In [1]:
```
import numpy as np
```

In [2]:
```
import pandas as pd
```

In [3]:
```
import seaborn as sns
```

In [4]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
# uploading of dataset
```

In [39]:
```
df = pd.read_csv('movies.csv')
```

In [40]:
```
df
```

Out[40]:

	index	movie_name	year_of_release	category	run_time	genre	imdb_rating	votes
0	1.0	The Godfather	(1972)	R	175 min	Crime, Drama	9.2	1,860,471
1	2.0	The Silence of the Lambs	(1991)	R	118 min	Crime, Drama, Thriller	8.6	1,435,344
2	3.0	Star Wars: Episode V - The Empire Strikes Back	(1980)	PG	124 min	Action, Adventure, Fantasy	8.7	1,294,805
3	4.0	The Shawshank Redemption	(1994)	R	142 min	Drama	9.3	2,683,302
4	5.0	The Shining	(1980)	R	146 min	Drama, Horror	8.4	1,025,560
...	...	...	...	...	...	...	...	...
94	95.0	The Usual Suspects	(1995)	R	106 min	Crime, Drama, Mystery	8.5	1,087,832
95	96.0	Cool Hand Luke	(1967)	GP	127 min	Crime, Drama	8.1	178,888
96	97.0	Eternal Sunshine of the Spotless Mind	(2004)	R	108 min	Drama, Romance, Sci-Fi	8.3	1,011,004
97	98.0	City Lights	(1931)	G	87 min	Comedy, Drama, Romance	8.5	186,059
98	99.0	The Matrix	(1999)	R	136 min	Action, Sci-Fi	8.7	1,916,083

99 rows × 9 columns

In [7]:
```
#basic operations
```

In [41]:
```
df.head()
```

Out[41]:

	index	movie_name	year_of_release	category	run_time	genre	imdb_rating	votes
0	1.0	The Godfather	(1972)	R	175 min	Crime, Drama	9.2	1,860,471
1	2.0	The Silence of the Lambs	(1991)	R	118 min	Crime, Drama, Thriller	8.6	1,435,344
2	3.0	Star Wars: Episode V - The Empire Strikes Back	(1980)	PG	124 min	Action, Adventure, Fantasy	8.7	1,294,805
3	4.0	The Shawshank Redemption	(1994)	R	142 min	Drama	9.3	2,683,302
4	5.0	The Shining	(1980)	R	146 min	Drama, Horror	8.4	1,025,560

In [42]:
```
df.tail()
```

Out[42]:

	index	movie_name	year_of_release	category	run_time	genre	imdb_rating	votes
94	95.0	The Usual Suspects	(1995)	R	106 min	Crime, Drama, Mystery	8.5	1,087,832
95	96.0	Cool Hand Luke	(1967)	GP	127 min	Crime, Drama	8.1	178,888
96	97.0	Eternal Sunshine of the Spotless Mind	(2004)	R	108 min	Drama, Romance, Sci-Fi	8.3	1,011,004
97	98.0	City Lights	(1931)	G	87 min	Comedy, Drama, Romance	8.5	186,059
98	99.0	The Matrix	(1999)	R	136 min	Action, Sci-Fi	8.7	1,916,083

In [43]:
```
df.shape
```

Out[43]:
```
(99, 9)
```

In [44]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   index            99 non-null     float64
 1   movie_name       99 non-null     object 
 2   year_of_release  99 non-null     object 
 3   category         99 non-null     object 
 4   run_time         99 non-null     object 
 5   genre            99 non-null     object 
 6   imdb_rating      99 non-null     float64
 7   votes            99 non-null     object 
 8   gross_total      98 non-null     object 
dtypes: float64(2), object(7)
memory usage: 7.1+ KB
```

In [45]:
```
df.describe()
```

Out[45]:

	index	imdb_rating
count	99.000000	99.000000
mean	50.000000	8.348485
std	28.722813	0.368772
min	1.000000	7.200000
25%	25.500000	8.100000
50%	50.000000	8.300000
75%	74.500000	8.600000
max	99.000000	9.300000

In [53]:
```
df.isnull().sum()
```

Out[53]:
```
index              0
movie_name         0
year_of_release    0
category           0
run_time           0
genre              0
imdb_rating        0
votes              0
gross_total        1
dtype: int64
```

In [60]:
```
mean = df.mean()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_17880\4294835594.py:1: FutureWarni
ng: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_onl
y=None' is deprecated. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  mean = df.mean()
```

In [61]:
```
mean
```

Out[61]:
```
index          50.000000
imdb_rating     8.348485
dtype: float64
```

In [62]:
```
df.fillna(mean,inplace=True)
```

In [64]:
```
mode=df.gross_total.mode
```

In [65]:
```
mode
```

Out[65]:
```
<bound method Series.mode of 0     $134.97M
1     $130.74M
2     $290.48M
3      $28.34M
4      $44.02M
        ...
94     $23.34M
95     $16.22M
96     $34.40M
97      $0.02M
98    $171.48M
Name: gross_total, Length: 99, dtype: object>
```

In [66]:
```
df.gross_total.fillna(mode,inplace=True)
```

In [67]:
```
df
```

Out[67]:

	index	movie_name	year_of_release	category	run_time	genre	imdb_rating	votes
0	1.0	The Godfather	(1972)	R	175 min	Crime, Drama	9.2	1,860,471
1	2.0	The Silence of the Lambs	(1991)	R	118 min	Crime, Drama, Thriller	8.6	1,435,344
2	3.0	Star Wars: Episode V - The Empire Strikes Back	(1980)	PG	124 min	Action, Adventure, Fantasy	8.7	1,294,805
3	4.0	The Shawshank Redemption	(1994)	R	142 min	Drama	9.3	2,683,302
4	5.0	The Shining	(1980)	R	146 min	Drama, Horror	8.4	1,025,560
...	...	...	...	...	...	...	...	...
94	95.0	The Usual Suspects	(1995)	R	106 min	Crime, Drama, Mystery	8.5	1,087,832
95	96.0	Cool Hand Luke	(1967)	GP	127 min	Crime, Drama	8.1	178,888
96	97.0	Eternal Sunshine of the Spotless Mind	(2004)	R	108 min	Drama, Romance, Sci-Fi	8.3	1,011,004
97	98.0	City Lights	(1931)	G	87 min	Comedy, Drama, Romance	8.5	186,059
98	99.0	The Matrix	(1999)	R	136 min	Action, Sci-Fi	8.7	1,916,083

