Exploratory Data Analysis of Netflix (US) Content

Introduction

Recently, I've recognized data science as an important skill to possess as an Engineer. I've since developed great interest and am determined to learn data science tools, starting with SQL. This is my first official data science project. In this document, I will present my process (including mistakes) throughout this EDA as well as the lessons I learnt.

Objectives

- To learn and get some practice using SQL programming language to query data
- Run an Exploratory Data Analysis on Netflix content

Resources used

- Dataset: two files (titles.csv and credits.csv) sourced from Kaggle- https://www.kaggle.com/datasets/victorsoeiro/netflix-tv-shows-and-movies
- PostgreSQL 14.3.1- For data querying
- Microsoft Excel- For charting purposes

Importing the data to PostgreSQL

To start, I first needed to create a database on PostgreSQL and import the CSV files.

I first create both the titles and credits table with the appropriate columns and their respective datatypes.

[1]

```
CREATE TABLE titles (
    id VARCHAR(20) PRIMARY KEY,
--Assigning id as primary key of titles table
    title VARCHAR(200),
    type VARCHAR(10),
    description VARCHAR(10000),
    release_year SMALLINT,
    age_certification VARCHAR(50),
    runtime SMALLINT,
    genre VARCHAR(1000),
    production_countries VARCHAR(50),
    seasons SMALLINT,
    imdb_id VARCHAR(50),
    imdb_score NUMERIC(3,1),
    imdb_votes INTEGER,
    tmdb_popularity NUMERIC (10,3),
    tmdb_score NUMERIC(3,1)
);
```

The reason "id" is inserted as VarChar data type is because they contain letters in their id, e.g., TM5074.

[2]

```
CREATE TABLE credits (
    person_id INTEGER PRIMARY KEY,
--assigning person_id as primary key of credits table
    id VARCHAR(20) REFERENCES titles(id),
--assigning id as foreign key & referencing to titles table
    name VARCHAR(100),
    character VARCHAR(500),
    role VARCHAR(10)
);
```

I then began to import the CSV files into the tables that I've created.

[3]

```
COPY titles
FROM 'D:\titles.csv'
DELIMITER ','
CSV HEADER;
```

[4]

```
COPY credits
FROM 'D:\credits.csv'
DELIMITER ','
CSV HEADER;
```

Upon running [4], I got an error indicating that I can't have a primary key that repeats. Therefore, I had to drop it as a primary key.

[5]

```
ALTER TABLE credits
DROP CONSTRAINT credits_pkey;
```

Data cleaning

Upon first inspection, I noticed a lot of null values in the tables. I also noticed that the genres and production countries columns were displayed as arrays, e.g. ['comedy','fantasy']. I did however create this column with a string datatype. In these columns, empty values were displayed as "[]". Since they are empty and basically NULL, I updated the table to reflect this.

[6]

```
UPDATE titles
SET genres = NULL
WHERE genres ILIKE '%[]%'
```

I then proceeded to identify how many NULL values there are in each column. [7] is the query for the title column, which I then repeated for the rest of the columns.

[7]

```
SELECT COUNT (*) from titles
WHERE title IS NULL
```

Total rows in titles table=5,806

id: 0
title: 1
type: 0
description: 18
release_year: 0
age_certification: 2610
runtime: 0
genres: 68
production_countries: 232
seasons: 3759
imdb_id: 444
imdb_score: 523
imdb_votes: 539
tmdb_popularity: 94
tmdb_score: 318

Total number of anomalies (missing values) found in titles = 8606

Note: The number of null values in seasons is equal to the number of 'MOVIE' in the database, hence, there are 0 'SHOW' with null values in seasons.

Total rows in credits table= 77,213

person_id: 0
id: 0
name: 0
character: 9627
role: 0

I am just going to keep these numbers in mind as in some cases, these numbers affect our confidence in the results, especially so for the age_certification column.

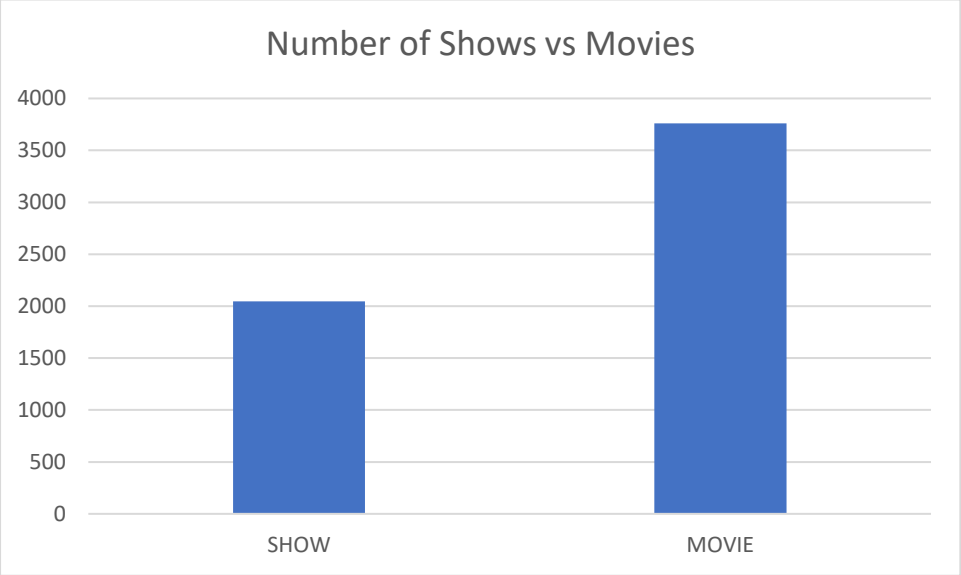
Exploratory data analysis

Before starting on the data analysis, I set out a bunch of questions that I want answered. I've ordered the questions here in increasing complexity, but I've also grouped similar queries together. Stay tuned for the last question as that really was a tough one to deal with as a beginner. I also used Microsoft Excel for data visualisation purposes.

