8a6ec703-2532-43eb-90a5-0e65d9d612d9

December 26, 2024

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 3

Great, Cooper! You've done a great job on all the comments and now your project has been accepted.

Thank you for your work and I wish you success in the following projects!

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 2

You did a good job with the comments, but I still have a few more comments, they are marked with version 2.0 and red. Please fix them and I will accept your project)

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 2

Also, please, when submitting a new version of the project, do not delete the reviewer's comments from the previous version.

Also, before sending a new version, please click `Kernel - > Restart & Run all` to make sure that your code is working

Hello Cooper

My name is Dima, and I will be reviewing your project.

You will find my comments in coloured cells marked as 'Reviewer's comment'. The cell colour will vary based on the contents - I am explaining it further below.

Note: Please do not remove or change my comments - they will help me in my future reviews and will make the process smoother for both of us.

You are also very welcome to leave your comments / describe the corrections you've done / ask me questions, marking them with a different colour. You can use the example below:

<div class="alert alert-info"; style="border-left: 7px solid blue"> Student's comment

0.1 Basic Python - Project

Review summary

Cooper, thanks for submitting the project. You've done a very good job and I enjoyed reviewing it.

- You completed all the tasks.
- Your code was optimal and easy to read.
- You wrote your own functions.

There are only a few critical comments that need to be corrected. You will find them in the red-colored cells in relevant sections. If you have any questions please write them when you return your project.

I'll be looking forward to getting your updated notebook.

0.2 Introduction

In this project, you will work with data from the entertainment industry. You will study a dataset with records on movies and shows. The research will focus on the "Golden Age" of television, which began in 1999 with the release of *The Sopranos* and is still ongoing.

The aim of this project is to investigate how the number of votes a title receives impacts its ratings. The assumption is that highly-rated shows (we will focus on TV shows, ignoring movies) released during the "Golden Age" of television also have the most votes.

0.2.1 Stages

Data on movies and shows is stored in the `/datasets/movies_and_shows.csv` file. There is no information about the quality of the data, so you will need to explore it before doing the analysis.

First, you'll evaluate the quality of the data and see whether its issues are significant. Then, during data preprocessing, you will try to account for the most critical problems.

Your project will consist of three stages: 1. Data overview 2. Data preprocessing 3. Data analysis

0.3 Stage 1. Data overview

Open and explore the data.

You'll need `pandas`, so import it.

[1]:
```
# importing pandas
import pandas as pd
```

Read the `movies_and_shows.csv` file from the `datasets` folder and save it in the `df` variable:

[2]:
```
# reading the files and storing them to df
df = pd.read_csv('/datasets/movies_and_shows.csv')
```

Print the first 10 table rows:

[3]:
```
# obtaining the first 10 rows from the df table
# hint: you can use head() and tail() in Jupyter Notebook without wrapping them␣
 ↪into print()
df.head(10)
```

[3]:

	name	Character	rOle	TITLE	Type	\
0	Robert De Niro	Travis Bickle	ACTOR	Taxi Driver	MOVIE	
1	Jodie Foster	Iris Steensma	ACTOR	Taxi Driver	MOVIE	
2	Albert Brooks	Tom	ACTOR	Taxi Driver	MOVIE	
3	Harvey Keitel	Matthew 'Sport' Higgins	ACTOR	Taxi Driver	MOVIE	

```
4  Cybill Shepherd                          Betsy  ACTOR  Taxi Driver  MOVIE
5     Peter Boyle                         Wizard  ACTOR  Taxi Driver  MOVIE
6   Leonard Harris      Senator Charles Palantine  ACTOR  Taxi Driver  MOVIE
7   Diahnne Abbott               Concession Girl  ACTOR  Taxi Driver  MOVIE
8      Gino Ardito           Policeman at Rally  ACTOR  Taxi Driver  MOVIE
9  Martin Scorsese  Passenger Watching Silhouette  ACTOR  Taxi Driver  MOVIE

   release Year              genres  imdb sc0re  imdb v0tes
0          1976  ['drama', 'crime']         8.2    808582.0
1          1976  ['drama', 'crime']         8.2    808582.0
2          1976  ['drama', 'crime']         8.2    808582.0
3          1976  ['drama', 'crime']         8.2    808582.0
4          1976  ['drama', 'crime']         8.2    808582.0
5          1976  ['drama', 'crime']         8.2    808582.0
6          1976  ['drama', 'crime']         8.2    808582.0
7          1976  ['drama', 'crime']         8.2    808582.0
8          1976  ['drama', 'crime']         8.2    808582.0
9          1976  ['drama', 'crime']         8.2    808582.0
```

<div class="alert alert-warning"; style="border-left: 7px solid gold"> Reviewer's comment, v. 1

Great, but in this cell we don't need to load the library again and get data from the file.