99 rows × 9 columns

In [69]:
```
df.isnull().sum()
```

Out[69]:
```
index              0
movie_name         0
year_of_release    0
category           0
run_time           0
genre              0
imdb_rating        0
votes              0
gross_total        0
dtype: int64
```

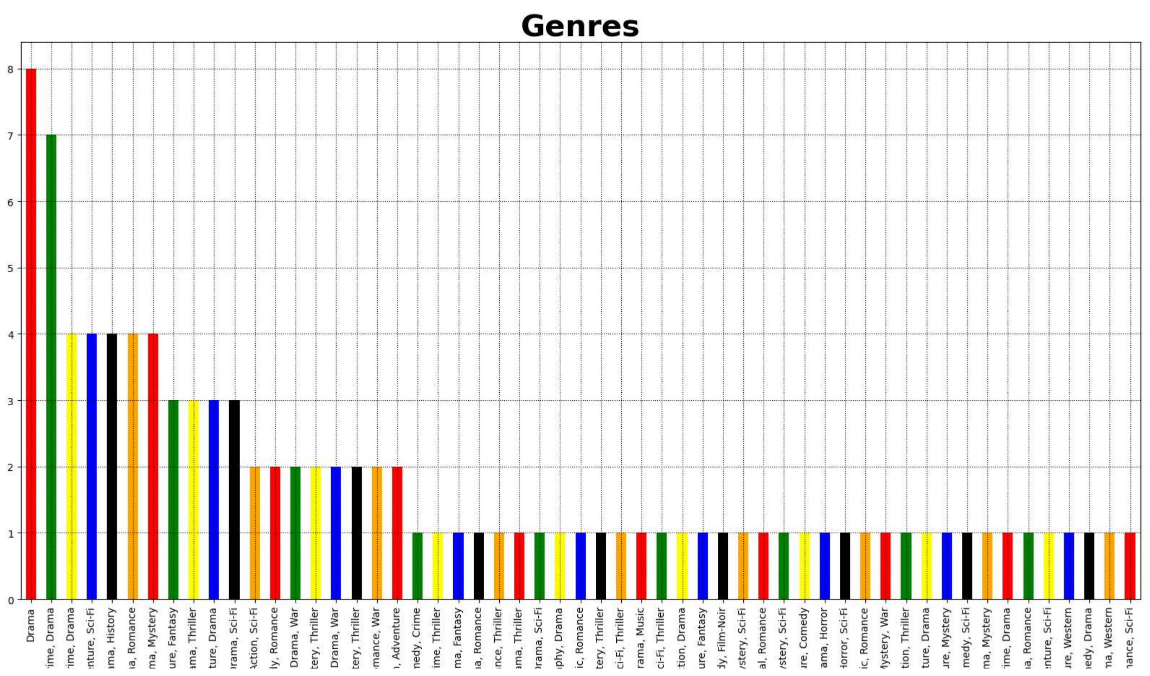
In [70]:
```
# EDA
```

In [ ]:
```
```

```
In [78]: df['genre'].value_counts()
```

Out[78]:
```
Drama                           8
Crime, Drama                    7
Action, Crime, Drama            4
Action, Adventure, Sci-Fi       4
Biography, Drama, History       4
Comedy, Drama, Romance          4
Crime, Drama, Mystery           4
Action, Adventure, Fantasy      3
Crime, Drama, Thriller          3
Action, Adventure, Drama        3
Action, Drama, Sci-Fi           3
Action, Sci-Fi                  2
Comedy, Romance                 2
Drama, War                      2
Mystery, Thriller               2
Adventure, Drama, War           2
Drama, Mystery, Thriller        2
Drama, Romance, War             2
Action, Adventure               2
Adventure, Comedy, Crime        1
Crime, Thriller                 1
Adventure, Drama, Fantasy       1
Crime, Drama, Romance           1
Mystery, Romance, Thriller      1
Biography, Drama, Thriller      1
Adventure, Drama, Sci-Fi        1
Biography, Drama                1
Drama, Music, Romance           1
Film-Noir, Mystery, Thriller    1
Mystery, Sci-Fi, Thriller       1
Biography, Drama, Music         1
Action, Sci-Fi, Thriller        1
Action, Drama                   1
Adventure, Fantasy              1
Adventure, Comedy, Film-Noir    1
Horror, Mystery, Sci-Fi         1
Comedy, Musical, Romance        1
Drama, Mystery, Sci-Fi          1
Animation, Adventure, Comedy    1
Drama, Horror                   1
Horror, Sci-Fi                  1
Comedy, Music, Romance          1
Drama, Mystery, War             1
Action, Thriller                1
Animation, Adventure, Drama     1
Action, Adventure, Mystery      1
Adventure, Comedy, Sci-Fi       1
Drama, Mystery                  1
Biography, Crime, Drama         1
Drama, Romance                  1
Adventure, Sci-Fi               1
Adventure, Western              1
Comedy, Drama                   1
Drama, Western                  1
Drama, Romance, Sci-Fi          1
Name: genre, dtype: int64
```

In [83]:
```
plt.figure(figsize=(20,10))
df['genre'].value_counts().plot(kind = 'bar' , color = ['red','green','yellow'
plt.grid(c='k', ls=':')
plt.xlabel('Types of Genres')
plt.title('Genres' , fontsize = '30' , fontweight = 'bold' , c = 'k')
plt.show()
```



