IMDb

HOMEWORK 3: DATA ANALYSIS

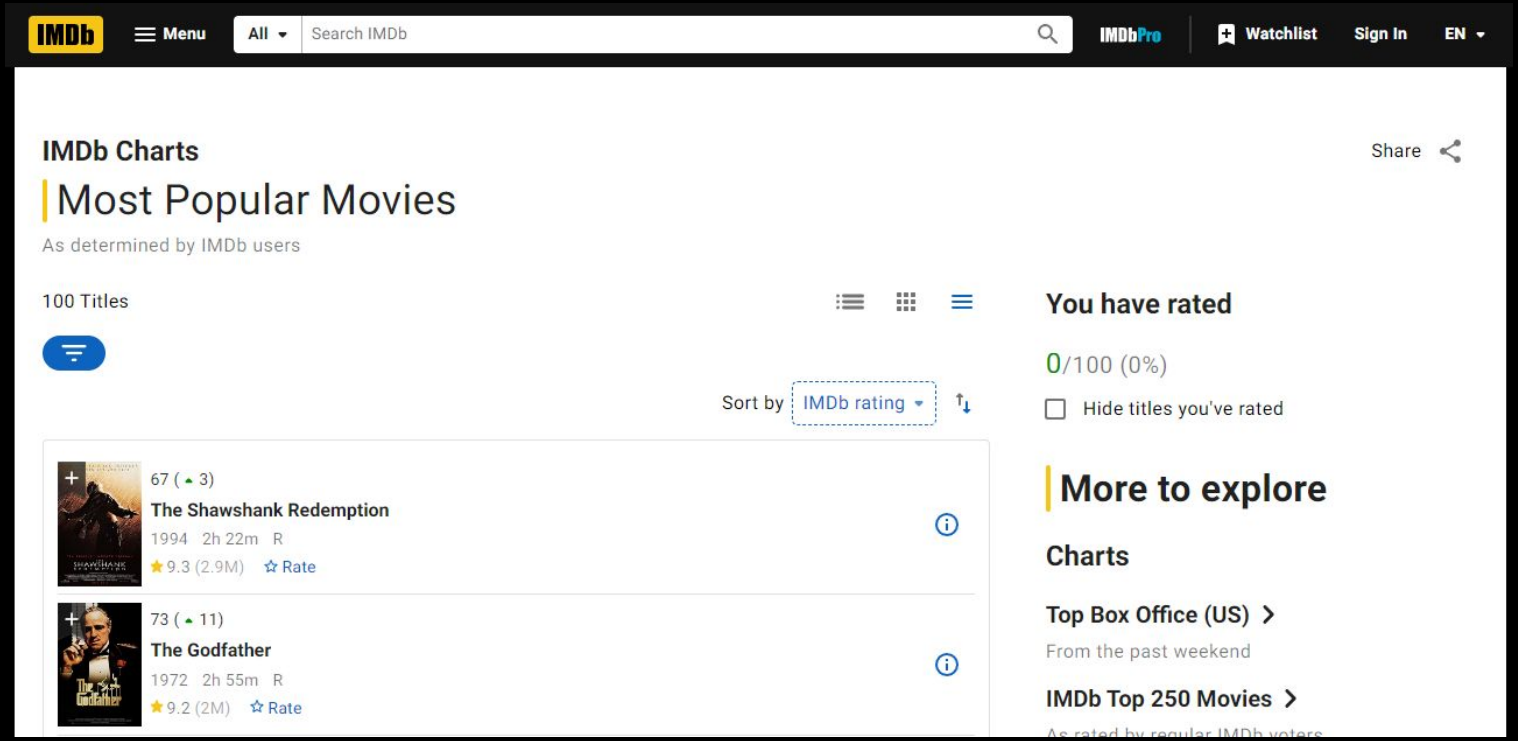
Presented by DAT204M Group 2

IMDb

INTRO AND ARCHITECTURE

SOURCE AND DATASET

Since the goal of the project is to create a dashboard about movie stats, we sourced our data from IMDb, the world's most popular and authoritative source for movie, TV and celebrity content.



SOURCE AND DATASET

Since the goal of the project is to create a dashboard about movie stats, we sourced our data from IMDb, the world's most popular and authoritative source for movie, TV and celebrity content.

```
movies_df = pd.DataFrame(load_progress()[0])
reviews_df = pd.DataFrame(load_progress()[1])

print("Movies DataFrame:")
movies_df.head(3)
```

Movies DataFrame:

	movie_code	Title	Year	Revenue	Budget	Runtime	Opening_weekend_US	Rating	Production_company	Genre	IMDb_code	release_date
0	278	The Shawshank Redemption	1994	28341469	25000000	142	28341469	R	Castle Rock Entertainment	Drama, Crime	tt0111161	1994-09-23
1	238	The Godfather	1972	245066411	6000000	175	245066411	R	Paramount Pictures, Alfran Productions, Americ...	Drama, Crime	tt0068646	1972-03-14
2	240	The Godfather Part II	1974	102600000	13000000	202	102600000	R	Paramount Pictures, The Coppola Company, Ameri...	Drama, Crime	tt0071562	1974-12-20

```
print("Reviews DataFrame:")
reviews_df.head(10)
```

Reviews DataFrame:

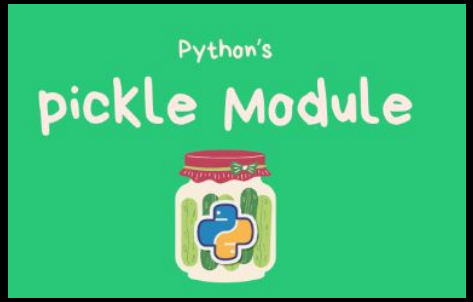
	movie_code	IMDb_code	review_date	rating_of_movie	actual_review
0	278	tt0111161	2016-04-29T18:08:41.892Z	9.0	very good movie 9.5/10 محمد الشعراوى
1	278	tt0111161	2016-07-10T00:16:50.561Z	10.0	Some birds aren't meant to be caged.\r\n\r\nTh...
2	278	tt0111161	2017-11-11T15:09:34.114Z	6.0	Make way for the best film ever made people. *...
3	278	tt0111161	2018-05-01T05:51:13.756Z	10.0	There is a reason why this movie is at the top...
4	278	tt0111161	2018-10-18T15:08:48.777Z	NaN	It's still puzzling to me why this movie exact...
5	278	tt0111161	2019-07-30T08:25:48.402Z	NaN	I will not say that the film is predictable, b...
6	278	tt0111161	2021-09-18T19:56:48.348Z	10.0	First time seeing this in probably close to 20...
7	278	tt0111161	2023-01-14T18:44:02.525Z	NaN	No 1 movie for all the time
8	278	tt0111161	2023-04-03T15:38:39.540Z	3.0	This is much more predictable and Hollywood th...
9	278	tt0111161	2023-06-29T17:48:41.064Z	NaN	This movie is great

TECHNOLOGIES USED

IMDb

The following platforms were used for data collection, processing, loading, and visualization:

THE MOVIE DB



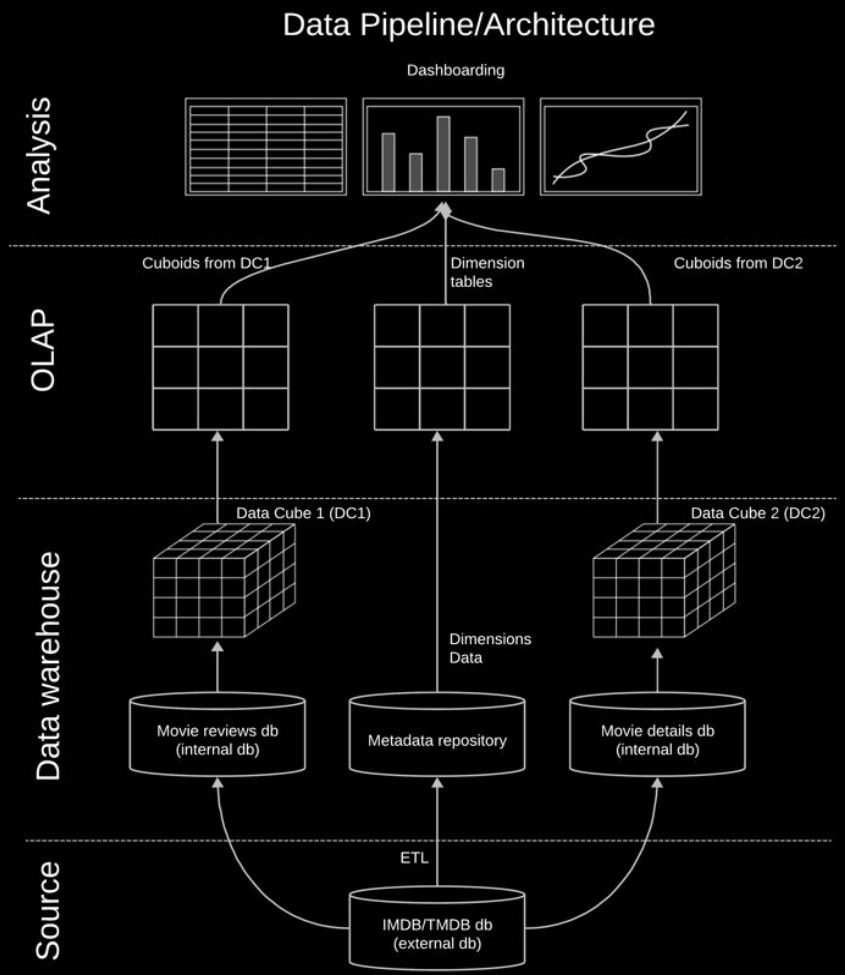
+ a b | e a u

DATA PIPELINE

We followed a 3-tier approach of data warehousing. From source, we have:

1. Data warehouse
2. OLAP
3. Analysis

(to be continued in OLAP section)



DATA PIPELINE

```python
def get_top_movies(api_key, page=1):
    url = f"{BASE_TMDB_URL}/movie/top_rated?api_key={api_key}&language=en-US&page={page}"
    response = requests.get(url)
    movies = response.json().get('results', [])
    top_movies = [(movie['title'], movie['id'], movie['release_date']) for movie in movies]
    return top_movies
```