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 2

`I had to reload the data and library or it wouldn't recognize df as an object`

This is not quite true. If you delete the reloading of the library and dataframe, then df will be recognized because it was loaded above

Please delete the first two lines of code in this cell

<div class="alert alert-warning"; style="border-left: 7px solid gold"> Reviewer's comment, v. 2

Also, to use blue comment style you could add **markdown cell** and just add in the beginning this code:

```
<div class="alert alert-info"; style="border-left: 7px solid blue">
<b>Student's comment</b>
```

<div class="alert alert-info"; style="border-left: 7px solid blue"> Student's comment I did not know that I could restart and run all. That fixed my issue with it not recognizing the objects. Thank you.

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 3

I was glad to help you :)

Obtain the general information about the table with one command:

[4]:
```
# obtaining general information about the data in df
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85579 entries, 0 to 85578
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0      name       85579 non-null  object
 1   Character     85579 non-null  object
 2   r0le          85579 non-null  object
 3   TITLE         85578 non-null  object
 4     Type        85579 non-null  object
 5   release Year  85579 non-null  int64
 6   genres        85579 non-null  object
 7   imdb sc0re    80970 non-null  float64
 8   imdb v0tes    80853 non-null  float64
dtypes: float64(2), int64(1), object(6)
memory usage: 5.9+ MB
```

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 1

Great - you've used a comprehensive set of methods to have a first look at the data.

The table contains nine columns. The majority store the same data type: object. The only exceptions are `'release Year'` (int64 type), `'imdb sc0re'` (float64 type) and `'imdb v0tes'` (float64 type). Scores and votes will be used in our analysis, so it's important to verify that they are present in the dataframe in the appropriate numeric format. Three columns (`'TITLE'`, `'imdb sc0re'` and `'imdb v0tes'`) have missing values.

According to the documentation: - `'name'` — actor/director's name and last name - `'Character'` — character played (for actors) - `'r0le '` — the person's contribution to the title (it can be in the capacity of either actor or director) - `'TITLE '` — title of the movie (show) - `'  Type'` — show or movie - `'release Year'` — year when movie (show) was released - `'genres'` — list of genres under which the movie (show) falls - `'imdb sc0re'` — score on IMDb - `'imdb v0tes'` — votes on IMDb

We can see three issues with the column names: 1. Some names are uppercase, while others are lowercase. 2. There are names containing whitespace. 3. A few column names have digit '0' instead of letter 'o'.

0.3.1 Conclusions

Each row in the table stores data about a movie or show. The columns can be divided into two categories: the first is about the roles held by different people who worked on the movie or show (role, name of the actor or director, and character if the row is about an actor); the second category is information about the movie or show itself (title, release year, genre, imdb figures).

It's clear that there is sufficient data to do the analysis and evaluate our assumption. However, to move forward, we need to preprocess the data.

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 1

Please note that it is highly recommended to add a conclusion / summary after each section and describe briefly your observations and / or major outcomes of the analysis.

0.4 Stage 2. Data preprocessing

Correct the formatting in the column headers and deal with the missing values. Then, check whether there are duplicates in the data.

[5]:
```
# the list of column names in the df table
print(df.columns)
```

```
Index(['   name', 'Character', 'r0le', 'TITLE', '  Type', 'release Year',
       'genres', 'imdb sc0re', 'imdb v0tes'],
      dtype='object')
```

Change the column names according to the rules of good style: * If the name has several words, use snake_case * All characters must be lowercase * Remove whitespace * Replace zero with letter 'o'

[6]:
```
# renaming columns
def clean_headers(header_list):
    cleaned_headers = []
    for header in header_list:
        header = header.strip()
        header = header.lower()
        header = header.replace('0', 'o')
        header = header.replace(' ', '_')
        cleaned_headers.append(header)
    return cleaned_headers
df.columns = clean_headers(df.columns)
```

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 1

This is a good way to rename the columns.

