Project Name - 📊 Amazon Prime TV Shows & Movies EDA

Project Type - EDA

Contribution - Individual

📝 Project Summary

This project focuses on analyzing Amazon Prime's vast catalog of TV shows and movies through exploratory data analysis (EDA). With the ever-increasing competition in the streaming industry, platforms like Amazon Prime Video continuously add a wide range of content to meet diverse viewer preferences. Understanding the structure and characteristics of this content is essential for drawing meaningful conclusions about what makes content successful, which genres dominate, how ratings vary, and how viewer preferences are reflected across different types of content.

The datasets used in this project provide detailed metadata about Amazon Prime's content offerings. The first dataset includes information about each title, such as the title name, content type (movie or show), description, release year, age certification, runtime, genres, production countries, IMDb and TMDB scores, popularity metrics, and the number of seasons (for shows). The second dataset complements the first by listing cast members, their roles (actor, director, etc.), and the characters they play in each movie or show.

By applying exploratory data analysis techniques Python libraries like Pandas, Matplotlib, and Seaborn, this project aims to uncover insights from these datasets. EDA helps us understand the data's structure, identify patterns, discover anomalies, and test hypotheses visually and statistically. The process involves data cleaning, handling missing values, and transforming columns for better interpretation.

Several key questions will guide the analysis:

- **Content Composition**: What is the distribution of movies vs. TV shows? Are there more recent releases, or is the catalog focused on older titles?
- **Genre Trends**: Which genres are most common across movies and TV shows? Are there any genres exclusive to TV or film?
- **Content Quality**: What are the average IMDb and TMDB scores? How do ratings differ between content typ Amazon Prime?
- **Runtime Analysis**: What is the average duration of movies and shows? Are there signifings or runtime?
- **Cast and Talent**: Which actors or directors appear most frequently? Does the presence of popular cast members correlate with high ratings or popularity?

Through visualizations such as bar plots, histograms, pie charts, heatmaps, and box plots, we will represent trends and relationships's target audcaste. By merging cast data with title information, we can also analyze the contribution of popular personalities to content performance.

This analysis is not only useful for viewers or researchers but can also provide valuable business insights. For instance, if a particular genre consistently scores higher or garners more popularity, Amazon could prioritize acquiring or producing similar content. Similarly, understanding which certifications or runtimes attract better reception can inform content strategy and marketing decisions.

In conclusion, this project will provide a data-driven overview of Amazon Prime's content catalog. The findings aim to highlight viewing patterns, quality indicators, and platform strategies from the perspective of content metadata. By extracting actionable insights, this EDA project showcases how raw data can reveal meaningful patterns and support better decision-making in digital entertainment.

GitHub Link -

https://github.com/Raviranjan1208/Amazon-Prime-EDA-using-Python

Problem Statement -

Amazon Prime Video offers a vast and diverse collection of TV shows and movies, but the platform lacks easily accessible insights into viewer preferences, content trends, and performance indicators. The objective of this project is to perform Exploratory Data Analysis (EDA) on Amazon Prime's catalog using available metadata (genres, release years, ratings, popularity, runtime, cast, etc.) to answer key business and consumer-focused questions. By uncovering patterns and correlations within the data, the goal is to identify what type of content performs well, which genres or certifications dominate, and how cast, ratings, or runtime influence a title's success on the platform.

🎯 Business Objective

The primary objective of this project is to perform an in-depth Exploratory Data Analysis (EDA) on Amazon Prime Video content to extract meaningful insights that can drive strategic business decisions.

Specifically, this analysis aims to:

1. **Understand Content Distribution**:
    - Analyze the distribution of genres, age certifications, runtimes, and release years.
    - Identify trends and patterns in content types over time.
2. **Evaluate Content Performance**:
    - Assess how content scores on IMDb and TMDb.
    - Identify characteristics of high-performing titles (e.g., genre, length, certification).
3. **Optimize User Engagement & Recommendations**:
    - Extract insights to improve personalized recommendations.
    - Understand what kind of content drives viewership and retention.
4. **Support Content Acquisition Strategy**:
    - Guide data-driven decisions in acquiring or producing content that aligns with viewer preferences and platform trends.
5. **Clean and Transform Data for Modeling**:
    - Prepare the dataset for further modeling or visualization by handling missing values, encoding categorical data, and deriving new features.

The insights from this analysis are expected to help Amazon Prime Video optimize its content catalog, enhance user experience, and make smarter business investments in content strategy.

Let's Begin !

1. Know Your Data

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Dataset Loading

In [2]:
```
# Here, I have loaded both the datsets
credits = pd.read_csv('credits.csv')
titles = pd.read_csv('titles.csv')
```

Dataset First look

In [3]:
```
credits
```

Out[3]:

	person_id	id	name	character	role
0	59401	ts20945	Joe Besser	Joe	ACTOR
1	31460	ts20945	Moe Howard	Moe	ACTOR
2	31461	ts20945	Larry Fine	Larry	ACTOR
3	21174	tm19248	Buster Keaton	Johnny Gray	ACTOR
4	28713	tm19248	Marion Mack	Annabelle Lee	ACTOR
...	...	...	...	...	...
124230	1938589	tm1054116	Sangam Shukla	Madhav	ACTOR
124231	1938565	tm1054116	Vijay Thakur	Sanjay Thakur	ACTOR
124232	728899	tm1054116	Vanya Wellens	Budhiya	ACTOR
124233	1938620	tm1054116	Vishwa Bhanu	Gissu	ACTOR
124234	1938620	tm1054116	Vishwa Bhanu	NaN	DIRECTOR

124235 rows × 5 columns

In [4]:
```
titles
```

Out[4]:

	id	title	type	description	release_year	age_certification	runtime	genres
0	ts20945	The Three Stooges	SHOW	The Three Stooges were an American vaudeville ...	1934	TV-PG	19	['comedy', 'family', 'animation', 'action', 'f...
1	tm19248	The General	MOVIE	During America’s Civil War, Union spies steal ...	1926	NaN	78	['action', 'drama', 'war', 'western', 'comedy'...
2	tm82253	The Best Years of Our Lives	MOVIE	It's the hope that sustains the spirit of ever...	1946	NaN	171	['romance', 'war', 'drama']
3	tm83884	His Girl Friday	MOVIE	Hildy, the journalist former wife of newspaper...	1940	NaN	92	['comedy', 'drama', 'romance']
4	tm56584	In a Lonely Place	MOVIE	An aspiring actress begins to suspect that her...	1950	NaN	94	['thriller', 'drama', 'romance']
...	...	...	...	...	...	...	...	...
9866	tm510327	Lily Is Here	MOVIE	Dallas and heroin have one thing in common: Du...	2021	NaN	93	['drama']
9867	tm1079144	Jay Nog: Something from Nothing	MOVIE	Something From Nothing takes you on a stand-up...	2021	NaN	55	['comedy']
9868	tm847725	Chasing	MOVIE	A cop from Chennai sets out to nab a dreaded d...	2021	NaN	116	['crime']
9869	tm1054116	Baikunth	MOVIE	This story is about prevalent caste problem, e...	2021	NaN	72	['family', 'drama']
9870	ts275838	Waking Up Eighty	SHOW	Kara Stewart, 16, is fed up with just about ev...	2021	NaN	10	['drama']

9871 rows × 15 columns

Dataset Rows & columns Count

To count rows and columns I used ***shape*** - EX - df.shape

In [5]:
```
credits.shape    # Rows = 124235, Columns = 5
```

Out[5]:
```
(124235, 5)
```

In [6]:
```
titles.shape     # Rows = 9871, Columns = 15
```

Out[6]:
```
(9871, 15)
```

Duplicate Values

In [7]:
```
# Count how many duplicate rows are in the credits data
credits.duplicated().sum()
```

Out[7]:
```
56
```

In [8]:
```
# Count how many duplicate rows are in the titles data
titles.duplicated().sum()
```

Out[8]:
```
3
```

Null Values

Counting Null value

In [9]:
```
# Displaying Column wise null value - Credits
credits.isnull().sum()
```

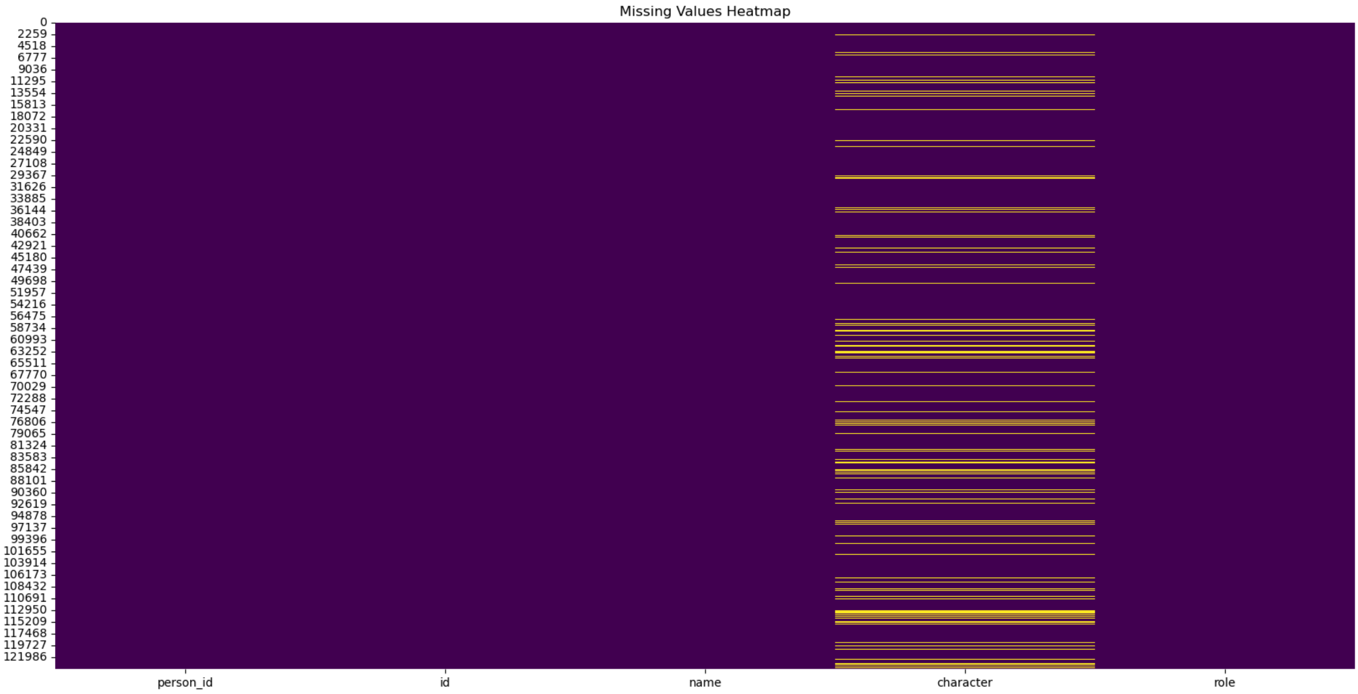
Out[9]:
```
person_id        0
id               0
name             0
character    16287
role             0
dtype: int64
```

In [10]:
```
# Displaying Column wise null value - titles
titles.isnull().sum()
```

