Table of Contents

1 - Introduction

🌟 Problem Statement

ZEE5, a prominent OTT streaming platform, aims to enhance user retention and engagement by delivering **highly personalized movie recommendations**. With a vast and ever-growing content library, the challenge lies in effectively identifying and surfacing movies that align with individual user preferences. The objective is to develop a **data-driven recommendation engine** that leverages behavioral data, collaborative patterns, and latent representations to drive intelligent content discovery.

🎯 Project Goal

To build a scalable and accurate **personalized movie recommender system** using collaborative filtering and matrix factorization techniques — enabling tailored content delivery based on user preferences and community trends.

✅ Key Objectives

- Load, preprocess, and merge user, movie, and rating datasets
- Conduct in-depth **exploratory data analysis** (EDA) to uncover user behavior, genre trends, and temporal engagement
- Build **Item-based Collaborative Filtering** models using **Pearson Correlation** and **Cosine Similarity**
- Train a **Matrix Factorization model** (SVD) using the `Surprise` library to uncover latent patterns
- Visualize learned embeddings (via PCA and t-SNE) to interpret similarity and genre clusters
- Evaluate model performance using **RMSE** and **MAPE**
- Implement a **User-based Collaborative Filtering** approach to simulate cold-start use cases
- Conclude with a **comparative recommendation analysis** across all modeling techniques
- Provide **business-driven insights** and recommendations for implementation

💼 Business Impact

A successful recommender system can:

- Enhance user satisfaction through relevant and timely suggestions
- Improve **watch time**, **click-through rates**, and **session length**
- Boost **content visibility** and reduce user churn
- Enable **targeted content marketing** and platform personalization at scale

🧠 Techniques Explored

This case study implements the following approaches:

- **Item-Based Collaborative Filtering**
  - Pearson Correlation
  - Cosine Similarity (KNN-based)
- **Matrix Factorization**
  - Singular Value Decomposition (SVD)
  - Embedding-based similarity via latent factors
- **Optional Extension**
  - User-Based Collaborative Filtering
  - Fuzzy Matching for Robust Input Handling
  - Hybrid Recommendation Analysis

🩹 Evaluation Metrics

- **RMSE (Root Mean Squared Error)** — measures prediction accuracy
- **MAPE (Mean Absolute Percentage Error)** — measures percentage-based error
- **Qualitative Recommendation Relevance**
- **Exploratory Insights & Visualization Quality**
- **Coverage, Novelty & Interpretability of Results**

2 - Imports and Data Loading

In [1]:
```
# !pip install scikit-surprise
```

In [2]:
```
# Core data handling
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessing & modeling
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
from sklearn.neighbors import NearestNeighbors

# For matrix factorization
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from surprise import accuracy

# System & warnings
import warnings
warnings.filterwarnings('ignore')

# Plot settings
sns.set(style='whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)
```

In [3]:
```
# Define column names
user_columns = ['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code']
movie_columns = ['MovieID', 'Title', 'Genres']
rating_columns = ['UserID', 'MovieID', 'Rating', 'Timestamp']


# Load each dataset (delimiter is '::') with skiprows=1 to ignore header row inside the data
users = pd.read_csv(r'M:\Business Cases\12 - Zee\Datasets\zee-users.dat', sep='::', names=user_columns, engine='python', skiprows=1)
movies = pd.read_csv(r'M:\Business Cases\12 - Zee\Datasets\zee-movies.dat', sep='::', names=movie_columns, engine='python', encoding='ISO-8859-1', skiprows=1)
ratings = pd.read_csv(r'M:\Business Cases\12 - Zee\Datasets\zee-ratings.dat', sep='::', names=rating_columns, engine='python', skiprows=1)


# Preview the data
print("Users:\n", users.head(), "\n")
print("Movies:\n", movies.head(), "\n")
print("Ratings:\n", ratings.head())
```

```
Users:
    UserID Gender  Age  Occupation Zip-code
0       1      F    1          10    48067
1       2      M   56          16    70072
2       3      M   25          15    55117
3       4      M   45           7    02460
4       5      M   25          20    55455

Movies:
    MovieID                               Title                        Genres
0        1                    Toy Story (1995)   Animation|Children's|Comedy
1        2                      Jumanji (1995)  Adventure|Children's|Fantasy
2        3             Grumpier Old Men (1995)                Comedy|Romance
3        4            Waiting to Exhale (1995)                  Comedy|Drama
4        5  Father of the Bride Part II (1995)                       Comedy

Ratings:
    UserID  MovieID  Rating  Timestamp
0       1     1193       5  978300760
1       1      661       3  978302109
2       1      914       3  978301968
3       1     3408       4  978300275
4       1     2355       5  978824291
```

Dataset Overview

The dataset is derived from the [MovieLens 1M Dataset], customized for ZEE5's movie catalog. It includes the following files:

1. `users.dat`

- Format: `UserID::Gender::Age::Occupation::Zip-code`
- Contains demographic information about each user

2. `movies.dat`

- Format: `MovieID::Title::Genres`
- Contains metadata about movies, including genre and title

3. `ratings.dat`

- Format: `UserID::MovieID::Rating::Timestamp`
- Contains explicit ratings (1–5) that users gave to movies
- Each user has rated at least 20 movies

File Name	Description
`zee-users.dat`	User demographic data
`zee-movies.dat`	Movie metadata with genres
`zee-ratings.dat`	User ratings and timestamps

3 - Data Cleaning & Preprocessing

In [4]:
```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   MovieID  3883 non-null   int64 
 1   Title    3883 non-null   object
 2   Genres   3883 non-null   object
dtypes: int64(1), object(2)
memory usage: 91.1+ KB
```

In [5]:
```
users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   UserID      6040 non-null   int64 
 1   Gender      6040 non-null   object
 2   Age         6040 non-null   int64 
 3   Occupation  6040 non-null   int64 
 4   Zip-code    6040 non-null   object
dtypes: int64(3), object(2)
memory usage: 236.1+ KB
```

In [6]:
```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype 
---  ------     --------------    ----- 
 0   UserID     1000209 non-null  int64 
 1   MovieID    1000209 non-null  int64 
 2   Rating     1000209 non-null  int64 
 3   Timestamp  1000209 non-null  int64 
dtypes: int64(4)
memory usage: 30.5 MB
```

3.1 - Clean `movies`, `ratings`, `users`

3.1.1 - Cleaning `movies`

In [7]:
```
# Movie Metadata Cleanup: Extract Year, Clean Titles & Parse Genres

# Extract year from movie title using regex
movies['Year'] = movies['Title'].str.extract(r'\((\d{4})\)', expand=False)

# Remove the year from the title
movies['Title'] = movies['Title'].str.replace(r'\(\d{4}\)', '', regex=True).str.strip()

# Split genres into list
movies['Genres'] = movies['Genres'].apply(lambda x: x.split('|') if isinstance(x, str) else [])

# Convert Year to numeric
movies['Year'] = pd.to_numeric(movies['Year'], errors='coerce')

# Preview the cleaned movie data
print(movies.info())
movies.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   MovieID  3883 non-null   int64 
 1   Title    3883 non-null   object
 2   Genres   3883 non-null   object
 3   Year     3883 non-null   int64 
dtypes: int64(2), object(2)
memory usage: 121.5+ KB
None
```

Out[7]:

	MovieID	Title	Genres	Year
0	1	Toy Story	[Animation, Children's, Comedy]	1995
1	2	Jumanji	[Adventure, Children's, Fantasy]	1995
2	3	Grumpier Old Men	[Comedy, Romance]	1995
3	4	Waiting to Exhale	[Comedy, Drama]	1995
4	5	Father of the Bride Part II	[Comedy]	1995

3.1.2 - Cleaning `ratings`

In [8]:
```
# --- Clean Ratings ---
# Convert Rating to integer
ratings['Rating'] = ratings['Rating'].astype(int)

# Convert Timestamp to datetime
ratings['Timestamp'] = pd.to_datetime(ratings['Timestamp'], unit='s')


# Preview cleaned Ratings Data
print("Ratings:\n", ratings.head(), "\n")
```

```
Ratings:
    UserID  MovieID  Rating           Timestamp
0       1     1193       5 2000-12-31 22:12:40
1       1      661       3 2000-12-31 22:35:09
2       1      914       3 2000-12-31 22:32:48
3       1     3408       4 2000-12-31 22:04:35
4       1     2355       5 2001-01-06 23:38:11
```

3.1.3 - Cleaning `users`

In [9]:
```
# --- Clean Users ---
# Optional: Map Age and Occupation to labels (for readability)
age_map = {
    1: "Under 18", 18: "18-24", 25: "25-34", 35: "35-44",
    45: "45-49", 50: "50-55", 56: "56+"
}
occupation_map = {
    0: "Other", 1: "Academic/Educator", 2: "Artist", 3: "Clerical/Admin",
    4: "College/Grad Student", 5: "Customer Service", 6: "Doctor/Healthcare",
    7: "Executive/Managerial", 8: "Farmer", 9: "Homemaker", 10: "K-12 Student",
    11: "Lawyer", 12: "Programmer", 13: "Retired", 14: "Sales/Marketing",
    15: "Scientist", 16: "Self-Employed", 17: "Technician/Engineer",
    18: "Tradesman/Craftsman", 19: "Unemployed", 20: "Writer"
}
```

```
users['Age'] = users['Age'].astype(int).map(age_map)
users['Occupation'] = users['Occupation'].astype(int).map(occupation_map)

# Preview cleaned Users Data
print("Users:\n", users.head())
```

```
Users:
    UserID Gender       Age            Occupation Zip-code
0       1      F  Under 18          K-12 Student    48067
1       2      M       56+         Self-Employed    70072
2       3      M     25-34             Scientist    55117
3       4      M     45-49  Executive/Managerial    02460
4       5      M     25-34                Writer    55455
```

3.2 - Merge Datasets

In [10]:
```
# Merge ratings with movies
df = pd.merge(ratings, movies, on='MovieID', how='inner')

# Merge result with users
df = pd.merge(df, users, on='UserID', how='inner')

# Preview merged dataset
print("Final Dataset Shape:", df.shape)
df.head()
```

```
Final Dataset Shape: (1000209, 11)
```

Out[10]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Year	Gender	Age	Occupation	Zip-code
0	1	1193	5	2000-12-31 22:12:40	One Flew Over the Cuckoo's Nest	[Drama]	1975	F	Under 18	K-12 Student	48067
1	1	661	3	2000-12-31 22:35:09	James and the Giant Peach	[Animation, Children's, Musical]	1996	F	Under 18	K-12 Student	48067
2	1	914	3	2000-12-31 22:32:48	My Fair Lady	[Musical, Romance]	1964	F	Under 18	K-12 Student	48067
3	1	3408	4	2000-12-31 22:04:35	Erin Brockovich	[Drama]	2000	F	Under 18	K-12 Student	48067
4	1	2355	5	2001-01-06 23:38:11	Bug's Life, A	[Animation, Children's, Comedy]	1998	F	Under 18	K-12 Student	48067