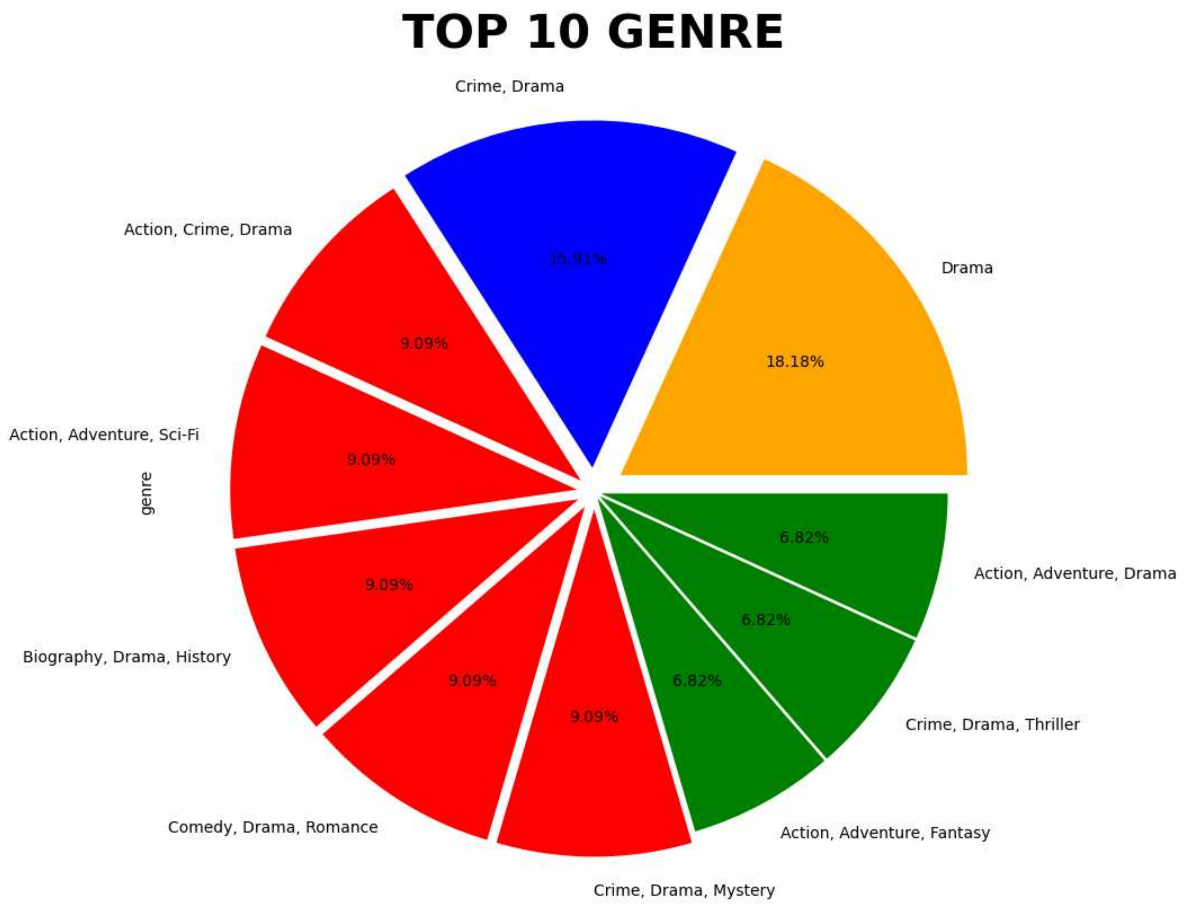
Top 10 Genre

In [84]:
```
x=df['genre'].value_counts(ascending = False).head(10)
```

In [85]:
```
x
```

Out[85]:
```
Drama                        8
Crime, Drama                 7
Action, Crime, Drama         4
Action, Adventure, Sci-Fi    4
Biography, Drama, History    4
Comedy, Drama, Romance       4
Crime, Drama, Mystery        4
Action, Adventure, Fantasy   3
Crime, Drama, Thriller       3
Action, Adventure, Drama     3
Name: genre, dtype: int64
```

In [89]:
```
plt.figure(figsize=(20,10))
x=df['genre'].value_counts(ascending = False).head(10).plot(kind = 'pie' ,
                     colors = ['orange','blue','red','red','red','red','red','
                    autopct= '%0.2f%%',
                     explode = [0.09,0.07,0.05,0.05,0.05,0.05,0.05,0.02,0.02,0
plt.title('TOP 10 GENRE', fontsize = '30',fontweight='bold', c='k')
plt.show()
```



Top 10 popular movies

In [140]:
```
B = df[['movie_name','run_time','imdb_rating',]].sort_values(by=['imdb_rating'
```

In [161]:
```
B
```

Out[161]:

	movie_name	run_time	imdb_rating
3	The Shawshank Redemption	142 min	9.3
0	The Godfather	175 min	9.2
13	Schindler's List	195 min	9.0
21	12 Angry Men	96 min	9.0
8	The Lord of the Rings: The Return of the King	201 min	9.0
...	...	...	...
84	O Brother, Where Art Thou?	107 min	7.7
39	As Good as It Gets	139 min	7.7
52	Lost in Translation	102 min	7.7
89	The Piano	121 min	7.5
48	Avanti!	144 min	7.2

