Data Mining Project

Members Names:

Name	ID
1- زياد محمد الخطيب	22010100
2- قمر مسلم برازي	22011438
3- عبد الرحمن احمد محمد	22011620
4- تبيان أشرف عبدالله	22011497
5- احمد عماد عبد الفتاح	22010027
6- محمد محمود نعيم	22010231

1) Dataset

- **Link:** https://www.kaggle.com/datasets/prishasawhney/imdb-dataset-top-2000-movies

- **Description:** **This dataset contains a list of the top 2000 movies as rated by users on IMDb (Internet Movie Database). Each entry in the dataset represents a movie and includes key information such as the title, year of release, genre, runtime, IMDb rating, and number of votes.**

- **The dataset contains 10 columns:**

  1. *The name of the movie*
  2. *The Year of release*
  3. *The running time of the movie (in minutes)*
  4. *It's IMDB Rating*
  5. *It's metascore*
  6. *The number of votes it got*
  7. *The Genre of the movie*
  8. *The director of the movie*
  9. *The cast of the movie*
  10. *Gross value*

- **objectives: perform some clustering algorithms (K-medoid, hierarchical) and find relations between columns and each other based on gross value.**

---

2) Code:

- **Import libraries & dataset.**

Import needed libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import datetime as dt
from wordcloud import WordCloud
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, cut_tree, fcluster
from sklearn.cluster import AgglomerativeClustering
from sklearn_extra.cluster import KMedoids
```

[2] ✓ 0.0s Python

read data set

```
df = pd.read_csv('imdb.csv')
```

[3] ✓ 0.0s Python

- pandas: for dataframes
- numbpy: for numerical problems (arrays , multidim array)
- matplot: visualization
- seaborn: visualization
- plotly.express: interactive visualization
- plotly.graph_objects: interactive visualization
- datetime: deal with time and date
- wordcloud: visualization of text data e.g. (frequency of each word)
- sklearn: clustering algorithms (K-medoid , hierarchical)

- ## Exploration

Display dataframe

```
# display the first 5 rows in our data
df.head()
# if we want to diplay all rows in the data set we can use to_string() function
# but this is not recommended in large data sets
```

[4] ✓ 0.0s

	Movie Name	Release Year	Duration	IMDB Rating	Metascore	Votes	Genre	Director	Cast	Gross
0	The Godfather	1972	175	9.2	100.0	2,002,655	Crime, Drama	Francis Ford Coppola	Marlon Brando	$134.97M
1	The Godfather Part II	1974	202	9.0	90.0	1,358,608	Crime, Drama	Francis Ford Coppola	Al Pacino	$57.30M
2	Ordinary People	1980	124	7.7	86.0	56,476	Drama	Robert Redford	Donald Sutherland	$54.80M
3	Lawrence of Arabia	1962	218	8.3	100.0	313,044	Adventure, Biography, Drama	David Lean	Peter O'Toole	$44.82M
4	Straw Dogs	1971	113	7.4	73.0	64,331	Crime, Drama, Thriller	Sam Peckinpah	Dustin Hoffman	NaN

Theoretical info.

```
# info function to display information about columns in the data set
df.info()
```

[5] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Movie Name    2000 non-null   object
 1   Release Year  2000 non-null   object
 2   Duration      2000 non-null   int64
 3   IMDB Rating   2000 non-null   float64
 4   Metascore     1919 non-null   float64
 5   Votes         2000 non-null   object
 6   Genre         2000 non-null   object
 7   Director      2000 non-null   object
 8   Cast          2000 non-null   object
 9   Gross         1903 non-null   object
dtypes: float64(2), int64(1), object(7)
memory usage: 156.4+ KB
```

❖ Statistical info. (just for numeric columns)

```python
# display Descriptive statistics of the data set using describe function
# like 5 number summary, mean and std
df.describe()
```

[6] ✓ 0.0s

	Duration	IMDB Rating	Metascore
count	2000.000000	2000.000000	1919.000000
mean	113.939000	6.922600	61.044294
std	22.946035	0.955618	17.937722
min	50.000000	1.500000	9.000000
25%	98.000000	6.400000	48.000000
50%	110.000000	7.000000	61.000000
75%	125.000000	7.600000	74.000000
max	271.000000	9.300000	100.000000

- **Exploration**

❖ Check for null values.

Before

```python
# Check the existance of null values in our data set
df.isnull().sum()
# As we can see below the ['Metascore', 'Gross'] columns have null values
```

[4] ✓ 0.0s

```
Movie Name      0
Release Year    0
Duration        0
IMDB Rating     0
Metascore      81
Votes           0
Genre           0
Director        0
Cast            0
Gross          97
dtype: int64
```

After drop

```python
# Remove null values
# (inplace = True) parameter to save the changes in the df data frame
df.dropna(inplace = True)
```

[5] ✓ 0.0s

```python
# Data frame without na values
df.isnull().sum()
```

[6] ✓ 0.0s

```
Movie Name      0
Release Year    0
Duration        0
IMDB Rating     0
Metascore       0
Votes           0
Genre           0
Director        0
Cast            0
Gross           0
dtype: int64
```

❖ Check for duplicated values.

```python
# Now let us check the existance of duplicated rows
df.duplicated().sum()
# There is no duplicated rows because the sum of them is 0
```

[7] ✓ 0.0s

```
0
```

No Duplicate values

❖ Remove non-digits from 'Release Year' Column and convert to numeric

```python
# Remove non-digits from the 'Realese Year' column
df['Release Year'] = df['Release Year'].str.replace(r'\D+', '', regex= True)

# Convert the data type of the 'Realese Year' from Object type to int
df['Release Year'] = df['Release Year'].astype('int')
```

[8] ✓ 0.0s

```python
# Now the data type of 'Realese Year' column become int
df.info()
```

[9] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 1870 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Movie Name    1870 non-null   object
 1   Release Year  1870 non-null   int32
 2   Duration      1870 non-null   int64
 3   IMDB Rating   1870 non-null   float64
 4   Metascore     1870 non-null   float64
 5   Votes         1870 non-null   object
 6   Genre         1870 non-null   object
 7   Director      1870 non-null   object
 8   Cast          1870 non-null   object
 9   Gross         1870 non-null   object
dtypes: float64(2), int32(1), int64(1), object(6)
memory usage: 153.4+ KB
```

❖ Change ['Gross' & 'votes'] data types from object to numeric.

```python
# we need to convert the type of some columns to be able to deal with them
def to_numeric(x):
    valid = "0123456789."
    x = "".join(["" if i not in valid else i for i in x]) #List comprehension
    return float(x) if x else None

# if value of x in (valid) we don't make any changes but if not in we replace it with "" (remove it)
# and convert this string to float after filtering

df["Gross"] = df["Gross"].apply(to_numeric)
df["Votes"] = df["Votes"].apply(to_numeric)
df.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 1870 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Movie Name    1870 non-null   object
 1   Release Year  1870 non-null   int32
 2   Duration      1870 non-null   int64
 3   IMDB Rating   1870 non-null   float64
 4   Metascore     1870 non-null   float64
 5   Votes         1870 non-null   float64
 6   Genre         1870 non-null   object
 7   Director      1870 non-null   object
 8   Cast          1870 non-null   object
 9   Gross         1870 non-null   float64
dtypes: float64(4), int32(1), int64(1), object(4)
memory usage: 153.4+ KB
```

❖ Clean outliers (unnormal values) in 'Gross' column by IQR method

1

```python
# first quartile & third quartile
q1 = df["Gross"].quantile(0.25)
q3 = df["Gross"].quantile(0.75)
```

[16]

```python
iqr = q3 - q1
print("Q1:", q1)
print("Q3:", q3)
print("IQR:", iqr)
```

[17]

```
Q1: 18.615000000000002
Q3: 88.44500000000001
IQR: 69.83000000000001
```

```python
# calculate upper & lower limits to specifiy the outliers
upper_limit = q3 + (1.5 * iqr)
lower_limit = q1 - (1.5 * iqr)
print("Upper limit:", upper_limit)
print("Lower limit:", lower_limit)
```

[18]

```
Upper limit: 193.19000000000003
Lower limit: -86.13000000000002
```

2

```python
# Any value greater than the upper_limit or less than the lower limit is outlier
df.loc[(df["Gross"] > upper_limit) | (df["Gross"] < lower_limit)]
# From the data below we can notice that we have 111 rows with outliers
```

[19]

	Movie Name	Release Year	Duration	IMDB Rating	Metascore	Votes	Genre	Director	Cast	Gross
32	Star Wars	1977	121	8.6	90.0	1444539.0	Action, Adventure, Fantasy	George Lucas	Mark Hamill	322.74
33	Star Wars: Episode V - The Empire Strikes Back	1980	124	8.7	82.0	1374400.0	Action, Adventure, Fantasy	Irvin Kershner	Mark Hamill	290.48
50	Jaws	1975	124	8.1	87.0	656761.0	Adventure, Mystery, Thriller	Steven Spielberg	Roy Scheider	260.00
52	The Exorcist	1973	122	8.1	83.0	452972.0	Horror	William Friedkin	Ellen Burstyn	232.91
73	Raiders of the Lost Ark	1981	115	8.4	86.0	1033883.0	Action, Adventure	Steven Spielberg	Harrison Ford	248.16
...	...	...	...	...	...	...	...	...	...	...
1854	Alice in Wonderland	2010	108	6.4	53.0	439658.0	Adventure, Family, Fantasy	Tim Burton	Mia Wasikowska	334.19
1870	The Twilight Saga: New Moon	2009	130	4.8	44.0	299214.0	Adventure, Drama, Fantasy	Chris Weitz	Kristen Stewart	296.62
1871	The Twilight Saga: Eclipse	2010	124	5.1	58.0	260006.0	Action, Adventure, Drama	David Slade	Kristen Stewart	300.53
1963	Monsters vs. Aliens	2009	94	6.4	56.0	175456.0	Animation, Action, Adventure	Rob Letterman	Conrad Vernon	198.35
1975	Alvin and the Chipmunks: The Squeakquel	2009	88	4.5	41.0	54236.0	Adventure, Comedy, Family	Betty Thomas	Jason Lee	219.61

111 rows × 10 columns

```python
# define new dataframe without the outliers
# Any value less than the upper_limit and greater than the lower limit is not outlier
new_df = df.loc[(df["Gross"] < upper_limit) & (df["Gross"] > lower_limit)]
print("before removing outliers:", len(df))
print("after removing outliers:", len(new_df))
print("Number of outliers:", len(df) - len(new_df))
```

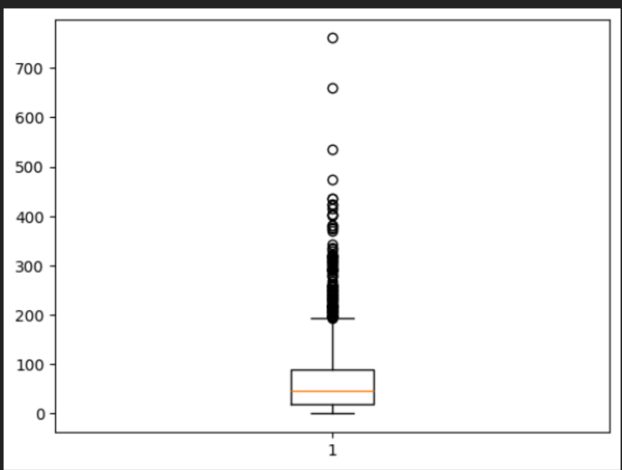
[20]

```
before removing outliers: 1870
after removing outliers: 1759
Number of outliers: 111
```

Before

```python
plt.boxplot(df["Gross"])
plt.show()
```

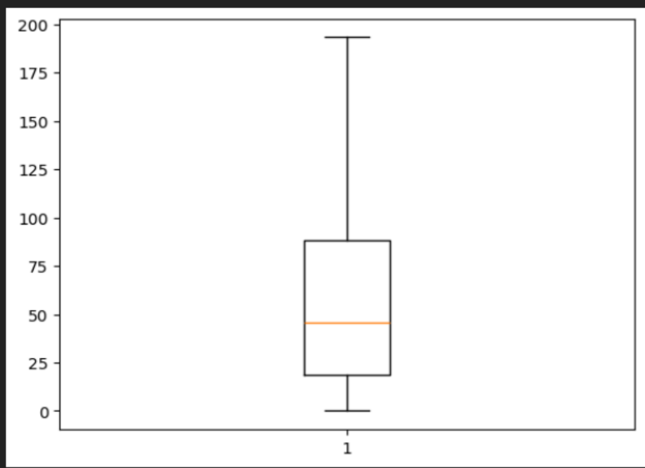
[14] ✓ 0.9s



After

```python
plt.boxplot(new_df["Gross"])
plt.show()
```

[21] ✓ 0.2s



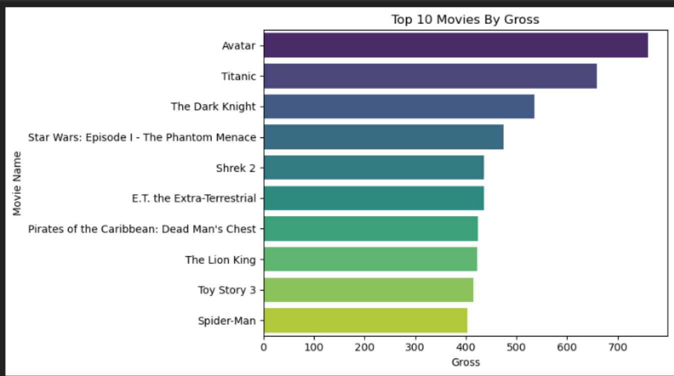
- ## Visualizations

❖ Finding top 10 movies for all time according to the Gross

```python
# This function show the top 10 movies acording to a parameter named column
# If we don't send a data frame as a second parameter the default is df
def top_10_movies(column, df = df):
    data = df.sort_values(by = column, ascending = False, ignore_index = True)
    data = data[["Movie Name", column]].head(10)
    plt.figure(figsize=(9, 5))
    sns.barplot(data = data, x = column, y = "Movie Name", palette = "viridis")
    plt.title(f"Top 10 Movies By {column}")
    plt.tight_layout()
    plt.show()
```
[23]

```python
# This function show the top 10 categories acording to a parameter named column
# Like the method above if we don't send a data frame as a second parameter the default is df
def top_10_by_category(column, df = df):
    data = df.groupby(column)["Gross"].sum()
    data = data.nlargest(10)
    plt.figure(figsize=(9, 5))
    sns.barplot(x = data.values, y = data.index, palette="viridis")
    plt.title(f"Top 10 {column} By Gross")
    plt.xlabel("Total Gross")
    plt.ylabel(column)
    plt.tight_layout()
    plt.show()
```
[24]

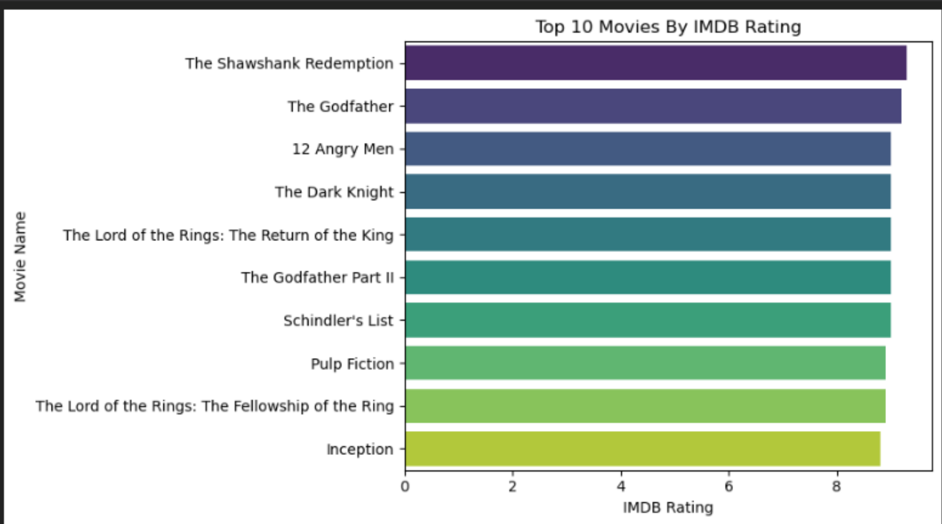
```python
top_ten_all_time = df.groupby(df["Movie Name"])["Gross"].sum().sort_values(ascending=False)
top_10_movies("Gross")
top_ten_all_time.head(10)
```
[27]