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 11 columns):
 #   Column      Non-Null Count    Dtype         
---  ------      --------------    -----         
 0   UserID      1000209 non-null  int64         
 1   MovieID     1000209 non-null  int64         
 2   Rating      1000209 non-null  int32         
 3   Timestamp   1000209 non-null  datetime64[ns]
 4   Title       1000209 non-null  object        
 5   Genres      1000209 non-null  object        
 6   Year        1000209 non-null  int64         
 7   Gender      1000209 non-null  object        
 8   Age         1000209 non-null  object        
 9   Occupation  1000209 non-null  object        
 10  Zip-code    1000209 non-null  object        
dtypes: datetime64[ns](1), int32(1), int64(3), object(6)
memory usage: 80.1+ MB
```

3.3 - Feature Engineering: Year, Genres, Timestamps

In [12]:
```
# Ensure Year is integer
df['Year'] = pd.to_numeric(df['Year'], errors='coerce').astype('Int64')
```

In [12]:
```
# Extract datetime components
df['WatchDate'] = df['Timestamp']
df['WatchYear'] = df['WatchDate'].dt.year
df['WatchMonth'] = df['WatchDate'].dt.month
df['WatchHour'] = df['WatchDate'].dt.hour
df['WatchDay'] = df['WatchDate'].dt.day_name()

# Create Decade of Release (e.g., 1990s, 2000s)
df['ReleaseDecade'] = (df['Year'] // 10 * 10).astype('Int64').astype(str) + 's'

# Preview
df[['Title', 'Genres', 'Year', 'ReleaseDecade', 'WatchDate', 'WatchDay', 'WatchHour']].head()
```

Out[12]:

	Title	Genres	Year	ReleaseDecade	WatchDate	WatchDay	WatchHour
0	One Flew Over the Cuckoo's Nest	[Drama]	1975	1970s	2000-12-31 22:12:40	Sunday	22
1	James and the Giant Peach	[Animation, Children's, Musical]	1996	1990s	2000-12-31 22:35:09	Sunday	22
2	My Fair Lady	[Musical, Romance]	1964	1960s	2000-12-31 22:32:48	Sunday	22
3	Erin Brockovich	[Drama]	2000	2000s	2000-12-31 22:04:35	Sunday	22
4	Bug's Life, A	[Animation, Children's, Comedy]	1998	1990s	2001-01-06 23:38:11	Saturday	23

3.4 - Check Missing/Invalid Values

In [13]:
```
# Check for missing values
missing_summary = pd.DataFrame({
    'Missing Values': df.isnull().sum(),
    'Percentage': (df.isnull().sum() / len(df)) * 100
})
missing_summary = missing_summary[missing_summary['Missing Values'] > 0]
missing_summary.sort_values(by='Percentage', ascending=False)
```

Out[13]:

	Missing Values	Percentage

3.5 - Final Dataset Overview

- After data cleaning, parsing, and merging the user, movie, and ratings datasets, we now have a unified dataset ready for analysis and modeling.
- The structure includes metadata such as genres, user demographics, and time-based features.

Below is a structural overview of the final dataset:

In [14]:
```
df.info()
```

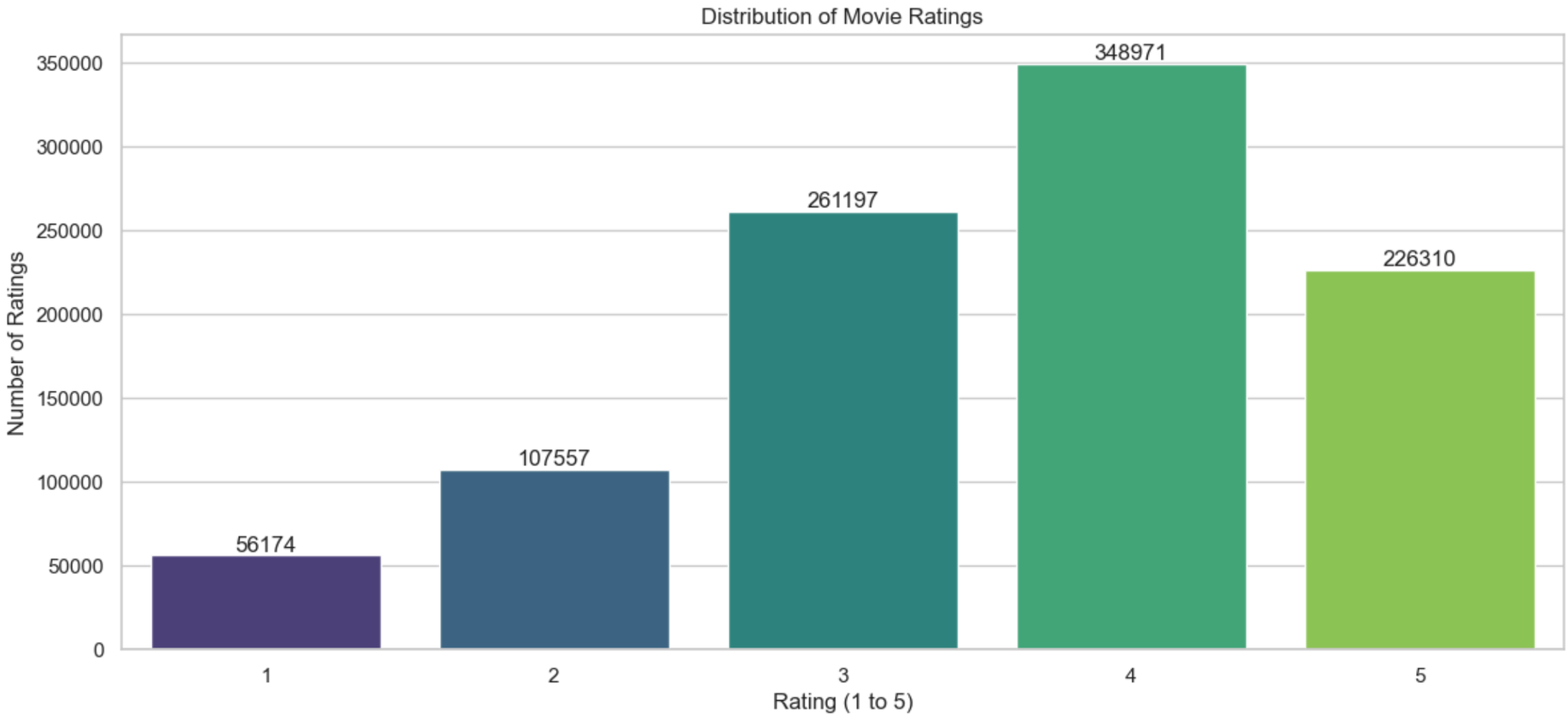
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 17 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   UserID         1000209 non-null  int64
 1   MovieID        1000209 non-null  int64
 2   Rating         1000209 non-null  int32
 3   Timestamp      1000209 non-null  datetime64[ns]
 4   Title          1000209 non-null  object
 5   Genres         1000209 non-null  object
 6   Year           1000209 non-null  Int64
 7   Gender         1000209 non-null  object
 8   Age            1000209 non-null  object
 9   Occupation     1000209 non-null  object
 10  Zip-code       1000209 non-null  object
 11  WatchDate      1000209 non-null  datetime64[ns]
 12  WatchYear      1000209 non-null  int32
 13  WatchMonth     1000209 non-null  int32
 14  WatchHour      1000209 non-null  int32
 15  WatchDay       1000209 non-null  object
 16  ReleaseDecade  1000209 non-null  object
dtypes: Int64(1), datetime64[ns](2), int32(4), int64(2), object(8)
memory usage: 115.4+ MB
```

4 - Exploratory Data Analysis (EDA)

4.1 - Rating Values Distribution

In [15]:

```python
# Plot distribution of rating values

plt.figure(figsize=(14,6))
ax = sns.countplot(x='Rating', data=df, palette='viridis')

# Add value labels on top of the bars
ax.bar_label(ax.containers[0], fmt='%d') # fmt='%d' ensures integer formatting

plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating (1 to 5)')
plt.ylabel('Number of Ratings')
plt.xticks(ticks=[0, 1, 2, 3, 4], labels=[1, 2, 3, 4, 5])
plt.show()
```



🔎 Interpretation:

- **4-Star** ratings dominate with `~349,000` ratings (*__The Highest__*) — suggesting that users are generally positive but not overly generous with 5-Stars.
- With `~260,000` ratings The **3-star** reviews represent a large portion (*__Second Highest__*_) of the dataset, indicating that many users gave a Neutral or Average_* response to the movies they watch.
- Over `~226,000` ratings are **5-Star** (*__Third Highest__*_), showing that many users reward movies they love — though_ less frequently than 4-Stars*.
- With `~56,000` (**1-star**) and `~108,000` (**2-star**) ratings, It Indicates that users don't rate movies harshly often — or *avoid rating movies they dislike*.
- Most users rate in the **3–5 range**, This skews the dataset toward positively-biased feedback, *a common trend in movie recommender datasets* (e.g., MovieLens, Netflix).

📌 Business Insight:

- Users tend to Rate Movies **Positively**, which implies that:
  - They're mostly rating movies **they've already chosen to watch and expect to enjoy**.
  - Ratings are **not normally distributed**, but **right-skewed toward higher values**.
- The **3-Star** Rating being the *Second Highest* reinforces that users aren't just polarized (1 or 5 stars), but also use the middle of the scale — useful for capturing **true preference signals** in the Modeling.

This Implies:

- **Model training**
  - Where Recommenders may **Over-Prioritize Higher-rated** content.
- **User satisfaction**

- Where Users **expect high-quality content**; So **Low-rated items should be Deprioritized**.

4.2 - User Demographics

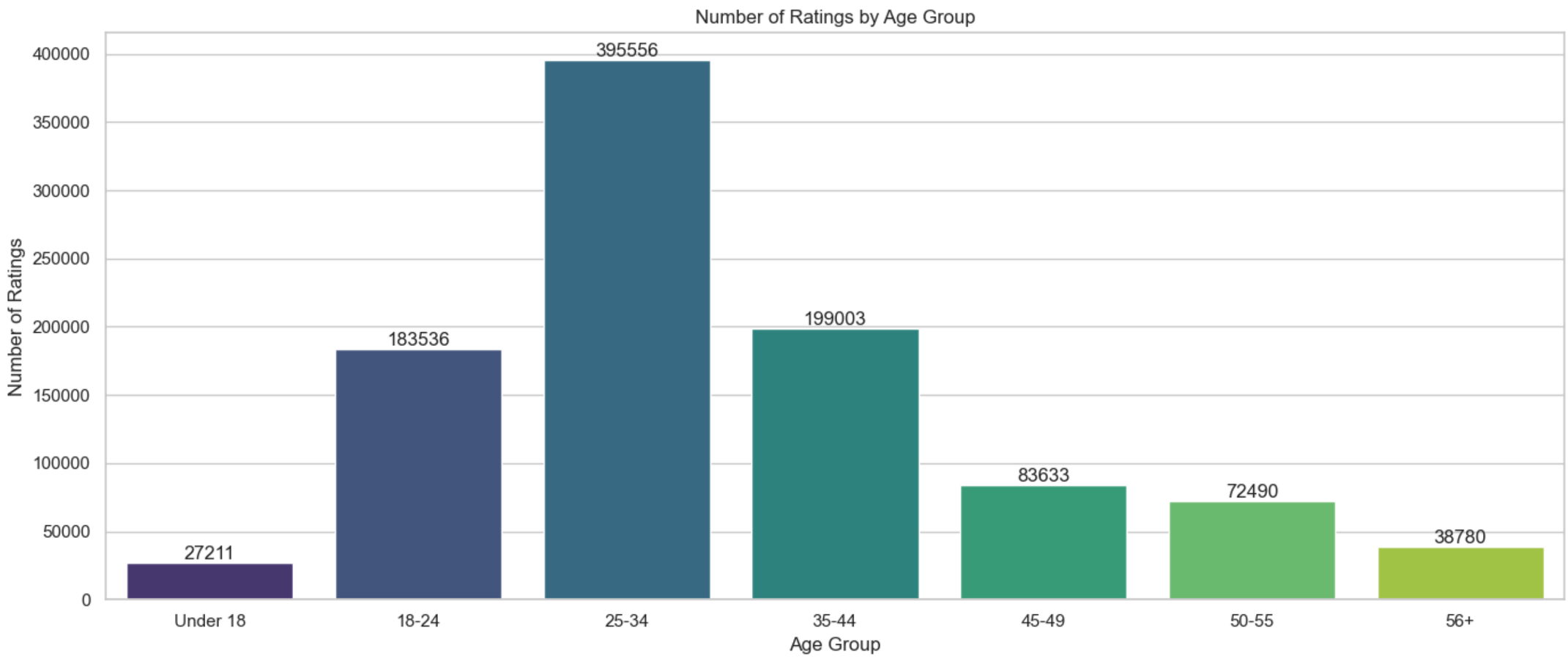
4.2.1 - Age

In [16]:
```
df['Age'].value_counts().sort_index().index
```

Out[16]:
```
Index(['18-24', '25-34', '35-44', '45-49', '50-55', '56+', 'Under 18'], dtype='object', name='Age')
```

