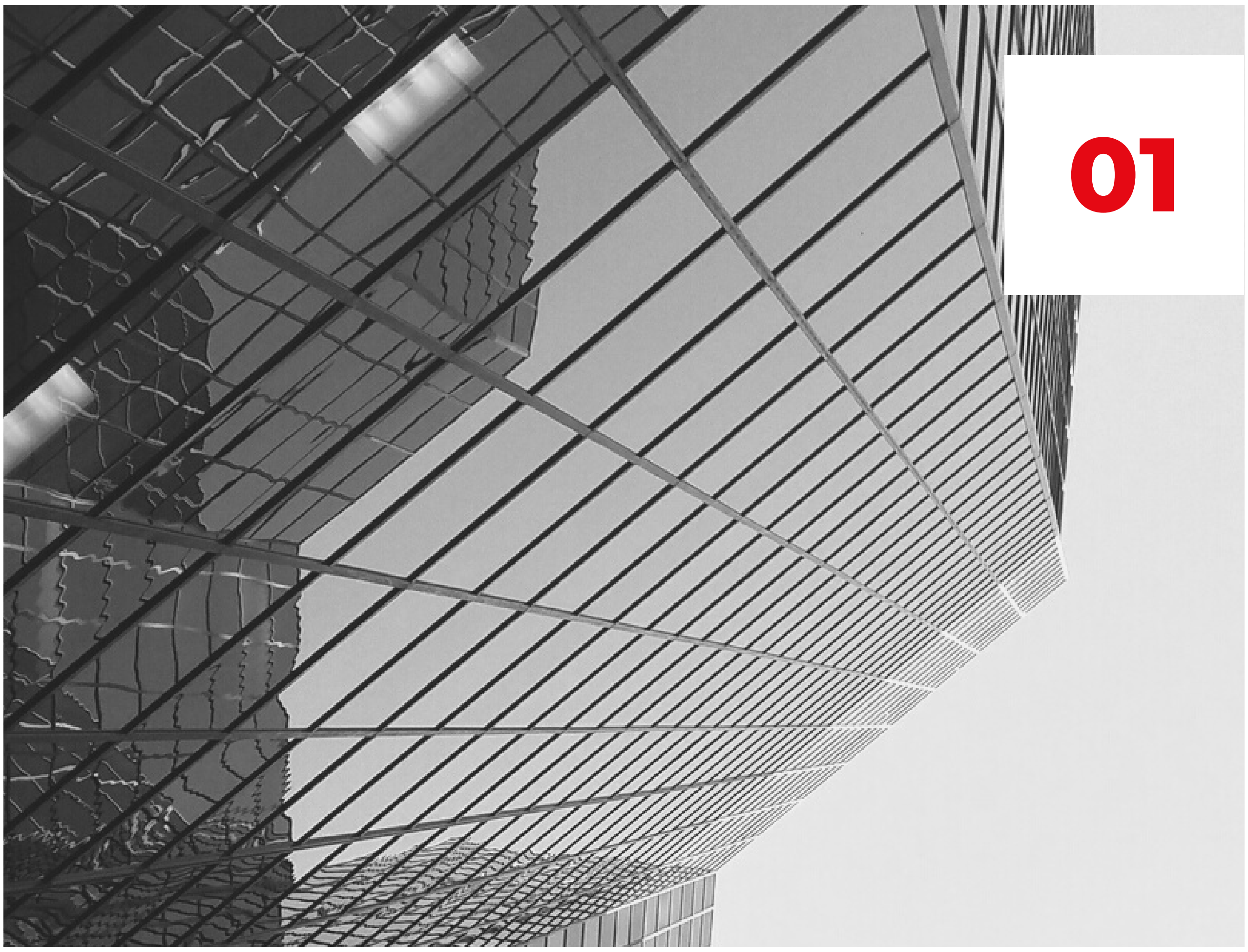
BUSINESS INTELLIGENCE AND DATABASE
MANAGEMENT SYSTEMS REPORT
JANUARY, 20224

NETFLIX PROJECT

Presented by

Azib Amir
Bargaoui Maram
Belaanes Hadil
Miraoui Hadil



01

ABOUT OUR PROJECT

This business intelligence project centers around the analysis of Netflix movies/shows and their IMDb rating data. The primary objective of this study is to explore the impact of various factors, including genre, region, hours_viewed, directors, and casts, on IMDb ratings. By diving into these correlations, we seek to uncover insights into customer behavior and preferences. The knowledge gained will empower Netflix to enhance their content offerings, fostering increased customer satisfaction and enabling the development of strategic initiatives. Ultimately, this project aims to contribute to Netflix's ongoing commitment to delivering an outstanding user experience and refining their business strategies.

