In [ ]:
```
#Netflix Analysis with Recommendation System
```

In [ ]:
```

```

In [1]:
```
#Importing Libraries and Packages
import pandas as pd
import numpy as np
import plotly.graph_objects as go
import plotly.express as px
import plotly.subplots as sp
import plotly.figure_factory as ff
from itertools import cycle
import re
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import string

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

#For Titles
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_colwidth', None)
```

In [ ]:
```

```

In [2]:
```
#Loading the Dataset
df = pd.read_csv(r'C:\Datasets\Recsystem\titles.csv')
credits_df = pd.read_csv(r'C:\Datasets\Recsystem\credits.csv')
```

In [3]:
```
#To see first 2 rows
df.head(2)
```

Out[3]:

	id	title	type	description	release_year	age_certification	runtime	genres	production_countries	seasons	imdb_id	imdb_score	imc
0	ts300399	Five Came Back: The Reference Films	SHOW	This collection includes 12 World War II-era propaganda films — many of which are graphic and offensive — discussed in the docuseries "Five Came Back."	1945	TV-MA	51	['documentation']	['US']	1.0	NaN	NaN	
1	tm84618	Taxi Driver	MOVIE	A mentally unstable Vietnam War veteran works as a night-time taxi driver in New York City where the perceived decadence and sleaze feed his urge for violent action.	1976	R	114	['drama', 'crime']	['US']	NaN	tt0075314	8.2	8

In [4]:
```
#To see datatypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5850 entries, 0 to 5849
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    5850 non-null   object 
 1   title                 5849 non-null   object 
 2   type                  5850 non-null   object 
 3   description           5832 non-null   object 
 4   release_year          5850 non-null   int64  
 5   age_certification     3231 non-null   object 
 6   runtime               5850 non-null   int64  
 7   genres                5850 non-null   object 
 8   production_countries  5850 non-null   object 
 9   seasons               2106 non-null   float64
 10  imdb_id               5447 non-null   object 
 11  imdb_score            5368 non-null   float64
 12  imdb_votes            5352 non-null   float64
 13  tmdb_popularity       5759 non-null   float64
 14  tmdb_score            5539 non-null   float64
dtypes: float64(5), int64(2), object(8)
memory usage: 685.7+ KB
```

In [5]:
```
#Descriptive statistics
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
release_year	5850.0	2016.417094	6.937726	1945.000000	2016.0000	2018.000	2020.0000	2022.000
runtime	5850.0	76.888889	39.002509	0.000000	44.0000	83.000	104.0000	240.000
seasons	2106.0	2.162868	2.689041	1.000000	1.0000	1.000	2.0000	42.000
imdb_score	5368.0	6.510861	1.163826	1.500000	5.8000	6.600	7.3000	9.600
imdb_votes	5352.0	23439.382474	95820.470909	5.000000	516.7500	2233.500	9494.0000	2294231.000
tmdb_popularity	5759.0	22.637925	81.680263	0.009442	2.7285	6.821	16.5900	2274.044
tmdb_score	5539.0	6.829175	1.170391	0.500000	6.1000	6.900	7.5375	10.000

In [6]:
```
#Checking Missing values
pd.DataFrame(df.isna().sum()).T
```

Out[6]:

	id	title	type	description	release_year	age_certification	runtime	genres	production_countries	seasons	imdb_id	imdb_score	imdb_votes	tmdb_populari
0	0	1	0	18	0	2619	0	0	0	3744	403	482	498	9

In [7]:
```
#Dropping the row with missing title
df.drop(df[df["title"].isna()].index, inplace=True)
```

In [8]:
```
#Checking duplicate rows and adding release year
duplicate_names_idx = df[df["title"].duplicated(keep=False)].sort_values(by="title")["title"].index
duplicate_names = df.loc[duplicate_names_idx, "title"].values
print(duplicate_names)
```

```
['A Lion in the House' 'A Lion in the House' 'A Love So Beautiful'
 'A Love So Beautiful' 'A Nightmare on Elm Street'
 'A Nightmare on Elm Street' 'A Second Chance' 'A Second Chance'
 'Always Be My Maybe' 'Always Be My Maybe' 'Black' 'Black' 'Bodyguard'
 'Bodyguard' 'Cargo' 'Cargo' 'Chosen' 'Chosen' 'Christine' 'Christine'
 'Cloudy with a Chance of Meatballs' 'Cloudy with a Chance of Meatballs'
 'Connected' 'Connected' 'Connected' 'Cowboy Bebop' 'Cowboy Bebop'
 'Danger Mouse' 'Danger Mouse' 'Don' 'Don' 'Dostana' 'Dostana' 'Fearless'
 'Fearless' 'Into the Wind' 'Into the Wind' 'Johnny Test' 'Johnny Test'
 'Kakegurui' 'Kakegurui' 'Love' 'Love' 'Love O2O' 'Love O2O' 'Ludo' 'Ludo'
 'Manhunt' 'Manhunt' 'Monster' 'Monster' 'Queen' 'Queen' 'Security'
 'Security' 'Sergio' 'Sergio' "She's Gotta Have It" "She's Gotta Have It"
 'Skylines' 'Skylines' 'Taxi Driver' 'Taxi Driver' 'The Call' 'The Call'
 'The Club' 'The Club' 'The Forest' 'The Forest' 'The Gift' 'The Gift'
 'The Gift' 'The Girl Next Door' 'The Girl Next Door' 'The Good Cop'
 'The Good Cop' 'The Heirs' 'The Heirs' 'The Land' 'The Land' 'The Motive'
 'The Motive' 'The One' 'The One' 'The Platform' 'The Platform'
 'Till Death' 'Till Death' 'Time Out' 'Time Out' 'Top Boy' 'Top Boy'
 'Vampires' 'Vampires' 'Wanted' 'Wanted' 'Whispers' 'Whispers' 'Zero'
 'Zero']
```

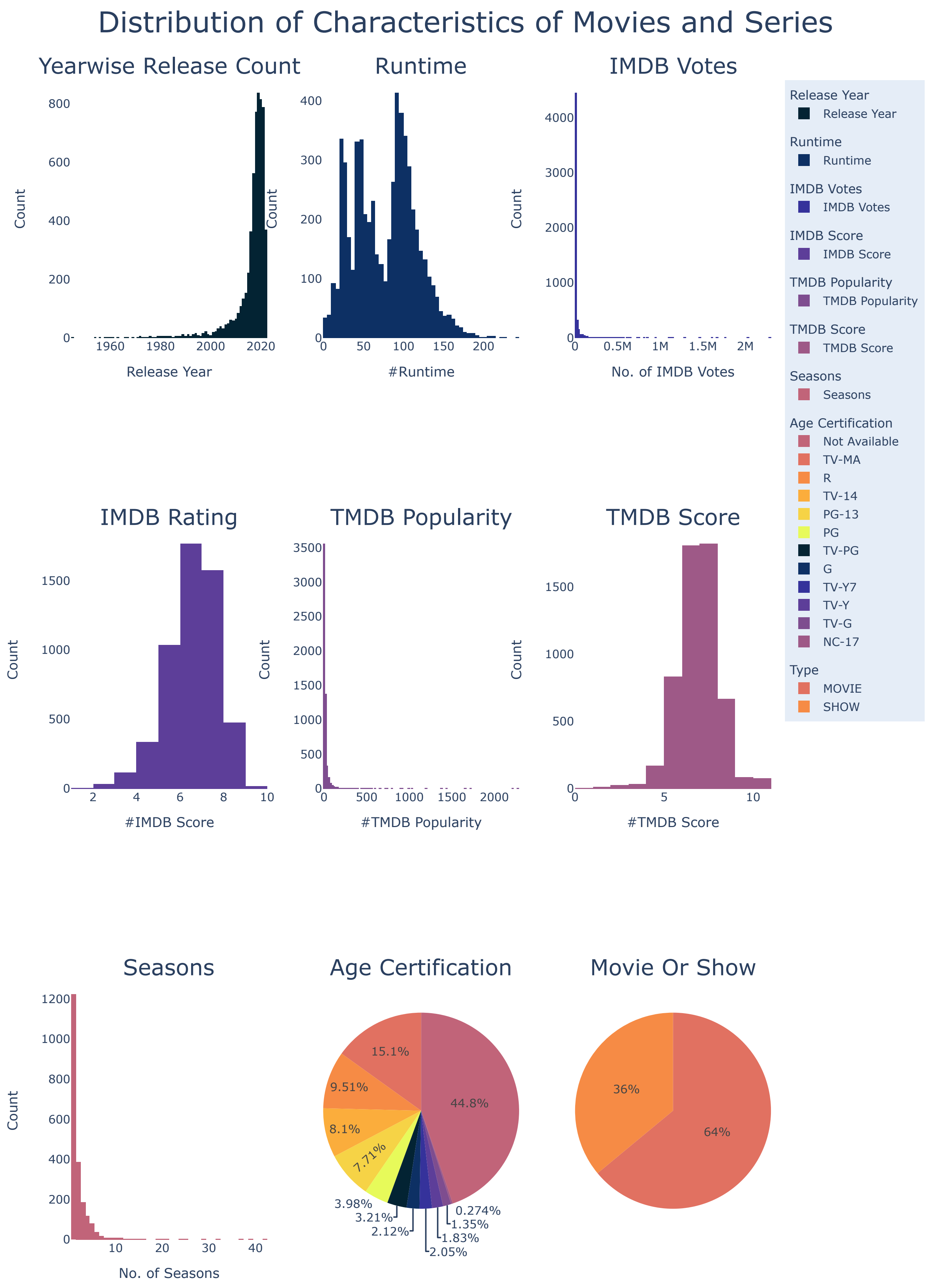
In [9]:
```
def title_release_year(x):
        return x.title + " (" + str(x.release_year) + ")"