In [17]:
```
# Order the age categories for better visualization
age_order = ["Under 18", "18-24", "25-34", "35-44", "45-49", "50-55", "56+"]
```

In [18]:
```
# Plot distribution of ratings by age group
plt.figure(figsize=(14,6))
ax = sns.countplot(data=df, x='Age', order=age_order, palette='viridis')   # 'pastel'

plt.title('Number of Ratings by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Number of Ratings')

# Add value labels using bar_label
ax.bar_label(ax.containers[0], fmt='%d')

# plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

- Users Aged **25-34** contributed `~395,000` Ratings, **The Highest** across All Segments. They form the **Core Audience** for the platform in terms of engagement.
- Users Aged **35-44**, with `~199,000` ratings, are **The Second-Highest** contributors. They form a key **Secondary Audience**, slightly older but still **highly involved in movie consumption**.
- Young adults (**18–24**) contributed `~183,536` Ratings, Indicating Strong Engagement but slightly less than the 35–44 group.
- Engagement drops significantly after age 44, Older users (**45–56+**) provide fewer ratings, possibly **due to lower digital interaction or platform usage**.
- Teenagers (**Under 18**) are the least engaged, with only `~27,211` ratings, this group contributes minimally — possibly **due to parental controls or content restrictions**.

📌 Business Insight:

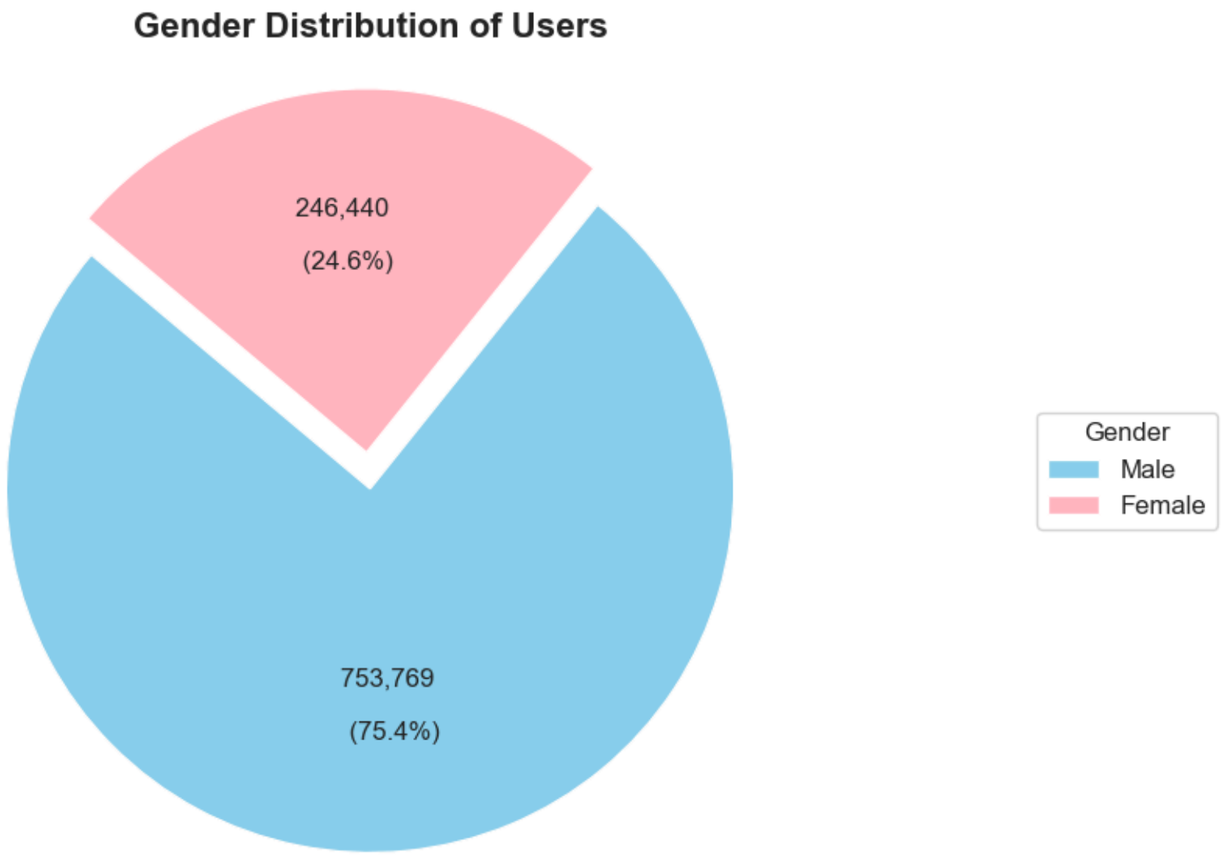
- The Platform's Most Active Audience falls within the `25-44 age range`, accounting for **over 50%** of all ratings.

This Implies:

- Personalization and content strategy should primarily target **Millennials** and **Early Gen X**.
- **Mobile-first**, **Fast-paced UI** experiences will likely resonate with `25-34` and `35-44` Age Segments.
- Senior segments may require `Simplified Interfaces` and `Curated Classics` or `Family-Friendly` Content to drive engagement.

4.2.2 - Gender

In [19]:
```
# Gender value counts and label mapping
gender_counts = df['Gender'].value_counts()
gender_map = {'M': 'Male', 'F': 'Female'}

# Prepare values and labels
sizes = gender_counts.values
labels = [gender_map[gender] for gender in gender_counts.index]

# Explode Shoes slice
explode = [0, 0.1]

# Generate pie chart
plt.figure(figsize=(10,6))
wedges, texts, autotexts = plt.pie(
    sizes,
    autopct=lambda pct: f"{int(round(pct/100.*sum(sizes))):,} \n\n ({pct:.1f}%)",
    startangle=140,
    colors=['skyblue', 'lightpink'],
    textprops={'fontsize': 12},
    explode=explode
)

# Legend with only gender names
plt.legend(wedges, labels, title="Gender", loc="center left", bbox_to_anchor=(1, 0.5), fontsize=12)
plt.title('Gender Distribution of Users', fontsize=16, fontweight='bold')
plt.axis('equal')
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

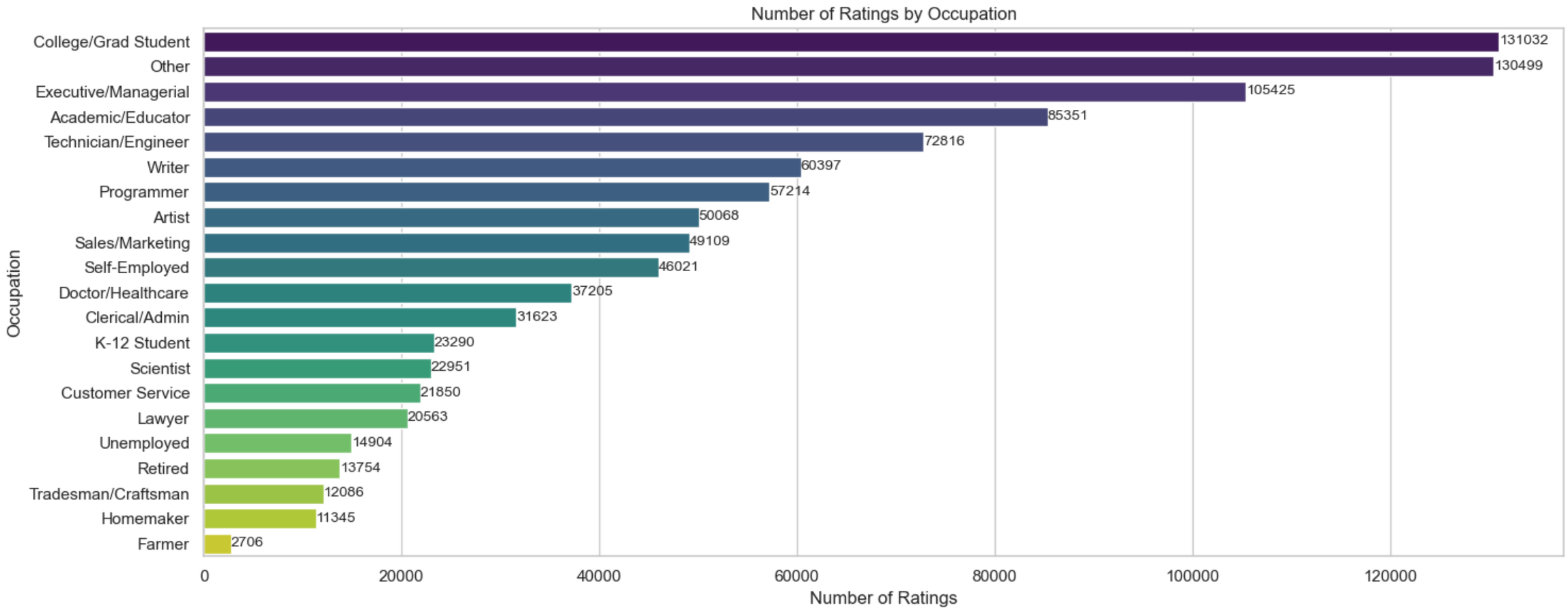
- **Male users are the Dominant Contributors**
  - Males account for `over 75%` of all Ratings - `753,769` ratings (**75.4%**), Indicating a Significant Gender Skew in engagement.
- **Female participation is comparatively low**
  - Female users make up Close to One-Fourth of the rating base - `246,440` ratings (**24.6%**). This suggests either:
    - Lower representation of female users on the platform, or
    - Lower rating activity per female user.
- **Potential Audience Gap**
  - There's an opportunity to improve engagement among female users through tailored content, personalized recommendations, or feature UX optimizations.

📌 Business Insight:

- The current user base is **Heavily Male-Dominated**. To expand audience reach and engagement:
  - Consider **Curating and Surfacing Content** that appeals to Under-represented Gender segments.
  - ***A/B test*** **UI features** or **marketing approaches** that speak better to female movie-goers.