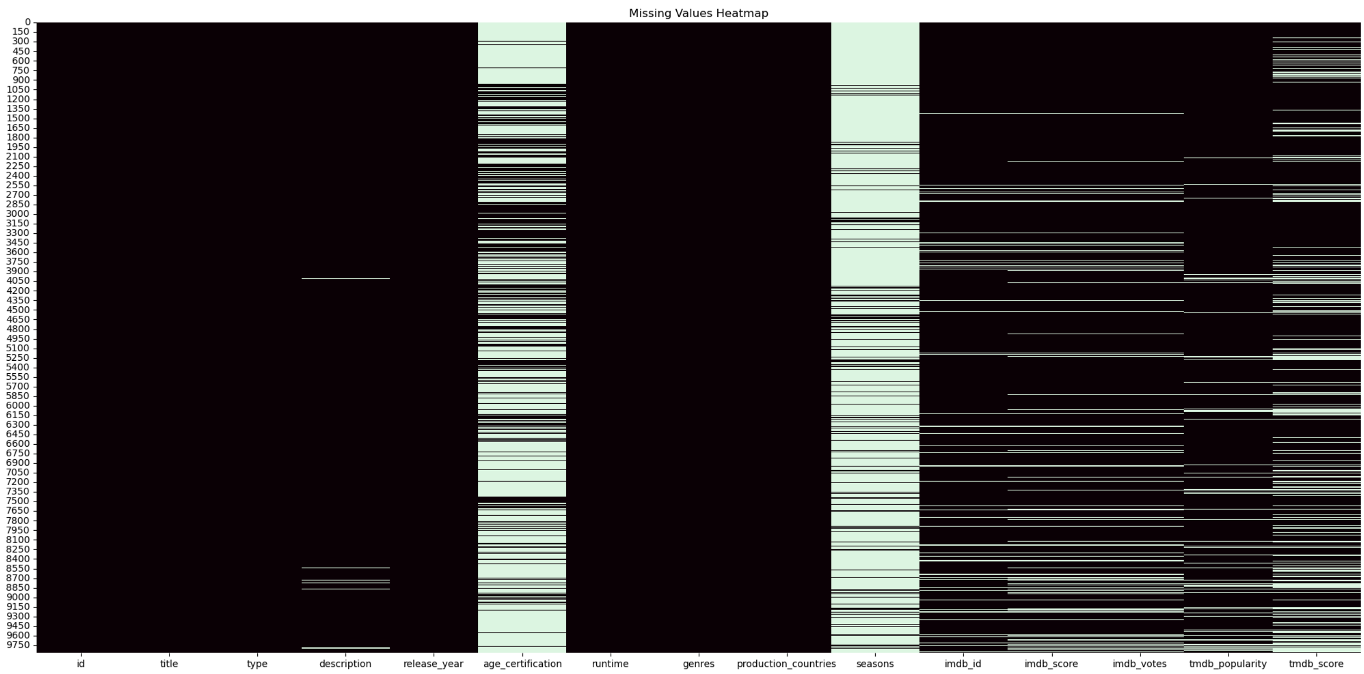
Out[10]:
```
id                         0
title                      0
type                       0
description              119
release_year               0
age_certification       6487
runtime                    0
genres                     0
production_countries       0
seasons                 8514
imdb_id                  667
imdb_score              1021
imdb_votes              1031
tmdb_popularity          547
tmdb_score              2082
dtype: int64
```

Visualizing the null values

In [11]:
```python
# Visualizing the null vlaues in credits
plt.figure(figsize=(20, 10))
sns.heatmap(credits.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()
```



In [12]:
```python
# Visualizing the null vlaues in titles
plt.figure(figsize=(25, 12))
sns.heatmap(titles.isnull(), cbar=False, cmap='mako')
plt.title("Missing Values Heatmap")
plt.show()
```



What did you know about your dataset?

ABout *Credits* data

This dataset contains information about the cast and crew involved in Amazon Prime content. Below is a summary of what I discovered:

🔢 Dataset Dimensions - Rows: 124235 & Columns: 5

📉 Missing Values The character column has 16,287 missing values, which is expected for roles like DIRECTOR or technical crew who don't play characters.

📉 Duplicate Values - 56 duplicate values are there.

🧾 Data Types - person_id is numeric. All other columns (id, name, character, role) are strings (object type).

Column	Description
`person_id`	Unique numeric identifier for a person
`id`	Identifier for the movie/show
`name`	Name of the person
`character`	Character played (if applicable)
`role`	Role type (e.g., ACTOR, DIRECTOR)

ABout *Titles* data

This dataset contains metadata for Amazon Prime movies and TV shows, including details like title, release year, ratings, and popularity metrics. Here are the key observations:

🔢 Dataset Dimensions - Rows: 9871 & Columns: 15

📉 Missing Values Some columns contain missing data:

description: 119 missing

age_certification: 6,487 missing

seasons: 8,514 missing (expected for movies)

imdb_id: 667 missing

imdb_score: 1,021 missing

imdb_votes: 1,031 missing

tmdb_popularity: 547 missing

tmdb_score: 2,082 missing

📉 Duplicate Values - 3 duplicate values are there.

🧾 Data Types - release_yeat and runtime is numeric. All other columns are strings and float.

Column Name	Description
`id`	Unique identifier for the content
`title`	Title of the movie or show
`type`	Format of content (MOVIE or SHOW)
`description`	Short summary of the content
`release_year`	Year of release

Column Name	Description
`age_certification`	Age rating (e.g., PG, R, TV-MA)
`runtime`	Duration in minutes
`genres`	List of genres (e.g., comedy, drama)
`production_countries`	Country codes (e.g., US, IN)
`seasons`	Number of seasons (NaN for movies)
`imdb_id`	IMDb identifier
`imdb_score`	IMDb rating score
`imdb_votes`	Number of IMDb votes
`tmdb_popularity`	Popularity score on TMDb
`tmdb_score`	TMDb rating score

2. Understanding Your Variables

Dataset columns

In [13]:
```
# Viewing Columns in credits
credits.columns
```

Out[13]:
```
Index(['person_id', 'id', 'name', 'character', 'role'], dtype='object')
```

In [14]:
```
# Viewing Columns in titles
titles.columns
```

Out[14]:
```
Index(['id', 'title', 'type', 'description', 'release_year',
       'age_certification', 'runtime', 'genres', 'production_countries',
       'seasons', 'imdb_id', 'imdb_score', 'imdb_votes', 'tmdb_popularity',
       'tmdb_score'],
      dtype='object')
```

Check Unique Values for each variable.

In [15]:
```
credits.nunique()
```

Out[15]:
```
person_id    80508
id            8861
name         79758
character    71097
role             2
dtype: int64
```

In [16]:
```
titles.nunique()
```

Out[16]:
```
id                      9868
title                   9737
type                       2
description             9734
release_year             110
age_certification         11
runtime                  207
genres                  2028
production_countries     497
seasons                   32
imdb_id                 9201
imdb_score                86
imdb_votes              3650
tmdb_popularity         5325
tmdb_score                89
dtype: int64
```

3. *Data Wrangling*

Data Wrangling (also called Data Munging) is the process of cleaning, transforming, and organizing raw data into a usable format for analysis. There are so many steps in data wrangling.

Step 1 - Merging Dataset

Now, Have to merge `credits` and `titles` dataset for further analysis

In [17]:
```
print(credits.shape)
credits.columns
```

```
(124235, 5)
```

Out[17]:
```
Index(['person_id', 'id', 'name', 'character', 'role'], dtype='object')
```

In [18]:
```
print(titles.shape)
titles.columns
```

```
(9871, 15)
```

Out[18]:
```
Index(['id', 'title', 'type', 'description', 'release_year',
       'age_certification', 'runtime', 'genres', 'production_countries',
       'seasons', 'imdb_id', 'imdb_score', 'imdb_votes', 'tmdb_popularity',
       'tmdb_score'],
      dtype='object')
```

Left Join - It returns all information from left data and mathed data from right data

I have performed `left` join here, supposing `titles` is left dataset and `credits` is right dataset. Because i need all data from titles

In [19]:
```
# Merging data based on common column 'id'
data = pd.merge(titles, credits, on='id', how='left')
```

In [20]:
```
data.head(3)
```

Out[20]:

	id	title	type	description	release_year	age_certification	runtime	genres	producti
0	ts20945	The Three Stooges	SHOW	The Three Stooges were an American vaudeville ...	1934	TV-PG	19	['comedy', 'family', 'animation', 'action', 'f...	
1	ts20945	The Three Stooges	SHOW	The Three Stooges were an American vaudeville ...	1934	TV-PG	19	['comedy', 'family', 'animation', 'action', 'f...	
2	ts20945	The Three Stooges	SHOW	The Three Stooges were an American vaudeville ...	1934	TV-PG	19	['comedy', 'family', 'animation', 'action', 'f...	

In [21]:
```
data.shape
```

Out[21]: (125354, 19)

In [22]:
```
data.columns
```

Out[22]:
```
Index(['id', 'title', 'type', 'description', 'release_year',
       'age_certification', 'runtime', 'genres', 'production_countries',
       'seasons', 'imdb_id', 'imdb_score', 'imdb_votes', 'tmdb_popularity',
       'tmdb_score', 'person_id', 'name', 'character', 'role'],
      dtype='object')
```

In [23]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125354 entries, 0 to 125353
Data columns (total 19 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   id                    125354 non-null  object 
 1   title                 125354 non-null  object 
 2   type                  125354 non-null  object 
 3   description           125163 non-null  object 
 4   release_year          125354 non-null  int64  
 5   age_certification     56857 non-null   object 
 6   runtime               125354 non-null  int64  
 7   genres                125354 non-null  object 
 8   production_countries  125354 non-null  object 
 9   seasons               8501 non-null    float64
 10  imdb_id               119978 non-null  object 
 11  imdb_score            118987 non-null  float64
 12  imdb_votes            118957 non-null  float64
 13  tmdb_popularity       124800 non-null  float64
 14  tmdb_score            114263 non-null  float64
 15  person_id             124347 non-null  float64
 16  name                  124347 non-null  object 
 17  character             108040 non-null  object 
 18  role                  124347 non-null  object 
dtypes: float64(6), int64(2), object(11)
memory usage: 18.2+ MB
```

In [24]:
```
data.describe()   # it gives statistical information
```

Out[24]:

	release_year	runtime	seasons	imdb_score	imdb_votes	tmdb_popularity	tm
count	125354.000000	125354.00000	8501.000000	118987.000000	1.189570e+05	124800.000000	1142
mean	1996.374715	95.30792	2.335372	5.970856	2.311206e+04	10.134483	
std	27.758800	30.39349	3.164860	1.243967	8.816389e+04	40.666765	
min	1912.000000	1.00000	1.000000	1.100000	5.000000e+00	0.000011	
25%	1983.000000	82.00000	1.000000	5.200000	2.790000e+02	1.858000	
50%	2009.000000	93.00000	1.000000	6.100000	1.227000e+03	3.864000	
75%	2017.000000	109.00000	2.000000	6.800000	7.039000e+03	8.787000	
max	2022.000000	549.00000	51.000000	9.900000	1.133692e+06	1437.906000	

In [25]:
```
data.duplicated().sum()
```

Out[25]:
```
168
```

In [26]:
```
data.nunique()
```

Out[26]:
```
id                      9868
title                   9737
type                       2
description             9734
release_year             110
age_certification         11
runtime                  207
genres                  2028
production_countries     497
seasons                   32
imdb_id                 9201
imdb_score                86
imdb_votes              3650
tmdb_popularity         5325
tmdb_score                89
person_id              80508
name                   79758
character              71097
role                       2
dtype: int64
```

In [27]:
```
data.isnull().sum()
```

Out[27]:
```
id                          0
title                       0
type                        0
description               191
release_year                0
age_certification       68497
runtime                     0
genres                      0
production_countries        0
seasons                116853
imdb_id                  5376
imdb_score               6367
imdb_votes               6397
tmdb_popularity           554
tmdb_score              11091
person_id                1007
name                     1007
character               17314
role                     1007
dtype: int64
```

Steps 2 - Droping duplicates

In [28]:
```
data.duplicated().sum()
```

Out[28]:
```
168
```

In [29]:
```
data.drop_duplicates(inplace = True)
# The (inplace = True) will make sure that the method does NOT return a new DataFrame, but it
```

In [30]:
```
data.duplicated().sum()  # Duplicates value has removed
```

Out[30]:
```
0
```

Step 3 - Droping Columns

Here, So many columns which not be needed. So, we can drop these columns

Here, I'm droping `description` and `imdb_id` column because that column not needed for the analysis and `age_certification` and `seasons` columns due to exessive null values

In [31]:
```
data.columns
```

Out[31]:
```
Index(['id', 'title', 'type', 'description', 'release_year',
       'age_certification', 'runtime', 'genres', 'production_countries',
       'seasons', 'imdb_id', 'imdb_score', 'imdb_votes', 'tmdb_popularity',
       'tmdb_score', 'person_id', 'name', 'character', 'role'],
      dtype='object')
```

In [32]:
```
# Droping Columns
data.drop(['description'], axis = 1, inplace = True)
data.drop(['age_certification'], axis = 1, inplace = True)
data.drop(['seasons'], axis = 1, inplace = True)
data.drop(['imdb_id'], axis = 1, inplace = True)
```

In [33]:
```
print(data.shape)
data.columns
```