OBJECTIVE 02

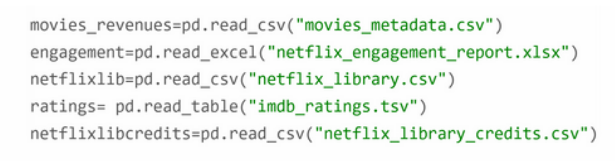
Phase 1 : Data Gathering

We obtained datasets from different sources on the internet as well as different formats:
[movies revenues]: a user-scraped dataset containing budget/revenue data of movies from Kaggle (CSV)
[imdb ratings]: official dataset of average rating scores of tv shows and movies (TSV)
[netflix-engagement-report]: official report from Netflix of number of hours viewed for movies and tv shows from January23-June23 (XLSX)
[netflix library + credits]: user-made dataset of the Netflix library including shows and movies as well as a dataset of credits including directors and actors (CSV)

Phase 2 : Data Preparation

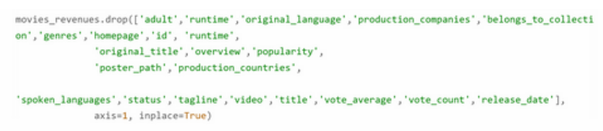
For the data preparation, we used Python to manipulate data and configure it for the data warehouse. At first, we converted all the data into one format (CSV) from different formats (CSV,XLXS,TSV). We changed the structure of the data so that it could be easily manipulated and managed, mainly using the *pandas* python library.
The operations are all available in the Jupyter Notebook file *ETL_process_notebook.ipynb*

1.We loaded our csv files into dataframe variables



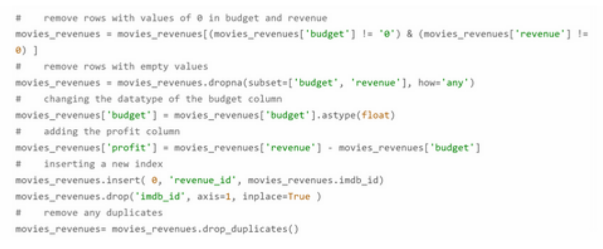
```
movies_revenues=pd.read_csv("movies_metadata.csv")
engagement=pd.read_excel("netflix_engagement_report.xlsx")
netflixlib=pd.read_csv("netflix_library.csv")
ratings= pd.read_table("imdb_ratings.tsv")
netflixlibcredits=pd.read_csv("netflix_library_credits.csv")
```

2.We removed some of the columns that won't be needed



```
movies_revenues.drop(['adult','runtime','original_language','production_companies','belongs_to_collecti
on','genres','homepage','id', 'runtime',
           'original_title','overview','popularity',
           'poster_path','production_countries',

'spoken_languages','status','tagline','video','title','vote_average','vote_count','release_date'],
           axis=1, inplace=True)
```