4.2.3 - Occupation

In [20]:
```
# Count plot of Occupation
plt.figure(figsize=(15,6))
ax = sns.countplot(y='Occupation', data=df, order=df['Occupation'].value_counts().index, palette='viridis')

# Add count labels
ax.bar_label(ax.containers[0], fmt='%d', fontsize=10)

plt.title('Number of Ratings by Occupation')
plt.xlabel('Number of Ratings')
plt.ylabel('Occupation')
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

⬆️ Top 5 Most Active Occupations

1. **College/Grad Students** – **131,032 ratings**
2. **Other** – **130,499 ratings** *(broad category, likely includes freelancers, undefined roles)*
3. **Executive/Managerial** – **105,425 ratings**
4. **Academic/Educator** – **85,351 ratings**
5. **Technician/Engineer** – **72,816 ratings**

> These Top Segments are **Highly Educated and Digitally Engaged**, suggesting they're Comfortable Exploring, Rating, and Interacting with Content Platforms.

⬇️ Bottom 5 Least Active Occupations

1. **Farmer** – **2,706 ratings**

2. **Homemaker** – **11,345 ratings**
3. **Tradesman/Craftsman** – **12,086 ratings**
4. **Retired** – **13,754 ratings**
5. **Unemployed** – **14,904 ratings**

> These roles may reflect users with either **Lower Digital Exposure**, **Less Free Time for Leisure Viewing**, or **Less Comfort with Rating Systems**.

📌 Business Insight:

> Users from Academic, Managerial, and Technical Backgrounds are **Power Users** on the Platform.

This Implies:

- The platform should **Prioritize UI/UX and features** that appeal to **Highly Educated, Tech-Savvy Users** — such as Intelligent Filtering, Watchlist History and Advanced Recommendation Explanations.
- Introducing **Content clusters for Academics, Students and Engineers** (e.g., Documentaries, Sci-Fi, Intellectual Thrillers) can deepen engagement in Top-Performing segments.
- There's a **Good potential in Under-represented groups** (e.g., farmers, homemakers, retirees). Simpler & Mobile-friendly UI, or curated movie bundles might help Improve Inclusivity and Platform reach.

4.3 - Genre Trends

4.3.1 - Genre Count Plot

In [21]:
```
df['Genres'].explode().value_counts().sort_index()
```

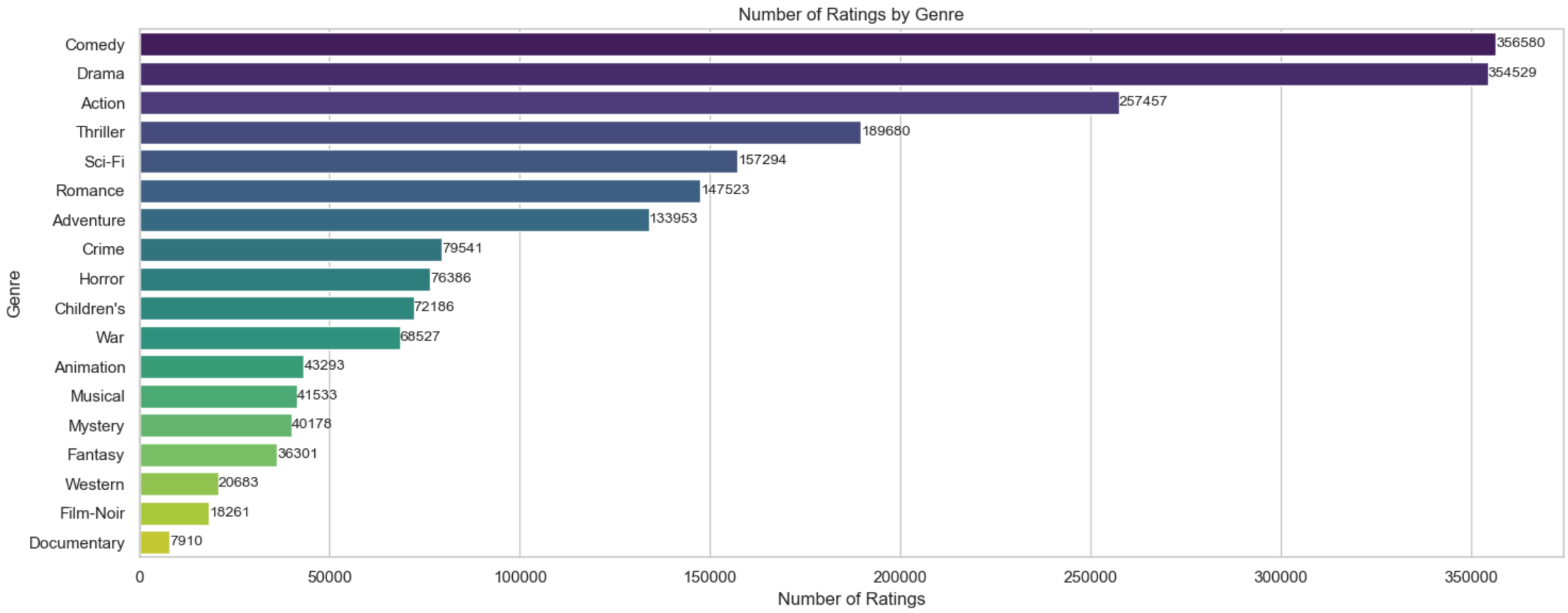
Out[21]:
```
Genres
Action         257457
Adventure      133953
Animation       43293
Children's      72186
Comedy         356580
Crime           79541
Documentary      7910
Drama          354529
Fantasy         36301
Film-Noir       18261
Horror          76386
Musical         41533
Mystery         40178
Romance        147523
Sci-Fi         157294
Thriller       189680
War             68527
Western         20683
Name: count, dtype: int64
```

In [22]:
```
# Since each movie can belong to multiple genres -

# Explode genre list into multiple rows
genre_df = df.explode('Genres')

# Count number of ratings per genre
genre_counts = genre_df['Genres'].value_counts()
```

In [23]:
```
genre_counts.sum()
```

Out[23]:
```
2101815
```

In [24]:
```
# Plot the genre counts
plt.figure(figsize=(15,6))
ax = sns.barplot(x=genre_counts.values, y=genre_counts.index, palette='viridis')

# Add count labels
ax.bar_label(ax.containers[0], fmt='%d', fontsize=10)

plt.title('Number of Ratings by Genre')
plt.xlabel('Number of Ratings')
plt.ylabel('Genre')
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

- **Comedy** and **Drama** Dominate User Preferences, with over `350,000` Ratings each, Closely followed by **Action** (`~257,000`).
- **Thriller** and **Sci-Fi** round out the `Top 5` , Showing Strong Interest in High-Energy and futuristic content.
- The **Least rated genres** include **Documentary**, **Film-Noir**, and **Western**, each receiving under `~21,000` Ratings.
- Genres like **Fantasy**, **Mystery**, and **Animation** fall in the Mid-tier — Modest, but with Potential.

📌 Business Insight:

- User engagement is **Heavily Skewed** toward Mainstream, Entertainment-Heavy Genres like **Comedy**, **Drama**, and **Action** — suggesting these categories should remain central to the Platform's Content Strategy.

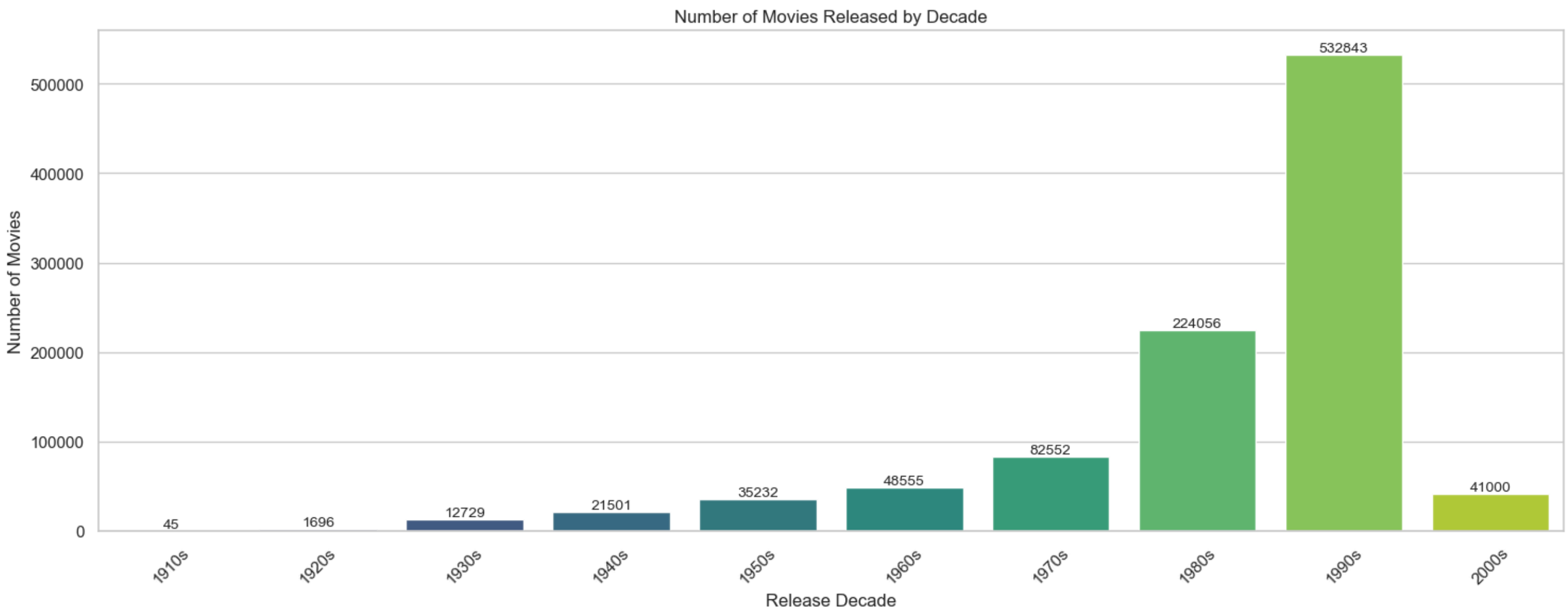
This Implies:

- **Recommendation Algorithms** should Prioritize Titles in High-engagement genres to Optimize Click-Through and Satisfaction Rates.

- There is room to **personalize discovery paths** for under-represented genres like **Documentary** and **Film-Noir** by:
  - Recommending them to Niche user clusters
  - Using Hybrid Models to introduce them alongside more popular genres
- Genre-specific campaigns can **Revive Interest in Low-Engagement Genres**, e.g., `"Forgotten Classics"` for Film-Noir or `"True Stories Week"` for Documentaries.