1. Number of shows vs movies

[8]

```
SELECT type, COUNT(*) FROM titles
GROUP BY type;
```



As illustrated in the chart above, there is almost double the number of movies than shows available on Netflix US. A truer comparison would be to compare the total runtime of each content; however, the dataset did not contain the total number of episodes of each show. I'm sure the show would have a much higher total runtime if this analysis was done.

2. Ranking movies and shows by ratings

I wanted to do a little ranking with some hypothetical filters. So, let's say I want the top 10 highest rated (IMDB) scifi movies from the UK or the USA.

[9]

```
SELECT
title,genres,imdb_score,runtime,release_year,age_certification,production_c
ountries
FROM titles
WHERE (production_countries LIKE '%US%' OR production_countries LIKE
'%GB%')
AND genres LIKE '%scifi%'
AND type='MOVIE' AND imdb_score IS NOT NULL
ORDER BY imdb_score DESC,imdb_votes DESC
LIMIT 10;
```

I also used the number of imdb votes to act as a tiebreaker in the case where the imdb scores are the same.

title	genres	imdb_score	runtime	release_yea	age_certifi	production_
Inception	['scifi', 'music', 'thriller', 'action']	8.8	148	2010	PG-13	['GB', 'US']
Her	['drama', 'romance', 'scifi']	8	126	2013	R	['US']
Blade Runner 20	['scifi', 'thriller', 'action', 'drama']	8	164	2017	R	['CA', 'HU',
Gattaca	['thriller', 'drama', 'scifi']	7.8	106	1997	PG-13	['US']
The Mitchells vs	['animation', 'comedy', 'family', 's	7.6	113	2021	PG	['US']
Looper	['thriller', 'scifi', 'drama', 'crime', 'a	7.4	119	2012	R	['US']
Invader ZIM: Ent	['drama', 'family', 'horror', 'action	7.4	71	2019	NULL	['US']
Starship Trooper	['scifi', 'thriller', 'action']	7.3	129	1997	R	['US']
Okja	['drama', 'scifi', 'action']	7.3	122	2017	PG-13	['KR', 'US']
Apollo 10Â½: A	['comedy', 'action', 'animation', 'r	7.3	98	2022	PG-13	['NL', 'US']

3. Finding cast of movies/shows

I wanted to take the chance to use JOINS. So, let's say I want to see the cast of the movie "Inception" along with who the director is. I also checked to see the differences in using JOINS and subqueries in scenarios like this and found that for most instances like this, JOINS are more efficient and cleaner.

[10]

```
SELECT titles.id,title,name,character,role
FROM titles
INNER JOIN credits
ON titles.id=credits.id
WHERE title ILIKE 'inception';
```

This successfully listed all the actors and the director for the movie "Inception".

4. What movie did a particular actor/director work in

Here, I wanted to do the opposite, find all the content that a particular person is credited to. I chose David Attenborough for this example. I also sorted the list by the IMDB score.

[11]

```
SELECT titles.id,title,name,role,type,release_year,genres,imdb_score
FROM titles
INNER JOIN credits
ON titles.id=credits.id
WHERE name ILIKE 'david attenborough'
ORDER BY imdb_score DESC;
```

id	title	name	role	type	release	genres	imdb_sc
ts85398	Our Planet	David Attenborough	ACTOR	SHOW	2019	['documentat	9.3
tm853783	David Attenborough: A Life o	David Attenborough	ACTOR	MOVIE	2020	['documentat	9
tm825102	Our Planet: Behind the Scene	David Attenborough	ACTOR	MOVIE	2019	['documentat	8.4
ts282131	Attenborough's Life in Colour	David Attenborough	ACTOR	SHOW	2021	['documentat	8.2
tm104154	Breaking Boundaries: The Sci	David Attenborough	ACTOR	MOVIE	2021	['documentat	7.8

5. What is the distribution of content from different release years?

At first, I was going to separate the release years into 5-year intervals, as this is how I've usually seen these sorts of data visualized. First, I checked for the earliest and latest release date available in the dataset.

[12]

```
SELECT MIN(release_year),MAX(release_year) FROM titles
```

I decided to ignore titles for the year 2022 since the year is not yet complete.

I was going run the following commands for each of the 5-year intervals. I also learnt from my friend who is a data analyst that loops (wasn't covered in my SQL course) can be used here, hence this could be done in a single query. I'll definitely keep that in mind for my next projects.

