ericks-on /
EDA-Film-industry

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

0 stars 1 fork 1 watching 1 Branch 0 Tags Activity

Public repository

main 1 Branch 0 Tags Go to file t Go to file + Add file Code

ericks-on finished recommendations 08d5755 · 8 hours ago

index_files	edit readme and graph	20 hours ago
.gitignore	new files	4 days ago
README.md	finished recommendations	8 hours ago
index.ipynb	finished recommendations	8 hours ago
movie_data_erd.jpeg	new files	4 days ago

README

Exploratory Data Analysis on the Film industry

Table of Contents

Business Understanding

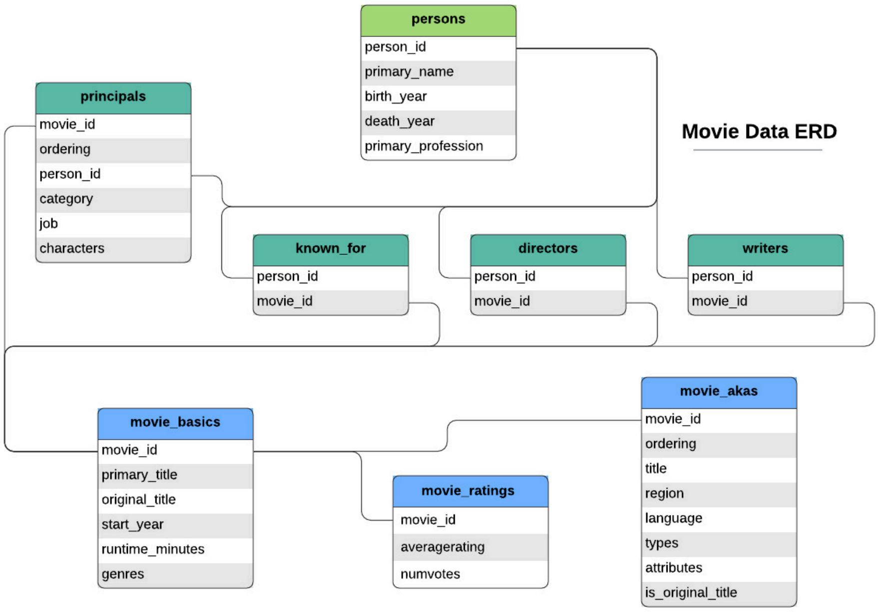
Your company now sees all the big companies creating original video content and they want to get in on the fun. They have decided to create a new movie studio, but they don't know anything about creating movies. You are charged with exploring what types of films are currently doing the best at the box office. You must then translate those findings into actionable insights that the head of your company's new movie studio can use to help decide what type of films to create.

Data Understanding

The data was collected from various locations, the different files have different formats.

- Box Office Mojo
- IMDB
- Rotten Tomatoes
- TheMovieDB
- The Numbers

Some are compressed CSV (comma-separated values) or TSV (tab-separated values), while the data from IMDB is located in a SQLite database.



Data Preparation

Loading the data

```
# Importing modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import seaborn as sns
import sqlite3

%matplotlib inline
```

```
# create a connection to sqlite3
conn = sqlite3.connect('zippedData/im.db')
```

```
# loading the data
bom_df = pd.read_csv('zippedData/bom.movie_gross.csv.gz')
rt_movie_info_df = pd.read_csv('zippedData/rt.movie_info.tsv.gz', delimiter='\t')
rt_reviews_df = pd.read_csv('zippedData/rt.reviews.tsv.gz', delimiter='\t', encoding='latin-1')
tmdb_df = pd.read_csv('zippedData/tmdb.movies.csv.gz')
tn_budgets_df = pd.read_csv('zippedData/tn.movie_budgets.csv.gz')
```

Box Office Mojo

```
bom_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

The data has 3397 entries with some data missing in some columns(studio, domestic_gross and foreign_gross).

There are 5 columns

Also the foreign gross is in string format

(a). cleaning 'foreign_gross' column

```
bom_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415,000,000	652000000	2010
1	Alice in Wonderland (2010)	BV	334,200,000	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296,000,000	664300000	2010
3	Inception	WB	292,600,000	535700000	2010
4	Shrek Forever After	P/DW	238,700,000	513900000	2010

We start off by converting the foreign_gross column to numeric data type(float64)

```
# first eliminate commas
bom_df['foreign_gross'] = bom_df.foreign_gross.map(
    lambda x: "".join(x.split(',')) if type(x) == str else x
)
bom_df['foreign_gross'] = bom_df.foreign_gross.astype(float)

bom_df.foreign_gross.dtype
```

```
dtype('float64')
```

Now that the column is in the correct format we can handle the missing values.

'**The Numbers**' dataset contains budgets for some movies. We can first check some of the missing entries in the dataset.

```
# The numbers dataset
tn_budgets_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

```
tn_budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5782 non-null   int64
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

no missing entries present

We start by cleaning the numeric columns to eliminate dollar sings and commas

```
def clean_money_cols(row):
    """Function to clean money columns in the tn_budgets df"""
    i = 3
    cols = ['production_budget', 'domestic_gross',      'worldwide_gross']
    while i < len(row):
        value = row[cols[i - 3]]
        if isinstance(value, str) and value.startswith('$'):
            # remove dollar sign
            value = value[1:]
            # eliminate the commas
            value = float(value.replace(',', ''))
        row[cols[i - 3]] = value
        # increment count
        i += 1
    return row

tn_budgets_df = tn_budgets_df.apply(
    lambda row: clean_money_cols(row), axis=1
)
```

Now that the amounts columns are in the right data type, we can add another column for foreign gross

```
foreign_gross = worldwide_gross - domestic_gross
```

```
# creating new column for foreign gross
tn_budgets_df['foreign_gross'] = (
    tn_budgets_df.worldwide_gross - tn_budgets_df.domestic_gross
)
tn_budgets_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	foreign_gross
0	1	Dec 18, 2009	Avatar	425,000,000	760,507,625	2,776,345,279	2,015,837,654
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410,600,000	241,063,875	1,045,663,875	804,600,000
2	3	Jun 7, 2019	Dark Phoenix	350,000,000	42,762,350	149,762,350	107,000,000
3	4	May 1, 2015	Avengers: Age of Ultron	330,600,000	459,005,868	1,403,013,963	944,008,095
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317,000,000	620,181,382	1,316,721,747	696,540,365

Fetching all the movies with missing foreign_gross

```
missing_foreign_gross = bom_df.loc[
    bom_df.foreign_gross.isna(),
    'title'
]

print('Movies missing foreign gross:', len(missing_foreign_gross))
```

```
Movies missing foreign gross: 1350
```

Now we try to get the foreign gross in the tn_budgets_df data

```
new_foreign_gross = tn_budgets_df.loc[
    tn_budgets_df.movie.isin(missing_foreign_gross),
    ['movie', 'foreign_gross']
]

# change columns to change movie to title
new_foreign_gross.columns = ['title', 'foreign_gross']

print('foreign gross entries found in "The Numbers" data:',
      len(new_foreign_gross))

new_foreign_gross.head()
```

```
foreign gross entries found in "The Numbers" data: 161
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	title	foreign_gross
588	Evolution	60,030,798
946	Rock Dog	14,727,942

	title	foreign_gross
1041	Bullet to the Head	13,108,140
1231	The Infiltrator	5,281,296
1290	All Eyez on Me	9,954,553

Found 161 of the missing values in the other dataframe. We now fill in the values in the bom dataframe

First we check if there are 0's in the new values which also indicate missing values and remove them

```
new_foreign_gross = new_foreign_gross.loc[
    new_foreign_gross.foreign_gross != 0
]

len(new_foreign_gross)
```

```
151
```

Down to 151. we now fill them in the dataframe

we start by defining a helper function

```
def fill_foreign_gross(row):
    """function to fill the foreign gross column"""
    if row.title in list(new_foreign_gross.title):
        row.foreign_gross = float(
            new_foreign_gross.loc[
                new_foreign_gross.title == row.title,
                'foreign_gross'
            ].values[0]
        )
    return row
```

filling the values

```
bom_df = bom_df.apply(
    lambda row: fill_foreign_gross(row),
    axis=1
)
```

```
bom_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2187 non-null   float64
 4   year            3387 non-null   int64  
dtypes: float64(2), int64(1), object(2)
memory usage: 132.4+ KB
```

for the remaining missing values we can fill with the mean foreign gross

```
# Handling the remaining missing values in the foreign_gross
foreign_mean = bom_df.foreign_gross.mean()

bom_df.foreign_gross.fillna(foreign_mean, inplace=True)

# check for missing values
bom_df.foreign_gross.isna().sum()
```

```
0
```