```
Movie Name
Avatar                                         760.51
Titanic                                        659.33
The Dark Knight                                534.86
Star Wars: Episode I - The Phantom Menace      474.54
Shrek 2                                        436.47
E.T. the Extra-Terrestrial                     435.11
Pirates of the Caribbean: Dead Man's Chest     423.32
The Lion King                                  422.78
Toy Story 3                                    415.00
Spider-Man                                     403.71
Name: Gross, dtype: float64
```

❖ Finding top 10 movies for all time according to the Rate

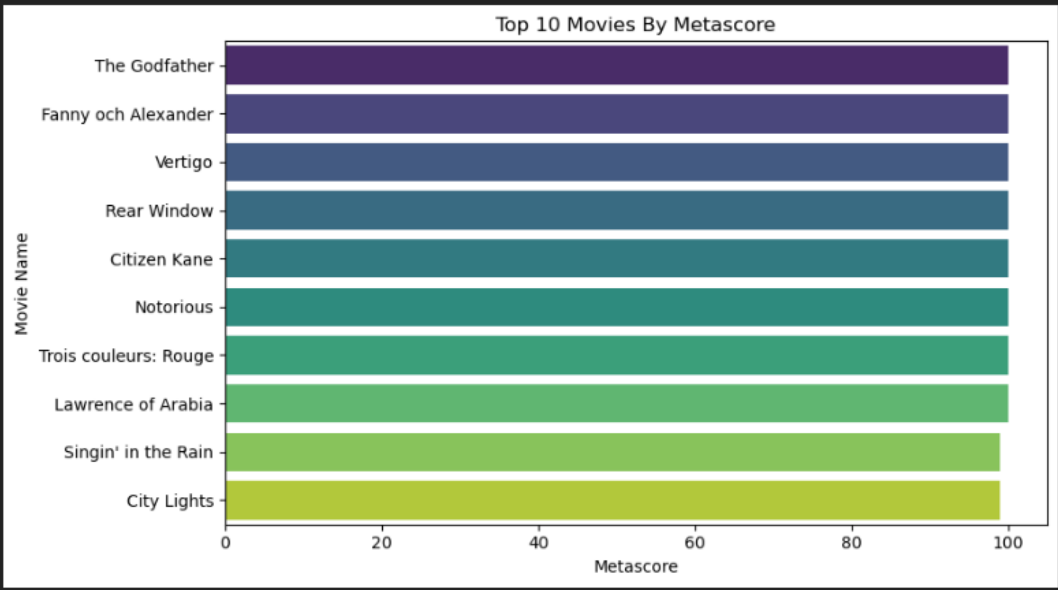
```python
top_10_movies("IMDB Rating")
```
[28]



❖ Finding top 10 movies for all time according to the Metascore

```
top_10_movies("Metascore")
```

[29]

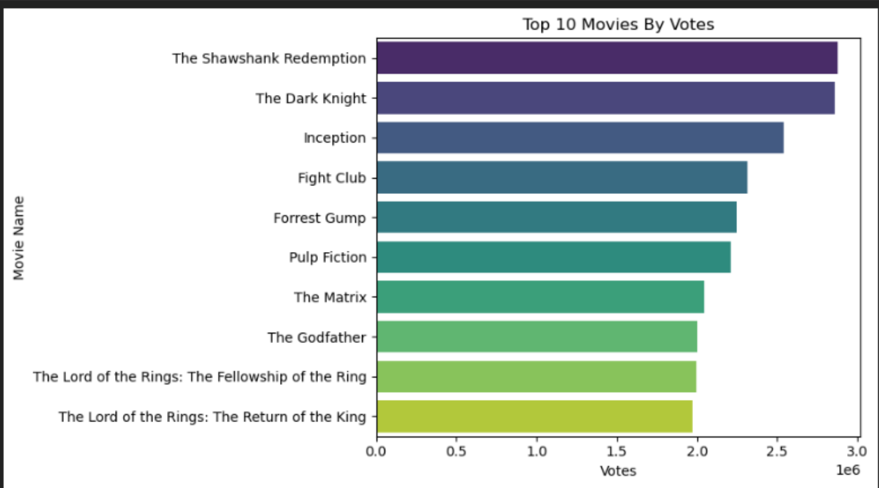


❖ Finding top 10 movies for all time according to the Votes

Finding top 10 movies for all time acording to the Votes

```
top_ten_Votes = df.groupby(df["Movie Name"])["Votes"].sum().sort_values(ascending=False)
top_10_movies("Votes")
top_ten_Votes.head(10)
```

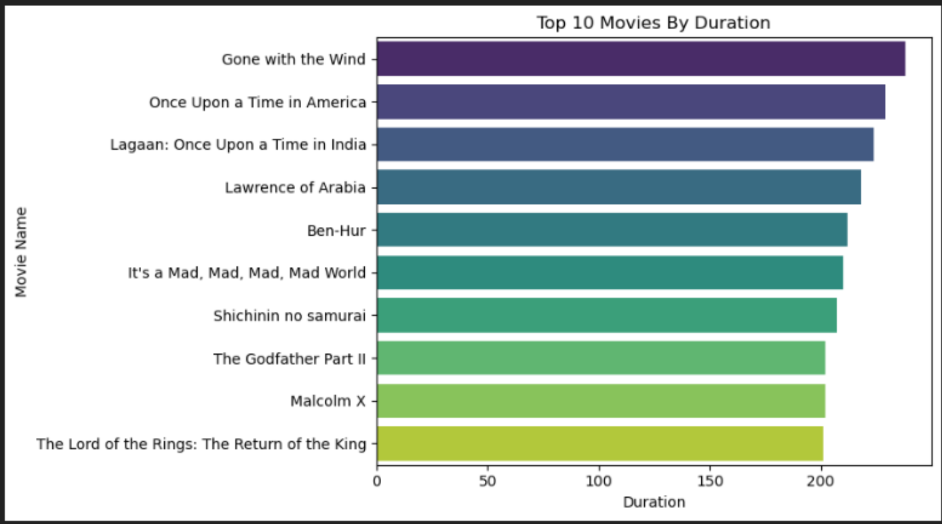
[30]



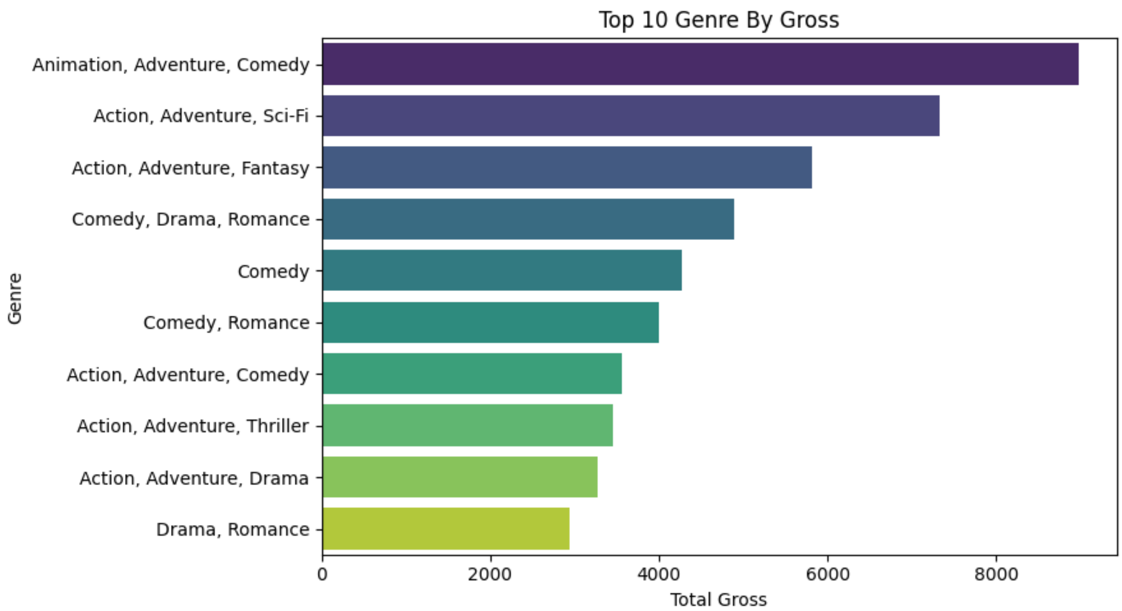

```
Movie Name
The Shawshank Redemption                          2875249.0
The Dark Knight                                   2857781.0
Inception                                         2538581.0
Fight Club                                        2311174.0
Forrest Gump                                      2245598.0
Pulp Fiction                                      2209021.0
The Matrix                                        2043258.0
The Godfather                                     2002655.0
The Lord of the Rings: The Fellowship of the Ring 1998243.0
The Lord of the Rings: The Return of the King     1970311.0
Name: Votes, dtype: float64
```

❖ Finding top 10 longest movies

```
top_10_movies("Duration")
```

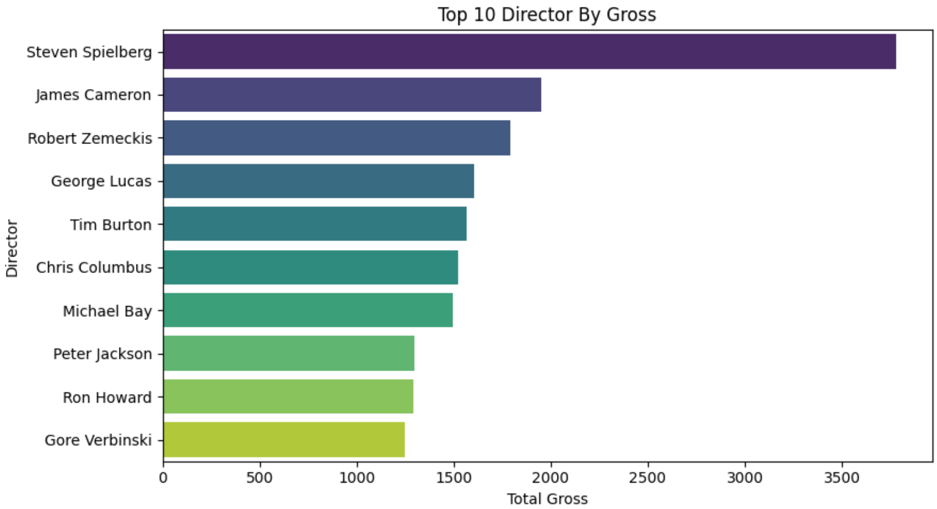


❖ Finding top 10 genre for all time according to the gross

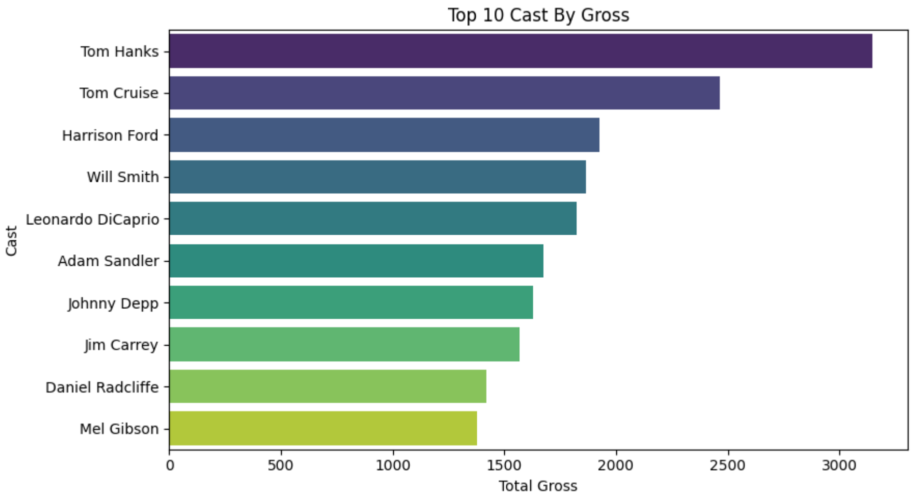


→ we notice from this figure that genres [animation – adv – comedy] are the most grossable.

❖ Finding top 10 director for all time according to the gross



❖ Finding top 10 cast for all time according to the gross

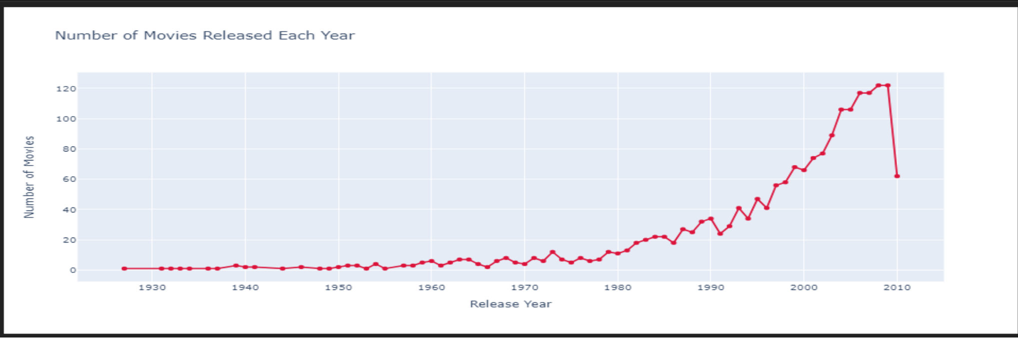


❖ Calculate number of movies released each year. (interactive)

Calculate number of movies released each year

```python
movies_per_year = df['Release Year'].value_counts().sort_index().reset_index()
movies_per_year.columns = ['Release Year', 'Number of Movies']

# Create an interactive line plot with Plotly
fig = px.line(movies_per_year, x='Release Year', y='Number of Movies', title='Number of Movies Released Each Year',
              labels={'Number of Movies': 'Number of Movies', 'Release Year': 'Release Year'},
              hover_data={'Number of Movies': True, 'Release Year': True},
              line_shape='linear',
              render_mode='svg',
              markers=True,
              color_discrete_sequence=['crimson'],
              width=1100, height=500)

# Show the plot
fig.show()

# From the following graph we can conclude that:-
# The number of movies has increased recently, especially in the year 2003 to 2009
# Back to a combination of industry growth, technological advancements, changing consumer preferences..
```

[33]



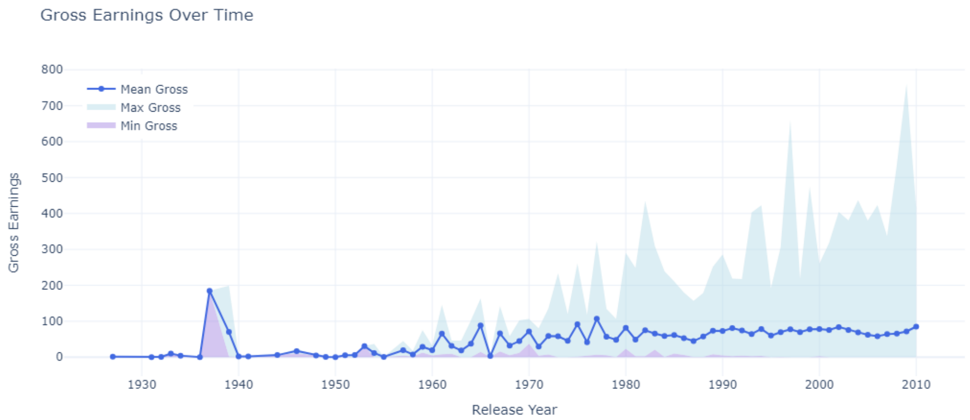
The number of movies has increased recently, especially in the year 2003 to 2009

Back to a combination of industry growth, technological advancements, changing consumer preferences...

❖ Calculate gross earnings statistics per year. (interactive)

Calculate gross earnings statistics per year

```python
gross_yearly = df.groupby('Release Year')['Gross'].agg(['min', 'max', 'mean']).reset_index()

# Create figure
fig = go.Figure()

# Add shaded area between min and max earnings
fig.add_trace(go.Scatter(x=gross_yearly['Release Year'], y=gross_yearly['min'],
                         fill='tonexty', mode='none', fillcolor='rgba(147, 112, 219, 0.4)',
                         line=dict(color='darkmagenta'),
                         name='Min Gross'))

fig.add_trace(go.Scatter(x=gross_yearly['Release Year'], y=gross_yearly['max'],
                         fill='tonexty', mode='none', fillcolor='rgba(173, 216, 230, 0.43)',
                         line=dict(color='lightblue'),
                         name='Max Gross'))

# Add line plot for mean earnings
fig.add_trace(go.Scatter(x=gross_yearly['Release Year'], y=gross_yearly['mean'],
                         mode='lines+markers', name='Mean Gross',
                         line=dict(color='royalblue', width=2)))

# Update layout
fig.update_layout(title='Gross Earnings Over Time',
                  xaxis_title='Release Year',
                  yaxis_title='Gross Earnings',
                  hovermode='x unified',
                  plot_bgcolor='rgba(0,0,0,0)',
                  legend=dict(x=0.02, y=0.98),
                  template='plotly_white',
                  width=1100, height=500)