[13]

```
SELECT COUNT(release_year) FROM titles
WHERE release_year BETWEEN 2012 AND 2016;
```

However, I then realized that though time is continuous, since the release years are just years, I can treat them as categories and therefore use GROUP BY. This way, I can get more precise since my intervals is just 1 year, and I can obtain this information in a single query.

[14]

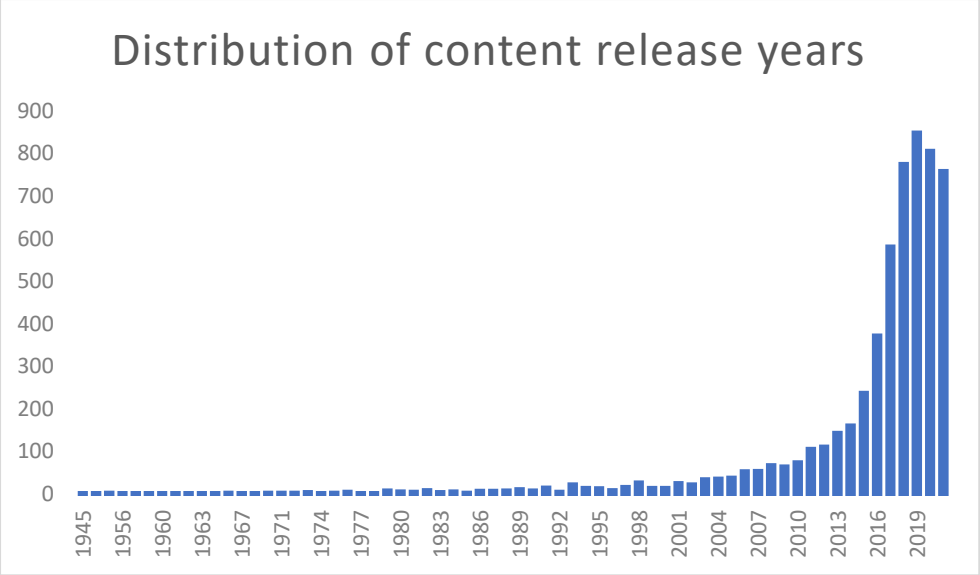
```
SELECT release_year, COUNT(*) FROM titles
WHERE release_year<2022
GROUP BY release_year
ORDER BY release_year;
```

If needed, I could also do so separately for shows and movies using the following syntax.

[15]

```
SELECT type, release_year, COUNT(*)AS total
FROM titles
WHERE release_year<2022
GROUP BY release_year, type
ORDER BY type, release_year;
```

I chose to simply illustrate for [14] in the histogram below.



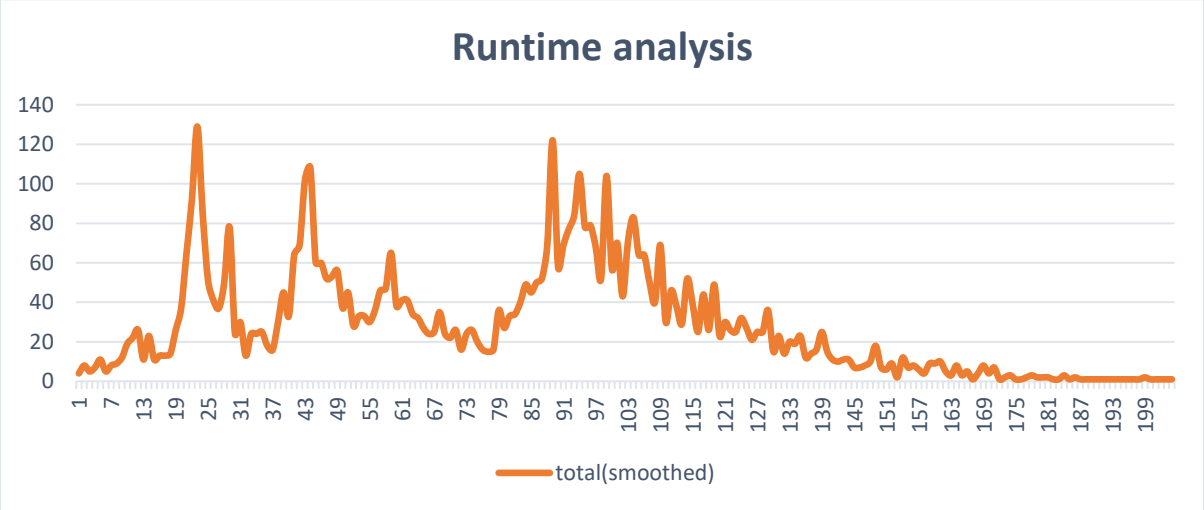
As you can see, much of the content was released in the past decade.

6. Analysis of runtimes

I did the same for runtimes.

[16]

```
SELECT runtime,COUNT(id) FROM titles
WHERE runtime IS NOT NULL AND runtime>0
GROUP BY runtime
ORDER BY runtime;
```



Unsurprisingly, there are big spikes around the 20- and 40-minute marks as these are common runtimes for tv shows. You can also see that most movies have a runtime of about 90-110 minutes before the runtimes slowly taper off.

7. What age groups are Netflix content geared towards

Though any insights derived from the age certification data is unreliable considering 45% of the data is null, I wanted to go ahead and get some insights anyways since this is just for practice.