Get API response and movie attributes

```python
def fetch_movie_details(tmdb_id, api_key):
    url = f"{BASE_TMDB_URL}/movie/{tmdb_id}?api_key={api_key}&language=en-US"
    response = requests.get(url)
    movie = response.json()

    imdb_id = movie.get('imdb_id')
    rating = movie.get('release_dates', {}).get('results', [{}])[0].get('certification', 'Not Rated')

    simplified_rating = 'Not Rated'
    if 'R' in rating:
        simplified_rating = 'R'
    elif 'PG' in rating:
        simplified_rating = 'PG'
    elif 'G' in rating:
        simplified_rating = 'G'

    return {
        'movie_code': movie.get('id'),
        'Title': movie.get('title'),
        'Year': movie.get('release_date', '').split('-')[0],
        'Revenue': movie.get('revenue'),
        'Budget': movie.get('budget'),
        'Runtime': movie.get('runtime'),
        'Opening_weekend_US': movie.get('revenue'),
        'Rating': simplified_rating,
        'Production_company': ', '.join([comp['name'] for comp in movie.get('production_companies', [])]),
        'Genre': ', '.join([genre['name'] for genre in movie.get('genres', [])]),
        'IMDb_code': imdb_id
    }
```

Extract movie attributes

DATA PIPELINE

```python
def fetch_tmdb_reviews(tmdb_id, api_key, imdb_code):
    url = f"{BASE_TMDB_URL}/movie/{tmdb_id}/reviews?api_key={api_key}&language=en-US"
    response = requests.get(url)
    reviews = response.json().get('results', [])
    review_data = []
    for review in reviews:
        review_data.append({
            'movie_code': tmdb_id,
            'IMDb_code': imdb_code,
            'review_date': review.get('created_at'),
            'rating_of_movie': review.get('author_details', {}).get('rating'),
            'actual_review': review.get('content')
        })
    return review_data
```

Get movie reviews based from imdb_code, and tmdb_id

```python
def save_progress(movie_data, reviews_data, filename='movies_data.pkl', reviews_filename='reviews_data.pkl'):
    try:
        with open(filename, 'wb') as f:
            pickle.dump(movie_data, f)
        print(f"Saved movie data to {filename}")
        with open(reviews_filename, 'wb') as f:
            pickle.dump(reviews_data, f)
        print(f"Saved reviews data to {reviews_filename}")
    except (pickle.PickleError, IOError) as e:
        print(f"Error saving data: {e}")

def load_progress(filename='movies_data.pkl', reviews_filename='reviews_data.pkl'):
    movie_data = []
    reviews_data = []

    try:
        if os.path.exists(filename):
            with open(filename, 'rb') as f:
                movie_data = pickle.load(f)
    except Exception as e:
        print(f"Error loading {filename}: {e}")

    try:
        if os.path.exists(reviews_filename):
            with open(reviews_filename, 'rb') as f:
                reviews_data = pickle.load(f)
    except Exception as e:
        print(f"Error loading {reviews_filename}: {e}")

    return movie_data, reviews_data
```

Utilized pickle module, to save and load progress

DATA PIPELINE

```python
def get_movies_and_reviews(movie_filename='movies_data.pkl', reviews_filename='reviews_data.pkl'):
    movie_data, reviews_data = load_progress(movie_filename, reviews_filename)
    approximate_entry_size_kb = 5
    data_size_limit_kb = 1 * 1024 * 1024  # 1GB in KB
    accumulated_size_kb = sum([len(str(movie)) / 1024 for movie in movie_data])
    page = (len(movie_data) // 20) + 1

    while accumulated_size_kb < data_size_limit_kb:
        top_movies = get_top_movies(TMDB_API_KEY, page)
        if not top_movies:
            break

        for title, tmdb_id, release_date in top_movies:
            if any(movie['movie_code'] == tmdb_id for movie in movie_data):
                continue

            movie_info = fetch_movie_details(tmdb_id, TMDB_API_KEY)
            imdb_code = movie_info.get('IMDb_code')
            reviews = fetch_tmdb_reviews(tmdb_id, TMDB_API_KEY, imdb_code)

            if reviews:
                movie_info['release_date'] = release_date
                movie_data.append(movie_info)
                reviews_data.extend(reviews)
                accumulated_size_kb += approximate_entry_size_kb
                print(f"Fetched data for: {title}")
                save_progress(movie_data, reviews_data)
                time.sleep(1)

            if accumulated_size_kb >= data_size_limit_kb:
                print(f"Reached data size limit of approximately {data_size_limit_kb / (1024 * 1024)} GB")
                break
        page += 1

    save_progress(movie_data, reviews_data)

    movie_df = pd.DataFrame(movie_data)
    reviews_df = pd.DataFrame(reviews_data)

    movie_df.to_csv(movie_filename, index=False)
    reviews_df.to_csv(reviews_filename, index=False)

    print(f"Data saved to {movie_filename} and {reviews_filename}")

    return movie_df, reviews_df
```

Main function:

1. Check if a pickle file is available
2. Set data size limit to 1gb
3. Extract attributes
4. Save progress
5. Convert to pandas to csv

SAMPLE RESULTS

IMDb

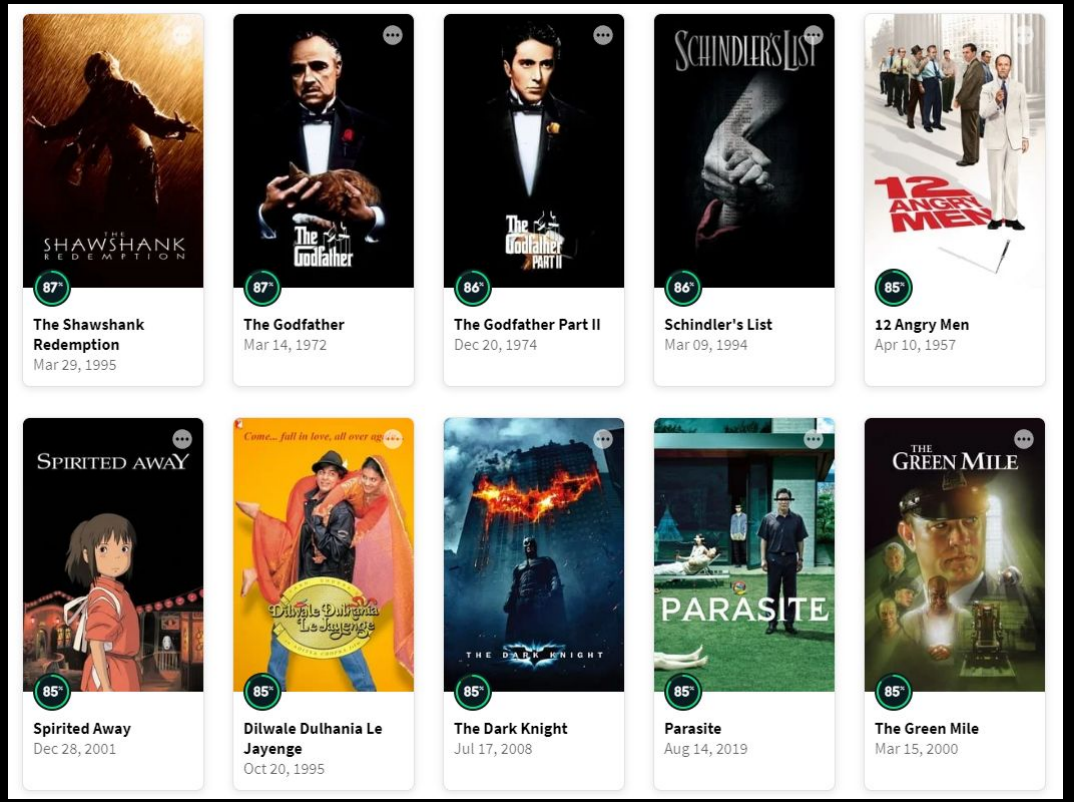
```
movie_data, reviews_data = get_movies_and_reviews()
save_progress(movie_data, reviews_data)
```

```
Fetched data for: The Shawshank Redemption
Saved movie data to movies_data.pkl
Saved reviews data to reviews_data.pkl
Fetched data for: The Godfather
Saved movie data to movies_data.pkl
Saved reviews data to reviews_data.pkl
Fetched data for: The Godfather Part II
Saved movie data to movies_data.pkl
Saved reviews data to reviews_data.pkl
Fetched data for: Schindler's List
Saved movie data to movies_data.pkl
Saved reviews data to reviews_data.pkl
```

PICKLE CONTINUATION

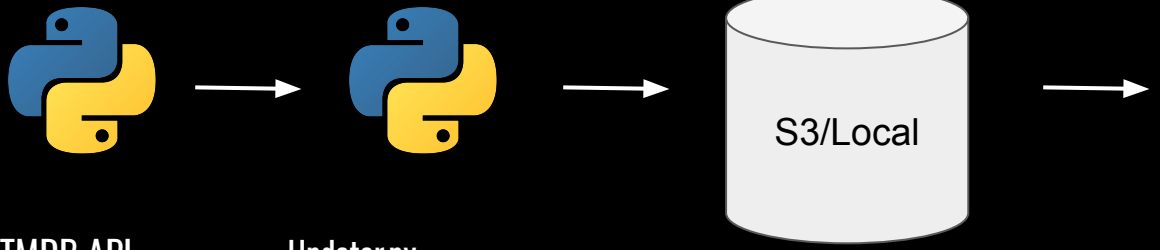
```
movie_data, reviews_data = get_movies_and_reviews()
save_progress(movie_data, reviews_data)
```

```
Fetched data for: 12 Angry Men
Saved movie data to movies_data.pkl
Saved reviews data to reviews_data.pkl
Fetched data for: Spirited Away
Saved movie data to movies_data.pkl
Saved reviews data to reviews_data.pkl
Fetched data for: Dilwale Dulhania Le Jayenge
Saved movie data to movies_data.pkl
Saved reviews data to reviews_data.pkl
```



ONLINE TRANSACTION PROCESSING (OLTP)

