A blue text on a black background

Description automatically generated

**SCTP JDE04**

**ZetaZenith**

# **Interim Project**

**Netflix Titles (Movies/TV Shows)**

**Submitted by:**

|  |
| --- |
| **Team Members** |
| **SITI RAHAYU BINTE JOHARI** |
| **SHUI HUI YAN, TAMMIE SARAH** |
| **OOI JUN SHENG** |

**Submission Date: 29/08/2024**

|  |
| --- |
| **Table Of Content** |

[**1. Background** 2](#_Toc175824019)

[**2. Problem Statement** 2](#_Toc175824020)

[**3. Assumptions** 3](#_Toc175824021)

[**4. Objectives** 3](#_Toc175824022)

[**5. Use Cases** 3](#_Toc175824023)

[**6. Source of Data** 6](#_Toc175824024)

[**7. Tools Used** 7](#_Toc175824025)

[**8. Methodology** 9](#_Toc175824026)

[**9. Analysis** 27](#_Toc175824027)

[**10. Challenges/Limitations** 30](#_Toc175824028)

[**11. Solutions** 31](#_Toc175824029)

[**12. Conclusion** 34](#_Toc175824030)

[**13. References** 36](#_Toc175824031)

[**14. Appendix** 37](#_Toc175824032)

[Section 1. 3NF Tables 37](#_Toc175824033)

[Section 2. IMDB Listicles Scraper 40](#_Toc175824034)

## **1. Background**

The purpose of this document specifies the overall requirements for the interim project. It explains the outline of the development process within the development environment and defines the scope of the project including problem statement, assumptions, objectives, use cases, the source of data, tools used, methodology for the ETL process, analysis of data, challenges or limitations we have encountered, how we circumvent it and our conclusion.

## **2. Problem Statement**

Netflix, as a leading global streaming service, continuously expands its content library to cater to a diverse and growing audience. To stay ahead of the competition, it is essential for Netflix to understand the distribution of genres within its catalog and how these genres have evolved over time. By analyzing genre trends, Netflix can make informed decisions on content acquisition, production, and recommendation strategies.

## **3. Assumptions**

1. The Netflix dataset consists of popular titles (Movies/TV shows) from 2018 to 2021.
2. There is existing demand for the titles if it is already streaming on Netflix.
3. Missing dates (NaT) found for date\_added assumed to be “2018-12-31”.
4. Missing ratings for show\_id: 5542, 5795, 5814 assumed to be the same “TV-MA”.

## **4. Objectives**

The primary objective of this report is to analyze the distribution and trends of genres in Netflix's catalog over time. The analysis aims to answer key questions such as:

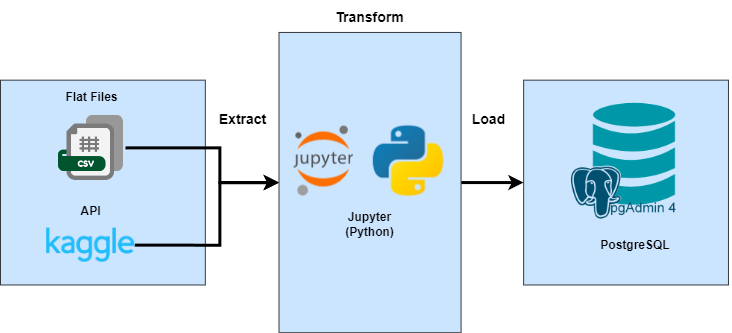
* Which genres have become more or less popular over the years?
* What are the most common genres in Netflix's catalog?
* How do genre preferences vary across different countries?
* Which countries are producing the most diverse set of genres?
* How can these insights inform Netflix’s content strategy, including content acquisition, production, and regional marketing efforts?

## **5. Use Cases**

The Jupyter Notebook will act as the centralized software to develop the extract-transform-load (ETL) process of this entire project.

Python modules will be imported to facilitate in extracting data from external sources, transform the data and finally loading them into PostgreSQL (PgAdmin).

The data architecture model for extract, transform and load (ETL) process:



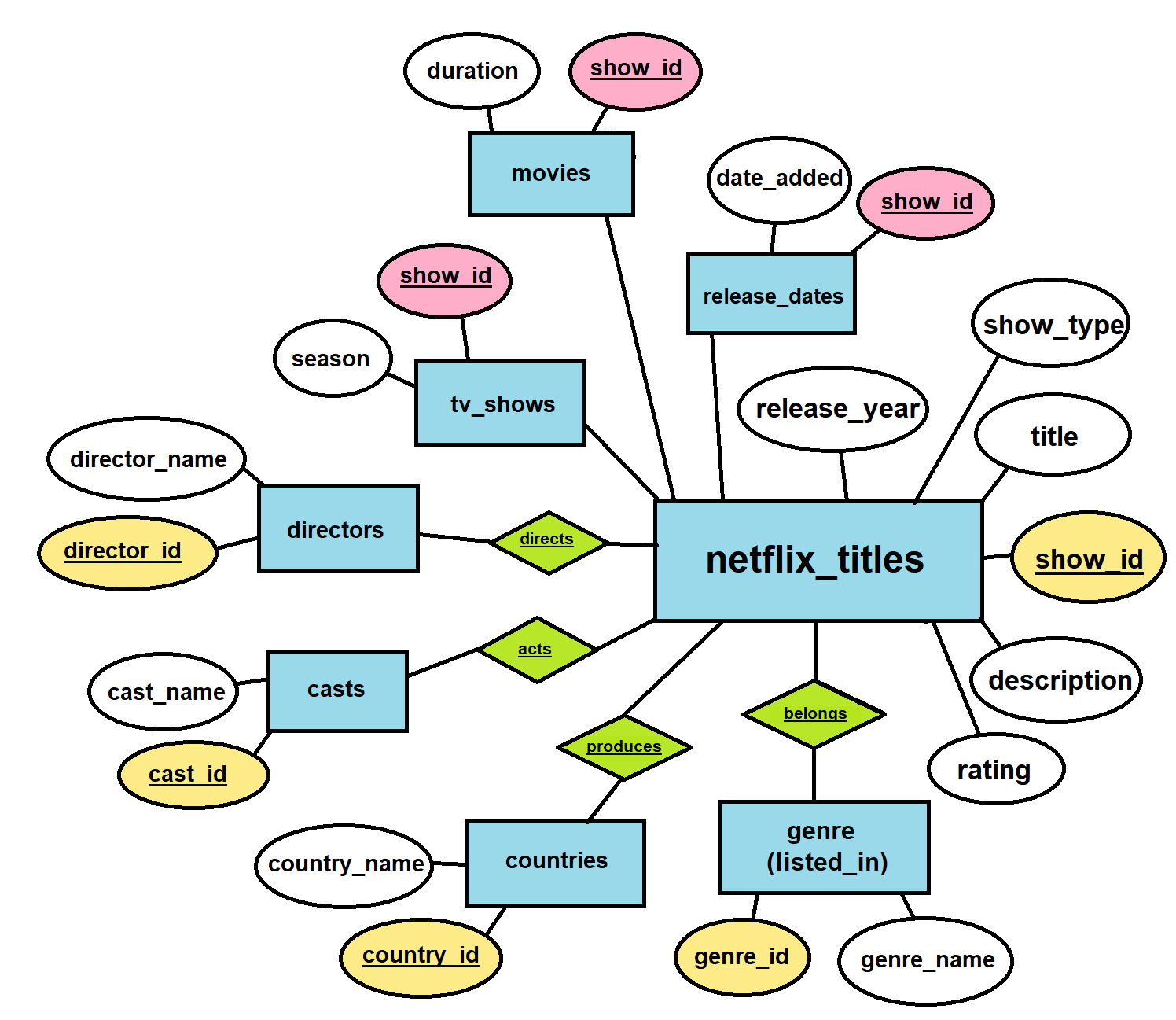
Data Extract:

* Data will be extracted from flat files and Kaggle API.
  + Kaggle API is used to extract data sets labelled Netflix shows on Kaggle’s repository. The dataset will be in flat file format (.csv).