4.3.2 - Movies Released by Decade

In [25]:
```
# Majority of Movies Relased in Which Decade?
decade_counts = df['ReleaseDecade'].value_counts().sort_index()

# Plot the decade counts
plt.figure(figsize=(15,6))
ax = sns.barplot(x=decade_counts.index, y=decade_counts.values, palette='viridis') # 'mako'

# Add count labels
ax.bar_label(ax.containers[0], fmt='%d', fontsize=10)
plt.title('Number of Movies Released by Decade')
plt.xlabel('Release Decade')
plt.ylabel('Number of Movies')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



🔍 **Interpretation:**

- The **1990s** saw **The Highest** number of movies released, with over `532,000 titles`, dominating all other decades.
- The **1980s** and **1970s** follow in **Second** and **Third** Place respectively, indicating a Major Boom in Film Production during those years.
- Interestingly, the **2000s** show a noticeable drop (`~41,000`), likely **due to dataset Cutoff or Incomplete Data rather than Actual Decline**.
- Decades prior to the 1970s show a **Steady Historical Rise**, but contribute fewer movies in total compared to modern decades.

📌 Business Insight:

- The Majority of movies in the dataset are from the **Late 20th Century**, especially the 1980s and 1990s — which aligns with High user engagement and Cultural Nostalgia for that era.

This Implies:

- Recommendation engines should **Leverage Decade-Based filtering** — especially surfacing 90s and 80s content prominently for users interested in `“Retro”` or `“Nostalgia”` themes.
- If the Platform intends to Grow engagement among Younger Users, Promoting Newer (**Post-2000s**) Titles may require **Better Metadata Coverage or API Enrichment**.
- For Curated Playlists and Featured content, **Decade-based Clustering** can enhance Discovery and Segment-based targeting.

4.3.3 - Genre x Decade Heatmap

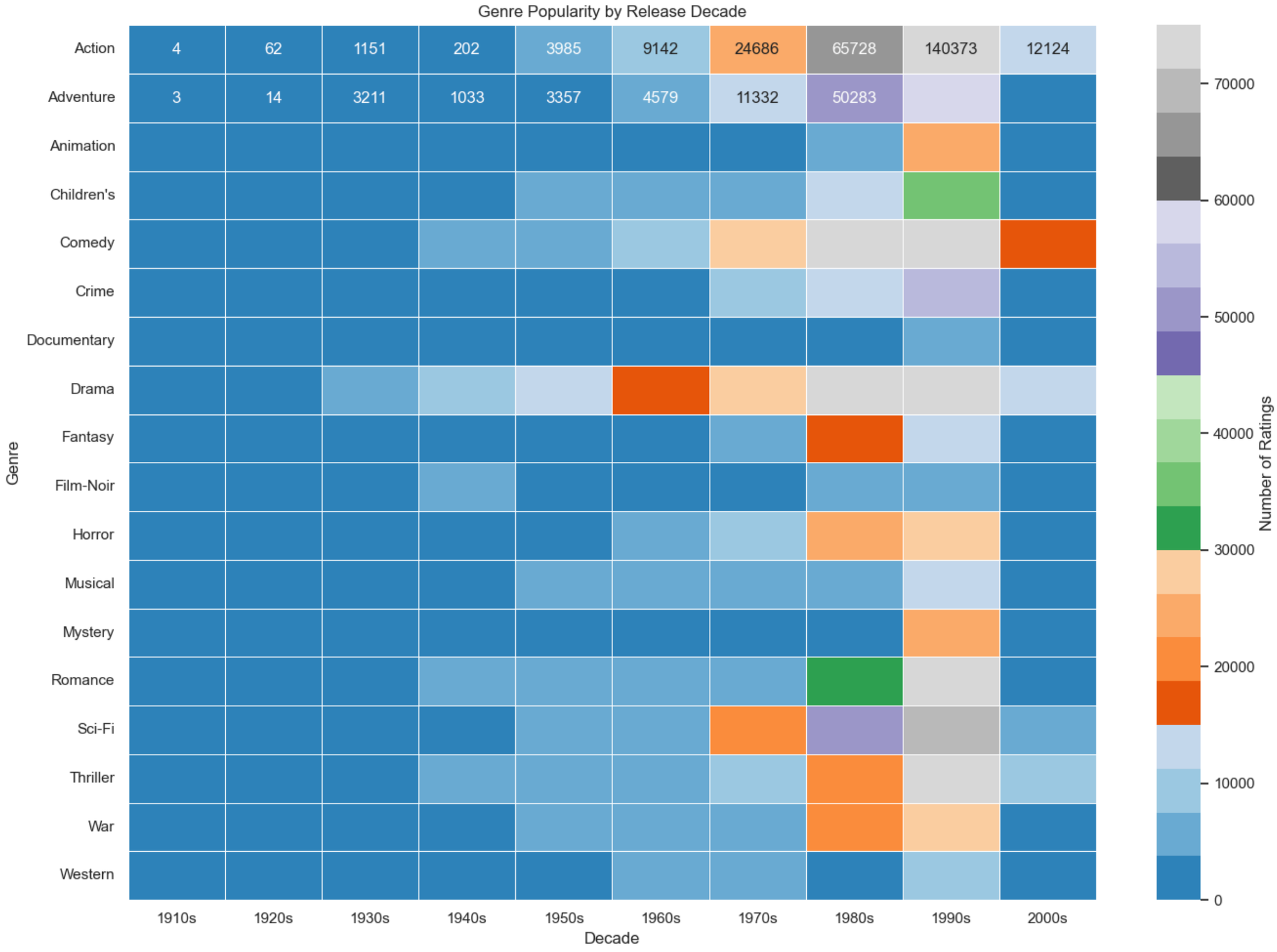
In [26]:
```
# Group by Decade and Genre
genre_decade_df = genre_df.groupby(['ReleaseDecade', 'Genres']).size().reset_index(name='Count')

# Pivot for heatmap
genre_pivot = genre_decade_df.pivot(index='Genres', columns='ReleaseDecade', values='Count').fillna(0)
```

In [27]:
```
genre_pivot
```

Out[27]:

ReleaseDecade Genres	1910s	1920s	1930s	1940s	1950s	1960s	1970s	1980s	1990s	2000s
Action	4.0	62.0	1151.0	202.0	3985.0	9142.0	24686.0	65728.0	140373.0	12124.0
Adventure	3.0	14.0	3211.0	1033.0	3357.0	4579.0	11332.0	50283.0	57831.0	2310.0
Animation	0.0	0.0	880.0	3236.0	3071.0	2142.0	2531.0	6005.0	22573.0	2855.0
Children's	0.0	0.0	2675.0	3388.0	4270.0	4632.0	6816.0	12153.0	35767.0	2485.0
Comedy	38.0	598.0	3355.0	3774.0	5997.0	10616.0	26515.0	94205.0	195967.0	15515.0
Crime	0.0	2.0	330.0	1391.0	2159.0	1078.0	7849.0	11310.0	53502.0	1920.0
Documentary	0.0	0.0	1.0	0.0	3.0	149.0	24.0	1800.0	5511.0	422.0
Drama	7.0	332.0	4603.0	7621.0	12387.0	16313.0	26886.0	74973.0	197745.0	13662.0
Fantasy	0.0	0.0	0.0	0.0	1585.0	542.0	4900.0	16400.0	12739.0	135.0
Film-Noir	0.0	0.0	308.0	4018.0	1901.0	1126.0	1185.0	4675.0	5048.0	0.0
Horror	0.0	238.0	2075.0	779.0	2670.0	4938.0	11109.0	23162.0	29050.0	2365.0
Musical	0.0	32.0	3075.0	2931.0	5904.0	7077.0	4018.0	6959.0	11337.0	200.0
Mystery	0.0	0.0	504.0	3601.0	2706.0	2739.0	2298.0	3691.0	24062.0	577.0
Romance	0.0	45.0	2598.0	4091.0	6102.0	4733.0	4191.0	30138.0	93489.0	2136.0
Sci-Fi	0.0	388.0	232.0	12.0	5377.0	6554.0	19281.0	49745.0	70100.0	5605.0
Thriller	0.0	32.0	982.0	4117.0	6052.0	7390.0	9386.0	22290.0	129345.0	10086.0
War	0.0	279.0	2239.0	2692.0	4980.0	5497.0	5441.0	19414.0	26676.0	1309.0
Western	0.0	0.0	29.0	205.0	1147.0	5078.0	4059.0	1443.0	8722.0	0.0

In [28]:
```
# Plotting the Heatmap for Genre Popularity by Release Decade

plt.figure(figsize=(14, 10))
sns.heatmap(
    genre_pivot,
    #cmap=sns.color_palette("magma", 140),       # or try 'magma', 'crest', 'rocket'
    cmap = sns.color_palette(palette='tab20c'),  # Using a categorical palette for better contrast
    annot=True,
    fmt='g',
    linewidths=0.5,
    vmin=0,
    vmax=75000,           # 🔑 key setting for better contrast
    cbar_kws={"label": "Number of Ratings"}
)

plt.title('Genre Popularity by Release Decade')
plt.xlabel('Decade')
plt.ylabel('Genre')
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

- **Drama** and **Comedy** Dominate across almost every decade, especially the **1990s**, with each crossing 190k+ ratings.
- **Action** and **Thriller** Show Significant Rise `Post-1980s`, Peaking in the `1990s`.
- Genres like **Sci-Fi**, **Romance**, and **Adventure** gained Momentum starting from the **1970s onward**.

- Niche genres like **Documentary**, **Film-Noir**, and **Western** maintain consistently low popularity throughout all decades.

📌 Business Insight:

- The Heatmap clearly shows that **User Interest in Genres is Heavily Decade-Dependent**, with Modern Genres (Action, Thriller, Sci-Fi) Booming in the 80s and 90s.

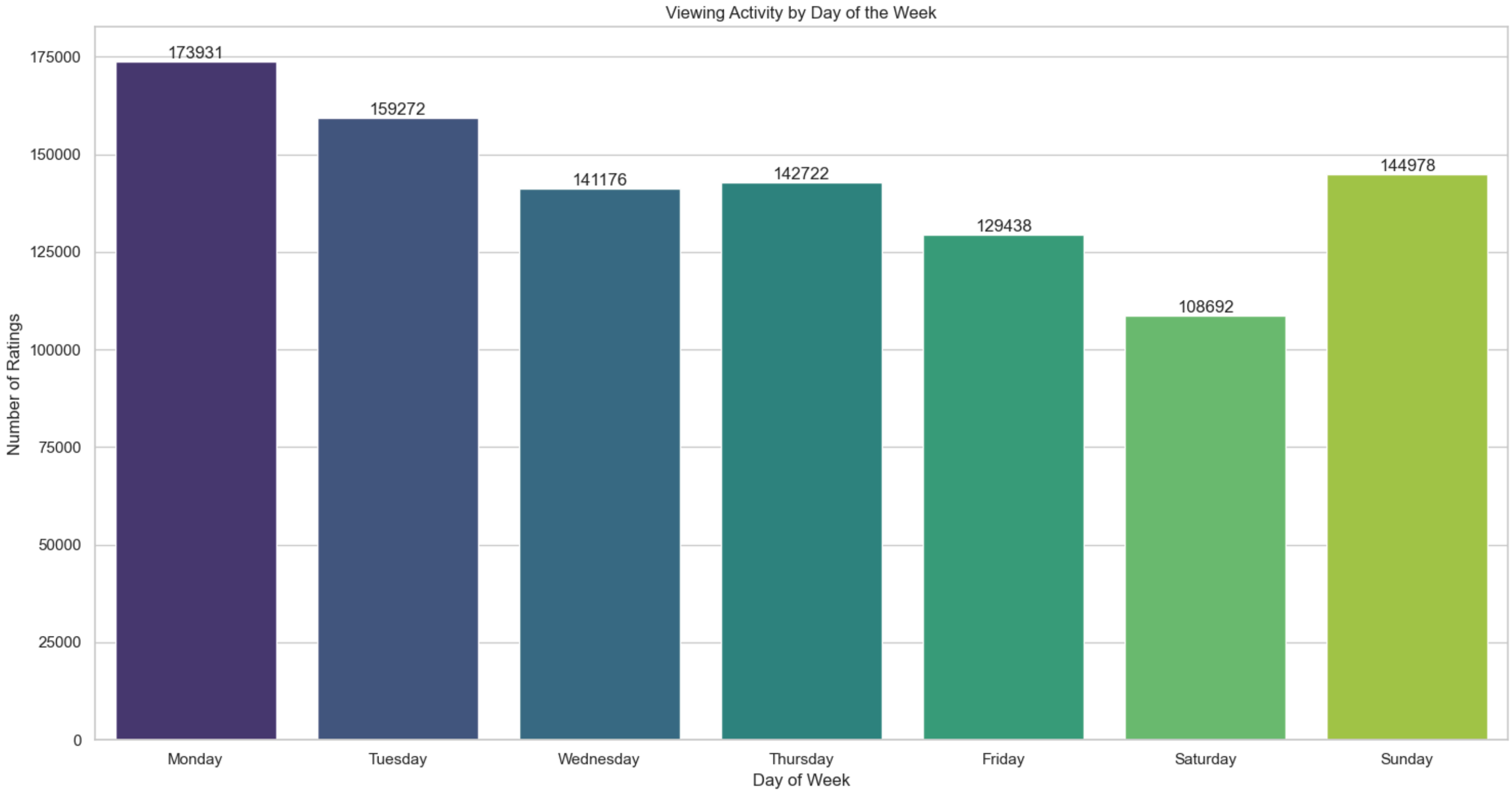
This Implies:

