IMPORTS AND SETUP

```
#allows the use of SQL
import duckdb
#data manipulation
import pandas as pd
#makes use of python os
import os
```

GETTING IMDB DATASETS

```
#download and load the imdb dataset
!wget https://datasets.imdbws.com/title.principals.tsv.gz
!wget https://datasets.imdbws.com/title.akas.tsv.gz
!wget https://datasets.imdbws.com/title.crew.tsv.gz
!wget https://datasets.imdbws.com/title.episode.tsv.gz
!wget https://datasets.imdbws.com/title.basics.tsv.gz
!wget https://datasets.imdbws.com/title.ratings.tsv.gz
!wget https://datasets.imdbws.com/name.basics.tsv.gz
```

```
2025-10-20 23:57:51 (61.9 MB/s) - ‘title.akas.tsv.gz.4’ saved [465498252/465498252]

--2025-10-20 23:57:51--  https://datasets.imdbws.com/title.crew.tsv.gz
Resolving datasets.imdbws.com (datasets.imdbws.com)... 13.249.98.61, 13.249.98.91, 13.249.98.73, ...
Connecting to datasets.imdbws.com (datasets.imdbws.com)|13.249.98.61|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 78287484 (75M) [binary/octet-stream]
Saving to: ‘title.crew.tsv.gz.4’

title.crew.tsv.gz.4 100%[===================>]  74.66M  26.3MB/s    in 2.8s

2025-10-20 23:57:54 (26.3 MB/s) - ‘title.crew.tsv.gz.4’ saved [78287484/78287484]

--2025-10-20 23:57:54--  https://datasets.imdbws.com/title.episode.tsv.gz
Resolving datasets.imdbws.com (datasets.imdbws.com)... 13.249.98.61, 13.249.98.91, 13.249.98.73, ...
Connecting to datasets.imdbws.com (datasets.imdbws.com)|13.249.98.61|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 51013504 (49M) [binary/octet-stream]
Saving to: ‘title.episode.tsv.gz.4’

title.episode.tsv.g 100%[===================>]  48.65M   167MB/s    in 0.3s

2025-10-20 23:57:54 (167 MB/s) - ‘title.episode.tsv.gz.4’ saved [51013504/51013504]

--2025-10-20 23:57:54--  https://datasets.imdbws.com/title.basics.tsv.gz
Resolving datasets.imdbws.com (datasets.imdbws.com)... 13.249.98.61, 13.249.98.91, 13.249.98.73, ...
Connecting to datasets.imdbws.com (datasets.imdbws.com)|13.249.98.61|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 212310324 (202M) [binary/octet-stream]
Saving to: ‘title.basics.tsv.gz.4’

title.basics.tsv.gz 100%[===================>] 202.47M  10.9MB/s    in 5.7s

2025-10-20 23:58:00 (35.8 MB/s) - ‘title.basics.tsv.gz.4’ saved [212310324/212310324]

--2025-10-20 23:58:00--  https://datasets.imdbws.com/title.ratings.tsv.gz
Resolving datasets.imdbws.com (datasets.imdbws.com)... 13.249.98.73, 13.249.98.61, 13.249.98.91, ...
Connecting to datasets.imdbws.com (datasets.imdbws.com)|13.249.98.73|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8195189 (7.8M) [binary/octet-stream]
Saving to: ‘title.ratings.tsv.gz.4’

title.ratings.tsv.g 100%[===================>]   7.82M  --.-KB/s    in 0.1s

2025-10-20 23:58:00 (73.9 MB/s) - ‘title.ratings.tsv.gz.4’ saved [8195189/8195189]

--2025-10-20 23:58:00--  https://datasets.imdbws.com/name.basics.tsv.gz
Resolving datasets.imdbws.com (datasets.imdbws.com)... 13.249.98.73, 13.249.98.61, 13.249.98.91, ...
Connecting to datasets.imdbws.com (datasets.imdbws.com)|13.249.98.73|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 291993269 (278M) [binary/octet-stream]
Saving to: ‘name.basics.tsv.gz.4’

name.basics.tsv.gz. 100%[===================>] 278.47M   167MB/s    in 1.7s

2025-10-20 23:58:02 (167 MB/s) - ‘name.basics.tsv.gz.4’ saved [291993269/291993269]
```