Data Transform:

* The dataset will then be transformed within the Jupyter Notebook environment, through:
  + Converting it into a pandas DataFrame.
  + Converting it to appropriate data types (dtypes).
  + Replacing “NA”, “NaT”, NULL and missing values.
  + Checking to ensure all fields are filled correctly.
  + Perform normalization up till 3rd Normal Form.
    - With proper junction tables created for many-to-many (N:M) relationships.
  + Data persistence by exporting .csv as backups once 3rd Normal Form is done and before loading into the database.
* The finalized 3rd Normal Form tables will be loaded into PostgreSQL (PgAdmin4)
  + New database “netflix\_db” will be created if it does not exist.
  + All tables (including junction tables) will be created along with the appropriate data type fields, primary and foreign key constraints.
  + Append into “netflix\_db” upon loading to 3rd Normalised DataFrames.
  + For data validation, the database will be queried after each creation and loading of table to ensure that the records in DataFrames matches the newly populated tables within “netflix\_db”.

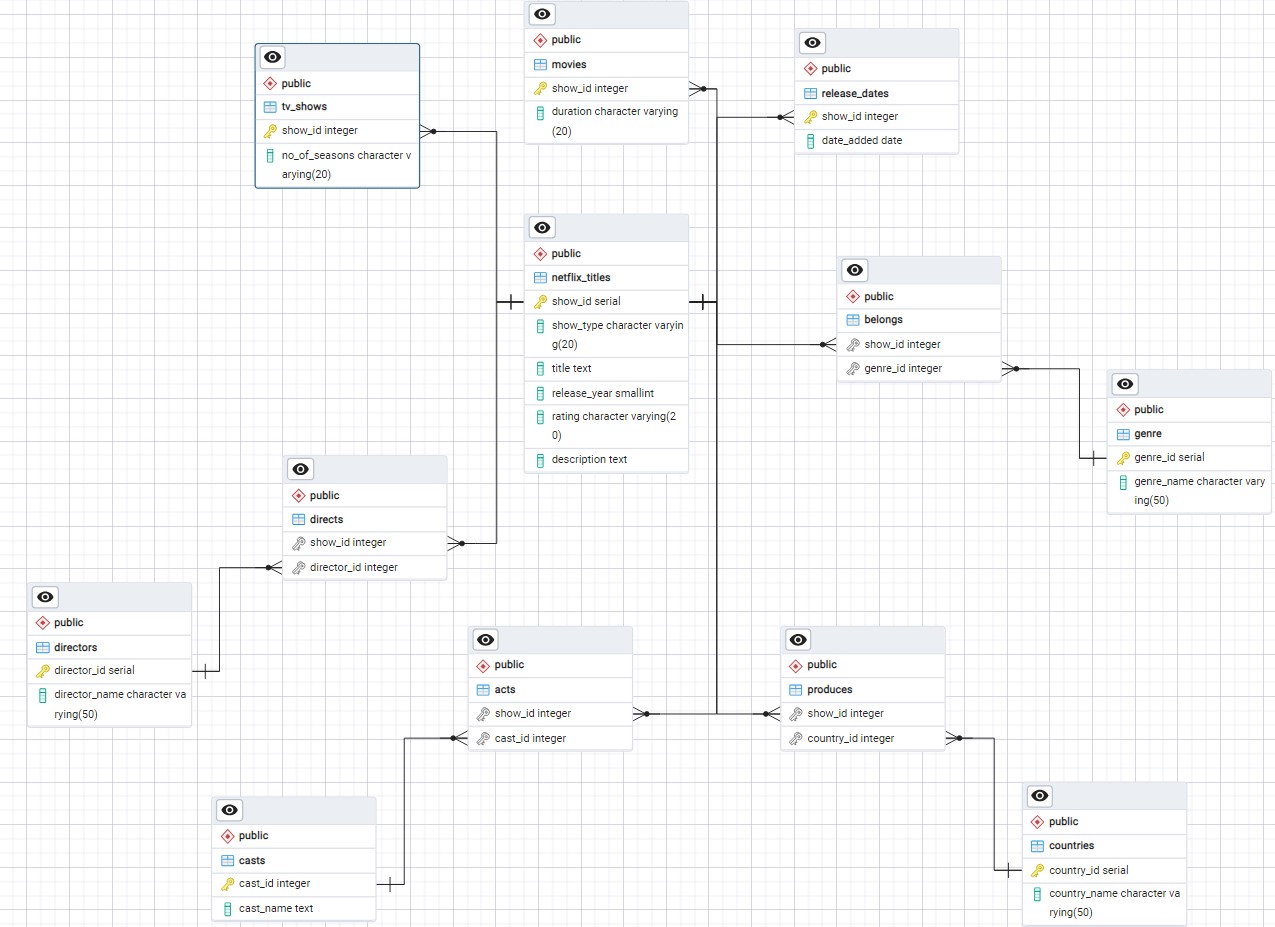
The finalised 3rd Normalised conceptual diagram looks like this:



Data Load:

* The 3rd Normalised DataFrames will be loaded into PostgreSQL within the Jupyter Notebook:
  + Using `SQLAlchemy` and `psycopg2` to perform the following functions:
    - Create database `netflix\_db` in PostgreSQL.
    - Create 12 tables within `netflix\_db` in PostgreSQL.
    - Set foreign key and primary constraints.
    - Load the 3rd Normalised DataFrames into the 12 tables via Notebook.

And upon loading into PostgreSQL database with referential integrity and constraints, the entity relationship diagram will be autogenerated:



## **6. Source of Data**

The data we have decided on will be “[Netflix Movies and TV Shows](https://www.kaggle.com/datasets/shivamb/netflix-shows)” published by Shivam Bansal on Kaggle. It was last updated 3 years ago but has popular titles from movies and tv shows, streamed on Netflix and up till the year of 2021.

The dataset also consists of the following columns:

|  |  |
| --- | --- |
| **Column Name** | **Description** |
| show\_id | Unique ID for every movie/tv show |
| type | A type of identifier – whether it is `movie` or `tv show` |
| title | Title of the movie/tv show |
| director | Director of the movie/tv show (can have one director or more) |
| cast | Actors involved in the movie/tv show (can have one actor or more) |
| country | Countries which the movie/tv show was produced (can have one or more) |
| date\_added | Date it was added on Netflix |
| release\_year | Actual release year of movie/tv show |
| rating | TV rating of the movie/tv show\* |
| duration | Total duration in minutes or number of seasons\* |
| listed\_in | Genre of the movie/tv show (can have one or more) |
| description | Summary of the movie/tv show |

*\*Note:*

* For rating it refers to the TV ratings not user or viewer ratings.
  + For example, `PG-13`, `TV-MA`, `TV-14` and so on.
* For duration, TV shows are generally given the value as the number of seasons.
  + For example, 3 season.
  + Whereas Movies are in the total runtime in minutes (e.g. 60 min).

## **7. Tools Used**

The main software used is Jupyter Notebook and coded in Python for the entire ETL process. PostgreSQL will be used at the very final stage of the pipeline to automatically generate the final entity-relationship diagram (ERD).

Extract

* import kaggle
  + To run API commands from Kaggle.
* import os
  + To run file check if the imported dataset from Kaggle exists on user’s path.

Transform

* Import pandas as pd
  + To perform standard DataFrame operations such as: cleaning, renaming, checking for null values, changing of data types, etc.
  + Importing and exporting of .csv files.
* pd.options.mode.chained\_assignment = None
  + This is to set “warnings” to not show up, as the nature of our work will cause many warnings to be flagged due to “chained assignments”.
  + This syntax essentially makes the notebook looks cleaner for the reader/user.

Load

* import psycopg2
  + psycopg2 is a PostgreSQL database adapter developed for Python.
  + It allows us to create, drop and modify tables in a pseudo-SQL environment but still retaining in our original working software.
  + We use it to create the database “netflix\_db”.
  + We also check for any existing instance of “netflix\_db”, so if there is already a table it will proceed to drop it before creating a new one.
* import SQLAlchemy
  + SQLAlchemy allow us to create connections to the local PostgreSQL client via a connection URI and can also be used to create tables with proper primary and foreign key constraints set.
  + It is also used to verify by reflecting all created tables to ensure that the tables within “netflix\_db” are created correctly.
  + It is also used to query the constraints from the information.schema table to verify that all the primary key and foreign key constraints are set correctly.
  + It is also used to load the DataFrames into the created tables via “to\_sql”.
  + Finally, as a last step of data validation, SQLAlchemy is used to query the database to ensure the number of records matches the DataFrame that was loaded into it.

