Download the dataset from: https://github.com/bellawillrise/Introduction-to-Numerical-Computing-in-Python/

Submit a pdf file, which is a rendered saved version of the jupyter notebook. Make sure to execute all the codes so the output can be viewed in the pdf.

Also include the link to the public github repository where the jupyter notebook for the assignment is uploaded.

Link to the github repository: https://github.com/NhilbertJayValente/CMSC_197.git

In [5]:
```
!git clone https://github.com/bellawillrise/Introduction-to-Numerical-Computing-in-Python.git
```

```
fatal: destination path 'Introduction-to-Numerical-Computing-in-Python' already exists and is not an empty dire
ctory.
```

In [6]:
```
cd Introduction-to-Numerical-Computing-in-Python/
```

```
C:\Users\ASUS\Downloads\Introduction-to-Numerical-Computing-in-Python
```

In [2]:
```
!pip install seaborn
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting seaborn
  Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
     ---------------------------------------- 294.9/294.9 kB 1.5 MB/s eta 0:00:00
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\asus\appdata\roaming\python\python310\site-p
ackages (from seaborn) (3.8.0)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in d:\programs\anaconda\lib\site-packages (from seaborn) (1
.26.0)
Requirement already satisfied: pandas>=1.2 in c:\users\asus\appdata\roaming\python\python310\site-packages (fro
m seaborn) (2.1.1)
Requirement already satisfied: pillow>=6.2.0 in d:\programs\anaconda\lib\site-packages (from matplotlib!=3.6.1,
>=3.4->seaborn) (9.4.0)
Requirement already satisfied: packaging>=20.0 in d:\programs\anaconda\lib\site-packages (from matplotlib!=3.6.
1,>=3.4->seaborn) (23.1)
Requirement already satisfied: python-dateutil>=2.7 in d:\programs\anaconda\lib\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (2.8.2)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\asus\appdata\roaming\python\python310\site-packages
(from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\asus\appdata\roaming\python\python310\site-package
s (from matplotlib!=3.6.1,>=3.4->seaborn) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asus\appdata\roaming\python\python310\site-package
s (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\asus\appdata\roaming\python\python310\site-packages
(from matplotlib!=3.6.1,>=3.4->seaborn) (1.1.1)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\appdata\roaming\python\python310\site-packages (fr
om matplotlib!=3.6.1,>=3.4->seaborn) (0.11.0)
Requirement already satisfied: pytz>=2020.1 in d:\programs\anaconda\lib\site-packages (from pandas>=1.2->seabor
n) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\asus\appdata\roaming\python\python310\site-packages (
from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in d:\programs\anaconda\lib\site-packages (from python-dateutil>=2.7->m
atplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Installing collected packages: seaborn
Successfully installed seaborn-0.13.2
```