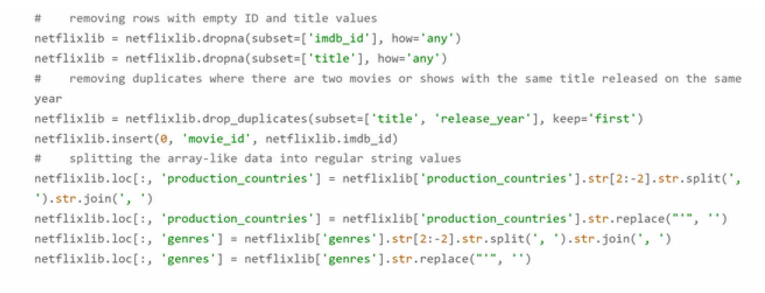
3. Some notable data transformations:



```
#    remove rows with values of 0 in budget and revenue
movies_revenues = movies_revenues[(movies_revenues['budget'] != '0') & (movies_revenues['revenue'] !=
0) ]
#    remove rows with empty values
movies_revenues = movies_revenues.dropna(subset=['budget', 'revenue'], how='any')
#    changing the datatype of the budget column
movies_revenues['budget'] = movies_revenues['budget'].astype(float)
#    adding the profit column
movies_revenues['profit'] = movies_revenues['revenue'] - movies_revenues['budget']
#    inserting a new index
movies_revenues.insert( 0, 'revenue_id', movies_revenues.imdb_id)
movies_revenues.drop('imdb_id', axis=1, inplace=True )
#    remove any duplicates
movies_revenues= movies_revenues.drop_duplicates()
```

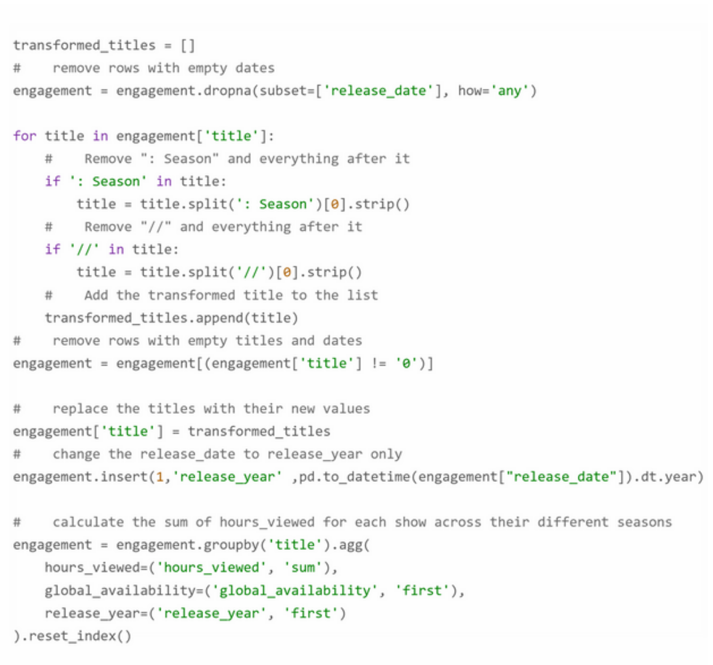
OBJECTIVE 02

netflix library



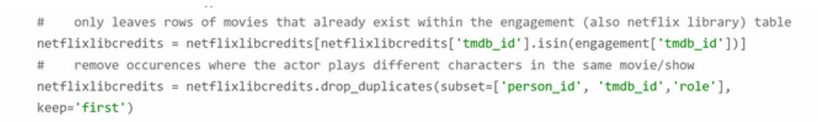
```
#    removing rows with empty ID and title values
netflixlib = netflixlib.dropna(subset=['imdb_id'], how='any')
netflixlib = netflixlib.dropna(subset=['title'], how='any')
#    removing duplicates where there are two movies or shows with the same title released on the same
year
netflixlib = netflixlib.drop_duplicates(subset=['title', 'release_year'], keep='first')
netflixlib.insert(0, 'movie_id', netflixlib.imdb_id)
#    splitting the array-like data into regular string values
netflixlib.loc[:, 'production_countries'] = netflixlib['production_countries'].str[2:-2].str.split(',
').str.join(', ')
netflixlib.loc[:, 'production_countries'] = netflixlib['production_countries'].str.replace("'", '')
netflixlib.loc[:, 'genres'] = netflixlib['genres'].str[2:-2].str.split(', ').str.join(', ')
netflixlib.loc[:, 'genres'] = netflixlib['genres'].str.replace("'", '')
```