df.loc[duplicate_names_idx, "title"] = df.loc[duplicate_names_idx].apply(title_release_year, axis=1)
```

In [ ]:
```

```

In [10]:
```
#EDA
```

In [11]:
```
#Dashboard 1
```

In [12]:
```
palette = cycle(px.colors.sequential.thermal)

fig = sp.make_subplots(
    rows=3,
    cols=3,
    horizontal_spacing=0.08,
    subplot_titles=[
        "Yearwise Release Count",
        "Runtime",
        "IMDB Votes",
        "IMDB Rating",
        "TMDB Popularity",
        "TMDB Score",
        "Seasons",
        "Age Certification",
        "Movie Or Show"],
    specs=[[{"type": "histogram"}, {"type": "histogram"}, {"type": "histogram"}],
           [{"type": "histogram"}, {"type": "histogram"}, {"type": "histogram"}],
           [{"type": "histogram"}, {"type": "pie"}, {"type": "pie"}]]
)
release_year = go.Histogram(
    x=df.release_year,
    name="Release Year",
    marker_color=next(palette),
    legendgroup="Release Year",
    legendgrouptitle_text="Release Year",
)

runtime = go.Histogram(
    x=df.runtime,
    nbinsx=int(df.__len__()/50),
    name="Runtime",
    marker_color=next(palette),
    legendgroup="Runtime",
    legendgrouptitle_text="Runtime",
)

imdb_votes = go.Histogram(
    x=df.imdb_votes,
    nbinsx=int(df.__len__()/50),
    name="IMDB Votes",
    marker_color=next(palette),
    legendgroup="IMDB Votes",
    legendgrouptitle_text="IMDB Votes",
)

imdb_score = go.Histogram(
    x=df.imdb_score,
    nbinsx=10,
    name="IMDB Score",
    marker_color=next(palette),
    legendgroup="IMDB Score",
    legendgrouptitle_text="IMDB Score",
)

tmdb_popularity = go.Histogram(
    x=df.tmdb_popularity,
    name="TMDB Popularity",
    nbinsx=int(df.__len__()/50),
    marker_color=next(palette),
    legendgroup="TMDB Popularity",
    legendgrouptitle_text="TMDB Popularity",
)

tmdb_score = go.Histogram(
    x=df.tmdb_score,
    name="TMDB Score",
    nbinsx=10,
    marker_color=next(palette),
    legendgroup="TMDB Score",
    legendgrouptitle_text="TMDB Score",
)

seasons = go.Histogram(
    x=df.seasons,
    name="Seasons",
    marker_color=next(palette),
    legendgroup="Seasons",
    legendgrouptitle_text="Seasons",
)

age_certification_counts = df.age_certification.value_counts()
age_certification_counts["Not Available"] = df.age_certification.isna().sum()
age_certification_dict = age_certification_counts.to_dict()

age_certification = go.Pie(
    labels=list(age_certification_dict.keys()),
    values=list(age_certification_dict.values()),
    name="Age Certification",
    hoverinfo="label+value+percent",
    marker_colors=[next(palette) for i in range(len(age_certification_dict))],
    legendgroup="Age Certification",
    legendgrouptitle_text="Age Certification",
)

type_counts = df.type.value_counts().to_dict()

type_ = go.Pie(
    labels=list(type_counts.keys()),
    values=list(type_counts.values()),
    name="Type",
    hoverinfo="label+value+percent",
    marker_colors=[next(palette) for i in range(len(type_counts))],
    legendgroup="Type",
    legendgrouptitle_text="Type",
)

fig.add_trace(release_year, row=1, col=1)
fig.update_xaxes(title_text="Release Year", row=1, col=1)
fig.update_yaxes(title_text="Count", row=1, col=1)

fig.add_trace(runtime, row=1, col=2)
fig.update_xaxes(title_text="#Runtime", row=1, col=2)
fig.update_yaxes(title_text="Count", row=1, col=2)

fig.add_trace(imdb_votes, row=1, col=3)
fig.update_xaxes(title_text="No. of IMDB Votes", row=1, col=3)
fig.update_yaxes(title_text="Count", row=1, col=3)

fig.add_trace(imdb_score, row=2, col=1)
fig.update_xaxes(title_text="#IMDB Score", row=2, col=1)
fig.update_yaxes(title_text="Count", row=2, col=1)

fig.add_trace(tmdb_popularity, row=2, col=2)
fig.update_xaxes(title_text="#TMDB Popularity", row=2, col=2)
fig.update_yaxes(title_text="Count", row=2, col=2)

fig.add_trace(tmdb_score, row=2, col=3)
fig.update_xaxes(title_text="#TMDB Score", row=2, col=3)
fig.update_yaxes(title_text="Count", row=2, col=3)

fig.add_trace(seasons, row=3, col=1)
fig.update_xaxes(title_text="No. of Seasons", row=3, col=1)
fig.update_yaxes(title_text="Count", row=3, col=1)

fig.add_trace(age_certification, row=3, col=2)

fig.add_trace(type_, row=3, col=3)

fig.update_annotations(font_size=23)

fig.update_layout(
    template="plotly",
    height=1400,
)

fig.update(
    layout_title_text="Distribution of Characteristics of Movies and Series",
    layout_title_font_size=30,
    layout_title_x=0.5,
    layout_paper_bgcolor='rgb(229, 237, 247)',
    layout_plot_bgcolor='rgb(229, 237, 247)',
)

fig.show()
```


Distribution of Characteristics of Movies and Series
Yearwise Release Count
Runtime
IMDB Votes
IMDB Rating
TMDB Popularity
TMDB Score
Seasons
Age Certification
Movie Or Show
Count
Release Year
#Runtime
No. of IMDB Votes
#IMDB Score
#TMDB Popularity
#TMDB Score
No. of Seasons
15.1%
9.51%
8.1%
7.71%
3.98%
3.21%
2.12%
2.05%
1.83%
1.35%
0.274%
44.8%
36%
64%
Release Year
Runtime
IMDB Votes
IMDB Score
TMDB Popularity
TMDB Score
Seasons
Age Certification
Not Available
TV-MA
R
TV-14
PG-13
PG
TV-PG
G
TV-Y7
TV-Y
TV-G
NC-17
Type
MOVIE
SHOW

In [ ]:

```
```

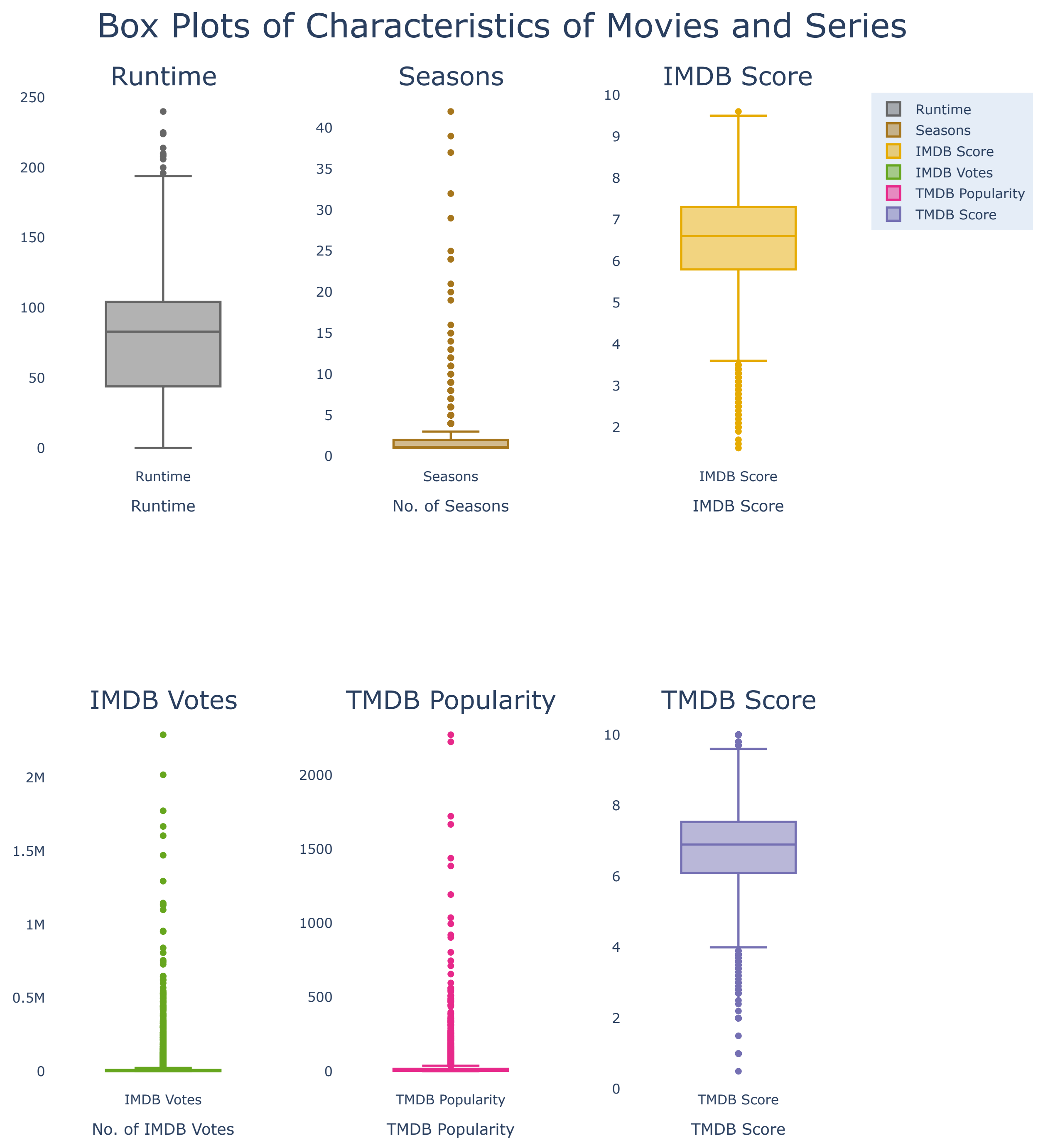
In [13]:

```
#Dashboard 2
```

In [14]:

```
#Boxplot of all the features
```

In [15]:
```
palette = cycle(px.colors.qualitative.Dark2_r)

fig = sp.make_subplots(
    rows=2, cols=3,
    subplot_titles=["Runtime",
                    "Seasons",
                    "IMDB Score",
                    "IMDB Votes",
                    "TMDB Popularity",
                    "TMDB Score",],
    specs=[[{"type": "box"}, {"type": "box"}, {"type": "box"}],
           [{"type": "box"}, {"type": "box"}, {"type": "box"}]],
)

runtime_box = go.Box(
    y=df.runtime,
    name="Runtime",
    marker_color=next(palette),
)

seasons_box = go.Box(
    y=df.seasons,
    name="Seasons",
    marker_color=next(palette),
)

imdb_score_box = go.Box(
    y=df.imdb_score,
    name="IMDB Score",
    marker_color=next(palette),
)

imdb_votes_box = go.Box(
    y=df.imdb_votes,
    name="IMDB Votes",
    marker_color=next(palette),
)

tmdb_popularity_box = go.Box(
    y=df.tmdb_popularity,
    name="TMDB Popularity",
    marker_color=next(palette),
)

tmdb_score_box = go.Box(
    y=df.tmdb_score,
    name="TMDB Score",
    marker_color=next(palette),
)

fig.add_trace(runtime_box, row=1, col=1)
fig.update_xaxes(title_text="Runtime", row=1, col=1)

fig.add_trace(seasons_box, row=1, col=2)
fig.update_xaxes(title_text="No. of Seasons", row=1, col=2)

fig.add_trace(imdb_score_box, row=1, col=3)
fig.update_xaxes(title_text="IMDB Score", row=1, col=3)

fig.add_trace(imdb_votes_box, row=2, col=1)
fig.update_xaxes(title_text="No. of IMDB Votes", row=2, col=1)

fig.add_trace(tmdb_popularity_box, row=2, col=2)
fig.update_xaxes(title_text="TMDB Popularity", row=2, col=2)

fig.add_trace(tmdb_score_box, row=2, col=3)
fig.update_xaxes(title_text="TMDB Score", row=2, col=3)

fig.update_layout(template="plotly", height=1080,)
fig.update_annotations(font_size=23)

fig.update(
    layout_title_text="Box Plots of Characteristics of Movies and Series",
    layout_title_font_size=30,
    layout_title_x=0.5,
    layout_paper_bgcolor='rgb(229, 237, 247)',
    layout_plot_bgcolor='rgb(229, 237, 247)',
)

fig.show()
```



In [ ]:

In [16]:
```
#Feature Generation
```

In [17]:
```
df["genres"] = df["genres"].apply(lambda x: re.findall("\w+", x))

genres = list(df["genres"].values)
genres = list(set([item for sublist in genres for item in sublist]))

for i, genre in enumerate(genres):
    df[genre] = df.genres.apply(lambda x: 1 if genre in x else 0).astype(int)

print("Number of Genres: ", len(genres))
print("Genres:", genres)
```

```
Number of Genres:  19
Genres: ['romance', 'comedy', 'war', 'thriller', 'western', 'music', 'fantasy', 'sport', 'action', 'reality', 'scifi', 'drama',
'horror', 'family', 'crime', 'history', 'documentation', 'european', 'animation']
```

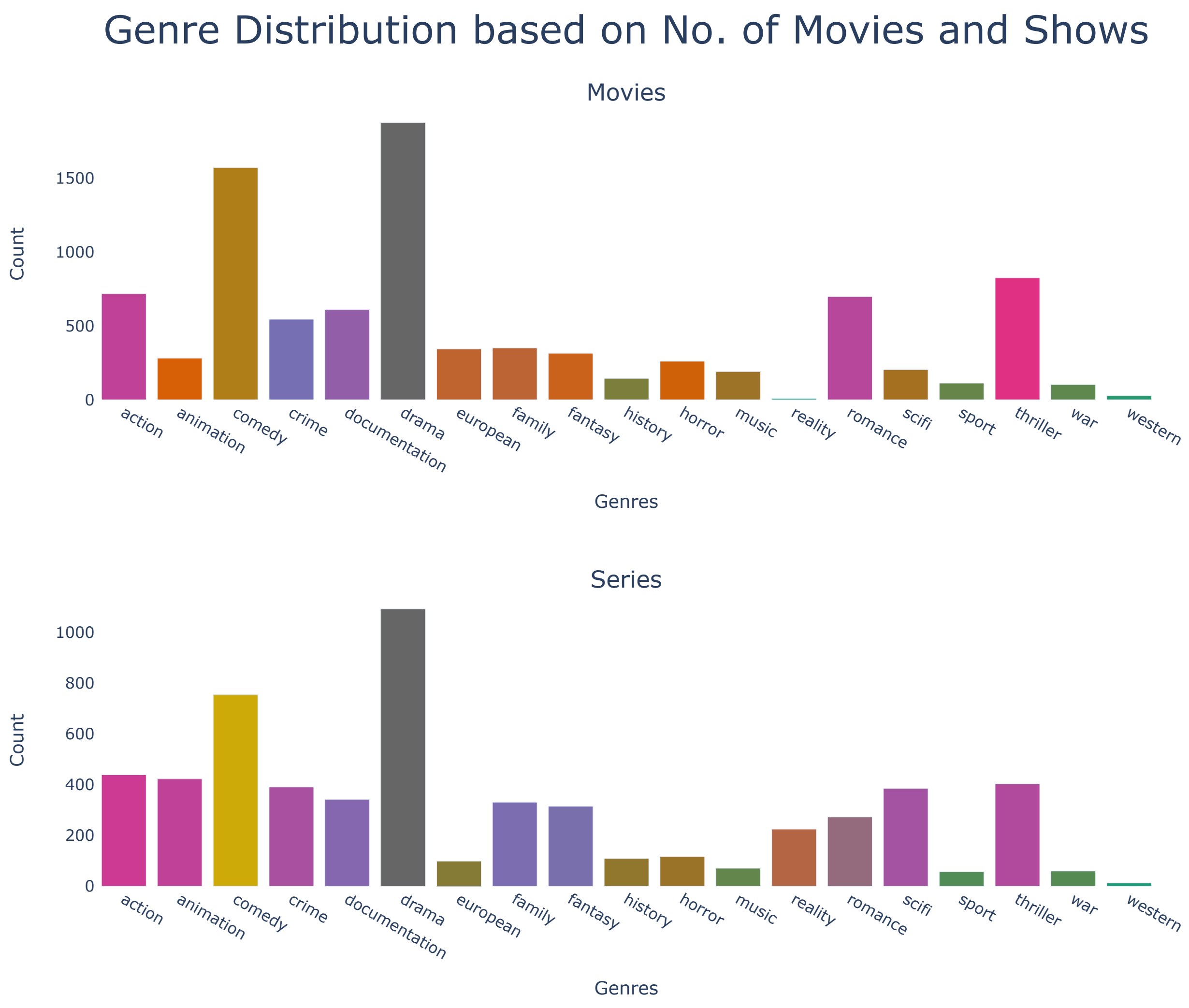
In [ ]:

```
```

In [18]:

```
#Genre Distribution based on count
```

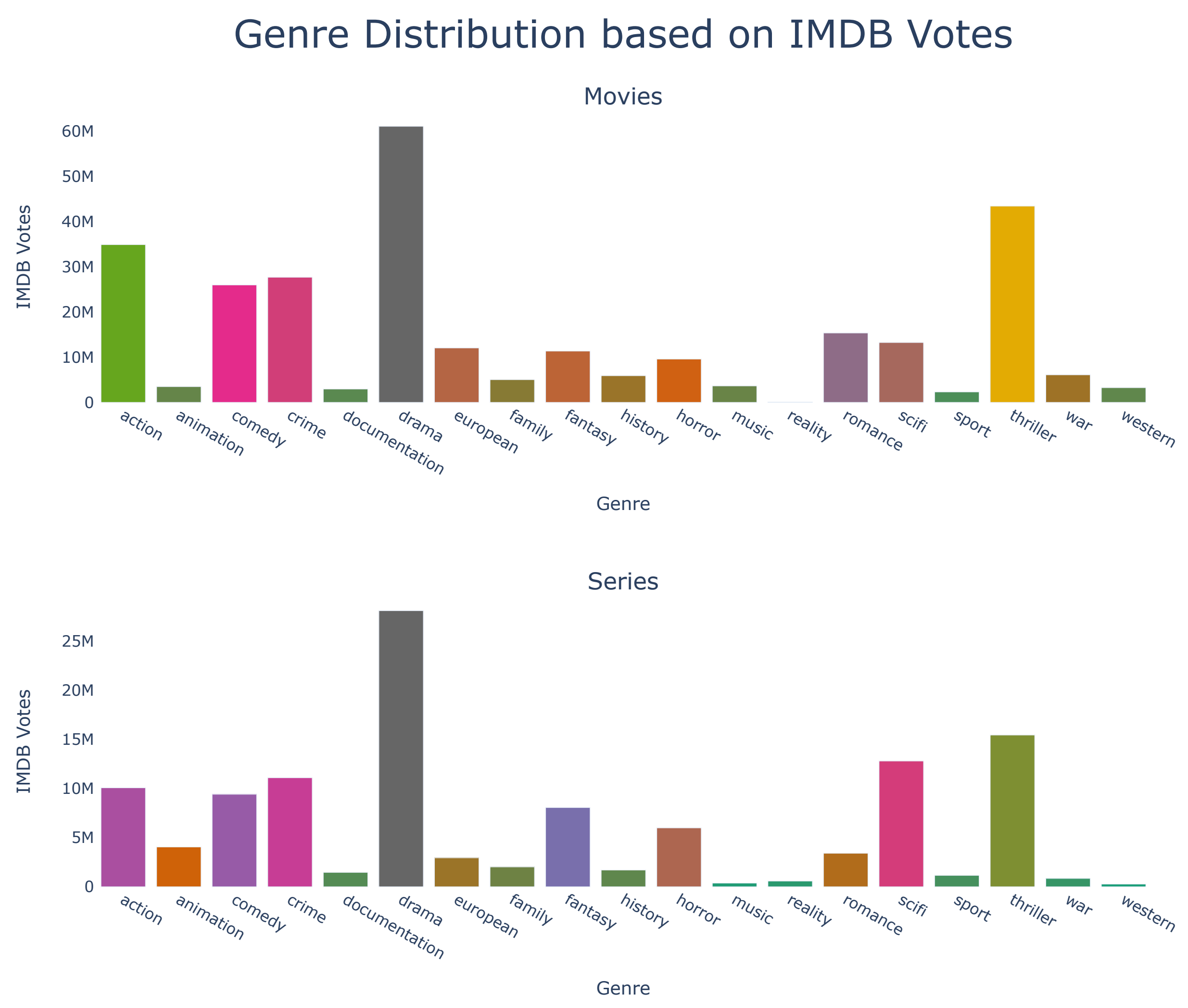
In [19]:
```
genre_movie_dict = {}

for genre in genres:
    genre_movie_dict[genre] = df.query("type == 'MOVIE'")[genre].sum()

genre_movie_dict = dict(sorted(genre_movie_dict.items(), key=lambda x: x[0]))

genre_series_dict = {}

for genre in genres:
    genre_series_dict[genre] = df.query("type == 'SHOW'")[genre].sum()

genre_series_dict = dict(sorted(genre_series_dict.items(), key=lambda x: x[0]))

fig = sp.make_subplots(
    rows=2,
    cols=1,
    subplot_titles=["Movies", "Series"],
)

genre_movie_count = go.Bar(
    x=list(genre_movie_dict.keys()),
    y=list(genre_movie_dict.values()),
    marker=dict(color=list(genre_movie_dict.values()),
                colorscale=px.colors.qualitative.Dark2),
    name="Movies",
)

genre_series_count = go.Bar(
    x=list(genre_series_dict.keys()),
    y=list(genre_series_dict.values()),
    marker=dict(color=list(genre_series_dict.values()),
                colorscale=px.colors.qualitative.Dark2),
    name="Series",
)

fig.add_trace(genre_movie_count, row=1, col=1)
fig.update_xaxes(title_text="Genres", row=1, col=1)
fig.update_yaxes(title_text="Count", row=1, col=1)

fig.add_trace(genre_series_count, row=2, col=1)
fig.update_xaxes(title_text="Genres", row=2, col=1)
fig.update_yaxes(title_text="Count", row=2, col=1)

fig.update(
    layout_title_text="Genre Distribution based on No. of Movies and Shows",
    layout_title_font_size=30,
    layout_title_x=0.5,
    layout_template="plotly",
    layout_showlegend=False,
    layout_height=800,
    layout_paper_bgcolor='rgb(229, 237, 247)',
    layout_plot_bgcolor='rgb(229, 237, 247)',
)

fig.update_annotations(font_size=18)

fig.show()
```



```
In [ ]:
```

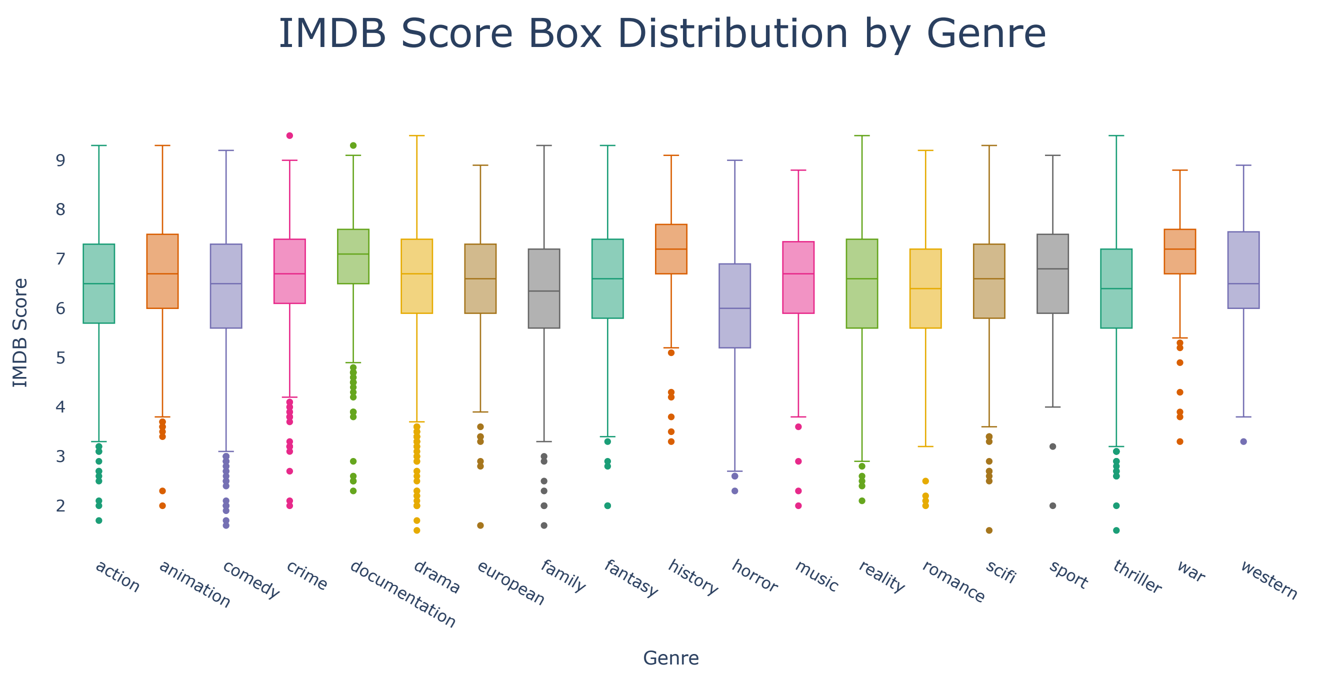
```
In [20]: #Based on IMDB Votes
```

In [21]:
```
genre_movies_popularity_dict = {}