(b). Handling missing data in domestic_gross column

We do the same for the domestic column, first try to get the missing values in the other dataframe, then fill with the mean

```
# Get the missing entries in the domestic gross columns
missing_domestic_gross = bom_df.loc[
    bom_df.domestic_gross.isna(),
    'title'
]

print('Movies missing domestic gross:', len(missing_domestic_gross))
```

```
Movies missing domestic gross: 28
```

```
new_domestic_gross = tn_budgets_df.loc[
    tn_budgets_df.movie.isin(missing_domestic_gross),
    ['movie', 'domestic_gross']
]

# change columns to change movie to title
new_domestic_gross.columns = ['title', 'domestic_gross']

print('domestic gross entries found in "The Numbers" data:',
      len(new_domestic_gross))

new_domestic_gross.head()
```

```
domestic gross entries found in "The Numbers" data: 2
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	title	domestic_gross
3735	It's a Wonderful Afterlife	0
5382	All the Boys Love Mandy Lane	0

We found two of the movies missing the domestic gross but its still 0 meaning that its also missing in the other data source.

Therefore we replace the missing values with mean

```
# get the mean
domestic_mean = bom_df.domestic_gross.mean()

# fill the null values with mean
bom_df['domestic_gross'] = bom_df.domestic_gross.fillna(domestic_mean)

bom_df.domestic_gross.isna().sum()
```

```
0
```

no more missing values in the domestic gross column

```
bom_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
```

```
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3387 non-null   float64
 3   foreign_gross   3387 non-null   float64
 4   year            3387 non-null   int64  
dtypes: float64(2), int64(1), object(2)
memory usage: 132.4+ KB
```

The studio column has 5 missing entries

```
bom_df.studio.fillna('missing', inplace=True)

bom_df.studio.isna().sum()
```

```
0
```

Creating new column for worldwide gross in the bom_df

```
# create new column for worldwide gross
bom_df['worldwide_gross'] = (
    bom_df.foreign_gross + bom_df.domestic_gross
)

bom_df.columns
```

```
Index(['title', 'studio', 'domestic_gross', 'foreign_gross', 'year',
       'worldwide_gross'],
      dtype='object')
```

The Box office mojo data is now clean

IMDB

The data is in form of a sqlite database

we first check the available tables

```
table_q = """
SELECT name
FROM sqlite_master
WHERE type='table';
"""

pd.read_sql(table_q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	name
0	movie_basics
1	directors
2	known_for
3	movie_akas
4	movie_ratings

	name
5	persons
6	principals
7	writers

1. movie_basics

```
mb_query = """
SELECT *
FROM movie_basics
"""

movie_basics = pd.read_sql(mb_query, conn)

movie_basics.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80	Comedy,Drama,Fantasy

This table contains the basic information about the movies eg title, and genre

```
movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

We notice missing data in some columns.

Lets start with the original_title column. Since there are few entries missing, we fill with the tag 'missing'

(a). original_title column

```
movie_basics.original_title.fillna('missing', inplace=True)

movie_basics.original_title.isna().sum()
```

```
0
```

(b). runtime_minutes column

For runtime_minutes, we can fill with the average

```
# get the mean runtime minutes
mean_runtime = movie_basics.runtime_minutes.mean()
movie_basics.runtime_minutes.fillna(mean_runtime, inplace=True)

movie_basics.runtime_minutes.isna().sum()
```

```
0
```

(c). genres column

For genres column we can try to get the missing genres in different datasets. But first we obtain the movies with missing genres

```
# movies with missing genres
missing_genres_df = movie_basics.loc[
    movie_basics['genres'].isna()
]

missing_genres_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
16	tt0187902	How Huang Fei-hong Rescued the Orphan from the...	How Huang Fei-hong Rescued the Orphan from the...	2011	86	None
22	tt0253093	Gangavataran	Gangavataran	2018	134	None
35	tt0306058	Second Coming	Second Coming	2012	95	None
40	tt0326592	The Overnight	The Overnight	2010	88	None
44	tt0330811	Regret Not Speaking	Regret Not Speaking	2011	86	None

We only need the title and the id

```
missing_genres_df = missing_genres_df.loc[
    :, ['movie_id', 'primary_title', 'original_title']
]

missing_genres_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	movie_id	primary_title	original_title
16	tt0187902	How Huang Fei-hong Rescued the Orphan from the...	How Huang Fei-hong Rescued the Orphan from the...
22	tt0253093	Gangavataran	Gangavataran
35	tt0306058	Second Coming	Second Coming
40	tt0326592	The Overnight	The Overnight
44	tt0330811	Regret Not Speaking	Regret Not Speaking

Lets start with the **Rotten tomatoes** data and check if it contains a genres column

```
rt_movie_info_df.columns
```

```
Index(['id', 'synopsis', 'rating', 'genre', 'director', 'writer',
       'theater_date', 'dvd_date', 'currency', 'box_office', 'runtime',
       'studio'],
      dtype='object')
```

```
rt_movie_info_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	synopsis	rating	genre	director	writer	theater_date	dvd_date	curren
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Sep 25, 2001	NaN
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012	Jan 1, 2013	$
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Apr 18, 2000	NaN
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994	Aug 27, 1997	NaN
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN	NaN	NaN

We can use the 'genre' column but there is no title column to compare to the data in missing_genres_df

Lets check **The Movie Db data**

```
tmdb_df.columns
```

```
Index(['Unnamed: 0', 'genre_ids', 'id', 'original_language', 'original_title',
       'popularity', 'release_date', 'title', 'vote_average', 'vote_count'],
      dtype='object')
```

```
tmdb_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	34	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	8
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	29	2010-03-26	How to Train Your Dragon	8
2	2	[12, 28, 878]	10138	en	Iron Man 2	29	2010-05-07	Iron Man 2	7
3	3	[16, 35, 10751]	862	en	Toy Story	28	1995-11-22	Toy Story	8
4	4	[28, 878, 12]	27205	en	Inception	28	2010-07-16	Inception	8

For this data, there is a title column to compare to, but the genres are id refferencing to a genres table which we dont have access to.

Finally **The Numbers** data

```
tn_budgets_df.columns
```

```
Index(['id', 'release_date', 'movie', 'production_budget', 'domestic_gross',
       'worldwide_gross', 'foreign_gross'],
      dtype='object')
```

There is no column we can use to get the genres

Since we cant get the genres from the other data sources, we can fill the entries with 'missing' tag

```
movie_basics.fillna('missing', inplace=True)

movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype
```

```
---  ------           --------------  -----
 0   movie_id         146144 non-null  object
 1   primary_title    146144 non-null  object
 2   original_title   146144 non-null  object
 3   start_year       146144 non-null  int64
 4   runtime_minutes  146144 non-null  float64
 5   genres           146144 non-null  object
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

There are no more missing values in the data.

Checking for duplicates in movie basics table

```
movie_basics.duplicated().sum()
```

```
0
```

No duplicates.

We completed cleaning the movie basics table.

2. movie_ratings

```
mr_query = """
SELECT *
FROM movie_ratings
"""

movie_ratings = pd.read_sql(mr_query, conn)
movie_ratings.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	movie_id	averagerating	numvotes
0	tt10356526	8	31
1	tt10384606	9	559
2	tt1042974	6	20
3	tt1043726	4	50352
4	tt1060240	6	21

```
movie_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   movie_id       73856 non-null  object
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
```

no missing values in the ratings table

Checking for duplicates

```
movie_ratings.duplicated().sum()
```

```
0
```

no duplicates in the movie ratings table

3. Genre ratings

```
movie_basics.genres.value_counts()
```

```
genres
Documentary                  32185
Drama                        21486
Comedy                        9177
missing                       5408
Horror                        4372
                             ...
Adventure,Music,Mystery          1
Documentary,Horror,Romance       1
Sport,Thriller                   1
Comedy,Sport,Western             1
Adventure,History,War            1
Name: count, Length: 1086, dtype: int64
```

Some genres are combined in one entry separated by a comma. We create a new df and separate each genre and ensure each has its own row.

we start by joining movie ratings and movie basics.

```
merged_ratings = movie_ratings.merge(movie_basics, on='movie_id', how='inner')
```

```
# create df as copy of ratings df
genre_df = merged_ratings.copy()

# split the genres
genre_df['genres'] = genre_df.genres.str.split(',')

# one genre in each row
genre_df = genre_df.explode('genres')

genre_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	movie_id	averagerating	numvotes	primary_title	original_title	start_year	runtime_minutes	genres
0	tt10356526	8	31	Laiye Je Yaarian	Laiye Je Yaarian	2019	117	Romance
1	tt10384606	9	559	Borderless	Borderless	2019	87	Documentary
2	tt1042974	6	20	Just Inès	Just Inès	2010	90	Drama
3	tt1043726	4	50352	The Legend of Hercules	The Legend of Hercules	2014	99	Action
3	tt1043726	4	50352	The Legend of Hercules	The Legend of Hercules	2014	99	Adventure