```
(125186, 15)
```

Out[33]:
```
Index(['id', 'title', 'type', 'release_year', 'runtime', 'genres',
       'production_countries', 'imdb_score', 'imdb_votes', 'tmdb_popularity',
       'tmdb_score', 'person_id', 'name', 'character', 'role'],
      dtype='object')
```

Steps 4 - Filling null values

In [34]:
```
data.dtypes
```

Out[34]:
```
id                      object
title                   object
type                    object
release_year             int64
runtime                  int64
genres                  object
production_countries    object
imdb_score             float64
imdb_votes             float64
tmdb_popularity        float64
tmdb_score             float64
person_id              float64
name                    object
character               object
role                    object
dtype: object
```

In [35]:
```
# Check and handle null values
print("Missing values:\n", data.isnull().sum())
```

```
Missing values:
 id                        0
title                     0
type                      0
release_year              0
runtime                   0
genres                    0
production_countries      0
imdb_score             6367
imdb_votes             6397
tmdb_popularity         554
tmdb_score            10995
person_id              1007
name                   1007
character             17284
role                   1007
dtype: int64
```

Above we can see there are multiple columns have null value and two types of data `float` and `string` which have null values

In [36]:
```
# Filling null values of string data type column
data["person_id"] = data["person_id"].fillna("NA")
data["name"] = data["name"].fillna("NA")
data["character"] = data["character"].fillna("NA")
data["role"] = data["role"].fillna("NA")
```

In [37]:
```
data.isnull().sum()
# See, filled null values
```

```
Out[37]: id                        0
         title                     0
         type                      0
         release_year              0
         runtime                   0
         genres                    0
         production_countries      0
         imdb_score             6367
         imdb_votes             6397
         tmdb_popularity         554
         tmdb_score            10995
         person_id                 0
         name                      0
         character                 0
         role                      0
         dtype: int64
```

```
In [38]: # Filling null values of float data type column
         data['imdb_score'] = data['imdb_score'].fillna(data['imdb_score'].median())
         data['imdb_votes'] = data['imdb_votes'].fillna(0)
         data['tmdb_popularity'] = data['tmdb_popularity'].fillna(0)
         data['tmdb_score'] = data['tmdb_score'].fillna(data['tmdb_score'].median())
```

```
In [39]: data.isnull().sum()
```

```
Out[39]: id                      0
         title                   0
         type                    0
         release_year            0
         runtime                 0
         genres                  0
         production_countries    0
         imdb_score              0
         imdb_votes              0
         tmdb_popularity         0
         tmdb_score              0
         person_id               0
         name                    0
         character               0
         role                    0
         dtype: int64
```

Step 5 - Converting data type

Convert `genres` and `production_countries` from string to list

Above, we can see columns "genres" and "production_countries" are stored as string representations of lists (e.g., "['comedy', 'drama']")

```
In [40]: import ast
         # This line imports the ast module, which stands for Abstract Syntax Trees.
         # The ast module lets you safely evaluate and convert strings into Python objects.
```

```
In [41]: data.head(3)
```

Out[41]:

	id	title	type	release_year	runtime	genres	production_countries	imdb_score	imd
0	ts20945	The Three Stooges	SHOW	1934	19	['comedy', 'family', 'animation', 'action', 'f...	['US']	8.6	
1	ts20945	The Three Stooges	SHOW	1934	19	['comedy', 'family', 'animation', 'action', 'f...	['US']	8.6	
2	ts20945	The Three Stooges	SHOW	1934	19	['comedy', 'family', 'animation', 'action', 'f...	['US']	8.6	

In [42]:
```
data['genres'] = data['genres'].apply(ast.literal_eval)
data['production_countries'] = data['production_countries'].apply(ast.literal_eval)
```

In [43]:
```
data.head(2)
```

Out[43]:

	id	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb
0	ts20945	The Three Stooges	SHOW	1934	19	[comedy, family, animation, action, fantasy, h...	[US]	8.6	
1	ts20945	The Three Stooges	SHOW	1934	19	[comedy, family, animation, action, fantasy, h...	[US]	8.6	

In [44]:
```
# Change data type list to string
data['genres'] = data['genres'].apply(lambda x: ', '.join(x))
data['production_countries'] = data['production_countries'].apply(lambda x: ', '.join(x))
```

In [45]:
```
# Created new feature: 'decade' - creates a new column decade to show the decade each title wa
data['decade'] = (data['release_year'] // 10) * 10
```

In [46]:
```
data.reset_index(drop=True, inplace=True)
data.head(2)
# Here, reseting the index because after removing duplicates records/rows indexing had deteri
```

Out[46]:

	id	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb
0	ts20945	The Three Stooges	SHOW	1934	19	comedy, family, animation, action, fantasy, ho...	US	8.6	
1	ts20945	The Three Stooges	SHOW	1934	19	comedy, family, animation, action, fantasy, ho...	US	8.6	

In [47]:

```
data.shape
```

Out[47]: (125186, 16)

In [48]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125186 entries, 0 to 125185
Data columns (total 16 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   id                    125186 non-null  object 
 1   title                 125186 non-null  object 
 2   type                  125186 non-null  object 
 3   release_year          125186 non-null  int64  
 4   runtime               125186 non-null  int64  
 5   genres                125186 non-null  object 
 6   production_countries  125186 non-null  object 
 7   imdb_score            125186 non-null  float64
 8   imdb_votes            125186 non-null  float64
 9   tmdb_popularity       125186 non-null  float64
 10  tmdb_score            125186 non-null  float64
 11  person_id             125186 non-null  object 
 12  name                  125186 non-null  object 
 13  character             125186 non-null  object 
 14  role                  125186 non-null  object 
 15  decade                125186 non-null  int64  
dtypes: float64(4), int64(3), object(9)
memory usage: 15.3+ MB
```

✅ Data Wrangling / Manipulations Done

📂 1. Dataset Loading

Loaded two datasets:

- `titles` : Metadata for movies and TV shows
- `credits` : Actor, director, and character information
- And read the data and get all information about them

🔗 2. Merge Operation

Merged `credits` and `titles` on the common key `id`, creating a master dataset with both content metadata and cast information.

📉 3. Missing Values Handling

Identified missing values in columns such as:

- `age_certification`
- `imdb_score`, `imdb_votes`, `tmdb_score`
- `description`, `seasons` (mostly missing in movies)

Next steps will involve imputing or dropping based on relevance.

🧹 4. Column Cleanup

Evaluated the necessity of columns:

- `imdb_id`: Currently not required unless deep linking or scraping is planned — likely to be dropped.
- Reviewed nested columns (`genres`, `production_countries`) which are stored as lists/strings — will be cleaned for better analysis.

🏷️ 5. Data Type Consistency

Ensured proper data types:

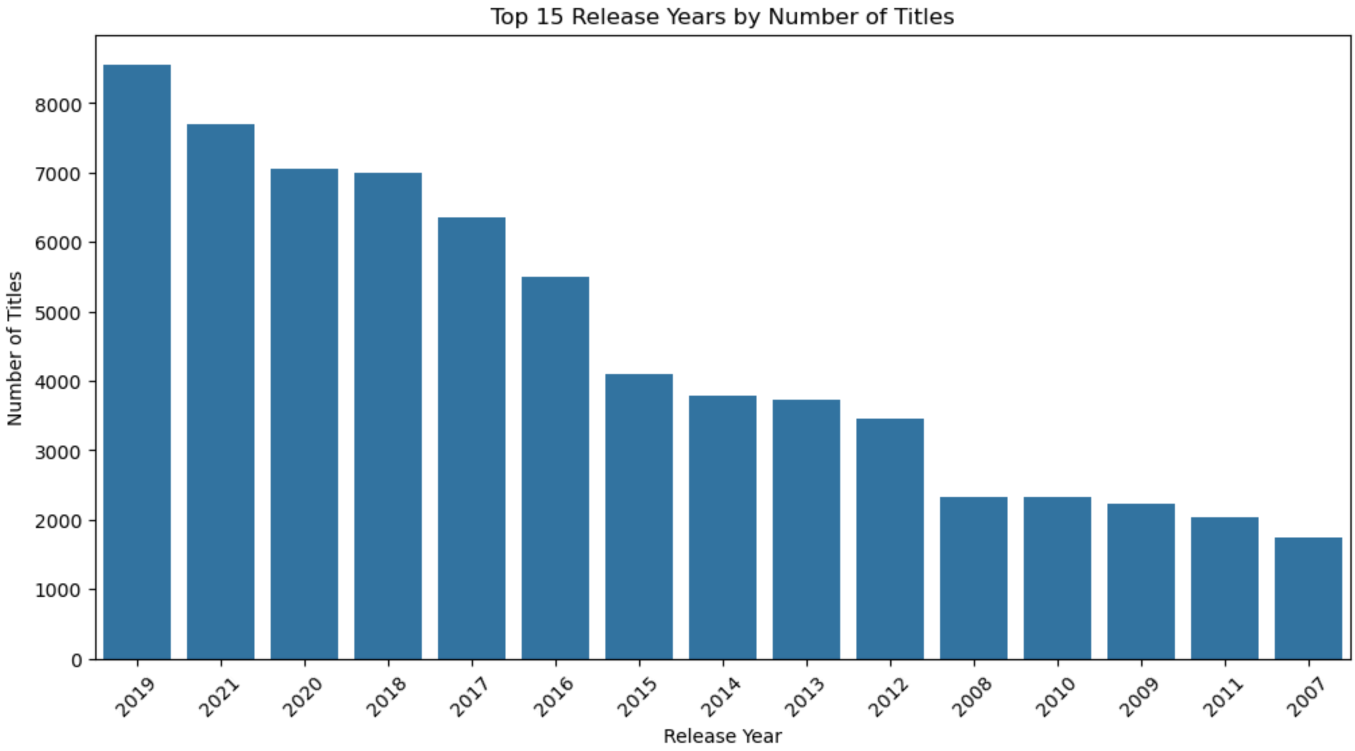
- **Numeric**: `runtime`, `imdb_score`, `tmdb_popularity`
- **Categorical**: `type`, `genres`

📊 Preliminary Insights

1. **Content Type**
    - The dataset includes both movies and TV shows, identifiable via the `type` column.
    - Initial review shows more movies than shows in the catalog.
2. **Genres Distribution**
    - Titles can belong to multiple genres (e.g., 'Comedy', 'Drama').
    - Common genres include: Drama, Comedy, Action, Romance
3. **Ratings Overview**
    - IMDb and TMDB scores range between 6 and 9 for most titles.
    - Many entries lack either score — particularly for older or less popular content.
4. **Cast Popularity**
    - Cast data reveals repeated actors and directors.
    - Will later evaluate if specific actors/directors correlate with higher scores.