- Recommender systems can **Leverage Decade + Genre Pairings** to Serve Highly Relevant Content (e.g., `90s Action fans`, `80s Sci-Fi lovers`).
- Niche genres can be revived using **Era-based Curation Strategies**, such as `"Vintage Westerns from the 60s"` or `"Classic Film-Noir from the 40s"`.

4.4 - Temporal Patterns

4.4.1 - Activity by Day-of-the-Week

In [29]:
```
# Order for weekdays
weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

plt.figure(figsize=(15,8))
ax = sns.countplot(x='WatchDay', data=df, order=weekday_order, palette='viridis')  # 'Set2'

# Add count labels
ax.bar_label(ax.containers[0], fmt='%d')

plt.title('Viewing Activity by Day of the Week')
plt.xlabel('Day of Week')
plt.ylabel('Number of Ratings')
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

- **Highest Activity** observed on **Monday (173,931)** and **Tuesday (159,272)**.
- Gradual Decline Mid-week, with **Saturday being the lowest (108,692)**.
- Slight Recovery on **Sunday (144,978)**.

📌 Business Insight:

- Contrary to Assumptions that Weekends would see more engagement, the Platform experiences **Peak Viewing on Weekdays**, particularly **Early in the Week**. This suggests users are more active during Workweek Evenings—possibly using Content as a Break or Background Companion.

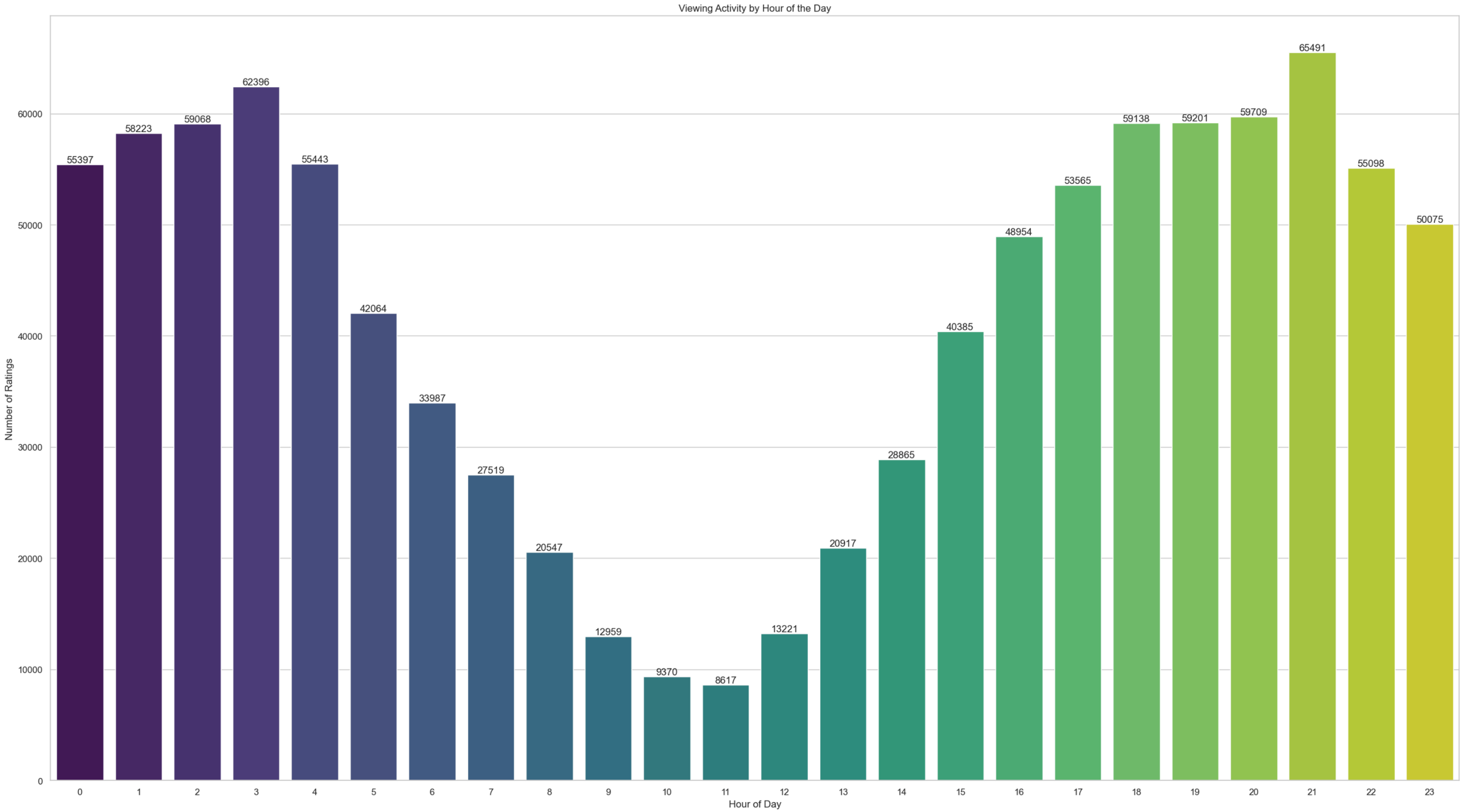
This Implies:

- `Content Drops` , `New Releases` , **and** `Personalized Notifications` **should be Prioritized for** `Mondays` **and** `Tuesdays` to Capture Peak Momentum.

- Consider **“Weekend Booster” campaigns** to uplift Saturday Viewership.
- **Test Weekday segmentation**: Tailor **Weekday content** for `Productivity Relief`, and **Weekend Content** for `Relaxation` or `Binge Sessions`.

4.4.2 - Activity by Hour

In [30]:
```
plt.figure(figsize=(25, 14))
ax = sns.countplot(x='WatchHour', data=df, palette='viridis')  # coolwarm, twilight_shifted

# Add value labels
ax.bar_label(ax.containers[0], fmt='%d')

plt.title('Viewing Activity by Hour of the Day')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Ratings')
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

✅ Peak Viewing Hours (Top 5)

- **21:00 (9 PM) — 65,491 ratings**
- **20:00 (8 PM)** — 59,709
- **19:00 (7 PM)** — 59,201
- **18:00 (6 PM)** — 59,138
- **03:00 (3 AM)** — 62,396 *(late-night watchers)*

> 🔥 **Evening hours (6–10 PM)** show the Highest User activity, clearly establishing them as Prime viewing time.

🚫 Lowest Activity Hours (Bottom 5)

- **11:00 (11 AM)** — **8,617 ratings**
- **10:00 (10 AM)** — 9,370
- **09:00 (9 AM)** — 12,959
- **12:00 (12 PM)** — 13,221
- **08:00 (8 AM)** — 20,547

> ⏳ Activity sharply dips during **mid-morning to early noon**, reflecting workday engagement drop-offs.

📌 Business Insight:

- Viewership patterns are strongly **Concentrated in the Evening**, coinciding with Post-Work Relaxation Hours. A **Notable Late-Night activity spike (~3 AM)** indicates a Dedicated Night Owl segment that Continues Watching or Rating content beyond Midnight.

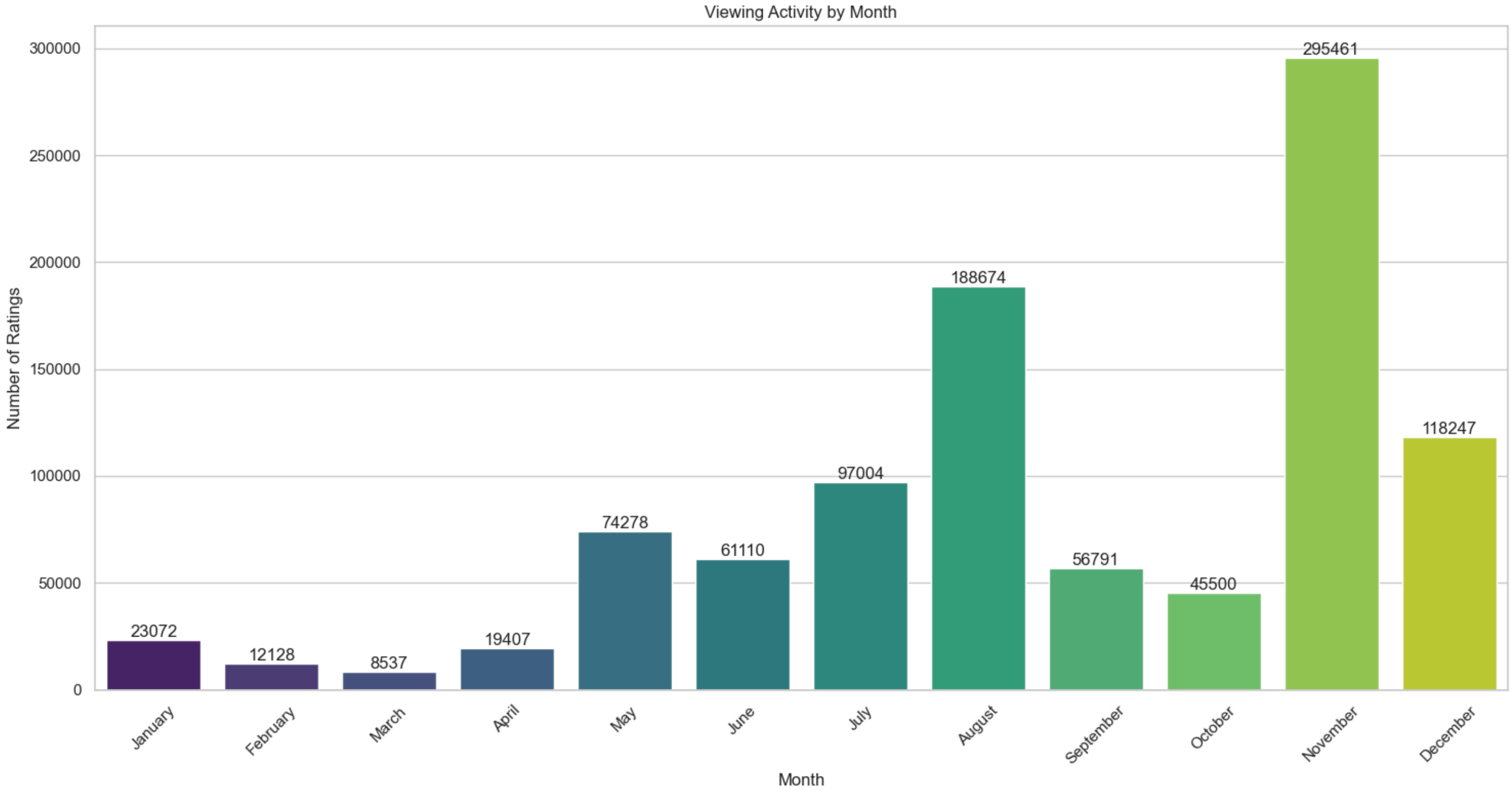
This Implies:

- `Content Promotions`, `Notifications`, or `New Releases` should be Timed around 6–10 PM to **Maximize Visibility**.
- The `Strong 3 AM spike` indicates a **Late-Night Viewer Base** — consider Enabling **Sleep-Time content bundles** (e.g., Short Series, Relaxing Genres).
- Mornings are **Least Effective for Engagement**, and *User Flow/Alerts* can be Minimized or Delayed until post-lunch hours.

4.4.3 - Activity by Month

In [31]:
```
# Month name from datetime
df['WatchMonthName'] = df['WatchDate'].dt.strftime('%B')
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']
```

In [32]:
```
# Plot the count of ratings by month

plt.figure(figsize=(15, 8))
ax = sns.countplot(x='WatchMonthName', data=df, order=month_order, palette='viridis')

# Add count labels
ax.bar_label(ax.containers[0], fmt='%d')

plt.title('Viewing Activity by Month')
plt.xlabel('Month')
plt.ylabel('Number of Ratings')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

- **Highest Viewing Activity** is observed in **November (295,461)**, followed by **August (188,674)** and **December (118,247)**.
- **Lowest Activity** is in **March (8,537)**, **February (12,128)**, and **April (19,407)**.
- A Strong Spike occurs toward the **End of the Year**, with a Noticeable Dip in Early Months.

📌 Business Insight:

- Viewer Engagement aligns with Holiday/Festival Seasons and Academic breaks. **November's Peak** may reflect Pre-Holiday downtime or Special Sontent drops. Similarly, **December** and **August** indicate seasonal or leisure-time spikes.

This Implies:

- **Plan premium content releases, ad campaigns, and promotional pushes during high-engagement months** — especially **November and August**.

- Use the **early months (Feb–Apr)** to experiment with content types, UI changes, or test features — as these periods pose **lower risk to user churn or dissatisfaction**.
- Optimize **Marketing Calendars** and **Budget Allocations** around Seasonal Behavior.

4.5 - Top Rated Movies & Rating Frequencies

In [33]:
```
df
```

Out[33]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Year	Gender	Age	Occupation	Zip-code	WatchDate	WatchYear	WatchMonth	WatchHour	WatchDay	ReleaseDecade	WatchMonthName
0	1	1193	5	2000-12-31 22:12:40	One Flew Over the Cuckoo's Nest	[Drama]	1975	F	Under 18	K-12 Student	48067	2000-12-31 22:12:40	2000	12	22	Sunday	1970s	December
1	1	661	3	2000-12-31 22:35:09	James and the Giant Peach	[Animation, Children's, Musical]	1996	F	Under 18	K-12 Student	48067	2000-12-31 22:35:09	2000	12	22	Sunday	1990s	December
2	1	914	3	2000-12-31 22:32:48	My Fair Lady	[Musical, Romance]	1964	F	Under 18	K-12 Student	48067	2000-12-31 22:32:48	2000	12	22	Sunday	1960s	December
3	1	3408	4	2000-12-31 22:04:35	Erin Brockovich	[Drama]	2000	F	Under 18	K-12 Student	48067	2000-12-31 22:04:35	2000	12	22	Sunday	2000s	December
4	1	2355	5	2001-01-06 23:38:11	Bug's Life, A	[Animation, Children's, Comedy]	1998	F	Under 18	K-12 Student	48067	2001-01-06 23:38:11	2001	1	23	Saturday	1990s	January
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1000204	6040	1091	1	2000-04-26 02:35:41	Weekend at Bernie's	[Comedy]	1989	M	25-34	Doctor/Healthcare	11106	2000-04-26 02:35:41	2000	4	2	Wednesday	1980s	April
1000205	6040	1094	5	2000-04-25 23:21:27	Crying Game, The	[Drama, Romance, War]	1992	M	25-34	Doctor/Healthcare	11106	2000-04-25 23:21:27	2000	4	23	Tuesday	1990s	April
1000206	6040	562	5	2000-04-25 23:19:06	Welcome to the Dollhouse	[Comedy, Drama]	1995	M	25-34	Doctor/Healthcare	11106	2000-04-25 23:19:06	2000	4	23	Tuesday	1990s	April
1000207	6040	1096	4	2000-04-26 02:20:48	Sophie's Choice	[Drama]	1982	M	25-34	Doctor/Healthcare	11106	2000-04-26 02:20:48	2000	4	2	Wednesday	1980s	April
1000208	6040	1097	4	2000-04-26 02:19:29	E.T. the Extra-Terrestrial	[Children's, Drama, Fantasy, Sci-Fi]	1982	M	25-34	Doctor/Healthcare	11106	2000-04-26 02:19:29	2000	4	2	Wednesday	1980s	April

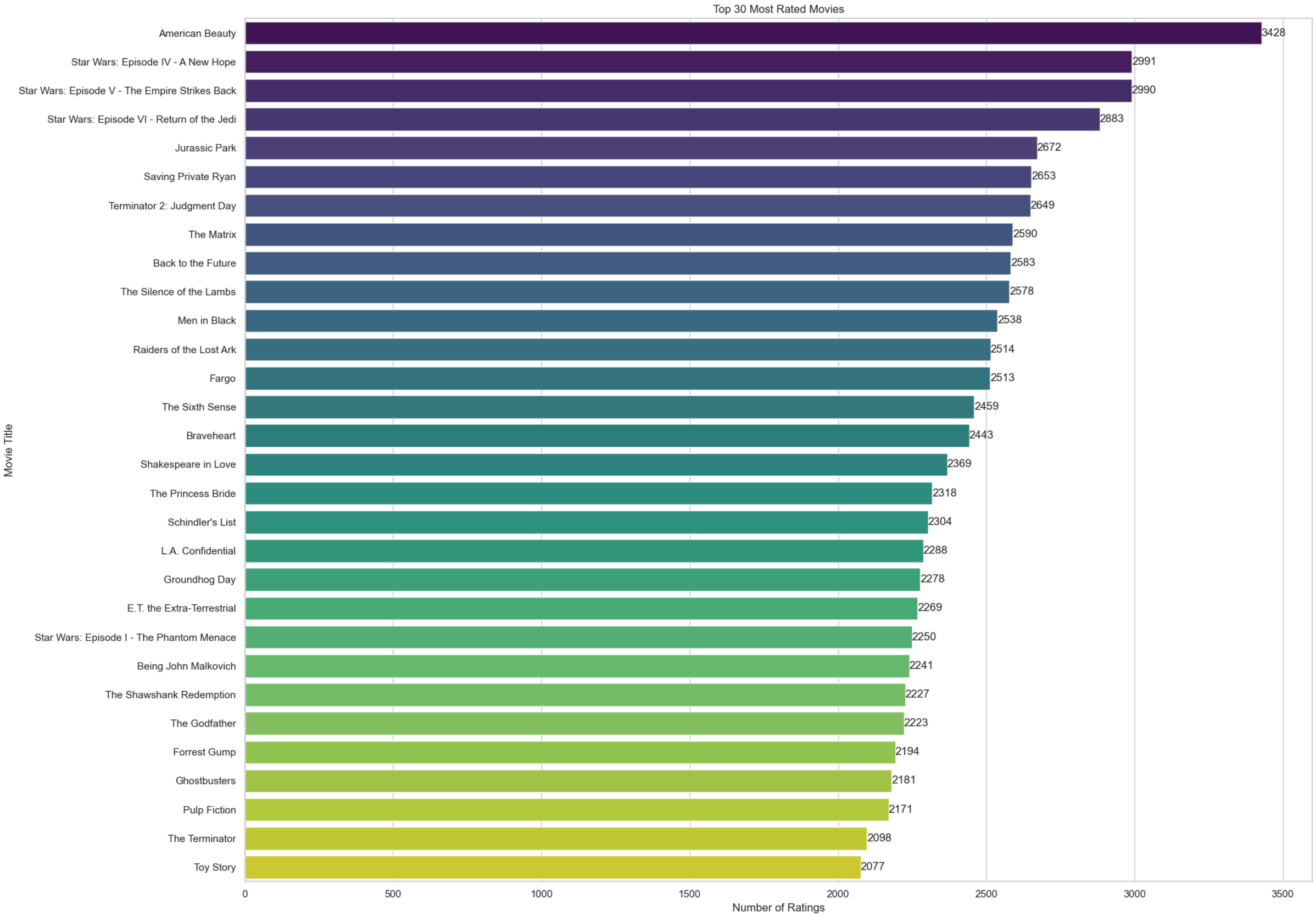
1000209 rows × 18 columns

In [34]:
```
# Fix movie titles with 'The', 'An', 'A' at the end

import re

def fix_title(title):
    # Matches 'Title, The', 'Title, An', 'Title, A'
    match = re.match(r'^(.*),\s(The|An|A)$', title)
    if match:
        return f"{match.group(2)} {match.group(1)}"
    return title

# Apply to the movies DataFrame
df['Title'] = df['Title'].apply(fix_title)
```

4.5.1 - Movies with Most Number of Ratings

In [35]:
```
# Top 30 most rated movies
top_rated_movies = df['Title'].value_counts().head(30)
```

In [36]:
```
# Top 30 Most Rated Movies

plt.figure(figsize=(20,14))
ax = sns.barplot(x=top_rated_movies.values, y=top_rated_movies.index, palette='viridis') # 'viridis', 'mako', 'rocket', 'crest', 'twilight_shifted', 'Blues_r'