for i, genre in enumerate(genres):
    genre_movies_popularity_dict[genre] = df.query("type == 'MOVIE'").groupby(genre)["imdb_votes"].sum().sort_index().__getitem_

genre_movies_popularity_dict = dict(sorted(genre_movies_popularity_dict.items(), key=lambda x: x[0]))

genre_series_popularity_dict = {}

for i, genre in enumerate(genres):
    genre_series_popularity_dict[genre] = df.query("type == 'SHOW'").groupby(genre)["imdb_votes"].sum().sort_index().__getitem__(

    genre_series_popularity_dict = dict(sorted(genre_series_popularity_dict.items(), key=lambda x: x[0]))

fig = sp.make_subplots(
    rows=2,
    cols=1,
    subplot_titles=["Movies", "Series"],
)

genre_movies_pop = go.Bar(
    x=list(genre_movies_popularity_dict.keys()),
    y=list(genre_movies_popularity_dict.values()),
    marker=dict(color=list(genre_movies_popularity_dict.values()),
                colorscale=px.colors.qualitative.Dark2),
    hoverinfo="x+y",
)

genre_series_pop = go.Bar(
    x=list(genre_series_popularity_dict.keys()),
    y=list(genre_series_popularity_dict.values()),
    marker=dict(color=list(genre_series_popularity_dict.values()),
                colorscale=px.colors.qualitative.Dark2),
    hoverinfo="x+y",
)

fig.add_trace(genre_movies_pop, row=1, col=1)
fig.update_xaxes(title_text="Genre", row=1, col=1)
fig.update_yaxes(title_text="IMDB Votes", row=1, col=1)
fig.update

fig.add_trace(genre_series_pop, row=2, col=1)
fig.update_xaxes(title_text="Genre", row=2, col=1)
fig.update_yaxes(title_text="IMDB Votes", row=2, col=1)

fig.update(
    layout_title_text="Genre Distribution based on IMDB Votes",
    layout_title_font_size=30,
    layout_title_x=0.5,
    layout_template="plotly",
    layout_showlegend=False,
    layout_height=800,
    layout_paper_bgcolor='rgb(229, 237, 247)',
    layout_plot_bgcolor='rgb(229, 237, 247)',
)

fig.update_annotations(font_size=18)

fig.show()
```

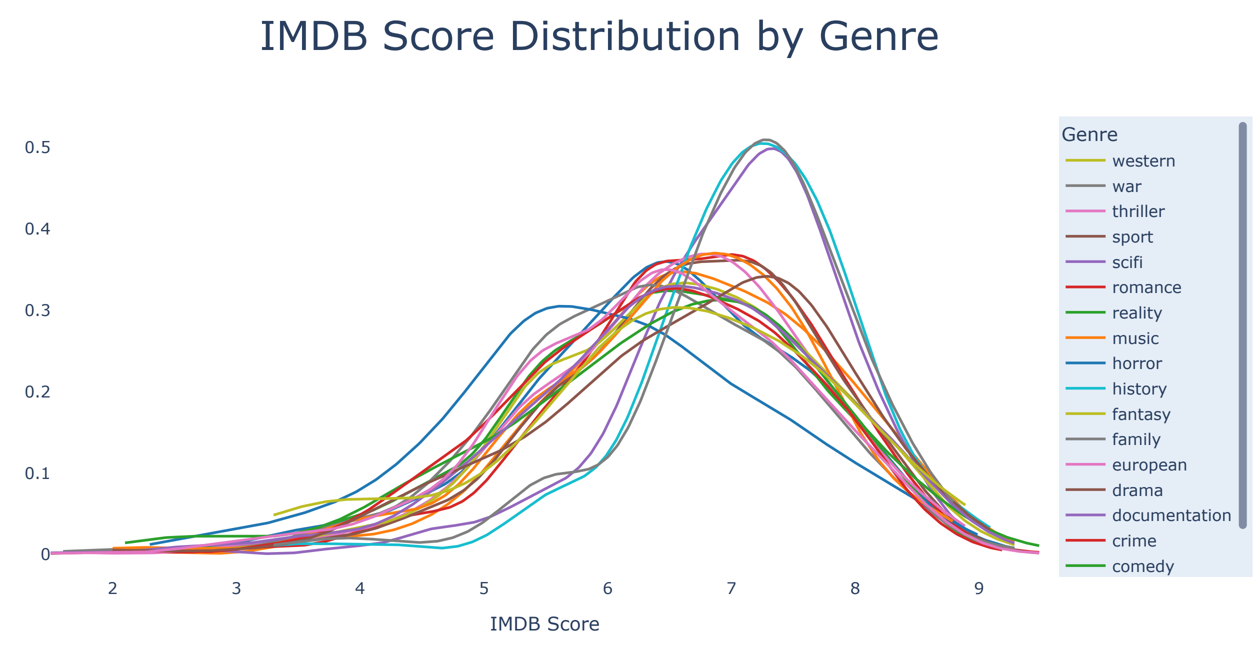

Genre Distribution based on IMDB Votes
Movies
IMDB Votes
60M
50M
40M
30M
20M
10M
0
action
animation
comedy
crime
documentation
drama
european
family
fantasy
history
horror
music
reality
romance
scifi
sport
thriller
war
western
Genre
Series
IMDB Votes
25M
20M
15M
10M
5M
0
Genre

```
In [ ]:
```

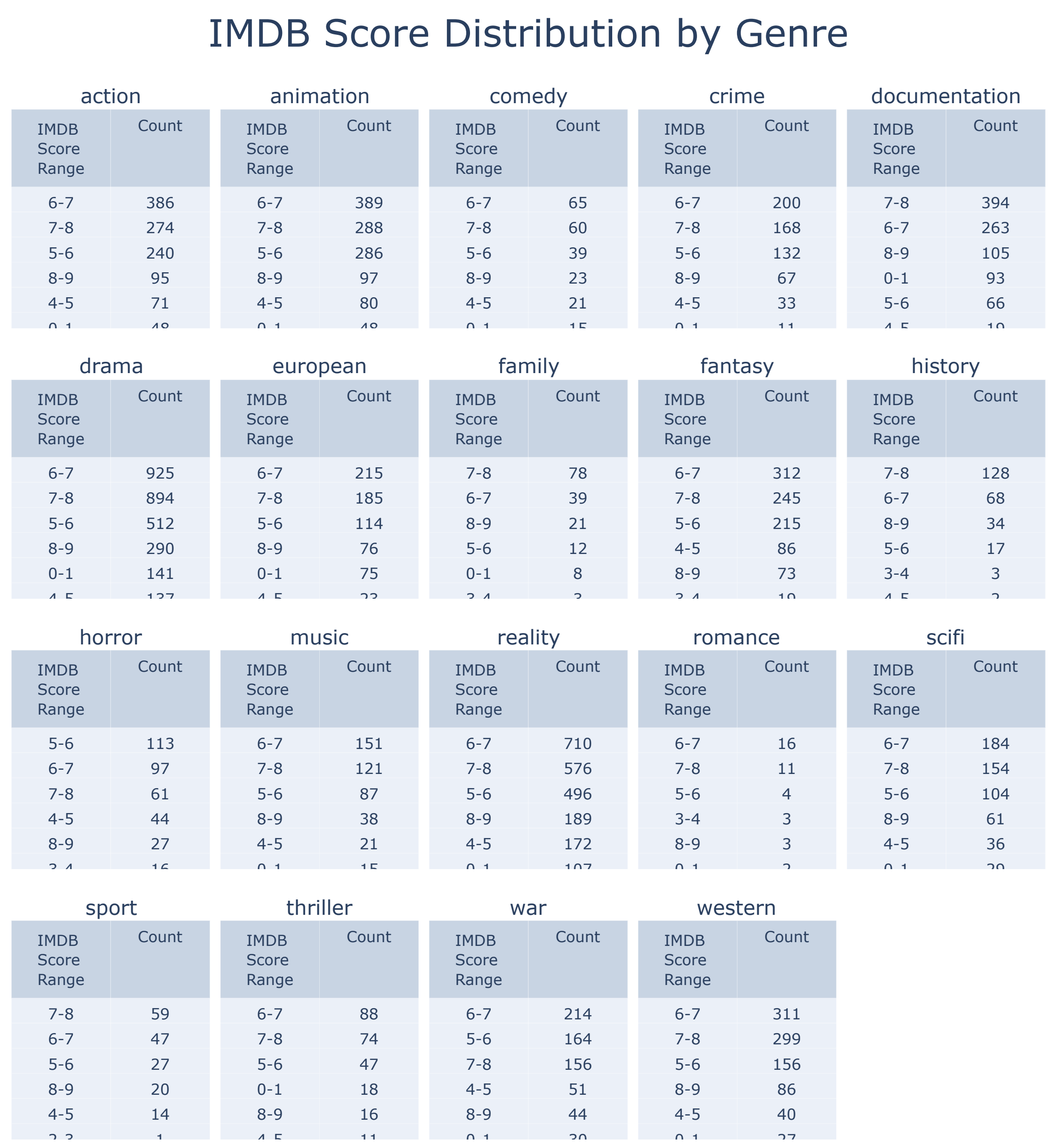
In [22]:
```
#IMDB Score Boxplot by Genre
palette = cycle(px.colors.qualitative.Dark2)