# Show plot
fig.show()

# As we can see in the graph below:-
# 1937, the average (mean) earnings from movies were higher compared to other years
# Movies were making more and more money each year
```
[34]



As we can see in the graph below: -

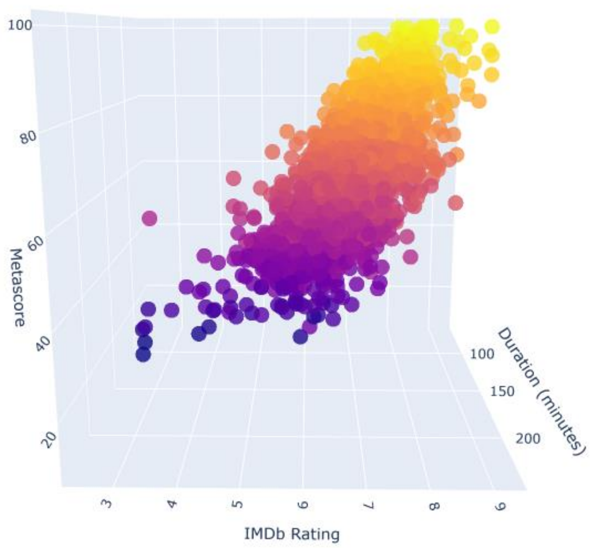
1937, the average (mean) earnings from movies were higher compared to other years

Movies were making more and more money each year

❖ How movies with different durations can still achieve critical and audience praise. (3d-interactive scatterplot)

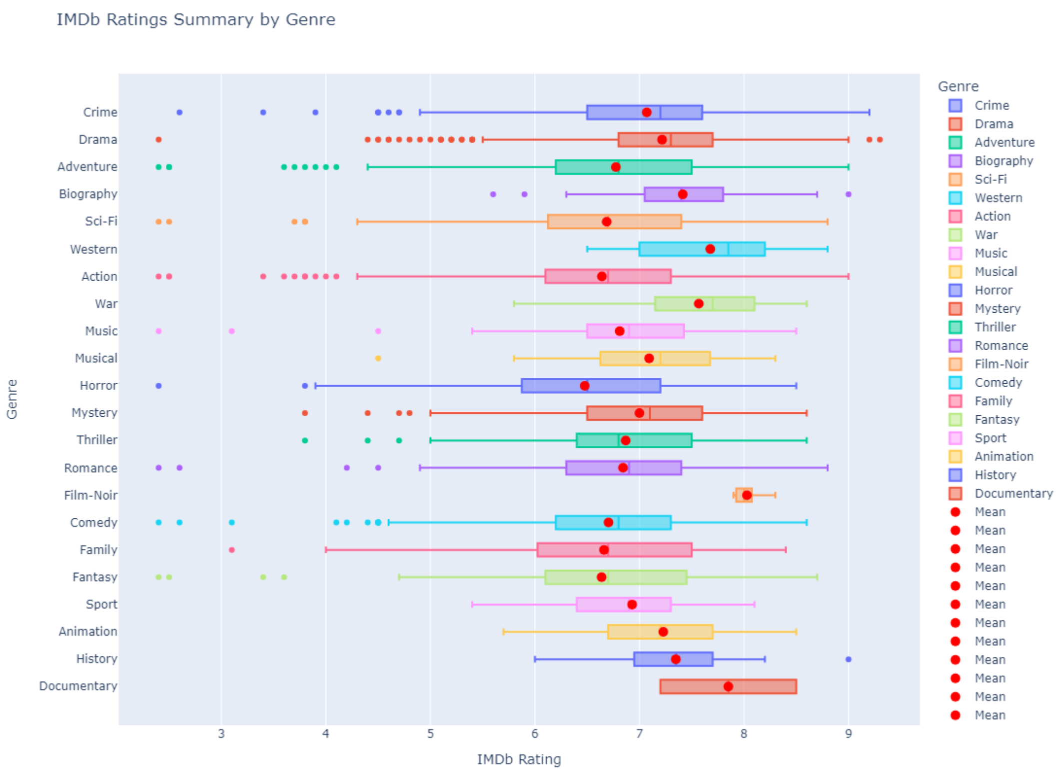
```python
hover_text = [f'Duration: {d} min<br>Metascore: {m}<br>IMDb Rating: {r}'
              for d, m, r in zip(df['Duration'], df['Metascore'], df['IMDB Rating'])]

# Create 3D scatter plot
fig = go.Figure(data=[go.Scatter3d(
    x=df['Duration'],
    y=df['IMDB Rating'],
    z=df['Metascore'],
    mode='markers',
    marker=dict(
        size=8,
        color=df['Metascore'],  # Color points based on Metascore
        colorscale='Plasma',  # Specify color scale
        opacity=0.8
    ),
    hovertext=hover_text,  # Set hover text
    hoverinfo='text'  # Show hover text
)])

# Update layout
fig.update_layout(scene=dict(
    xaxis_title='Duration (minutes)',
    yaxis_title='IMDb Rating',
    zaxis_title='Metascore'
),
    width=800,
    height=800
)

# Show plot
fig.show()

# This plot highlights how movies with different durations can still achieve critical and audience praise
```

[35]



❖ Distribution of IMBD Ratings by Genre. (interactive)

```python
# Split genres and create a new DataFrame with each genre in a separate row
genre_df = df.assign(Genre=df['Genre'].str.split(', ')).explode('Genre')

# Group by genre and calculate summary statistics for IMDb ratings
genre_summary = genre_df.groupby('Genre')['IMDB Rating'].describe().reset_index()

# Plotting
fig = px.box(genre_df, x='IMDB Rating', y='Genre', color='Genre',
             labels={'IMDB Rating': 'IMDb Rating', 'Genre': 'Genre'},
             title='IMDb Ratings Summary by Genre')

# Add red points for the mean of each box
for i, genre in enumerate(genre_summary['Genre']):
    mean_rating = genre_summary.loc[i, 'mean']
    fig.add_scatter(x=[mean_rating], y=[genre], mode='markers', marker=dict(color='red', size=10),
                    name='Mean')

# Update layout
fig.update_layout(width=1100, height=850)
fig.show()
```



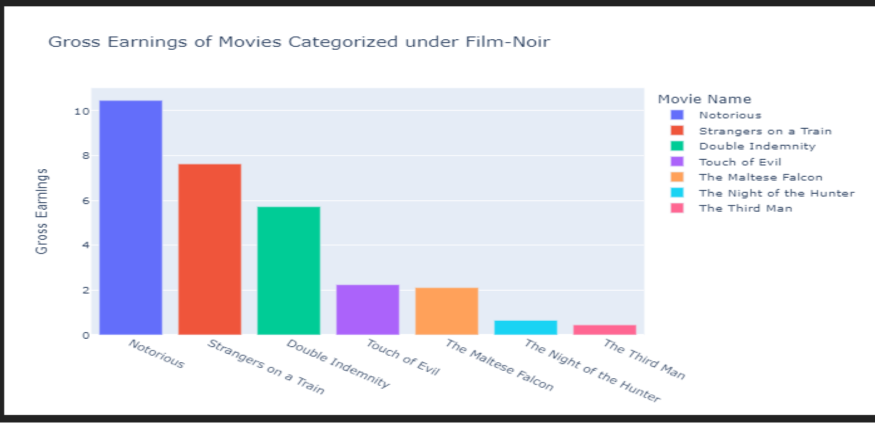
 # Genre with the lowest mean rating: Horror
 # Genre with the highest mean rating: Film-Noir

❖ Gross Earnings of Movies Categorized under Film-Noir. (interactive)

Gross Earnings of Movies Categorized under Film-Noir

```python
# Filter the dataframe for movies categorized under Film-Noir genre
Film_Noir_movies = df[df['Genre'].str.contains('Film-Noir', na=False)]

# Sort the movies by gross earnings
Film_Noir_movies_sorted = Film_Noir_movies.sort_values(by='Gross', ascending=False)

# Create a bar plot
fig = px.bar(Film_Noir_movies_sorted, x='Movie Name', y='Gross', color='Movie Name',
             title='Gross Earnings of Movies Categorized under Film-Noir',
             labels={'Gross': 'Gross Earnings', 'Movie Name': 'Movie Name'},
             hover_data={'IMDB Rating': True})

# Update layout
fig.update_layout(xaxis_title=None, yaxis_title='Gross Earnings',width=800, height=500)

# Show plot
fig.show()

# Films categorized under Film-Noir may have higher average ratings, their gross earnings might not necessarily reflect this
```

[38]

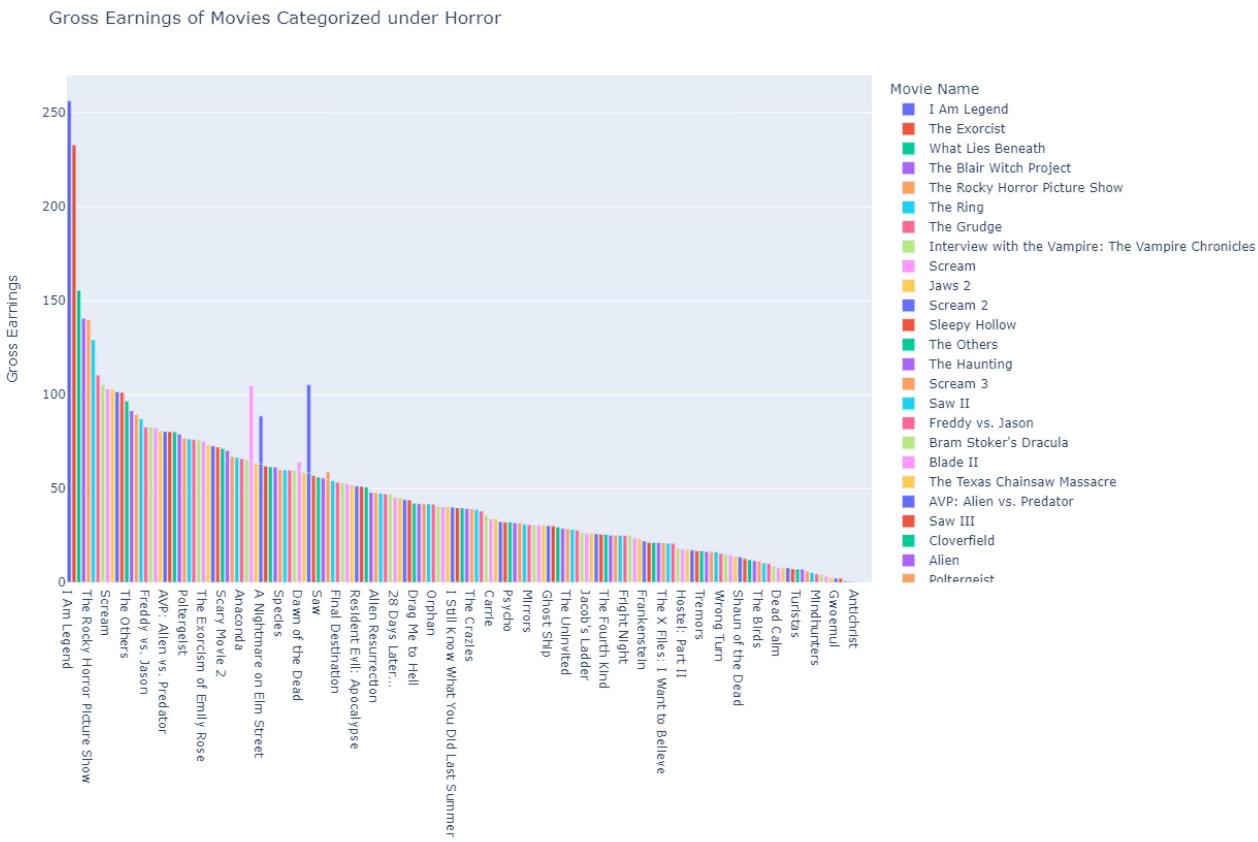


Films categorized under Film-Noir may have higher average ratings, their gross earnings might not necessarily reflect this

❖ Gross Earnings of Movies Categorized under Horror. (interactive)

```python
# Filter the dataframe for movies categorized under Horror genre
Horror_movies = df[df['Genre'].str.contains('Horror', na=False)]

# Sort the movies by gross earnings
Horror_movies_sorted = Horror_movies.sort_values(by='Gross', ascending=False)

# Create a bar plot
fig = px.bar(Horror_movies_sorted, x='Movie Name', y='Gross',color='Movie Name',
             title='Gross Earnings of Movies Categorized under Horror',
             labels={'Gross': 'Gross Earnings', 'Movie Name': 'Movie Name'},
             hover_data={'IMDB Rating': True})  # Include IMDb rating in hover data

# Update layout
fig.update_layout(xaxis_title=None, yaxis_title='Gross Earnings',width=1250, height=850)

# Show plot
fig.show()

# Films under the Horror genre seem to generate higher profits despite potentially lower average ratings
```

[39]



Films under the Horror genre seem to generate higher profits despite potentially lower average ratings

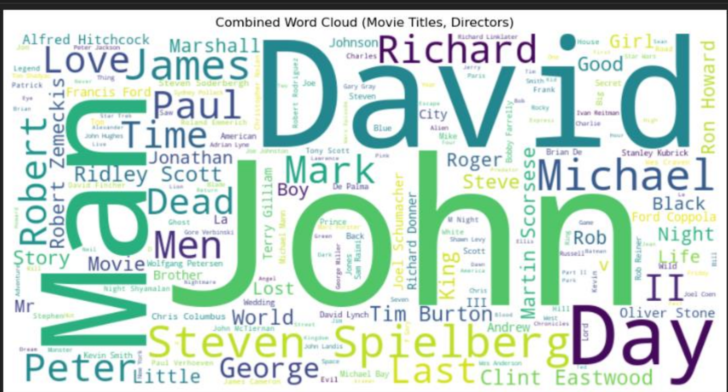
➔ We Notice that the Horror genre have higher Gross & Profits than Film-noir despite of Horror is the lowest mean rating and Film-noir is the highest mean rating.

❖ Word Count Plot (frequency of each word)

Word Cloud plot

```
combined_text = ' '.join(df['Movie Name']) + ' ' + ' '.join(df['Director'])
wordcloud_combined = WordCloud(width=800, height=400, background_color='white').generate(combined_text)
plt.figure(figsize=(18, 6))
plt.imshow(wordcloud_combined, interpolation='bilinear')
plt.title('Combined Word Cloud (Movie Titles, Directors)')
plt.axis('off')
plt.show()

# Larger and bolder words indicate higher frequency or prominence.
```

[40]

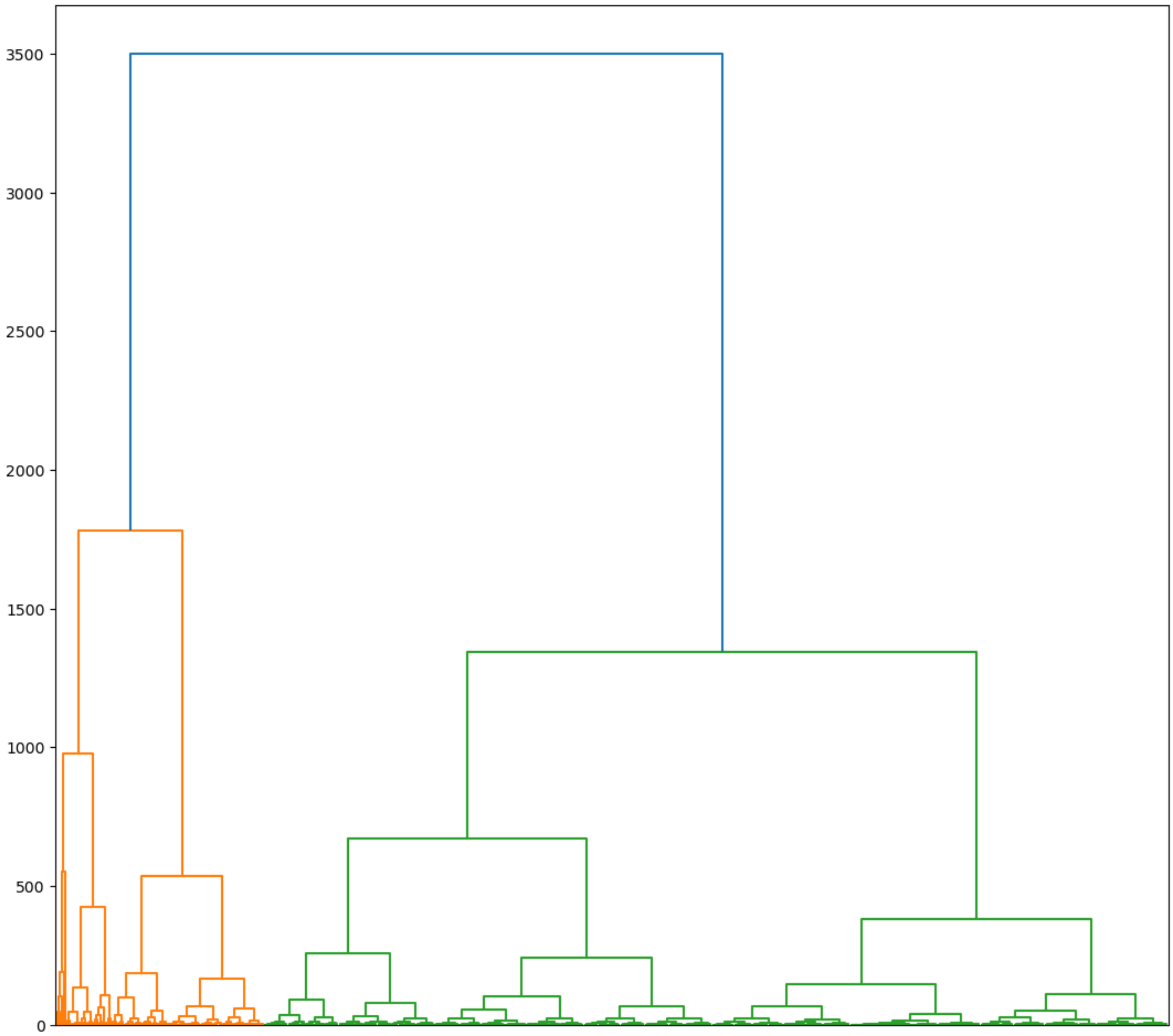


Larger and bolder words indicate higher frequency or prominence.

- ## Hierarchical Clustering

❖ Gross Feature Dendrogram

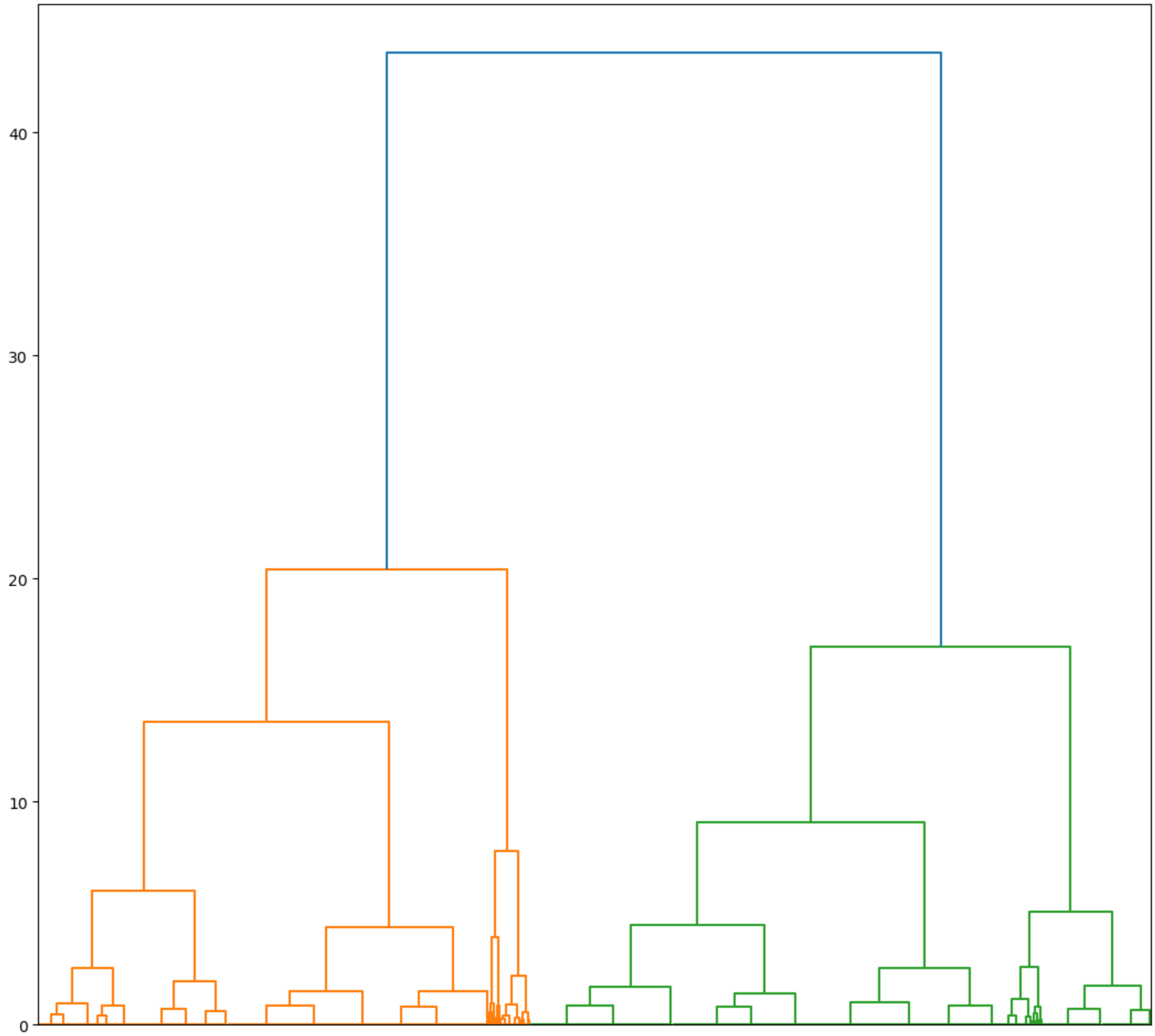
```
# Financial Performance
x1 = df[['Gross']]