Chart - 1

In [49]:
```
# Top 15 release years
plt.figure(figsize=(12,6))
sns.countplot(data=data, x='release_year', order=data['release_year'].value_counts().index[:1
plt.xticks(rotation=45)
plt.title('Top 15 Release Years by Number of Titles')
plt.xlabel('Release Year')
plt.ylabel('Number of Titles')
plt.show()
```



1. Why did you pick the specific chart?

The **countplot** is ideal for visualizing the frequency of categorical data. In this case, it helps us understand how many titles were released each year. Focusing on the top 15 years provides a clear and digestible view of the most active release periods without clutter.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- There was a significant increase in content release between 2018 and 2021, indicating a strategic push to expand the platform's library.

3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

- `Content Planning`: Identifying peak content release years can help replicate successful content strategies in the future.
- `Resource Allocation`: Understand which years were highly productive to analyze underlying resource or investment patterns.
- `User Engagement Analysis`: Correlating years with increased user engagement or subscription growth can guide future content drops.

Chart - 2

In [50]:
```
# top 15 diretors with most directed movies
filtered_directors = data[data['role'] == 'DIRECTOR']
```

```python
filtered_directors.head(3)          # I got the director wise data
```

Out[50]:

	id	title	type	release_year	runtime	genres	production_countries	imdb_score	imd
25	tm19248	The General	MOVIE	1926	78	action, drama, war, western, comedy, european	US	8.2	
26	tm19248	The General	MOVIE	1926	78	action, drama, war, western, comedy, european	US	8.2	
57	tm82253	The Best Years of Our Lives	MOVIE	1946	171	romance, war, drama	US	8.1	

In [51]:

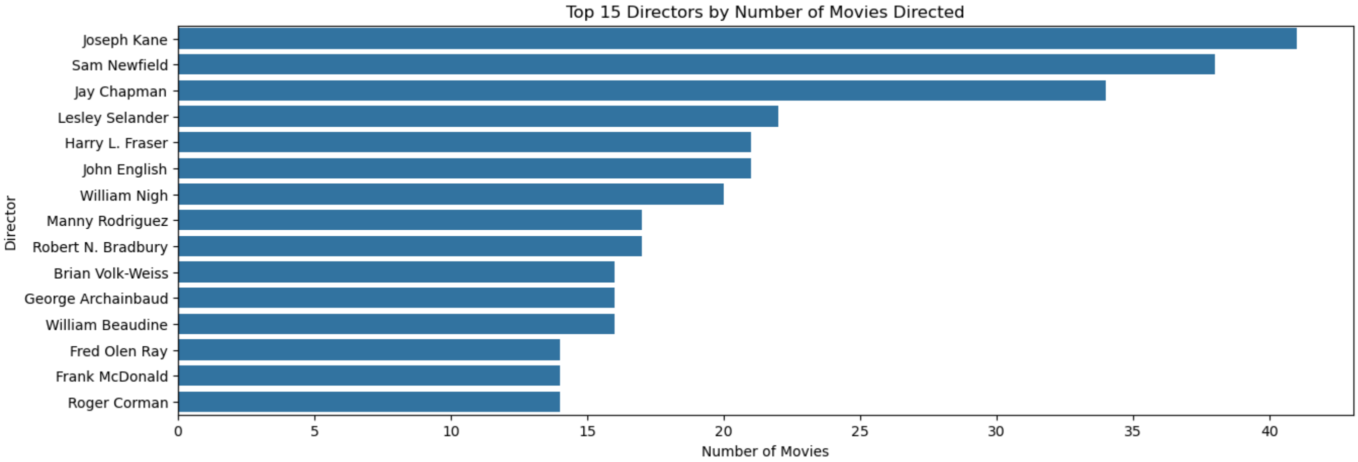
```python
top_directors = filtered_directors['name'].value_counts().head(15)
top_directors
```

Out[51]:

```
name
Joseph Kane           41
Sam Newfield          38
Jay Chapman           34
Lesley Selander       22
Harry L. Fraser       21
John English          21
William Nigh          20
Manny Rodriguez       17
Robert N. Bradbury    17
Brian Volk-Weiss      16
George Archainbaud    16
William Beaudine      16
Fred Olen Ray         14
Frank McDonald        14
Roger Corman          14
Name: count, dtype: int64
```

In [52]:

```python
plt.figure(figsize=(15,5))
sns.barplot(x=top_directors.values, y=top_directors.index)
plt.title('Top 15 Directors by Number of Movies Directed')
plt.xlabel('Number of Movies')
plt.ylabel('Director')
plt.show()
```



1. Why did you pick the specific chart?

A **horizontal bar** plot is ideal for showing the ranking of categorical data like director names, especially when the labels (names) are long. It clearly visualizes which directors have directed the most content, making it easy to compare across the top 15.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- Certain directors have created significantly more content on the platform.
- This may indicate either popularity, strong viewer engagement, or high trust from the platform.
- It helps identify the most prolific content creators.

3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

- The platform can use this data to collaborate more with these top-performing directors.
- It can also promote their existing works to increase viewer retention and watch time.
- Furthermore, understanding viewer preference by linking director names with viewer ratings could help in future content acquisition and personalized recommendations.

Chart - 3

In [53]:
```
# top 20 Actors
filtered_actors = data[data['role'] == 'ACTOR']
filtered_actors.head()
```

Out[53]:

	id	title	type	release_year	runtime	genres	production_countries	imdb_score	imd
0	ts20945	The Three Stooges	SHOW	1934	19	comedy, family, animation, action, fantasy, ho...	US	8.6	
1	ts20945	The Three Stooges	SHOW	1934	19	comedy, family, animation, action, fantasy, ho...	US	8.6	
2	ts20945	The Three Stooges	SHOW	1934	19	comedy, family, animation, action, fantasy, ho...	US	8.6	
3	tm19248	The General	MOVIE	1926	78	action, drama, war, western, comedy, european	US	8.2	
4	tm19248	The General	MOVIE	1926	78	action, drama, war, western, comedy, european	US	8.2	

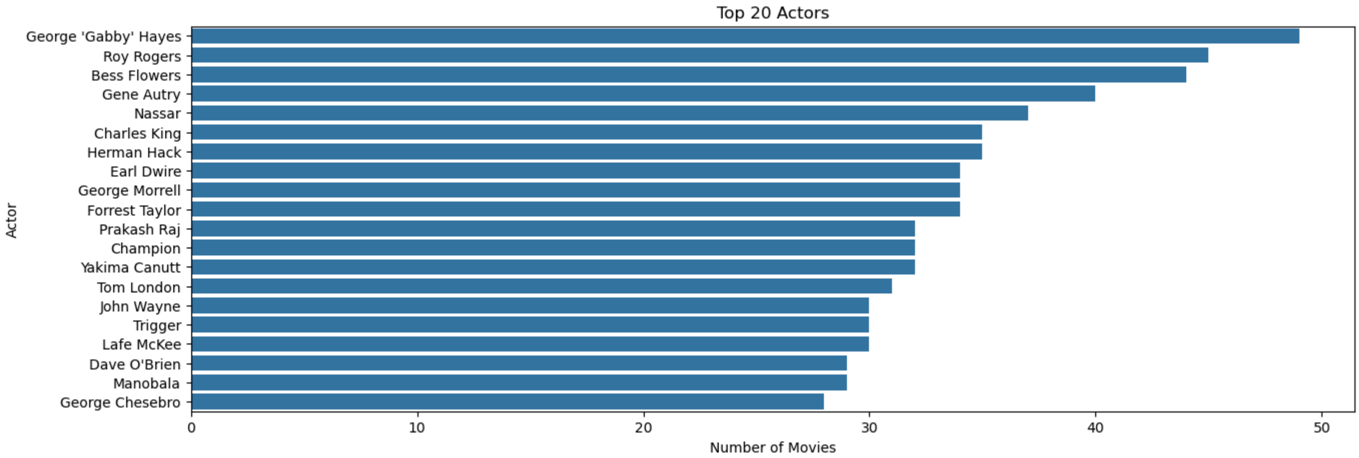
In [54]:

```
top_actors = filtered_actors['name'].value_counts().head(20)
top_actors
```

Out[54]:

```
name
George 'Gabby' Hayes    49
Roy Rogers              45
Bess Flowers            44
Gene Autry              40
Nassar                  37
Charles King            35
Herman Hack             35
Earl Dwire              34
George Morrell          34
Forrest Taylor          34
Prakash Raj             32
Champion                32
Yakima Canutt           32
Tom London              31
John Wayne              30
Trigger                 30
Lafe McKee              30
Dave O'Brien            29
Manobala                29
George Chesebro         28
Name: count, dtype: int64
```

```
In [55]: plt.figure(figsize=(15,5))
sns.barplot(x=top_actors.values, y=top_actors.index)
plt.title('Top 20 Actors')
plt.xlabel('Number of Movies')
plt.ylabel('Actor')
plt.show()
```



1. Why did you pick the specific chart?

A **horizontal bar** chart is ideal for comparing categorical data when the category names (in this case, actor names) are long. It allows better readability and clearly shows the relative number of movies each actor has appeared in.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- The chart reveals the top 20 most featured actors on Amazon Prime.
- Certain actors have significantly more titles than others, suggesting their content may be highly valued by the platform.
- This could also indicate potential popularity, content availability, or contractual relationships with specific studios or actors.

3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

- Helps Amazon Prime identify key actors driving content volume.
- Useful for marketing strategies (e.g., recommending popular actor content).
- Supports licensing decisions or future collaborations with frequently featured actors to retain or attract subscribers.