99 rows × 3 columns

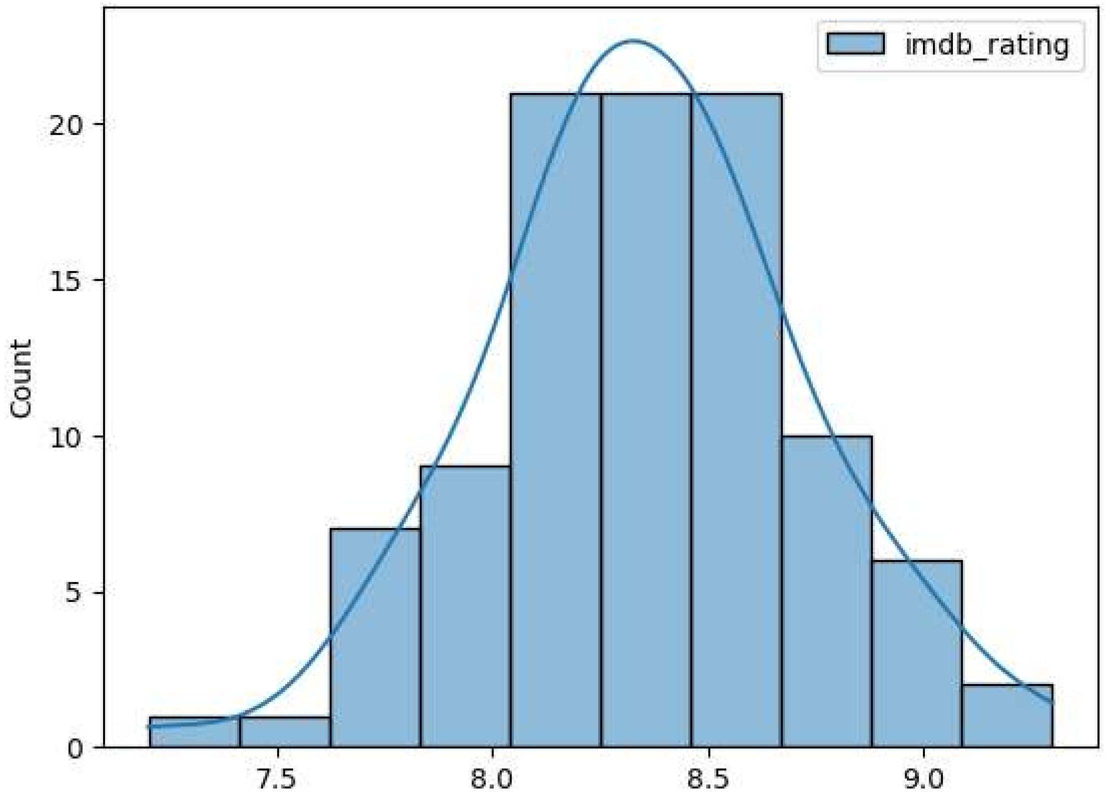
In [141]:
```
y=B[:10]
```

In [142]:
```
y
```

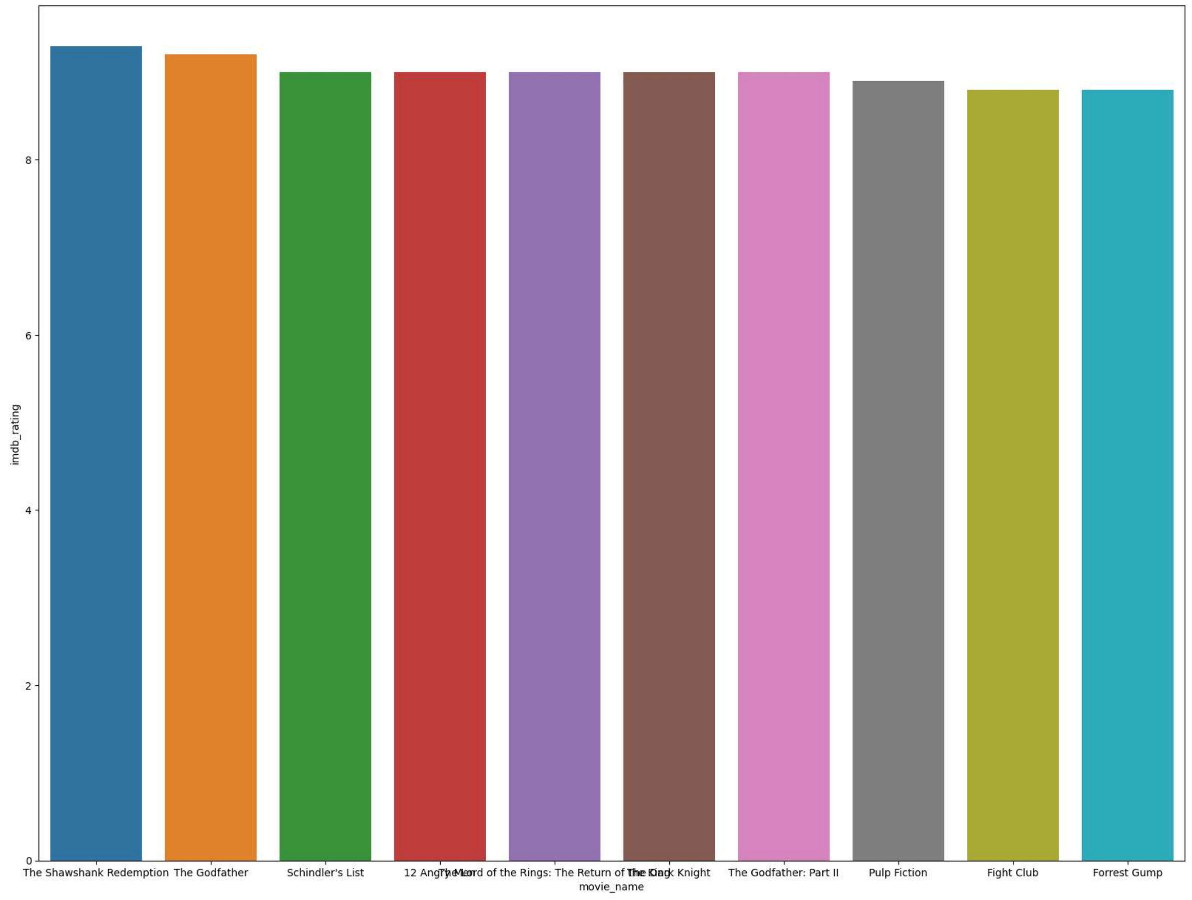
Out[142]:

	movie_name	run_time	imdb_rating
3	The Shawshank Redemption	142 min	9.3
0	The Godfather	175 min	9.2
13	Schindler's List	195 min	9.0
21	12 Angry Men	96 min	9.0
8	The Lord of the Rings: The Return of the King	201 min	9.0
10	The Dark Knight	152 min	9.0
11	The Godfather: Part II	202 min	9.0
28	Pulp Fiction	154 min	8.9
55	Fight Club	139 min	8.8
40	Forrest Gump	142 min	8.8