fig = go.Figure()

for i, genre in enumerate(sorted(genres)):
    temp = df[df[genre] == 1]

    fig.add_trace(
        go.Box(
            y=temp['imdb_score'],
            name=genre,
            marker_color=next(palette),
            marker_size=5,
            line_width=1,
            hovertemplate="<b>%{y:.2f}</b>"+f"<br>{genre}<br>Count-{len(temp)}<extra></extra>",
        )
    )

    fig.update_layout(
    title="IMDB Score Box Distribution by Genre",
    title_font_size=30,
    title_x=0.5,
    yaxis_title="IMDB Score",
    xaxis_title="Genre",
    template="plotly",
    margin=dict(
        l=40,
        r=30,
        b=80,
        t=100,
    ),
    showlegend=False,
    paper_bgcolor='rgb(229, 237, 247)',
    plot_bgcolor='rgb(229, 237, 247)',
)

fig.show()
```



In [23]:
```
#KDE Plot for Genre
fig = ff.create_distplot(
    [df[(df[genre] == 1) & (df['imdb_score'].notna())]['imdb_score'] for genre in sorted(genres)],
    sorted(genres),
    show_hist=False,
    show_rug=False,
)

fig.update_layout(
    title="IMDB Score Distribution by Genre",
    title_font_size=30,
    title_x=0.5,
    xaxis_title="IMDB Score",
    template="plotly",
    paper_bgcolor='rgb(229, 237, 247)',
    plot_bgcolor='rgb(229, 237, 247)',
    legend_title="Genre",
)

fig.show()
```



In [24]:
```
#IMDB Score Category
distrib = df['imdb_score'].map(lambda x: f"{int(np.nan_to_num(x)*10//10)}-{int((np.nan_to_num(x)*10//10)+1)}")
df.insert(12, 'imdb_score_range', distrib)

score_range_dict = {}

for i, genre in enumerate(sorted(genres)):
    score_range_dict[genre] = df.groupby(genre)['imdb_score_range'].value_counts().__getitem__(1).to_dict()

    fig = sp.make_subplots(
    rows=4,
    cols=5,
    subplot_titles=sorted(genres),
    specs=[[{'type': 'table'}]*5]*4,
    horizontal_spacing=0.01,
    vertical_spacing=0.05,
)

for i, (key, value) in enumerate(score_range_dict.items()):
    fig.append_trace(
        go.Table(
            header=dict(
                values=["IMDB Score Range", "Count"],
                align="center",
            ),
            cells=dict(
                values=[list(value.keys()), list(value.values())],
               align="center",
            )
        ),
        row=i%4+1,
        col=i%5+1,
    )

fig.update_layout(
    title_text="IMDB Score Distribution by Genre",
    title_font_size=30,
    title_x=0.5,
    height=1000,
    autosize=True,
)
fig.show()
```



IMDB Score Distribution by Genre

action

IMDB Score Range	Count
6-7	386
7-8	274
5-6	240
8-9	95
4-5	71

animation

IMDB Score Range	Count
6-7	389
7-8	288
5-6	286
8-9	97
4-5	80

comedy

IMDB Score Range	Count
6-7	65
7-8	60
5-6	39
8-9	23
4-5	21

crime

IMDB Score Range	Count
6-7	200
7-8	168
5-6	132
8-9	67
4-5	33

documentation

IMDB Score Range	Count
7-8	394
6-7	263
8-9	105
0-1	93
5-6	66

drama

IMDB Score Range	Count
6-7	925
7-8	894
5-6	512
8-9	290
0-1	141

european

IMDB Score Range	Count
6-7	215
7-8	185
5-6	114
8-9	76
0-1	75

family

IMDB Score Range	Count
7-8	78
6-7	39
8-9	21
5-6	12
0-1	8

fantasy

IMDB Score Range	Count
6-7	312
7-8	245
5-6	215
4-5	86
8-9	73

history

IMDB Score Range	Count
7-8	128
6-7	68
8-9	34
5-6	17
3-4	3

horror

IMDB Score Range	Count
5-6	113
6-7	97
7-8	61
4-5	44
8-9	27

music

IMDB Score Range	Count
6-7	151
7-8	121
5-6	87
8-9	38
4-5	21

reality

IMDB Score Range	Count
6-7	710
7-8	576
5-6	496
8-9	189
4-5	172

romance

IMDB Score Range	Count
6-7	16
7-8	11
5-6	4
3-4	3
8-9	3

scifi

IMDB Score Range	Count
6-7	184
7-8	154
5-6	104
8-9	61
4-5	36

sport

IMDB Score Range	Count
7-8	59
6-7	47
5-6	27
8-9	20
4-5	14

thriller

IMDB Score Range	Count
6-7	88
7-8	74
5-6	47
0-1	18
8-9	16

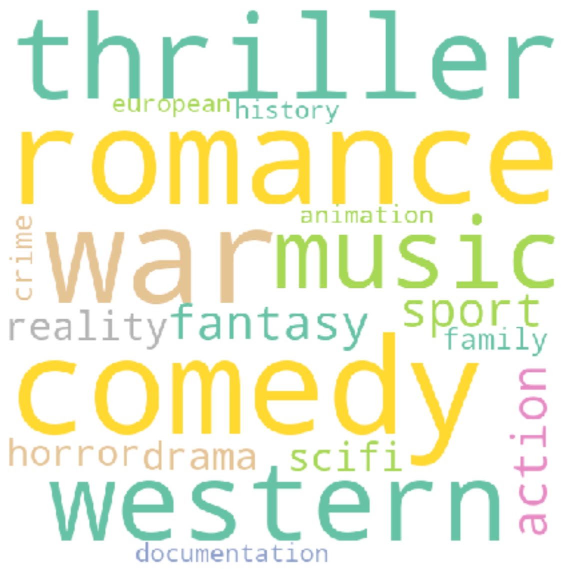
war

IMDB Score Range	Count
6-7	214
5-6	164
7-8	156
4-5	51
8-9	44

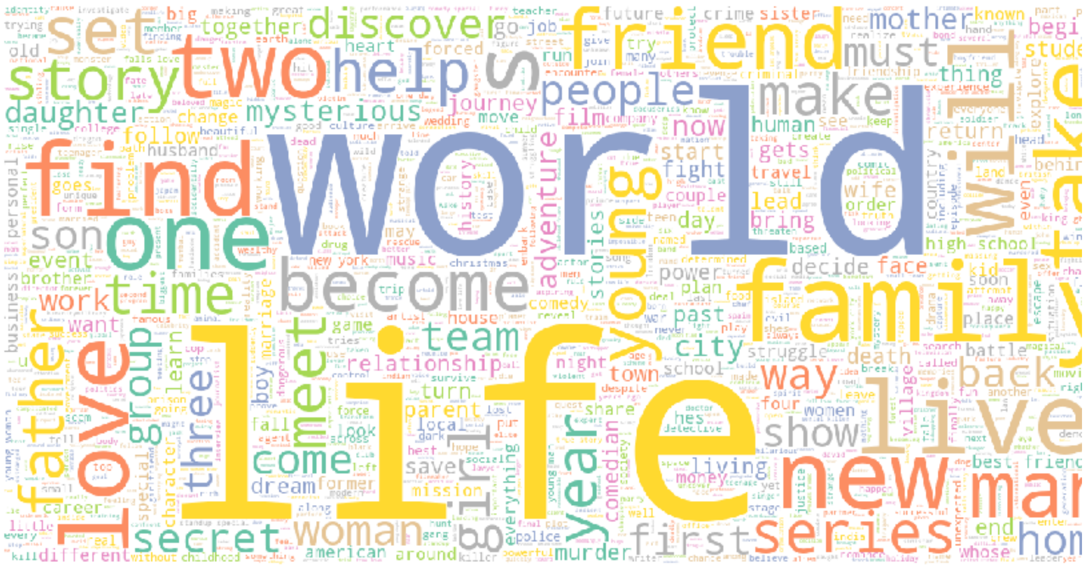
western

IMDB Score Range	Count
6-7	311
7-8	299
5-6	156
8-9	86
4-5	40

In [25]:
```
#World cloud of Genres
word_cloud =  WordCloud(
    width=600,
    height=600,
    max_words=1000000,
    background_color="white",
    colormap="Set2",