Chart - 4

```
In [56]: data_dacade = data['decade'].value_counts().sort_index()
```

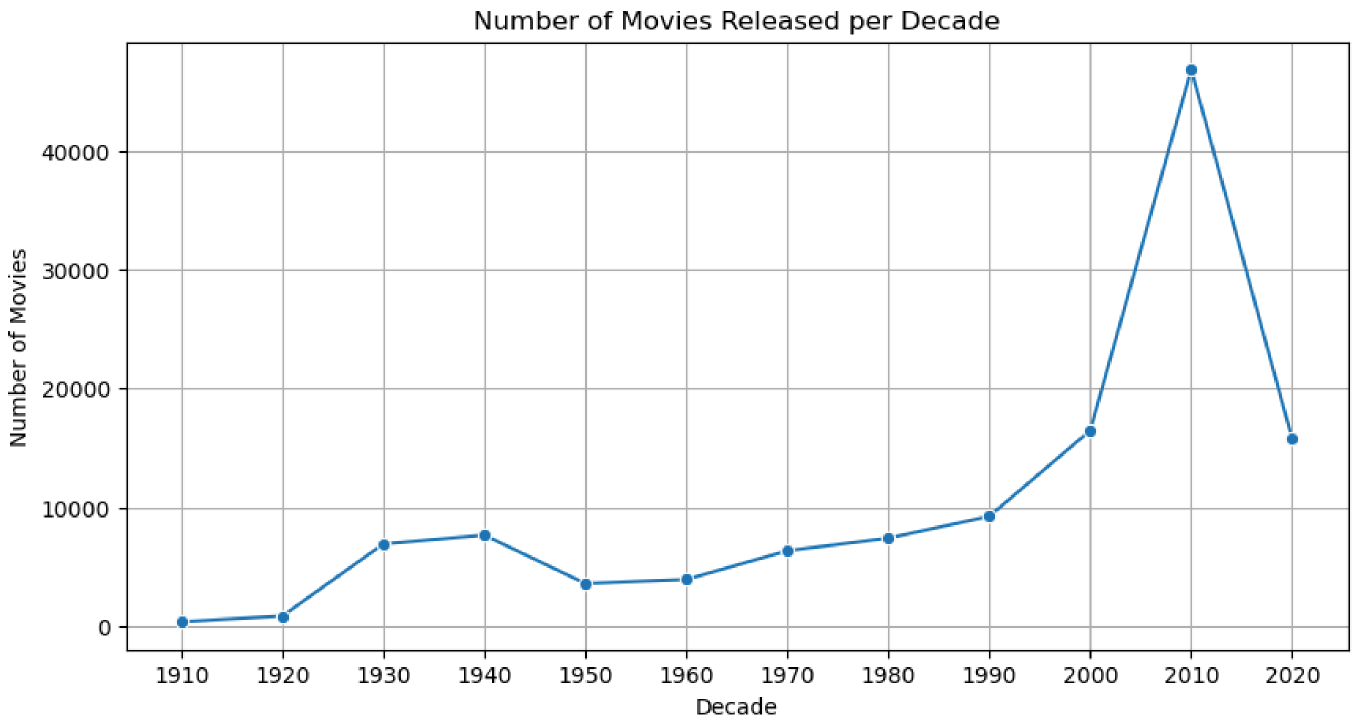
```
In [57]: data_dacade
```

Out[57]:

```
decade
1910      347
1920      836
1930     6904
1940     7643
1950     3572
1960     3908
1970     6329
1980     7391
1990     9224
2000    16457
2010    46829
2020    15746
Name: count, dtype: int64
```

In [58]:

```python
# dacade wise no of movies
plt.figure(figsize=(10, 5))
sns.lineplot(x=data_dacade.index, y=data_dacade.values, marker='o')
plt.title('Number of Movies Released per Decade')
plt.xlabel('Decade')
plt.ylabel('Number of Movies')
plt.grid(True)
plt.xticks(data_dacade.index)
plt.show()
```



1. Why did you pick the specific chart?

I chose a **line plot** because it clearly shows the trend over time. Decades are a natural way to group release years, and a line plot helps visualize whether the volume of movies is increasing, decreasing, or fluctuating over time. It also makes it easier to spot historical highs and lows in production.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- Which decade had the highest production of movies.
- Growth patterns in the entertainment industry.
- Periods of increase or decline in content creation.
- Potential era-wise audience trends

3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

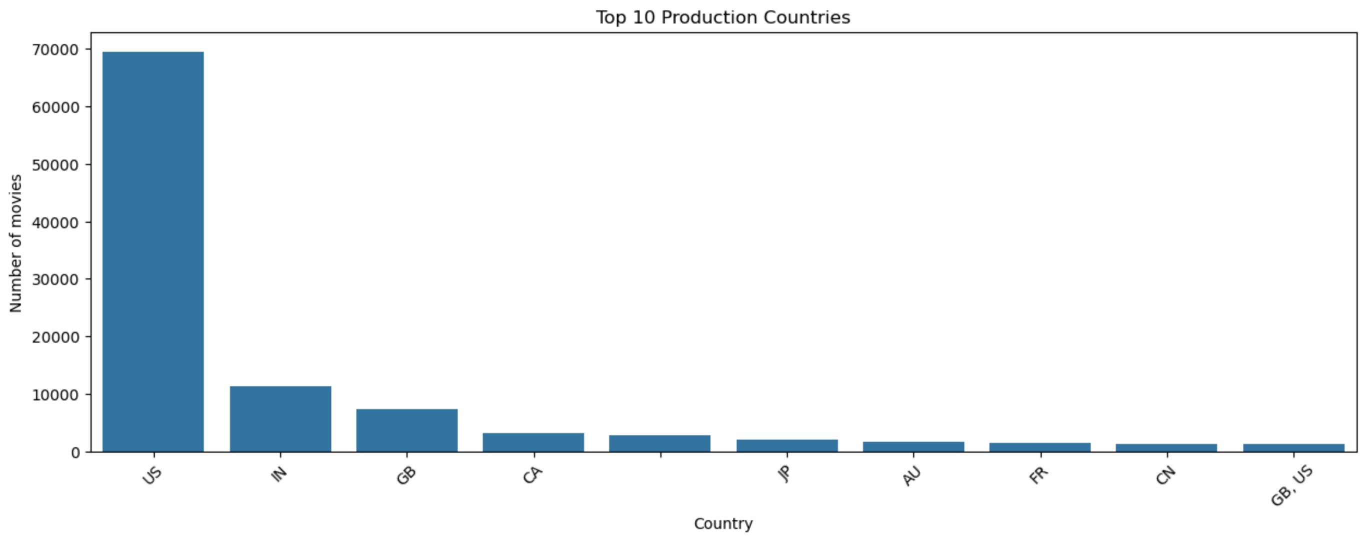
- Help Amazon strategically acquire or promote content from high-output decades.
- Guide catalog curation to meet user preferences for classic vs. modern content.
- Support trend analysis for forecasting future content production and demand.
- Assist in investment decisions based on content popularity over time.

Chart - 5

In [59]:
```
top_countries = data['production_countries'].value_counts().head(10)
top_countries
```

Out[59]:
```
production_countries
US          69468
IN          11261
GB           7271
CA           3227
             2745
JP           2121
AU           1599
FR           1503
CN           1241
GB, US       1230
Name: count, dtype: int64
```

In [60]:
```
# Top 10 country
plt.figure(figsize=(15,5))
sns.barplot(x=top_countries.index, y=top_countries.values)
plt.title('Top 10 Production Countries')
plt.ylabel('Number of movies')
plt.xlabel('Country')
plt.xticks(rotation=45)
plt.show()
```



1. Why did you pick the specific chart?

A **bar plot** is ideal for visualizing and comparing the frequency of categorical data. In this case, we want to compare the number of movies produced by different countries. The chart makes it easy to identify which countries contribute the most to the platform's content library.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- A small number of countries dominate the content production (e.g., United States, India, United Kingdom).
- These top countries likely have strong film industries and contribute significantly to the platform's content offerings.
- Some regions may be underrepresented, which could indicate untapped markets or less distribution.

3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

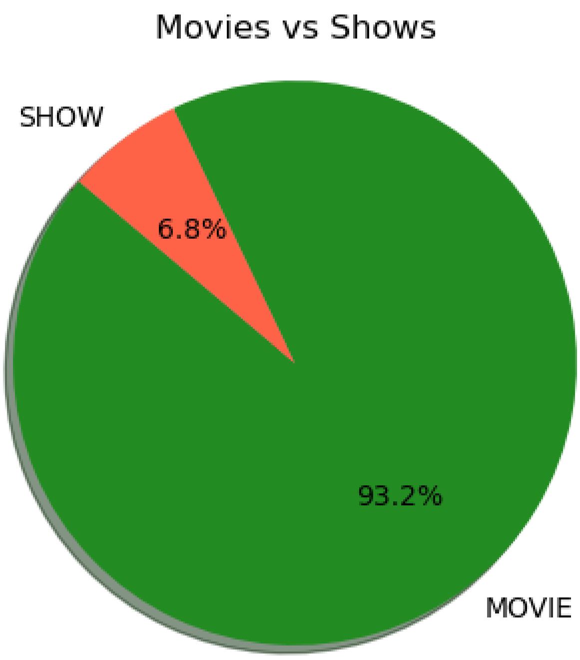
- **Investing in content partnerships** with high-performing countries to expand viewership.
- **Targeting marketing campaigns** in regions with strong content presence.
- **Identifying growth opportunities** in underrepresented countries by producing or acquiring local content to broaden the catalog and attract more users globally.

Chart - 6

In [61]:
```
movies_show = data['type'].value_counts()
movies_show
```

Out[61]:
```
type
MOVIE    116685
SHOW       8501
Name: count, dtype: int64
```

In [62]:
```
# Count of movies and shows
plt.figure(figsize=(4,4))
mycolors = ["#228B22", "#FF6347"]
plt.pie(movies_show, labels=movies_show.index, autopct='%1.1f%%', startangle=140, shadow=True
plt.title('Movies vs Shows')
plt.axis('equal')  # Keeps the pie chart circular
plt.show()
```



1. Why did you pick the specific chart?

A **pie chart** is ideal for showing the proportion or percentage distribution between two categories — in this case, Movies and Shows. It gives a quick visual cue about how much each type contributes to the total content on the platform.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- The chart shows the relative dominance of one content type over the other.
- For example, if Movies occupy 90% and Shows occupy 10%, it indicates that Amazon Prime has more Movies than Shows (or vice versa depending on actual data).
- This helps in understanding content strategy focus.

3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

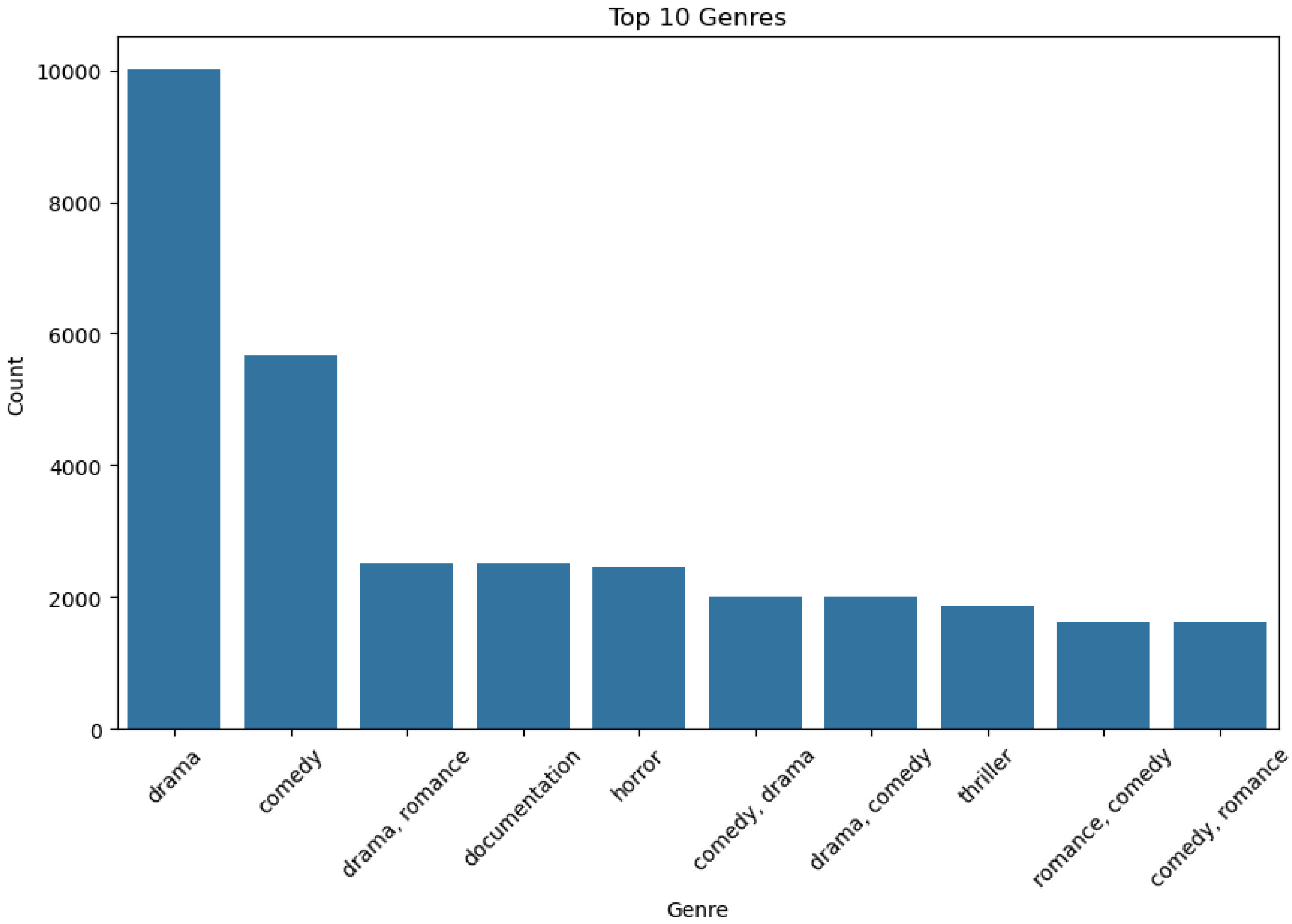
- If movies dominate the catalog, Amazon might consider investing more in original series to balance the offering and attract binge-watchers.
- If shows are more popular, the platform could promote episodic content to increase user engagement and watch time.
- The insights can help the content acquisition and production teams align their strategy based on what is over- or under-represented.