Data Visualisation

* pandas
  + Data Manipulation: The `pandas` library was used extensively for data manipulation, including merging DataFrames, handling missing values, and performing group-by operations to prepare the data for visualization.
* matplotlib and seaborn
  + Data Visualization: `Matplotlib` and `Seaborn` were the primary libraries used for creating the visualizations. These libraries provided a wide range of customization options for the plots, ensuring clarity and visual appeal.
  + Customization: Titles, labels, color palettes, and figure sizes were customized to enhance the readability and interpretability of the visualizations.

Others

* PostgreSQL (PgAdmin4)
  + To verify that tables are all correct.
  + To auto generate the entity-relationship-diagram (ERD).
* Draw.io (https://app.diagrams.net/)
  + To draft out the Data architecture model.
* Documentations
  + From various Python modules to check how each function work: psycopg2, SQLAlchemy, Kaggle API, PostgreSQL and pandas

## **8. Methodology**

Data Cleaning

The dataset originally had the ‘show\_id’ as a string, which is wrong if we plan to normalize the table. To fix this, we remove the character ‘s’ in front of the ‘show\_id’ and then convert it into integer using .astype(int).



We then further change the format to int64 and apply the same methodology to ’release\_year’:



The rationale for this is because when we normalize the table there will be too many records, so having int64 essentially allows for more records instead of constraining to limited integer values.

|  |  |  |
| --- | --- | --- |
| **INT16** | **INT32** | **INT64** |
| Takes 2 bytes in memory | Takes 4 bytes in memory | Takes 8 bytes in memory |
| -32,768 to 32,767 | -2,147,483,648 to 2,147,483,647 | -9223372036854775808 to +9223372036854775807. |

Although the dataset we are dealing is small in comparison to commercial data, we are also managing a fairly small size project/environment, so having int64 provides a piece of mind for later.

We also isolated the columns ‘show\_id’ and ‘title’ into a new DataFrame for each of them. To count the number of unique entries and ensure no duplicated entries.

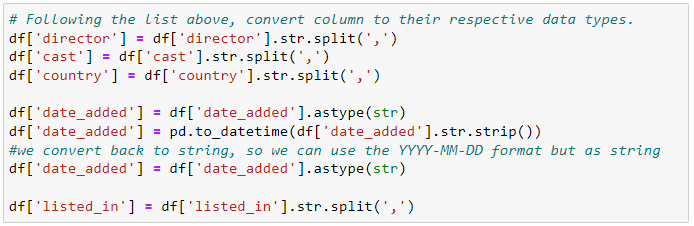
This was done in the following steps:

1. Isolate DataFrame into titles and show\_id respectively using df[[‘col\_name’]]
2. Use df.info() to count entries (8807).
3. Use df.drop\_duplicates() to drop duplicated entries.
4. Run df.info() again to count entries (8807).

|  |  |
| --- | --- |
| **Before .drop\_duplicates()** | **Before .drop\_duplicates()** |
|  |  |
| **After .drop\_duplicates()** | **After .drop\_duplicates()** |
|  |  |

The result we have gotten for both titles and show\_id indicates that they retain 8807 entries.

The data types for each other columns were also changed to string format via .str.split to easily filter out missing or NULL values and also for using the .explode() method when we atomize the columns with multiple values such as directors, casts, countries and listed\_in.



For checking of NULL values, we first use df.isnull.sum():

A screenshot of a computer

Description automatically generated

Based on some basic reasoning, we have decided on the following to clean up NULL values:

1. director – should be listed as “Unknown”
2. cast – same as director
3. country – same
4. rating – put as “NA”

The rest such as duration will be dealt with later (as there are minutes and number of seasons).

A computer code with text

Description automatically generated with medium confidence

And running a second check using df.isnull.sum():

A screenshot of a computer

Description automatically generated

And then for duration, we will run it separately:

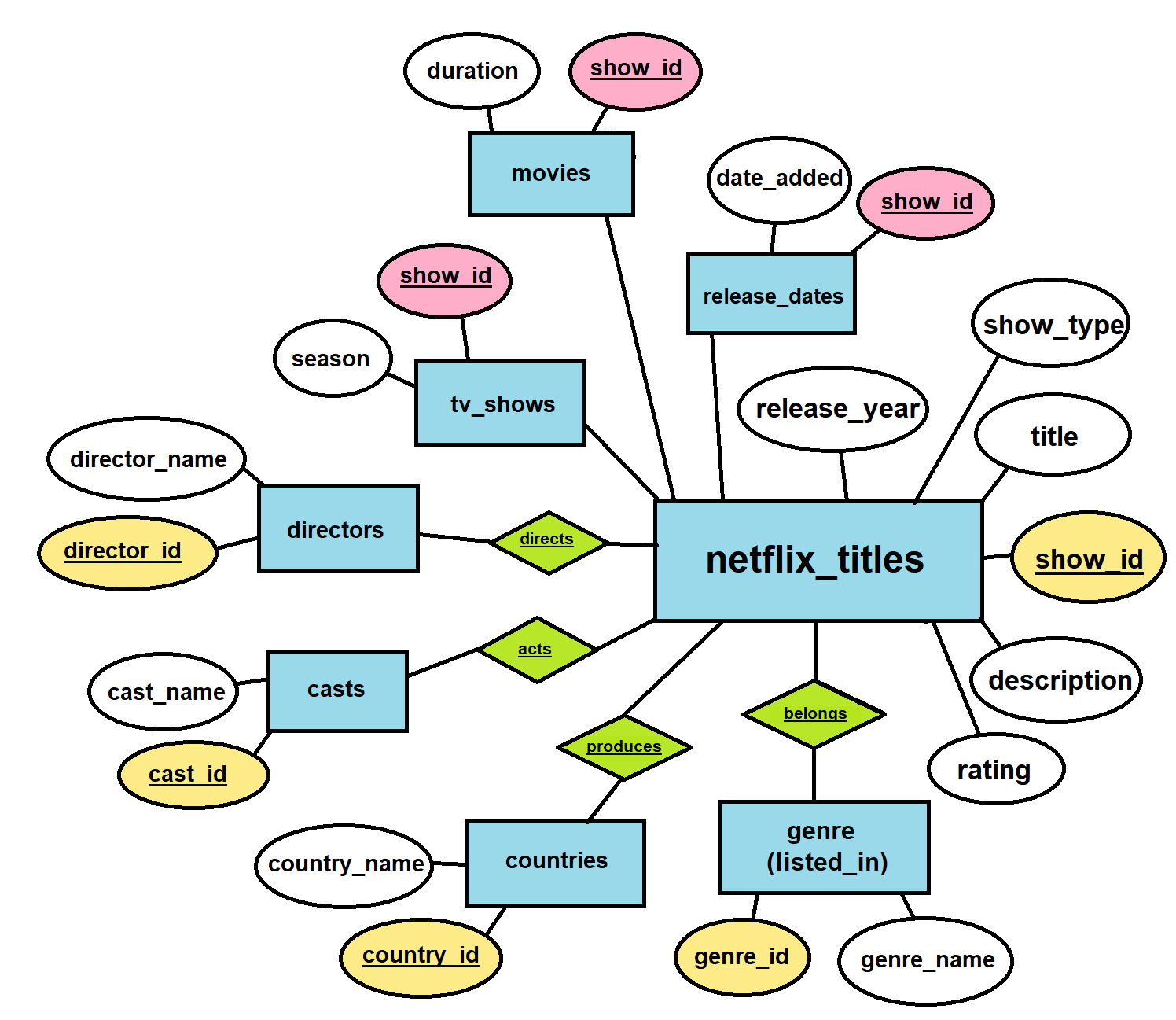
A screenshot of a computer

Description automatically generated

We then spotted that the rating were actually supposed to be duration values instead, so it was swapped over and the rating for show\_id ‘5542’, ‘5795’ and ‘5814’ is set to TV-MA based on Google Search results for the show\_id(s) as mentioned.

Normalisation (3rd NF)

After cleaning the table and coming up with the conceptual diagram:



We will have to split the cleaned DataFrame into 12 tables as shown on the diagram, which are:

1. netflix\_titles
2. movies
3. tv\_shows
4. release\_dates
5. directors
6. directs
7. casts
8. acts
9. countries
10. produces
11. genre
12. belongs

The reason why there are 12 tables is because we have taken into consideration 1NF and 2NF to reach 3NF.

A screenshot of a computer

Description automatically generated

Firstly, to satisfy 1NF, the records must have atomic values, so we will have to split genre, directors and casts into their respective tables, as these columns hold multiple values in a single cell.