```
genre_df.genres.value_counts().index
```

```
Index(['Drama', 'Documentary', 'Comedy', 'Thriller', 'Horror', 'Action',
       'Romance', 'Crime', 'Adventure', 'Biography', 'Family', 'Mystery',
       'History', 'Sci-Fi', 'Fantasy', 'Music', 'Animation', 'Sport', 'War',
       'missing', 'Musical', 'News', 'Western', 'Reality-TV', 'Adult',
       'Game-Show', 'Short'],
      dtype='object', name='genres')
```

4. Directors

```
dir_query = """
SELECT  *
FROM
    directors d
JOIN
    persons p
USING(person_id)
"""

directors = pd.read_sql(dir_query, conn)
directors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 291171 entries, 0 to 291170
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   movie_id            291171 non-null  object 
 1   person_id           291171 non-null  object 
 2   primary_name        291171 non-null  object 
 3   birth_year          68608 non-null   float64
 4   death_year          1738 non-null    float64
 5   primary_profession  290187 non-null  object 
dtypes: float64(2), object(4)
memory usage: 13.3+ MB
```

missing values in the birth_year, death_year and primary proffession columns.

For birth_year and death_year we fill missing value with 0 to represent missing.

```
# handle missing birth year
directors['birth_year'].fillna(0, inplace=True)

# handle missing death year
directors['death_year'].fillna(0, inplace=True)

directors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 291171 entries, 0 to 291170
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   movie_id            291171 non-null  object 
 1   person_id           291171 non-null  object 
 2   primary_name        291171 non-null  object 
 3   birth_year          291171 non-null  float64
 4   death_year          291171 non-null  float64
 5   primary_profession  290187 non-null  object 
dtypes: float64(2), object(4)
memory usage: 13.3+ MB
```

Only primary profession has missing values. For this we fill with the tag '**director**', since it is contained in the directors table.

```
directors['primary_profession'].fillna('director', inplace=True)

directors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 291171 entries, 0 to 291170
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   movie_id            291171 non-null  object 
 1   person_id           291171 non-null  object 
 2   primary_name        291171 non-null  object 
 3   birth_year          291171 non-null  float64
 4   death_year          291171 non-null  float64
 5   primary_profession  291171 non-null  object 
dtypes: float64(2), object(4)
memory usage: 13.3+ MB
```

Now we check for duplicates

```
directors.duplicated().sum()
```

```
127638
```

The table contains many duplicates

```
directors.drop_duplicates(inplace=True)

directors.duplicated().sum()
```

```
0
```

5. Writers

```
wr_query = """
SELECT *
FROM
    writers w
JOIN
    persons p
USING(person_id)
"""

writers = pd.read_sql(wr_query, conn)
writers.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	movie_id	person_id	primary_name	birth_year	death_year	primary_profession
0	tt0285252	nm0899854	Tony Vitale	1,964	NaN	producer,director,writer
1	tt0438973	nm0175726	Steve Conrad	1,968	NaN	writer,producer,director
2	tt0438973	nm1802864	Sean Sorensen	NaN	NaN	producer,writer
3	tt0462036	nm1940585	Bill Haley	NaN	NaN	director,writer,producer
4	tt0835418	nm0310087	Peter Gaulke	NaN	NaN	writer,actor,director

```
writers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 255871 entries, 0 to 255870
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   movie_id            255871 non-null  object 
 1   person_id           255871 non-null  object 
 2   primary_name        255871 non-null  object 
 3   birth_year          52917 non-null   float64
 4   death_year          4078 non-null    float64
 5   primary_profession  255029 non-null  object 
dtypes: float64(2), object(4)
memory usage: 11.7+ MB
```

Same columns are missing in the writers table just as in directors table. We use the same method to handle the missing values but, for this table, we use the tag 'writer' for primary proffession.

```
# Filling the birth column
writers['birth_year'].fillna(0, inplace=True)

# filling the death column
writers['death_year'].fillna(0, inplace=True)

# filling the primary proffession column
writers['primary_profession'].fillna('writer', inplace=True)

writers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 255871 entries, 0 to 255870
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   movie_id            255871 non-null  object 
 1   person_id           255871 non-null  object 
 2   primary_name        255871 non-null  object 
 3   birth_year          255871 non-null  float64
 4   death_year          255871 non-null  float64
 5   primary_profession  255871 non-null  object 
dtypes: float64(2), object(4)
memory usage: 11.7+ MB
```

Lets check for duplicates

```
# check duplicates
writers.duplicated().sum()
```

```
77521
```

We drop the duplicates from the writers table

```
writers.drop_duplicates(inplace=True)

# check duplicates
writers.duplicated().sum()
```

```
0
```

Data Analysis

```
# suppressing scientific notation and adding commas for thousands separators
pd.options.display.float_format = '{:,.0f}'.format
```

This is to improve the readability of numerical data by suppressing scientific notation and add commas as thousands separators.

1. Best income generating studios

We use the Box Office Mojo data

```
bom_df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	title	studio	domestic_gross	foreign_gross	year	worldwide_gross
0	Toy Story 3	BV	415,000,000	652,000,000	2010	1,067,000,000
1	Alice in Wonderland (2010)	BV	334,200,000	691,300,000	2010	1,025,500,000
2	Harry Potter and the Deathly Hallows Part 1	WB	296,000,000	664,300,000	2010	960,300,000
3	Inception	WB	292,600,000	535,700,000	2010	828,300,000
4	Shrek Forever After	P/DW	238,700,000	513,900,000	2010	752,600,000

(a). summary statistics

```
bom_summary = bom_df.loc[
    :,
    ['domestic_gross', 'foreign_gross', 'worldwide_gross']
].describe()
bom_summary
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	domestic_gross	foreign_gross	worldwide_gross
count	3,387	3,387	3,387
mean	28,745,845	70,151,173	98,897,018
std	66,704,973	107,498,910	162,851,053
min	100	600	4,900
25%	122,500	8,000,000	18,700,000
50%	1,400,000	70,151,173	70,185,873
75%	28,745,845	70,151,173	73,251,173
max	936,700,000	960,500,000	1,518,900,000

The mean earnings for the movies are around:

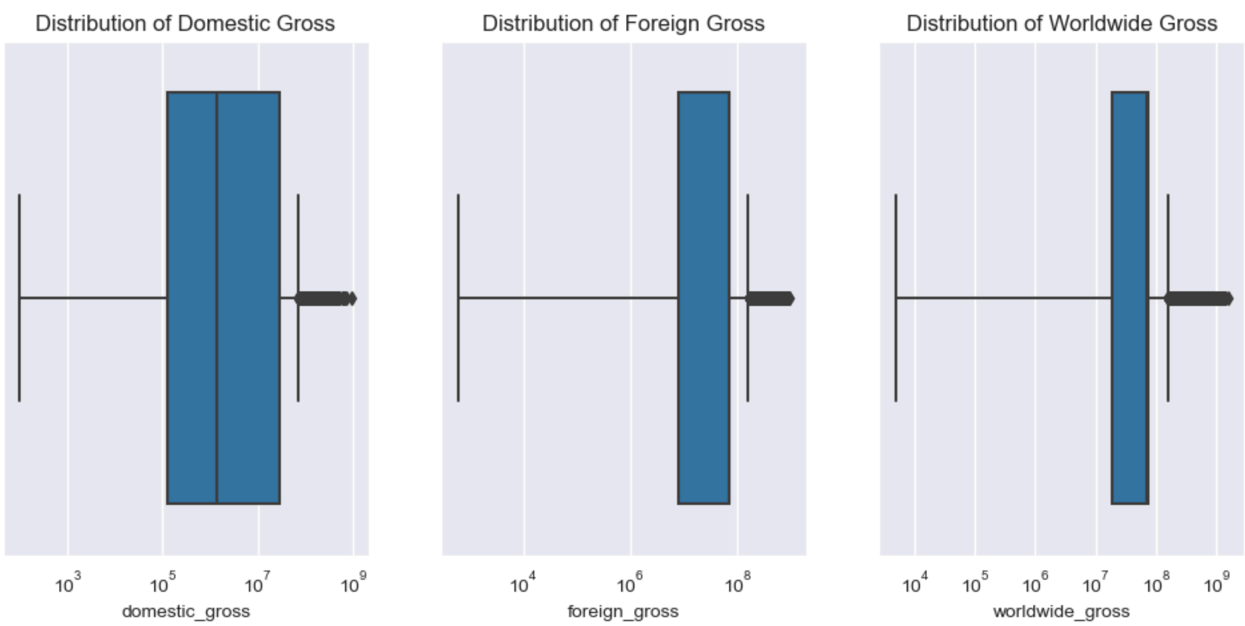
- $$$ 28M for Domestic gross
- $$$ 70M for Foreign gross
- $$$ 99M for worldwide gross