I can simply find the count for each age certification with the query:-

[17]

```
SELECT age_certification, COUNT(*) FROM titles
WHERE age_certifIcation IS NOT NULL
GROUP BY age_certification
ORDER BY COUNT(*) DESC;
```

"TV-MA"	841
"R"	575
"TV-14"	470
"PG-13"	440
"PG"	246
"TV-PG"	186
"G"	131
"TV-Y7"	112
"TV-Y"	105
"TV-G"	76
"NC-17"	14

I noticed a lot of different but seemingly similar age certifications, so I did some research to see if I can simplify them. I found the following:

TV-Y7 -- suitable for kids over 7
G -- general
NC-17 -- for adults only
TV-Y -- suitable for preschool children
PG -- parental guidance suggested
TV-G -- general
TV-PG -- parental guidance suggested
TV-14 -- not suitable for under 14
PG-13 -- not suitable for under 13
R -- for adults only
TV-MA -- for adults only

I felt that I can distill these certifications into the following 4 categories:

Preteens, Teens+, Adults and General.

I felt that this insight was more valuable in this form, so I decided to create a new column for these categories. This way, this insight can be retrieved in a single query.

[18]

```
ALTER titles
ADD COLUMN age_cat VARCHAR(20)
```

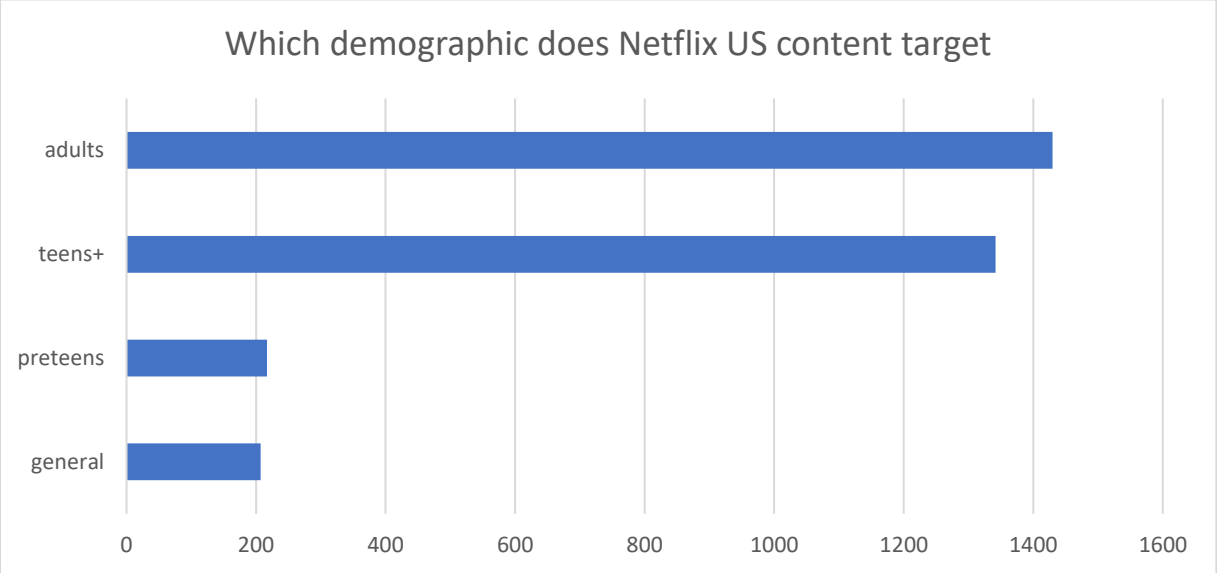
[19]

```
UPDATE titles
SET age_cat = CASE
    WHEN (age_certification='TV-Y' OR age_certification='TV-Y7') THEN
'preteens'
    WHEN (age_certification='PG' OR age_certification='TV-PG'
          OR age_certification='TV-14' OR age_certification='PG-13') THEN
'teens+'
    WHEN (age_certification='TV-MA' OR age_certification='R'
          OR age_certification='NC-17') THEN 'adults'
    WHEN (age_certification='TV-G' OR age_certification='G') THEN 'general'
    ELSE NULL
END
```

I then used the new column to find the number of titles in each of these categories.

[20]

```
SELECT age_cat, COUNT(*) FROM titles
WHERE age_cat IS NOT NULL
GROUP BY age_cat
ORDER BY COUNT(*) DESC;
```