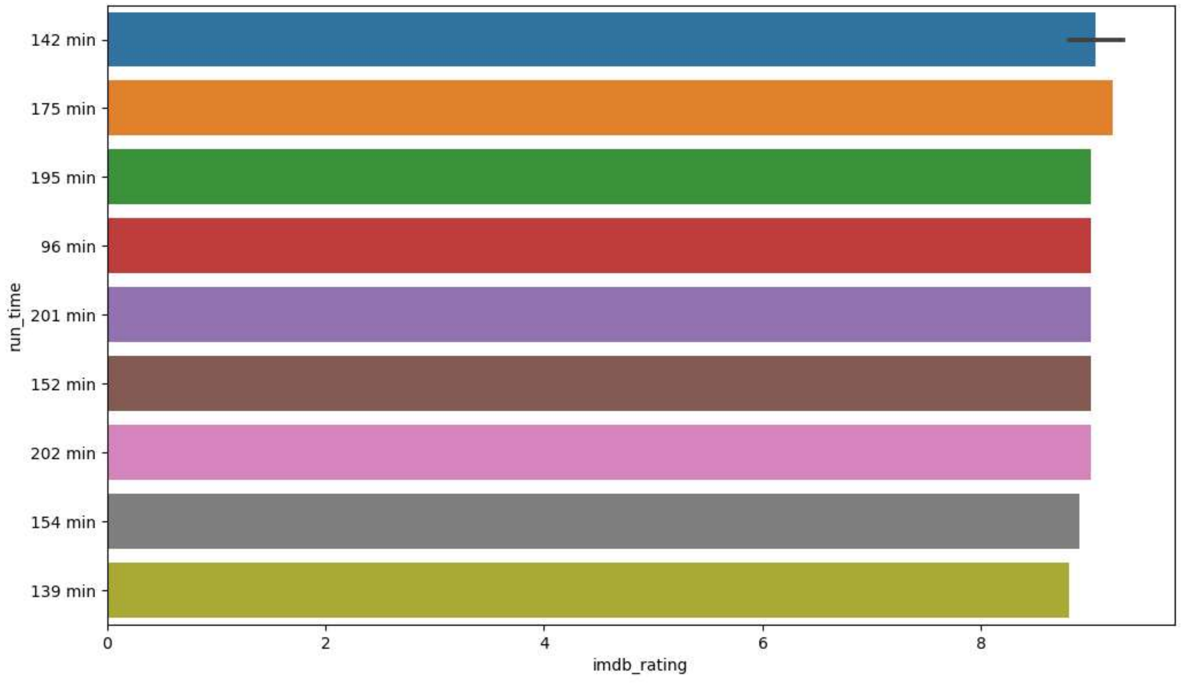
In [143]:
```
sns.histplot(B,kde=True)
plt.show()
```



In [145]:
```
plt.figure(figsize=(20,15))
sns.barplot(data = y ,x= 'movie_name',y= 'imdb_rating')
plt.show()
```



In [146]:
```
plt.figure(figsize=(12,7))
sns.barplot(data = y ,x= 'imdb_rating',y= 'run_time')
plt.show()
```



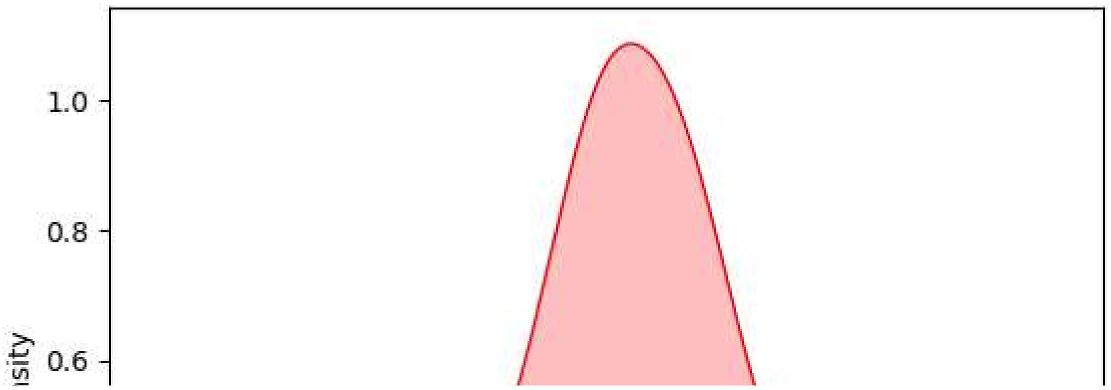
```
In [ ]:
```

```
In [147]: sns.kdeplot(data=B,x='imdb_rating',shade=True,color='r')
          plt.show()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_17880\2631213710.py:1: FutureWa
rning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=B,x='imdb_rating',shade=True,color='r')
```



```
In [ ]:
```

```
In [124]: #dist plot
In [126]: sns.distplot(df['imdb_rating'])
          plt.grid()
          plt.show()
```