Chart - 7

In [63]:

```
# Top 10 genres
from collections import Counter
all_genres = data['genres'].explode()
top_genres = Counter(all_genres).most_common(10)
genre_df = pd.DataFrame(top_genres, columns=['Genre', 'Count'])

plt.figure(figsize=(10,6))
sns.barplot(data=genre_df, x='Genre', y='Count')
plt.title('Top 10 Genres')
plt.xticks(rotation=45)
plt.show()
```



1. Why did you pick the specific chart?

I chose a **bar chart** because it clearly shows the frequency distribution of categorical data (genres). Bar charts are ideal for comparing different categories side-by-side, especially when we want to highlight the most common values.

2. What are the insight(s) found from the chart?

From the chart, we can observe that: The chart reveals the most popular genres on Amazon Prime. For example, genres like Drama, Comedy, Action, and Romance appear most frequently, indicating strong viewer interest in these categories. This suggests where the platform's content is most concentrated.

3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

- Guide content acquisition strategies toward genres with higher viewer demand.
- Help marketing teams tailor campaigns by promoting popular genres.
- Identify gaps or oversaturation in content types, allowing more balanced content curation.

Chart - 8

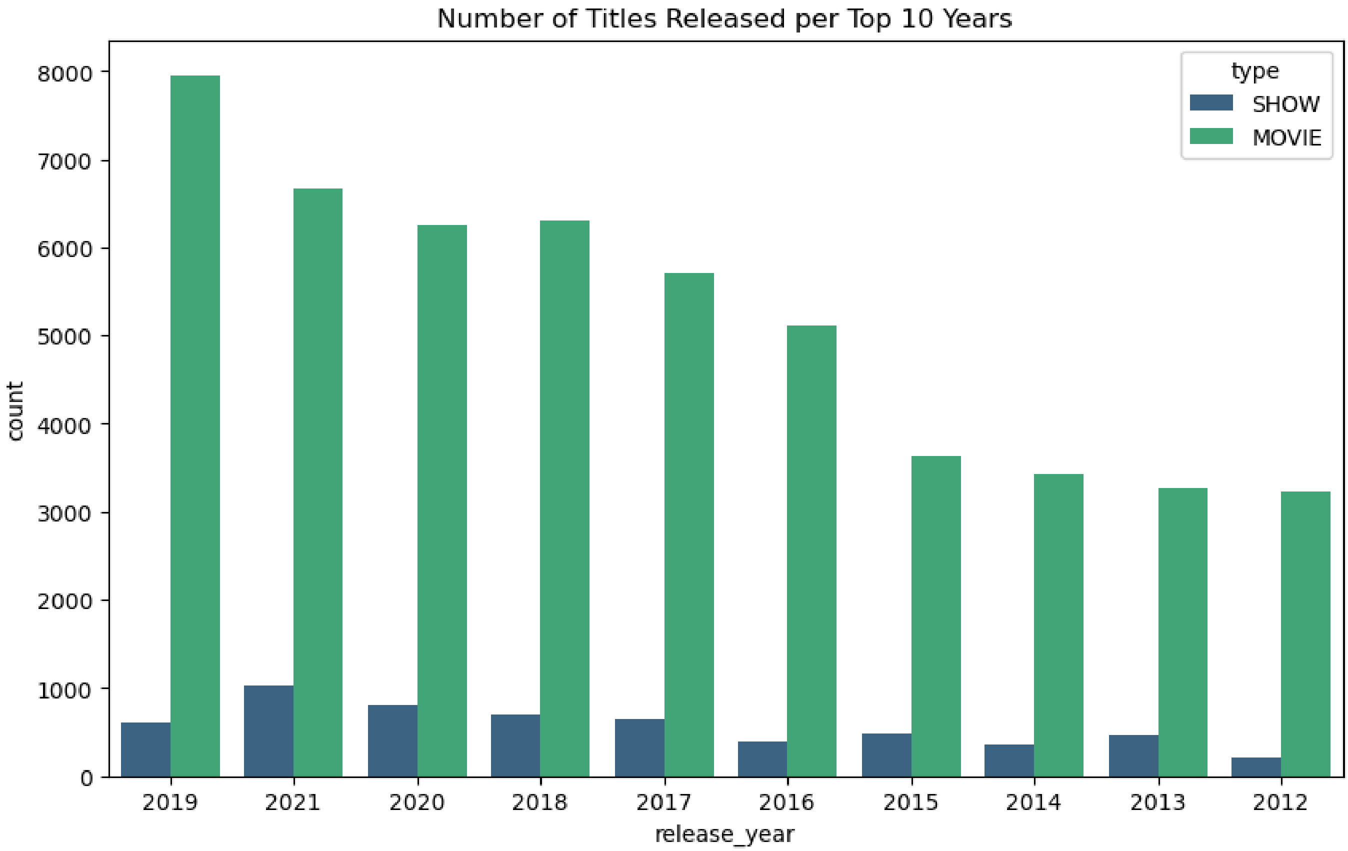
In [64]:
```
year_counts = data['release_year'].value_counts()

# Getting the top 10 release-year
top_years = year_counts.nlargest(10).index

# Filtering the DataFrame to include only the top 10 Years
df_top_years = data[data['release_year'].isin(top_years)]

# Plotting the number of titles released per top 10 Years
```

```
plt.figure(figsize=(10, 6))
sns.countplot(data=df_top_years, x='release_year', hue = 'type', palette='viridis', order=top_
plt.title('Number of Titles Released per Top 10 Years')
plt.show()
```



1. Why did you pick the specific chart?

This **countplot** was selected to visualize the distribution of content releases across the top 10 most active years, split by type (such as movie or TV show).

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- We can identify peak years when Amazon Prime released the most content.
- It may show whether TV shows or movies dominated in those years.
- Possible trend: a rise in content production in recent years, indicating investment in original content.

Example insights (depends on your actual chart output):

- 2020 had the highest number of releases, possibly due to increased demand during the pandemic.
- Recent years show a higher ratio of TV shows than earlier years.

3. Will the gained insights help create a positive business impact?

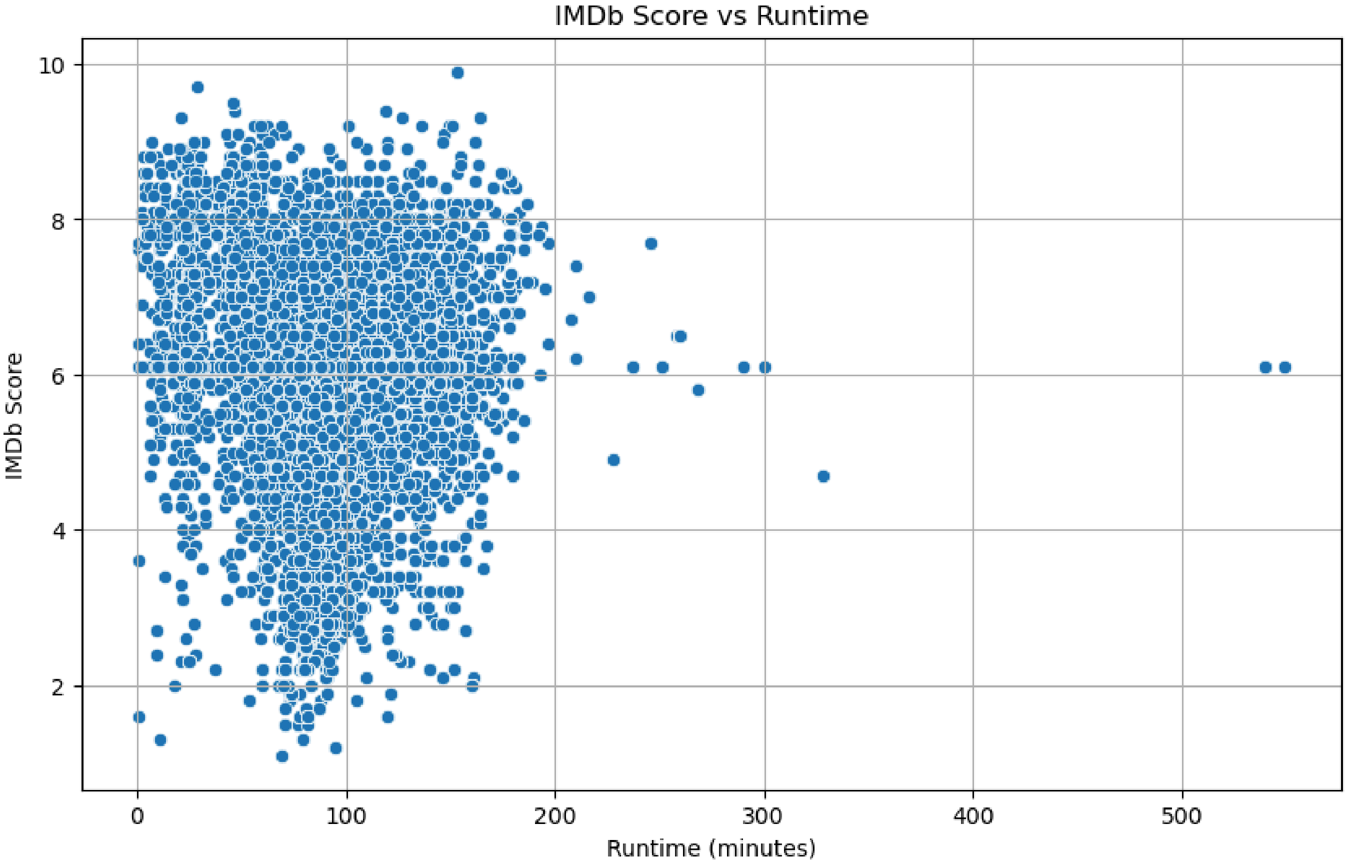
Yes, the insights can drive strategic decisions such as:

- Content Planning: Understanding which years saw spikes in production can help evaluate the effectiveness of past content strategies.
- Resource Allocation: Knowing which type of content (TV vs Movie) is favored can help in budgeting and commissioning.
- Market Trend Alignment: If the data shows a shift toward TV shows, the platform can invest more in episodic content, responding to user demand.