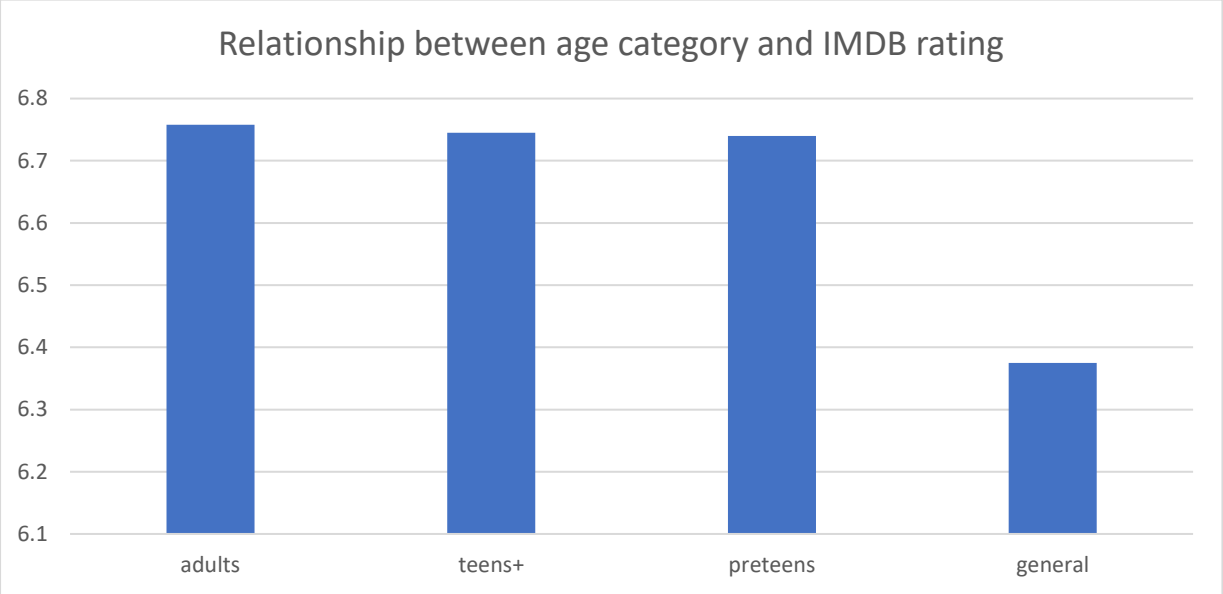
These results show that the vast majority of Netflix content possess mainly the equivalent of an "R" or a "PG-13" rating. To see if this is intentional, this should be compared with the age certification data of all content released. Once again, it should be noted that about half of the titles have missing certifications.

8. Relationship between age category and ratings

Since making the age_cat column, I was curious to investigate the relationship between age category and IMDB ratings.

[21]

```
SELECT age_cat,AVG(imdb_score) FROM titles
WHERE imdb_score IS NOT NULL AND age_cat IS NOT NULL
GROUP BY age_cat
ORDER BY AVG(imdb_score) DESC;
```



As illustrated in the chart above, the ratings are similar across the categories of adults, teens+ and preteens but severely drops off for the general category. Perhaps this is because those movies are not targeting any specific demographic.

9. Relationship between imdb score and tmdb score

I noticed that there were 2 ratings from 2 different rating websites, IMDB and TMDB. I thought it'd be good to compare the two. I figured that the best way to do this would be with a box and whisker plot.

At first, I used the following syntax to obtain the relevant information needed to create a box and whisker plot.

[22]

```
SELECT
    type,
    ROUND(AVG(imdb_score),3) AS avg_imdb,
    MIN(imdb_score) AS IMDB_min,
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY imdb_score) AS "IMDB_q1",
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY imdb_score) AS IMDB_median,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY imdb_score) AS "IMDB_q3",
```

```
    MAX(imdb_score) AS IMDB_max,
    ROUND(AVG(tmdb_score),3) AS avg_tmdb,
    MIN(tmdb_score) AS TMDB_min,
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY tmdb_score) AS "TMDB_q1",
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY tmdb_score) AS TMDB_median,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY tmdb_score) AS "TMDB_q3",
    MAX(tmdb_score) AS TMDB_max
FROM titles
WHERE imdb_votes>(SELECT PERCENTILE_CONT(0.05) WITHIN GROUP (ORDER BY
imdb_votes) FROM titles)
AND tmdb_popularity>(SELECT PERCENTILE_CONT(0.05) WITHIN GROUP (ORDER BY
tmdb_popularity) FROM titles)
GROUP BY type;
```

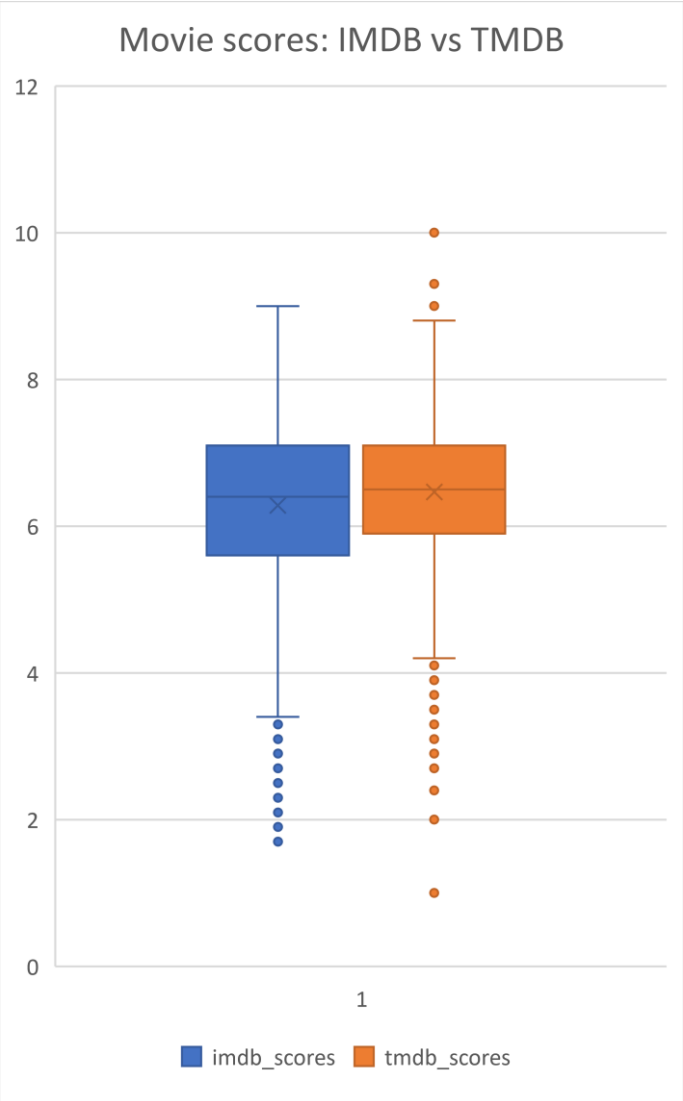
This turned out to be unnecessary as it's much better to extract the relevant data to excel to directly create the box and whisker plot. This turned out to be useful anyways as I learnt how to obtain percentiles in SQL.