The earnings for the movies range from:

- $$$ 100 to $$$ 936M for Domestic gross
- $$$ 600 to $$$ 960M for Foreign gross
- $$$ 4k to $$$ 1B for worldwide gross

(b). Distribution of earnings

```
# creating the figure and axes
fig, axes = plt.subplots(ncols=3, figsize=(12, 5))

# set the style
sns.set_style('darkgrid')

# plot distribution for domestic gross
sns.boxplot(
    data=bom_df,
    x='domestic_gross',
    ax=axes[0]
)



# plot distribution for foreign gross
sns.boxplot(
    data=bom_df,
    x='foreign_gross',
    ax=axes[1]
)



# plot distribution for worldwide gross
sns.boxplot(
    data=bom_df,
    x='worldwide_gross',
    ax=axes[2]
)

# setting scale to log
axes[0].set_xscale('log')
axes[1].set_xscale('log')
axes[2].set_xscale('log')


# Labelling
axes[0].set_title('Distribution of Domestic Gross')
axes[1].set_title('Distribution of Foreign Gross')
axes[2].set_title('Distribution of Worldwide Gross');
```



we notice some outliers in the three categories. Mostly above 100M for all the groups.

Using cbook from matplotlib, we can get the exact values.

```
from matplotlib import cbook
```

```
# Domestic gross stats
domestic_stats = cbook.boxplot_stats(bom_df.worldwide_gross)

# Foreign gross stats
foreign_stats = cbook.boxplot_stats(bom_df.foreign_gross)

# Worldwide gross stats
worldwide_stats = cbook.boxplot_stats(bom_df.worldwide_gross)
```

We focus on the worldwide gross since its the overal earnings.

```
# worldwide gross stats without fliers
print('\nWorldwide Gross Stats:')
for key in worldwide_stats[0]:
    if key != 'fliers':
        print(key, worldwide_stats[0][key])
```

```
Worldwide Gross Stats:
mean 98897018.2935503
iqr 54551173.22656608
cilo 68714251.64081404
cihi 71657494.81231812
whishi 155000000.0
whislo 4900.0
q1 18700000.0
med 70185873.22656608
q3 73251173.22656608
```

From this stats we get more information and a precise range compared to the summary statistics.

- The range of worldwide earnings is between 4900 and 155M. Values outside this range are considered outliers.
- we also get the quartiles and confidence intervals

(c). Studios with highest Earnings

```
len(bom_df.studio.value_counts())
```

```
258
```

There are 257 Studios asociated with the movies. We get the studios with the highest earnings (worldwide gross)

```
# Get total earnings for each studio
total_studio_earnings = bom_df.groupby(
    'studio'
)['worldwide_gross'].sum()

# sorting the studios according to earnings
total_studio_earnings.sort_values(
    ascending=False,
    inplace=True
)

# get top 10 studios according to earnings
top_10_studios = total_studio_earnings[:10].reset_index()
top_10_studios
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

</style>

	studio	worldwide_gross
0	BV	44,213,702,543
1	WB	31,412,801,305
2	Fox	31,020,580,426
3	Uni.	29,967,617,711
4	Sony	22,714,388,634
5	Par.	19,924,826,363
6	WB (NL)	10,334,796,287
7	LGF	9,251,733,541
8	IFC	6,693,197,233
9	Magn.	5,972,688,551

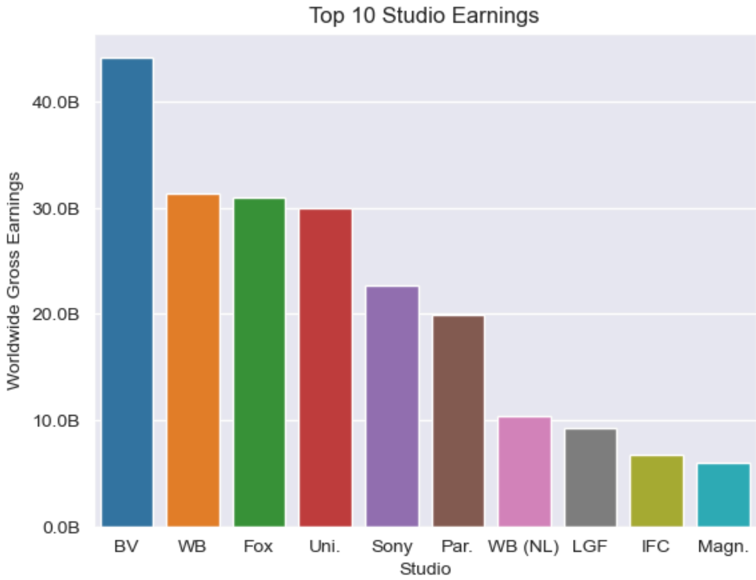
Some of the top 10 earning studios include:

- BV studios
- Warner Bros studios
- Fox Studios
- universal studios
- Sony
- Paramount
- Warner Bros. (New Line)
- Lionsgate Films (LGF)
- Independent Film Channel (IFC)
- Magnolia Pictures

Plot of the Top 10 Studio Earnings

All the top 10 studios have earnings of more than a billion. We first fix the scale to display in billions.

```
# bar plot of studio earnings
barplot = sns.barplot(
    data=top_10_studios,
    x='studio',
    y='worldwide_gross'
)
# labelling
barplot.set_title('Top 10 Studio Earnings')
barplot.set_xlabel('Studio')
barplot.set_ylabel('Worldwide Gross Earnings')

# format y-axis to show in billions
def billions(x, pos):
    return '%1.1fB' % (x * 1e-9)

formatter = FuncFormatter(billions)
barplot.yaxis.set_major_formatter(formatter);
```



(d). Distribution of earnings over the years

```
# Creating grouped df of earnings by the years
yearly_earnings = bom_df.groupby(
    'year'
)['worldwide_gross'].sum().reset_index()
```

```
# Plotting the disribution over the years
ax = sns.lineplot(data=yearly_earnings, x='year', y='worldwide_gross')

# labelling
ax.set_title('Distribution of earnings over the years')
ax.set_xlabel('Year')
ax.set_ylabel('Worldwide Gross Earnings')

# convert axes to billions
ax.yaxis.set_major_formatter(formatter)
```

```
c:\Users\mutis\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and
will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\mutis\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and
will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Over time, there has been a general upward trend in movie earnings. The biggest income was reported in 2016, and since then, it has decreased. The overall trend is still positive despite this recent decrease, suggesting that movie revenues are expected to rise in the long run.

2. Best Ratings

```
# enabling decimals
pd.options.display.float_format = None
```

We use the IMDB data to get some insights based on ratings.