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
# %matplotlib inline
```

In [7]:
```
data = pd.read_csv("data/movie_metadata_cleaned.csv")
```

In [8]:
```
data.head(2)
```

Out[8]:

	Unnamed: 0	movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_nar
0	0	b'Avatar'	Color	James Cameron	723.0	178.0	0.0	855.0	Joel Dav Moc
1	1	b"Pirates of the Caribbean: At World's End"	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloc

2 rows × 29 columns

In [16]:
```
list(data.columns)
```

Out[16]:
```
['Unnamed: 0',
 'movie_title',
 'color',
 'director_name',
 'num_critic_for_reviews',
 'duration',
 'director_facebook_likes',
 'actor_3_facebook_likes',
 'actor_2_name',
 'actor_1_facebook_likes',
 'gross',
 'genres',
 'actor_1_name',
 'num_voted_users',
 'cast_total_facebook_likes',
 'actor_3_name',
 'facenumber_in_poster',
 'plot_keywords',
 'movie_imdb_link',
 'num_user_for_reviews',
 'language',
 'country',
 'content_rating',
 'budget',
 'title_year',
 'actor_2_facebook_likes',
 'imdb_score',
 'aspect_ratio',
 'movie_facebook_likes']
```

Get the top 10 directors with most movies directed and use a boxplot for their gross earnings

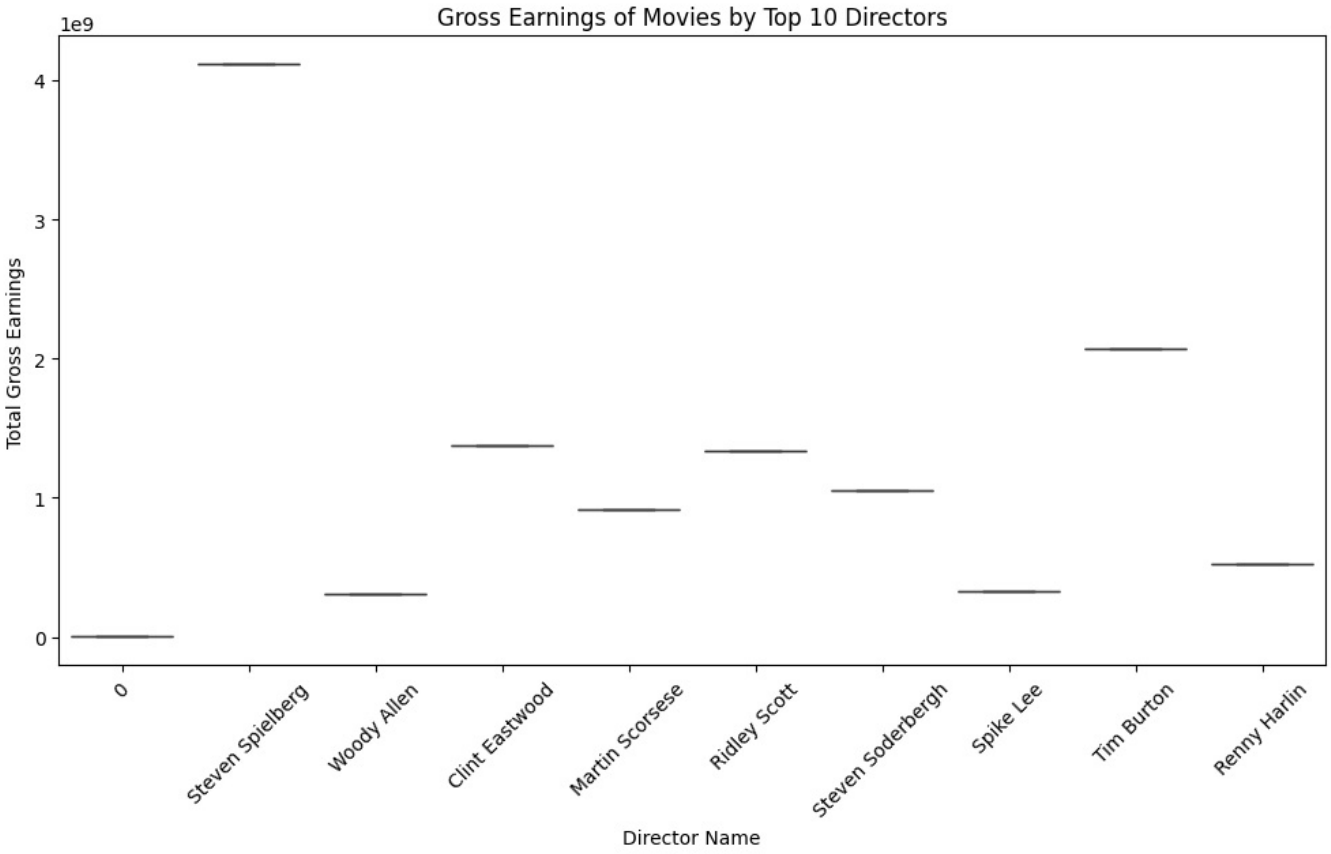
In [83]:
```
# Used the agg command to perform multiple aggregation operations
movie_counts_per_director = data.groupby('director_name').agg(
    movie_count=('movie_title','count'),
    total_gross=('gross', 'sum')
).reset_index()
```

In [22]:
```
top_10_directors = movie_counts_per_director.sort_values(by="movie_count", ascending=False)
top_10_directors[:10]
```

Out[22]:

	director_name	movie_count	total_gross
0	0	104	1.039304e+06
2159	Steven Spielberg	26	4.114233e+09
2378	Woody Allen	22	3.083454e+08
392	Clint Eastwood	20	1.378321e+09
1478	Martin Scorsese	20	9.202871e+08
1903	Ridley Scott	17	1.337772e+09
2158	Steven Soderbergh	16	1.050730e+09
2102	Spike Lee	16	3.285004e+08
2221	Tim Burton	16	2.071275e+09
1862	Renny Harlin	15	5.239759e+08

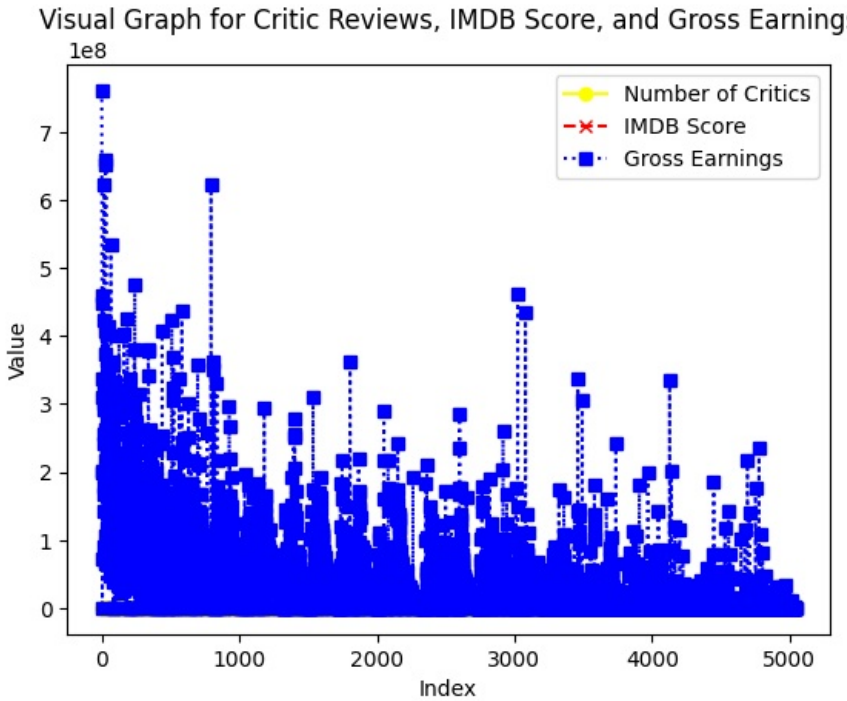
In [49]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='director_name', y='total_gross', data=top_10_directors[:10])
plt.xticks(rotation=45)
plt.title('Gross Earnings of Movies by Top 10 Directors')
plt.xlabel('Director Name')
plt.ylabel('Total Gross Earnings')
plt.show()
```



