IMPORTING LIBARRIES AND DATA

```
#load the necessary library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
#load the dataset
movie = pd.read_csv('tmdb_movie_data.csv')
movie.head()
```

	release_date	runtime	budget	revenue	vote_average	vote_count	tmdb_id	imdb_id	genres	
	2022-04-27	101	0	0	6.352	145	953300	tt19034332	Documentary	h
	2018-08-31	98	0	0	7.100	142	538002	tt6893836	Documentary	h
	1998-06-18	88	90000000	304320254	7.903	10132	10674	tt0120762	Animation, Family, Adventure	
	2019-07-24	162	95000000	392105159	7.426	14234	466272	tt7131622	Comedy, Drama, Thriller	h
	1996-01-02	100	0	0	6.000	1	334904	tt0274805	Comedy, Crime, Drama	h

```
movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4127 entries, 0 to 4126
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   title          4127 non-null   object
 1   tmdb_title     4127 non-null   object
 2   release_date   4100 non-null   object
 3   runtime        4127 non-null   int64
 4   budget         4127 non-null   int64
 5   revenue        4127 non-null   int64
 6   vote_average   4127 non-null   float64
 7   vote_count     4127 non-null   int64
 8   tmdb_id        4127 non-null   int64
 9   imdb_id        4028 non-null   object
 10  genres         4059 non-null   object
 11  tmdb_url       4127 non-null   object
 12  profit         4127 non-null   int64
 13  profit_margin  1989 non-null   float64
dtypes: float64(2), int64(6), object(6)
memory usage: 451.5+ KB
```

DATA PREPROCESSING

```
#only selecting necessary columns
movie_df = movie[['title', 'imdb_id', 'release_date', 'genres', 'runtime', 'budget', 'revenue', 'vote_average', 'vote_count']]
movie_df.head()
```

	title	imdb_id	release_date	genres	runtime	budget	revenue	vote_average	vote_count
0	The Mystery of Marilyn Monroe: The Unheard Tapes	tt19034332	2022-04-27	Documentary	101	0	0	6.352	145
1	They'll Love Me When I'm Dead	tt6893836	2018-08-31	Documentary	98	0	0	7.100	142
2	Mulan	tt0120762	1998-06-18	Animation, Family, Adventure	88	90000000	304320254	7.903	10132
3	Once Upon a Time... in Hollywood	tt7131622	2019-07-24	Comedy, Drama, Thriller	162	95000000	392105159	7.426	14234

```
#getting records that have a budget more than zero
movie_df = movie_df.loc[movie_df['budget'] > 0].copy()
```

```
movie_df.shape
```

```
(1508, 9)
```

```
movie_df.columns
```

```
Index(['title', 'imdb_id', 'release_date', 'genres', 'runtime', 'budget',
       'revenue', 'vote_average', 'vote_count'],
      dtype='object')
```

MERGING CREW RATINGS WITH MOVIE

```
crew_avg = pd.read_csv('movie_rating_avg.csv')
crew_avg.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

```
#before merging changing the name of primarytitle to title
crew_avg.rename(columns={'primaryTitle': 'title'}, inplace=True)

crew_avg.rename(columns={'tconst': 'imdb_id'}, inplace=True)
crew_avg.head()
```

	imdb_id	title	startYear	runtimeMinutes	genres	movie_rating	avg_cast_rating	director_rating	wr
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	

```
#merging only matching data records
movie_df = movie_df.merge(crew_avg, on='imdb_id', how='inner')
```

```
movie_df.tail()
```

	title_x	imdb_id	release_date	genres_x	runtime	budget	revenue	vote_average	vote_count	title_y	start
1368	Io Capitano	tt14225838	2023-09-07	Adventure, Drama	121	13272819	0	7.799	778	Io Capitano	2
1369	The Dunes	tt6910678	2021-09-30	Thriller	84	55000	0	5.600	5	The Dunes	2
1370	Fall	tt15325794	2022-08-11	Thriller	107	3000000	17363261	7.125	4294	Fall	2
1371	Glossary of Broken Dreams	tt7209510	2018-03-16	Documentary, Animation, Comedy, History	98	15000	0	6.800	11	Glossary of Broken Dreams	2
1372	The VelociPastor	tt1843303	2018-09-28	Action, Horror, Comedy	75	36000	0	5.200	222	The VelociPastor	2

```
movie_df.columns
```

```
Index(['title', 'imdb_id', 'release_date', 'runtime', 'budget', 'revenue',
       'tmdb_rating', 'vote_count', 'runtimeMinutes', 'imdb_rating',
       'avg_cast_rating', 'director_rating', 'writer_rating',
       'composer_rating', 'cinematographer_rating', 'editor_rating', 'genre'],
      dtype='object')
```

HANDLING COLUMNS

```
#dropping unnecessary columns
movie_df.drop(columns=['startYear',
       'title_y'], inplace=True)

#renaming the columns to match imdb and tmdb
movie_df.rename(columns={'vote_average': 'tmdb_rating', 'movie_rating': 'imdb_rating', 'title_x': 'title'}, inplace=True)
```

Handling 'genre_x' and 'genre_y' columns

```
def merge_genres(gx, gy):
    if pd.isna(gx): gx = ''
    if pd.isna(gy): gy = ''

    # Split by comma, remove spaces, lowercase for consistency
    genres = set([g.strip().title() for g in (gx + ',' + gy).split(',') if g.strip()])
    return ', '.join(sorted(genres))  # Optional sorting for consistency

movie_df['genre'] = movie_df.apply(lambda row: merge_genres(row['genres_x'], row['genres_y']), axis=1)

#dropping 'genres_x' and 'genres_y' columns
movie_df = movie_df.drop(columns=['genres_x', 'genres_y'])
```

Getting the average for the runtimes

So after searching online it seems that the runtimeMinutes is closer to the actual time than runtime. So we can either average or drop, but we are dropping the runtime for now