LOAD THE IMDB DATA

Loading the imdb data using sql for an easier process of concatenating columns.

```
# Connect to DuckDB
con = duckdb.connect()
```

```
#Title of movies
con.execute("""
CREATE TABLE title_basics AS
SELECT tconst, titleType, primaryTitle, startYear, runtimeMinutes, genres FROM read_csv_auto('title.basics.tsv.gz', delim='\t',
""")

# Title ratings
con.execute("""
CREATE TABLE title_ratings AS
SELECT * FROM read_csv_auto('title.ratings.tsv.gz', delim='\t', header=True);
""")

# Names (people)
con.execute("""
CREATE TABLE name_basics AS
SELECT nconst, primaryName, knownForTitles FROM read_csv_auto('name.basics.tsv.gz', delim='\t', header=True);
""")

#names and roles of everyone
con.execute("""
CREATE TABLE title_principals AS
SELECT tconst, nconst, category, job FROM read_csv_auto('title.principals.tsv.gz', delim='\t', header=True);
""")
```

```
<duckdb.duckdb.DuckDBPyConnection at 0x79419e6a36f0>
```

OBTAINING INDIVIDUAL RATING AVERAGE FOR EACH CAST AND CREW PER MOVIE

1. Obtaining the weighted average of each person's rating based on their movie rating.
2. Filtering movies that have votes above 5000.
3. Filtering movies from the year 2018 to 2025

```
#person_ratings
con.execute("""
CREATE OR REPLACE TABLE person_ratings AS
SELECT
    p.nconst,
    p.primaryName,
    ROUND(SUM(r.averageRating * r.numVotes) * 1.0 / SUM(r.numVotes), 2) AS person_rating
FROM name_basics p
JOIN title_principals tp ON p.nconst = tp.nconst
JOIN title_ratings r ON tp.tconst = r.tconst
JOIN title_basics t ON tp.tconst = t.tconst
WHERE t.titleType = 'movie'
  AND r.numVotes > 5000
  AND t.startYear != '\\N'
  AND CAST(t.startYear AS INTEGER) BETWEEN 2018 AND 2025
GROUP BY p.nconst, p.primaryName
""")
```

```
<duckdb.duckdb.DuckDBPyConnection at 0x79419e6a36f0>
```

Adding each person rating to the movies they were involved in, based on their roles in the movie (actor, actress, directors, writers, composers, cinematographers, and editors. Those are the only roles we are exploring here).

```
import duckdb
import pandas as pd

# Assuming 'con' is your DuckDB connection
df = con.execute("""
SELECT
    t.tconst,
    t.primaryTitle,
    CAST(t.startYear AS INTEGER) AS startYear,
    t.runtimeMinutes,
    t.genres,
```

```
    -- Combine actors' names and their ratings into one string
    string_agg(DISTINCT CASE WHEN tp.category = 'actor' THEN n.primaryName || ' (' || pr.person_rating || ')' END, ', ') AS acto
    string_agg(DISTINCT CASE WHEN tp.category = 'actress' THEN n.primaryName || ' (' || pr.person_rating || ')' END, ', ') AS act
    string_agg(DISTINCT CASE WHEN tp.category = 'director' THEN n.primaryName || ' (' || pr.person_rating || ')' END, ', ') AS di
    string_agg(DISTINCT CASE WHEN tp.category = 'writer' THEN n.primaryName || ' (' || pr.person_rating || ')' END, ', ') AS writ
    string_agg(DISTINCT CASE WHEN tp.category = 'composer' THEN n.primaryName || ' (' || pr.person_rating || ')' END, ', ') AS co
    string_agg(DISTINCT CASE WHEN tp.category = 'cinematographer' THEN n.primaryName || ' (' || pr.person_rating || ')' END, ',
    string_agg(DISTINCT CASE WHEN tp.category = 'editor' THEN n.primaryName || ' (' || pr.person_rating || ')' END, ', ') AS edit

FROM title_basics t
JOIN title_ratings r ON t.tconst = r.tconst
JOIN title_principals tp ON t.tconst = tp.tconst
JOIN name_basics n ON tp.nconst = n.nconst
JOIN person_ratings pr ON n.nconst = pr.nconst

WHERE t.titleType = 'movie'
  AND r.numVotes > 5000
  AND t.startYear != '\\N'
  AND CAST(t.startYear AS INTEGER) BETWEEN 2018 AND 2025

GROUP BY t.tconst, t.primaryTitle, startYear, t.runtimeMinutes, t.genres

""").fetchdf()
```