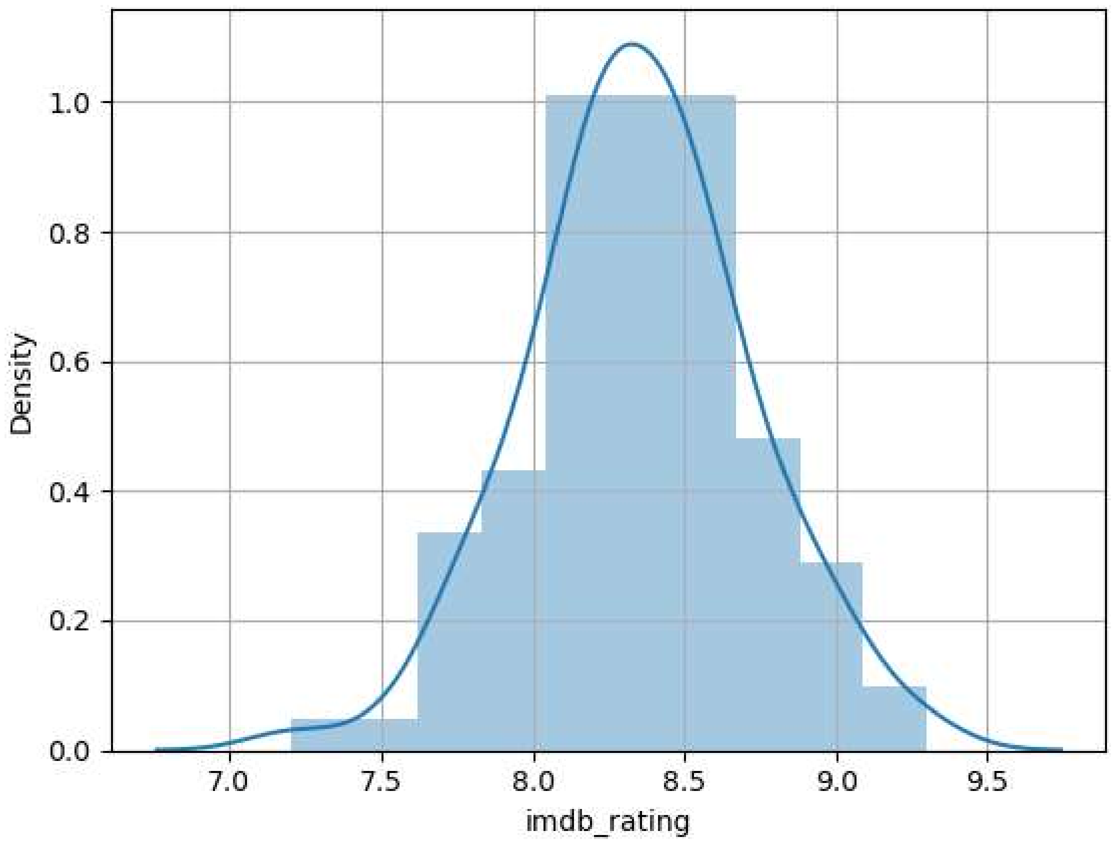
```
C:\Users\DELL\AppData\Local\Temp\ipykernel_17880\1240891931.py:1: UserWarnin
g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gis
t.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df['imdb_rating'])
```



Least 10 unpopular movie w.r.t imdb rating

```
In [136]: C = df[['movie_name','run_time','imdb_rating',]].sort_values(by=['imdb_rating'
```

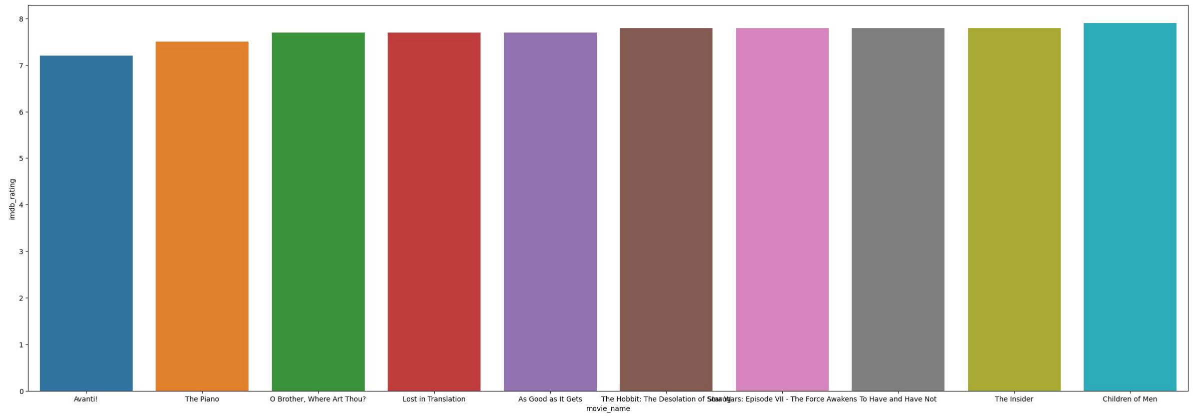
```
In [138]: c = C[:10]
```

In [139]:
```
c
```

Out[139]:

	movie_name	run_time	imdb_rating
48	Avanti!	144 min	7.2
89	The Piano	121 min	7.5
84	O Brother, Where Art Thou?	107 min	7.7
52	Lost in Translation	102 min	7.7
39	As Good as It Gets	139 min	7.7
80	The Hobbit: The Desolation of Smaug	161 min	7.8
79	Star Wars: Episode VII - The Force Awakens	138 min	7.8
65	To Have and Have Not	100 min	7.8
91	The Insider	157 min	7.8
58	Children of Men	109 min	7.9

In [152]:
```
plt.figure(figsize=(30,10))
sns.barplot(data = c ,x= 'movie_name',y= 'imdb_rating')
plt.show()
```



Least 10 popular movies w.r.t run time

In [ ]:
```

```

In [153]:
```
E = df[['movie_name','run_time','imdb_rating',]].sort_values(by=['run_time'] ,
```