Z = linkage(x1, method='ward', metric='euclidean')

labelList = list(df['Gross'])
plt.figure(figsize=(13, 12))
dendrogram(
Z,
orientation='top',
labels=labelList,
distance_sort='descending',
leaf_font_size=16)
plt.show()

# From the following dendrogram we can conclude that :-
# The clusters at the bottom represent movies with distinct low levels of financial success
# The separation between major clusters shows a clear distinction between movies with high and low gross performance
# The blue line indicates an outlier movie with significantly higher gross performance compared to the rest
```
[41]



 # The dense clustering at the bottom indicates a wide range of audience engagement levels, The blue lines represent movies that have won the appealing of the audience

❖ IMBD Rating Dendrogram

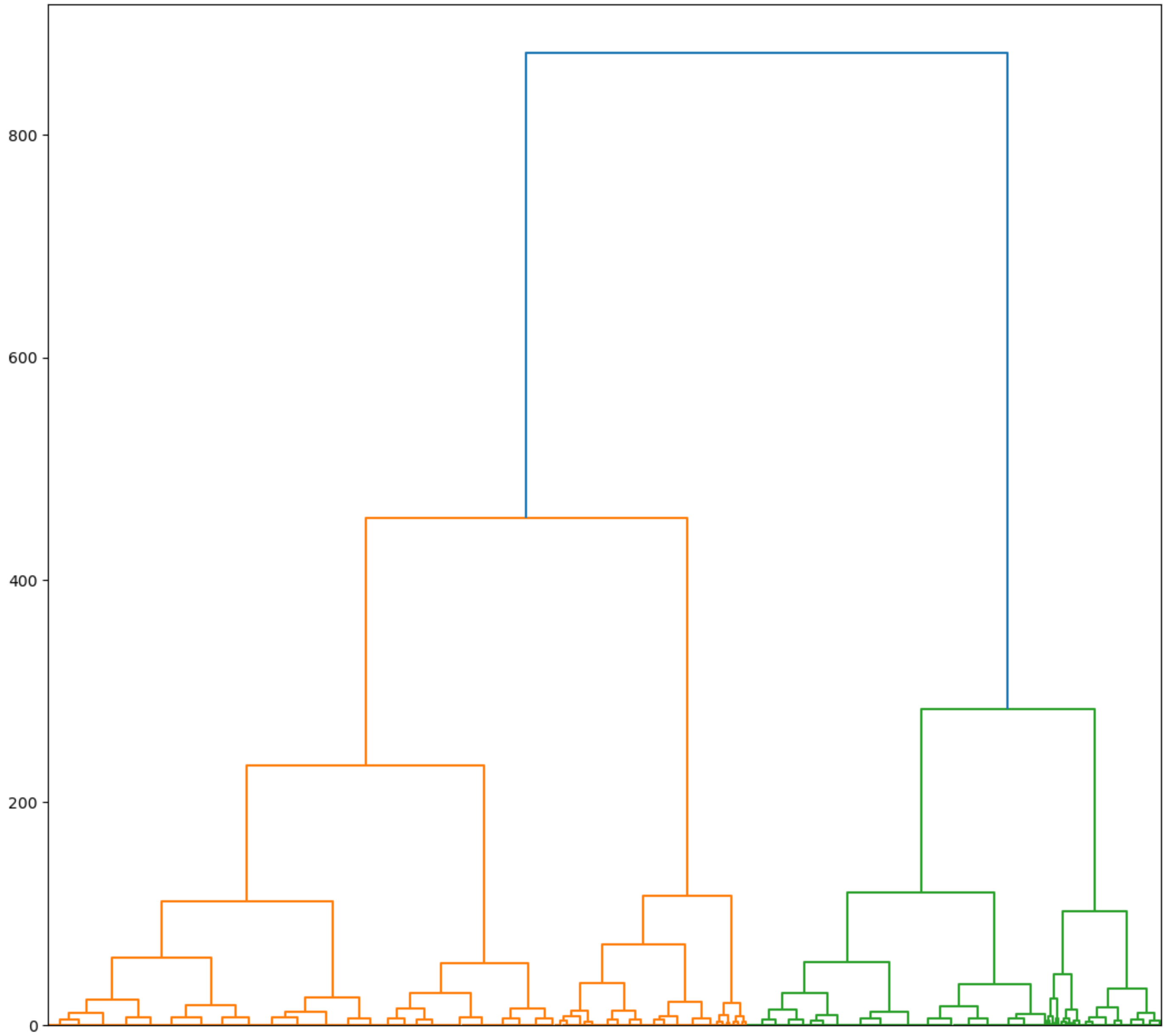
```python
# High and Low IMDB Rating movies
x2 = df[['IMDB Rating']]

Z = linkage(x2, method='ward', metric='euclidean')

labelList = list(df['IMDB Rating'])
plt.figure(figsize=(13, 12))
dendrogram(
Z,
orientation='top',
labels=labelList,
distance_sort='descending',
leaf_font_size=16)
plt.show()

# The tall lines in the graph represent movies with exceptionally high IMDB ratings
```
[42]



 # The tall lines in the graph represent movies with exceptionally high IMDB ratings

❖ Metascore Dendrogram

```python
# High and low metascore movies
x3 = df[['Metascore']]

Z = linkage(x3, method='ward', metric='euclidean')

labelList = list(df['Metascore'])
plt.figure(figsize=(13, 12))
dendrogram(
Z,
orientation='top',
labels=labelList,
distance_sort='descending',
leaf_font_size=16)
plt.show()

# The uniformity of the orange lines in the dendrogram below shows a consistent range of Metascore values across the dataset
# The lone blue line represents a movie with an outstanding Metascore, potentially a critically acclaimed masterpiece
```
[43]



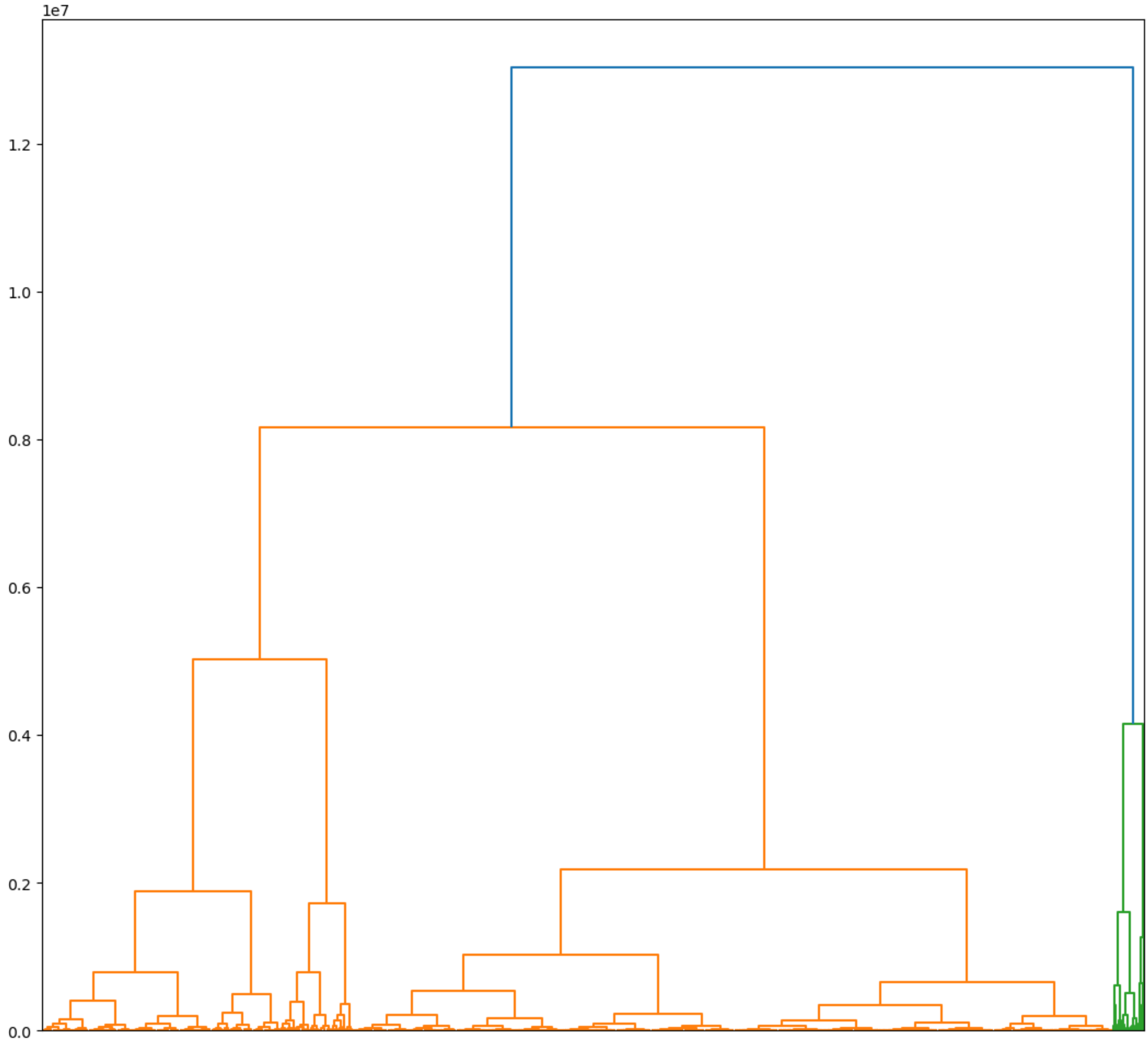
The uniformity of the orange lines in the dendrogram below shows a consistent range of Metascore values across the dataset
The lone blue line represents a movie with an outstanding Metascore, potentially a critically acclaimed masterpiece

❖ Votes Dendrogram

4. Votes Dendrogram

```python
# High and low rating movies
x4 = df[['Votes']]

Z = linkage(x4, method='ward', metric='euclidean')

labelList = list(df['Votes'])
plt.figure(figsize=(13, 12))
dendrogram(
Z,
orientation='top',
labels=labelList,
distance_sort='descending',
leaf_font_size=16)
plt.show()

# The dense clustering at the bottom indicates a wide range of audience engagement levels
# The blue lines represent movies that have won the appealing of the audience
```

[44]

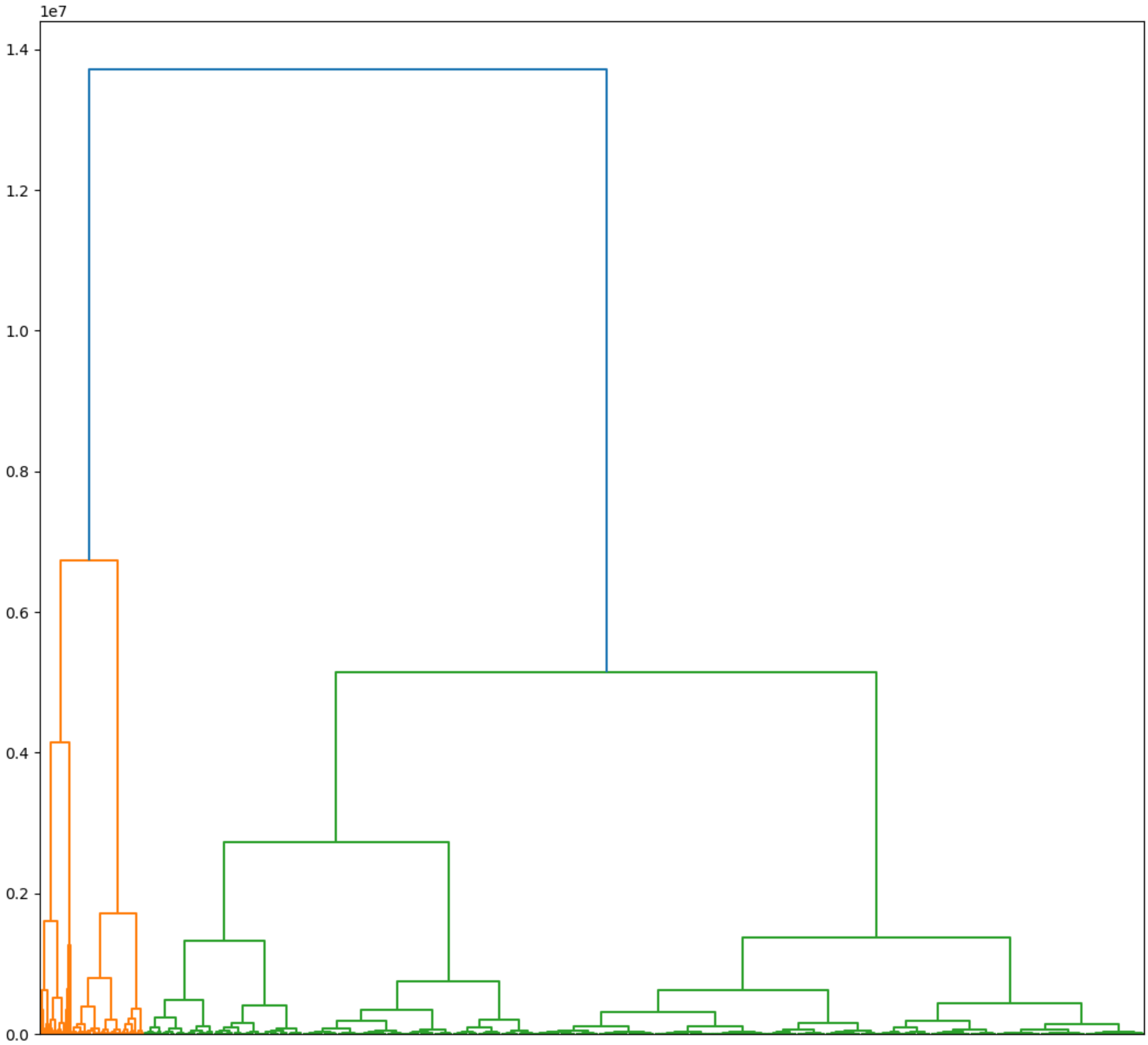


 # The dense clustering at the bottom indicates a wide range of audience engagement levels
 # The blue lines represent movies that have won the appealing of the audience

❖ IMDB Rating, Votes, Metascore Dendrogram

```python
# Rating and Popularity
x5 = df[['IMDB Rating','Votes','Metascore']]

