Basic Python - Project

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.

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

Stage 1. Data overview

Open and explore the data.

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

In [14]:
```
# importing pandas
import pandas as pd
```

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

In [15]:
```
# reading the files and storing them to df
```

```
df = pd.read_csv('/datasets/movies_and_shows.csv')
```

Print the first 10 table rows:

In [16]:
```
# obtaining the first 10 rows from the df table
# hint: you can use head() and tail() in Jupyter Notebook without wrapping t
df.head(10)
```

Out[16]:

	name	Character	r0le	TITLE	Type	release Year	genres	imdb sc0re	imdb v0tes
0	Robert De Niro	Travis Bickle	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
1	Jodie Foster	Iris Steensma	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
2	Albert Brooks	Tom	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
3	Harvey Keitel	Matthew 'Sport' Higgins	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
4	Cybill Shepherd	Betsy	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
5	Peter Boyle	Wizard	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
6	Leonard Harris	Senator Charles Palantine	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
7	Diahnne Abbott	Concession Girl	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
8	Gino Ardito	Policeman at Rally	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0
9	Martin Scorsese	Passenger Watching Silhouette	ACTOR	Taxi Driver	MOVIE	1976	['drama', 'crime']	8.2	808582.0

Obtain the general information about the table with one command:

In [17]:
```
# 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
```

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'.

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.

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.

In [18]:
```
# the list of column names in the df table
df.columns
```

Out[18]:
```
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'

In [19]:
```
# renaming columns
column_names = list(df.columns)
column_names
```

Out[19]:
```
['   name',
 'Character',
 'r0le',
 'TITLE',
 '  Type',
 'release Year',
 'genres',
 'imdb sc0re',
 'imdb v0tes']
```

In [20]:
```
#remove white space
def remove_white_space(dataframe,col_names):
    '''will remove white space in the column'''
    for col in col_names:
        dataframe = dataframe.rename(columns = {col:col.strip()})
    return dataframe

df = remove_white_space(df,column_names)
column_names = list(df.columns)
```

```
column_names
```

Out[20]:
```
['name',
 'Character',
 'r0le',
 'TITLE',
 'Type',
 'release Year',
 'genres',
 'imdb sc0re',
 'imdb v0tes']
```

In [21]:
```
#make columns lowercase
def lower_case(dataframe,col_names):
    '''will make columns lowercase'''
    for col in col_names:
        dataframe = dataframe.rename(columns = {col:col.lower()})
    return dataframe

df = lower_case(df,column_names)
column_names = list(df.columns)
column_names
```

Out[21]:
```
['name',
 'character',
 'r0le',
 'title',
 'type',
 'release year',
 'genres',
 'imdb sc0re',
 'imdb v0tes']
```

In [22]:
```
#replace 0 with o
def replace_zero(dataframe,col_names):
    '''will make columns that have zero with o'''
    for col in col_names:
        dataframe = dataframe.rename(columns = {col:col.replace('0','o')})
    return dataframe

df = replace_zero(df,column_names)
column_names = list(df.columns)
column_names
```

Out[22]:
```
['name',
 'character',
 'role',
 'title',
 'type',
 'release year',
 'genres',
 'imdb score',
 'imdb votes']
```

In [23]:
```
#use snake case
def replace_zero(dataframe,col_names):
    '''will make columns with more than one words use snake case'''
    for col in col_names:
        dataframe = dataframe.rename(columns = {col:col.replace(' ','_')})
    return dataframe