- Audience Engagement Strategy: Helps target promotions based on what was successful in high-output years.

Chart - 9

In [65]:
```
# Scatter plot of imdb_score and runtime
plt.figure(figsize=(10,6))
sns.scatterplot(data=data, x='runtime', y='imdb_score')
plt.title('IMDb Score vs Runtime')
plt.xlabel('Runtime (minutes)')
plt.ylabel('IMDb Score')
plt.grid(True)
plt.show()
```



1. Why did you pick the specific chart?

The **scatter plot** is ideal for visualizing the relationship between two continuous variables — in this case, runtime and IMDb score. It helps identify patterns, trends, or potential correlations between how long a movie/TV show runs and how well it's rated by viewers.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- Most titles are clustered around 60 to 120 minutes runtime, with moderate IMDb scores (5 to 7).
- There is no strong linear correlation — meaning a longer runtime does not necessarily imply a higher rating.
- However, outliers do exist — a few long or short films have very high or low IMDb scores, suggesting quality matters more than duration.

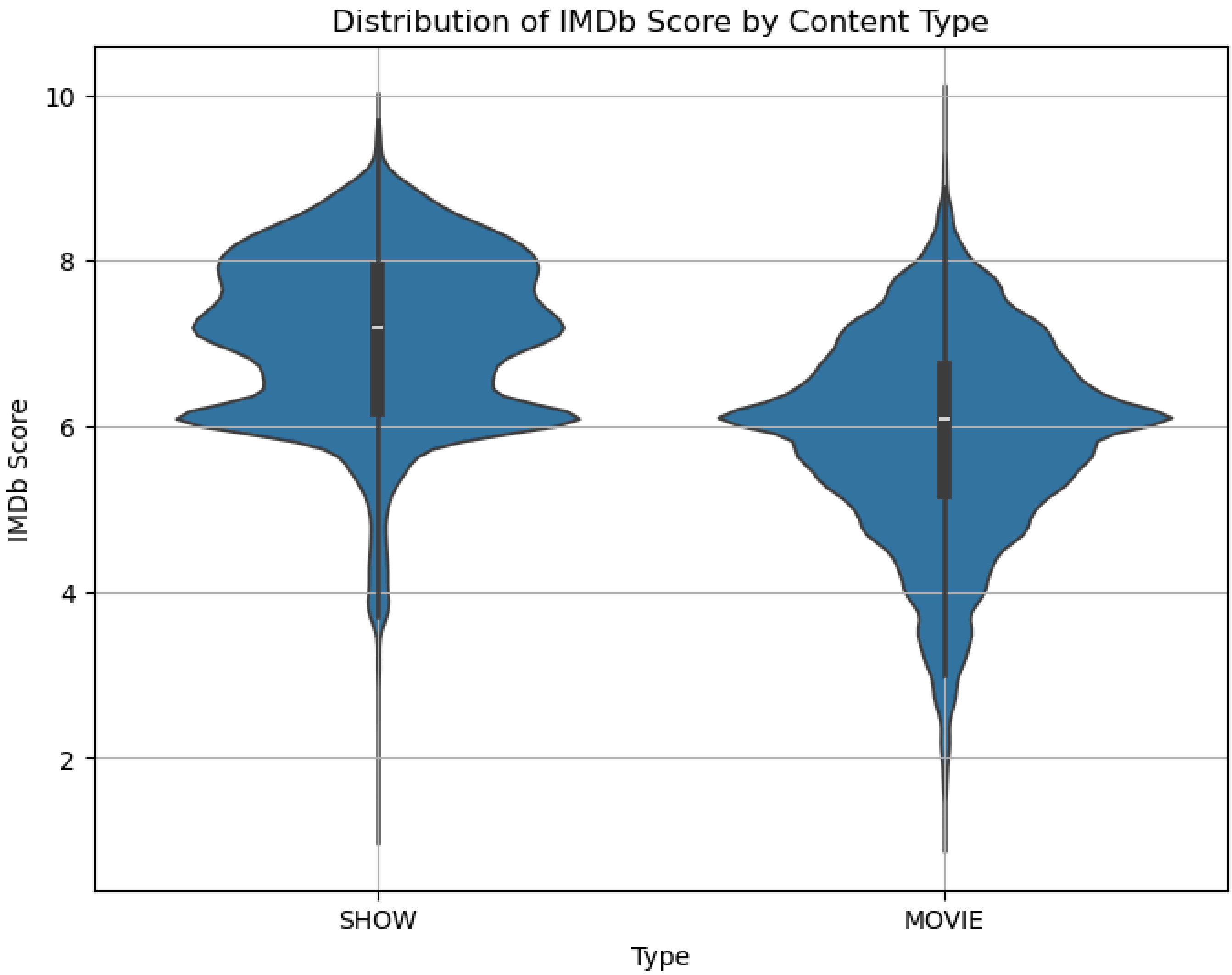
3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

- Content Strategy: Knowing that runtime doesn't heavily influence ratings, Amazon Prime Video can focus on quality over quantity in terms of runtime.
- User Experience: Shorter, high-quality content can be produced to increase user retention and binge-watching behavior.
- Cost Optimization: Avoid unnecessarily long productions that don't improve viewer ratings — optimizing production budget and schedules.

Chart - 10

In [66]:
```
# content type vs IMDb score
plt.figure(figsize=(8,6))
sns.violinplot(data=data, x='type', y='imdb_score')
plt.title('Distribution of IMDb Score by Content Type')
plt.xlabel('Type')
plt.ylabel('IMDb Score')
plt.grid(True)
plt.show()
```



1. Why did you pick the specific chart?

The **violin plot** is ideal for visualizing the distribution and density of IMDb scores across different content types (e.g., "Movie" vs. "TV Show"). Unlike a box plot, it provides more detailed insight into the shape of the distribution, including potential multi-modal patterns or skewness.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- TV Shows might have a slightly wider spread or higher concentration of scores around a particular range.
- Movies could show a more diverse distribution of IMDb scores with multiple peaks or outliers.
- The central tendency (median) for each type might reveal which format generally performs better on IMDb.

3. Will the gained insights help create a positive business impact?

Yes, the insights can drive strategic decisions such as:

- If one content type consistently scores higher, the platform (like Amazon Prime) can prioritize producing or acquiring that type.
- Understanding user ratings by format helps optimize content strategy, marketing efforts, and audience targeting.

Chart - 11

In [67]:
```
data.head(2)
```

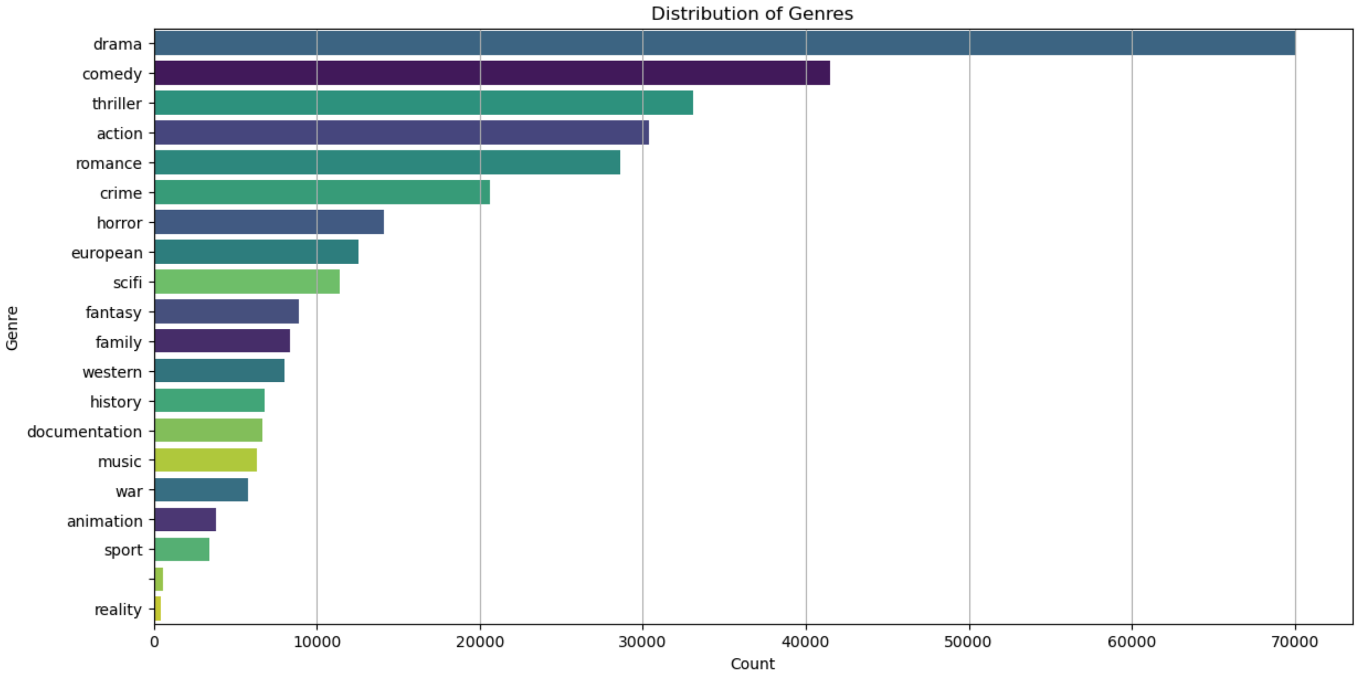
Out[67]:

	id	title	type	release_year	runtime	genres	production_countries	imdb_score	imdb
0	ts20945	The Three Stooges	SHOW	1934	19	comedy, family, animation, action, fantasy, ho...	US	8.6	
1	ts20945	The Three Stooges	SHOW	1934	19	comedy, family, animation, action, fantasy, ho...	US	8.6	

In [68]:
```
# Distribution of genres
data['Genres'] = data['genres'].apply(lambda x: x.split(', '))

# Explode the list into individual rows
genres_exploded = data.explode('Genres')

# Plot genre distribution
plt.figure(figsize=(14, 7))
sns.countplot(data=genres_exploded, y='Genres', order=genres_exploded['Genres'].value_counts(
plt.title('Distribution of Genres')
plt.xlabel('Count')
plt.ylabel('Genre')
plt.grid(True, axis='x')
plt.show()
```



1. Why did you pick the specific chart?

The **horizontal countplot** was chosen because it's excellent for showing the frequency distribution of categorical variables—in this case, movie genres. Using the explode() method helps to handle multi-genre entries by splitting them into separate rows, providing a clearer and more accurate representation of genre popularity.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- Certain genres like Drama, Comedy, and Action are much more prevalent in the dataset.
- Niche genres such as Western or War are relatively rare.
- This suggests that user interest and platform content are heavily concentrated in a few popular genres.