Z = linkage(x5, method='ward', metric='euclidean')

labelList = list(zip(df['IMDB Rating'], df['Votes'], df['Metascore']))
plt.figure(figsize=(13, 12))
dendrogram(
Z,
orientation='top',
labels=labelList,
distance_sort='descending',
leaf_font_size=16)
plt.show()

# Clusters formed at higher levels group movies based on their overall reception, considering ratings, votes, and Metascore
```

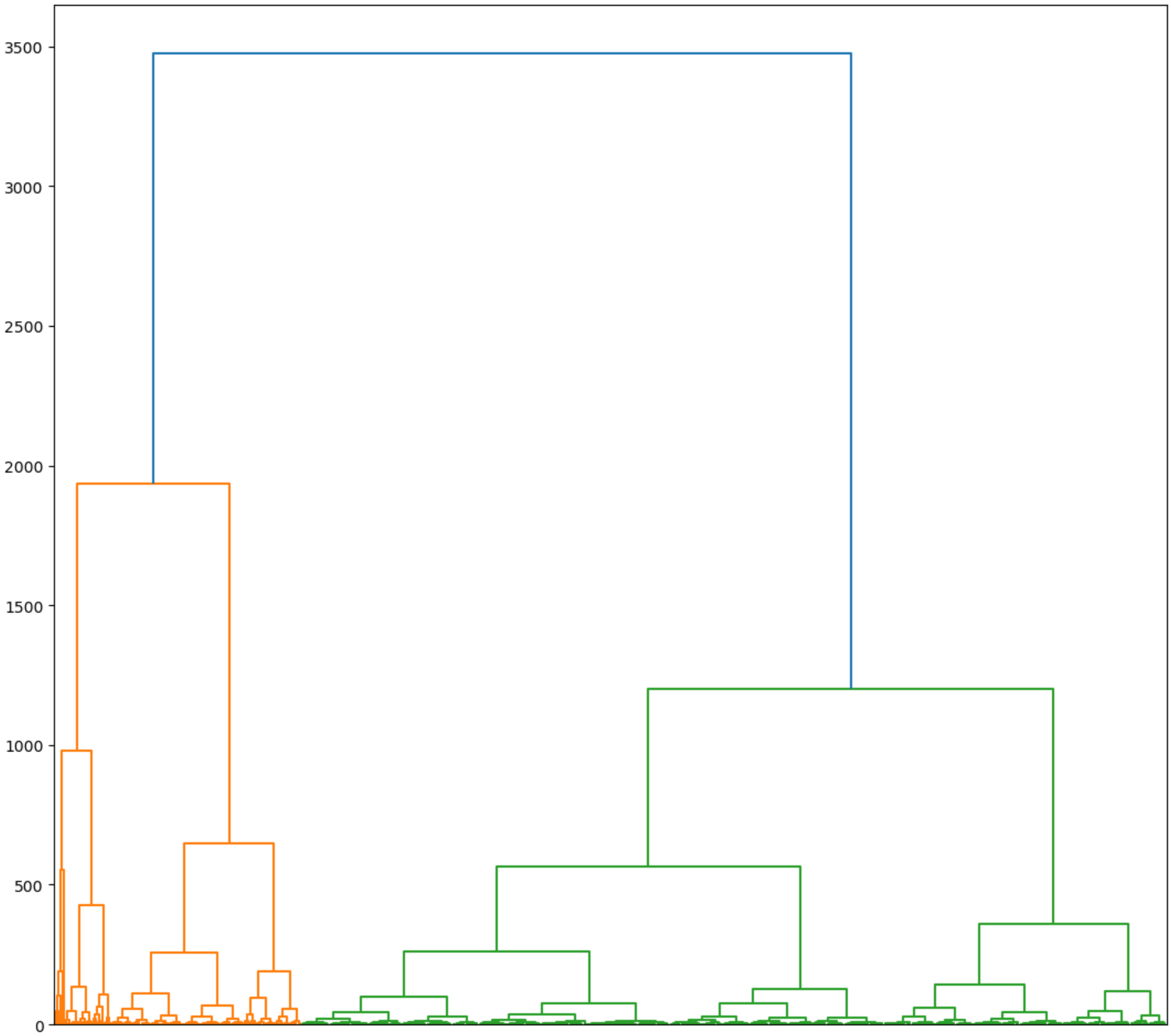
[45]



Clusters formed at higher levels group movies based on their overall reception, considering ratings, votes, and Metascore

❖ Gross, IMDB Rating Dendrogram

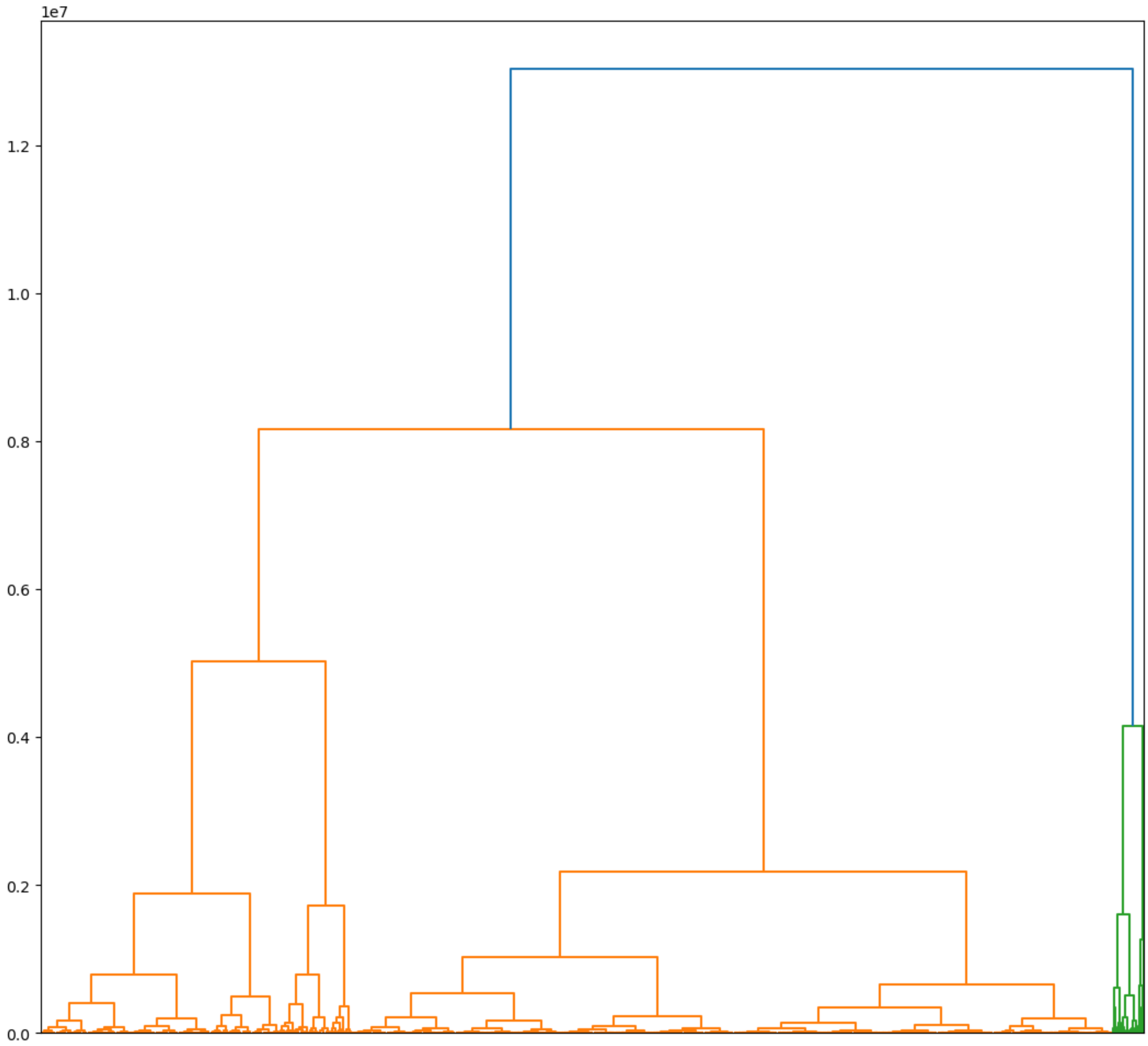
```python
#Financial success and IMDB Rating
x6 = df[['Gross','IMDB Rating']]
Z = linkage(x6, method='ward', metric='euclidean')

labelList = list(zip(df['Gross'], df['IMDB Rating']))
plt.figure(figsize=(13, 12))
dendrogram(
Z,
orientation='top',
labels=labelList,
distance_sort='descending',
leaf_font_size=16)
plt.show()

# Clusters formed based on a movie's financial success and critical acclaim (IMDB Rating) show the intersection of commerce and art in the movie industry.
```
[46]



 # Clusters formed based on a movie's financial success and critical acclaim (IMDB Rating) show the intersection of commerce and art in the movie industry.

❖ Gross, Votes Dendrogram

```
#Financial success and audience reception
x7 = df[['Gross','Votes']]

Z = linkage(x7, method='ward', metric='euclidean')

labelList = list(zip(df['Gross'], df['Votes']))
plt.figure(figsize=(13, 12))
dendrogram(
Z,
orientation='top',
labels=labelList,
distance_sort='descending',
leaf_font_size=16)
plt.show()

# Clusters based on audience reception and financial success shows the balance between pleasing crowds and making money
# The blue line shows where financial success and audience popularity intersect
```
[47]

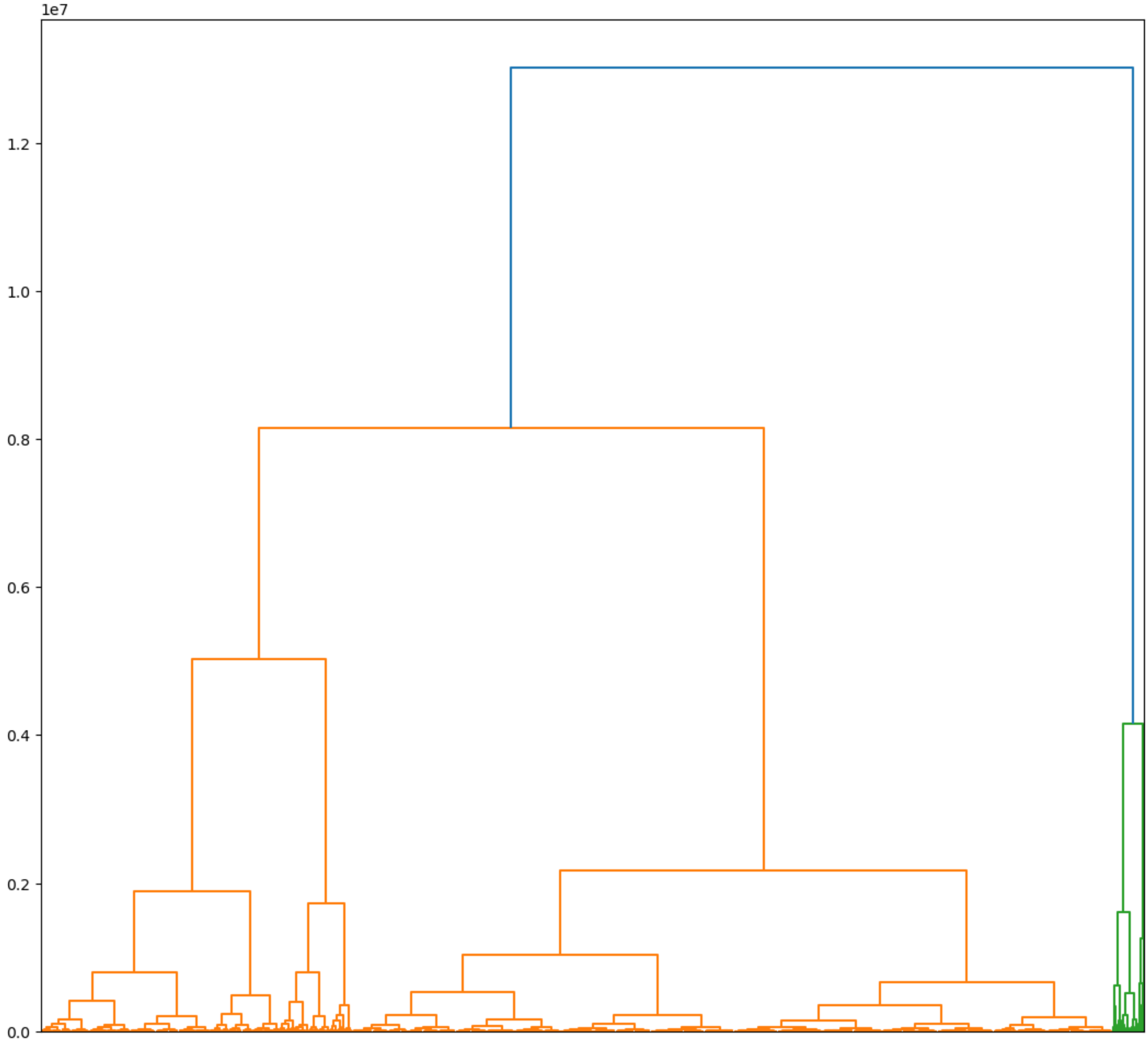


 # Clusters based on audience reception and financial success shows the balance between pleasing crowds and making money

 # The blue line shows where financial success and audience popularity intersect

❖ IMDB Rating, Votes Dendrogram

```python
# Critical acclaim and Popular appeal
x8 = df[['IMDB Rating','Votes']]

Z = linkage(x8, method='ward', metric='euclidean')

labelList = list(zip(df['IMDB Rating'], df['Votes'],))
plt.figure(figsize=(13, 12))
dendrogram(
Z,
orientation='top',
labels=labelList,
distance_sort='descending',
leaf_font_size=16)
plt.show()

# Clusters formed based on ratings and audience shows the dynamics between critical acclaim and public reception and appeal
```
[48]



 # Clusters formed based on ratings and audience shows the dynamics between critical acclaim and public reception and appeal

❖ Add the cluster labels for each row in the dataframe

```
df['Clusters'] = fcluster(Z,2) # 2 is the no. of clusters
df

# Here we add a new column named 'Clusters' which indicate the cluster of each row in our data set
# As we notice from the output below all rows belong to the same cluster which is 1
# That is because we use Agglomerative (bottom-up) approach which combines all the data points into the same cluster
```
[49]

	Movie Name	Release Year	Duration	IMDB Rating	Metascore	Votes	Genre	Director	Cast	Gross	Clusters
0	The Godfather	1972	175	9.2	100.0	2002655.0	Crime, Drama	Francis Ford Coppola	Marlon Brando	134.97	1
1	The Godfather Part II	1974	202	9.0	90.0	1358608.0	Crime, Drama	Francis Ford Coppola	Al Pacino	57.30	1
2	Ordinary People	1980	124	7.7	86.0	56476.0	Drama	Robert Redford	Donald Sutherland	54.80	1
3	Lawrence of Arabia	1962	218	8.3	100.0	313044.0	Adventure, Biography, Drama	David Lean	Peter O'Toole	44.82	1
5	Close Encounters of the Third Kind	1977	138	7.6	90.0	216050.0	Drama, Sci-Fi	Steven Spielberg	Richard Dreyfuss	132.09	1
...	...	...	...	...	...	...	...	...	...	...	...
1995	The Young Victoria	2009	105	7.2	64.0	66235.0	Biography, Drama, History	Jean-Marc Vallée	Emily Blunt	11.00	1
1996	Tooth Fairy	2010	101	5.0	36.0	49527.0	Comedy, Family, Fantasy	Michael Lembeck	Dwayne Johnson	60.02	1
1997	The Informant!	2009	108	6.5	66.0	67318.0	Biography, Comedy, Crime	Steven Soderbergh	Matt Damon	33.31	1
1998	Youth in Revolt	2009	90	6.4	63.0	75956.0	Comedy, Drama, Romance	Miguel Arteta	Michael Cera	15.28	1
1999	Quarantine	2008	89	6.0	53.0	77075.0	Horror, Sci-Fi, Thriller	John Erick Dowdle	Jennifer Carpenter	31.69	1

1870 rows × 11 columns

- ## K-medoid Clustering

```
# List all movie names in a variable called labels
labels = list(df['Movie Name'])
```
[50]

```
# StandardScaler function to standarize data to make them consistent
scale = StandardScaler()
```
[51]

1. Clustering based on IMDB Rating & Gross

```
# We will take 2 columns to perform k-medoids clustring on them
imdbr_gross = df.loc[:, ['IMDB Rating', 'Gross']]

# As we can see from the output below the values is far from each other
# So we will scale them to apply K_Medoids
imdbr_gross.head()
```
[52]

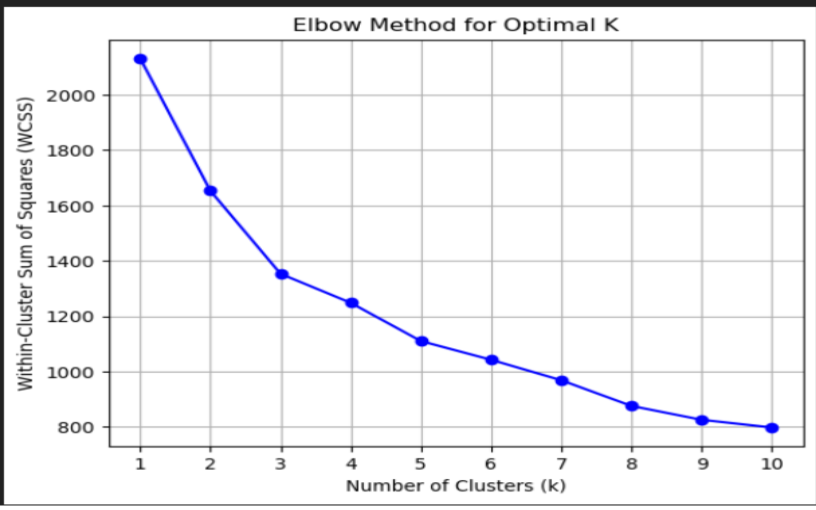
	IMDB Rating	Gross
0	9.2	134.97
1	9.0	57.30
2	7.7	54.80
3	8.3	44.82
5	7.6	132.09

```
scaled_imdbr_gross = scale.fit_transform(imdbr_gross)

# Here is the data after standarization
scaled_imdbr_gross
```
[53]

```
array([[ 2.50734377,  0.92922658],
       [ 2.28762293, -0.13348634],
       [ 0.85943748, -0.16769237],
       ...,
       [-0.45888756, -0.4617274 ],
       [-0.56874798, -0.70842128],
       [-1.00818966, -0.48389291]])
```

This method determines what is the optimal number of clusters in our data set.