Check the result. Print the names of the columns once more:

[7]:
```
# checking result: the list of column names
print(df.columns)
```

```
Index(['name', 'character', 'role', 'title', 'type', 'release_year', 'genres',
       'imdb_score', 'imdb_votes'],
      dtype='object')
```

0.4.1 Missing values

First, find the number of missing values in the table. To do so, combine two `pandas` methods:

[8]:
```
# calculating missing values
missing_counts = df.isnull().sum()
missing_counts
```

[8]:
```
name              0
character         0
```

```
role               0
title              1
type               0
release_year       0
genres             0
imdb_score      4609
imdb_votes      4726
dtype: int64
```

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 1

Please note that we do not need to re-download the library and data, we need to download them only at the very beginning of the project

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 1

The isnull() method is selected to find the missing values, it's great!

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 2

And again, there won't be any problems with df here if you don't download the dataframe again :)

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 3

Perfect!

Not all missing values affect the research: the single missing value in `'title'` is not critical. The missing values in columns `'imdb_score'` and `'imdb_votes'` represent around 6% of all records (4,609 and 4,726, respectively, of the total 85,579). This could potentially affect our research. To avoid this issue, we will drop rows with missing values in the `'imdb_score'` and `'imdb_votes'` columns.

[9]:
```
# dropping rows where columns with title, scores and votes have missing values
df = df.dropna(subset=["title", "imdb_score", "imdb_votes"])
```

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 1

Perfect!

Make sure the table doesn't contain any more missing values. Count the missing values again.

[10]:
```
# counting missing values

missing_counts = df.isnull().sum()
print(missing_counts)
```

```
name            0
character       0
role            0
title           0
type            0
release_year    0
genres          0
```

```
imdb_score      0
imdb_votes      0
dtype: int64
```

0.4.2 Duplicates

Find the number of duplicate rows in the table using one command:

[11]:
```
# counting duplicate rows
duplicates = df.duplicated().sum()
duplicates
```

[11]:
```
6994
```

Review the duplicate rows to determine if removing them would distort our dataset.

[12]:
```
# Produce table with duplicates (with original rows included) and review last 5 rows
duplicates = df[df.duplicated(keep=False)]
duplicates
```

[12]:

	name	character	role	\
7560	Philip Greene	Baseball Fan (uncredited)	ACTOR	
7561	Philip Greene	Baseball Fan (uncredited)	ACTOR	
14502	Dan Levy	Reporter	ACTOR	
14512	Dan Levy	Reporter	ACTOR	
18951	Nicolas Le Nev??	unknown	DIRECTOR	
…	…	…	…	
85569	Jessica Cediel	Liliana Navarro	ACTOR	
85570	Javier Gardeaz?bal	Agust??n "Peluca" Ort??z	ACTOR	
85571	Carla Giraldo	Valery Reinoso	ACTOR	
85572	Ana Mar??a S?nchez	Lourdes	ACTOR	
85577	Isabel Gaona	Cacica	ACTOR	

	title	type	release_year	\
7560	How Do You Know	MOVIE	2010	
7561	How Do You Know	MOVIE	2010	
14502	A Very Harold & Kumar Christmas	MOVIE	2011	
14512	A Very Harold & Kumar Christmas	MOVIE	2011	
18951	Sammy & Co	SHOW	2014	
…	…	…	…	
85569	Lokillo	MOVIE	2021	
85570	Lokillo	MOVIE	2021	
85571	Lokillo	MOVIE	2021	
85572	Lokillo	MOVIE	2021	
85577	Lokillo	MOVIE	2021	

	genres	imdb_score	imdb_votes

```
7560     ['comedy', 'drama', 'romance']         5.4     50383.0
7561     ['comedy', 'drama', 'romance']         5.4     50383.0
14502  ['comedy', 'fantasy', 'romance']         6.2     69562.0
14512  ['comedy', 'fantasy', 'romance']         6.2     69562.0
18951         ['animation', 'european']         5.7        31.0
…                                   …         …         …
85569                       ['comedy']         3.8        68.0
85570                       ['comedy']         3.8        68.0
85571                       ['comedy']         3.8        68.0
85572                       ['comedy']         3.8        68.0
85577                       ['comedy']         3.8        68.0

[13988 rows x 9 columns]
```

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 1

And again, please do not reload the data and the library

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 2

And again, please, just delete first 2 rows

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 3

Well done!