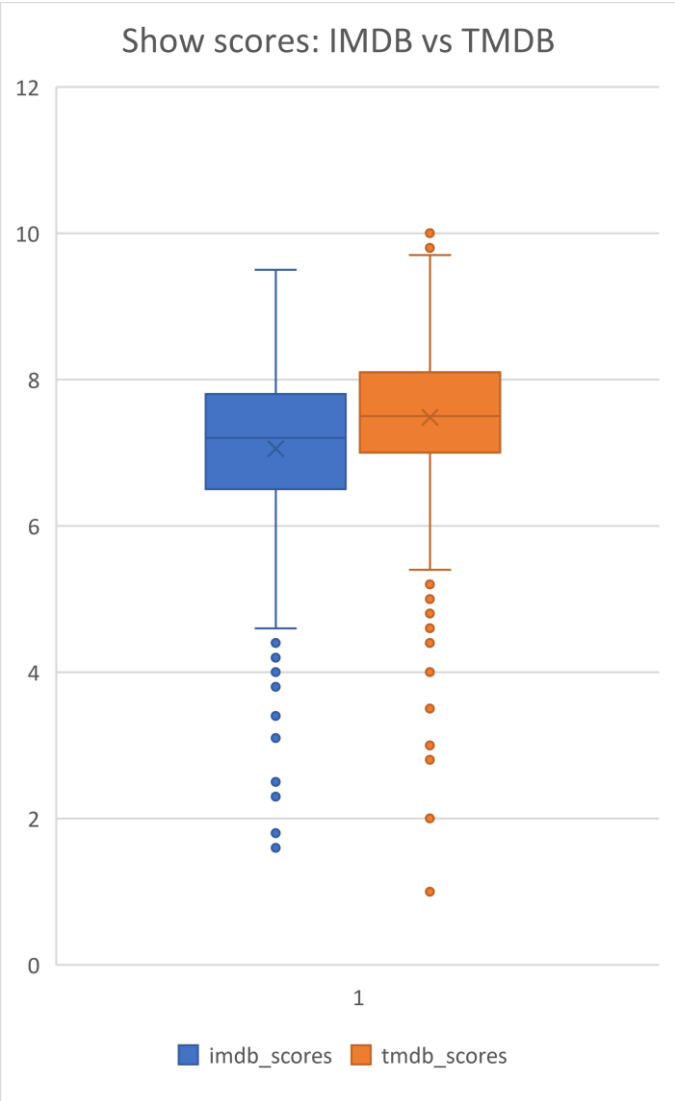
To extract the relevant data, I used the following syntax:

[23]

```
SELECT
type,
imdb_score,
tmdb_score
FROM titles
WHERE imdb_votes>(SELECT PERCENTILE_CONT(0.05) WITHIN GROUP (ORDER BY
imdb_votes) FROM titles)
AND tmdb_popularity>(SELECT PERCENTILE_CONT(0.05) WITHIN GROUP (ORDER BY
tmdb_popularity) FROM titles)
-- Removing scores with the lowest 5% of votes since these votes might not
be an accurate representation
ORDER BY type;
```

I used the percentile function to remove the scores with the number of votes in the bottom 5th percentile. This is because scores can be unreliable if there aren't enough votes to truly represent the consensus.





As illustrated in the two box and whisker plots above, TMDB has a higher average and median ratings for both movies and shows, more significantly so in the case of shows. Another thing to note is that although TMDB has more extreme scores on both the high and low ends, their scores lie within a much tighter range.