```python
wcss = []
for i in range(1, 11):
    kmedoids = KMedoids(n_clusters=i, random_state=0)
    kmedoids.fit(scaled_imdbr_gross)
    wcss.append(kmedoids.inertia_)

# Plot the elbow curve
plt.plot(range(1, 11), wcss, marker='o', linestyle='-', color='b')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.xticks(np.arange(1, 11, 1))
plt.grid(True)
plt.show()

# This method determines what is the optimal number of clusters in our data set
# The best number of clusters located at the Elbow point (which in our case is 3)
```



The best number of clusters located at the Elbow point (which in our case is 3)

```python
# Perform k_Medoids on our scaled data ['IMDB Rating', 'Gross'] columns
# n_clusters parameter is 3 as we choose above
imdbr_gorss_km = KMedoids(n_clusters=3).fit(scaled_imdbr_gross)

# Create an array with clusters labels (the cluster that the data point belong to)
imdbr_gorss_km_labels = imdbr_gorss_km.labels_
print("The cluster labels are:\n", imdbr_gorss_km_labels, '\n')

# Determine the medoids
imdbr_gorss_medoids = imdbr_gorss_km.cluster_centers_
print("The medoids are:\n", imdbr_gorss_medoids)
```

```
The cluster labels are:
 [2 0 0 ... 1 1 1]

The medoids are:
 [[ 0.63971664 -0.50646888]
 [-0.78846882 -0.314231  ]
 [ 0.20027496  1.06372469]]
```

K-Medoids Silhouette Score of IMDB Rating & gross (Accuracy)

```
kmedoids_silhouette_score = silhouette_score(scaled_imdbr_gross, imdbr_gorss_km_labels)
print("K-Medoids Silhouette Score:", kmedoids_silhouette_score)
```

[255] ✓ 0.0s

```
K-Medoids Silhouette Score: 0.4009900122179122
```

Here we add a new column named 'imdbr_gorss_clusters' which indicate the cluster of each row in our data set

```
df['imdbr_gross_clusters'] = imdbr_gorss_km_labels
df
# Here we add a new column named 'imdbr_gorss_clusters' which indicate the cluster of each row in our data set
```

[56]

	Movie Name	Release Year	Duration	IMDB Rating	Metascore	Votes	Genre	Director	Cast	Gross	Clusters	imdbr_gross_clusters
0	The Godfather	1972	175	9.2	100.0	2002655.0	Crime, Drama	Francis Ford Coppola	Marlon Brando	134.97	1	2
1	The Godfather Part II	1974	202	9.0	90.0	1358608.0	Crime, Drama	Francis Ford Coppola	Al Pacino	57.30	1	0
2	Ordinary People	1980	124	7.7	86.0	56476.0	Drama	Robert Redford	Donald Sutherland	54.80	1	0
3	Lawrence of Arabia	1962	218	8.3	100.0	313044.0	Adventure, Biography, Drama	David Lean	Peter O'Toole	44.82	1	0
5	Close Encounters of the Third Kind	1977	138	7.6	90.0	216050.0	Drama, Sci-Fi	Steven Spielberg	Richard Dreyfuss	132.09	1	2
...	...	...	...	...	...	...	...	...	...	...	...	...
1995	The Young Victoria	2009	105	7.2	64.0	66235.0	Biography, Drama, History	Jean-Marc Vallée	Emily Blunt	11.00	1	0
1996	Tooth Fairy	2010	101	5.0	36.0	49527.0	Comedy, Family, Fantasy	Michael Lembeck	Dwayne Johnson	60.02	1	1
1997	The Informant!	2009	108	6.5	66.0	67318.0	Biography, Comedy, Crime	Steven Soderbergh	Matt Damon	33.31	1	1
1998	Youth in Revolt	2009	90	6.4	63.0	75956.0	Comedy, Drama, Romance	Miguel Arteta	Michael Cera	15.28	1	1
1999	Quarantine	2008	89	6.0	53.0	77075.0	Horror, Sci-Fi, Thriller	John Erick Dowdle	Jennifer Carpenter	31.69	1	1

1870 rows × 12 columns

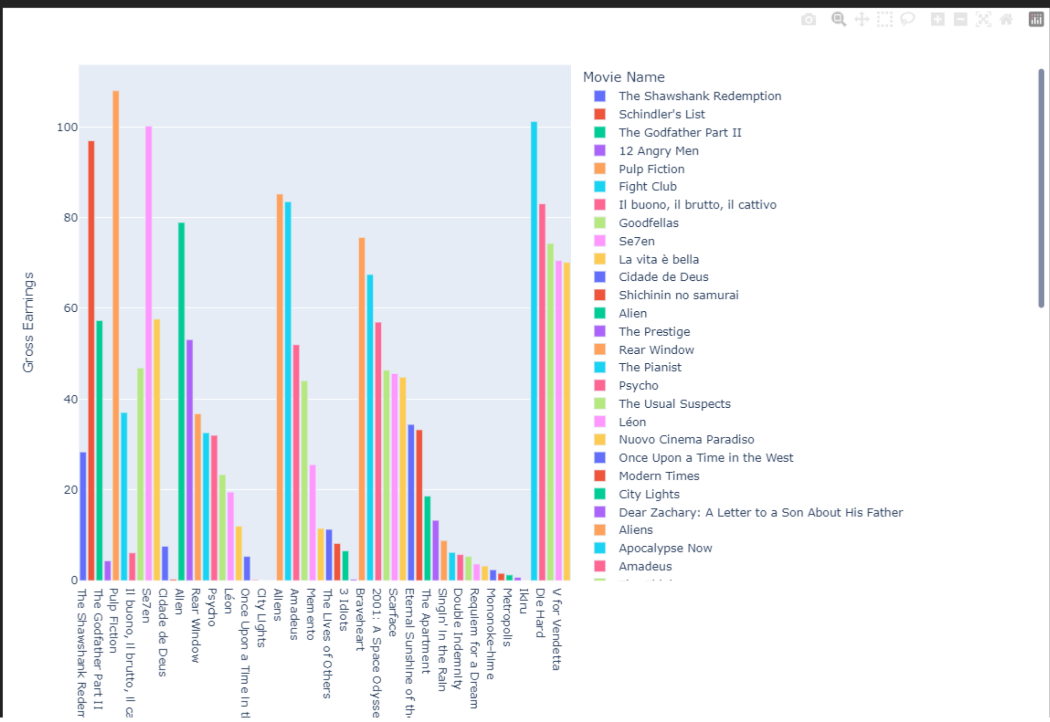
```
df_sorted_gross = new_df.sort_values(by = ['IMDB Rating', 'Gross'], ascending=False)
```

[57]

❖ Cluster 0 (low gross high rating)

```
fig = px.bar(df_sorted_gross.loc[df['imdbr_gross_clusters'] == 0].head(60), x = 'Movie Name', y = 'Gross', color = 'Movie Name', hover_data={'IMDB Rating': True})
fig.update_layout(xaxis_title=None, yaxis_title='Gross Earnings', width=1100, height=850)
fig.show()
```

[58]



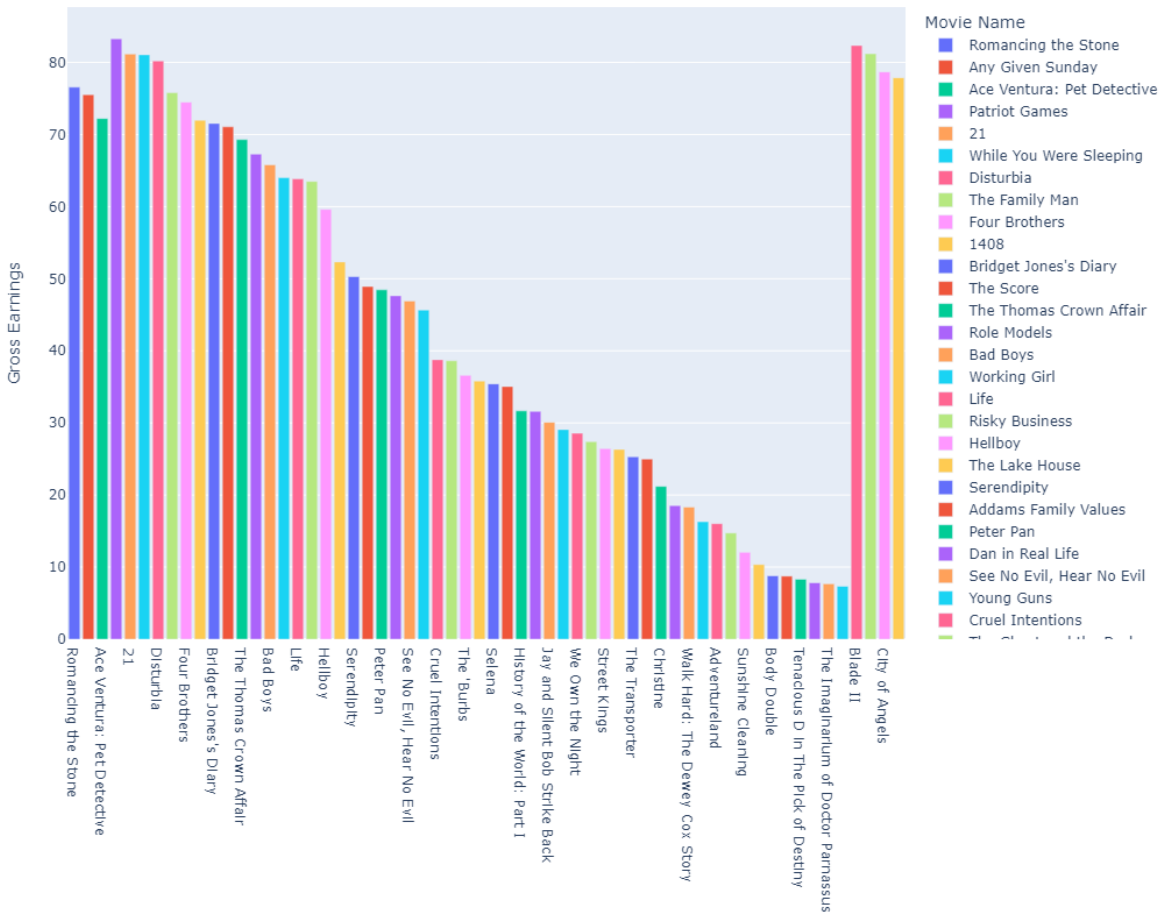
This bar plot represent cluster 0 sorted by IMDB Rating first then gross (Priority to IMDB Rating)

From the graph we can conclude that films with a high rating not necessarily have a high gross

❖ Cluster 1 (low gross low rating)

```python
# Cluster 1 plot
# These films have low rating and low gross (not attractive to audiences)
fig = px.bar(df_sorted_gross.loc[df['imdbr_gross_clusters'] == 1].head(60), x = 'Movie Name', y = 'Gross', color = 'Movie Name', hover_data={'IMDB Rating': True})
fig.update_layout(xaxis_title=None, yaxis_title='Gross Earnings', width=1100, height=850)
fig.show()
```

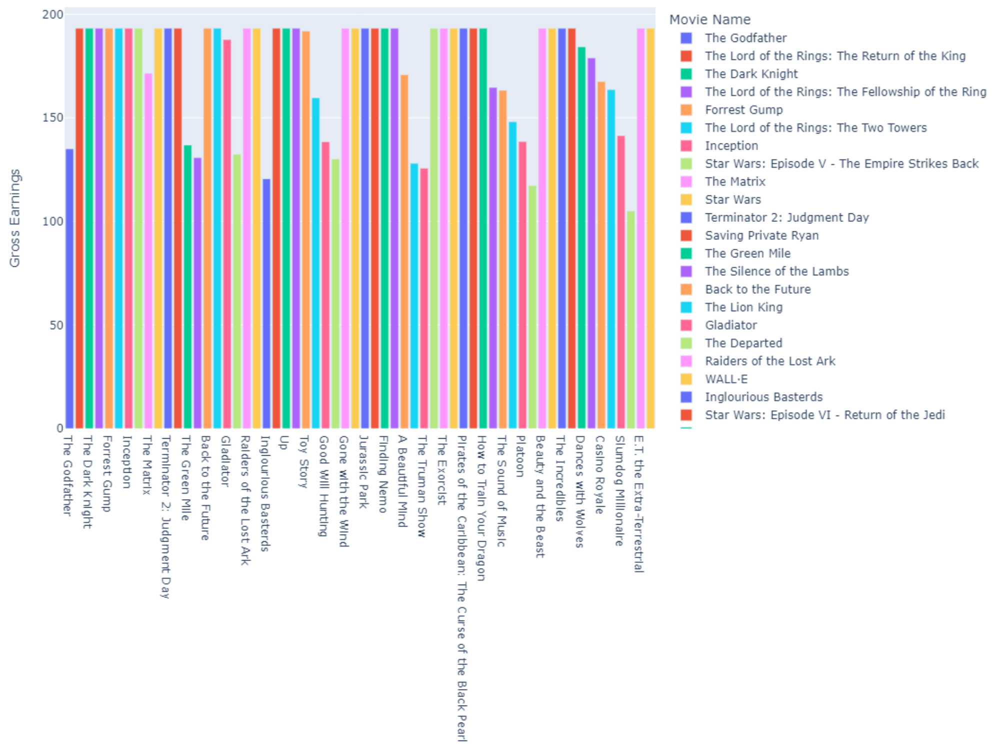
[59]



These films have low rating and low gross (not attractive to audiences)

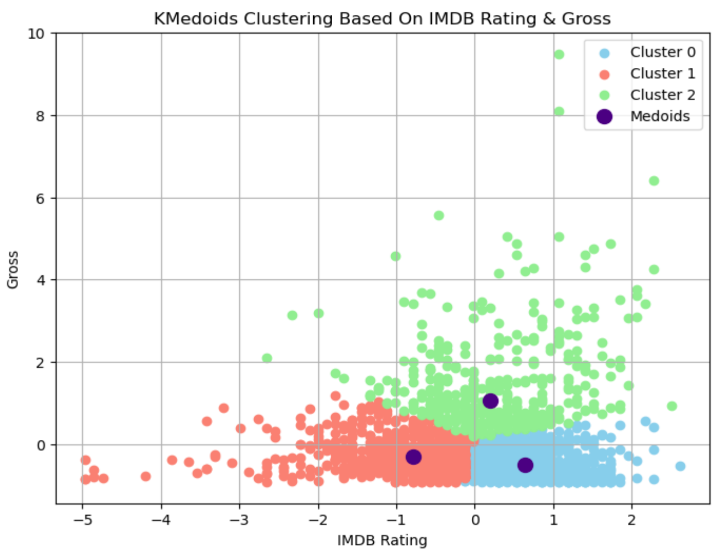
❖ Cluster 2 (high gross high rating)

```python
fig = px.bar(df_sorted_gross.loc[df['imdbr_gross_clusters'] == 2].head(60), x = 'Movie Name', y = 'Gross', color = 'Movie Name', hover_data={'IMDB Rating': True})
fig.update_layout(xaxis_title=None, yaxis_title='Gross Earnings', width=1100, height=850)
fig.show()
```



These films have high rating and high gross (masterpiece movies)

```python
plt.figure(figsize = (8, 6))

# We will use scatter plot to visualize our scaled_imdbr_gross data frame

# Plot Cluster 0
plt.scatter(x = scaled_imdbr_gross[df['imdbr_gross_clusters'] == 0, 0], y = scaled_imdbr_gross[df['imdbr_gross_clusters'] == 0, 1], c='skyblue', label='Cluster 0')

# Plot Cluster 1
plt.scatter(x = scaled_imdbr_gross[df['imdbr_gross_clusters'] == 1,0], y = scaled_imdbr_gross[df['imdbr_gross_clusters'] == 1, 1], c='salmon', label='Cluster 1')

# Plot Cluster 2
plt.scatter(x = scaled_imdbr_gross[df['imdbr_gross_clusters'] == 2,0], y = scaled_imdbr_gross[df['imdbr_gross_clusters'] == 2,1], c='lightgreen', label='Cluster 2')


# Place our medoids in the graph with a different color than the clusters to distinguish them
plt.scatter(imdbr_gorss_medoids[:, 0], imdbr_gorss_medoids[:, 1], marker='o', c='indigo', s=100, label='Medoids')

plt.title('KMedoids Clustering Based On IMDB Rating & Gross')
plt.xlabel('IMDB Rating')
plt.ylabel('Gross')