- Best rated genres
- Best rated Writers
- Best rated Directors

(a). Best rated genres

We use the genres df

We start by grouping the data by the genre and getting the average rating

```
# grouping data by genre
genre_ratings = genre_df.groupby('genres')[['averagerating', 'numvotes']].mean()

# round rating to 1 decimal point
genre_ratings['averagerating'] = genre_ratings['averagerating'].round(1)

# converting numvotes to integer
genre_ratings['numvotes'] = genre_ratings['numvotes'].astype(int)

# sorting the values
genre_ratings.sort_values(
    by=['averagerating', 'numvotes'],
    ascending=False,
    inplace=True
)

# reset index
genre_ratings = genre_ratings.reset_index()

# get least rated genre
least_rated = genre_ratings.loc[
    genre_ratings.averagerating == genre_ratings.averagerating.min(),
    'genres'
]

# get best rated genre
best_rated = genre_ratings.loc[
    genre_ratings.averagerating == genre_ratings.averagerating.max(),
    'genres'
]

print('Best Rated:', best_rated.values[0])
print('Least Rated:', least_rated.values[0])

genre_ratings
```

```
Best Rated: Short
Least Rated: Adult
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

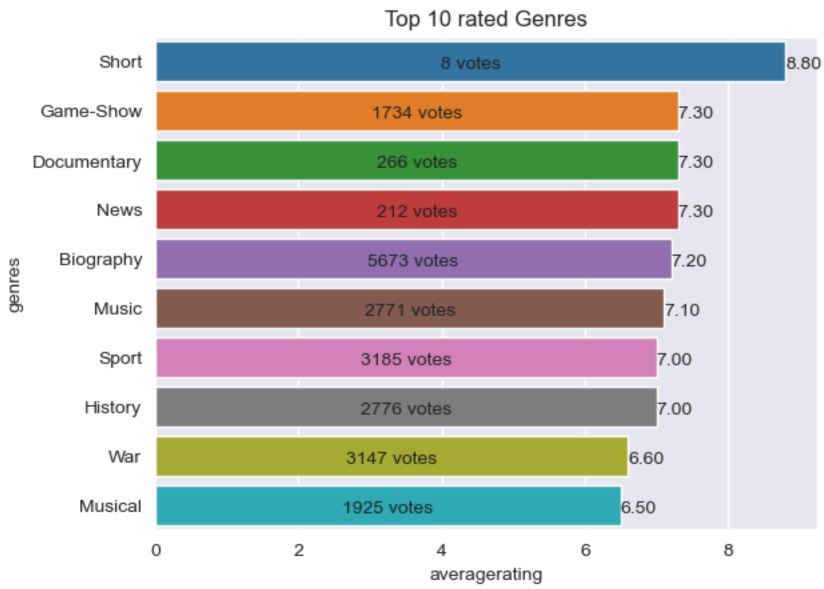
	genres	averagerating	numvotes
0	Short	8.8	8
1	Game-Show	7.3	1734
2	Documentary	7.3	266
3	News	7.3	212
4	Biography	7.2	5673
5	Music	7.1	2771
6	Sport	7.0	3185
7	History	7.0	2776
8	War	6.6	3147
9	Musical	6.5	1925
10	Reality-TV	6.5	27
11	missing	6.5	24
12	Drama	6.4	3883
13	Family	6.4	2531
14	Adventure	6.2	22067
15	Animation	6.2	8808
16	Crime	6.1	8594
17	Romance	6.1	4084
18	Comedy	6.0	4297
19	Fantasy	5.9	12387
20	Western	5.9	8758
21	Mystery	5.9	8113
22	Action	5.8	14476
23	Thriller	5.6	5860
24	Sci-Fi	5.5	19474
25	Horror	5.0	3112
26	Adult	3.8	54

Short films are the best rated with Adult films being the least favourite.

Visualizing the top 10 genres

```
# creating figure and axis
fig, ax = plt.subplots()

# plotting the bar plot
sns.barplot(
    data=genre_ratings[:10],
    y='genres',
    x='averagerating',
    ax=ax,
    orient='h'
)
i = 0
votes = genre_ratings[:10].numvotes
# Labelling the number of votes and including the ratings
for p in ax.patches:
    # labelling the ratings
    ax.annotate(
        f'{p.get_width():.2f}',
        (p.get_width() + .25, p.get_y() + p.get_height()),
        ha='center', va='center',
        xytext=(0, 9),
        textcoords='offset points'
```

```
        )
        # labelling the number of votes
        ax.annotate(
            f'{votes[i]} votes',
            (p.get_width() / 2., p.get_y() + p.get_height()),
            ha='center', va='center',
            xytext=(0, 9),
            textcoords='offset points'
        )
        i += 1

    # labelling the title
    ax.set_title('Top 10 rated Genres');
```



The top ten genres are shown in the bar graph according to user ratings. The genres are represented by the x-axis, while the ratings are displayed on the y-axis. The quantity of votes a genre received helps to further classify genres with similar ratings, resulting in a more accurate ranking. The height of the bars reflects the average rating for each genre, and each bar is color-coded to help differentiate between them.

(b). Best rated Directors

The best directors are those whose movies are highly rated.

We first merge the ratings and directors tables

```
director_ratings = directors.merge(
    movie_ratings,
    on='movie_id',
    how='inner'
)
director_ratings.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	movie_id	person_id	primary_name	birth_year	death_year	primary_profession	averagerating
0	tt0285252	nm0899854	Tony Vitale	1964.0	0.0	producer,director,writer	3.9
1	tt0462036	nm1940585	Bill Haley	0.0	0.0	director,writer,producer	5.5
2	tt0835418	nm0151540	Jay Chandrasekhar	1968.0	0.0	director,actor,writer	5.0

	movie_id	person_id	primary_name	birth_year	death_year	primary_profession	averagerating
3	tt0878654	nm0089502	Albert Pyun	1954.0	0.0	director,writer,producer	5.8
4	tt0878654	nm2291498	Joe Baile	0.0	0.0	producer,director,camera_department	5.8

From the data, some directors are deceased. We filter the data to include only directors who are alive.

```
director_ratings = director_ratings.loc[
    director_ratings.death_year == 0
]

director_ratings.death_year.value_counts()
```

```
death_year
0.0    85331
Name: count, dtype: int64
```

We first get the number of movies each director has featured in.

```
director_movie_count = director_ratings.groupby(
    ['person_id']
).size().sort_values(ascending=False)

# resetting the index and naming count column
director_movie_count = director_movie_count.reset_index(name='moviecount')

print('Highest movie count:', director_movie_count.moviecount.iloc[0])
print('Lowest movie count:', director_movie_count.moviecount.iloc[-1])

director_movie_count.head()
```

```
Highest movie count: 39
Lowest movie count: 1
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	person_id	moviecount
0	nm5954636	39
1	nm2551464	37
2	nm3583561	34
3	nm4341114	31
4	nm2563700	30

The data includes directors with varying levels of experience, ranging from those who have directed only one film to those with over 200 movies. To categorize their experience, we create a new column with the following classifications:

- Beginner: 1-3 movies
- Intermediate: 4-5 movies
- Experienced: 6-15 movies
- Highly Experienced: 16-20 movies
- Veteran: 20+ movies

```
# function to categorize the experience
def set_experience(val):
```

```
    if val <= 3:
        return 'beginner'
    elif val > 3 and val <= 5:
        return 'intermediate'
    elif val > 5 and val <= 15:
        return 'experienced'
    elif val > 15 and val <=20:
        return 'highly experienced'
    else:
        return 'veteran'
```

Creating the experience column

```
director_movie_count['experience'] = director_movie_count.moviecount.map(
    lambda x: set_experience(x)
)
director_movie_count.head()
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

Cell In[155], line 1
----> 1 director_movie_count['experience'] = director_movie_count.moviecount.map(
      2     lambda x: set_experience(x)
      3 )
      4 director_movie_count.head()


NameError: name 'director_movie_count' is not defined
```

Next, we group the data by directors and calculate the average ratings of their movies as well as the average number of votes. This allows us to analyze the performance of each director based on the reception and popularity of their films.

```
# grouping by the directiors
ratings_by_directors = director_ratings.groupby(
    ['person_id', 'primary_name']
)[
    ['averagerating', 'numvotes']
].mean()

# reseting the index
ratings_by_directors.reset_index(inplace=True)


ratings_by_directors.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	person_id	primary_name	averagerating	numvotes
0	nm0000095	Woody Allen	6.700000	106068.375000
1	nm0000108	Luc Besson	6.350000	113490.500000
2	nm0000110	Kenneth Branagh	6.928571	160110.714286
3	nm0000118	John Carpenter	5.600000	38287.000000
4	nm0000123	George Clooney	6.266667	118783.000000

The average number of votes and the average rating of each director's film are then determined by grouping the data by director. This makes it possible for us to evaluate each director's work in light of the reviews and box office success of their respective projects. Next, we transform the average number of votes to integer numbers and round the average ratings to one decimal place.

```
# rounding the ratings column
ratings_by_directors.averagerating = ratings_by_directors.averagerating.round(1)

# convert votes to integers
ratings_by_directors.numvotes = ratings_by_directors.numvotes.astype(int)

# sorting
ratings_by_directors.sort_values(
    by=['averagerating', 'numvotes'],
    ascending=False,
    inplace=True
)

ratings_by_directors
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	person_id	primary_name	averagerating	numvotes
31643	nm3388005	Stephen Peek	10.0	20
51014	nm7223265	Loreto Di Cesare	10.0	8
52109	nm7633303	Lindsay Thompson	10.0	7
50148	nm6925060	Tristan David Luciotti	10.0	6
54795	nm8791543	Emre Oran	10.0	6
...	...	...	...	...
54832	nm8809512	Erik Alarik	1.0	8
23645	nm2277264	Koki Ebata	1.0	7
40737	nm4728793	Takeo Urakami	1.0	7
28377	nm2947112	Shinju Funabiki	1.0	6
44133	nm5328929	Samuele Dalò	1.0	5

56784 rows × 4 columns

It is noted that the average rating for the best directors is 10, whereas the lowest have an average rating of 1. We also look at the total number of films that each director has starred in, since this has a big impact on the director's grade.