netflix engagement



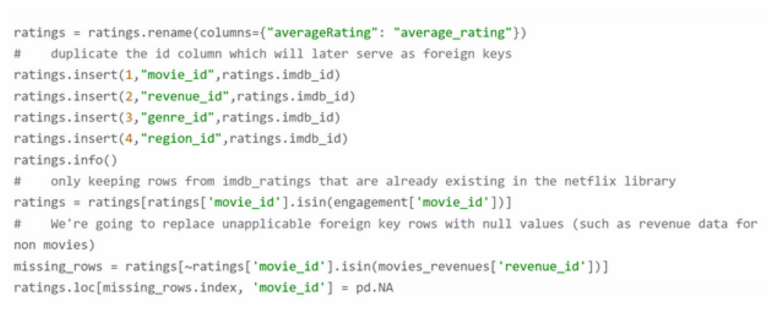
```
transformed_titles = []
#    remove rows with empty dates
engagement = engagement.dropna(subset=['release_date'], how='any')

for title in engagement['title']:
    #    Remove ": Season" and everything after it
    if ': Season' in title:
        title = title.split(': Season')[0].strip()
    #    Remove "//" and everything after it
    if '//' in title:
        title = title.split('//')[0].strip()
    #    Add the transformed title to the list
    transformed_titles.append(title)
#    remove rows with empty titles and dates
engagement = engagement[(engagement['title'] != '0')]

#    replace the titles with their new values
engagement['title'] = transformed_titles
#    change the release_date to release_year only
engagement.insert(1,'release_year' ,pd.to_datetime(engagement["release_date"]).dt.year)

#    calculate the sum of hours_viewed for each show across their different seasons
engagement = engagement.groupby('title').agg(
    hours_viewed=('hours_viewed', 'sum'),
    global_availability=('global_availability', 'first'),
    release_year=('release_year', 'first')
).reset_index()
```

show/movie credits (cast+directors)



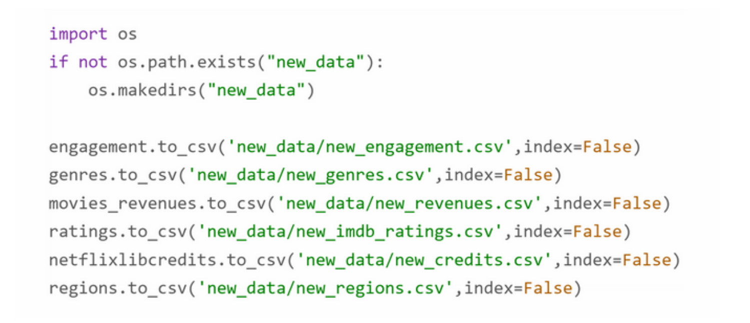
```
#    only leaves rows of movies that already exist within the engagement (also netflix library) table
netflixlibcredits = netflixlibcredits[netflixlibcredits['tmdb_id'].isin(engagement['tmdb_id'])]
#    remove occurences where the actor plays different characters in the same movie/show
netflixlibcredits = netflixlibcredits.drop_duplicates(subset=['person_id', 'tmdb_id','role'],
keep='first')
```

imdb_ratings(fact table)



```
ratings = ratings.rename(columns={"averageRating": "average_rating"})
#    duplicate the id column which will later serve as foreign keys
ratings.insert(1,"movie_id",ratings.imdb_id)
ratings.insert(2,"revenue_id",ratings.imdb_id)
ratings.insert(3,"genre_id",ratings.imdb_id)
ratings.insert(4,"region_id",ratings.imdb_id)
ratings.info()
#    only keeping rows from imdb_ratings that are already existing in the netflix library
ratings = ratings[ratings['movie_id'].isin(engagement['movie_id'])]
#    We're going to replace unapplicable foreign key rows with null values (such as revenue data for
non movies)
missing_rows = ratings[~ratings['movie_id'].isin(movies_revenues['revenue_id'])]
ratings.loc[missing_rows.index, 'movie_id'] = pd.NA
```

Exporting to csv files:



```
import os
if not os.path.exists("new_data"):
    os.makedirs("new_data")

engagement.to_csv('new_data/new_engagement.csv',index=False)
genres.to_csv('new_data/new_genres.csv',index=False)
movies_revenues.to_csv('new_data/new_revenues.csv',index=False)
ratings.to_csv('new_data/new_imdb_ratings.csv',index=False)
netflixlibcredits.to_csv('new_data/new_credits.csv',index=False)
regions.to_csv('new_data/new_regions.csv',index=False)
```