).generate(str(" ".join(genres)))

fig = go.Figure()
fig.update_layout(
    width=600,
    height=600,
    xaxis_showticklabels=False,
    yaxis_showticklabels=False,
)
fig.add_trace(go.Image(z=word_cloud))
fig.show()
```



```
In [26]: #Worldcloud of description
         all_descriptions = " ".join(df["description"].astype(str).to_list()).lower()
         all_desc_nopunct = all_descriptions.translate(str.maketrans("", "", string.punctuation)).replace("—", "").replace("-", "").split(
         all_desc_clean = " ".join(
             [word for word in all_desc_nopunct if word not in list(STOPWORDS)]
         )

         word_cloud = WordCloud(
             width=1200,
             height=600,
             max_words=1000000,
             background_color="white",
             colormap="Set2",
         ).generate(all_desc_clean)

         fig = go.Figure()
         fig.update_layout(
             width=1200,
             height=600,
             xaxis_showticklabels=False,
             yaxis_showticklabels=False,
         )
         fig.add_trace(go.Image(z=word_cloud))
         fig.show()
```



In [27]:
```
#Top 5 Movies with High IMDB votes and score
df.query("type == 'MOVIE'")[['release_year',
                             'title',
                             'type',
                             'runtime',
                             'imdb_score',
                             'imdb_votes',
                             'genres']]\
    .sort_values(by=["imdb_votes", "imdb_score"],
                 ascending=False)\
    .head(5)\
    .reset_index(drop=True)
```

Out[27]:

	release_year	title	type	runtime	imdb_score	imdb_votes	genres
0	2010	Inception	MOVIE	148	8.8	2294231.0	[action, scifi, music, thriller]
1	1994	Forrest Gump	MOVIE	142	8.8	2021343.0	[drama, romance]
2	2012	The Dark Knight Rises	MOVIE	165	8.4	1669067.0	[thriller, action, drama, crime]
3	1995	Se7en	MOVIE	127	8.6	1606270.0	[crime, thriller, drama]
4	2012	Django Unchained	MOVIE	165	8.4	1472668.0	[western, drama]

In [28]:
```
#Top 5 Series with High IMDB votes and Score
df.query("type == 'SHOW'")[['release_year',
                            'title',
                            'type',
                            'runtime',
                            'imdb_score',
                            'imdb_votes',
                            'genres',
                            ]]\
    .sort_values(by=["imdb_votes", "imdb_score"],
                 ascending=False)\
    .head(5)\
    .reset_index(drop=True)
```

Out[28]:

	release_year	title	type	runtime	imdb_score	imdb_votes	genres
0	2008	Breaking Bad	SHOW	48	9.5	1775990.0	[drama, crime, thriller]
1	2016	Stranger Things	SHOW	61	8.7	1101055.0	[scifi, thriller, drama, fantasy, horror]
2	2010	The Walking Dead	SHOW	46	8.2	956604.0	[action, drama, scifi, horror, thriller]
3	2011	Black Mirror	SHOW	59	8.8	526383.0	[drama, scifi, thriller, european]
4	2013	Peaky Blinders	SHOW	58	8.8	511668.0	[crime, drama, european]

In [29]:
```
# Best movie or TV show for every Genre in terms of both IMDB votes and Score
best_by_genre = pd.DataFrame(columns=df.columns.tolist() + ["selected_genre"])

for i, genre in enumerate(sorted(genres)):
    best_genre_data = df.query(f"{genre} == 1").sort_values(by=["imdb_votes", "imdb_score"], ascending=False).reset_index().head(
    best_genre_data["selected_genre"] = genre

    best_by_genre = pd.concat([best_by_genre, best_genre_data], ignore_index=True).reset_index(drop=True)

best_by_genre[['release_year', 'title', 'selected_genre', 'imdb_score']]
```

Out[29]:

	release_year	title	selected_genre	imdb_score
0	2010	Inception	action	8.8
1	2014	The Flash	animation	7.6
2	2000	Snatch	comedy	8.3
3	2008	Breaking Bad	crime	9.5
4	2002	Road to Perdition	documentation	7.7
5	1994	Forrest Gump	drama	8.8
6	1994	Léon: The Professional	european	8.5
7	2014	The Flash	family	7.6
8	2016	Stranger Things	fantasy	8.7
9	2017	Dunkirk	history	7.8
10	2016	Stranger Things	horror	8.7
11	2010	Inception	music	8.8
12	2002	Top Gear	reality	8.7
13	1994	Forrest Gump	romance	8.8
14	2010	Inception	scifi	8.8
15	2013	Rush	sport	8.1
16	2010	Inception	thriller	8.8
17	2014	The Imitation Game	war	8.0
18	2012	Django Unchained	western	8.4

In [30]:
```
#All the best TV show yearwise with Highest IMDB score

gb = df.query("type == 'SHOW'").sort_values(by=["release_year", "imdb_score"], ascending=[True, False]).groupby("release_year")
gb.first()[["title", "imdb_score"]]
```

Out[30]:
```
                                                     title  imdb_score
release_year
        1945             Five Came Back: The Reference Films         NaN
        1969                   Monty Python's Flying Circus         8.8
        1972                 Monty Python's Fliegender Zirkus         8.1
        1981                            Danger Mouse (1981)         7.4
        1982                                    Knight Rider         6.9
        1983                                 Wheel of Fortune         6.7
        1984                                 Thomas & Friends         6.5
        1987                                     Fireman Sam         6.1
        1988                                       High Risk         3.8
        1989                                        Seinfeld         8.9
        1991                                 My First Errand         NaN
        1992                                 Barney & Friends         3.8
        1993                                   Power Rangers         6.5
        1994                             The Magic School Bus         7.8
        1995                         Neon Genesis Evangelion         8.5
        1996                                     Kenan & Kel         7.8
        1997                                   Stargate SG-1         8.4
        1998                             Cowboy Bebop (1998)         8.9
        1999                                       One Piece         8.8
        2000                                          Okupas         9.0
        2001                               Trailer Park Boys         8.6
        2002                                      Still Game         8.9
        2003                                Chappelle's Show         8.8
        2004                                   The Staircase         7.8
        2005                                        Khawatir         9.5
        2006                                      DEATH NOTE         9.0
        2007                                       Heartland         8.4
        2008                                    Breaking Bad         9.5
        2009                                  Midnight Diner         8.6
        2010                       The Great British Baking Show         8.6
        2011                                   Hunter x Hunter         9.0
        2012                                  Call the Midwife         8.5
        2013                                  Attack on Titan         9.0
        2014                 Raja, Rasoi Aur Anya Kahaniyaan         8.9
        2015                                      Reply 1988         9.2
        2016    Leah Remini: Scientology and the Aftermath         9.0
        2017                                  Crazy Delicious         8.9
        2018                                        #ABtalks         9.6
        2019                                      Our Planet         9.3
        2020                                   The Last Dance         9.1
        2021                                          Arcane         9.0
        2022                                    Heartstopper         8.7
```

In [31]:
```
#Longest duration movies in every genere
longest_runtime = pd.DataFrame(columns=df.columns.tolist() + ["selected_genre"])

for i, genre in enumerate(sorted(genres)):
    temp = df[df[genre] == 1].sort_values(by=['runtime'], ascending=False).reset_index(drop=True)
    first = temp.groupby(genre).first()
    first["selected_genre"] = genre
    longest_runtime = pd.concat([longest_runtime, first], ignore_index=True).reset_index(drop=True)