```
# merging the ratings to include movies count
ratings_by_directors = ratings_by_directors.merge(
    director_movie_count,
    on='person_id',
    how='inner'
)

ratings_by_directors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56784 entries, 0 to 56783
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   person_id     56784 non-null  object
```

```
 1   primary_name   56784 non-null  object
 2   averagerating  56784 non-null  float64
 3   numvotes       56784 non-null  int32
 4   moviecount     56784 non-null  int64
 5   experience     56784 non-null  object
dtypes: float64(1), int32(1), int64(1), object(3)
memory usage: 2.4+ MB
```

```
# change data type of moviecount column
ratings_by_directors.moviecount = ratings_by_directors.moviecount.astype('Int32')
# include sorting by movie count
ratings_by_directors.sort_values(
    by=['averagerating', 'moviecount', 'numvotes'],
    ascending=False,
    inplace=True
)

ratings_by_directors
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

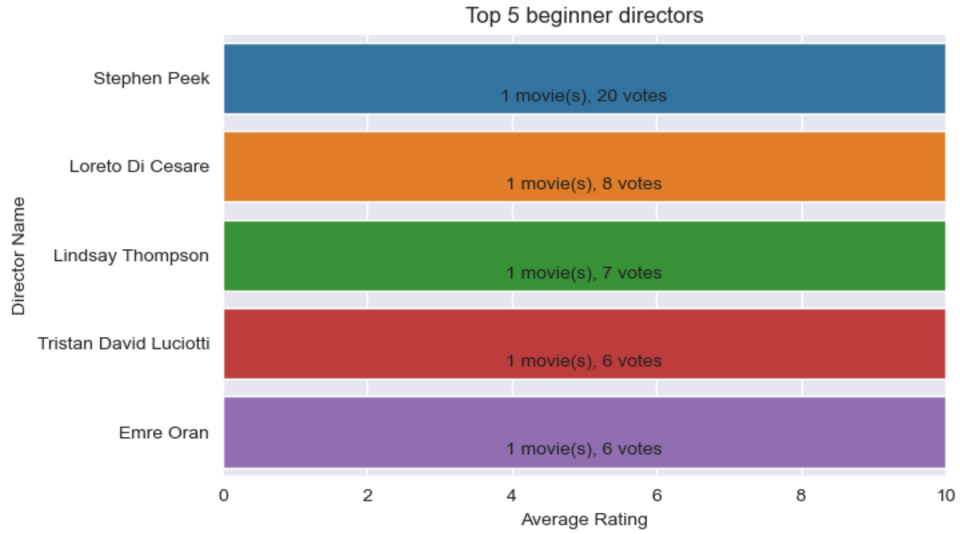
	person_id	primary_name	averagerating	numvotes	moviecount	experience
0	nm3388005	Stephen Peek	10.0	20	1	beginner
1	nm7223265	Loreto Di Cesare	10.0	8	1	beginner
2	nm7633303	Lindsay Thompson	10.0	7	1	beginner
3	nm6925060	Tristan David Luciotti	10.0	6	1	beginner
4	nm8791543	Emre Oran	10.0	6	1	beginner
...	...	...	...	...	...	...
56779	nm8809512	Erik Alarik	1.0	8	1	beginner
56780	nm2277264	Koki Ebata	1.0	7	1	beginner
56781	nm4728793	Takeo Urakami	1.0	7	1	beginner
56782	nm2947112	Shinju Funabiki	1.0	6	1	beginner
56783	nm5328929	Samuele Dalò	1.0	5	1	beginner

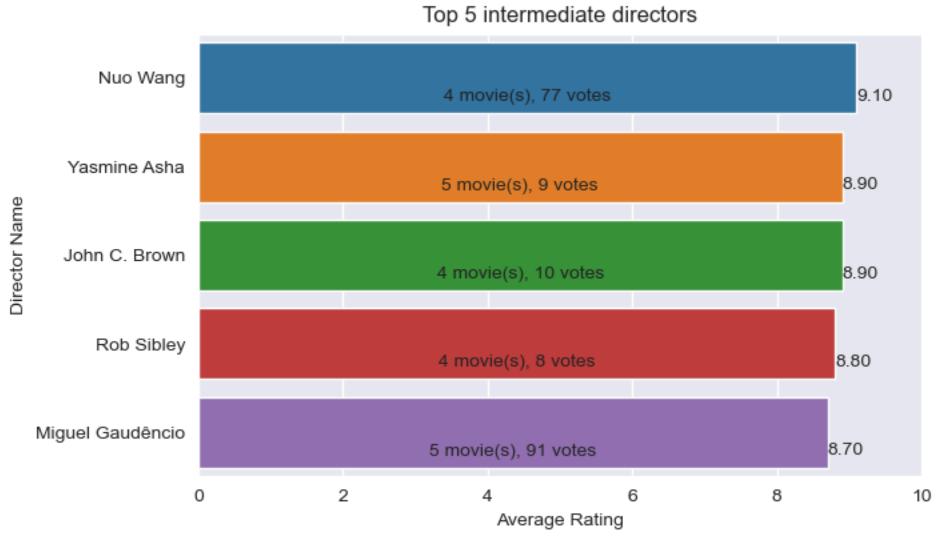
56784 rows × 6 columns

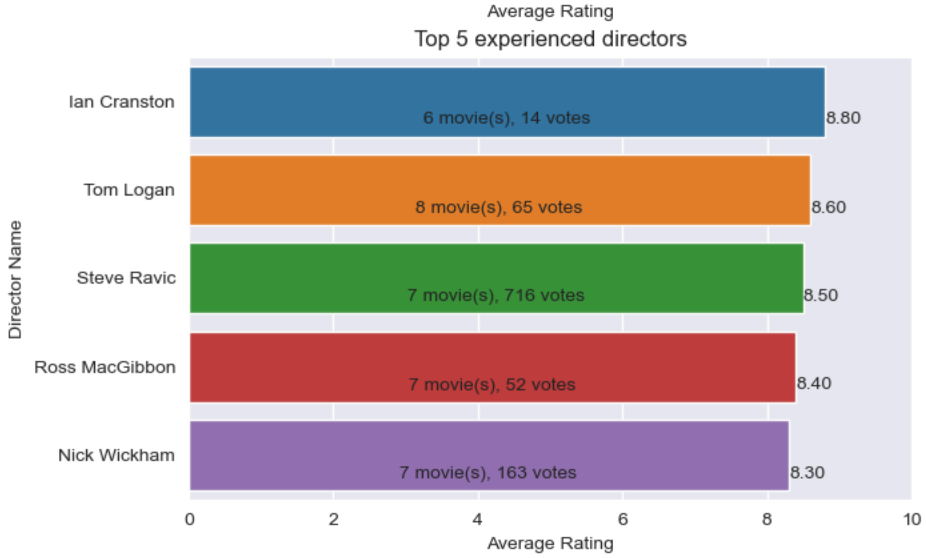
We can now group the directors by experience and compare them.

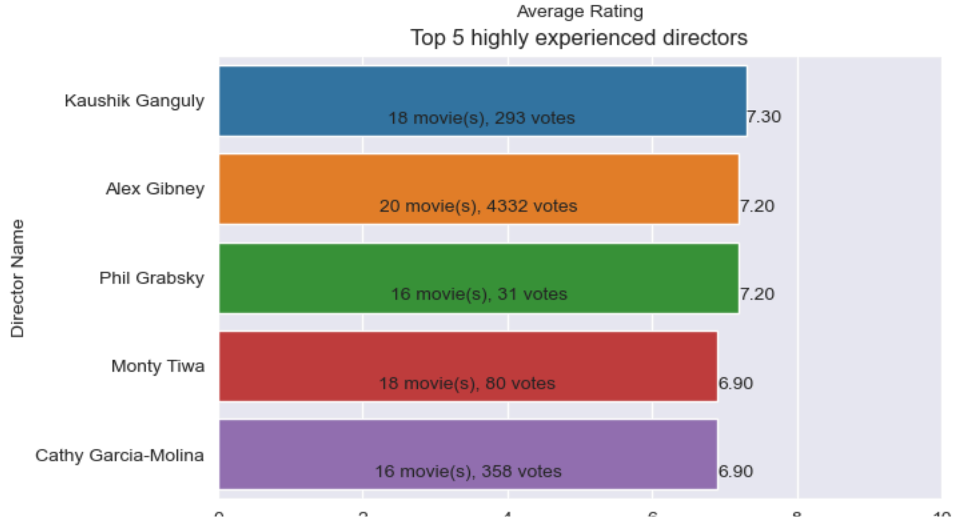
```
# top 5 beginner ditectors
top_5_beginner_directors = ratings_by_directors.loc[
    ratings_by_directors.experience == 'beginner'
][:5]

# top 5 intermediate ditectors
top_5_intermediate_directors = ratings_by_directors.loc[
    ratings_by_directors.experience == 'intermediate'
][:5]

# top 5 experienced ditectors
top_5_experienced_directors = ratings_by_directors.loc[
    ratings_by_directors.experience == 'experienced'
][:5]

# top 5 highly experienced ditectors
top_5_highly_directors = ratings_by_directors.loc[
    ratings_by_directors.experience == 'highly experienced'
```