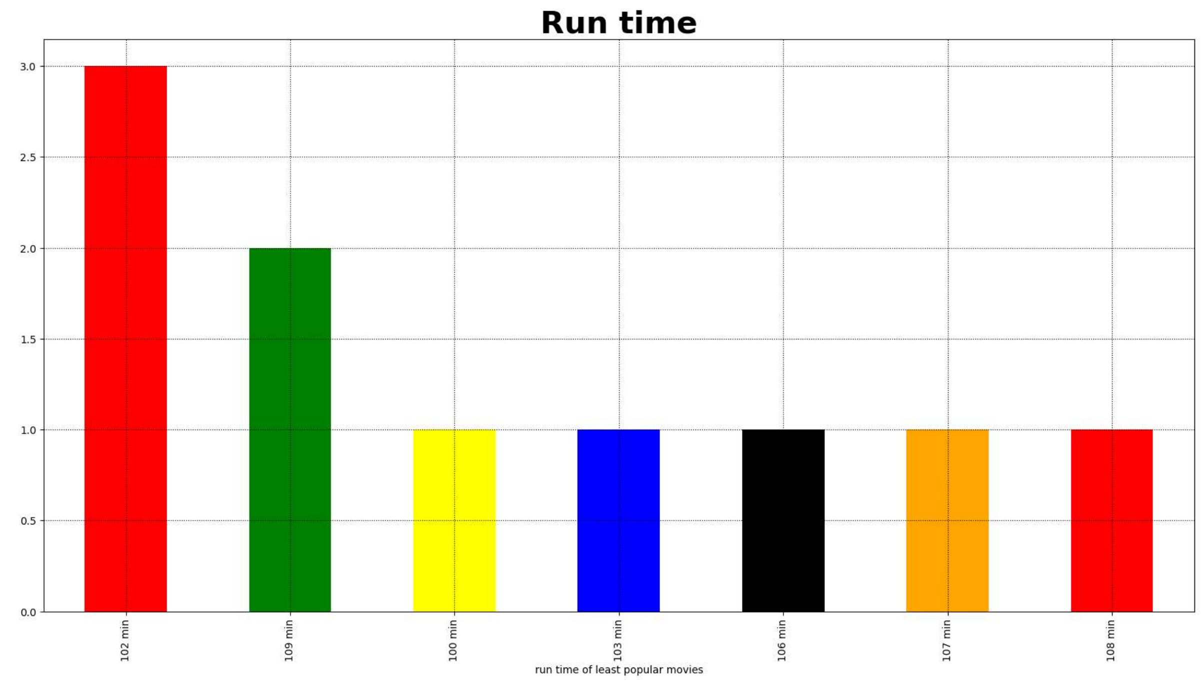
In [154]:
```
e = E[:10]
```

In [155]:
```
e
```

Out[155]:

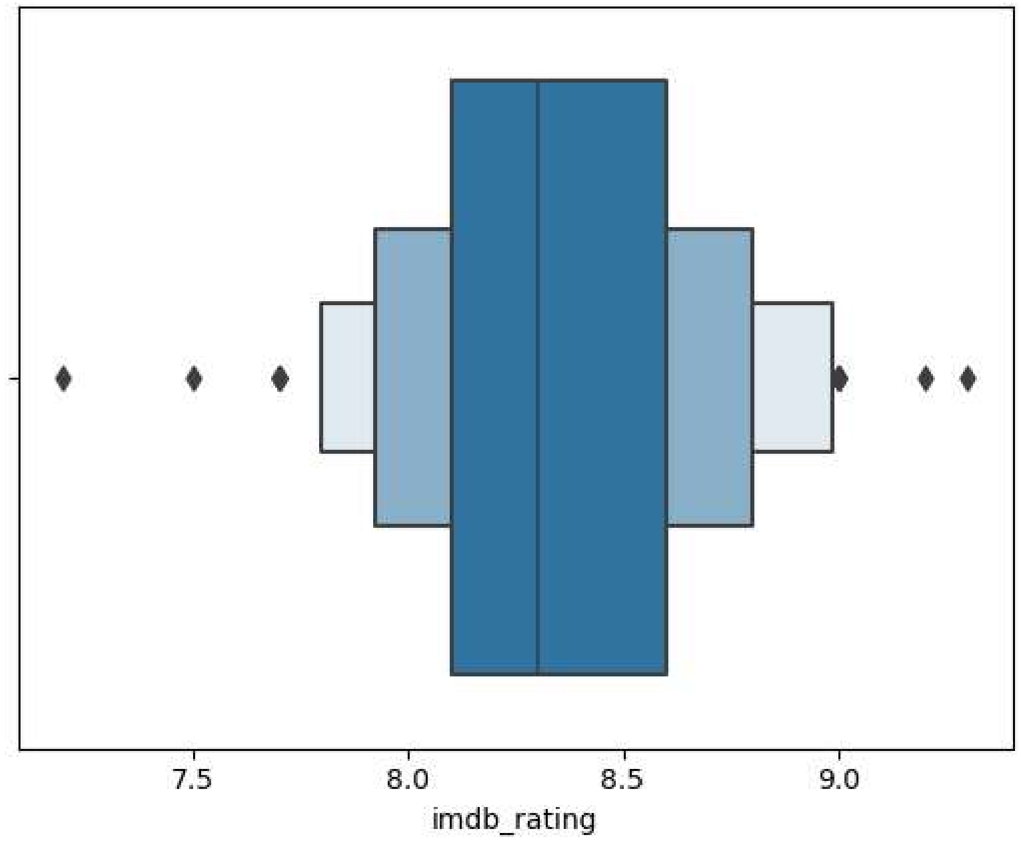
	movie_name	run_time	imdb_rating
65	To Have and Have Not	100 min	7.8
52	Lost in Translation	102 min	7.7
5	Casablanca	102 min	8.5
54	Requiem for a Dream	102 min	8.3
41	Singin' in the Rain	103 min	8.3
94	The Usual Suspects	106 min	8.5
84	O Brother, Where Art Thou?	107 min	7.7
96	Eternal Sunshine of the Spotless Mind	108 min	8.3
58	Children of Men	109 min	7.9
72	The Thing	109 min	8.2

In [157]:
```
plt.figure(figsize=(20,10))
e['run_time'].value_counts().plot(kind = 'bar' , color = ['red','green','yello
plt.grid(c='k', ls=':')
plt.xlabel('run time of least popular movies')
plt.title('Run time' , fontsize = '30' , fontweight = 'bold' , c = 'k')
plt.show()
```