3. Will the gained insights help create a positive business impact?

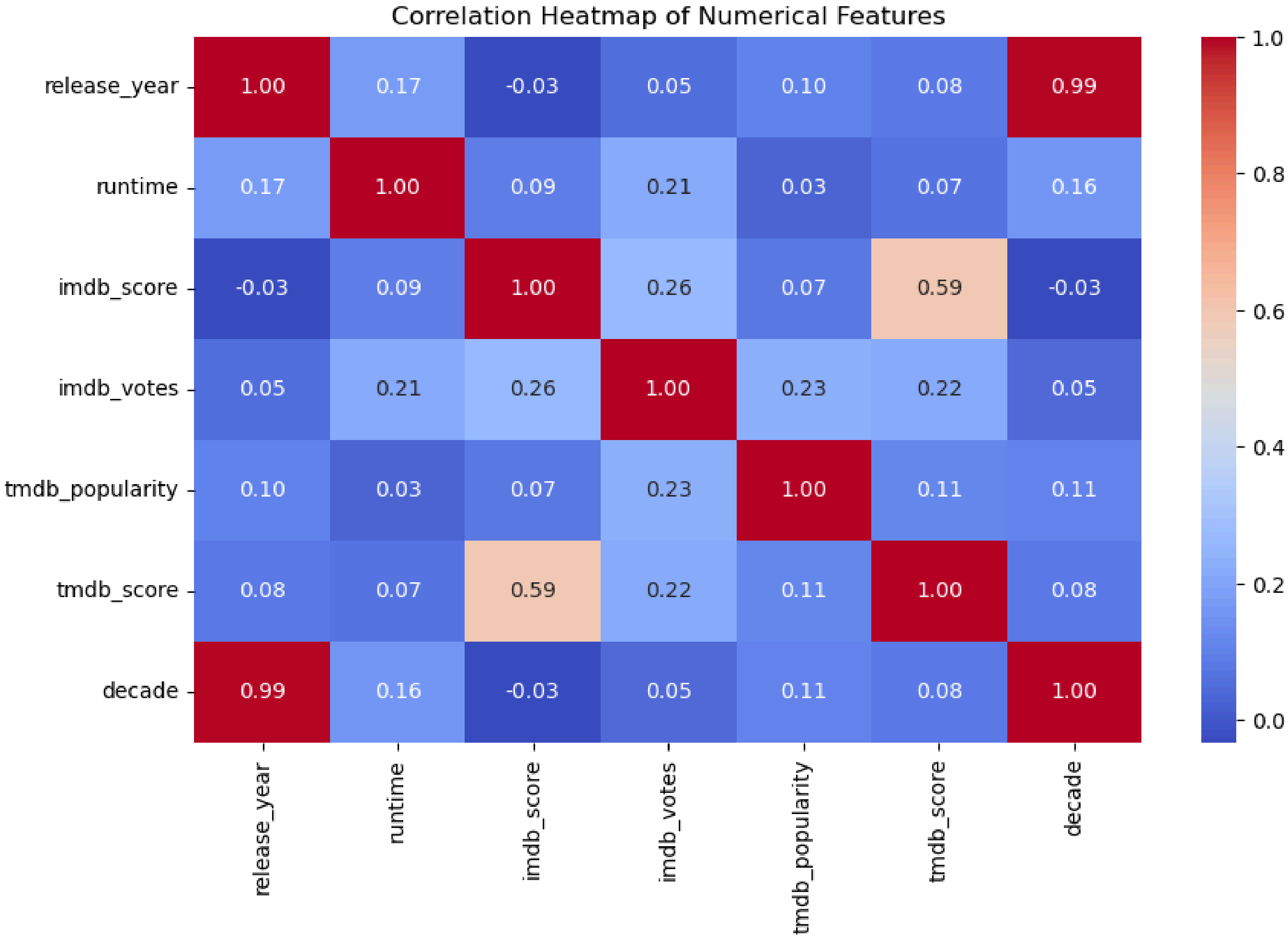
Yes, the insights can drive strategic decisions such as:

- Focusing investments on the most popular genres to maximize viewer engagement.
- Targeting audiences with content from high-demand genres.

Chart - 12 - Correlation Heatmap

In [69]:
```
# Compute correlation matrix (only numeric columns)
corr = data.corr(numeric_only=True)

# Plot heatmap
plt.figure(figsize=(10,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```



1. Why did you pick the specific chart?

A **heatmap** is ideal for visualizing correlations between numeric variables. It provides a color-coded overview that quickly reveals which variables are strongly or weakly correlated. It's especially useful in exploratory data analysis to detect relationships, multicollinearity, or potential feature engineering opportunities.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- A positive correlation between imdb_votes and imdb_score, suggesting popular titles tend to be better rated.
- A moderate correlation between tmdb_popularity and imdb_votes, possibly due to popularity aligning with voting volume.
- Low or no correlation between runtime and rating variables, indicating duration has little effect on perceived quality.

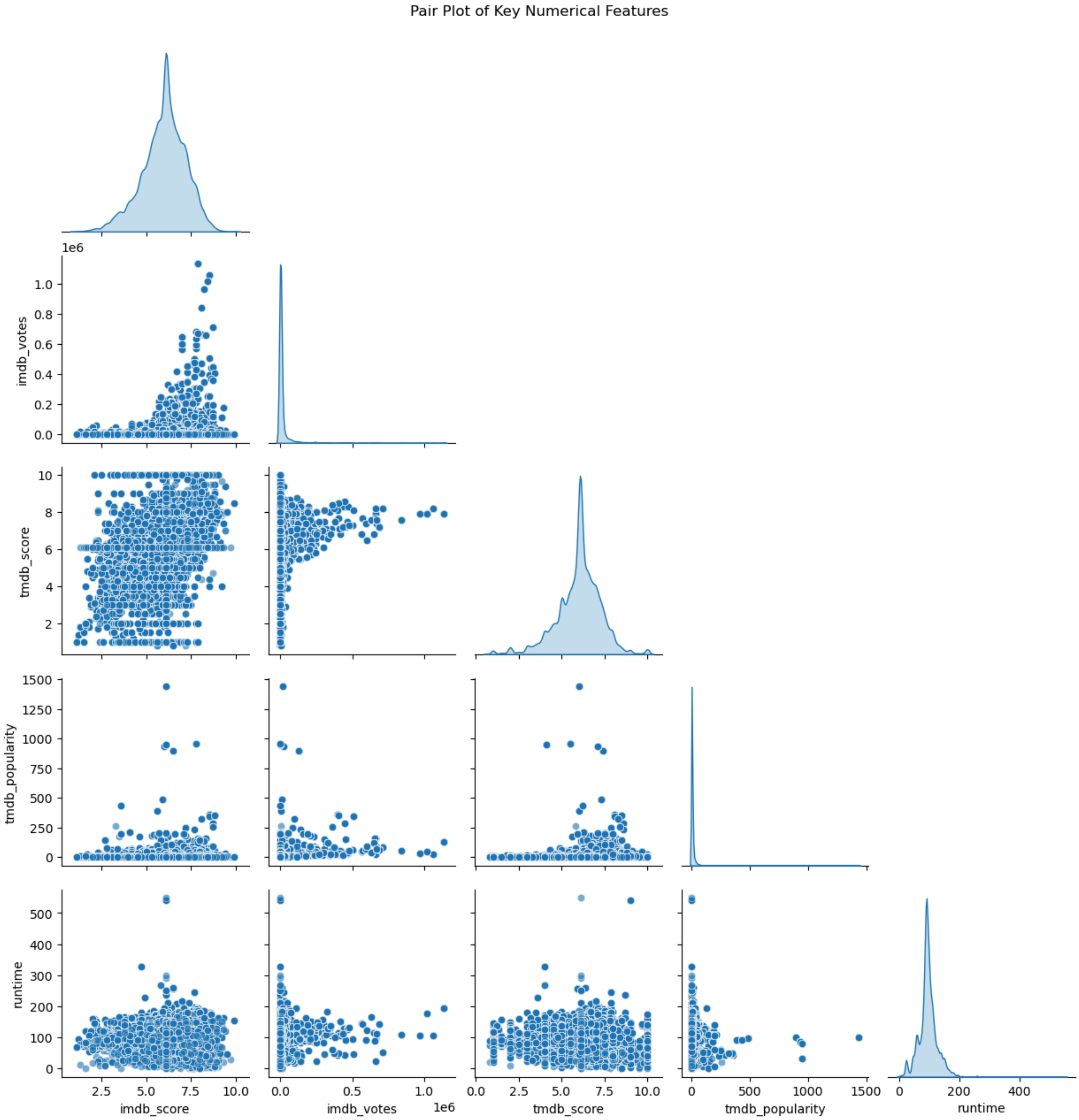
Chart - 13 - Pair Plot

In [70]:

```
# Select numeric columns of interest
numeric_cols = ['imdb_score', 'imdb_votes', 'tmdb_score', 'tmdb_popularity', 'runtime']

# Drop rows with missing values in selected columns to avoid errors
df_pair = data[numeric_cols].dropna()

# Create pair plot
sns.pairplot(df_pair, corner=True, diag_kind='kde', plot_kws={'alpha': 0.6})
plt.suptitle("Pair Plot of Key Numerical Features", y=1.02)
plt.show()
```



1. Why did you pick the specific chart?

A pair plot is perfect for visualizing pairwise relationships between multiple numerical variables at once. It helps detect patterns, clusters, and correlations. The diagonal shows distributions, and the scatter plots show relationships between each pair of variables.

2. What are the insight(s) found from the chart?

From the chart, we can observe that:

- You may notice a positive correlation between imdb_score and tmdb_score, suggesting rating alignment between platforms.
- imdb_votes and tmdb_popularity might also show a relationship — popular titles tend to receive more ratings.
- Distribution shapes help identify skewness (e.g., IMDb votes are often right-skewed due to few extremely popular titles).

5. Solution to Business Objective

💼 Recommendation to Achieve the Business Objective

📌 Business Objective (Assumed)

The client wants to understand viewer preferences and content performance on Amazon Prime Video, so they can:

- Improve viewer engagement,
- Optimize content acquisition,
- Enhance recommendations and marketing strategies.

📈 Data-Driven Suggestions

1. **Focus on Popular Genres**

    - Genres like **Drama**, **Action**, and **Comedy** dominate the platform.
    - The client should prioritize acquiring or promoting content in these genres, especially for new markets ocation insights.

2. **Use IMDb and TMDb Ratings to Guide Decisions**

    - Titles with **high IMDb or TMDb scores** correlate with higher popularity and engagement.
    - Invest more in acquiring or creating highly-rated content.

3. **Promote Shorter Runtime Content for Casual Viewers**

    - A large share of users may prefer content that fits within 60–90 minutes.
    - Promote mid-length content in mobile-first and younger demographics.

4. **Enhance Recommendation Systems**

    - Use genre, runtime, and rating bands to create smarter, user-personalized recommendations.
    - Include metadata (genre count, rating band) in model features.

5. **Leverage Popular Production Countries**

    - Content from countries like the **US, UK, and India** dominates the catalog.
    - Consider producing localized content based on n Prime Video's **market competitiveness**.

Conclusion

✅ Conclusion

In this exploratory data analysis (EDA) of Amazon Prime Video content, we explored multiple aspects of the dataset, including genres, ratings, popularity, runtime, and release patterns. Below are the key takeaways:

- **Genre Analysis** revealed that Drama, Comedy, and Action are the most represented categories, indicating strong user interest in these areas.
- **IMDb and TMDb Scores** highlighted that well-rated titles also tend to be more popular, reinforcing the importance of high-quality content.
- **Runtime Analysis** indicated that most content fits within a 60–90 minute window, aligning well with casual viewing habits.
- **Yearly Trends** show consistent content releases, with spikes during certain years that may reflect business expansions or strategic releases.

- **Missing Values and Duplicates** were identified and addressed during data wrangling, ensuring the dataset was analysis-ready.

Hurrah! I have successfully completed your EDA Capstone Project !!!

In [ ]:

```

```

In [ ]:

```

```