df = replace_zero(df,column_names)
column_names = list(df.columns)
column_names
```

Out[23]:
```
['name',
 'character',
 'role',
 'title',
 'type',
 'release_year',
 'genres',
 'imdb_score',
 'imdb_votes']
```

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

In [24]:
```
# checking result: the list of column names
df.columns
```

Out[24]:
```
Index(['name', 'character', 'role', 'title', 'type', 'release_year', 'genre
s',
       'imdb_score', 'imdb_votes'],
      dtype='object')
```

Missing values

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

In [25]:
```
# calculating missing values
df.isna().sum()
```

Out[25]:
```
name              0
character         0
role              0
title             1
type              0
release_year      0
genres            0
imdb_score     4609
imdb_votes     4726
dtype: int64
```

We identified missing values in several columns. While the missing value in `'title'`

isn't critical, missing values in `'imdb_score'` and `'imdb_votes'` affect around 6% of the data, which could impact our analysis. To ensure data integrity, we'll drop all rows with missing values.

In [26]:
```
# dropping rows where columns with scores, and votes have missing values
df = df.dropna(subset = ['imdb_votes','imdb_score']).reset_index(drop=True)
```

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

In [27]:
```
# counting missing values
df.isna().sum()
```

Out[27]:
```
name            0
character       0
role            0
title           0
type            0
release_year    0
genres          0
imdb_score      0
imdb_votes      0
dtype: int64
```

Duplicates

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

In [28]:
```
# counting duplicate rows
df.duplicated().sum()
```

Out[28]:
```
6994
```

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

In [29]:
```
# Produce table with duplicates (with original rows included) and review las
duplicates = df[df.duplicated()]
duplicates
```

Out[29]:

	name	character	role	title	type	release_year	g
7420	Philip Greene	Baseball Fan (uncredited)	ACTOR	How Do You Know	MOVIE	2010	['co 'd 'rom
14071	Dan Levy	Reporter	ACTOR	A Very Harold & Kumar Christmas	MOVIE	2011	['co 'fa 'rom
18359	Nicolas Le Nev??	unknown	DIRECTOR	Sammy & Co	SHOW	2014	['anim 'euro
21837	John Iii Franklin	Himself	ACTOR	Last Chance U	SHOW	2016	['document 's
28746	Claudio Roca	Nicol?s	ACTOR	Narcos: Mexico	SHOW	2018	['drama', 'c
...	...	...	...	...	...	...	
80843	Jessica Cediel	Liliana Navarro	ACTOR	Lokillo	MOVIE	2021	['cor
80844	Javier Gardeaz?-bal	Agust??n "Peluca" Ort??z	ACTOR	Lokillo	MOVIE	2021	['cor
80845	Carla Giraldo	Valery Reinoso	ACTOR	Lokillo	MOVIE	2021	['cor
80846	Ana Mar??a S?nchez	Lourdes	ACTOR	Lokillo	MOVIE	2021	['cor
80851	Isabel Gaona	Cacica	ACTOR	Lokillo	MOVIE	2021	['cor

6994 rows × 9 columns

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

In [30]:
```
# removing duplicate rows
df = df.drop_duplicates().reset_index(drop=True)
```

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

In [31]:
```
# checking for duplicates
df.duplicated().sum()
```

Out[31]:
```
0
```

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

In [32]:
```
# viewing unique type names
df['type'].unique()
```

Out[32]:
```
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`).

In [33]:
```
# function for replacing implicit duplicates
def replace_wrong_show(dataframe,column_name,wrong_shows_list, correct_show)
    dataframe[column_name] = dataframe[column_name].replace(wrong_shows_list
    return dataframe
```

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

In [36]:
```
show_list = [x for x in list(df['type'].unique()) if 'movie' not in x.lower(
show_list
```

Out[36]:
```
['tv show', 'shows', 'tv shows', 'tv series', 'tv', 'SHOW']
```

In [37]:
```
# removing implicit duplicates
df = replace_wrong_show(dataframe=df,column_name='type',wrong_shows_list=shc
```

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

In [38]:
```
# viewing unique genre names
df['type'].unique()
```

Out[38]:
```
array(['MOVIE', 'the movie', 'SHOW', 'movies'], dtype=object)
```

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.

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).

In [41]:
```
# using conditional indexing modify df so it has only titles released after
# give the slice of dataframe new name

df_after_1999 = df[df['release_year'] >= 1999]
df_after_1999.head()
```

Out[41]:

	name	character	role	title	type	release_year	genres	imdb_scor
1661	Jeff Probst	Himself - Host	ACTOR	Survivor	SHOW	2000	['reality']	7.
1952	Benicio del Toro	Franky Four Fingers	ACTOR	Snatch	MOVIE	2000	['crime', 'comedy']	8.
1953	Dennis Farina	Cousin Avi	ACTOR	Snatch	MOVIE	2000	['crime', 'comedy']	8.
1954	Vinnie Jones	Bullet Tooth Tony	ACTOR	Snatch	MOVIE	2000	['crime', 'comedy']	8.
1955	Brad Pitt	Mickey O'Neil	ACTOR	Snatch	MOVIE	2000	['crime', 'comedy']	8.