Plot the following variables in one graph:

- num_critic_for_reviews
- IMDB score
- gross

In [82]:
```
# Set o, x, and s as data markers
plt.plot(data['num_critic_for_reviews'], label='Number of Critics', color='yellow', linestyle='-', marker='o')
plt.plot(data['imdb_score'], label='IMDB Score', color='red', linestyle='--', marker='x')
plt.plot(data['gross'], label='Gross Earnings', color='blue', linestyle=':', marker='s')

plt.title('Visual Graph for Critic Reviews, IMDB Score, and Gross Earnings')
plt.xlabel('Index')
plt.ylabel('Value')

plt.legend()
plt.show()
```



Compute Sales (Gross - Budget), add it as another column

Compute Sales (Gross - Budget), add it as another column

In [36]:
```
data = data.dropna(subset=['gross', 'budget'])
```

In [43]:
```
data['sales'] = data['gross'] - data['budget']
data[['gross', 'budget', 'sales']].head(10)
```

Out[43]:

	gross	budget	sales
0	760505847.0	237000000.0	523505847.0
1	309404152.0	300000000.0	9404152.0
2	200074175.0	245000000.0	-44925825.0
3	448130642.0	250000000.0	198130642.0
4	0.0	0.0	0.0
5	73058679.0	263700000.0	-190641321.0
6	336530303.0	258000000.0	78530303.0
7	200807262.0	260000000.0	-59192738.0
8	458991599.0	250000000.0	208991599.0
9	301956980.0	250000000.0	51956980.0

Which directors garnered the most total sales?