There are two clear duplicates in the printed rows. We can safely remove them. Call the `pandas` method for getting rid of duplicate rows:

[13]:
```
# removing duplicate rows
df_no_dupe = df.drop_duplicates()
```

Check for duplicate rows once more to make sure you have removed all of them:

[14]:
```
# checking for duplicates
duplicates = df.duplicated().sum()
```

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 1

Great, you found and removed the duplicates. And did very thorough checks to make sure the duplicates are gone.

Now get rid of implicit duplicates in the `'type'` column. For example, the string `'SHOW'` can be written in different ways. These kinds of errors will also affect the result.

Print a list of unique `'type'` names, sorted in alphabetical order. To do so: * Retrieve the intended dataframe column * Apply a sorting method to it * For the sorted column, call the method that will return all unique column values

[15]:
```
# viewing unique type names
unique_types = df['type'].unique()
unique_types
```

[15]:
```
array(['MOVIE', 'the movie', 'tv show', 'shows', 'movies', 'tv shows',
       'tv series', 'tv', 'SHOW'], dtype=object)
```

Look through the list to find implicit duplicates of `'show'` (`'movie'` duplicates will be ignored since the assumption is about shows). These could be names written incorrectly or alternative names of the same genre.

You will see the following implicit duplicates: * `'shows'` * `'SHOW'` * `'tv show'` * `'tv shows'` * `'tv series'` * `'tv'`

To get rid of them, declare the function `replace_wrong_show()` with two parameters: * `wrong_shows_list=` — the list of duplicates * `correct_show=` — the string with the correct value

The function should correct the names in the `'type'` column from the `df` table (i.e., replace each value from the `wrong_shows_list` list with the value in `correct_show`).

[16]:
```
# function for replacing implicit duplicates
def replace_wrong_show(dataframe, wrong_shows_list, correct_show):
    for wrong_show in wrong_shows_list:
        df['type'] = df['type'].replace(wrong_show, correct_show, regex=True)

wrong_shows_list = ['shows', 'tv show', 'tv shows', 'tv series', 'tv', 'SHOW␣
 ↪SHOW']

correct_show = 'SHOW'



print(df)
```

```
                    name                character      role        title  \
0          Robert De Niro            Travis Bickle     ACTOR  Taxi Driver
1            Jodie Foster            Iris Steensma     ACTOR  Taxi Driver
2           Albert Brooks                      Tom     ACTOR  Taxi Driver
3           Harvey Keitel  Matthew 'Sport' Higgins     ACTOR  Taxi Driver
4         Cybill Shepherd                    Betsy     ACTOR  Taxi Driver
…                     …                        …       …          …
85574    Adelaida Buscato              Mar??a Paz     ACTOR      Lokillo
85575  Luz Stella Luengas             Karen Bayona     ACTOR      Lokillo
85576        In??s Prieto                    Fanny     ACTOR      Lokillo
85577        Isabel Gaona                   Cacica     ACTOR      Lokillo
85578      Julian Gaviria                  unknown  DIRECTOR      Lokillo

            type  release_year              genres  imdb_score  imdb_votes
0          MOVIE          1976  ['drama', 'crime']         8.2    808582.0
1          MOVIE          1976  ['drama', 'crime']         8.2    808582.0
2          MOVIE          1976  ['drama', 'crime']         8.2    808582.0
3          MOVIE          1976  ['drama', 'crime']         8.2    808582.0
4          MOVIE          1976  ['drama', 'crime']         8.2    808582.0
```

```
…          …           …                …          …          …
85574  the movie          2021          ['comedy']        3.8        68.0
85575  the movie          2021          ['comedy']        3.8        68.0
85576  the movie          2021          ['comedy']        3.8        68.0
85577      MOVIE          2021          ['comedy']        3.8        68.0
85578  the movie          2021          ['comedy']        3.8        68.0

[80853 rows x 9 columns]
```