In [42]:
```
# repeat conditional indexing so df has only shows (movies are removed as re
df_only_shows = df_after_1999[df_after_1999['type']=='SHOW']
df_only_shows.head()
```

Out[42]:

	name	character	role	title	type	release_year	genres	imdb_
1661	Jeff Probst	Himself - Host	ACTOR	Survivor	SHOW	2000	['reality']	
2073	Mayumi Tanaka	Monkey D. Luffy (voice)	ACTOR	One Piece	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa...	
2074	Kazuya Nakai	Roronoa Zoro (voice)	ACTOR	One Piece	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa...	
2075	Akemi Okamura	Nami (voice)	ACTOR	One Piece	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa...	
2076	Kappei Yamaguchi	Usopp (voice)	ACTOR	One Piece	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa...	

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.

In [44]:
```
# rounding column with scores
df_only_shows['imdb_score_rounded'] = df_only_shows['imdb_score'].agg(lambda

#checking the outcome with tail()
df_only_shows.tail()
```

```
/tmp/ipykernel_33/1103168508.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_only_shows['imdb_score_rounded'] = df_only_shows['imdb_score'].agg(lamb
da x: round(x))
```

Out[44]:

	name	character	role	title	type	release_year	genres	imdb
73831	Maneerat Kam-Uan	Ae	ACTOR	Let's Eat	SHOW	2021	['drama', 'comedy']	
73832	Rudklao Amratisha	unknown	ACTOR	Let's Eat	SHOW	2021	['drama', 'comedy']	
73833	Jaturong Mokjok	unknown	ACTOR	Let's Eat	SHOW	2021	['drama', 'comedy']	
73834	Pisamai Wilaisak	unknown	ACTOR	Let's Eat	SHOW	2021	['drama', 'comedy']	
73835	Sarawut Wichiensarn	unknown	DIRECTOR	Let's Eat	SHOW	2021	['drama', 'comedy']	

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

In [49]:
```
# Use groupby() for scores and count all unique values in each group, print
df_only_shows.groupby('imdb_score_rounded')['name'].count()
```

Out[49]:
```
imdb_score_rounded
2       24
3       27
4      180
5      592
6     2494
7     4706
8     4842
9      557
10       8
Name: name, dtype: int64
```

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.

In [52]:
```
# filter dataframe using two conditions (scores to be in the range 4-9)
df_range = df_only_shows[(df_only_shows['imdb_score_rounded'] >= 4) & (df_or
df_range.head()
```

Out[52]:

	name	character	role	title	type	release_year	genres	imdb_
1661	Jeff Probst	Himself - Host	ACTOR	Survivor	SHOW	2000	['reality']	
2073	Mayumi Tanaka	Monkey D. Luffy (voice)	ACTOR	One Piece	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa...	
2074	Kazuya Nakai	Roronoa Zoro (voice)	ACTOR	One Piece	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa...	
2075	Akemi Okamura	Nami (voice)	ACTOR	One Piece	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa...	
2076	Kappei Yamaguchi	Usopp (voice)	ACTOR	One Piece	SHOW	1999	['animation', 'action', 'comedy', 'drama', 'fa...	

In [57]:
```
# group scores and corresponding average number of votes, reset index and pr
df_grouped_scores = df_range.groupby('imdb_score_rounded')['imdb_votes'].agg
df_grouped_scores
```

Out[57]:

	imdb_score_rounded	mean
0	4	5277.583333
1	5	3143.942568
2	6	3481.717322
3	7	8727.068211
4	8	30299.460967
5	9	126904.109515

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

In [61]:
```
# round column with averages
df_grouped_scores['mean'] = df_grouped_scores['mean'].round()

# rename columns
```

```
df_grouped_scores = df_grouped_scores.rename(columns = {'imdb_score_rounded'
df_grouped_scores

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

Out[61]:

	imdb score	mean
5	9	126904.0
4	8	30299.0
3	7	8727.0
0	4	5278.0
2	6	3482.0
1	5	3144.0

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

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.