IMDb



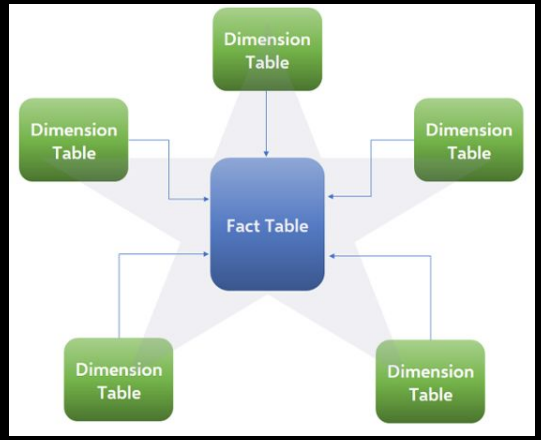
TMDB API

Source

Updater.py

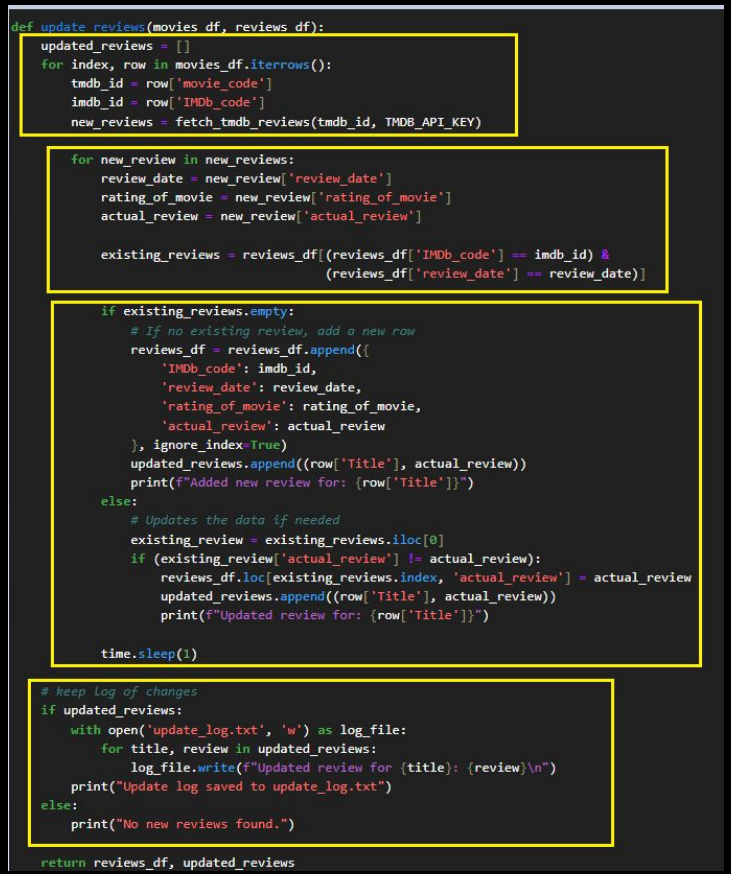
“New transactions” and Transform data

Data Warehouse



Schema

ONLINE TRANSACTION PROCESSING (OLTP)



```
def update_reviews(movies_df, reviews_df):
    updated_reviews = []
    for index, row in movies_df.iterrows():
        tmdb_id = row['movie_code']
        imdb_id = row['IMDb_code']
        new_reviews = fetch_tmdb_reviews(tmdb_id, TMDB_API_KEY)

        for new_review in new_reviews:
            review_date = new_review['review_date']
            rating_of_movie = new_review['rating_of_movie']
            actual_review = new_review['actual_review']

            existing_reviews = reviews_df[(reviews_df['IMDb_code'] == imdb_id) &
                                          (reviews_df['review_date'] == review_date)]

            if existing_reviews.empty:
                # If no existing review, add a new row
                reviews_df = reviews_df.append({
                    'IMDb_code': imdb_id,
                    'review_date': review_date,
                    'rating_of_movie': rating_of_movie,
                    'actual_review': actual_review
                }, ignore_index=True)
                updated_reviews.append((row['Title'], actual_review))
                print(f"Added new review for: {row['Title']}")
            else:
                # Updates the data if needed
                existing_review = existing_reviews.iloc[0]
                if (existing_review['actual_review'] != actual_review):
                    reviews_df.loc[existing_reviews.index, 'actual_review'] = actual_review
                    updated_reviews.append((row['Title'], actual_review))
                    print(f"Updated review for: {row['Title']}")

            time.sleep(1)

    # keep log of changes
    if updated_reviews:
        with open('update_log.txt', 'w') as log_file:
            for title, review in updated_reviews:
                log_file.write(f"Updated review for {title}: {review}\n")
        print("Update log saved to update_log.txt")
    else:
        print("No new reviews found.")

    return reviews_df, updated_reviews
```

1. Initializes variables and iterates through the movies DataFrame.
2. Fetches new reviews for each movie using the movie_code and IMDb_code columns.
3. Checks if a review already exists and updates, otherwise, it adds a new review.
4. Logs the changes to a file.
5. Returns the updated DataFrame and a list of updated reviews.

SAMPLE EXECUTION

IMDb

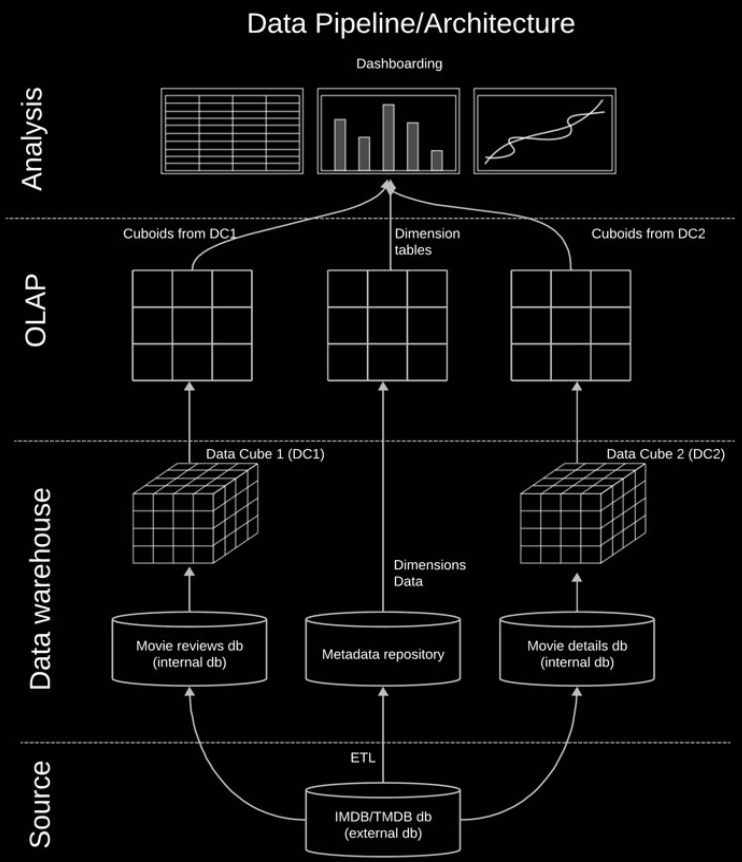
```python
if __name__ == "__main__":
    movies_df = pd.read_pickle('movies_data.pkl')
    reviews_df = pd.read_pickle('reviews_data.pkl')

    reviews_df, updated_reviews = update_reviews(movies_df, reviews_df)

    reviews_df.to_pickle('updated_reviews_data.pkl')
```

```
No new reviews found.
```

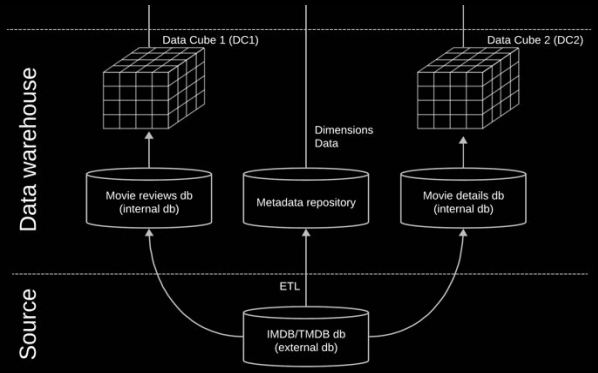
DATA ARCHITECTURE



Three-tier Architecture

1. Data warehouse - ETL the data from outside server (OLTP) then dump the pkl files to local drive.
2. OLAP
3. Analysis

DATA ARCHITECTURE - WAREHOUSE



Data Warehouse

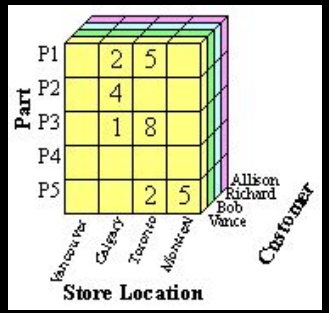
1. Convert pkl file to data frame
2. Data cleaning
3. Construct fact tables (data cubes) and dimension tables from raw data and metadata details

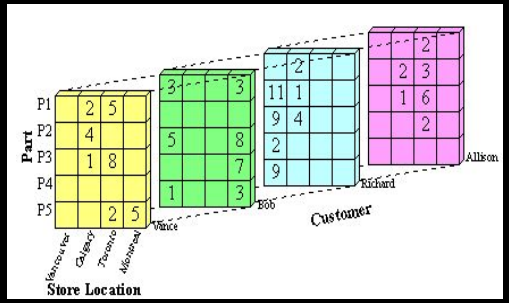
Movie details

```
return {
    'movie_code': movie.get(
    'Title': movie.get('titl
    'Year': movie.get('relea
    'Revenue': movie.get('re
    'Budget': movie.get('buc
    'Runtime': movie.get('ru
    'Opening_weekend_US': mo
    'Rating': simplified_rat
    'Production_company': ',
    'Genre': ', '.join([genr
    'IMDb_code': imdb_id
}
```