In [45]:
```
director_sales = data.groupby('director_name')['sales'].sum().reset_index()
```

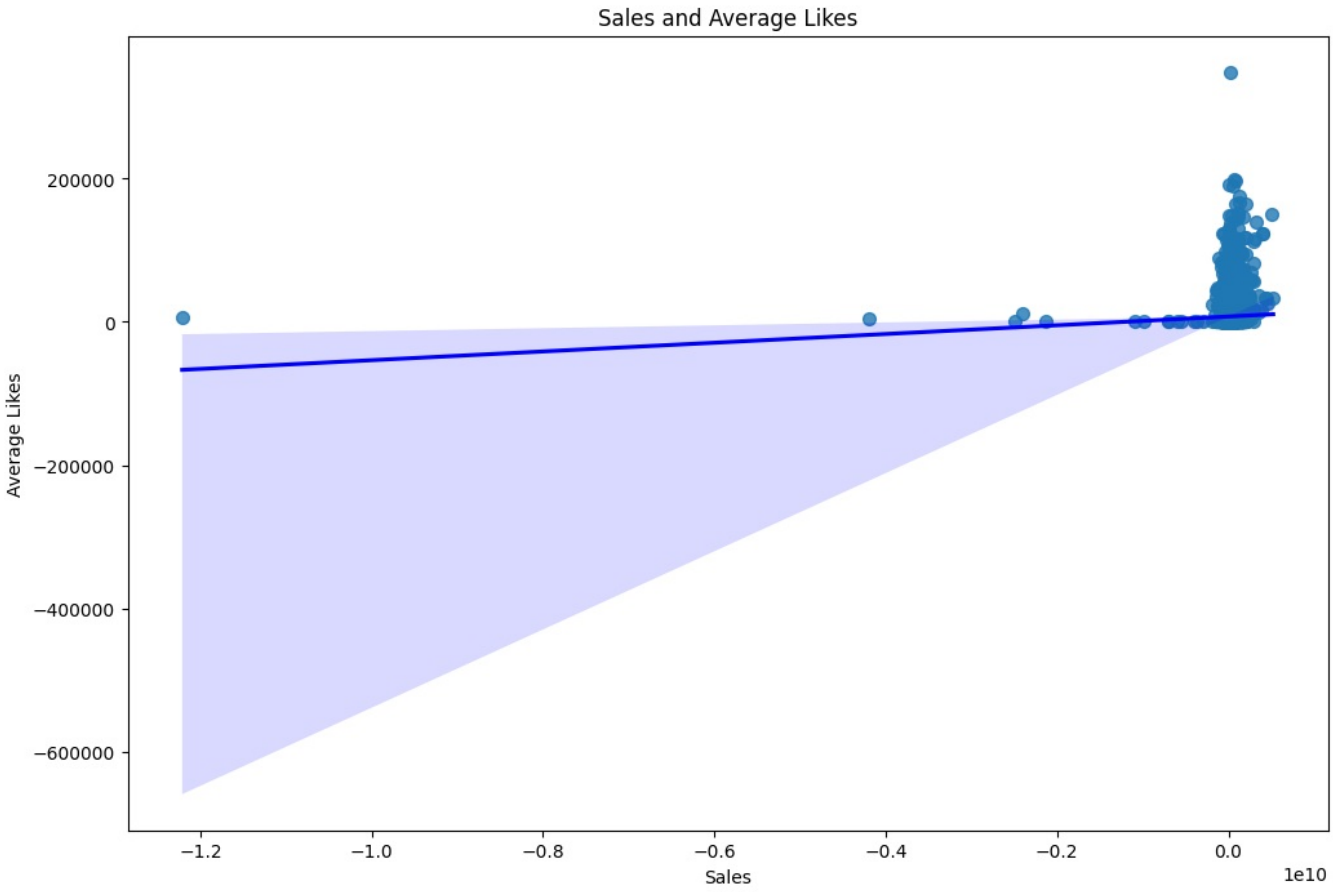
In [48]:
```
top_directors_sales = director_sales.sort_values(by='sales', ascending=False)
top_directors_sales[:10]
```