A diagram of a television show

Description automatically generated

For 2NF, there must be no partial dependencies, meaning that all non-key attributes must depend on the primary key, which is why movies and tv\_shows have their own tables, as the duration or number of seasons columns depends on the type of the show (whether it is movie or tv).

And finally, to reach 3NF, all the non-key columns must be directly dependent on the primary key, which explains why the netflix\_titles have attributes: release\_year, show\_type, title, rating and description as they all depend on the primary key: show\_id and nothing else.

Similarly, the 3NF concept is applied to the other DataFrames by including a junction table too.

To translate this logic to Python, we perform manual splitting of the tables one by one, such as using the method .explode() on tables with multiple values:

A screenshot of a computer program

Description automatically generated

We also made sure they are no duplicated director names, by remove white spaces or padding via .str.strip() and change all cases to upper via .str.upper() and dropping the duplicates\* like so:

A screenshot of a computer program

Description automatically generated

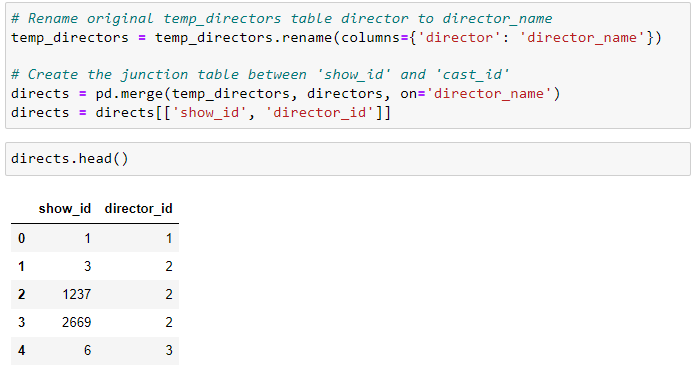
\**Note*: The rationale for .str.upper() is because some actors will not be flagged/detected by pandas as duplicates, if the alphabet cases are different. For instance, JuJu, JuJU, JUJU, juju are all the same but not flagged as duplicates!

After that we assign a new unique id to each director called `director\_id` using the index + 1:

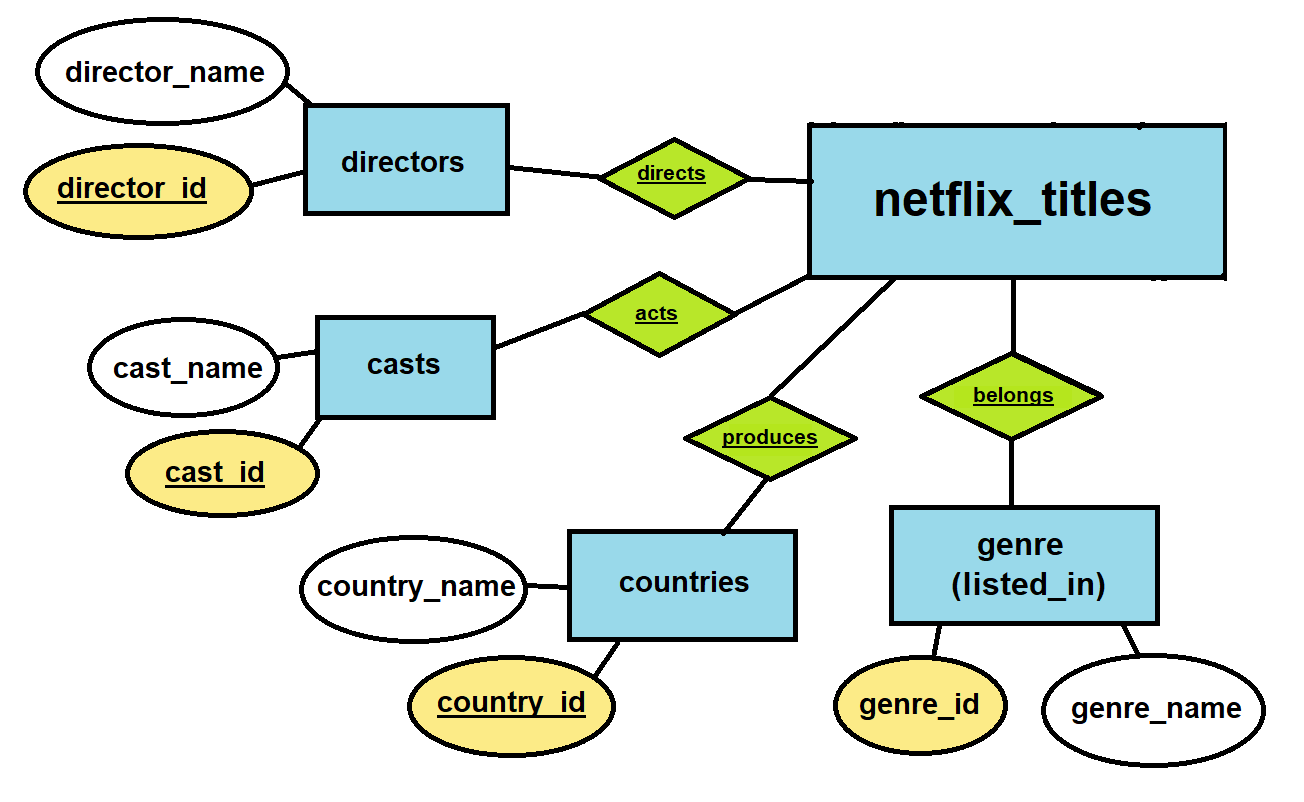
A screenshot of a computer

Description automatically generated

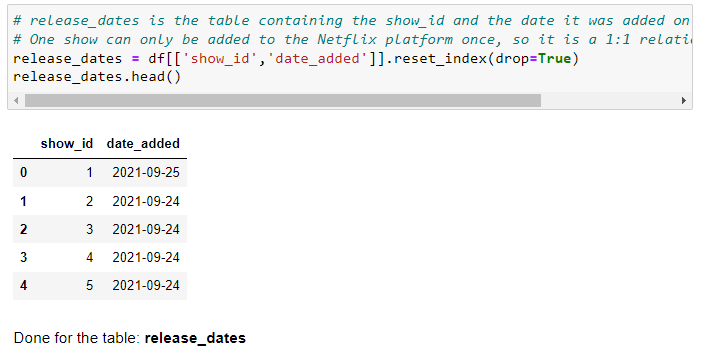
And then, we perform a merge with the old DataFrame to get the junction table, for example in this case, while we create the directors table with ‘director\_id’ and ‘director\_name’. We can perform a merge with the old table to get the junction table, to get the results of only: ‘show\_id’ and ‘director\_id’ which are both to be used as foreign keys later.



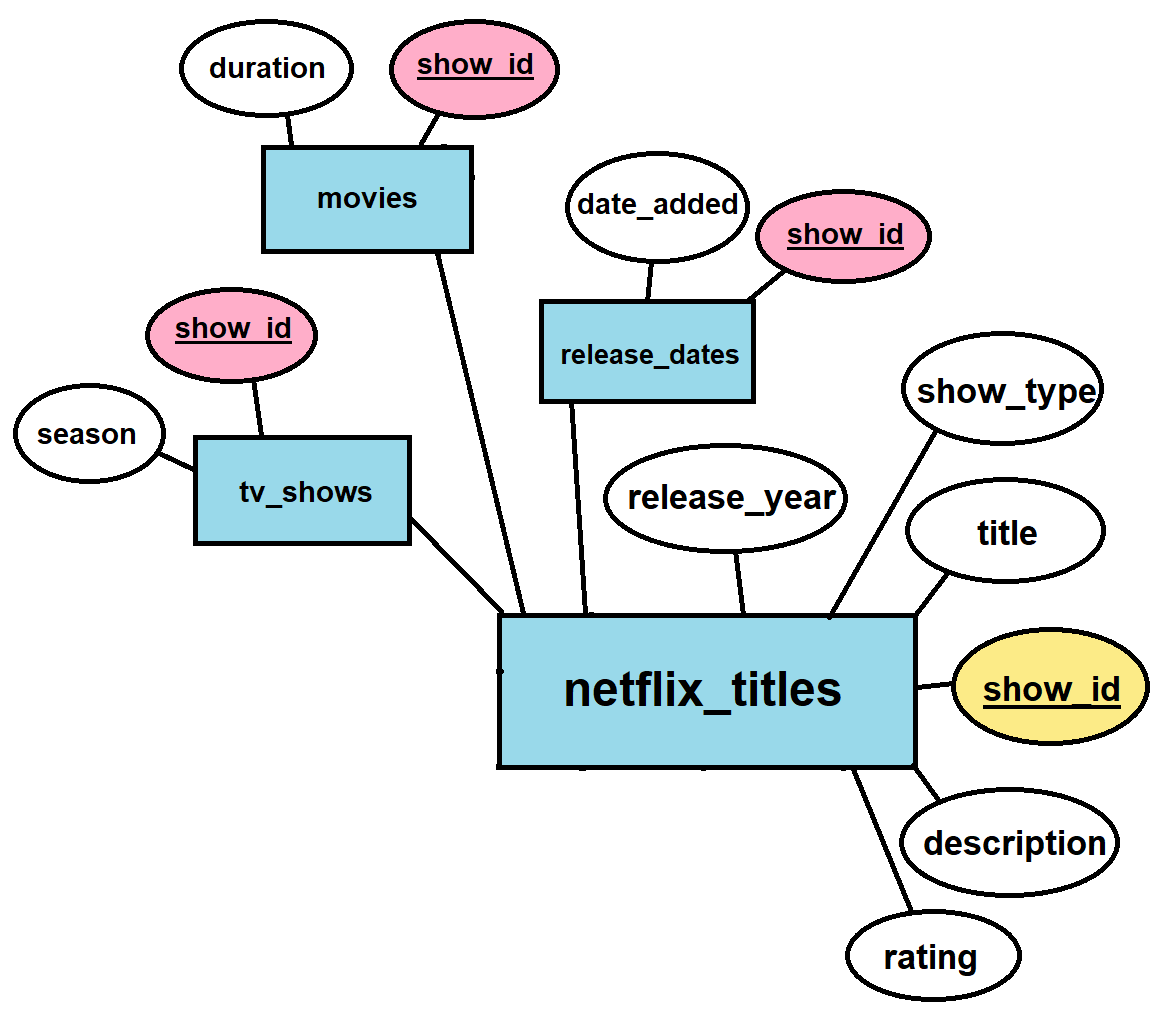
This process, is repeated for the tables with N:M relationships, such as directors, casts, countries and genres:



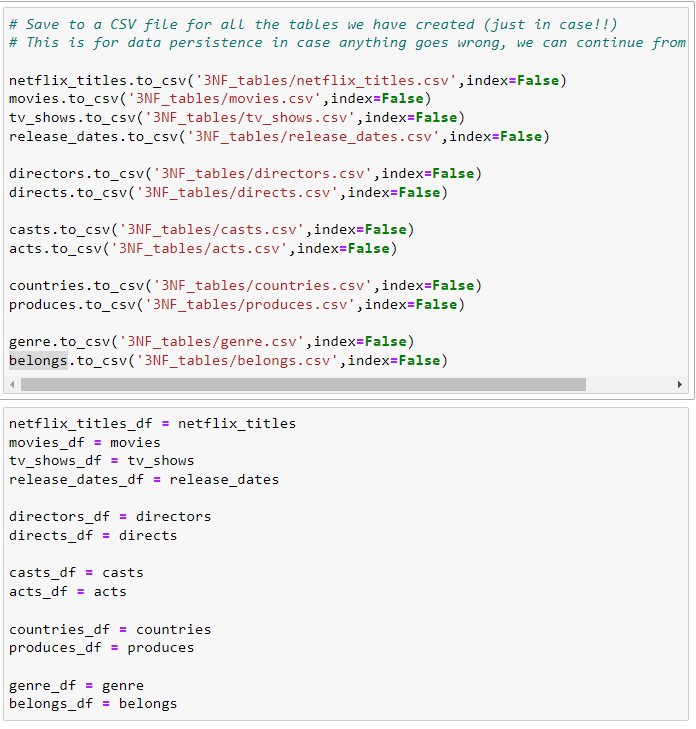
As for the 3 other tables: movies, tv\_shows and release\_dates with 1:1 relationship, it can be done by splitting up the original DataFrame respectively, assigning a unique id to each and renaming the columns. For example, for the table: release\_dates:



Applying the above coding logic to the movies and tv\_shows DataFrames, will fulfilled the tables with 1:1 relationship on the conceptual diagram (*see below*):



Once all 12 DataFrames has been created, we can export the DataFrames via pandas df.to\_csv to ensure data persistence and as a form of backup in case anything goes wrong within the pipeline.



And finally, renaming all 12 DataFrames with a suffix \_df:

A screenshot of a computer program

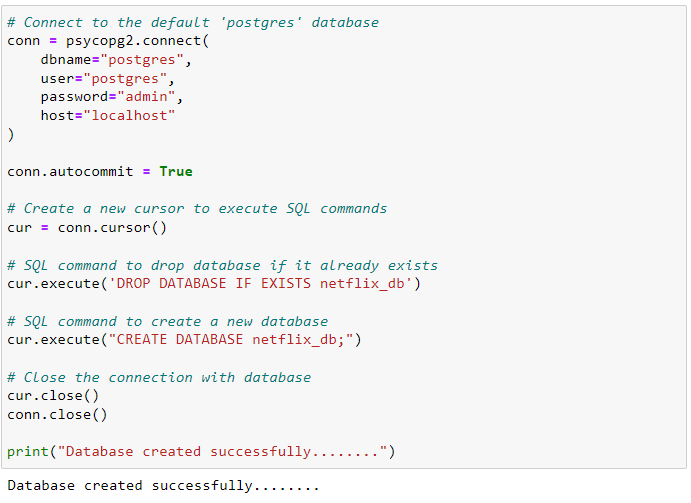
Description automatically generated

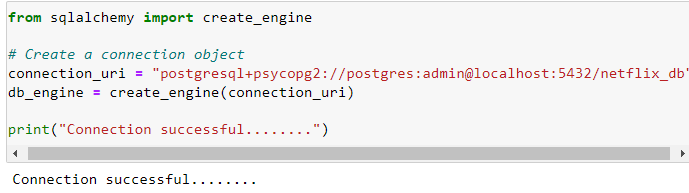
The reason why is because the same names will be used later for creating the tables for PostgreSQL, this is to prevent confusion when running the pipeline and maintaining code readability for any user.

Additionally, if the DataFrame and table names were the same when trying to load using SQLAlchemy, it will throw an error as it cannot be the same name, so adding the suffix at this point also make sense.

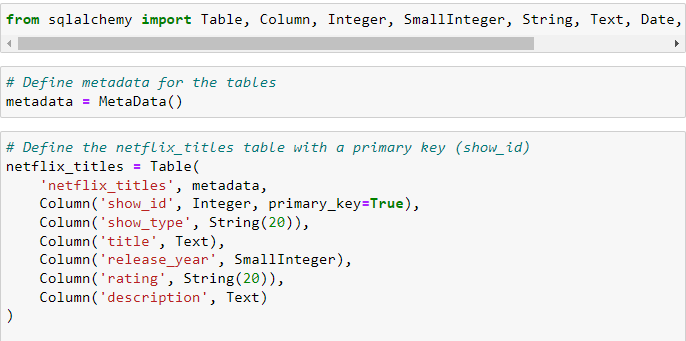
Loading to PostgreSQL Database

To load our 12 DataFrames into the PostgreSQL database, we can use a combination of psycopg2 and SQLAlchemy. First, a connection object can be created by creating a connection URI and engine via SQLAlchemy, which we can use to create cursors to create our database in PostgreSQL called ‘netflix\_db’.

After which, we can amend the connection URI to connect to our ’netflix\_db’ and creating an engine with the help of SQLAlchemy again:



This allows us to create our tables which can be loaded on to 'netflix\_db’ later. To create the tables, first, we define the metadata which will store all the table objects and the schema we defined for the tables.



For many-to-many relationship tables and their respective junction tables, this is how we establish the referential integrity and the primary key and foreign key constraints:



And then, by calling the metadata object and its associated function create\_all, PostgreSQL will create the tables within the ’netflix\_db’ if it has not been created:



For more information about the syntax, refer to SQLAlchemy 2.0 Documentation for constraints and indexes, see references [3].

Before proceeding to load the newly created tables with our normalized DataFrames, we also verified the constraints of our tables within the ‘netflix\_db’ database:

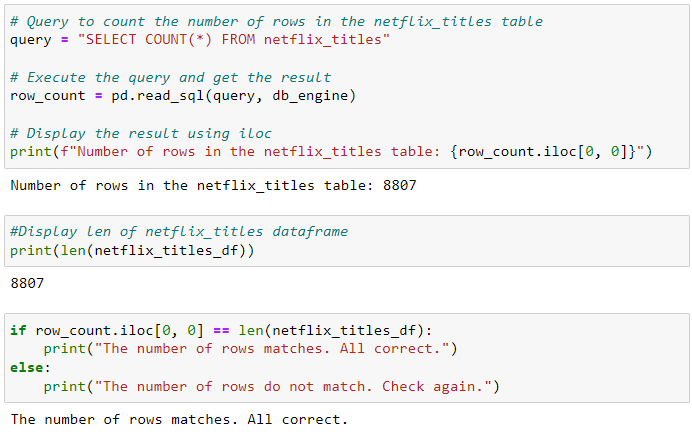


Syntax is taken from PostgreSQL documentation [4], refer to references for more information.

Finally, we can use SQLAlchemy’s df.to\_sql function to load the 12 normalised DataFrames into the tables created within the ‘netflix\_db’ database.

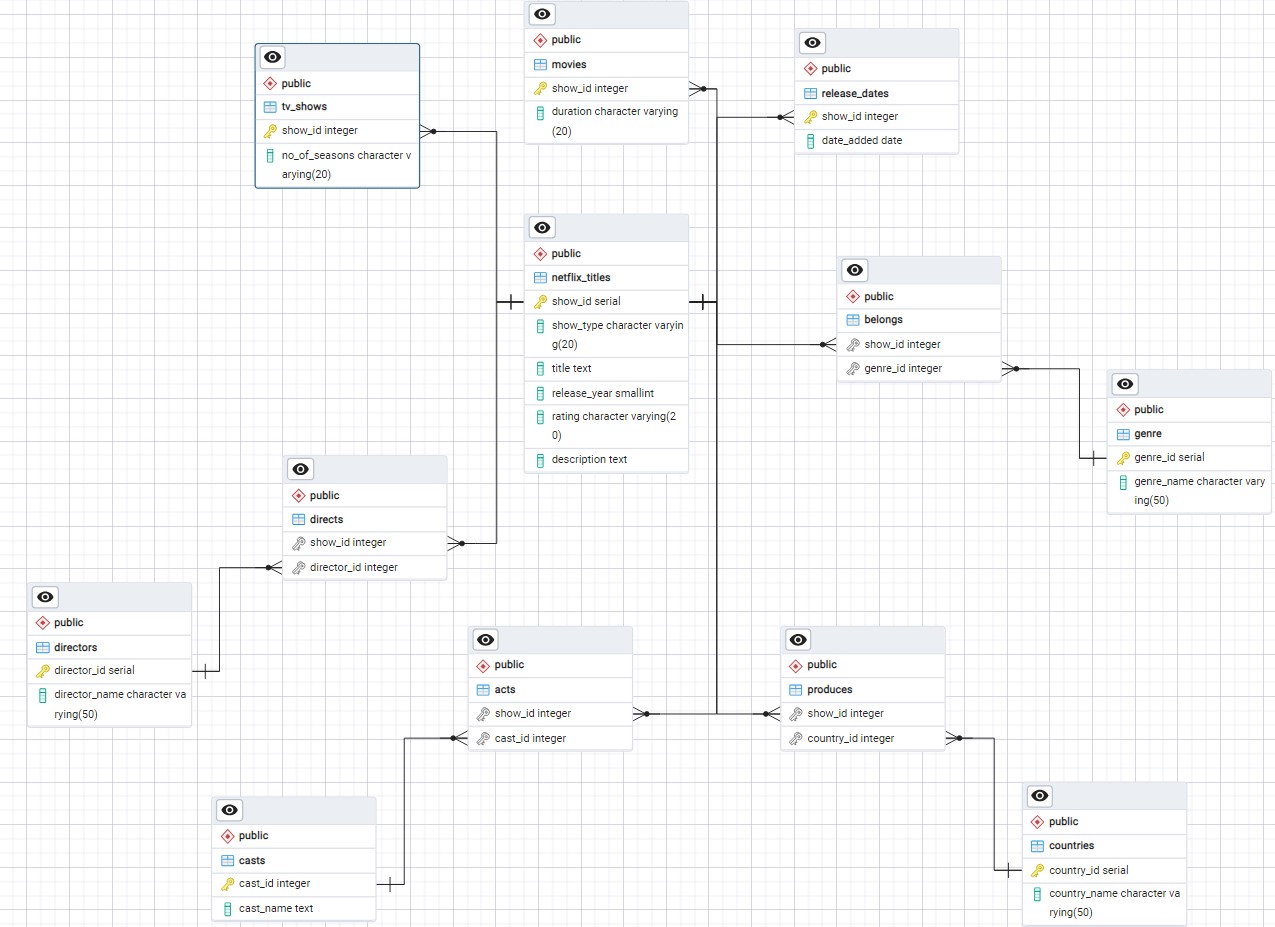
After loading, we perform data validation by ensuring that the record in the table matches the rows in their respective DataFrame by:

1. Querying all records from the table using SELECT count (\*)
2. Length of DataFrame using: len(df\_name)



This step will be repeated for the remaining 11 DataFrames and tables within the `netflix\_db` until all has been successfully loaded.

As a final form of check, PostgreSQL will autogenerate the entity-relationship-diagram (ERD) as shown:



Data Visualization

Merging DataFrames: DataFrames containing information on titles, genres, and countries were merged to create a comprehensive dataset (`netflix\_with\_genres\_and\_countries`) that could be used for detailed analysis.

 The following visualizations were created to explore and present the findings:

 1. **Bar Charts**:

    - Top 10 Most Common Genres: A bar chart was used to display the most frequently occurring genres in Netflix’s catalog. This visualization highlighted the genres with the highest number of titles.

   -Top Countries Producing Documentaries: Another bar chart was created to show the top countries that produce the most documentaries, providing insight into regional strengths in documentary filmmaking.

 2. **Clustered Bar Charts**:

    - Top Genres by Country (Top 10 Countries): Clustered bar charts were used to compare the popularity of different genres across the top 10 content-producing countries. This allowed for a clear comparison of genre preferences across regions.

 3. **Line Plots**:

    - Trends in Genre Popularity Over the Last Few Years: Line plots were utilized to track the changes in genre popularity over time. This visualization helped identify emerging trends and declining genres, offering insights into shifts in audience preferences.

## **9. Analysis**

The key objectives of this analysis are:

* To identify the most common genres across the entire Netflix catalogue.
* To analyse genre popularity trends over the last few years.
* To explore the regional differences in genre preferences and production.
* To understand which countries, contribute the most diverse set of genres.

The data visualisation focuses on the following:

* Top 10 Most Common Genres
* Top Genres by Country (Top 10 Countries)
* Top Countries Producing Documentaries
* Trends in Genre Popularity Over The Last Few Years
* Top 10 Countries Watching International Movies
* Top 10 Countries By Total Content
* Top 10 Countries With The Most Diverse Set of Genres

Top 10 Most Common Genres

A graph with different colored bars

Description automatically generated

This analysis identifies the most frequently occurring genres in Netflix’s catalogue. The bar chart highlights the top 10 genres that dominate Netflix’s offerings.

* **Dominance of Certain Genres**: Genres like "Drama," "Comedy," and "Documentaries" are among the most common, reflecting broad audience appeal and Netflix’s strategic focus on these genres.
* **Strategic Implications**: Netflix can leverage the popularity of these genres to continue investing in similar content, while also exploring opportunities to diversify offerings in less common genres.

Top Genres by Country (Top 10 Countries)

A screenshot of a graph

Description automatically generated

This analysis examines the top genres across the top 10 countries that contribute the most content to Netflix’s catalogue.

* **Regional Preferences**: The analysis reveals that certain genre, such as "Drama" and "Comedy," are universally popular, while others, like "International Movies," show stronger regional appeal.
* **Strategic Implications**: Netflix should consider regional preferences when curating content libraries and developing original content tailored to specific markets.

Top Countries Producing Documentaries

A graph with a bar graph

Description automatically generated

This section highlights the top countries that are leading in documentary production.