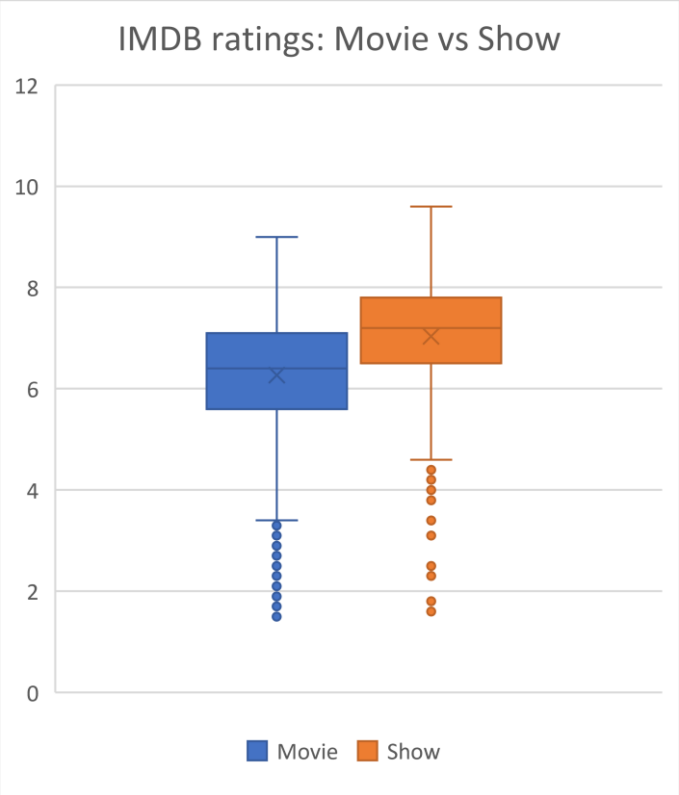
Another insight to note is that the IMDB averages are lower than the medians, indicating that their scores skew a little more towards the downside. In other words, this could indicate that reviewers are harsher on bad movies than they are appreciative of good movies. This is also illustrated with IMDB only having anomalies on the low end but none on the high end. On the other hand, TMDB's average rating is in line with the median.

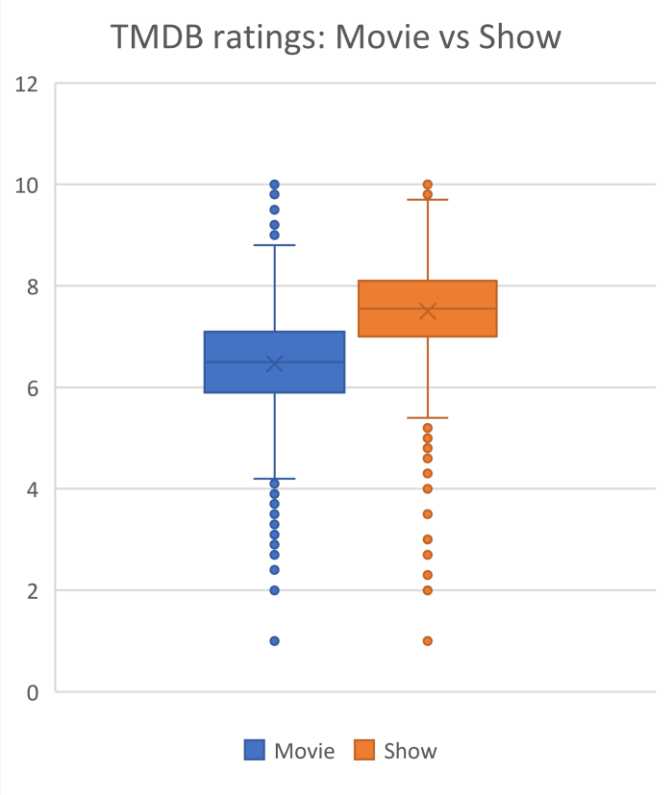
10. Average ratings of shows vs movies

I wanted to also compare the ratings between shows and movies on both IMDB and TMDB. Once more, I'll use a box and whisker plot to descriptively illustrate the difference in ratings.

[24]

```
SELECT
type,
imdb_score,
FROM titles
WHERE imdb_score IS NOT NULL
AND imdb_votes>(SELECT PERCENTILE_CONT(0.05) WITHIN GROUP (ORDER BY
imdb_votes) FROM titles)
-- Removing scores with the lowest 5% of votes since these votes might not
be an accurate representation
ORDER BY type;
```





In both IMDB and TMDB, shows are rated higher than movies, significantly more so on TMDB. In fact, the 25th percentile of shows is almost higher than that of the 75th percentile of movies. This could create a distorted perception on how good/bad a movie or show really is. Therefore, IMDB may provide a more balanced review in this aspect but the difference in rating is still not ideal. Or perhaps, the long-form nature of shows might allow to have more in-depth story-telling; translating to better viewer experiences than movies.

11. Relationship between genres and ratings

This was the question that I was most curious about personally. Ironically, it was also the most complex one to answer; due to the data format that the genre was stored as.

I have long felt that IMDB ratings are more based on whether films are "Oscar-worthy" rather than how enjoyable the movie really is. As a result, it was my perception that genres like "comedy" and "horror" have lower ratings in general compared to "drama", for example.

Moving on to the data analysis, there were a few issues. First, the data is displayed as if it is an array, i.e. [‘drama’,’crime’,’romance’]. However, I had stored it as a string when I created the table. Another issue was that most of the titles had multiple genres. In order to make a comparison, I had to isolate the genres.

I knew that I had to convert the data type from a string to an array and then explode them into multiple rows, so that there is one genre per row. Hence, I decided to create a new table as I felt that I needed to manipulate the data quite a bit to extract the genre. This way, the database can remain untouched and I'm not fearful of making a mistake.