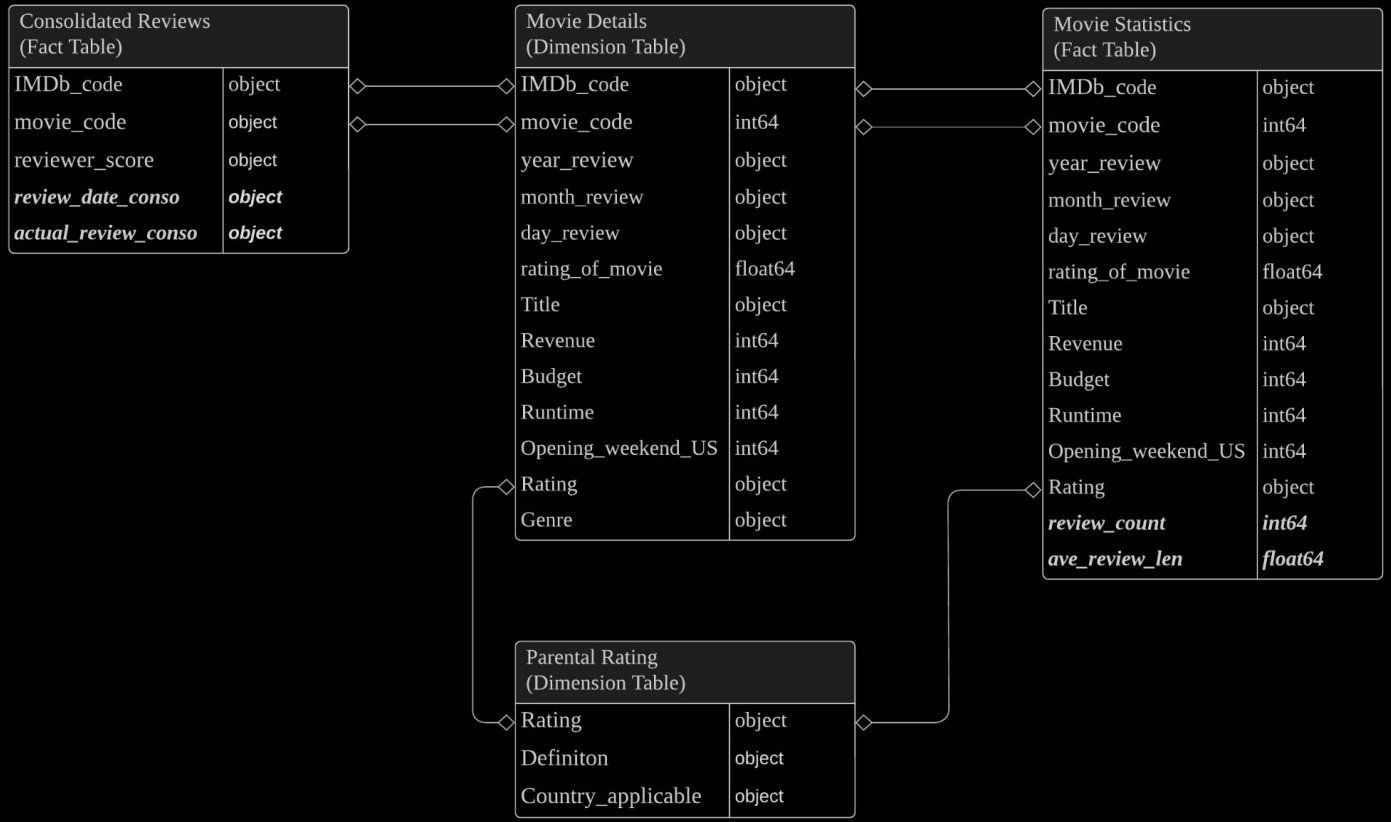
Movie reviews

```
for review in reviews:
    review_data.append({
        'movie_code': tmc
        'IMDb_code': imdb
        'review_date': re
        'rating_of_movie'
        'actual_review':
    })
```





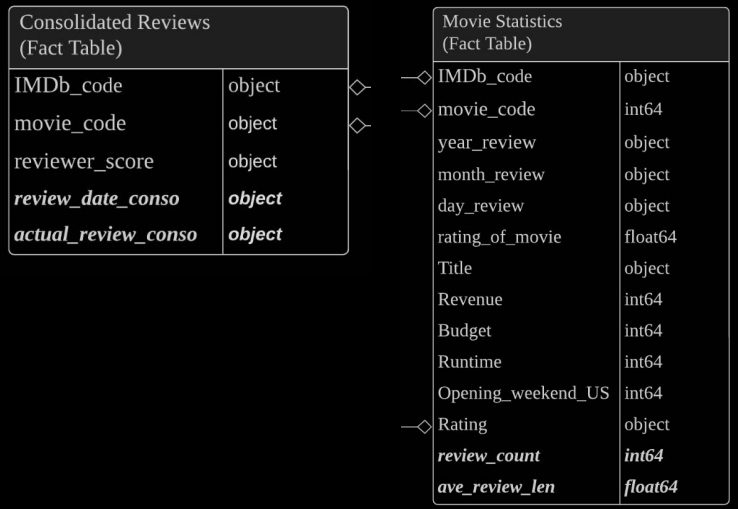
DATA ARCHITECTURE - CONSTELLATION SCHEMA (STAR)



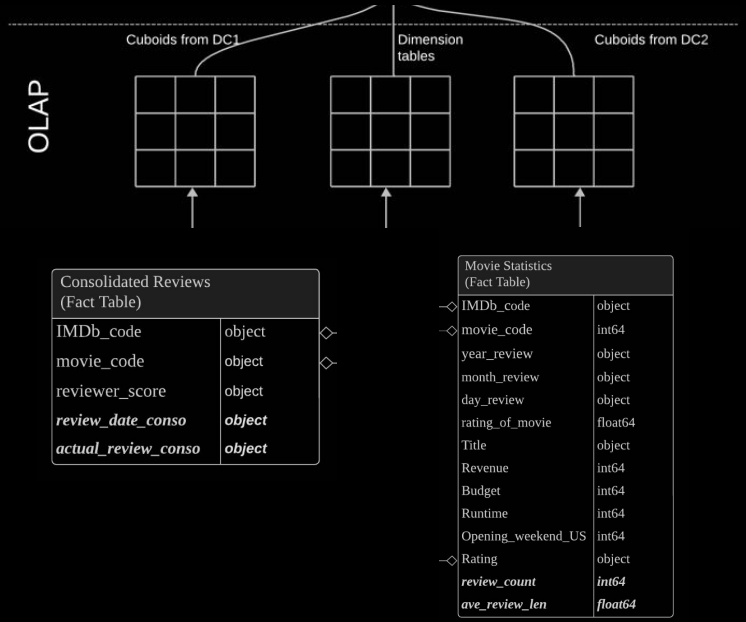
DATA ARCHITECTURE - CONSTELLATION SCHEMA (STAR)

Advantages using constellation schema

1. The user will have to access lesser data volume depending on the task (NLP vs basic movie analytics)
2. Data are broken down into smaller data cubes (e.g. movie statistics and movie reviews)
3. Still has the flexibility of connecting and processing both fact tables



DATA ARCHITECTURE - OLAP



Initial OLAP operations

- Movie reviews table (bottom left) can give all the reviews per movie and reviewer score. All the stop words are removed.
- Movie statistics table (bottom right) can readily give the average review length and count of reviews per movie, release date, rating of movie.
- Revenue, budget, genre, runtime and parental rating are added for easy access.

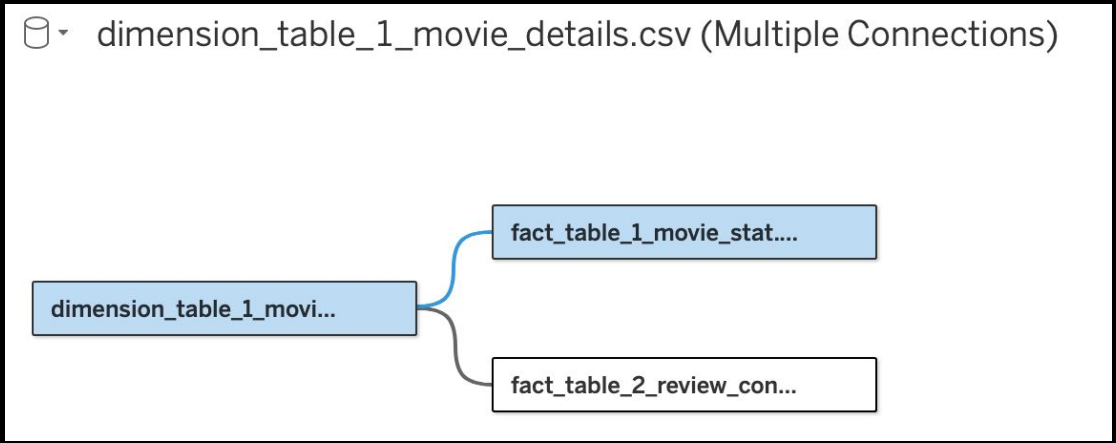
IMDb

DASHBOARD DEMO

https://public.tableau.com/app/profile/lizelle.cruz/viz/IMDBDashboard_17223681730490/IMDBDashboard?publish=yes

DATA SOURCE CONNECTION

We linked the dimension table and fact tables using the IMDb code in order to build dashboards taking variables from across the three data sources.