longest_runtime[['title', 'release_year', 'runtime', 'selected_genre']]
```

Out[31]:

	title	release_year	runtime	selected_genre
0	Lagaan: Once Upon a Time in India	2001	224	action
1	Mobile Suit Gundam III: Encounters in Space	1982	144	animation
2	Hum Aapke Hain Koun..!	1994	206	comedy
3	Bonnie & Clyde	2013	240	crime
4	A Lion in the House (2006)	2006	225	documentation
5	Bonnie & Clyde	2013	240	drama
6	Bonnie & Clyde	2013	240	european
7	4K Fireplace	2015	181	family
8	Zero (2018)	2018	180	fantasy
9	Jodhaa Akbar	2008	214	history
10	Apocalypse Now Redux	2001	196	horror
11	No Direction Home: Bob Dylan	2005	208	music
12	4K Fireplace	2015	181	reality
13	Lagaan: Once Upon a Time in India	2001	224	romance
14	Zero (2018)	2018	180	scifi
15	Lagaan: Once Upon a Time in India	2001	224	sport
16	The Irishman	2019	209	thriller
17	Jodhaa Akbar	2008	214	war
18	Wyatt Earp	1994	191	western

In [32]:
```
#Longest season ever made with seasons
df.sort_values(by=['seasons'], ascending=False)\
    .reset_index(drop=True)\
    .head(5)\
    .loc[:, ["title", "release_year", "seasons", "genres"]]
```

Out[32]:

	title	release_year	seasons	genres
0	Survivor	2000	42.0	[reality]
1	Wheel of Fortune	1983	39.0	[family]
2	The Challenge	1998	37.0	[reality, comedy, drama, scifi]
3	Top Gear	2002	32.0	[comedy, reality, european, music]
4	Power Rangers	1993	29.0	[action, scifi, fantasy, family]

In [ ]:

In [ ]:

In [33]:
```
#Recommendation System
```

In [ ]:

In [34]:
```
#Getting Director's name and all the actors name from the credits table
df["director"] = pd.merge(
    df, credits_df[credits_df["role"] == "DIRECTOR"], on="id", how="left"
)["name"].replace(np.nan, None)

df["actors"] = pd.merge(
    df,
    pd.merge(df, credits_df[credits_df["role"] == "ACTOR"], on="id", how="left")
    .groupby("id")["name"]
    .apply(lambda x: x.tolist() if x is not np.nan else None),
    on="id",
    how="left",
)["name"].apply(lambda x: ["" if i is np.nan else str(i) for i in x])

df["actors"] = df["actors"].replace(np.nan, "")
```

In [35]:
```
#Creating a list of words containing all the information about the movie or show
# with director, actors, genres, descriptionn and production country

df["overview"] = (
    (
        df["title"].astype(str)
        + " "
        + df["description"].astype(str)
        + " "
        + df["genres"].apply(lambda x: " ".join(x))
        + " "
        + df["director"].astype(str)
        + " "
        + df["actors"].apply(lambda x: "" if x is [] else " ".join(x))
        + " "
    )
    .str.lower()
    .str.replace("\n", " ")
    .str.replace("—", "")
    .str.translate(str.maketrans("", "", string.punctuation))
)
```

In [36]:
```
#Count vectorizer
count = CountVectorizer(stop_words="english", ngram_range=(1, 5))
count_matrix = count.fit_transform(df["overview"])

cosine_sim = cosine_similarity(count_matrix, count_matrix)

indices = pd.Series(df.index, index=df["title"])
```

In [37]:
```
#Recommendation system funtion to get top 10 movies or shows based on the movie one watched.
count = CountVectorizer(stop_words="english", ngram_range=(1, 5))
count_matrix = count.fit_transform(df["overview"])

cosine_sim = cosine_similarity(count_matrix, count_matrix)

indices = pd.Series(df.index, index=df["title"])
```

In [38]:
```
#Recommendation function to get top 10 movies or shows/ on who watched
def get_recommendations(title, cosine_sim, top_k=5):

    idx = indices[title]

    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[0:20]
    movie_indices = [i[0] for i in sim_scores if i[0] != idx]

    return (
        df.iloc[movie_indices]
        .sort_values(["imdb_votes", "imdb_score"], ascending=False)[
            ["title", "description", "genres", "imdb_score"]
        ]
        .reset_index(drop=True)
        .head(top_k)
    )
```

In [39]:
```
get_recommendations("The Dark Knight Rises", cosine_sim=cosine_sim, top_k=5)
```

Out[39]:

	title	description	genres	imdb_score
0	Forrest Gump	A man with a low IQ has accomplished great things in his life and been present during significant historic events—in each case, far exceeding what anyone imagined he could do. But despite all he has achieved, his one true love eludes him.	[drama, romance]	8.8
1	The Departed	To take down South Boston's Irish Mafia, the police send in one of their own to infiltrate the underworld, not realizing the syndicate has done likewise. While an undercover cop curries favor with the mob kingpin, a career criminal rises through the police ranks. But both sides soon discover there's a mole among them.	[drama, thriller, crime, action]	8.5
2	Full Metal Jacket	A pragmatic U.S. Marine observes the dehumanizing effects the U.S.-Vietnam War has on his fellow recruits from their brutal boot camp training to the bloody street fighting in Hue.	[war, drama]	8.3
3	Sherlock Holmes	Eccentric consulting detective Sherlock Holmes and Doctor John Watson battle to bring down a new nemesis and unravel a deadly plot that could destroy England.	[crime, thriller, action]	7.6
4	War of the Worlds	Ray Ferrier is a divorced dockworker and less-than-perfect father. Soon after his ex-wife and her new husband drop off his teenage son and young daughter for a rare weekend visit, a strange and powerful lightning storm touches down.	[action, thriller, scifi]	6.5

In [40]:
```
get_recommendations("Taxi Driver (1976)", cosine_sim=cosine_sim, top_k=5)
```

Out[40]:

	title	description	genres	imdb_score
0	GoodFellas	The true story of Henry Hill, a half-Irish, half-Sicilian Brooklyn kid who is adopted by neighbourhood gangsters at an early age and climbs the ranks of a Mafia family under the guidance of Jimmy Conway.	[drama, crime]	8.7
1	The Irishman	Pennsylvania, 1956. Frank Sheeran, a war veteran of Irish origin who works as a truck driver, accidentally meets mobster Russell Bufalino. Once Frank becomes his trusted man, Bufalino sends him to Chicago with the task of helping Jimmy Hoffa, a powerful union leader related to organized crime, with whom Frank will maintain a close friendship for nearly twenty years.	[crime, drama, history, thriller]	7.8
2	Once Upon a Time in America	A former Prohibition-era Jewish gangster returns to the Lower East Side of Manhattan over thirty years later, where he once again must confront the ghosts and regrets of his old life.	[crime, drama, european]	8.3
3	Chappelle's Show	Dave Chappelle's singular point of view is unleashed through a combination of laidback stand-up and street-smart sketches.	[comedy, music]	8.8
4	Delhi Crime	As Delhi reels in the aftermath of a gang rape, DCP Vartika Chaturvedi leads a painstaking search for the culprits. Based on the 2012 Nirbhaya case.	[drama, crime]	8.5

In [41]:
```
get_recommendations("GoodFellas", cosine_sim=cosine_sim, top_k=5)
```

Out[41]:

	title	description	genres	imdb_score
0	The Dark Knight Rises	Following the death of District Attorney Harvey Dent, Batman assumes responsibility for Dent's crimes to protect the late attorney's reputation and is subsequently hunted by the Gotham City Police Department. Eight years later, Batman encounters the mysterious Selina Kyle and the villainous Bane, a new terrorist leader who overwhelms Gotham's finest. The Dark Knight resurfaces to protect a city that has branded him an enemy.	[thriller, action, drama, crime]	8.4
1	Catch Me If You Can	A true story about Frank Abagnale Jr. who, before his 19th birthday, successfully conned millions of dollars worth of checks as a Pan Am pilot, doctor, and legal prosecutor. An FBI agent makes it his mission to put him behind bars. But Frank not only eludes capture, he revels in the pursuit.	[drama, crime]	8.1
2	Taxi Driver (1976)	A mentally unstable Vietnam War veteran works as a night-time taxi driver in New York City where the perceived decadence and sleaze feed his urge for violent action.	[drama, crime]	8.2
3	The Irishman	Pennsylvania, 1956. Frank Sheeran, a war veteran of Irish origin who works as a truck driver, accidentally meets mobster Russell Bufalino. Once Frank becomes his trusted man, Bufalino sends him to Chicago with the task of helping Jimmy Hoffa, a powerful union leader related to organized crime, with whom Frank will maintain a close friendship for nearly twenty years.	[crime, drama, history, thriller]	7.8
4	Once Upon a Time in America	A former Prohibition-era Jewish gangster returns to the Lower East Side of Manhattan over thirty years later, where he once again must confront the ghosts and regrets of his old life.	[crime, drama, european]	8.3

In [ ]:
```
#Recommendation system for Netflix dataset.
```