Create a csv file

```
df.to_csv('movies_ratings.csv', index=False)
```

```
# Now 'final_df' is a Pandas DataFrame — view it with:
print(df.head())  # Show first 5 rows

# Or, for a nicer view in Jupyter notebooks:
df.head()

# To see the full structure:
print(df.info())

# To see summary statistics:
print(df.describe())
```

```
      tconst              primaryTitle  startYear runtimeMinutes  \
0   tt8523334               City Hunter       2018             91
1   tt4180560                 Otherhood       2019            100
2   tt8201852      You Should Have Left       2020             93
3   tt1630029  Avatar: The Way of Water       2022            192
4  tt11245972                    Scream       2022            114

                    genres  \
0       Action,Comedy,Crime
1                    Comedy
2    Horror,Mystery,Thriller
3   Action,Adventure,Fantasy
4    Horror,Mystery,Thriller

                                          actors  \
0  Didier Bourdon (6.5), Gérard Jugnot (6.46), Ka...
1  Stephen Kunken (6.59), Sinqua Walls (6.32), Ja...
2  Kevin Bacon (6.16), Colin Blumenau (5.4), Eli ...
3  Stephen Lang (6.83), Sam Worthington (6.95), C...
4  Dylan Minnette (5.77), David Arquette (6.24), ...

                                        actresses  \
0  Pamela Anderson (6.58), Élodie Fontan (6.25), ...
1  Patricia Arquette (6.1), Angela Bassett (7.33)...
2  Lowri Ann Richards (5.4), Avery Tiiu Essex (5....
3  CCH Pounder (7.35), Sigourney Weaver (7.29), K...
4  Neve Campbell (6.13), Courteney Cox (6.34), Je...

                                        directors  \
0                          Philippe Lacheau (6.5)
1                             Cindy Chupack (6.1)
2                              David Koepp (6.23)
3                            James Cameron (7.29)
4  Matt Bettinelli-Olpin (6.49), Tyler Gillett (6...

                                         writers  \
0  Philippe Lacheau (6.5), Julien Arruti (6.5), T...
1  Mark Andrus (6.1), Cindy Chupack (6.1), Willia...
```

```
2          David Koepp (6.23), Daniel Kehlmann (5.4)
3  Amanda Silver (7.05), Josh Friedman (7.27), Ja...
4  Kevin Williamson (6.23), James Vanderbilt (6.3...

                                    composers          cinematographers  \
0  Michaël Tordjman (6.5), Maxime Desprez (6.5)    Vincent Richard (5.25)
1                          Marcelo Zarvos (6.8)        Declan Quinn (7.8)
2                          Geoff Zanelli (6.48)       Angus Hudson (6.04)
3                         Simon Franglen (7.38)  Russell Carpenter (7.44)
4                            Brian Tyler (6.39)   Brett Jutkiewicz (6.63)

                                         editors
0          Antoine Vareille (5.51), Marc David (6.5)
1               Sunny Hodge (6.1), Kevin Tent (7.65)
2                             Derek Ambrosi (5.48)
3  David Brenner (7.68), John Refoua (7.5), Steph...
4                              Michel Aller (6.25)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4135 entries, 0 to 4134
Data columns (total 12 columns):
```

```
df.head()
```

	tconst	primaryTitle	startYear	runtimeMinutes	genres	actors	actresses	directors	writers	compose
0	tt8523334	City Hunter	2018	91	Action,Comedy,Crime	Didier Bourdon (6.5), Gérard Jugnot (6.46), Ka...	Pamela Anderson (6.58), Élodie Fontan (6.25), ...	Philippe Lacheau (6.5)	Philippe Lacheau (6.5), Julien Arruti (6.5), T...	Mich Tordjm (6 Maxi Desp (6
1	tt4180560	Otherhood	2019	100	Comedy	Stephen Kunken (6.59), Sinqua Walls (6.32), Ja...	Patricia Arquette (6.1), Angela Bassett (7.33)...	Cindy Chupack (6.1)	Mark Andrus (6.1), Cindy Chupack (6.1), Willia...	Marc Zar (6
2	tt8201852	You Should Have Left	2020	93	Horror,Mystery,Thriller	Kevin Bacon (6.16), Colin Blumenau (5.4), Eli ...	Lowri Ann Richards (5.4), Avery Tiiu Essex (5....	David Koepp (6.23)	David Koepp (6.23), Daniel Kehlmann (5.4)	Ge Zar (6.
3	tt1630029	Avatar: The Way of Water	2022	192	Action,Adventure,Fantasy	Stephen Lang (6.83), Sam Worthington (6.95), C...	CCH Pounder (7.35), Sigourney Weaver (7.29), K...	James Cameron (7.29)	Amanda Silver (7.05), Josh Friedman (7.27), Ja...	Sim Frang (7.
4	tt11245972	Scream	2022	114	Horror,Mystery,Thriller	Dylan Minnette (5.77), David Arquette (6.24), ...	Neve Campbell (6.13), Courteney Cox (6.34), Je...	Matt Bettinelli-Olpin (6.49), Tyler Gillett (6...	Kevin Williamson (6.23), James Vanderbilt (6.3...	Brian Ty (6.