# Legend function to more information about the graph
plt.legend()

plt.grid()
plt.show()
```



The cluster in the upper right corner could represent high-grossing, high-rated movies. These might be blockbuster films that are both critically acclaimed and popular with audiences.

The cluster in the lower left corner could represent low-grossing, low-rated movies. These might be independent films or art house films that have not found a wide audience.

The other clusters could represent movies that fall somewhere in between, in terms of both gross and rating.

❖ Conclusion

```
# Low 'Gross' High 'Rating' (Cluster 0)
print('Number of movies cluster 0: ', df.loc[df['imdbr_gross_clusters'] == 0].shape[0])
print('cluster 0 mean gross : ', df.loc[df['imdbr_gross_clusters'] == 0]['Gross'].mean())
print('cluster 0 mean rating : ', df.loc[df['imdbr_gross_clusters'] == 0]['IMDB Rating'].mean())

print('#------------\n')


# low 'Gross' low 'Rating' (Cluster 1)
print('Number of movies cluster 1: ', df.loc[df['imdbr_gross_clusters'] == 1].shape[0])
print('cluster 1 mean gross : ', df.loc[df['imdbr_gross_clusters'] == 1]['Gross'].mean())
print('cluster 1 mean rating : ', df.loc[df['imdbr_gross_clusters'] == 1]['IMDB Rating'].mean())

print('#------------\n')

# high 'Gross' high 'Rating' (Cluster 2)
print('Number of movies cluster 2: ', df.loc[df['imdbr_gross_clusters'] == 2].shape[0])
print('cluster 2 mean gross : ', df.loc[df['imdbr_gross_clusters'] == 2]['Gross'].mean())
print('cluster 2 mean rating : ', df.loc[df['imdbr_gross_clusters'] == 2]['IMDB Rating'].mean())
```
[62]

```
Number of movies cluster 0:  794
cluster 0 mean gross :  31.199319899244337
cluster 0 mean rating :  7.547229219143577
#------------

Number of movies cluster 1:  686
cluster 1 mean gross :  46.89472303206997
cluster 1 mean rating :  6.057434402332361
#------------

Number of movies cluster 2:  390
cluster 2 mean gross :  175.51994871794872
cluster 2 mean rating :  7.149230769230769
```

2. Clustering based on IMDB Rating & Duration

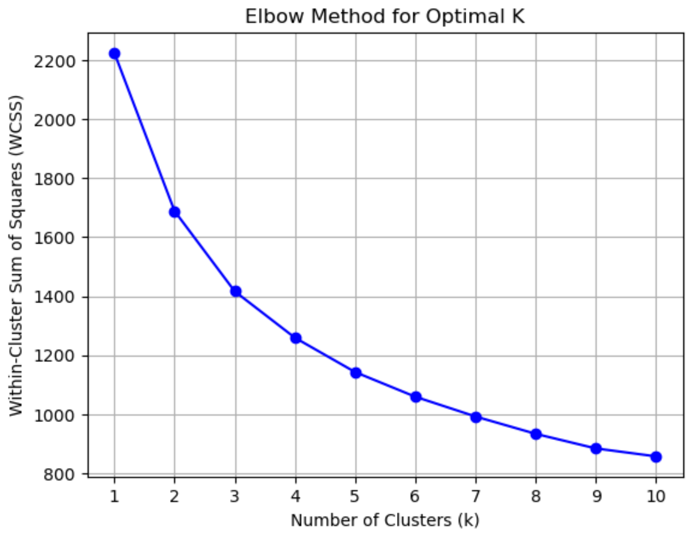
```
imdbr_duration = df.loc[:, ['IMDB Rating', 'Duration']]

# Like the point 1 we will perform scaling here for the same reason
imdbr_duration.head()
```
[63]

	IMDB Rating	Duration
0	9.2	175
1	9.0	202
2	7.7	124
3	8.3	218
5	7.6	138

```
scaled_imdbr_duration = scale.fit_transform(imdbr_duration)

# The data after standarization
scaled_imdbr_duration
```
[64]

```
array([[ 2.50734377,  2.75843486],
       [ 2.28762293,  3.97982044],
       [ 0.85943748,  0.45137321],
       ...,
       [-0.45888756, -0.27241083],
       [-0.56874798, -1.08666789],
       [-1.00818966, -1.13190439]])
```

```python
wcss = []
for i in range(1, 11):
    kmedoids = KMedoids(n_clusters=i, random_state=0)
    kmedoids.fit(scaled_imdbr_duration)
    wcss.append(kmedoids.inertia_)
# Plot the elbow curve

plt.plot(range(1, 11), wcss, marker='o', linestyle='-', color='b')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.xticks(np.arange(1, 11, 1))
plt.grid(True)
plt.show()

# As we can see there is no Elbow point because the duration, gross relation can not clustring the data well
# So we will choose 3 as a random number of clusters
```
[65]



As we can see there is no Elbow point because the duration, gross relation cannot clustering the data well

So, we will choose 3 as a random number of clusters

K-Medoids Silhouette Score of IMDB Rating & Duration (Accuracy)

```python
kmedoids_silhouette_score2 = silhouette_score(scaled_imdbr_duration, imdbr_duration_km_labels)
print("K-Medoids Silhouette Score:", kmedoids_silhouette_score2)
```
[267] ✓ 0.0s

```
K-Medoids Silhouette Score: 0.3363601181170214
```

```python
# Perform k_Medoids on our scaled data ['IMDB Rating', 'Duration'] columns
imdbr_duration_km = KMedoids(n_clusters=3).fit(scaled_imdbr_duration)

imdbr_duration_km_labels =imdbr_duration_km.labels_
print("The cluster labels are:\n", imdbr_duration_km_labels, '\n')

imdbr_duration_medoids = imdbr_duration_km.cluster_centers_
print("The medoids are:\n", imdbr_duration_medoids)
```
[66] Python

```
The cluster labels are:
 [1 1 1 ... 0 0 0]

The medoids are:
 [[-0.89832924 -0.45335685]
 [ 0.63971664  0.85850174]
 [ 0.4199958  -0.54382985]]
```

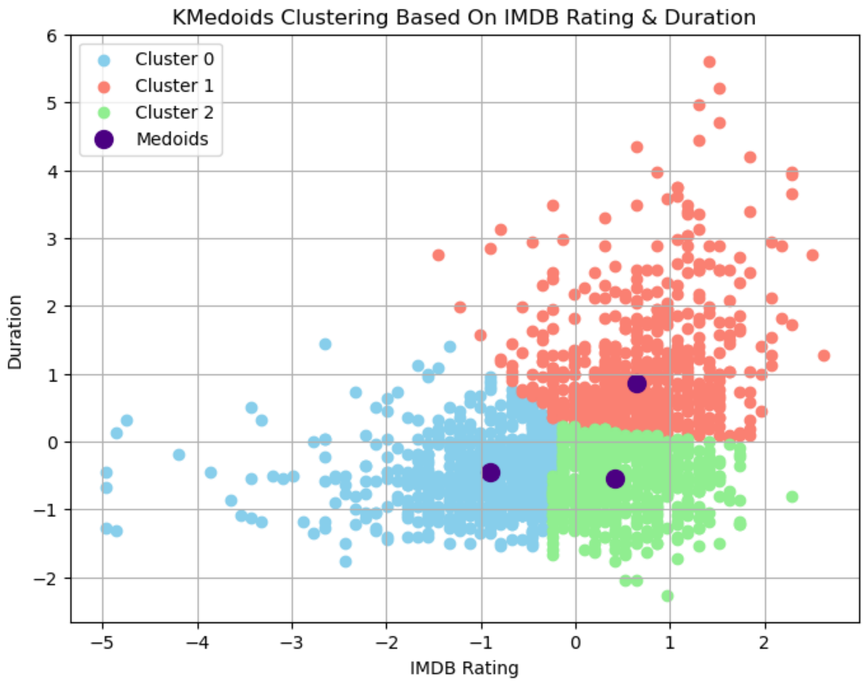
```python
df['imdbr_duration_clusters'] = imdbr_duration_km_labels
df
```
[67] Python

	Movie Name	Release Year	Duration	IMDB Rating	Metascore	Votes	Genre	Director	Cast	Gross	Clusters	imdbr_gross_clusters	imdbr_duration_clusters
0	The Godfather	1972	175	9.2	100.0	2002655.0	Crime, Drama	Francis Ford Coppola	Marlon Brando	134.97	1	2	1
1	The Godfather Part II	1974	202	9.0	90.0	1358608.0	Crime, Drama	Francis Ford Coppola	Al Pacino	57.30	1	0	1
2	Ordinary People	1980	124	7.7	86.0	56476.0	Drama	Robert Redford	Donald Sutherland	54.80	1	0	1
3	Lawrence of Arabia	1962	218	8.3	100.0	313044.0	Adventure, Biography, Drama	David Lean	Peter O'Toole	44.82	1	0	1
5	Close Encounters of the Third Kind	1977	138	7.6	90.0	216050.0	Drama, Sci-Fi	Steven Spielberg	Richard Dreyfuss	132.09	1	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	The Young Victoria	2009	105	7.2	64.0	66235.0	Biography, Drama, History	Jean-Marc Vallée	Emily Blunt	11.00	1	0	2
1996	Tooth Fairy	2010	101	5.0	36.0	49527.0	Comedy, Family, Fantasy	Michael Lembeck	Dwayne Johnson	60.02	1	1	0
1997	The Informant!	2009	108	6.5	66.0	67318.0	Biography, Comedy, Crime	Steven Soderbergh	Matt Damon	33.31	1	1	0
1998	Youth in Revolt	2009	90	6.4	63.0	75956.0	Comedy, Drama, Romance	Miguel Arteta	Michael Cera	15.28	1	1	0
1999	Quarantine	2008	89	6.0	53.0	77075.0	Horror, Sci-Fi, Thriller	John Erick Dowdle	Jennifer Carpenter	31.69	1	1	0

1870 rows × 13 columns

From the following graph we can conclude that:-

```python
plt.figure(figsize=(8, 6))

# Plot Cluster 0
plt.scatter(x = scaled_imdbr_duration[df['imdbr_duration_clusters'] == 0, 0], y = scaled_imdbr_duration[df['imdbr_duration_clusters'] == 0, 1], c='skyblue', label='Cluster 0')

# Plot Cluster 1
plt.scatter(x = scaled_imdbr_duration[df['imdbr_duration_clusters'] == 1,0], y = scaled_imdbr_duration[df['imdbr_duration_clusters'] == 1, 1], c='salmon', label='Cluster 1')

# Plot Cluster 2
plt.scatter(x = scaled_imdbr_duration[df['imdbr_duration_clusters'] == 2,0], y = scaled_imdbr_duration[df['imdbr_duration_clusters'] == 2,1], c='lightgreen', label='Cluster 2')


# Place our medoids in the graph with a different color than the clusters to distinguish them
plt.scatter(imdbr_duration_medoids[:, 0], imdbr_duration_medoids[:, 1], marker='o', c='indigo', s=100, label='Medoids')

plt.title('KMedoids Clustering Based On IMDB Rating & Duration')
plt.xlabel('IMDB Rating')
plt.ylabel('Duration')

# Legend function to more information about the graph
plt.legend()

plt.grid()
plt.show()
```



The movies in the upper right cluster tend to have high IMDB ratings and long durations. These movies might be epics, dramas, or documentaries.

The movies in the lower left cluster tend to have low IMDB ratings and short durations. These movies might be comedies, horror films, or action films.

There are a few movies that are outliers, meaning that they do not fit neatly into any of the clusters. These movies might be cult classics or films that defy genre classification.

❖ Conclusion

```python
print('Number of movies cluster 0: ', df.loc[df['imdbr_duration_clusters'] == 0].shape[0])
print('cluster 0 mean Duration : ', df.loc[df['imdbr_duration_clusters'] == 0]['Duration'].mean())
print('cluster 0 mean rating : ', df.loc[df['imdbr_duration_clusters'] == 0]['IMDB Rating'].mean())

print('#------------\n')

print('Number of movies cluster 1: ', df.loc[df['imdbr_duration_clusters'] == 1].shape[0])
print('cluster 1 mean gross : ', df.loc[df['imdbr_duration_clusters'] == 1]['Duration'].mean())
print('cluster 1 mean rating : ', df.loc[df['imdbr_duration_clusters'] == 1]['IMDB Rating'].mean())

print('#------------\n')

print('Number of movies cluster 2: ', df.loc[df['imdbr_duration_clusters'] == 2].shape[0])
print('cluster 2 mean gross : ', df.loc[df['imdbr_duration_clusters'] == 2]['Duration'].mean())
print('cluster 2 mean rating : ', df.loc[df['imdbr_duration_clusters'] == 2]['IMDB Rating'].mean())

# We can not well defined the proberties of each cluster
```

[69]

```
Number of movies cluster 0:  659
cluster 0 mean Duration :  104.14415781487102
cluster 0 mean rating :  5.973899848254932
#------------

Number of movies cluster 1:  591
cluster 1 mean gross :  138.4331641285956
cluster 1 mean rating :  7.523011844331642
#------------

Number of movies cluster 2:  620
cluster 2 mean gross :  101.2516129032258
cluster 2 mean rating :  7.343870967741936
```

3. Check popularity and Gross

```
# Create a data frame named gross_votes has only ['Gross', 'Votes'] columns
gross_votes = df[['Gross', 'Votes']]
gross_votes.head()
```

[70]

	Gross	Votes
0	134.97	2002655.0
1	57.30	1358608.0
2	54.80	56476.0
3	44.82	313044.0
5	132.09	216050.0

```
scaled_gross_votes = scale.fit_transform(gross_votes)

# The data after standarization
scaled_gross_votes
```

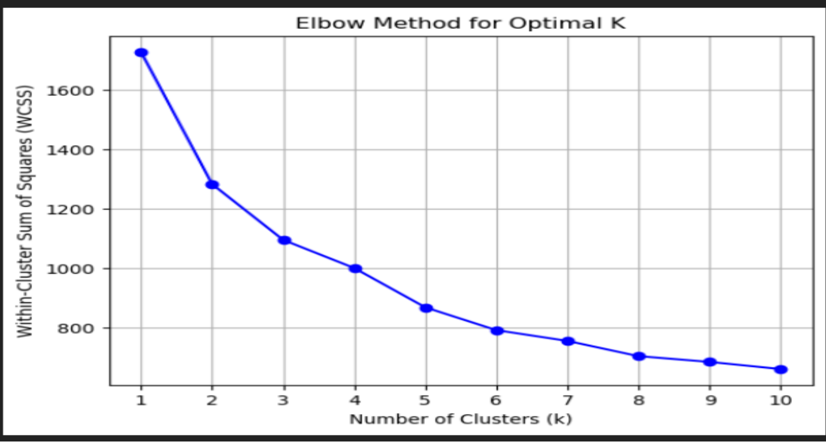
[71]

```
array([[ 0.92922658,  6.25029489],
       [-0.13348634,  3.97621857],
       [-0.16769237, -0.62150109],
       ...,
       [-0.4617274 , -0.58321889],
       [-0.70842128, -0.55271883],
       [-0.48389291, -0.54876773]])
```

Plot the elbow curve

```
wcss = []
for i in range(1, 11):
    kmedoids = KMedoids(n_clusters=i, random_state=0)
    kmedoids.fit(scaled_gross_votes)
    wcss.append(kmedoids.inertia_)

# Plot the elbow curve
plt.plot(range(1, 11), wcss, marker='o', linestyle='-', color='b')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.xticks(np.arange(1, 11, 1))
plt.grid(True)
plt.show()

# The Elbow point is 3 as we see in the graph below
```

[72]



The Elbow point is 3 as we see in the graph below

K-Medoids Silhouette Score of popularity & gross (Accuracy)

```
kmedoids_silhouette_score3 = silhouette_score(scaled_gross_votes, gross_votes_km_labels)
print("K-Medoids Silhouette Score:", kmedoids_silhouette_score3)
```

[ ]

```
K-Medoids Silhouette Score: 0.3787232189550813
```

```python
# Perform k_Medoids on our scaled data ['Gross', 'Votes'] columns

gross_votes_km = KMedoids(n_clusters=3).fit(scaled_gross_votes)

gross_votes_km_labels = gross_votes_km.labels_
print("The cluster labels are:\n", gross_votes_km_labels, '\n')

gross_votes_medoids = gross_votes_km.cluster_centers_
print("The medoids are:\n", gross_votes_medoids)
```
[73]

```
The cluster labels are:
 [0 0 1 ... 1 1 1]