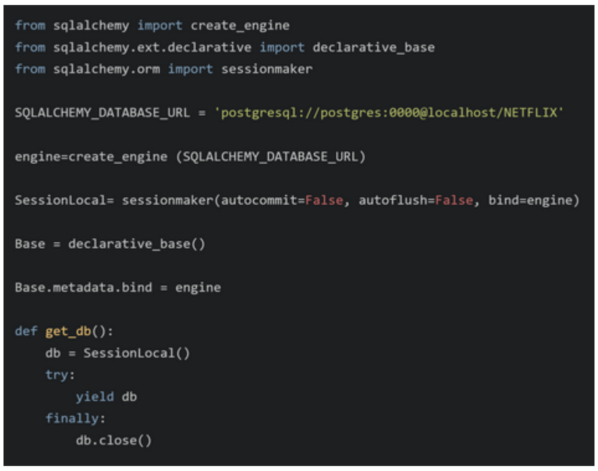
OBJECTIVE 02

Phase 3 : : Data Storage

1- Storage

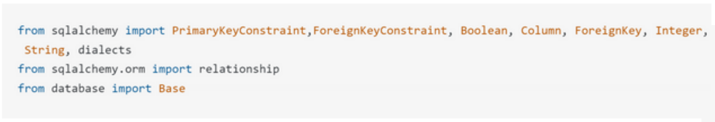
For data storage, we employed the use of sqlalchemy to map the tables, we chose Postgresql as the database hosting server and management system since we can easily load our data using python code using the psycopg2 library.

the database.py file ensures the connection to the database, one must simply put their database personal information in the SQLALCHEMY_DATABASE_URL in this format ‘postgresql://<username>:<password>@<host>/<database_name>’

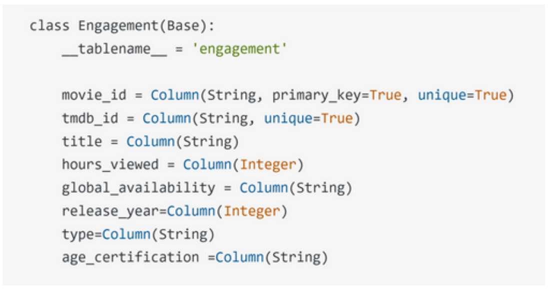


```python
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = 'postgresql://postgres:0000@localhost/NETFLIX'

engine=create_engine (SQLALCHEMY_DATABASE_URL)

SessionLocal= sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

Base.metadata.bind = engine

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

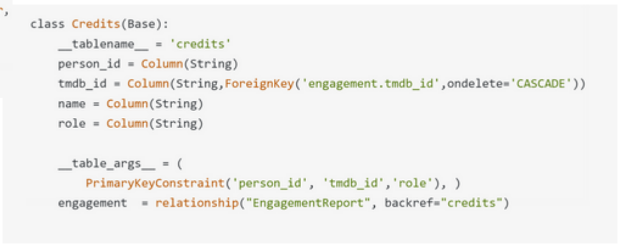
The following is the code for our classes stored in the models folder:



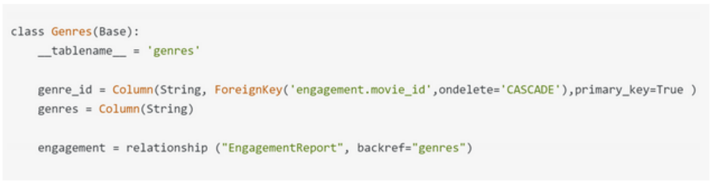
```python
from sqlalchemy import PrimaryKeyConstraint,ForeignKeyConstraint, Boolean, Column, ForeignKey, Integer,
 String, dialects
from sqlalchemy.orm import relationship
from database import Base
```



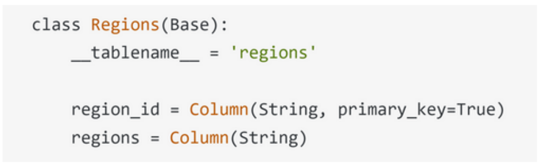
```python
class Engagement(Base):
    __tablename__ = 'engagement'