OVERALL AVERAGE FOR ALL CAST AND CREW IN A MOVIE

```
#person_ratings
con.execute("""
CREATE OR REPLACE TABLE person_ratings AS
SELECT
    p.nconst,
    p.primaryName,
    ROUND(SUM(r.averageRating * r.numVotes) * 1.0 / SUM(r.numVotes), 2) AS person_rating
FROM name_basics p
JOIN title_principals tp ON p.nconst = tp.nconst
JOIN title_ratings r ON tp.tconst = r.tconst
JOIN title_basics t ON tp.tconst = t.tconst
WHERE t.titleType = 'movie'
  AND r.numVotes > 5000
  AND t.startYear != '\\N'
  AND CAST(t.startYear AS INTEGER) BETWEEN 2018 AND 2025
GROUP BY p.nconst, p.primaryName

""")
```

```
<duckdb.duckdb.DuckDBPyConnection at 0x79419e6a36f0>
```

```
import duckdb
import pandas as pd

# Assuming 'con' is your DuckDB connection
df = con.execute("""
SELECT
    t.tconst,
    t.primaryTitle,
    CAST(t.startYear AS INTEGER) AS startYear,
    t.runtimeMinutes,
    t.genres,
    r.averageRating AS movie_rating,

    -- Average of all actor/actress ratings
    ROUND(AVG(CASE WHEN tp.category IN ('actor', 'actress') THEN pr.person_rating END), 2) AS avg_cast_rating,

    -- Single director rating (or avg if multiple)
    ROUND(AVG(CASE WHEN tp.category = 'director' THEN pr.person_rating END), 2) AS director_rating,

    -- Same for writer and composer
    ROUND(AVG(CASE WHEN tp.category = 'writer' THEN pr.person_rating END), 2) AS writer_rating,
    ROUND(AVG(CASE WHEN tp.category = 'composer' THEN pr.person_rating END), 2) AS composer_rating,

    -- Same for cinematographer and editor
    ROUND(AVG(CASE WHEN tp.category = 'cinematographer' THEN pr.person_rating END), 2) AS cinematographer_rating,
    ROUND(AVG(CASE WHEN tp.category = 'editor' THEN pr.person_rating END), 2) AS editor_rating

FROM title_basics t
JOIN title_ratings r ON t.tconst = r.tconst
JOIN title_principals tp ON t.tconst = tp.tconst
JOIN name_basics n ON tp.nconst = n.nconst
JOIN person_ratings pr ON n.nconst = pr.nconst

WHERE t.titleType = 'movie'
  AND r.numVotes > 5000
  AND t.startYear != '\\N'
  AND CAST(t.startYear AS INTEGER) BETWEEN 2018 AND 2025

GROUP BY t.tconst, t.primaryTitle, startYear, runtimeMinutes, genres, r.averageRating


""").fetchdf()
```

```
df.to_csv('movie_rating_avg.csv', index=False)
```

```
df.head()
```

	tconst	primaryTitle	startYear	runtimeMinutes	genres	movie_rating	avg_cast_rating	director_rating
0	tt19034332	The Mystery of Marilyn Monroe: The Unheard Tapes	2022	101	Biography,Crime,Documentary	6.2	6.20	6.20
1	tt6893836	They'll Love Me When I'm Dead	2018	98	Biography,Documentary	7.4	NaN	7.70
2	tt4566758	Mulan	2020	115	Action,Adventure,Drama	5.8	6.04	5.75
3	tt7131622	Once Upon a Time... in Hollywood	2019	161	Comedy,Drama	7.6	7.35	7.60
4	tt21279806	Scoop	2024	102	Biography,Drama	6.5	6.50	6.50