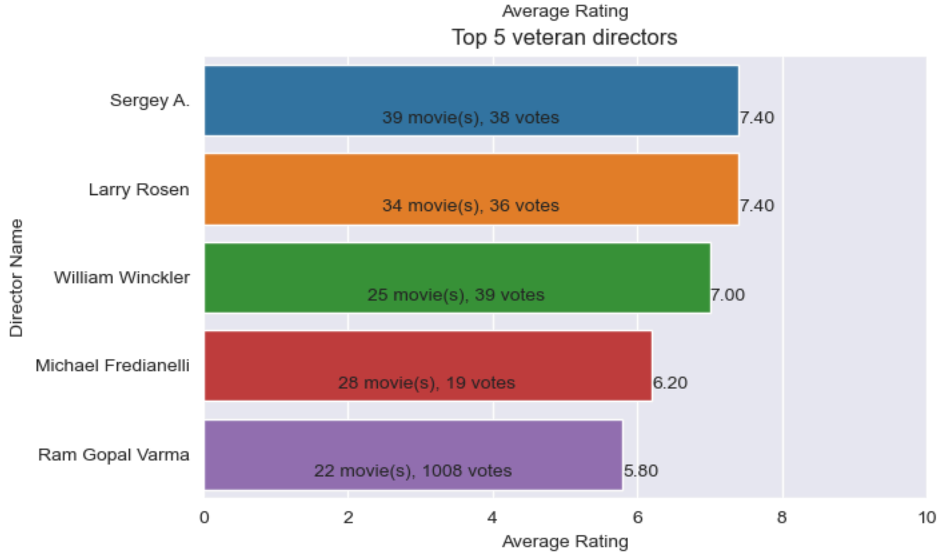
```
][:5]

# top 5 veteran ditectors
top_5_veteran_directors= ratings_by_directors.loc[
    ratings_by_directors.experience == 'veteran'
][:5]



# create figure and axes
fig, axes = plt.subplots(nrows=5, figsize=(7, 25))

# list of all dataframes
data_list = [
    top_5_beginner_directors,
    top_5_intermediate_directors,
    top_5_experienced_directors,
    top_5_highly_directors,
    top_5_veteran_directors
]

# plotting the data
for i, data in enumerate(data_list):
    sns.barplot(
        data=data,
        x='averagerating',
        y='primary_name',
        ax=axes[i]
    )
    # including the number of votes and movie count
    n = 0
    for p in axes[i].patches:
        # labelling the rating
        axes[i].annotate(
            f'{p.get_width():.2f}',
            (p.get_width() + .25, p.get_y() + p.get_height()),
            ha='center', va='center',
            xytext=(0, 9),
            textcoords='offset points'
        )
        # labelling the number of votes
        axes[i].annotate(
            f'{data.moviecount.iloc[n]} movie(s), {data.numvotes.iloc[n]} votes',
            (p.get_width() / 2., p.get_y() + p.get_height()),
            ha='center', va='center',
            xytext=(0, 9),
            textcoords='offset points'
        )
        n += 1

    # labeling axes
    axes[i].set_title(f'Top 5 {data.experience.iloc[i]} directors')
    axes[i].set_xlabel('Average Rating')
    axes[i].set_ylabel('Director Name')
    axes[i].set_xlim(0, 10)
```


Top 5 beginner directors
Director Name
Stephen Peek
1 movie(s), 20 votes
Loreto Di Cesare
1 movie(s), 8 votes
Lindsay Thompson
1 movie(s), 7 votes
Tristan David Luciotti
1 movie(s), 6 votes
Emre Oran
1 movie(s), 6 votes
0
2
4
6
8
10
Average Rating


Top 5 intermediate directors
Director Name
Nuo Wang
4 movie(s), 77 votes
9.10
Yasmine Asha
5 movie(s), 9 votes
8.90
John C. Brown
4 movie(s), 10 votes
8.90
Rob Sibley
4 movie(s), 8 votes
8.80
Miguel Gaudêncio
5 movie(s), 91 votes
8.70
0
2
4
6
8
10
Average Rating


Top 5 experienced directors
Director Name
Ian Cranston
6 movie(s), 14 votes
8.80
Tom Logan
8 movie(s), 65 votes
8.60
Steve Ravic
7 movie(s), 716 votes
8.50
Ross MacGibbon
7 movie(s), 52 votes
8.40
Nick Wickham
7 movie(s), 163 votes
8.30
0
2
4
6
8
10
Average Rating


Top 5 highly experienced directors
Director Name
Kaushik Ganguly
18 movie(s), 293 votes
7.30
Alex Gibney
20 movie(s), 4332 votes
7.20
Phil Grabsky
16 movie(s), 31 votes
7.20
Monty Tiwa
18 movie(s), 80 votes
6.90
Cathy Garcia-Molina
16 movie(s), 358 votes
6.90



Above is the list of graphs of the top directors based on their movie ratings, also categorized by their experience levels.

Various factors can vary based on the directors' experience, such as:

- salary
- quality of movies
- audience reception
- production budgets

Beginner directors might have lower salaries and fewer resources, but as their experience grows, they tend to produce higher quality films, receive better audience ratings, and command higher salaries. Veteran directors, with extensive experience, often have established reputations, allowing them to secure larger budgets and attract top talent, further enhancing the quality and success of their movies.

Interestingly, beginners sometimes tend to have higher ratings. This can be attributed to the effect of having directed only a few movies, which may result in their ratings being skewed by a small sample size. A single highly-rated movie can disproportionately elevate their average rating. As directors gain more experience and their body of work grows, their ratings might normalize and provide a more comprehensive view of their overall performance.

These graphs provide a comprehensive view of how directors' experience levels correlate with their average movie ratings, showcasing the impact of experience on their career achievements and industry recognition.

(c). Best Rated writers

Just like the directors we use the same method to get the top rated writers.

i. Merge writers and ratings table

```
# merging writers and movie ratings
writer_ratings = writers.merge(
    movie_ratings,
    on='movie_id',
    how='inner'
)
writer_ratings.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	movie_id	person_id	primary_name	birth_year	death_year	primary_profession	averagerating	numvotes
0	tt0285252	nm0899854	Tony Vitale	1964.0	0.0	producer,director,writer	3.9	219

	movie_id	person_id	primary_name	birth_year	death_year	primary_profession	averagerating	numvotes
1	tt0462036	nm1940585	Bill Haley	0.0	0.0	director,writer,producer	5.5	18
2	tt0835418	nm0310087	Peter Gaulke	0.0	0.0	writer,actor,director	5.0	8147
3	tt0835418	nm0841532	Gerry Swallow	0.0	0.0	writer,actor,miscellaneous	5.0	8147
4	tt0878654	nm0284943	Randall Fontana	0.0	0.0	writer,director,actor	5.8	875

ii. Remove dead writers

```
writer_ratings = writer_ratings.loc[
    writer_ratings.death_year == 0
]

# check the death year collumn
writer_ratings.death_year.value_counts()
```

```
death_year
0.0    109319
Name: count, dtype: int64
```

iii. Get movies count for each writer

Getting the number of movies each writer has written.

```
writer_movie_count = writer_ratings.groupby(
    ['person_id']
).size().sort_values(ascending=False)

# resetting the index and naming count column
writer_movie_count = writer_movie_count.reset_index(name='moviecount')

# Creating the experience column
writer_movie_count['experience'] = writer_movie_count.moviecount.map(
    lambda x: set_experience(x)
)

print('Highest movie count:', writer_movie_count.moviecount.iloc[0])
print('Lowest movie count:', writer_movie_count.moviecount.iloc[-1])

writer_movie_count.head()
```

```
Highest movie count: 40
Lowest movie count: 1
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	person_id	moviecount	experience
0	nm5954636	40	veteran
1	nm3057599	32	veteran
2	nm3583561	32	veteran
3	nm0893128	32	veteran
4	nm0598531	32	veteran

iv. Getting average ratings of each writer

```
# grouping by the writers
ratings_by_writers = writer_ratings.groupby(
    ['person_id', 'primary_name']
)[
    ['averagerating', 'numvotes']
].mean()

# reseting the index
ratings_by_writers.reset_index(inplace=True)
```

```
ratings_by_writers.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	person_id	primary_name	averagerating	numvotes
0	nm0000092	John Cleese	7.450000	89365.000
1	nm0000095	Woody Allen	6.700000	106068.375
2	nm0000101	Dan Aykroyd	5.200000	186788.000
3	nm0000108	Luc Besson	5.905556	87079.500
4	nm0000116	James Cameron	6.950000	161411.000

we then round the ratings to one decimal point and convert the votes to integer values.