* **Documentary Production Leaders**: Countries like the United States and the United Kingdom are leading in producing documentaries, likely due to their established film industries and global reach.
* **Strategic Implications**: Netflix can focus on acquiring high-quality documentaries from these countries to enhance its library, particularly as documentaries continue to gain popularity among audiences.

Trends in Genre Popularity Over The Last Few Years

A graph of different colored lines

Description automatically generated

This analysis tracks the popularity of various genres over the last few years, identifying which genres have grown or declined.

* **Emerging Genres**: The analysis shows a rise in the popularity of genres like "Documentaries" and "International Movies," indicating a growing and shifting audience interest in diverse and factual content.
* **Strategic Implications**: Netflix should consider increasing investment in emerging genres while re-evaluating the demand for declining genres. Strategic adjustments will help Netflix to better align with changing viewers’ interests and maintain its competitiveness in the market.

Top 10 Countries Watching International Movies

A graph of a number of titles

Description automatically generated

This analysis focuses on the top 10 countries with the highest engagement in watching international movies.

* **Global Reach**: Countries like India and France are among the top consumers of international movies, reflecting a growing interest in diverse global content.
* **Strategic Implications**: Netflix can further expand its international content offerings to cater to these markets, enhancing localization efforts to appeal to a global audience.

Top 10 Countries By Total Content

A graph of a number of titles

Description automatically generated

This section identifies the top 10 countries that contribute the most content to Netflix’s catalogue.

* **Content Powerhouses**: The United States and India lead in total content contributions, showcasing their dominance in content production.
* **Strategic Implications**: Netflix should continue to strengthen partnerships in these regions to maintain a robust and diverse content library.

Top 10 Countries With The Most Diverse Set of Genres

A graph of different colored bars

Description automatically generated

This analysis explores which countries produce the most diverse range of genres, highlighting the richness of their content offerings.

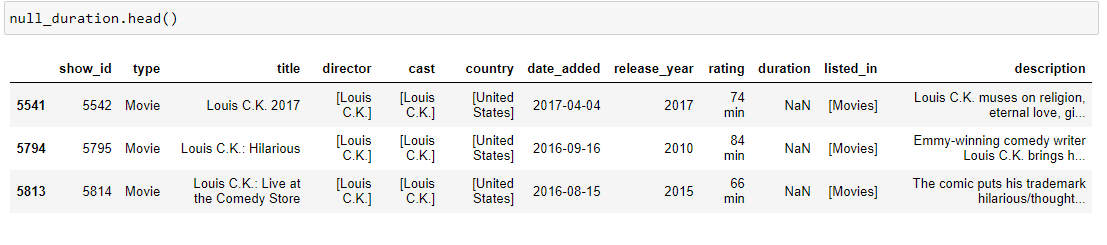
* **Diversity Leaders**: Countries like the United States and United Kingdom exhibit the most genre diversity, likely due to their multicultural societies and well-developed entertainment industries.
* **Strategic Implications**: Netflix can emphasize content diversity from these countries in its global marketing strategies, promoting a wide array of genres to cater to diverse audience tastes.

The insights from this analysis suggest that Netflix should continue to invest in diverse content offerings, particularly in emerging genres and international movies. Regional preferences should guide content curation and marketing efforts, ensuring that Netflix remains a leader in the global streaming market. Further research could involve analysing viewership data to better understand audience engagement with different genres, exploring the impact of global events on genre popularity, and investigating the role of Netflix Originals in shaping genre trends.

## **10. Challenges/Limitations**

Wrongly Filled Columns

The Netflix dataset had some of their column values swapped. Such as in this example:

The rating columns for were filled with the duration and the duration column with NaN values. This error was only spotted when we tried to perform the 3rd normal form operations for the tables.

NaT Values

The NaT values (which are NA values for data type date columns) were only found at the loading phase of the pipeline. It was during the part of table creation for “release\_dates” that an error occurred with the constraints (date value cannot be NA). 12 columns were found to be occupied with “NaT” values instead of their respective “YYYY-MM-DD" format.

Dataset Exclude User Ratings

The Netflix dataset extracted from Kaggle excludes user ratings, which means that by only using the dataset we have certain limitations to work around when it comes to finding out the most popular genre, titles (movies or tv shows), etc. The rating column that was provided by the original dataset refers to the TV and film ratings for age-appropriateness, such as PG, M18, etc.

Null Values Popping Up During Data Visualisation Process

We also found out that some DataFrames such as countries were showing up with null values. Upon manual inspection of the flat file which we exported as part of the data persistence procedure, that was not the case. So instead of null values, it was showing up as empty fields. Which is why when we ran the code .isna() it was not picked up by pandas. This resulted in some of the charts being plotted weirdly or out of context.

## **11. Solutions**

Resolving Wrongly Filled Columns

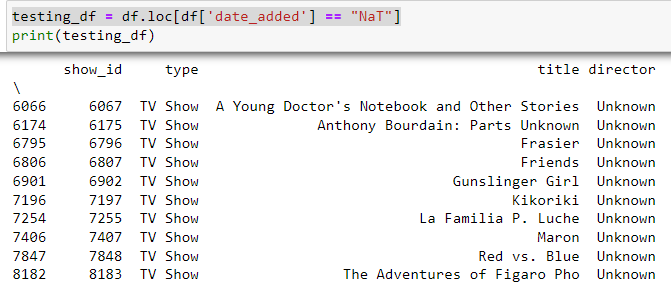
In this scenario, it was easily resolved as we can go back to the data frame and check for null values within the duration column. Which resulted in show\_id: 5542, 5795 and 5814, and coincidentally were directed by Louis C.K.



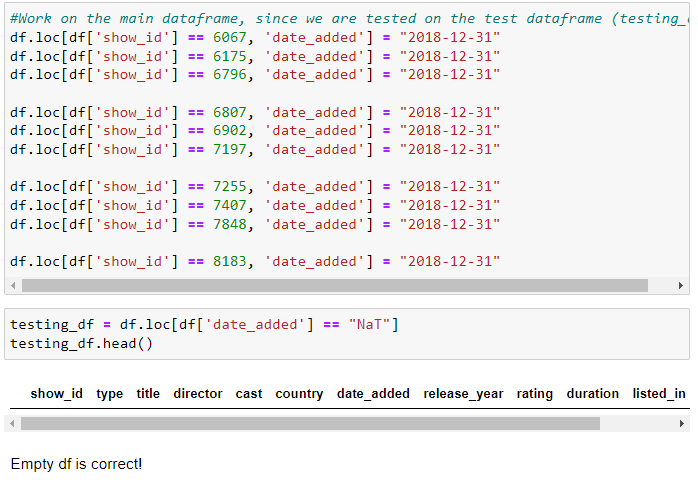
Since it is only 3 columns, we first swapped the duration back to its intended column ‘duration’. And by fill up the actual rating for the three shows as “TV-MA”.

Resolving NaT Values

Like the scenario as before, we can go back to the transform phase and pinpoint which columns are showing up as “NaT” for the column ‘date\_added’.



After locating the columns and their respective show\_id, we assumed that these shows were mostly likely added around 2018, hence replacing the ”NAT” fields with ”2018-12-31” instead.



After replacing the values, we run the syntax again to ensure that there are no more ”NaT” values within the main DataFrame before we try to proceed with 3rd Normal Form.

Resolving Null Values Popping up During Data Visualisation Process

To fix this problem, we first amended the code to check for NULL values. So instead of:

|  |
| --- |
| missing\_checker = countries['country\_id'].isna().sum() print(f"Number of missing or empty values in 'country\_id': {missing\_checker}") |

We change included additional syntax to count in empty fields within columns like so:

|  |
| --- |
| missing\_checker = countries['country\_id'].isna().sum() + (countries['country\_id'] == '').sum() print(f"Number of missing or empty values in 'country\_id': {missing\_checker}") |

We also went back to the transform process to include the syntax for stripping white spaces during the .explode step:

A screen shot of a computer code

Description automatically generated

After which we ran the checker as form of validation and all NULL values and empty spaces were removed.

Resolving User Ratings

We will probably have to rely on additional data sources externally to supplement the Netflix dataset, one such example can be from the listicles generated by IMDB users. Those listicles are usually a list of movies and/or TV shows which are a user's favourites.

So, we can scrap those listicles, sorted by the actual IMDB ratings in descending order and then take the top 20 titles of each listicle.