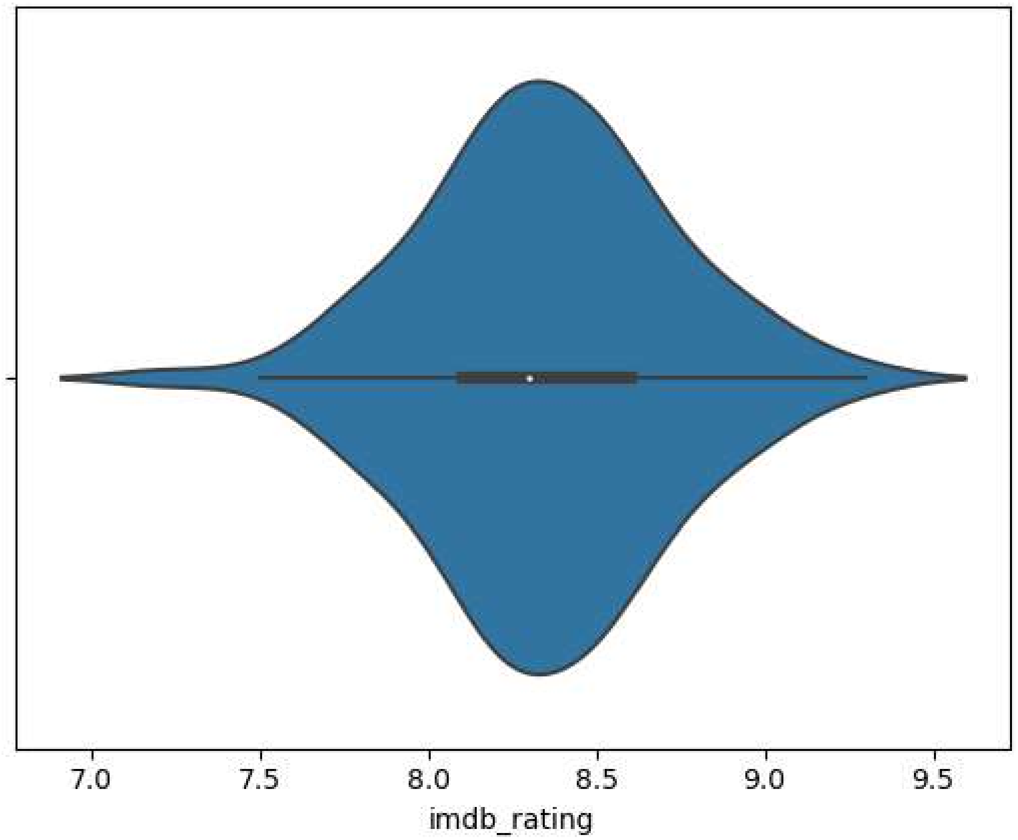
In [ ]:

In [162]:
```
sns.boxenplot(data=B,x='imdb_rating')
plt.show()
```



In [ ]:

In [163]:
```
sns.violinplot(data=df,x='imdb_rating')
plt.show()
```



Comparison w.r.t movie,imdb rating and votes

In [ ]:
```
```

In [168]:
```
F = df[['movie_name','imdb_rating','votes']].sort_values(by=['votes'] , ascend
```

In [169]:
```
F
```

Out[169]:

	movie_name	imdb_rating	votes
56	No Country for Old Men	8.2	977,336
7	Indiana Jones and the Raiders of the Lost Ark	8.4	969,143
87	A Beautiful Mind	8.2	935,549
79	Star Wars: Episode VII - The Force Awakens	7.8	933,771
89	The Piano	7.5	89,819
...	...	...	...
42	Braveheart	8.4	1,040,416
4	The Shining	8.4	1,025,560
96	Eternal Sunshine of the Spotless Mind	8.3	1,011,004
6	One Flew Over the Cuckoo's Nest	8.7	1,010,102
81	Mad Max: Fury Road	8.1	1,006,158

99 rows × 3 columns

In [170]:
```
f = F[:10]
```

In [171]:
```
f
```

Out[171]:

	movie_name	imdb_rating	votes
56	No Country for Old Men	8.2	977,336
7	Indiana Jones and the Raiders of the Lost Ark	8.4	969,143
87	A Beautiful Mind	8.2	935,549
79	Star Wars: Episode VII - The Force Awakens	7.8	933,771
89	The Piano	7.5	89,819
25	Die Hard	8.2	887,967
16	Alien	8.5	885,635
68	Slumdog Millionaire	8.0	848,344
54	Requiem for a Dream	8.3	845,362
27	Taxi Driver	8.2	836,871

In [ ]:
```
```

In [ ]:
```
```

Questions and Answers

Question 1: What is Pandas, and why is it commonly used in data cleaning tasks?

Ans. Pandas is an open-source data manipulation and analysis library for Python. It provides data structures for efficiently storing and manipulating large datasets and tools for working with structured data seamlessly. It further provides Data Import/Export, Data Exploration,Data Transformation, Indexing and Slicing And Integration with other libraries. Hence it is commonly used for Data cleaning.

Question 2: Given a DataFrame with missing values, how would you check for missing values in each column and count the total number of missing values?

Ans. With the help of isnull( ) command we will be able to find the missing values in each column . The syntax for the following will be df.isnull( ).sum( )

Question 3: How can you remove duplicates from a DataFrame while retaining the first occurrence of each unique row?

Ans. With the help of syntax " df.drop_duplicates( ) " we will be able to remove duplicates.

Question 4: If you have a DataFrame with a column containing string values, how can you convert all the values in that column to lowercase?

Ans. Suppose we are having a dataset Names having Uppercase elements the to convert it to lower case we use the syntax, " df[Name].str.lower( ) "

Question 5: How do you replace missing values in a DataFrame with a specific value, like 0, for a particular column?

Ans. Suppose we are having dataset A then , to replace the missing value with ' 0 ' we will use the syntax " df[A].fillna(0, inplace= True) ".

Question 6: If you have a DataFrame with a datetime column, how can you extract the year, month, and day into separate columns?

Ans. import pandas as pddata = {'Date': ['2021-01-15', '2022-02-20', '2023-03-25']},df = pd.DataFrame(data), df['Date'] = pd.to_datetime(df['Date']), df['Year'] = df['Date'].dt.year , df['Month'] = df['Date'].dt.month , df['Day'] = df['Date'].dt.day , print(df)

Question 7: How can you filter rows in a DataFrame where a specific column's values meet a certain condition (e.g., all rows where 'age' is greater than 30)?

Ans. With the help of syntax " df [df[Age] > 30] " we can filter the age greater than 30

from the given dataset , comprising of diffrent ages

Question 8: What is the purpose of the .apply() function in Pandas, and how would you use it to create a new column based on values from existing columns?

Ans. The apply() function in Pandas is used to apply a function along the axis of a DataFrame or Series. It allows you to perform a custom operation on each element of a DataFrame or Series. Suppose we are having to datasets A and B, then { def custom_function(row):, return row['A'] * row['B'] , Use apply() to create a new column 'C' based on values from 'A' and 'B' , df['C'] = df.apply(custom_function, axis=1 }

Question 9: Suppose you want to merge two DataFrames, 'df1' and 'df2,' on a common column 'key.' How would you perform this merge operation in Pandas?

Ans. We can perform a merge operation on two DataFrames using the merge()