```
# rounding the ratings column
ratings_by_writers.averagerating = ratings_by_writers.averagerating.round(
    1)

# convert votes to integers
ratings_by_writers.numvotes = ratings_by_writers.numvotes.astype(int)

# sorting
ratings_by_writers.sort_values(
    by=['averagerating', 'numvotes'],
    ascending=False,
    inplace=True
)

ratings_by_writers
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	person_id	primary_name	averagerating	numvotes
63710	nm6680574	Brian Baucum	10.0	8
66116	nm7223265	Loreto Di Cesare	10.0	8
67762	nm7633303	Lindsay Thompson	10.0	7

	person_id	primary_name	averagerating	numvotes
71714	nm8791543	Emre Oran	10.0	6
15236	nm10616933	Ivana Diniz	10.0	5
...	...	...	...	...
71763	nm8809512	Erik Alarik	1.0	8
34829	nm2947112	Shinju Funabiki	1.0	6
60173	nm6008960	Eva Toulová	1.0	5
74437	nm9854007	Giueppe di Giorgio	1.0	5
74438	nm9854008	Roberto Attolini	1.0	5

74705 rows × 4 columns

We then include the movie count and experience by merging with the movie count df.

```
# merging the ratings to include movies count
ratings_by_writers = ratings_by_writers.merge(
    writer_movie_count,
    on='person_id',
    how='inner'
)

ratings_by_writers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74705 entries, 0 to 74704
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   person_id      74705 non-null  object 
 1   primary_name   74705 non-null  object 
 2   averagerating  74705 non-null  float64
 3   numvotes       74705 non-null  int32  
 4   moviecount     74705 non-null  int64  
 5   experience     74705 non-null  object 
dtypes: float64(1), int32(1), int64(1), object(3)
memory usage: 3.1+ MB
```

We then sort the records according to the rating, then movie count and finally the number of votes.

```
# change data type of moviecount column
ratings_by_writers.moviecount = ratings_by_writers.moviecount.astype(
    'Int32')
# include sorting by movie count
ratings_by_writers.sort_values(
    by=['averagerating', 'moviecount', 'numvotes'],
    ascending=False,
    inplace=True
)

ratings_by_writers
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	person_id	primary_name	averagerating	numvotes	moviecount	experience
0	nm6680574	Brian Baucum	10.0	8	1	beginner
1	nm7223265	Loreto Di Cesare	10.0	8	1	beginner

	person_id	primary_name	averagerating	numvotes	moviecount	experience
2	nm7633303	Lindsay Thompson	10.0	7	1	beginner
3	nm8791543	Emre Oran	10.0	6	1	beginner
4	nm10616933	Ivana Diniz	10.0	5	1	beginner
...	...	...	...	...	...	...
74700	nm8809512	Erik Alarik	1.0	8	1	beginner
74701	nm2947112	Shinju Funabiki	1.0	6	1	beginner
74702	nm6008960	Eva Toulová	1.0	5	1	beginner
74703	nm9854007	Giueppe di Giorgio	1.0	5	1	beginner
74704	nm9854008	Roberto Attolini	1.0	5	1	beginner

74705 rows × 6 columns

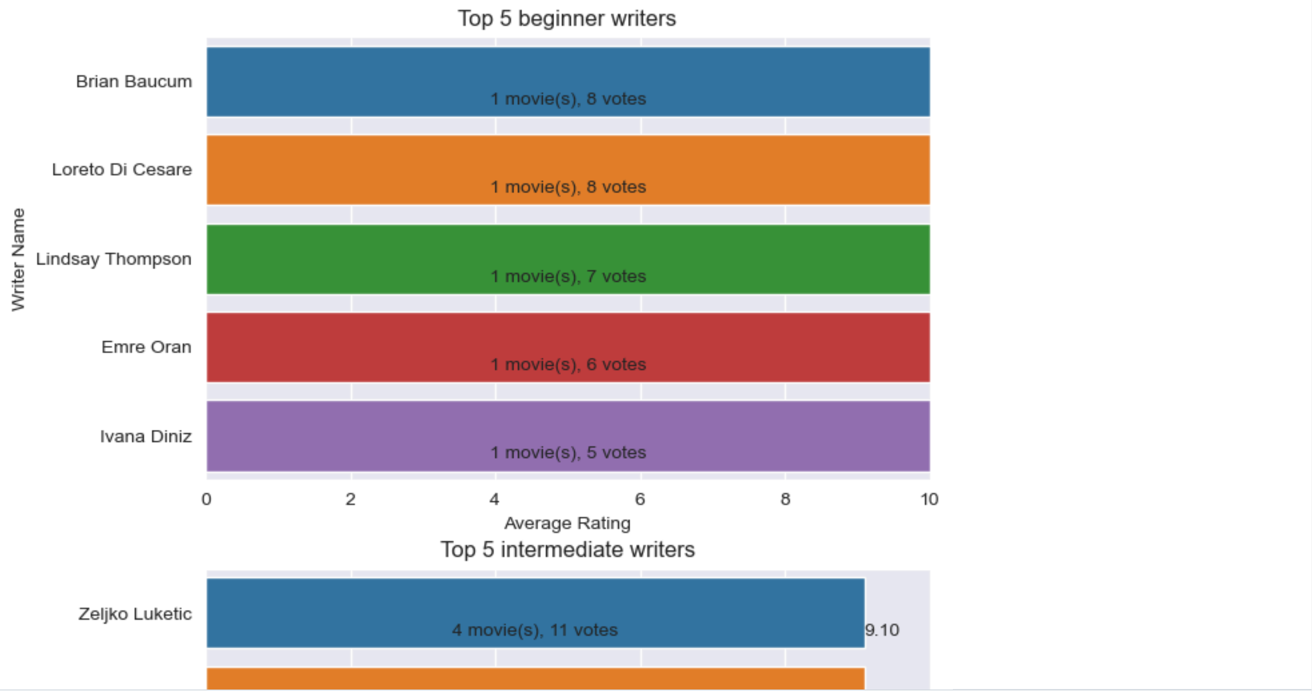
v. Grouping according to experience

```
# top 5 beginner ditectors
top_5_beginner_writers = ratings_by_writers.loc[
    ratings_by_writers.experience == 'beginner'
][:5]

# top 5 intermediate ditectors
top_5_intermediate_writers = ratings_by_writers.loc[
    ratings_by_writers.experience == 'intermediate'
][:5]

# top 5 experienced ditectors
top_5_experienced_writers = ratings_by_writers.loc[
    ratings_by_writers.experience == 'experienced'
][:5]

# top 5 highly experienced ditectors
top_5_highly_writers = ratings_by_writers.loc[
    ratings_by_writers.experience == 'highly experienced'
][:5]

# top 5 veteran ditectors
top_5_veteran_writers = ratings_by_writers.loc[
    ratings_by_writers.experience == 'veteran'
][:5]
```

vi. Plotting the Top Writers

```
# create figure and axes
fig, axes = plt.subplots(nrows=5, figsize=(7, 25))

# list of all dataframes
data_list = [
    top_5_beginner_writers,
    top_5_intermediate_writers,
    top_5_experienced_writers,
    top_5_highly_writers,
    top_5_veteran_writers
]

# plotting the data
for i, data in enumerate(data_list):
    sns.barplot(
        data=data,
        x='averagerating',
        y='primary_name',
        ax=axes[i]
    )
    # including the number of votes and movie count
    n = 0
    for p in axes[i].patches:
        # labelling the rating
        axes[i].annotate(
            f'{p.get_width():.2f}',
            (p.get_width() + .25, p.get_y() + p.get_height()),
```

```
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points'
            )
            # labelling the number of votes
            axes[i].annotate(
                f'{data.moviecount.iloc[n]} movie(s), {data.numvotes.iloc[n]} votes',
                (p.get_width() / 2., p.get_y() + p.get_height()),
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points'
            )
            n += 1

        # labeling axes
        axes[i].set_title(f'Top 5 {data.experience.iloc[i]} writers')
        axes[i].set_xlabel('Average Rating')
        axes[i].set_ylabel('Writer Name')
        axes[i].set_xlim(0, 10)
```



Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Languages

- **Jupyter Notebook** 100.0%