Call `replace_wrong_show()` and pass it arguments so that it clears implicit duplicates and replaces them with `SHOW`:

[17]:
```
# removing implicit duplicates
replace_wrong_show(df, wrong_shows_list, correct_show)
```

Make sure the duplicate names are removed. Print the list of unique values from the `'type'` column:

[18]:
```
# viewing unique genre names
print(df['type'].unique())
```

```
['MOVIE' 'the movie' 'SHOW' 'movies']
```

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 1

Yes, this is what was needed!

0.4.3 Conclusions

We detected three issues with the data:

- Incorrect header styles
- Missing values
- Duplicate rows and implicit duplicates

The headers have been cleaned up to make processing the table simpler.

All rows with missing values have been removed.

The absence of duplicates will make the results more precise and easier to understand.

Now we can move on to our analysis of the prepared data.

0.5 Stage 3. Data analysis

Based on the previous project stages, you can now define how the assumption will be checked. Calculate the average amount of votes for each score (this data is available in the `imdb_score` and `imdb_votes` columns), and then check how these averages relate to each other. If the averages for shows with the highest scores are bigger than those for shows with lower scores, the assumption appears to be true.

Based on this, complete the following steps:

- Filter the dataframe to only include shows released in 1999 or later.

- Group scores into buckets by rounding the values of the appropriate column (a set of 1-10 integers will help us make the outcome of our calculations more evident without damaging the quality of our research).
- Identify outliers among scores based on their number of votes, and exclude scores with few votes.
- Calculate the average votes for each score and check whether the assumption matches the results.

To filter the dataframe and only include shows released in 1999 or later, you will take two steps. First, keep only titles published in 1999 or later in our dataframe. Then, filter the table to only contain shows (movies will be removed).

[19]:
```
# using conditional indexing modify df so it has only titles released after␣
↪1999 (with 1999 included)
# give the slice of dataframe new name
shows_1999 = df[df['release_year'] >= 1999]
```

[20]:
```
# repeat conditional indexing so df has only shows (movies are removed as␣
↪result)
shows_1999 = shows_1999[shows_1999['type'] == 'SHOW']
shows_1999
```

[20]:

	name	character	role	title	\
1664	Jeff Probst	Himself - Host	ACTOR	Survivor	
2076	Mayumi Tanaka	Monkey D. Luffy (voice)	ACTOR	One Piece	
2077	Kazuya Nakai	Roronoa Zoro (voice)	ACTOR	One Piece	
2078	Akemi Okamura	Nami (voice)	ACTOR	One Piece	
2079	Kappei Yamaguchi	Usopp (voice)	ACTOR	One Piece	
…	…	…	…	…	
85433	Maneerat Kam-Uan	Ae	ACTOR	Let's Eat	
85434	Rudklao Amratisha	unknown	ACTOR	Let's Eat	
85435	Jaturong Mokjok	unknown	ACTOR	Let's Eat	
85436	Pisamai Wilaisak	unknown	ACTOR	Let's Eat	
85437	Sarawut Wichiensarn	unknown	DIRECTOR	Let's Eat	

	type	release_year	genres	\
1664	SHOW	2000	['reality']	
2076	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa…	
2077	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa…	
2078	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa…	
2079	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa…	
…	…	…	…	
85433	SHOW	2021	['drama', 'comedy']	
85434	SHOW	2021	['drama', 'comedy']	
85435	SHOW	2021	['drama', 'comedy']	
85436	SHOW	2021	['drama', 'comedy']	
85437	SHOW	2021	['drama', 'comedy']	

```
       imdb_score  imdb_votes
1664          7.4     24687.0
2076          8.8    117129.0
2077          8.8    117129.0
2078          8.8    117129.0
2079          8.8    117129.0
…             …          …
85433         8.2         5.0
85434         8.2         5.0
85435         8.2         5.0
85436         8.2         5.0
85437         8.2         5.0

[15082 rows x 9 columns]
```

The scores that are to be grouped should be rounded. For instance, titles with scores like 7.8, 8.1, and 8.3 will all be placed in the same bucket with a score of 8.

