel('Runtime'): This line sets the label for the y-axis as 'Runtime', indicating the average runtime of movies.

The resulting bar plot will show the top 10 years with the highest average movie runtimes. Each bar represents a year, and the height of the bar represents the average runtime of movies released in that year. This visualization allows you to see how movie runtimes have varied over the years and identify the years with the longest average runtimes based on the 'Runtime' column in the dataset.

```
Df_run= df[df["year"] ==2021]

Df_run.Runtime.mean()
```

The code you provided calculates the average runtime of movies released in the year 2021 in the DataFrame 'df'. Df_run = df[df["year"] == 2021]: This line filters the DataFrame 'df' to select only the rows where the 'year' column has a value of 2021. It computes the average movie runtime for all the movies released in 2021.So, the result of this code will be the average runtime of movies that were released in the year 2021. It provides insight into the typical length of movies released during that specific year in your dataset.

```
Genre_lang =[]

For i in df.Language.unique():

  Df_lang =df[df["Language"]==i]

  Df_lang_genre =df_lang.Genre.value_counts().nlargest(1)

  #print(f'**********************************************')

  #print(f'Language: {i}\n', df_lang_genre)

  Genre_lang.append((i,df_lang_genre))
```

The code you provided appears to be an iterative process for analyzing and collecting information about the most common movie genres for each unique language in the DataFrame 'df'. Genre_lang = []: This line initializes an empty list called 'genre_lang' that will be used to store information about the most common movie genres for each unique language.for i in df.Language.unique():: This line starts a loop that iterates through the unique values in the 'Language' column of the DataFrame 'df'. Genre_lang.append((i, df_lang_genre)): This line appends a tuple to the 'genre_lang' list.

The resulting 'genre_lang' list will contain pairs of languages and their most common movie genres. It can be used to understand which genres are prevalent in movies of different languages in your dataset. You can uncomment the print statements to see the results printed in a more human-readable format.

```
Df_lang = pd.DataFrame(genre_lang, columns = ['Language', 'Genre'])

Df_lang.sort_values(by=['Language'],ignore_index=True)
```

The code you provided is used to create a new DataFrame named 'df_lang' and populate it with the information collected in the 'genre_lang' list. Df_lang = pd.DataFrame(genre_lang, columns=['Language', 'Genre']): This line creates a new DataFrame 'df_lang' using the data from the 'genre_lang' list. The 'genre_lang' list contains tuples where each tuple has two elements: the language and the most common genre for movies in that language.

The resulting 'df_lang' DataFrame will have two columns: 'Language' and 'Genre'. The rows will be sorted based on the 'Language' column in alphabetical order. This DataFrame provides a structured and ordered view of the most common movie genres for each unique language, making it easier to analyze and interpret the data.

```
Fig = px.scatter(x=df['Runtime'], y=df['Title'])

Fig.show()
```

The code you provided uses the plotly.express library to create a scatter plot.fig = px.scatter(x=df['Runtime'], y=df['Title']): This line sets up a scatter plot using the 'Runtime'

column as the x-axis (horizontal) and the ‘Title’ column as the y-axis (vertical).fig.show(): This line displays the scatter plot using the ‘fig’ object. The plot will appear in your Python environment or as part of an interactive visualization if you are using Jupyter Notebook or another environment that supports Plotly.

The resulting scatter plot will show individual data points where each point represents a movie. The ‘Runtime’ values will be on the x-axis, and the ‘Title’ of the movies will be on the y-axis. Scatter plots like this are useful for visualizing the distribution and relationships between two numerical variables, in this case, the runtime of movies and their titles. However, it’s worth noting that using movie titles on the y-axis of a scatter plot is not a typical or recommended way to visualize data, as it can be quite crowded and not very informative. It might be more appropriate to use other types of plots, like bar charts or histograms, for visualizing such data.