    movie_id = Column(String, primary_key=True, unique=True)
    tmdb_id = Column(String, unique=True)
    title = Column(String)
    hours_viewed = Column(Integer)
    global_availability = Column(String)
    release_year=Column(Integer)
    type=Column(String)
    age_certification =Column(String)
```



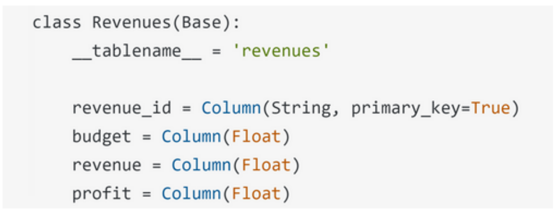
```python
class Credits(Base):
    __tablename__ = 'credits'
    person_id = Column(String)
    tmdb_id = Column(String,ForeignKey('engagement.tmdb_id',ondelete='CASCADE'))
    name = Column(String)
    role = Column(String)

    __table_args__ = (
        PrimaryKeyConstraint('person_id', 'tmdb_id','role'), )
    engagement  = relationship("EngagementReport", backref="credits")
```



```python
class Genres(Base):
    __tablename__ = 'genres'

    genre_id = Column(String, ForeignKey('engagement.movie_id',ondelete='CASCADE'),primary_key=True )
    genres = Column(String)

    engagement = relationship ("EngagementReport", backref="genres")
```



```python
class Regions(Base):
    __tablename__ = 'regions'

    region_id = Column(String, primary_key=True)
    regions = Column(String)
```



```python
class Revenues(Base):
    __tablename__ = 'revenues'

    revenue_id = Column(String, primary_key=True)
    budget = Column(Float)
    revenue = Column(Float)
    profit = Column(Float)
```

OBJECTIVE 02

```
class ImdbRating(Base):

    __tablename__ = 'fact_imdb_ratings'
    imdb_id = Column(String,primary_key=True)

    movie_id = Column(String, ForeignKey(
        'engagement.movie_id',ondelete='CASCADE'),nullable=True)
    revenue_id = Column(String, ForeignKey(
        'revenues.revenue_id',ondelete='CASCADE'),nullable=True)
    region_id = Column(String, ForeignKey(
        'regions.region_id',ondelete='CASCADE'),nullable=True)
    genre_id = Column(String, ForeignKey(
        'genres.genre_id',ondelete='CASCADE'),nullable=True)

    average_rating = Column(Float)