Out[48]:

	director_name	sales
2159	Steven Spielberg	2.451332e+09
765	George Lucas	1.386641e+09
923	James Cameron	1.199626e+09
1219	Joss Whedon	1.000887e+09
335	Chris Columbus	9.417076e+08
1787	Peter Jackson	9.009693e+08
2221	Tim Burton	8.242755e+08
374	Christopher Nolan	8.082276e+08
1158	Jon Favreau	7.693815e+08
695	Francis Lawrence	7.555020e+08

Plot sales and average likes as a scatterplot. Fit it with a line.

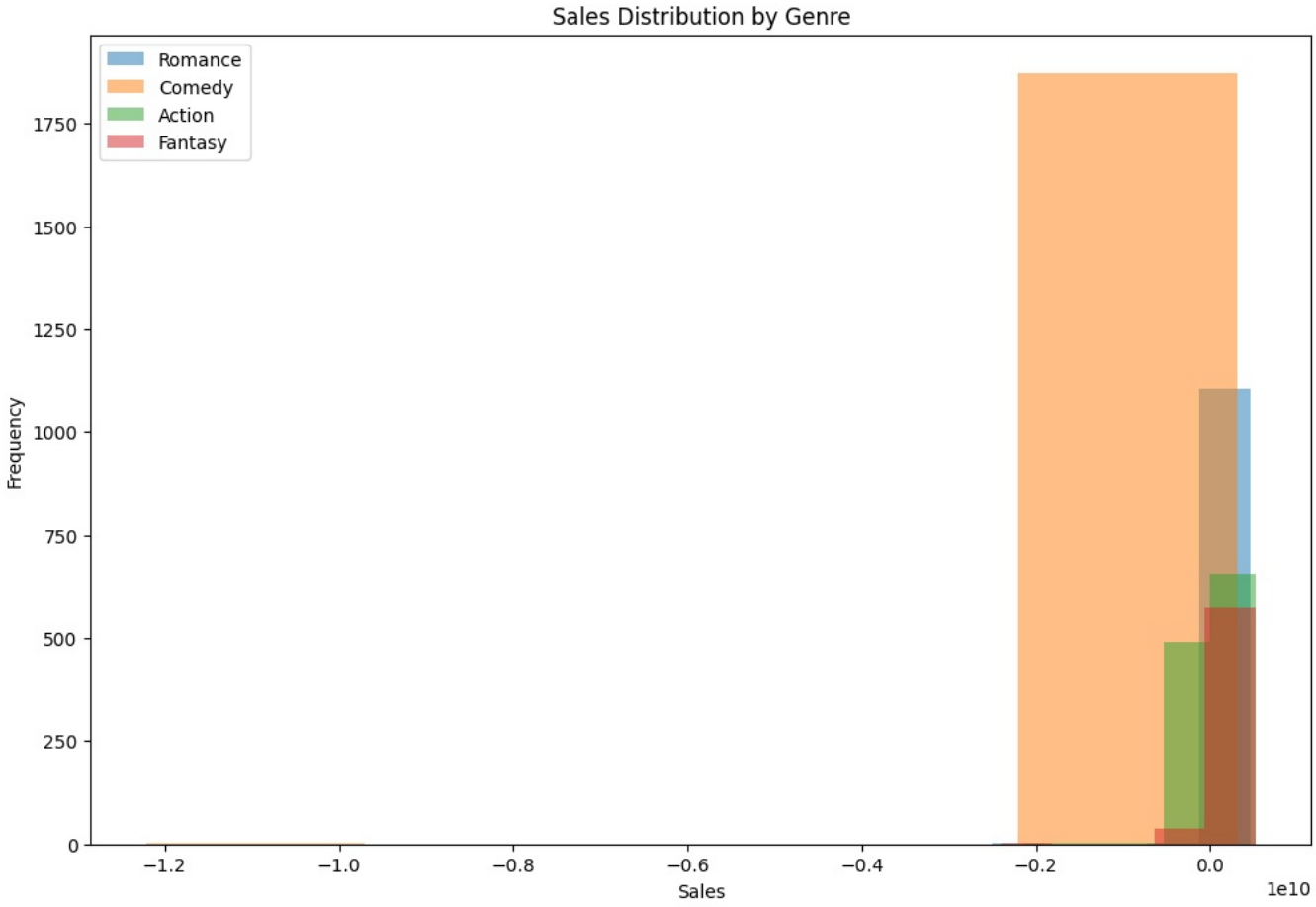
In [81]:
```
plt.figure(figsize=(12, 8))
# Set the size of the scatter plots to 50 and color of the regression line to blue
sns.regplot(x='sales', y='movie_facebook_likes', data=data, scatter_kws={'s':50}, line_kws={'color':'blue'})
plt.title('Sales and Average Likes')
plt.xlabel('Sales')
plt.ylabel('Average Likes')
plt.show()
```