But in this case, we have decided to pivot around this limitation by assuming that the Netflix dataset has already been built by the provider (Shivam Bansal/Kaggle) and consists of popular titles. And any titles that are already streaming on Netflix are mostly likely to be popular among its viewers globally.

## **12. Conclusion**

Summary of Findings

The analysis of Netflix's genre distribution and trends reveals that "Drama," "Comedy," and "Documentaries" are the most common genres. Emerging genres like "International Movies" and "Documentaries" have grown in popularity. Regional differences highlight that the United States and India lead in content production, with varying genre preferences across countries. Additionally, the United States and United Kingdom produce the most diverse set of genres.

 The insights gained from this analysis offer several strategic directions for Netflix:

* **Content Acquisition and Production**: Given the rise in popularity of certain genres, Netflix should consider investing more heavily in producing and acquiring content in these emerging areas, such as documentaries and international movies. This will help the platform stay relevant and meet the evolving tastes of its global audience.
* **Regional Content Strategies**: The variability in genre preferences across different countries suggests that Netflix should tailor its content offerings and marketing strategies to regional tastes. By focusing on the genres that resonate most in specific markets, Netflix can enhance viewer engagement and satisfaction.
* **Promoting Content Diversity**: Countries producing a diverse set of genres should be highlighted in Netflix’s global catalog, offering viewers a rich and varied content experience. This approach not only caters to diverse audience preferences but also promotes lesser-known genres and titles.
* **Expanding Global Reach**: With a strong demand for international movies, Netflix has an opportunity to further expand its global reach by increasing the availability and visibility of international content. This could involve more localized content production and strategic partnerships in key markets.

Web Scraping

Additionally, given a longer period of time frame to work with we can also include DataFrames from web scraping. A prototype web scraper for IMDB listicles was coded in Python/Jupyter Notebook.

The scraper make used of the BeautifulSoup and requests packages. It will first test the website to get its status code, and then parse the website’s HTML code. And then it will proceed to find all ‘h3’ titles (which are movie titles), append them into an empty list and then converted into a DataFrame with the columns title and count, with count representing the number of times the title has been mentioned by users across different listicles on IMDB.

This is to compensate for the lack of user ratings as pointed out earlier in the challenges/limitations section of this report. It will also allow for more ways to plot our data visualisations such as finding out the most popular movie titles and then further group accordingly to their genres, etc.

For more information, refer to section 2 of Appendix.

Further Optimisations

We can also convert the data types for movie durations in string format into time format. And create functions for data validation instead of repeating the same set of codes for different columns within a DataFrame. This will improve the code readability within the Jupyter Notebook.

Loading into Cloud Database/Link with Microsoft BI

More data visualization can actually be done in Microsoft Power BI since it allows joining of tables, easy drag & drop interface and setting dimensions. It will also be useful for anyone who is looking at our project but with no technical background as it is an interactive dashboard. So with this concept in mind, perhaps we can load our tables from PostgreSQL into a cloud database such as Azure SQL Database and Power BI.

## **13. References**

[1] J. Cherian, “Kaggle API – the missing Python documentation,” *Techno Whisp*, May 31, 2023. https://technowhisp.com/kaggle-api-python-documentation/

[2] “Getting started — pandas 2.2.2 documentation.” https://pandas.pydata.org/docs/getting\_started/index.html

[3] “SQLAlchemy Documentation — SQLAlchemy 2.0 documentation.” https://docs.sqlalchemy.org/

[4] “PostgreSQL: documentation,” *The PostgreSQL Global Development Group*. https://www.PostgreSQL.org/docs/

[5] H. Team, “How to Delete a Character from a String in Python,” *Hostman*. https://hostman.com/tutorials/how-to-delete-a-character-from-a-string-in-python/

## **14. Appendix**

### Section 1. 3NF Tables

The original table can be found on page 7 [[6. Source Of Data](#_6._Source_of)].

And after performing the normalisation up till to their respective 3rd Normal Form, we have the following tables:

* PK = Primary Key
* FK = Foreign Key

Table 1: netflix\_titles

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| show\_id [PK] | int64 | The id of the Netflix movie or tv show. |
| show\_type | string | The type of show (movie or tv show). |
| title | string | The title of the Netflix movie or tv show. |
| release\_year | int64 | The release year of the movie or tv show. |
| rating | string | The TV ratings (e.g. PG13) |
| description | string | Description of the movie or tv show. |

Table 2: movies

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| show\_id [PK/FK] | int64 | The id of the Netflix movie or tv show. |
| duration | string | The total duration of the movie in minutes (e.g. 60 min) |

Table 3: tv\_shows

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| show\_id [PK/FK] | int64 | The id of the Netflix movie or tv show. |
| no\_of\_seasons | string | The number of seasons for the tv show. (e.g. 2 seasons) |

Table 4: release\_dates

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| show\_id [PK/FK] | int64 | The id of the Netflix movie or tv show. |
| dates\_added | string | The date it was added on Netflix. (e.g. YYYY-MM-DD) |

Table 5: directors

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| director\_id [PK] | int64 | The id of the director. |
| director\_name | string | The name of the director. |

Table 6: directs (Junction/Linking Table)

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| show\_id [FK] | int64 | The id of the Netflix movie or tv show. |
| director\_id [FK] | int64 | The id of the director. |

Table 7: casts

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| cast\_id [PK] | int64 | The id of the cast. |
| cast\_name | string | The name of the cast. |

Table 8: acts (Junction/Linking Table)

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| show\_id [FK] | int64 | The id of the Netflix movie or tv show. |
| cast\_id [FK] | int64 | The id of the cast. |

Table 9: countries

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| country\_id [PK] | int64 | The id of the country. |
| country\_name | string | The name of the country. |

Table 10: produces (Junction/Linking Table)

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| show\_id [FK] | int64 | The id of the Netflix movie or tv show. |
| country\_id [FK] | int64 | The id of the country. |

Table 11: genre

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| genre\_id [PK] | int64 | The id of the genre. |
| genre\_name | string | The name of the genre. |

Table 12: belongs (Junction/Linking Table)

|  |  |  |
| --- | --- | --- |
| **Column name** | **dtype** | **description** |
| show\_id [FK] | int64 | The id of the Netflix movie or tv show. |
| genre \_id [FK] | int64 | The id of the genre. |

### Section 2. IMDB Listicles Scraper

The Python Scraper uses requests, bs4 from BeautifulSoup and pandas to scrap popular movie/tv titles on Netflix across the time period of 2018 to 2021, sorted according to their popularity (descending order). These listicles are made by different actual users on IMDB, so the results scrapped will not be bias and has some randomness to it.

So it will first use a header to disguise the scraper as “Chrome” to bypass scraper restrictions from IMDB, as without it, we will fetch an error status code of 403:

A computer code with text

Description automatically generated with medium confidence

Afterwards, it will traverse through the lists of URL, using a for loop to get all the status code for each link and make sure it is all 200 (which means the request was successful). We set a variable check = 1 which will only turn check = -1 when a non 200 status code is fetch, in that scenario it will print an error message.

A screenshot of a computer code

Description automatically generated

Once checking all URLs are working fine, the next step was to find use BeautifulSoup to parse the HTML code from those links and extract the specific data which we want. So in this case, the movie or tv show titles are actually h3 classes. So we can simply use another for loop to get find all the h3 classes to get the title names.

A screen shot of a computer code

Description automatically generated

However, we will still need to store all those information into a list, but we do not want everything from the listicles as it will take too long to process. Hence, we slice the collated titles for each URL using [:20] – which is to get the top 20, strip the title by removing the numbers of each title and finally append them into one long list.

A screenshot of a computer code

Description automatically generated

And we store them into a DataFrame with columns named ‘Title’. The result will be something like this:

A screenshot of a computer

Description automatically generated

However, as shown above, the result is still not ideal as the labels (1., 2., …) are still within the DataFrame. The solution to this is actually using regex and delete characters from a string. More information can be found within this tutorial in the reference page [5].

A screenshot of a computer

Description automatically generated

The final steps are to count the number of times each title has appeared within the DataFrame, create a new column name ‘count’ to store it next to their respective title. This can be done using the following code and will result in something similar to this:

A screenshot of a computer

Description automatically generated

And we can use another slicing to get the final top 20 from this DataFrame and then export it as a flat file (.csv) like so:

A screenshot of a computer program

Description automatically generated