    revenues = relationship("Revenues", backref="fact_imdb_ratings")
    genres = relationship("Genres", backref="fact_imdb_ratings")
    engagement = relationship("Engagement", backref="fact_imdb_ratings")
    regions = relationship ("Regions", backref="fact_imdb_ratings")
```

ImdbRating (fact table): representing the average vote score for each movie/tv show

Regions: represents the region where movie/show was made

Revenues: represents the budget revenue and profit earned by movies only

Genres: represents the genres of the tv show or movie

Credits: represents the credits given to tv show (actors and directors)

Engagement: represents the hours_viewed of each tv show or movie from JAN-JUN23 along with other characteristics

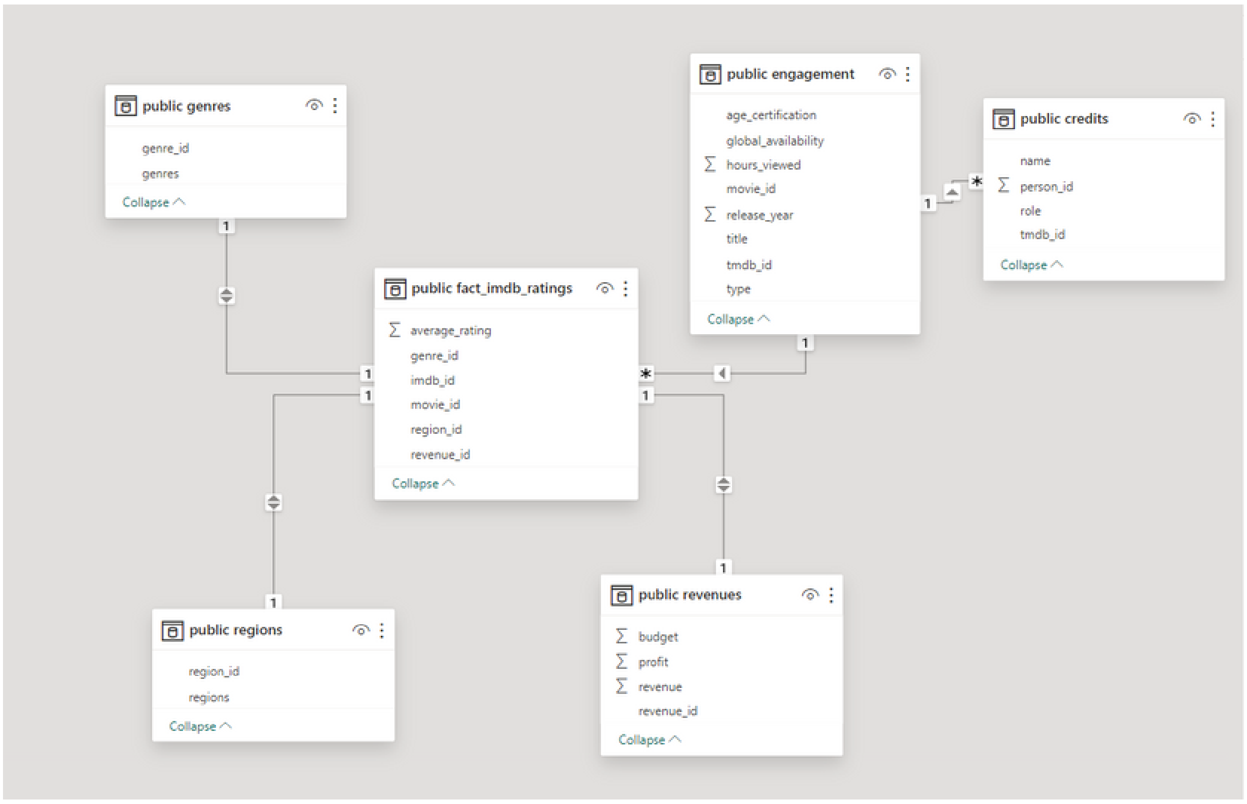
2- Fact

We chose the average rating to be the fact of our schema

3- Dimensions

Genres
Regions
Revenues
Hours viewed
Role

4- Data Warehouse Schema



OBJECTIVE

02

Phase 4 : Data Analysis

We employed a variety of metrics to comprehensively analyze the data and extract meaningful insights from it.

- The most profitable genres are family, fantasy and animation
- The most frequent found genre in Netflix's catalogue is drama, comedy, and documentary
- Out of the primary genres, documentaries, war, drama, crime, and history, earn the highest average of user vote ratings.
- The lowest rated genres are horror and romance
- Shows released in 2021 have the highest viewer rates, followed by 2020, 2019, and 2022. This might be explained by the gap of 1-2 years taken between each tv show season, which means people would often rewatch old seasons to catch up on new ones.
- From Jan22-June23, the most viewed shows or movies were Gabby's dollhouse, Bridgerton, and Orange is the new black.
- The most popular combination of genres is comedy and drama.
- 11.88% of the shows and movies are not available globally
- Non-globally available movies are less profitable.
- There is a negative correlation between the average_rating and the number of hours viewed
- Riverdale is the highest non-available globally show, which was also the 5th most watched show overall
- Globally-available shows and movies have only a very slight advantage on average votes than the non available ones
- There is not a huge correlation between age_certification and the average voting score
- Forrest Gump, Inception, and Goodfellas are the highest rated movies on Netflix
- Action, drama, thriller, comedy, and animation are the most demanding genres budget-wise, with music, history and war being the least.

03

CONCLUSION

Although the datasets used for this project were smaller than they could have been due to limitations and challenges (other datasets are either too small or too large), we were able to equally extract useful insights, such as the most profitable genres, the top rated genres, and the different relationships and correlations between the different variables.