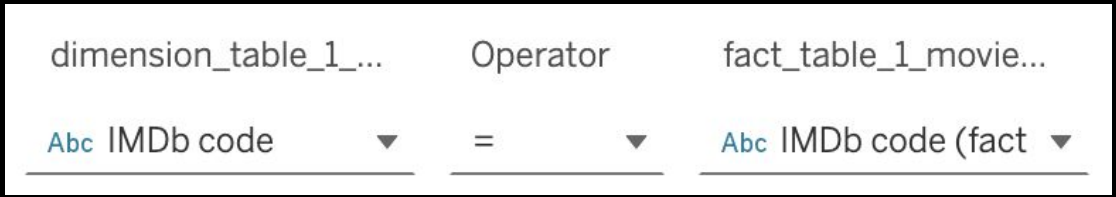
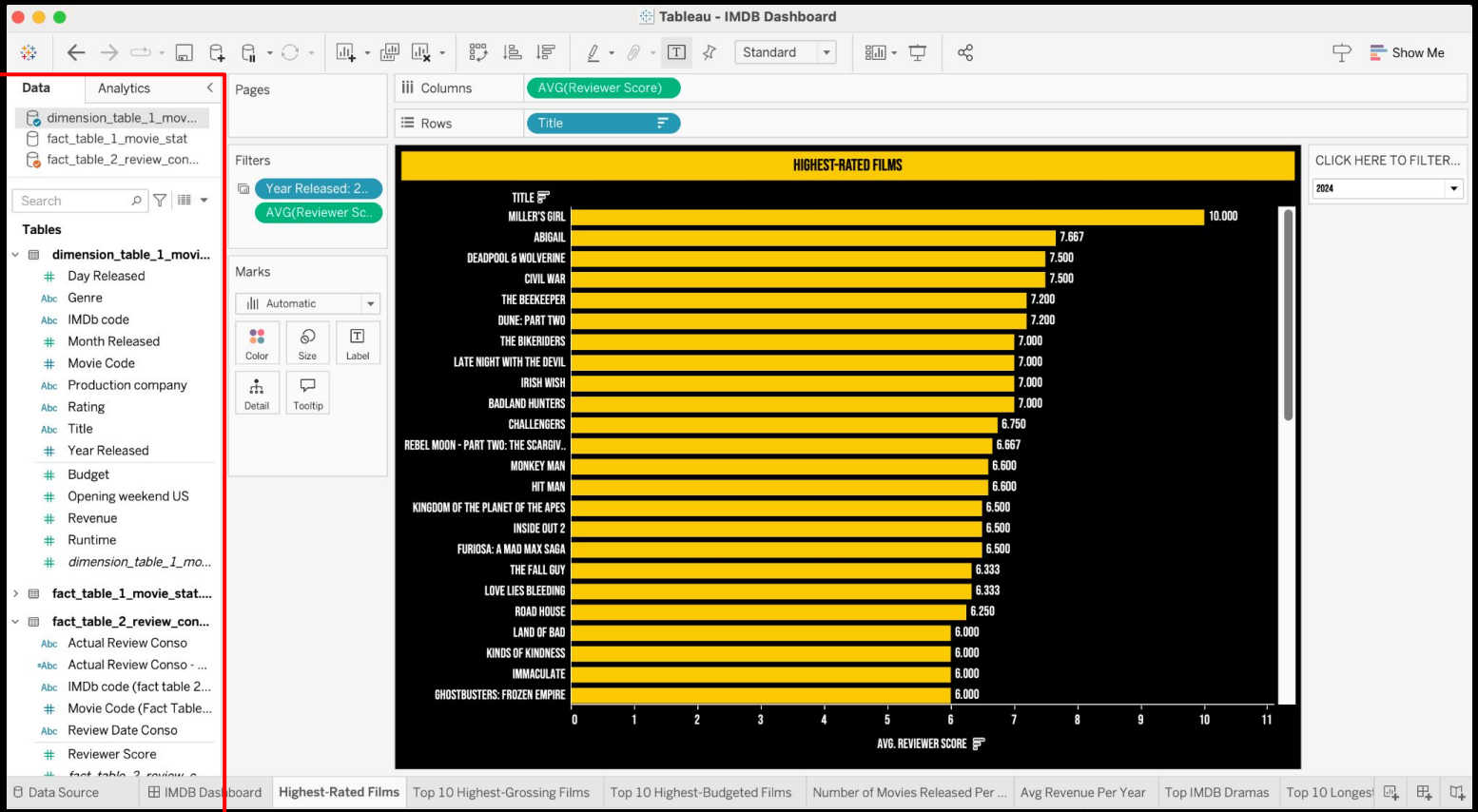


TABLEAU WORKBOOK

IMDb



DASHBOARD

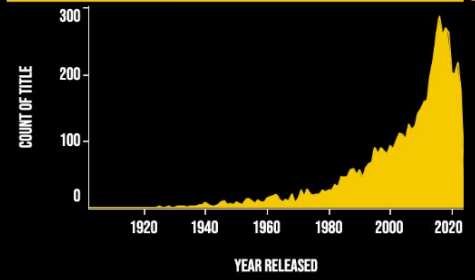
IMDb

IMDb MOVIE DASHBOARD

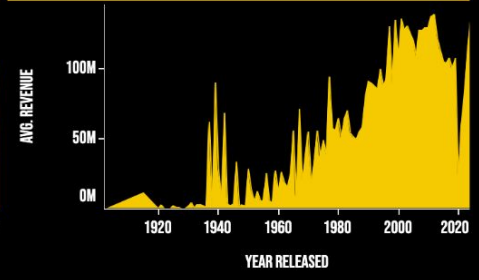
CLICK HERE TO FILTER TOP FILMS BY YEAR

2024

NUMBER OF MOVIES RELEASED PER YEAR



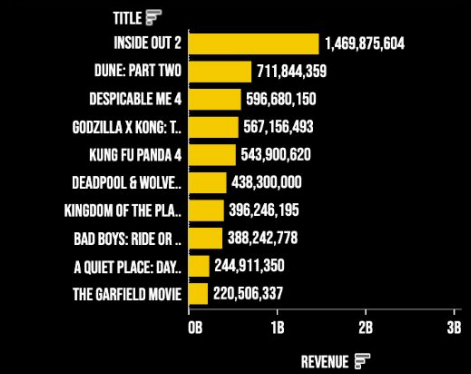
AVERAGE MOVIE REVENUE PER YEAR



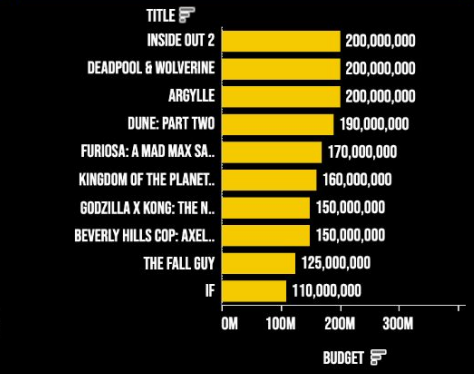
TOP IMDB GENRES

MOST USED WORDS IN REVIEWS

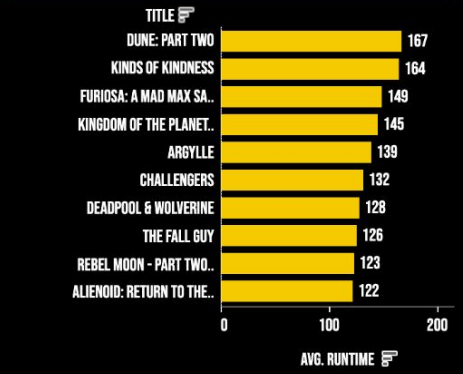
TOP 10 HIGHEST-GROSSING FILMS



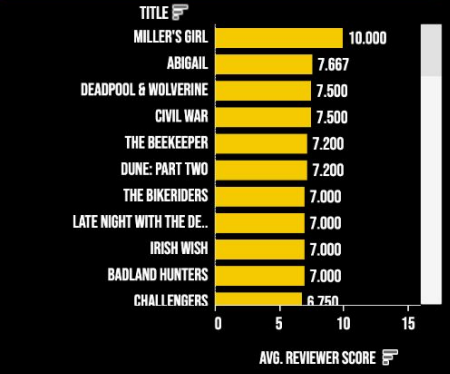
TOP 10 HIGHEST-BUDGETED FILMS



TOP 10 LONGEST FILMS

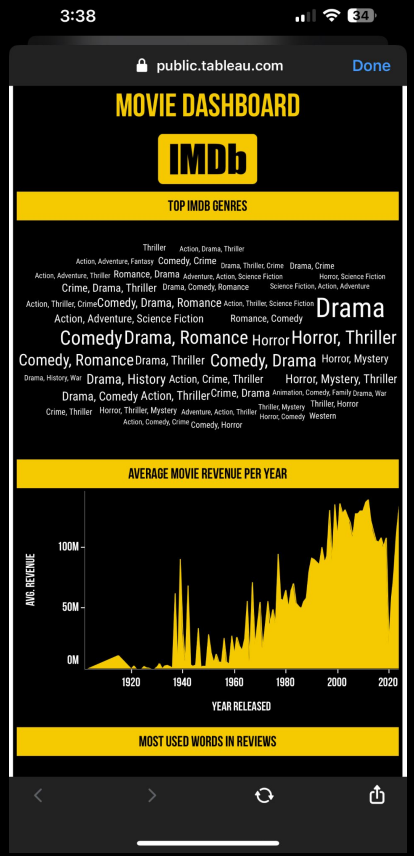


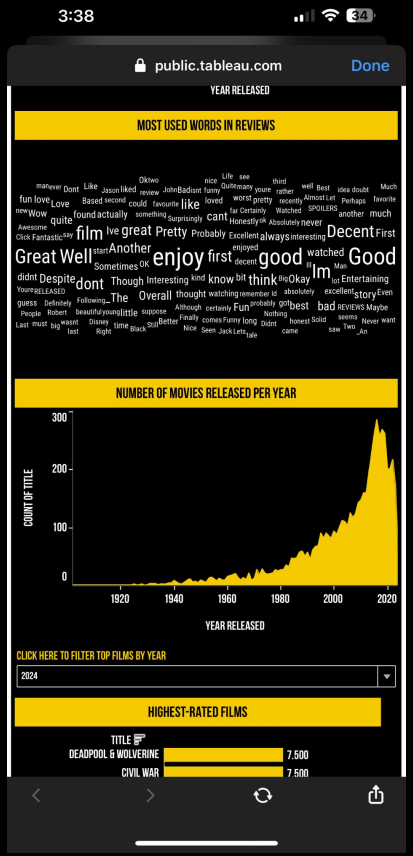
HIGHEST-RATED FILMS

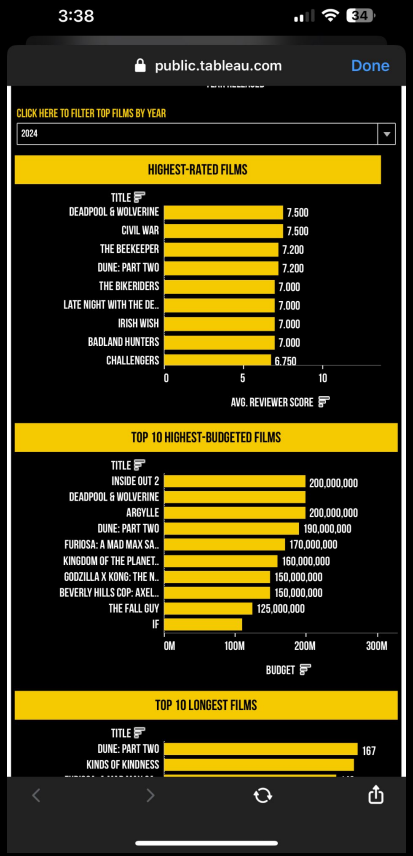


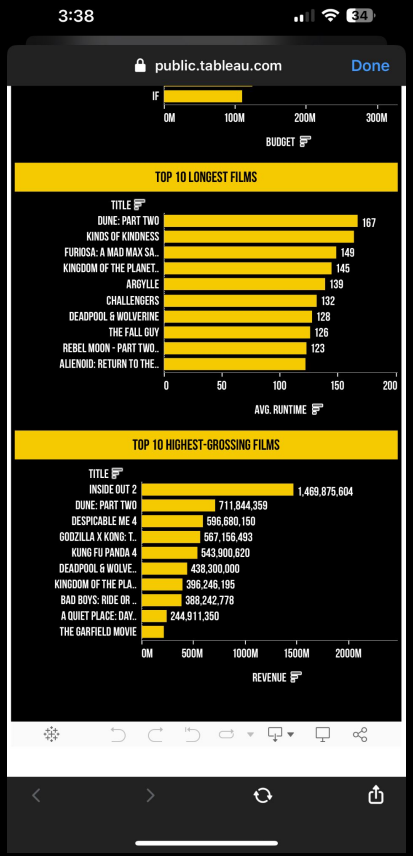
DASHBOARD - MOBILE VIEW

IMDb







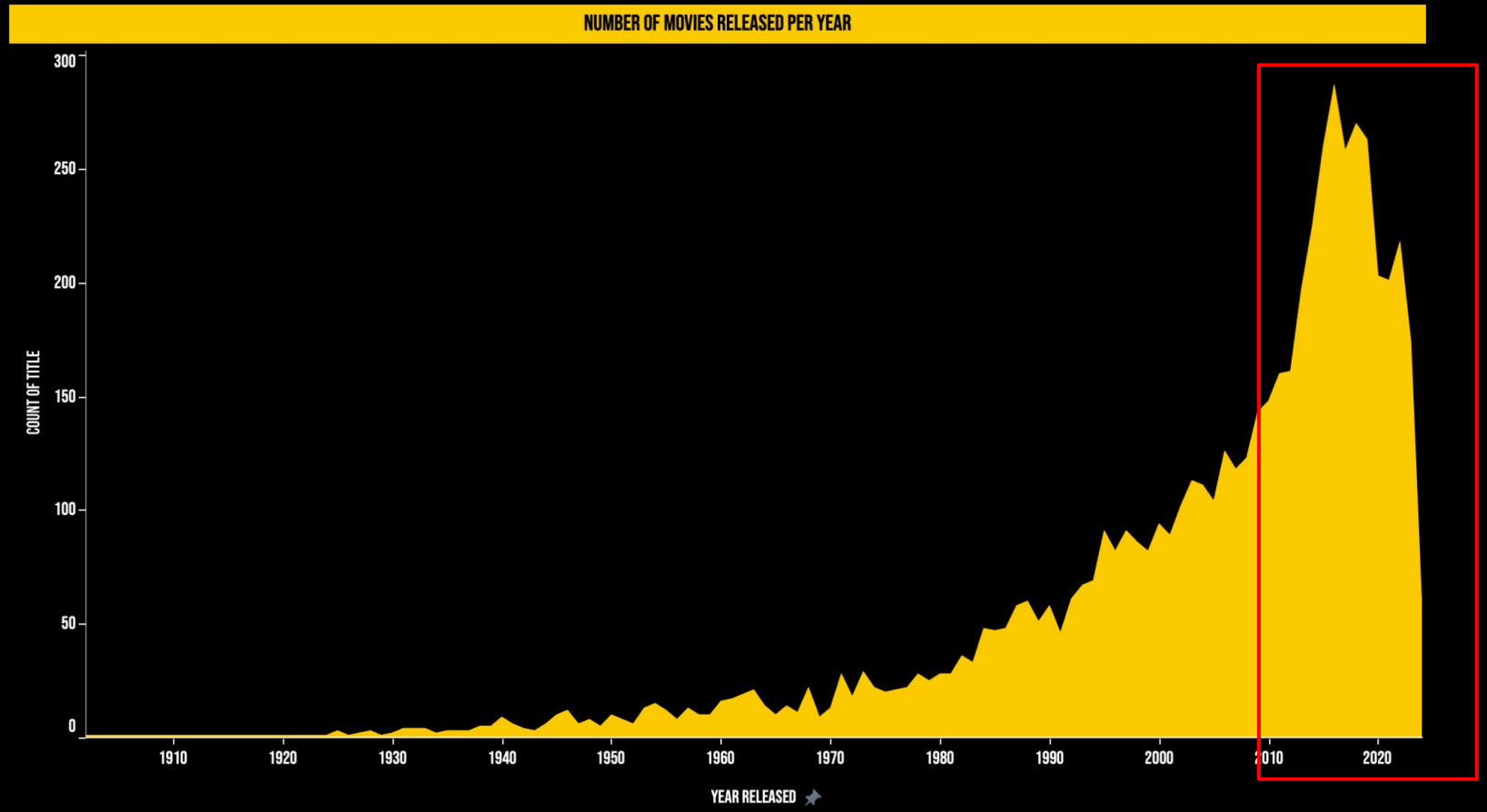


IMDb

DATA ANALYSIS AND INSIGHTS

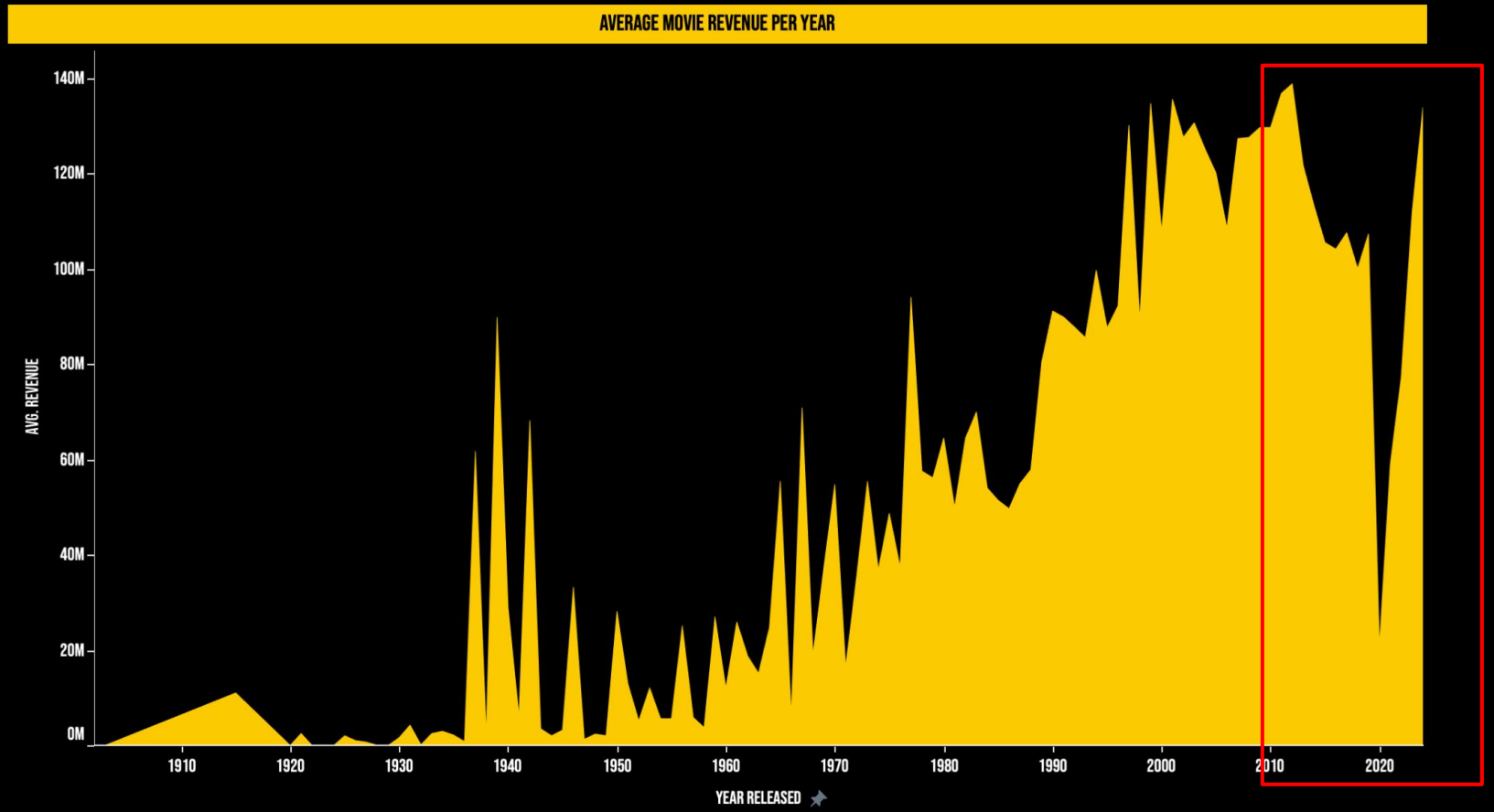
THE PANDEMIC AFFECTED THE FILM INDUSTRY

IMDb



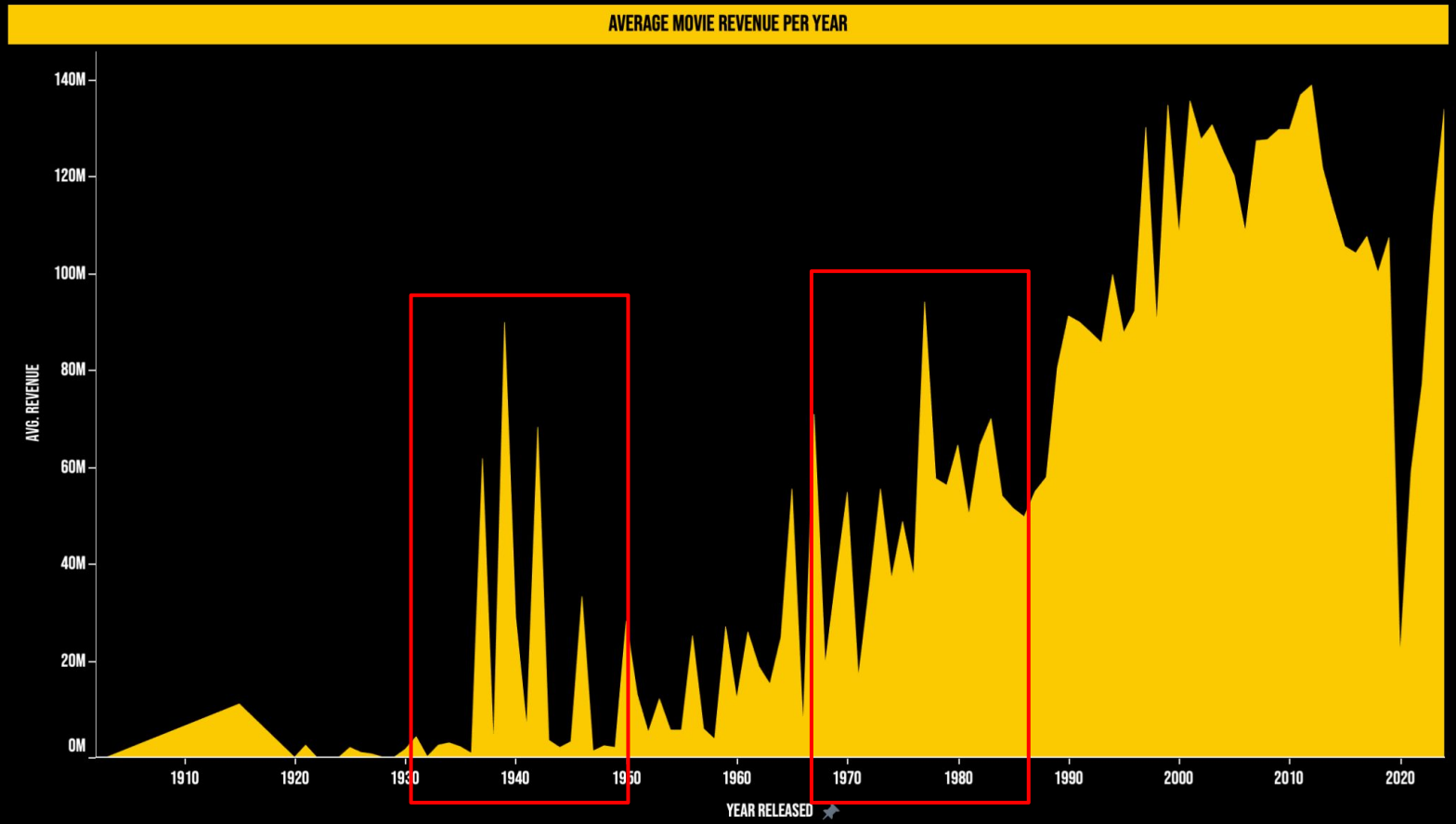
THE PANDEMIC AFFECTED THE FILM INDUSTRY

IMDb



PEAKS IN 1939 AND 1977

IMDb



DRAMA IS THE PREVAILING GENRE IN THE DATABASE

IMDb

TOP IMDB GENRES

THERE IS GENERALLY A POSITIVE SENTIMENT TOWARDS MOVIES

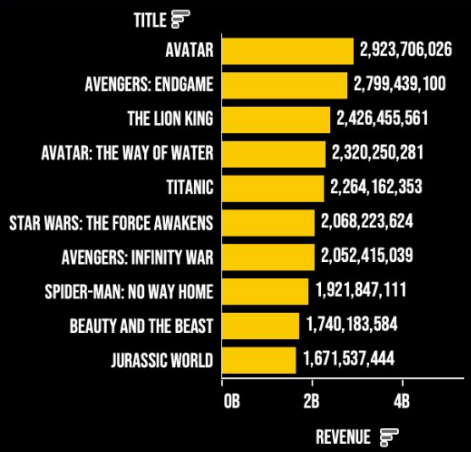
IMDb

MOST USED WORDS IN REVIEWS

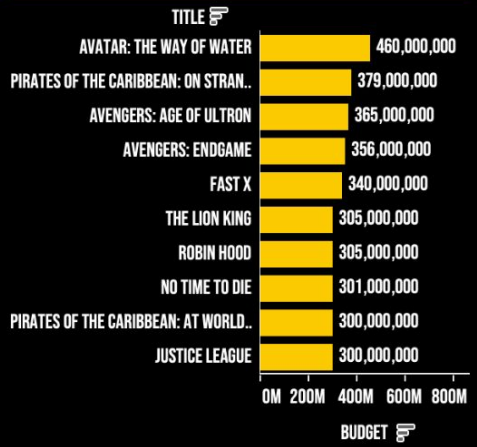
ALL-TIME STATISTICS: MOVIE FRANCHISES PREVAIL

IMDb

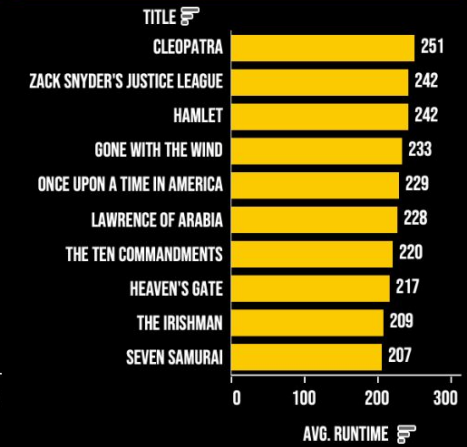
TOP 10 HIGHEST-GROSSING FILMS



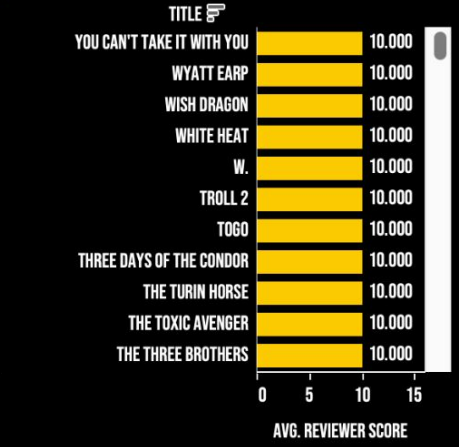
TOP 10 HIGHEST-BUDGETED FILMS

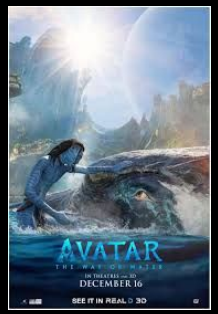


TOP 10 LONGEST FILMS



HIGHEST-RATED FILMS

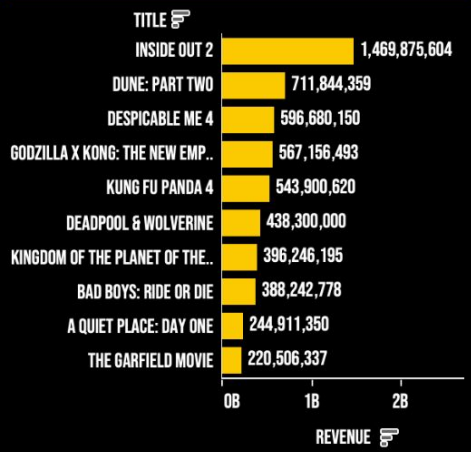




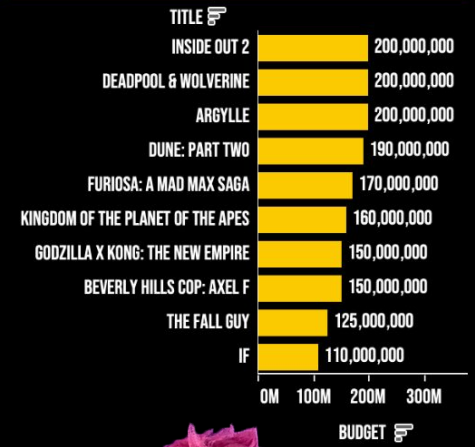
2024 STATISTICS

IMDb

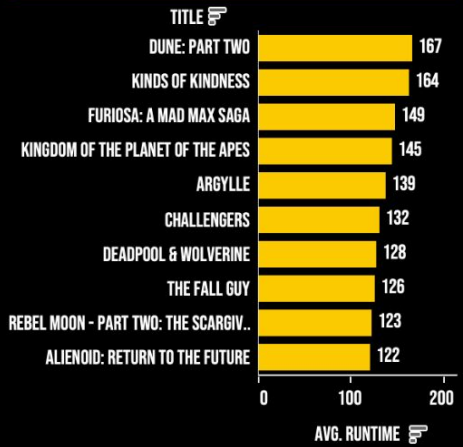
TOP 10 HIGHEST-GROSSING FILMS



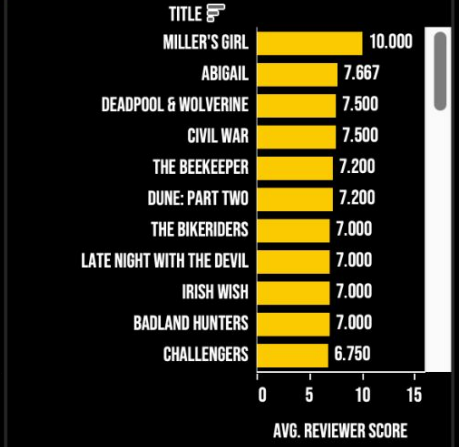
TOP 10 HIGHEST-BUDGETED FILMS



TOP 10 LONGEST FILMS



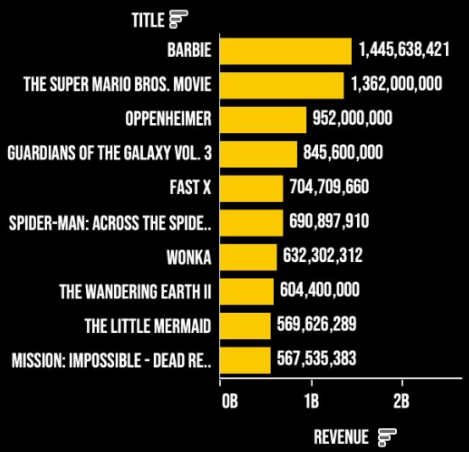
HIGHEST-RATED FILMS



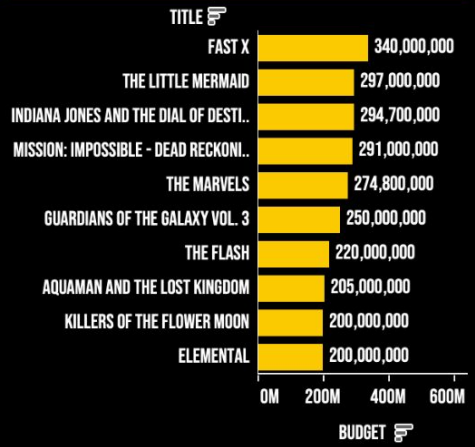
2023 STATISTICS

IMDb

TOP 10 HIGHEST-GROSSING FILMS

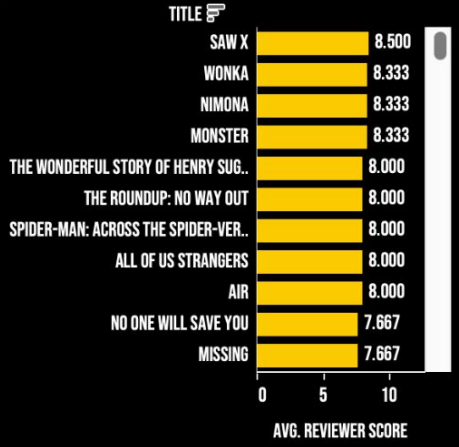


TOP 10 HIGHEST-BUDGETED FILMS



TOP 10 LONGEST FILMS

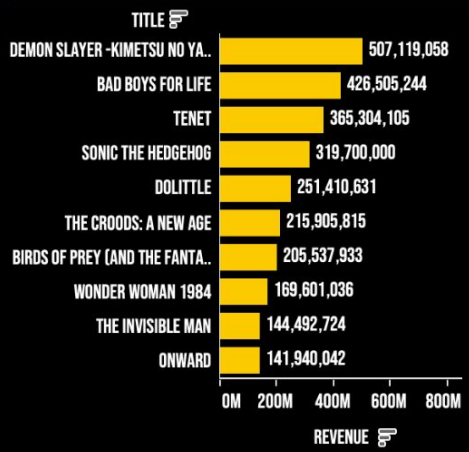
HIGHEST-RATED FILMS



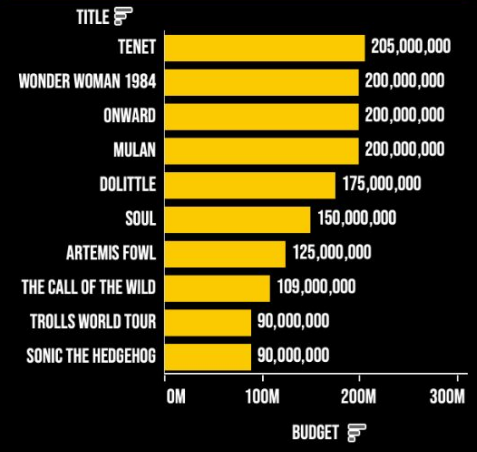
2020: NO MOVIE REACHED 1 BILLION IN REVENUE

IMDb

TOP 10 HIGHEST-GROSSING FILMS

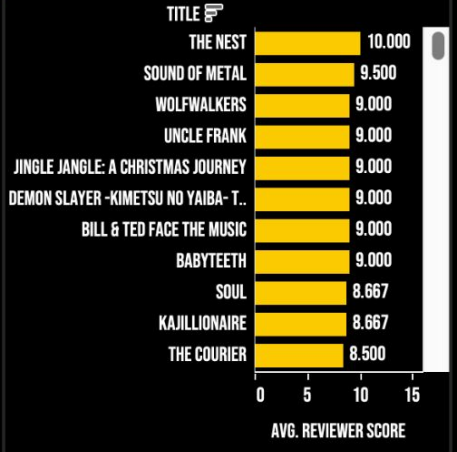


TOP 10 HIGHEST-BUDGETED FILMS



TOP 10 LONGEST FILMS

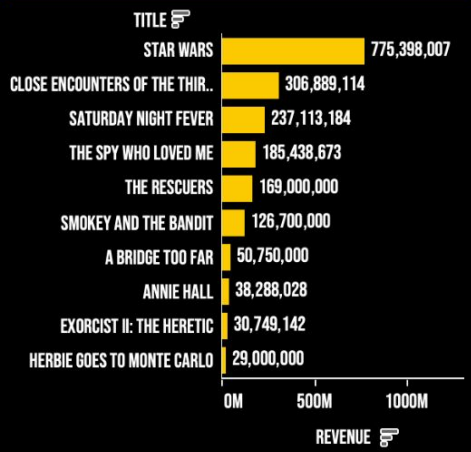
HIGHEST-RATED FILMS



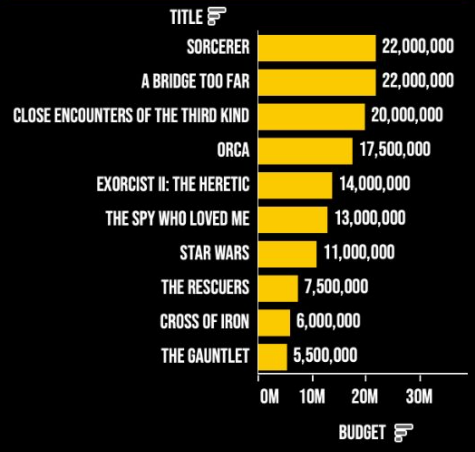
1977: THE RISE OF STAR WARS

IMDb

TOP 10 HIGHEST-GROSSING FILMS

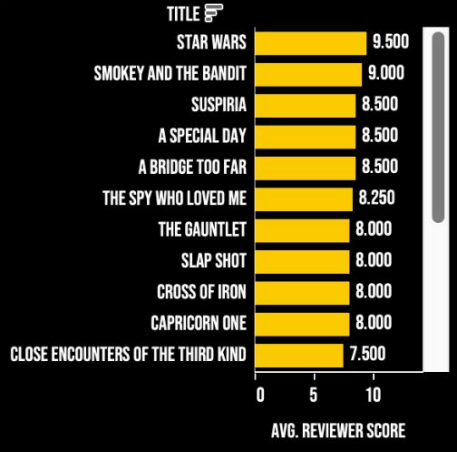


TOP 10 HIGHEST-BUDGETED FILMS



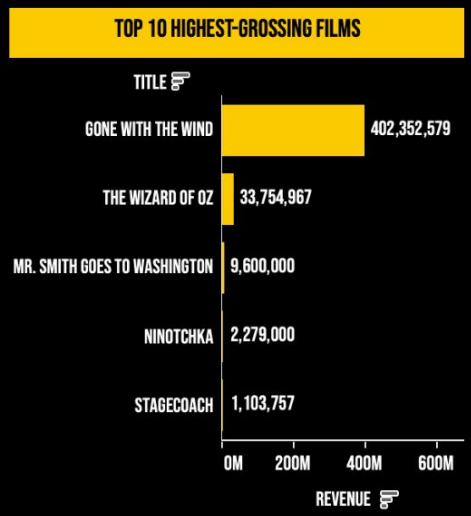
TOP 10 LONGEST FILMS

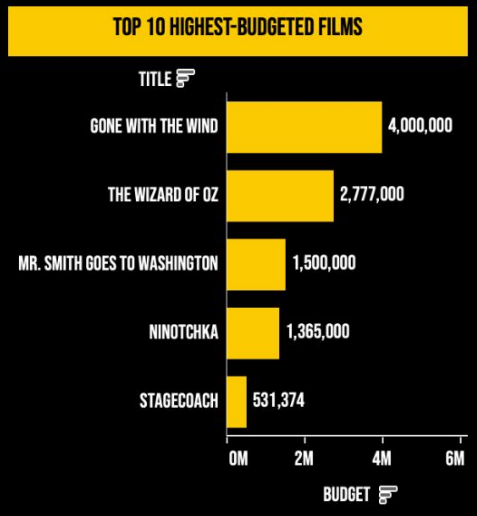
HIGHEST-RATED FILMS

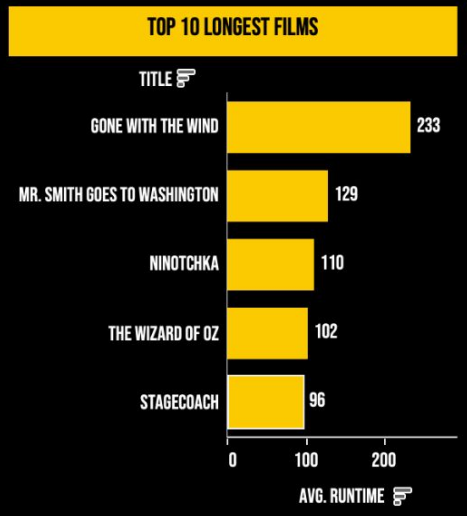


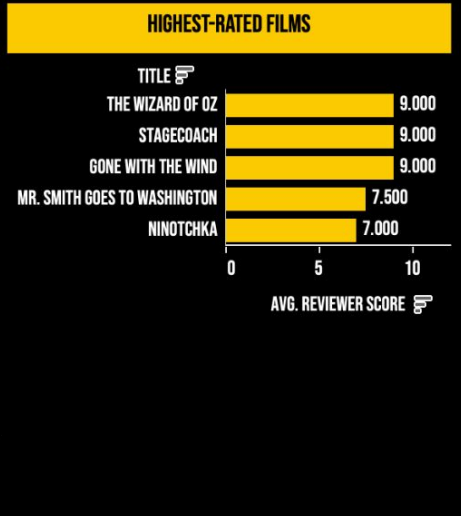
1939: GONE WITH THE WIND

IMDb









Gone with the Wind, American epic film, released in 1939, that was one of the best known and most successful films of all time. It enjoyed a more-than-30-year reign as the all-time Hollywood box office champion, and it won eight Academy Awards (in addition to two honorary awards). Based on the runaway best-selling 1936 novel *Gone with the Wind* by Margaret Mitchell, the movie is almost four hours long and includes an intermission.

IMDb

LEARNINGS AND BLOCKERS

LEARNINGS AND BLOCKERS

LEARNINGS	BLOCKERS
• Utilize pickle module • Understanding Movie as Business • Understanding Schema and design	• Tableau Licensing • Stop Words for Review Word Cloud • Data warehouse designing • Utilize various python modules

IMDb

THANK YOU!