[21]:
```
# rounding column with scores
shows_1999['imdb_score'] = shows_1999['imdb_score'].round()
#checking the outcome with tail()
print(df.tail())
```

```
                    name     character      role    title       type  \
85574    Adelaida Buscato    Mar??a Paz     ACTOR  Lokillo  the movie
85575  Luz Stella Luengas  Karen Bayona     ACTOR  Lokillo  the movie
85576        In??s Prieto         Fanny     ACTOR  Lokillo  the movie
85577        Isabel Gaona        Cacica     ACTOR  Lokillo      MOVIE
85578      Julian Gaviria       unknown  DIRECTOR  Lokillo  the movie

       release_year      genres  imdb_score  imdb_votes
85574          2021  ['comedy']         3.8        68.0
85575          2021  ['comedy']         3.8        68.0
85576          2021  ['comedy']         3.8        68.0
85577          2021  ['comedy']         3.8        68.0
85578          2021  ['comedy']         3.8        68.0
```

It is now time to identify outliers based on the number of votes.

[22]:
```
# Use groupby() for scores and count all unique values in each group, print the␣
 ↪result
grouped_votes = shows_1999.groupby('imdb_score')['name'].nunique()
grouped_votes
```

[22]:
```
imdb_score
2.0       24
3.0       27
4.0      174
```

```
5.0      581
6.0     2365
7.0     4342
8.0     4194
9.0      542
10.0       8
Name: name, dtype: int64
```

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 1

Not quite right, now `df` is used in the grouping, but after all, we made changes in `shows_1999`

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 2

Perfect!

Based on the aggregation performed, it is evident that scores 2 (24 voted shows), 3 (27 voted shows), and 10 (only 8 voted shows) are outliers. There isn't enough data for these scores for the average number of votes to be meaningful.

To obtain the mean numbers of votes for the selected scores (we identified a range of 4-9 as acceptable), use conditional filtering and grouping.

[23]:
```
# filter dataframe using two conditions (scores to be in the range 4-9)
filtered_df = shows_1999[(shows_1999['imdb_score'] >= 4) &␣
 ↪(shows_1999['imdb_score'] <= 9)]
# group scores and corresponding average number of votes, reset index and print␣
 ↪the result
filtered_df = filtered_df.groupby('imdb_score')['imdb_votes'].mean().
 ↪reset_index()
filtered_df
```

[23]:

	imdb_score	imdb_votes
0	4.0	6544.480000
1	5.0	2846.192475
2	6.0	3168.731413
3	7.0	8012.183953
4	8.0	28361.942115
5	9.0	121401.017153

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 1

And again, please use `filtered_df` in the 4th row in the grouping, not `df`

<div class="alert alert-danger"; style="border-left: 7px solid red"> Reviewer's comment, v. 2

Almost everything is as it should be, but in the row:

```
filtered_df = filtered_df[(filtered_df['imdb_score'] >= 4) & (filtered_df['imdb_score'] <= 9)]
```

We should filter based on `shows_1999`, not `filtered_df`

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 3

Very well!

Now for the final step! Round the column with the averages, rename both columns, and print the dataframe in descending order.

[24]:
```
# round column with averages
filtered_df['imdb_votes'] = filtered_df['imdb_votes'].round()
# rename columns

# print dataframe in descending order
filtered_df = filtered_df.sort_values(by='imdb_votes', ascending=False)
print(filtered_df)
```

```
   imdb_score  imdb_votes
5         9.0    121401.0
4         8.0     28362.0
3         7.0      8012.0
0         4.0      6544.0
2         6.0      3169.0
1         5.0      2846.0
```

The assumption macthes the analysis: the shows with the top 3 scores have the most amounts of votes.

0.6 Conclusion

The research done confirms that highly-rated shows released during the "Golden Age" of television also have the most votes. While shows with score 4 have more votes than ones with scores 5 and 6, the top three (scores 7-9) have the largest number. The data studied represents around 94% of the original set, so we can be confident in our findings.

<div class="alert alert-success"; style="border-left: 7px solid green"> Reviewer's comment, v. 1

Overall conclusion is an important part, where we should include the summary of the outcomes of the project.