[25]

```
SELECT id,genres,imdb_score
INTO genre
FROM titles
WHERE imdb_votes>(SELECT PERCENTILE_CONT(0.05) WITHIN GROUP (ORDER BY
imdb_votes) FROM titles)
AND genres IS NOT NULL
AND type='MOVIE';
```

I’ve now created a new table called genre with the columns id, genres and imdb_score (with the top 95th percentile of votes).

Then, I wanted to remove the outer square brackets "[]" from the table, which I achieved with the following syntax.

[26]

```
UPDATE genre
SET genres=BTRIM(genres,'[]');
```

Now the data in the table has been converted as shown here.

From "['drama','crime','comedy']" to 'drama','crime','comedy'

Next was converting the data type. At first, I managed to convert it, but it would take the whole list as a single value, i.e., not separating them into each genre.

After some time, I eventually figured out that I can use the comma as a separator with the unnest function.

[27]

```
SELECT unnest(string_to_array(genres, ', ')) AS genre
FROM genre;
```

I then had the problem of how I would use the exploded rows to retrieve the avg imdb scores for each genre. I first tried to use it as a subquery since it's a select statement. However, functions like UNNEST aren't allowed as a subquery. I then tried to use UPDATE to modify the table further using the statement below, but again, all this retrieved was an error message.

[28]

```
UPDATE genre
SET genres = UNNEST(string_to_array(genres,', '));
```

Eventually, I recalled from my SQL course that you can create a virtual table using a VIEW function and it worked!

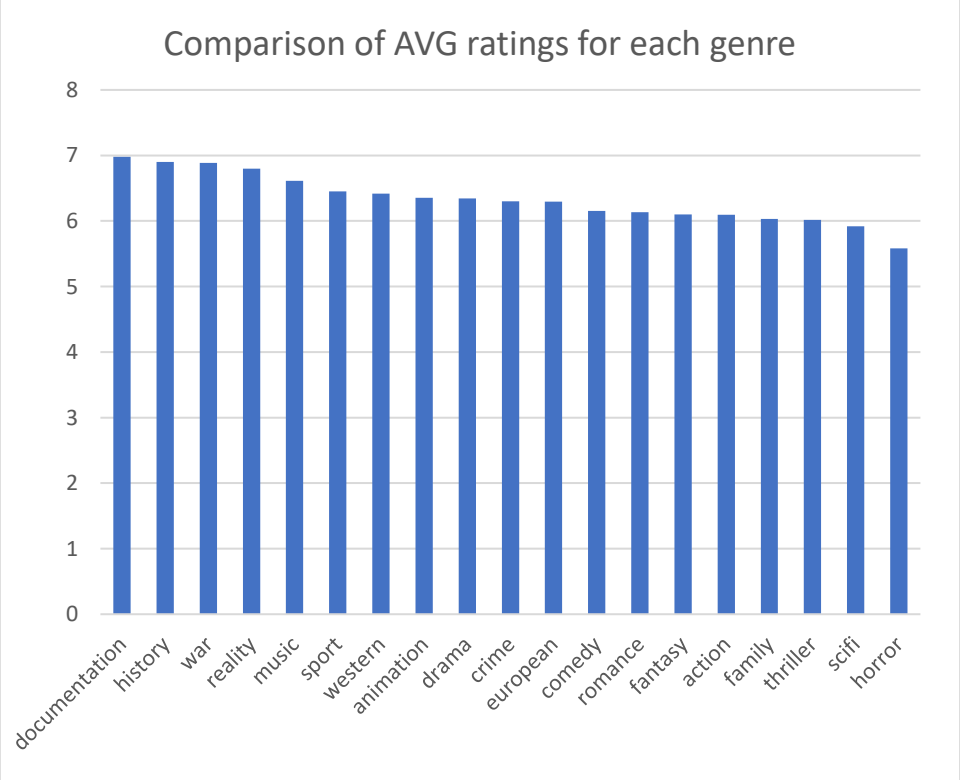
[29]

```
CREATE VIEW sta_genre AS
SELECT ID,imdb_score, UNNEST(string_to_array(genres, ', ')) AS genre
FROM genre;
```

I figure that if I can create a temporary table as such, I must be able to modify the table, but alas I was able to successfully explode the table such that each row contains one genre. I then used this table to retrieve the avg imdb scores for each genre:

[30]

```
SELECT genre,ROUND(AVG(imdb_score),3) AS avg_imdb_score
FROM sta_genre
GROUP BY genre
ORDER BY AVG(imdb_score) DESC;
```



Interestingly, the top 4 genres in terms of average IMDB ratings, are all based on “true stories”, perhaps it’s because true stories trigger a more emotional response. On the other end of the spectrum, “thriller”, “scifi” and “horror” are arguably the genres that require the most imagination. Horror is something I’ve anecdotally noticed as a genre that receives exceptionally low ratings, and it rings true on this chart.

Conclusion

Throughout the process of doing this project, I felt more and more comfortable using SQL programming language to query data. It was rewarding to be able to instantly see the results of my inputs and this project has definitely reaffirmed my decision to pursue learning data science. I look forward to my next project, perhaps with more complex Data Visualisation tools.