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 [1]:
```python
import pandas as pd
```

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

In [2]:
```python
df = pd.read_csv("/datasets/movies_and_shows.csv")
```

Print the first 10 table rows:

In [3]:
```python
df.head(10)
```

Out[3]:

	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:

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 [4]:
```python
df.describe()
```

Out[4]:

	release Year	imdb sc0re	imdb v0tes
count	85579.000000	80970.000000	8.085300e+04
mean	2015.879994	6.425877	5.978271e+04
std	7.724668	1.122655	1.846287e+05
min	1954.000000	1.500000	5.000000e+00
25%	2015.000000	5.700000	1.266000e+03
50%	2018.000000	6.500000	5.448000e+03
75%	2021.000000	7.200000	3.360900e+04
max	2022.000000	9.500000	2.294231e+06

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 [5]:
```python
df.columns
```

Out[5]:
```
Index(['   name', 'Character', 'r0le', 'TITLE', '  Type', 'release Year',
       'genres', 'imdb sc0re', 'imdb v0tes'],
      dtype='object')
```

In [6]:
```python
df = df.rename(
columns = {'   name':"name",
           'Character': "character",
           'r0le': "role", 'TITLE': "title",
           '  Type': "type",
           'release Year': "release_year",
           'genres': "genres",
           'imdb sc0re': "imdb_score",
           'imdb v0tes': "imdb_votes"
          })
```

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

In [7]:
```python
df.columns
```

Out[7]:
```
Index(['name', 'character', 'role', 'title', 'type', 'release_year', 'genres',
       '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 [8]:
```python
df.isna().sum()
```

Out[8]:
```
name               0
character          0
role               0
title              1
type               0
release_year       0
genres             0
imdb_score      4609
imdb_votes      4726
dtype: int64
```

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.

In [9]:
```python
df.dropna(axis=0, inplace= True)
df
```

Out[9]:

	name	character	role	title	type	release_year	genres	imdb_score	imdb_votes
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
...	...	...	...	...	...	...	...	...	...
85574	Adelaida Buscato	Mar??a Paz	ACTOR	Lokillo	the movie	2021	['comedy']	3.8	68.0
85575	Luz Stella Luengas	Karen Bayona	ACTOR	Lokillo	the movie	2021	['comedy']	3.8	68.0
85576	In??s Prieto	Fanny	ACTOR	Lokillo	the movie	2021	['comedy']	3.8	68.0
85577	Isabel Gaona	Cacica	ACTOR	Lokillo	MOVIE	2021	['comedy']	3.8	68.0
85578	Julian Gaviria	unknown	DIRECTOR	Lokillo	the movie	2021	['comedy']	3.8	68.0

80853 rows × 9 columns

Duplicates

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

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

In [10]:
```python
df.isna().sum()
```

Out[10]:
```
name            0
character       0
role            0
title           0
type            0
release_year    0
genres          0
imdb_score      0
imdb_votes      0
dtype: int64
```

In [11]:
```python
df.duplicated().sum()
```

Out[11]:
```
6994
```

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

In [12]:
```python
df.duplicated().tail()
```

Out[12]:
```
85574    False
85575    False
85576    False
85577     True
85578    False
dtype: bool
```

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 [13]:
```python
removed_duplicates = df.drop_duplicates()
```

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

In [14]:
```python
removed_duplicates.duplicated().sum()
```

Out[14]:
```
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 [15]:
```python
show_type = df['type'].unique()
```

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 [16]:
```python
def replace_wrong_show(wrong_type, correct_type):
    correct = df["type"].replace(wrong_type, correct_type)
    return correct
```

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

In [17]:
```python
df["type"] = replace_wrong_show(["shows", "tv series", "tv", "tv shows", "tv show"], "SHOW")
```

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

In [18]:
```python
df["type"].unique()
```

Out[18]:
```
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 [19]:
```python
millenium = df[(df["release_year"] >= 1999) & (df["type"] == "SHOW")]
```

In [20]:
```python
df = df[df["type"] == "SHOW"]
```

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 [25]:
```python
shows_only = millenium["imdb_score"].round(0)

shows_only.tail()
```

Out[25]:
```
85433    8.0
85434    8.0
85435    8.0
85436    8.0
85437    8.0
Name: imdb_score, dtype: float64
```

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

In [22]:
```python
outlier_votes = df.groupby('imdb_votes')["title"].count()
```

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 [23]:
```python
average = df[df["imdb_votes"] >= 4].groupby("imdb_votes")
average = df[df["imdb_votes"] <= 9].groupby("imdb_votes")# filter dataframe using two conditions (scores to be in the range 4-9)
# group scores and corresponding average number of votes, reset index and print the result
df = average.mean().reset_index()
df.round(decimals=0).astype("Int64")
```

Out[23]:

	imdb_votes	release_year	imdb_score
0	5	2021	7
1	6	2016	6
2	7	2013	6
3	8	2021	8
4	9	2011	8

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

In [24]:
```python
df.columns

df = df.rename(
columns = {'imdb_votes': "average_imdb_votes",
           'release_year': "release_year",
           'imdb_score': "average_imdb_score"
          })


df.sort_values(by="average_imdb_votes", ascending=False).round()
```

Out[24]:

	average_imdb_votes	release_year	average_imdb_score
4	9.0	2011.0	8.0
3	8.0	2021.0	8.0
2	7.0	2013.0	6.0
1	6.0	2016.0	6.0
0	5.0	2021.0	7.0

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.