The medoids are:
 [[ 1.63209207  1.08276649]
 [-0.63426261 -0.46581238]
 [ 0.16013822 -0.17155637]]
```

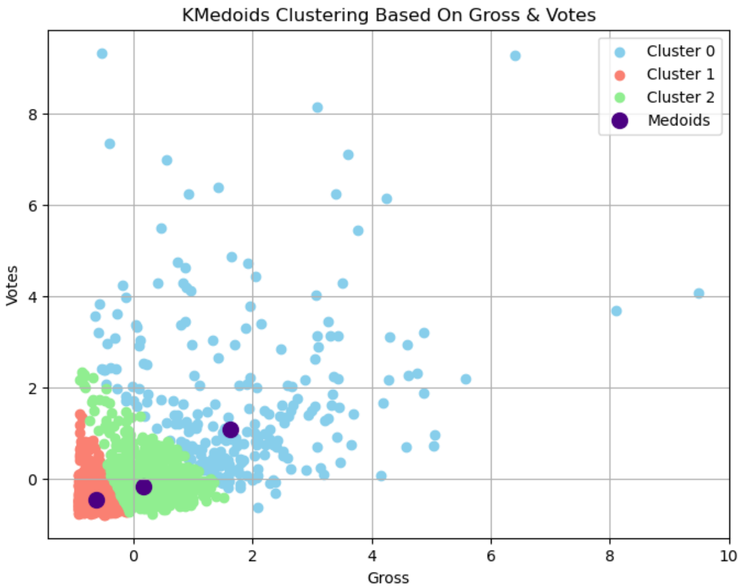
```python
df['gross_votes_clusters'] = gross_votes_km_labels
df
```
[74] Python

	Movie Name	Release Year	Duration	IMDB Rating	Metascore	Votes	Genre	Director	Cast	Gross	Clusters	imdbr_gross_clusters	imdbr_duration_clusters	gross_votes_clusters
0	The Godfather	1972	175	9.2	100.0	2002655.0	Crime, Drama	Francis Ford Coppola	Marlon Brando	134.97	1	2	1	0
1	The Godfather Part II	1974	202	9.0	90.0	1358608.0	Crime, Drama	Francis Ford Coppola	Al Pacino	57.30	1	0	1	0
2	Ordinary People	1980	124	7.7	86.0	56476.0	Drama	Robert Redford	Donald Sutherland	54.80	1	0	1	1
3	Lawrence of Arabia	1962	218	8.3	100.0	313044.0	Adventure, Biography, Drama	David Lean	Peter O'Toole	44.82	1	0	1	2
5	Close Encounters of the Third Kind	1977	138	7.6	90.0	216050.0	Drama, Sci-Fi	Steven Spielberg	Richard Dreyfuss	132.09	1	2	1	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	The Young Victoria	2009	105	7.2	64.0	66235.0	Biography, Drama, History	Jean-Marc Vallée	Emily Blunt	11.00	1	0	2	1
1996	Tooth Fairy	2010	101	5.0	36.0	49527.0	Comedy, Family, Fantasy	Michael Lembeck	Dwayne Johnson	60.02	1	1	0	2
1997	The Informant!	2009	108	6.5	66.0	67318.0	Biography, Comedy, Crime	Steven Soderbergh	Matt Damon	33.31	1	1	0	1
1998	Youth in Revolt	2009	90	6.4	63.0	75956.0	Comedy, Drama, Romance	Miguel Arteta	Michael Cera	15.28	1	1	0	1
1999	Quarantine	2008	89	6.0	53.0	77075.0	Horror, Sci-Fi, Thriller	John Erick Dowdle	Jennifer Carpenter	31.69	1	1	0	1

1870 rows × 14 columns

 # From the following graph we can conclude that: -

```python
plt.figure(figsize=(8, 6))

# Plot Cluster 0
plt.scatter(x = scaled_gross_votes[df['gross_votes_clusters'] == 0, 0], y = scaled_gross_votes[df['gross_votes_clusters'] == 0, 1], c='skyblue', label='Cluster 0')

# Plot Cluster 1
plt.scatter(x = scaled_gross_votes[df['gross_votes_clusters'] == 1,0], y = scaled_gross_votes[df['gross_votes_clusters'] == 1, 1], c='salmon', label='Cluster 1')

# Plot Cluster 2
plt.scatter(x = scaled_gross_votes[df['gross_votes_clusters'] == 2,0], y = scaled_gross_votes[df['gross_votes_clusters'] == 2, 1], c='lightgreen', label='Cluster 2')

# Place our medoids in the graph with a different color than the clusters to distinguish them
plt.scatter(gross_votes_medoids[:, 0], gross_votes_medoids[:, 1], marker='o', c='indigo', s=100, label='Medoids')

plt.title('KMedoids Clustering Based On Gross & Votes')
plt.xlabel('Gross')
plt.ylabel('Votes')

# Legend function to more information about the graph
plt.legend()

plt.grid()
plt.show()
```
[75]



 # high 'Votes' high 'Gross' (cluster 0)

 # low 'Votes' low 'Gross' (cluster 1)

 # (Cluster 2) represent normal movies that does not have high votes not high gross

 # From the information below we can conclude that the 'Gross' is highly related with 'Votes'

❖ Conclusion

```python
# high 'Votes' high 'Gross' (cluster 0)
print('Number of movies cluster 0: ', df.loc[df['gross_votes_clusters'] == 0].shape[0])
print('cluster 0 median votes : ', df.loc[df['gross_votes_clusters'] == 0]['Votes'].median())
print('cluster 0 median gross : ', df.loc[df['gross_votes_clusters'] == 0]['Gross'].median())

print('#------------\n')

# low 'Votes' low 'Gross' (cluster 1)
print('Number of movies cluster 1: ', df.loc[df['gross_votes_clusters'] == 1].shape[0])
print('cluster 1 median votes : ', df.loc[df['gross_votes_clusters'] == 1]['Votes'].median())
print('cluster 1 median gross : ', df.loc[df['gross_votes_clusters'] == 1]['Gross'].median())

print('#------------\n')

# Cluster 2 represent normal movies that does not have high votes nor high gross
print('Number of movies cluster 2: ', df.loc[df['gross_votes_clusters'] == 2].shape[0])
print('cluster 1 median votes : ', df.loc[df['gross_votes_clusters'] == 2]['Votes'].median())
print('cluster 1 median gross : ', df.loc[df['gross_votes_clusters'] == 2]['Gross'].median())

print('#------------\n')

# From the information below we can conclude that the 'Gross' is highly related with 'Votes'
```

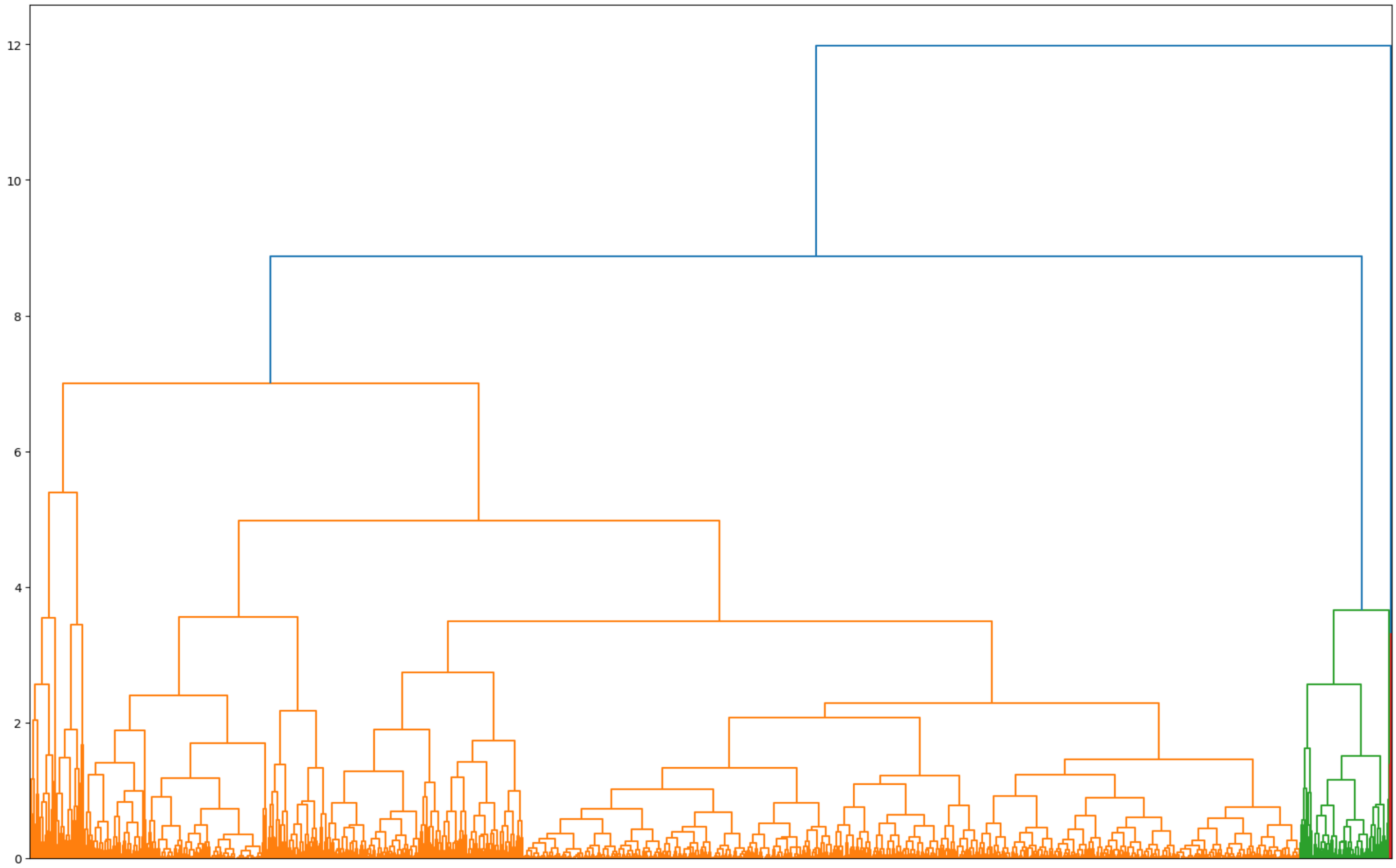
```
Number of movies cluster 0:  250
cluster 0 median votes :  571804.0
cluster 0 median gross :  182.715
#------------

Number of movies cluster 1:  953
cluster 1 median votes :  95068.0
cluster 1 median gross :  20.21
#------------

Number of movies cluster 2:  667
cluster 1 median votes :  181498.0
cluster 1 median gross :  76.57
#------------
```

- ## Hierarchical & KMedoids Comparison

❖ IMDB Rating & Gross

```
[77] imdb_gross_dendo = linkage(scaled_imdb_gross, method='complete', metric='euclidean')
```

```
[78] plt.figure(figsize=(20, 13))
dendrogram(
imdb_gross_dendo,
orientation='top',
labels=labels,
distance_sort='descending',
leaf_font_size=16)
plt.show()
```



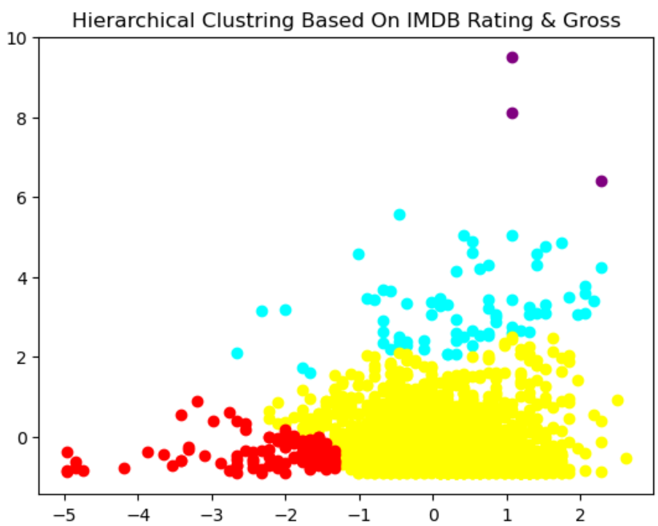
```
hc = AgglomerativeClustering(n_clusters=4, metric='euclidean', linkage='complete')
```

```
scaled_imdb_gross_hc = hc.fit_predict(scaled_imdb_gross)
```

```
scaled_imdb_gross_hc
```

```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

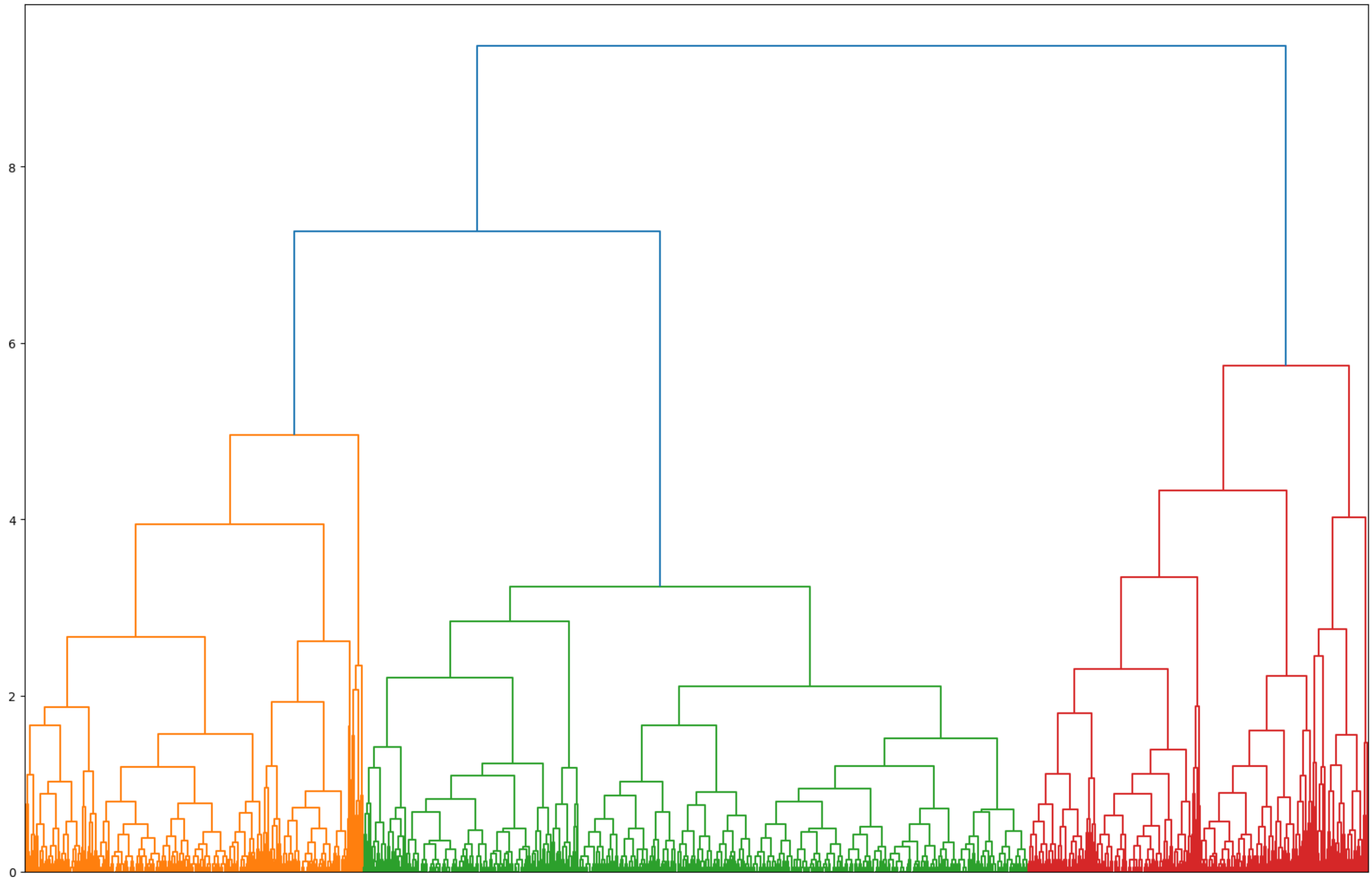
```
plt.scatter(scaled_imdb_gross[scaled_imdb_gross_hc == 0,0],scaled_imdb_gross[scaled_imdb_gross_hc ==0,1], c = 'cyan')
plt.scatter(scaled_imdb_gross[scaled_imdb_gross_hc == 1,0],scaled_imdb_gross[scaled_imdb_gross_hc ==1,1], c= 'yellow')
plt.scatter(scaled_imdb_gross[scaled_imdb_gross_hc == 2,0],scaled_imdb_gross[scaled_imdb_gross_hc ==2,1], c = 'red')
plt.scatter(scaled_imdb_gross[scaled_imdb_gross_hc == 3,0],scaled_imdb_gross[scaled_imdb_gross_hc ==3,1], c = 'purple')
plt.title("Hierarchical Clustring Based On IMDB Rating & Gross")
```



❖ IMDB Rating & Duration

```
[83] imdb_duration_dendo = linkage(scaled_imdbr_duration, method='complete', metric='euclidean')
```

```
[84] plt.figure(figsize=(20, 13))
     dendrogram(
     imdbr_duration_dendo,
     orientation='top',
     labels=labels,
     distance_sort='descending',
     leaf_font_size=16)
     plt.show()
```




```
scaled_imdbr_duration_hc = hc.fit_predict(scaled_imdbr_duration)
```

```
scaled_imdbr_duration_hc
```

```
array([3, 3, 2, ..., 2, 0, 0], dtype=int64)
```

```
plt.scatter(scaled_imdbr_duration[scaled_imdbr_duration_hc == 0,0],scaled_imdbr_duration[scaled_imdbr_duration_hc ==0,1], c = 'cyan')
plt.scatter(scaled_imdbr_duration[scaled_imdbr_duration_hc == 1,0],scaled_imdbr_duration[scaled_imdbr_duration_hc ==1,1], c= 'yellow')
plt.scatter(scaled_imdbr_duration[scaled_imdbr_duration_hc == 2,0],scaled_imdbr_duration[scaled_imdbr_duration_hc ==2,1], c = 'red')
plt.title("Hierarchical Clustring Based On IMDB Rating & Duration")
```