Which of these genres are the most profitable? Plot their sales using different histograms, superimposed in the same axis.

- Romance
- Comedy
- Action
- Fantasy

In [80]:
```
genres = ['Romance', 'Comedy', 'Action', 'Fantasy']

plt.figure(figsize=(12, 8))

for genre in genres:
#     Genre is not case sensitive and rows with missing values is not included in the results
    genre_sales = data[data['genres'].str.contains(genre, case=False, na=False)]['sales']

#     Created 5 bins for the data and set the transparency at 50%
    plt.hist(genre_sales, bins=5, alpha=0.5, label=genre)

plt.title('Sales Distribution by Genre')
plt.xlabel('Sales')
plt.ylabel('Frequency')

plt.legend()
plt.show()
```



For each of movie, compute average likes of the three actors and store it as a new variable

Read up on the mean function.

Store it as a new column, average_actor_likes.

In [78]:
```
# Taking the average likes of each actor for each row
data['average_actor_likes'] = data[['actor_1_facebook_likes', 'actor_2_facebook_likes', 'actor_3_facebook_likes
data['average_actor_likes']
```

Out[78]:
```
0         930.333333
1       15333.333333
2        3851.333333
3       24333.333333
4          47.666667
             ...
5039      584.333333
5040        0.000000
5041      718.000000
5042       41.666667
5043        0.000000
Name: average_actor_likes, Length: 5044, dtype: float64
```

Copying the whole dataframe

In [73]:
```
df = data.copy()
df.head()
```

Out[73]:

	Unnamed: 0	movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_nar
0	0	b'Avatar'	Color	James Cameron	723.0	178.0	0.0	855.0	Joel Dav Moc
1	1	b"Pirates of the Caribbean: At World's End"	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloc
2	2	b'Spectre'	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinne
3	3	b'The Dark Knight Rises'	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Ba
4	4	b'Star Wars: Episode VII - The Force Awakens ...	0	Doug Walker	0.0	0.0	131.0	0.0	Rob Walk

5 rows × 31 columns

Min-Max Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.

The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range. We can apply the min-max scaling in Pandas using the .min() and .max() methods.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Normalize each numeric column (those that have types integer or float) of the copied dataframe (df)

In [76]:
```
# Selecting columns that have integer or floats as their data types
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
```

In [77]:
```
# Normalizing by using the min-max approach
df[numeric_columns] = df[numeric_columns].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
df.head()
```

Out[77]:

	Unnamed: 0	movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_nar
0	0.000000	b'Avatar'	Color	James Cameron	0.889299	0.941799	0.000000	0.037174	Joel Da Moc
1	0.000198	b"Pirates of the Caribbean: At World's End"	Color	Gore Verbinski	0.371464	0.894180	0.024478	0.043478	Orlando Blo
2	0.000397	b'Spectre'	Color	Sam Mendes	0.740467	0.783069	0.000000	0.007000	Rory Kinn
3	0.000595	b'The Dark Knight Rises'	Color	Christopher Nolan	1.000000	0.867725	0.956522	1.000000	Christian B
4	0.000793	b'Star Wars: Episode VII - The Force Awakens ...	0	Doug Walker	0.000000	0.000000	0.005696	0.000000	Rob Wal

5 rows × 31 columns

In [ ]: