Table of Contents

1. Movie Recommendation

1.1 Introduction

In today's day and age, it is very common to see a smartphone at every corner of the world. It has become central and important to our everyday lives (Skierkowski and Wood, 2012). The capabilities of this technology includes networking, messaging and entertainment. Entertainment in a form of streaming platforms have cause a shift in movies-goers and allowing them to enjoy their favourite movie or TV shows at the comfort of their home or during commute (Jaremko-Greenwold, 2024). With the ease of the smartphone technology, people are able to watch movies anytime and anywhere.

Disney+ and Netflix are some streaming platform that can be used on a smartphone, tablet and smart tv. This allow seamless connection from one device to another and continuing from where it was last stop. Another feature that hook users to the continue using the app is the recommendations section. A curated list of movies or shows that is unique to every user, that uses the user's previous watched titles and return recommended movies or shows that they might like.

1.2 Objective

The objective for this study is to find similar connection to different movie title and create a recommendation system that can recommend similar movies based from a movie. This will help to understand the backend works using algorithm to recommend movies on the streaming platform.

1.3 Dataset Loading

The dataset used in this project is the TMDB 5000 Movie Dataset downloaded from the kaggle website. The dataset consist of 2 csv file, a movies file and a credits file. It was first uploaded on google drive where I used the google colab to get the dataset. Loading the file from google drive requires an additional step of mounting the drive first before able direct the path to the location of the csv file as shown in figure 1.

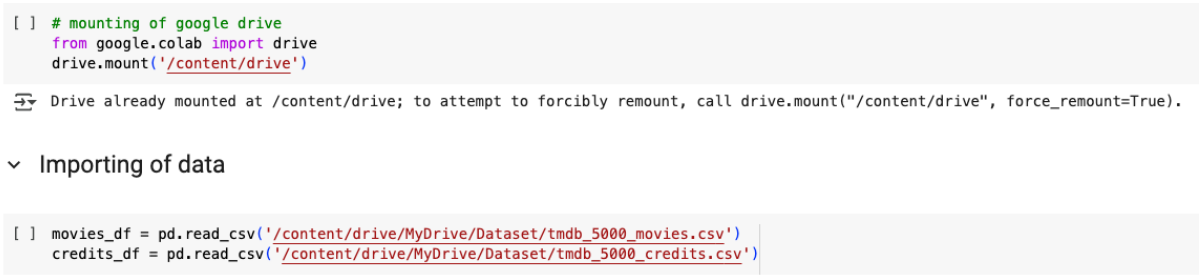


Figure 1: Mounting of drive and importing of data

1.4 Data Understanding

```
Shape of the Movies dataset: (4803, 20)

Movies Dataset info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   budget                4803 non-null   int64
 1   genres                4803 non-null   object
 2   homepage              1712 non-null   object
 3   id                    4803 non-null   int64
 4   keywords              4803 non-null   object
 5   original_language     4803 non-null   object
 6   original_title        4803 non-null   object
 7   overview              4800 non-null   object
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object
 10  production_countries  4803 non-null   object
 11  release_date          4802 non-null   object
 12  revenue               4803 non-null   int64
 13  runtime               4801 non-null   float64
 14  spoken_languages      4803 non-null   object
 15  status                4803 non-null   object
 16  tagline               3959 non-null   object
 17  title                 4803 non-null   object
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB
```

```
Shape of the Credits dataset: (4803, 4)

Credits Dataset info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   movie_id  4803 non-null   int64
 1   title     4803 non-null   object
 2   cast      4803 non-null   object
 3   crew      4803 non-null   object
dtypes: int64(1), object(3)
memory usage: 150.2+ KB
```

Figure 2: Dataset information

After loading the dataset, the next step was to understand the data. By running the shape and info function on the datasets, the results can be outputted as shown in figure 2. It shows that movie dataset consist of 4803 rows and 20 columns while the credits dataset consists of 4803 rows and 4 columns. As both consist the same number of rows, both dataset can be merged together in the later steps. Before merging both dataset, I also checked if there are any duplicate values to ensure that the both dataset can be merged on movie id column. The results return to have no duplicate values in both dataset.

```
# check for any duplicates
print("Duplicate values in movies dataset:", movies_df.duplicated().sum())
print("Duplicate values in credits dataset:", credits_df.duplicated().sum())
```

```
Duplicate values in movies dataset: 0
Duplicate values in credits dataset: 0
```

Figure 3: Checking for duplicate values

1.5 Data preparation

```
# rename credit dataset movie_id column to id
credits_df.rename(columns={'movie_id': 'id'}, inplace=True)
```

```
# confirm the rename column
credits_df.head()
```

	id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

```
# merge movies and credits dataset
movies_df = movies_df.merge(credits_df, on='id')
```

Figure 4: Merging both dataset

As confirmed that there were no duplicated values, merging both dataset can be done. Figure 4 shows that the 'movie_id' column title from the credits dataset renamed to 'id' so to match the column title from the movies dataset. Both dataset are then merged together on 'id'.

budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_compa
237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Inge Film Partners",
300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt D Pictures", "id": 2
245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Colu Pictures", "id {"n
250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950	[{"name": "Leger Pictures", "id":
260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	[{"name": "Walt D Pictures", "id

Figure 5: Updated Dataset

After merging both dataset, I notice some columns data are in Json-file format. As seen in figure 5, the 'genres' and 'production_companies' are in this kind of format. Therefore, I used a regular expression operation to go through the json-like data and return the string. I applied this function on 'genres', 'production_companies' column. As there was a 'crew' column in the dataset, extracting out the director of the movie can be useful for a recommender system. Therefore I apply the code that will extract the name when the job is equal to director.

```
# turning json format column into a list
import re

def extract(s):
    # Check if the input string is not empty and contains valid JSON-like data
    if s and isinstance(s, str) and '{' in s and '}' in s:
        pattern = r'"name": "(.*?)"'
        return '|'.join(re.findall(pattern, s))
    else:
        return 'Unknown'
```

```
# applying extract to genre column
movies_df['genres'] = movies_df['genres'].apply(extract)

# extracting production company
movies_df['production_companies'] = movies_df['production_companies'].apply(extract)

# extracting name of director from crew column
movies_df['director'] = movies_df['crew'].apply(lambda x: [i['name'] for i in eval(x) if i['job'] == 'Director'])
```

Figure 6: Extracting relevant columns

genres	production_companies	director
ActionlAdventurelFantasylScience Fiction	Ingenious Film PartnerslTwentieth Century Fox ...	[James Cameron]
AdventurelFantasylAction	Walt Disney PictureslJerry Bruckheimer FilmslS...	[Gore Verbinski]
ActionlAdventurelCrime h	Columbia PictureslDanjaqlB24	[Sam Mendes]
ActionlCrimelDramalThriller	Legendary PictureslWarner Bros.lDC Entertainme...	[Christopher Nolan]
ActionlAdventurelScience Fiction	Walt Disney Pictures	[Andrew Stanton]

Figure 7: Returned outputs

Extracting the year from the release date was also done as I needed to used it during the exploratory data analysis step. Figure 8 shows the code to extract the release year and the results for the movies dataset. As there were a few columns with duplicate 'title' column, I have also dropped the duplicates leaving only the 'original_title' column.

```
# get the year from the release date
movies_df['release_year'] = pd.to_datetime(movies_df['release_date']).dt.year

# get the month from the release date
movies_df['release_month'] = pd.to_datetime(movies_df['release_date']).dt.month

# set the year and month to whole numbers
movies_df['release_year'] = movies_df['release_year'].astype(int)
movies_df['release_month'] = movies_df['release_month'].astype(int)
```

release_year	release_month
2009	12
2007	5
2015	10
2012	7
2012	3

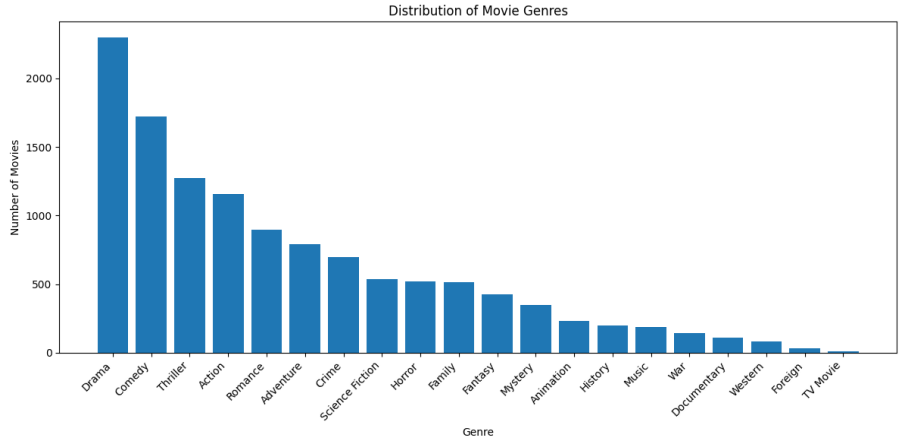
Figure 8: Code and output for extracting year and month

```
# title_x  and title_y are the same to original title, therefore drop these columns
movies_df.drop(['title_x', 'title_y'], axis=1, inplace=True)
```

Figure 9: Dropping duplicated columns

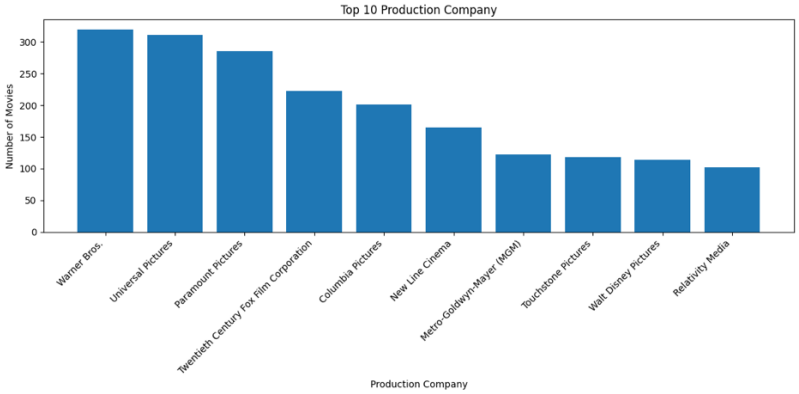
1.6 Exploratory Data Analysis (EDA)

Following the preparation of the dataset, EDA can be performed on the movies dataset which can be observed in the following figures.



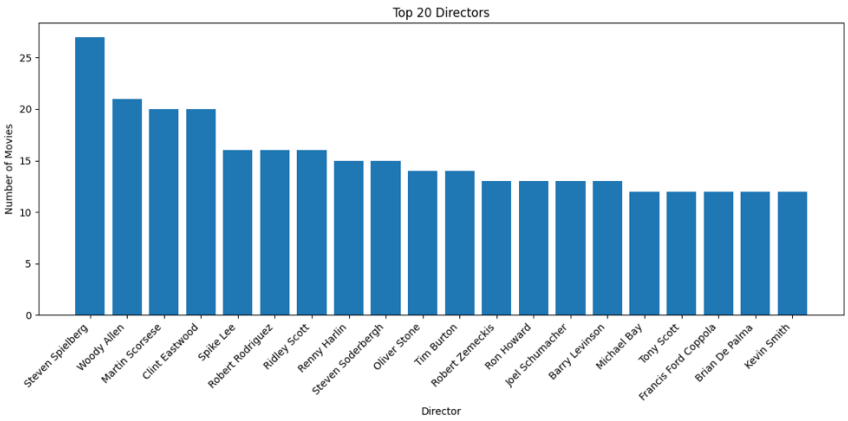

```
Count of each genres:
 Drama              2297
Comedy              1722
Thriller            1274
Action              1154
Romance              894
Adventure            790
Crime                696
Science Fiction      535
Horror               519
Family               513
Fantasy              424
Mystery              348
Animation            234
History              197
Music                185
War                  144
Documentary          110
Western               82
Foreign               34
TV Movie               8
```

Figure 10: Distribution of movie genres




```
Top 10 Production Companies:
 Warner Bros.                          319
Universal Pictures                     311
Paramount Pictures                     285
Twentieth Century Fox Film Corporation 222
Columbia Pictures                      201
New Line Cinema                        165
Metro-Goldwyn-Mayer (MGM)              122
Touchstone Pictures                    118
Walt Disney Pictures                   114
Relativity Media                       102
```

Figure 11: Top 10 Production Company




```
Top 20 Directors:
 Steven Spielberg    27
Woody Allen          21
Martin Scorsese      20
Clint Eastwood       20
Spike Lee            16
Robert Rodriguez     16
Ridley Scott         16
Renny Harlin         15
Steven Soderbergh    15
Oliver Stone         14
Tim Burton           14
Robert Zemeckis      13
Ron Howard           13
Joel Schumacher      13
Barry Levinson       13
Michael Bay          12
Tony Scott           12
Francis Ford Coppola 12
Brian De Palma       12
Kevin Smith          12
```

Figure 12: Top 20 Directors

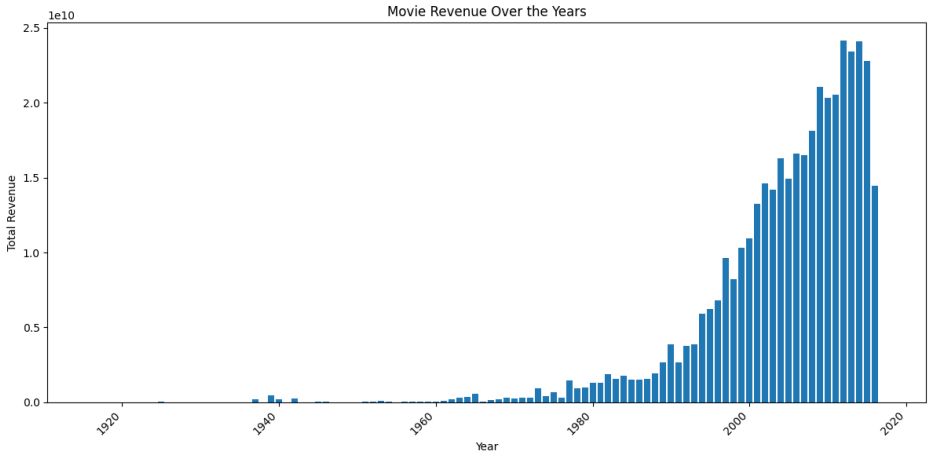
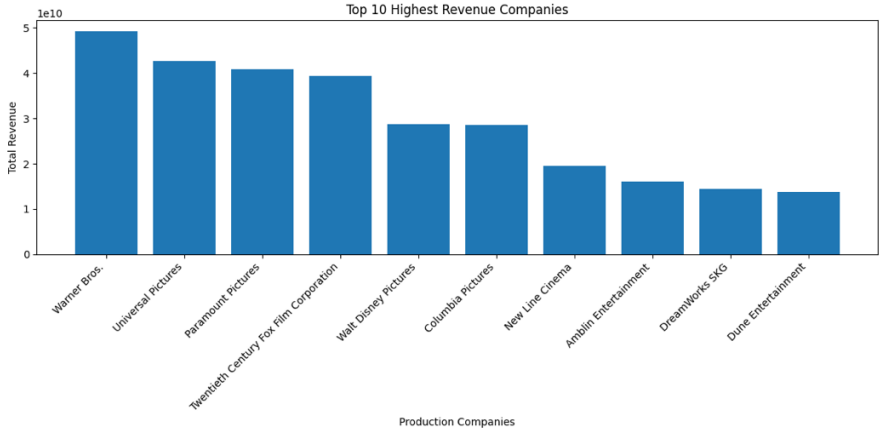


Figure 13: Movie Revenue over the years




```
Top 10 Production Companies:
 Warner Bros.                          49155747874
Universal Pictures                     42588465942
Paramount Pictures                     40878523165
Twentieth Century Fox Film Corporation 39357151309
Walt Disney Pictures                   28683256048
Columbia Pictures                      28599634775
New Line Cinema                        19444865804
Amblin Entertainment                   16090835147
DreamWorks SKG                         14384533626
Dune Entertainment                     13797504190
```

Figure 14: Top 10 production companies with highest revenue

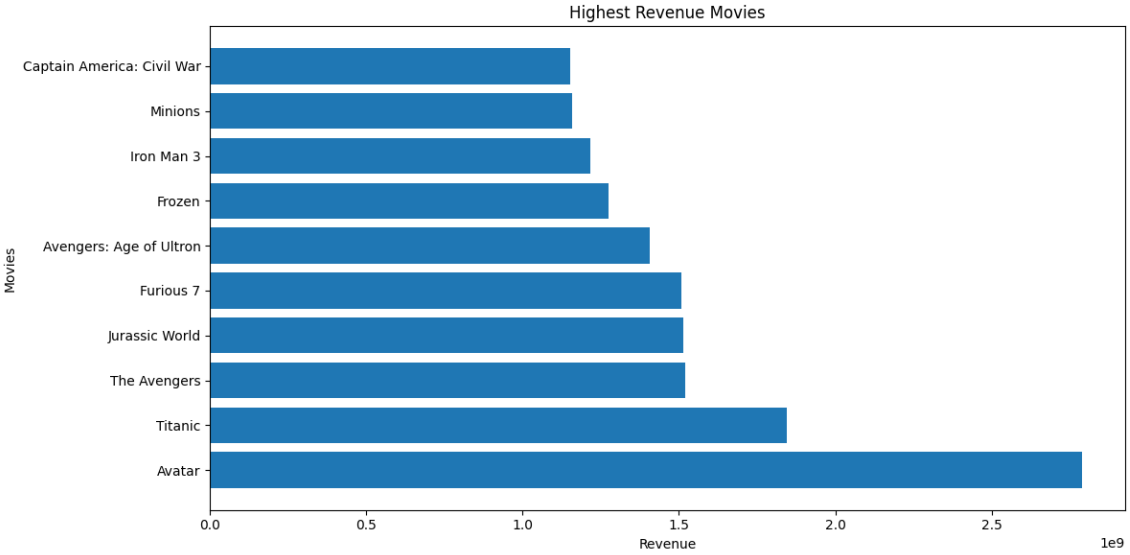


Figure 15: Top 10 Highest Revenue Movies

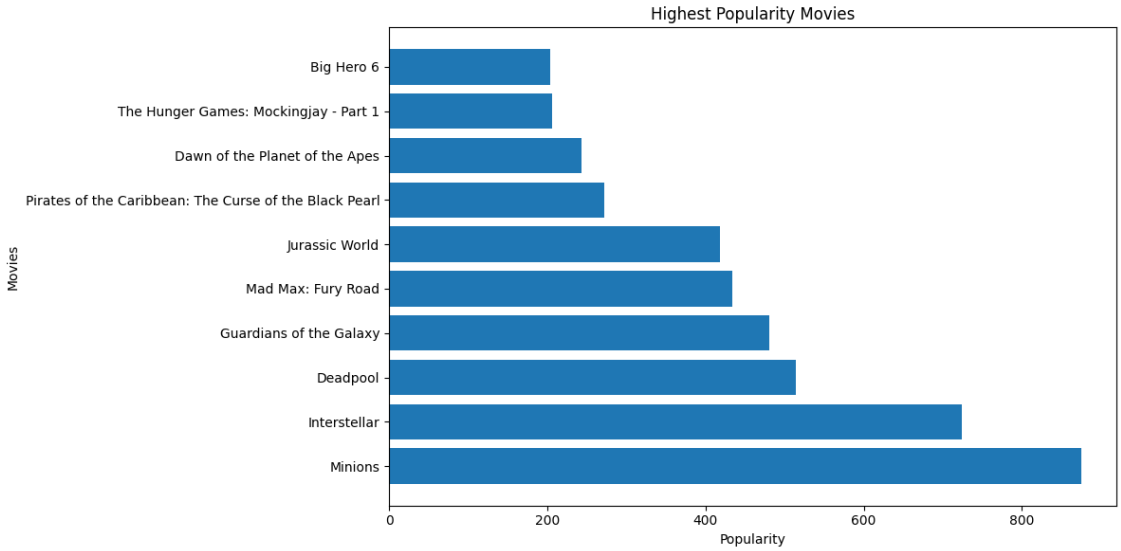


Figure 16: Top 10 Most popular movies

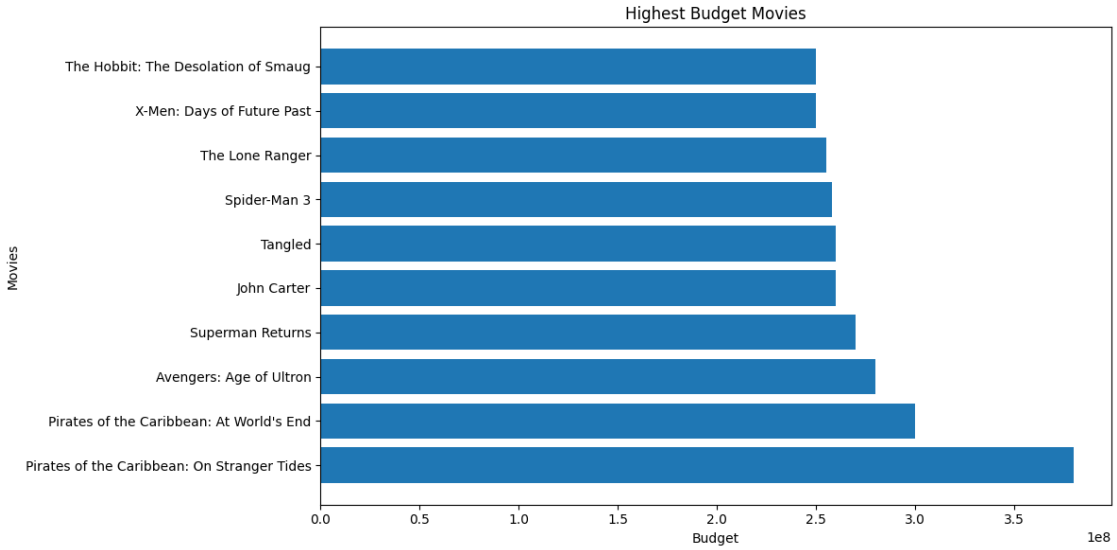


Figure 17: Top 10 Movies with the most budget

From analysing the bar plots, about half of the movie dataset are categorised as drama for the genre with 2297 count. Comedy and thriller are the next highest count for genres. It can also be observed that Steven Spielberg has directed the most movies with the count of 27. Over the years, there is an upward trend in movie revenue as seen in figure 13. Therefore I analyse the revenue trend on production companies and movies.

Firstly was to analyse the top 10 production company. From figure 11, the company that produced the most movies is Warner Bros with 319 movies, followed closely by Universal Pictures and Paramount Pictures with 311 and 285 movies produced respectively. After, I proceed to compare the revenue generated by these companies, which also give the same ranking of Warner Bros, Universal Pictures and Paramount Pictures. This could also relate to the more movies produced which brought in more revenue. Next, I compare the revenue, budget and popularity within each movie. Avatar have the highest movie revenue, generating over 2.5 billion. However Avatar was produced by Twentieth Century Fox, although isn't in the top 3, it is the fourth highest production company in revenue. Minions have garnered the highest popularity among all the other movies. The 2 Pirates of the Caribbean movie franchise have the highest budget with over 300 hundred million.

1.7 Movie Recommender System

1.7.1 Recommender System Building

```
[ ] # Selecting the needed columns as a copy
    movies2_df = movies_df[['id', 'original_title', 'production_companies', 'director', 'genres']].copy()

    # confirmation
    movies2_df.head()
```

	id	original_title	production_companies	director	genres
0	19995	Avatar	Ingenious Film Partners|Twentieth Century Fox ...	[James Cameron]	Action|Adventure|Fantasy|Science Fiction
1	285	Pirates of the Caribbean: At World's End	Walt Disney Pictures|Jerry Bruckheimer Films|S...	[Gore Verbinski]	Adventure|Fantasy|Action
2	206647	Spectre	Columbia Pictures|Danjaq|B24	[Sam Mendes]	Action|Adventure|Crime
3	49026	The Dark Knight Rises	Legendary Pictures|Warner Bros.|DC Entertainme...	[Christopher Nolan]	Action|Crime|Drama|Thriller
4	49529	John Carter	Walt Disney Pictures	[Andrew Stanton]	Action|Adventure|Science Fiction

Figure 18: Feature Selection

Building of a movie recommender system was done after all the analysis and selecting the relevant columns to that can help recommend movies to the user. The columns that can be used were the 'production companies', 'director' and 'genres'. A copy of the selected columns were created as a new data frame as shown in figure 18.

Count Vectorization imported from the sklearn library helps to convert the a collection of text into numerals. For this case, the count vectorizer was use on the 3 selected columns ('genres', 'production company' & 'director').

```
# fit director data into matrix
from sklearn.feature_extraction.text import CountVectorizer

#Define a new CountVectorizer object and create vectors for the director
count_vectorizer = CountVectorizer(stop_words='english')

#Replace NaN with an empty string
movies2_df['genres'] = movies2_df['genres'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data
genres_count_matrix = count_vectorizer.fit_transform(movies2_df['genres'])

# get the shape of matrix
genres_count_matrix.shape
```

```
(4802, 23)
```

```
[ ] # Convert the 'director' column to a string representation before fitting
    movies2_df['director'] = movies2_df['director'].apply(lambda x: ' '.join(x) if isinstance(x, list) else str(x))

    # Transform the data
    director_count_matrix = count_vectorizer.fit_transform(movies2_df['director'])

    # Fitting to cosine_sim
    director_cosine_sim = cosine_similarity(director_count_matrix, director_count_matrix)
```

```
[ ] # setting on production companies
    pc_count_matrix = count_vectorizer.fit_transform(movies2_df['production_companies'])

    # Fitting to cosine_sim
    pc_cosine_sim = cosine_similarity(pc_count_matrix, pc_count_matrix)
```

Figure 19: Count Vectorizer

The metrics used help to provide the movie recommendation was by the cosine similarity. Cosine similarity is useful in text analysis which can aid in analysing the text data of the columns. A function was created that recommends movie based on the given title and using the different cosine similarity to provide the recommended movies.

```python
# Function that takes in movie title as input and outputs most similar movies
def movies_recommendations(title, cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return movies2_df['original_title'].iloc[movie_indices]
```

Figure 20: Function to recommend movies

1.7.2 Recommend movies based on genres

```python
# testing the system
movies_recommendations('Superman Returns', genres_cosine_sim)
```

	original_title
10	Superman Returns
14	Man of Steel
46	X-Men: Days of Future Past
61	Jupiter Ascending
232	The Wolverine
813	Superman
870	Superman II
3494	Beastmaster 2: Through the Portal of Time
72	Suicide Squad
238	Teenage Mutant Ninja Turtles

```python
movies_recommendations('Tangled', genres_cosine_sim)
```

	original_title
34	Monsters University
57	WALL·E
120	Madagascar: Escape 2 Africa
137	Kung Fu Panda 2
146	Madagascar 3: Europe's Most Wanted
177	Turbo
194	Dinosaur
255	Home on the Range
328	Finding Nemo
430	Lilo & Stitch

Figure 21: Results of recommended movies based on genres

The results in figure 21 shows the recommended movies of a given title and based on their genres to output similar movies of a similar genre. The movie ‘Superman Returns’ is more action based therefore the system returned action based films to the user. Another example is the movie ‘Tangled’, it is a animation film, therefore the results returned other animation films similar to ‘Tangled’.

1.7.3 Recommended movies based on director

Next, I used the system to return movies based on their director. In the results for the movie 'Deadpool', the top movie recommended was 'Alice in Wonderland'. However, upon searching on the internet, I found that the director for both movies were different. Tim Muller directed the 'Deadpool' movie while Tim Burton directed for 'Alice in Wonderland'. The system probably found similarity with the first name of both director and resulted in the recommendation. However, the system was able to recommend similar director for the movie 'The Avengers', with the given of 'Serenity'. Both movies were directed by Joss Whedon.

```
movies_recommendations('Deadpool', director_cosine_sim)
```

	original_title
32	Alice in Wonderland
117	Charlie and the Chocolate Factory
127	Mad Max: Fury Road
133	Dark Shadows
173	Happy Feet Two
184	Home
187	Puss in Boots
215	4: Rise of the Silver Surfer
278	Planet of the Apes
385	Happy Feet

Figure 22: Results for Deadpool movie based on director

```
movies_recommendations('The Avengers', director_cosine_sim)
```

	original_title
16	The Avengers
1294	Serenity
0	Avatar
1	Pirates of the Caribbean: At World's End
2	Spectre
3	The Dark Knight Rises
4	John Carter
5	Spider-Man 3
6	Tangled
8	Harry Potter and the Half-Blood Prince

Figure 23: Results for The Avengers movie based on director

1.7.4 Recommendation based on production company

```
# Testing the system
movies_recommendations('Deadpool', pc_cosine_sim)
```

	original_title
64	X-Men: Apocalypse
3355	Devil's Due
91	Independence Day: Resurgence
3029	Kung Pow: Enter the Fist
1111	Victor Frankenstein
242	Fantastic Four
232	The Wolverine
766	Garfield: A Tail of Two Kitties
46	X-Men: Days of Future Past
282	True Lies

```
movies_recommendations('John Carter', pc_cosine_sim)
```

	original_title
255	Home on the Range
288	The Hunchback of Notre Dame
465	Fantasia 2000
646	The Kid
752	My Favorite Martian
1045	The Princess Diaries 2: Royal Engagement
1055	The Muppets
1206	The Odd Life of Timothy Green
1496	Snow Dogs
1595	Glory Road

Figure 24: Recommendation results based on production company

By using the system based on the production company, it was able to return results of movies from the same production company. 'Deadpool' and 'X-Men: Apocalypse' were both produced by Twentieth Century Fox. 'John Carter' and 'Home of the Range' were also both produced by Walt Disney Pictures. This shows that the system was able to accurately recommend similar production company.

1.7.5 Recommendation based on the selected feature

After seeing that the system is working, I combined the 3 features (genres, production company & director) to give result in a highly similar movie to the given. The results are as shown in figure 25 with most of the movies recommended for 'Deadpool' similar to the results based on production company.

```
# testing the system
movies_recommendations('Deadpool', combined_cosine_sim)
```

	original_title
64	X-Men: Apocalypse
242	Fantastic Four
3029	Kung Pow: Enter the Fist
91	Independence Day: Resurgence
232	The Wolverine
766	Garfield: A Tail of Two Kitties
46	X-Men: Days of Future Past
1918	Big Trouble in Little China
195	Night at the Museum: Secret of the Tomb
819	Date Night

```
[ ] movies_recommendations('Avatar', combined_cosine_sim)
```

	original_title
1652	Dragonball Evolution
587	The Abyss
282	True Lies
335	Rise of the Planet of the Apes
101	X-Men: First Class
292	Eragon
507	Independence Day
91	Independence Day: Resurgence
1201	Predators
243	Night at the Museum

Figure 25: Recommendation results

1.8 Conclusion

In conclusion, a movie recommendation system can be based on many factors. The system built for this project were the basics and to understand the fundamentals. There can be some improvement made on the system, where the recommendation can accurately get the director name. The important features that I have gathered that helps in a movie recommendation are the genres and director. It is most commonly seen in the streaming platform. Further addition can also include recommended movies based on the user ratings.

1.9 References

Us, A. J. W. (2024, June 17). *Movie theaters are being forced to evolve*. Theweek. https://theweek.com/culture-life/film/movie-theaters-dying-evolving

Anshari, M., & Alas, Y. (2015). Smartphones habits, necessities, and big data challenges. *The Journal of High Technology Management Research*, *26*(2), 177–185. https://doi.org/10.1016/j.hitech.2015.09.005

Skierkowski, D., & Wood, R. M. (2011). To text or not to text? The importance of text messaging among college-aged youth. *Computers in Human Behavior*, *28*(2), 744–756. https://doi.org/10.1016/j.chb.2011.11.023