# Add value labels
ax.bar_label(ax.containers[0], fmt='%d')

plt.title('Top 30 Most Rated Movies')
plt.xlabel('Number of Ratings')
plt.ylabel('Movie Title')
plt.tight_layout()
plt.show()
```



🔎 Interpretation:

- The **Top 5 most-rated movies** are:

    1. **American Beauty** – 3,428 ratings
    2. **Star Wars: Episode IV - A New Hope** – 2,991
    3. **Star Wars: Episode V - The Empire Strikes Back** – 2,990
    4. **Star Wars: Episode VI - Return of the Jedi** – 2,883
    5. **Jurassic Park** – 2,672
- A **Notable Drop** of `~440 ratings` Exists between the **1st** and **2nd place** (`American Beauty` vs. `Star Wars IV`), while the next Few Titles are Tightly grouped (e.g., just 1 Rating Separates Star Wars IV & V).

📌 Business Insight:

- A Handful of Iconic Titles Dominate Viewer Attention, especially **Cult Classics and Blockbuster Franchises** like `Star Wars` and `Jurassic Park` . These movies not only have mass appeal but also foster **High Engagement** and likely **Repeat Interactions**.

This Implies:

- **Leverage high-engagement titles** for `featured placements` , `Curated Collections` , or `Personalized “because you liked”` Campaigns.
- Explore **Licensing Similar High-Appeal** `Franchises` **or** `Sequels/Spinoffs` to keep Engaged Users on the Platform.
- Use these movies as **Reference Anchors** to Measure Engagement Potential of Newer or Similar Genre Films.

4.5.2 - Movies with Highest Rating

In [37]:
```
# Filter movies with at least 100 ratings - A minimum vote threshold helps avoid misleading high scores from just a few ratings
movie_rating_counts = df.groupby('Title').size()
popular_movies = movie_rating_counts[movie_rating_counts >= 100].index

# Calculate average rating
avg_ratings = df[df['Title'].isin(popular_movies)].groupby('Title')['Rating'].mean().sort_values(ascending=False).head(15)
```

In [38]:
```
# Plotting the top 15 highest rated movies with at least 100 ratings

plt.figure(figsize=(20,16))
ax = sns.barplot(x=avg_ratings.values, y=avg_ratings.index, palette='viridis') # 'viridis', 'mako', 'rocket', 'crest', 'twilight_shifted', 'Blues_r', 'Greens_r'

# Add value labels
ax.bar_label(ax.containers[0], fmt='%.2f')

plt.title('Top 15 Highest Rated Movies (Min 100 Ratings)')
plt.xlabel('Average Rating')
plt.ylabel('Movie Title')
plt.tight_layout()
plt.show()
```

Top 15 Highest Rated Movies (Min 100 Ratings)

Movie Title

Seven Samurai (The Magnificent Seven) (Shichinin no samurai) 4.56
The Shawshank Redemption 4.55
The Godfather 4.52
A Close Shave 4.52
The Usual Suspects 4.52
Schindler's List 4.51
The Wrong Trousers 4.51
Sunset Blvd. (a.k.a. Sunset Boulevard) 4.49
Raiders of the Lost Ark 4.48
Rear Window 4.48
Paths of Glory 4.47
Star Wars: Episode IV - A New Hope 4.45
The Third Man 4.45
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb 4.45

🔎 Interpretation:

- The **Top 5 highest-rated movies** are:
    1. **Seven Samurai (The Magnificent Seven)** – 4.56
    2. **The Shawshank Redemption** – 4.55
    3. **The Godfather** – 4.52
    4. **A Close Shave** – 4.52
    5. **The Usual Suspects** – 4.52
- These titles span **Classic Cinema, Critically Acclaimed Dramas, and Animated Shorts**, reflecting Diverse User appreciation for `storytelling`, `direction`, and `cultural legacy`.
- The Rating values are Tightly packed `between 4.45 and 4.56`, suggesting a **Consistent Ceiling** for Top-Rated Content.

📌 Business Insight:

- Users display a **Strong Preference for** `Timeless Classics`, `Critically Acclaimed Thrillers`, **and** `High-Quality Animation`. These films have Not only Survived the Test of Time but continue to Resonate across Generations.

This Implies:

- Promote **Critically Acclaimed Classics** for New Users to build early trust in Recommendations.
- Curate a **"Top Rated of All Time"** or **"Critics' Choice"** Playlist to Enhance Perceived Value.
- For Retention, Surface **top-rated niche gems** like `A Close Shave` to Movie-Savvy Audiences who Appreciate Quality Over Popularity.

4.6 - EDA Overview – Key Takeaways

Ratings Distribution

- 4-star and 3-star ratings dominate → users skew positive.
- 5-star ratings are common but less than 4s.
- Very few 1- and 2-star ratings → users avoid harsh reviews.
- Right-skewed distribution is ideal for positive bias modeling.

User Age Segments

- Users aged **25–34** contribute the most ratings (~395k).
- 35–44 and 18–24 also highly active → core audience: **18–44**.
- Sharp engagement drop after age 45.
- Teens (<18) are the least active group.

Gender Distribution

- Male users dominate: **75%+ of total ratings**.
- Female engagement is low → opportunity for gender-targeted growth.
- Gender imbalance may skew collaborative results.
- Recommend UX/personalization improvements for women.

Occupation Engagement

- Students, executives, and educators are top raters.
- Farmers, retirees, and homemakers show minimal activity.
- Educated users = power users → design for tech-savvy minds.
- Simpler UI may help broaden participation in under-represented groups.

Genre Preferences

- Comedy, Drama, and Action lead by rating count.
- Sci-Fi and Thriller are also highly favored.
- Documentary, Film-Noir, and Western are least engaged.
- Recommenders should focus on top genres but promote niche content via hybrid paths.

Movies by Decade

- **1990s** dominate by a wide margin, followed by 80s and 70s.
- Older decades contribute less; 2000s show artificial drop (data cutoff).
- Strong potential in retro/nostalgic content curation.

Genre × Decade Heatmap

- Drama and Comedy dominate across all decades.
- Action, Sci-Fi, and Thriller spike post-1980s.
- Interest in genres shifts over decades → personalization opportunity.
- Niche content is decade-stable but underwatched.

Activity by Day

- Mondays and Tuesdays show **highest engagement**.
- Saturday is lowest → contradicts weekend-viewing assumption.
- Weekdays ideal for drops; weekends need strategic uplift.

Activity by Hour

- Peak viewing between **6 PM – 10 PM**.
- Surprise spike at **3 AM** shows late-night bingeing behavior.
- Morning hours (9 AM–12 PM) are the least active.
- Optimize launches for evening and explore late-night bundles.

Monthly Trends

- November, August, and December are peak months.
- February to April show lowest engagement.
- Align content calendar with holiday/vacation cycles.
- Early months ideal for soft launches or A/B testing.

Most Rated Movies

- *American Beauty* and *Star Wars* titles top the list.
- Engagement clustered around franchises and mainstream hits.
- Use these as anchors for similarity models or featured rails.

Top Rated Movies

- *Seven Samurai*, *Shawshank*, *Godfather* dominate average ratings.
- Users reward classics, intelligent storytelling, and niche gems.
- Great opportunity for "Critics' Picks" or high-trust playlists.

5 - Recommender System

5.1 - User–Item Interaction Matrix

To build collaborative filtering recommenders, we first need to create a **User–Item matrix**, where:

- Rows represent **users**
- Columns represent **movies**
- Values represent **ratings**

This matrix will be sparse (most users don't rate most movies), but it's the foundation for computing similarities.

We'll create:

- A raw ratings matrix using `pivot_table`
- Optionally visualize the matrix sparsity

In [39]:
```
# Pivot table: rows = users, columns = movies, values = ratings
user_item_matrix = df.pivot_table(index='UserID', columns='Title', values='Rating')

# Preview the matrix
user_item_matrix.head()
```

Out[39]:

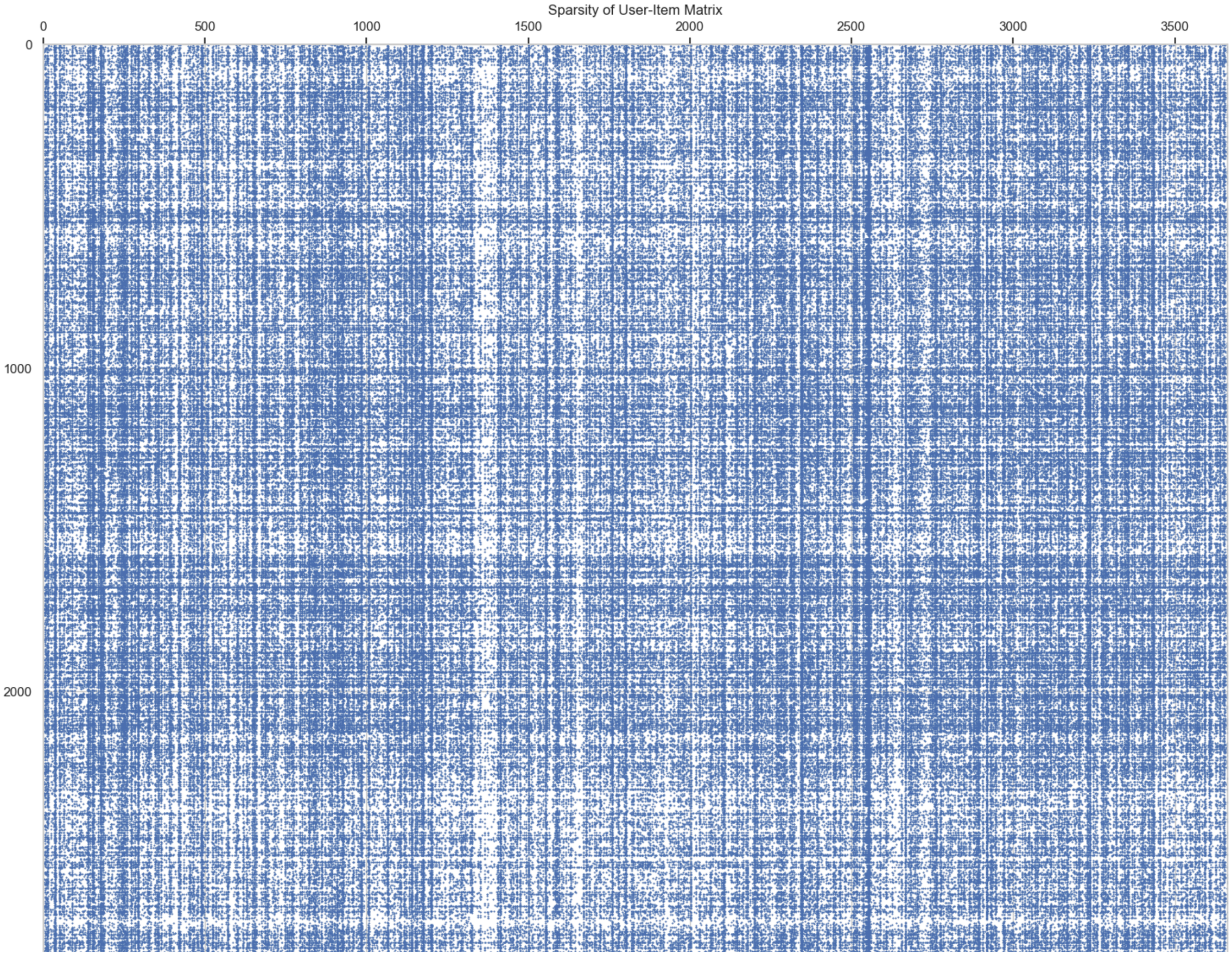
Title	$1,000,000 Duck	'Night Mother	'Til There Was You	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	187	2 Days in the Valley	...	Young Guns	Young Guns II	Young Sherlock Holmes	Young and Innocent	Your Friends and Neighbors	Zachariah	Zero Effect	Zero Kelvin (Kjærlighetens kjøtere)	Zeus and Roxanne	eXistenZ
UserID																					
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	5.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

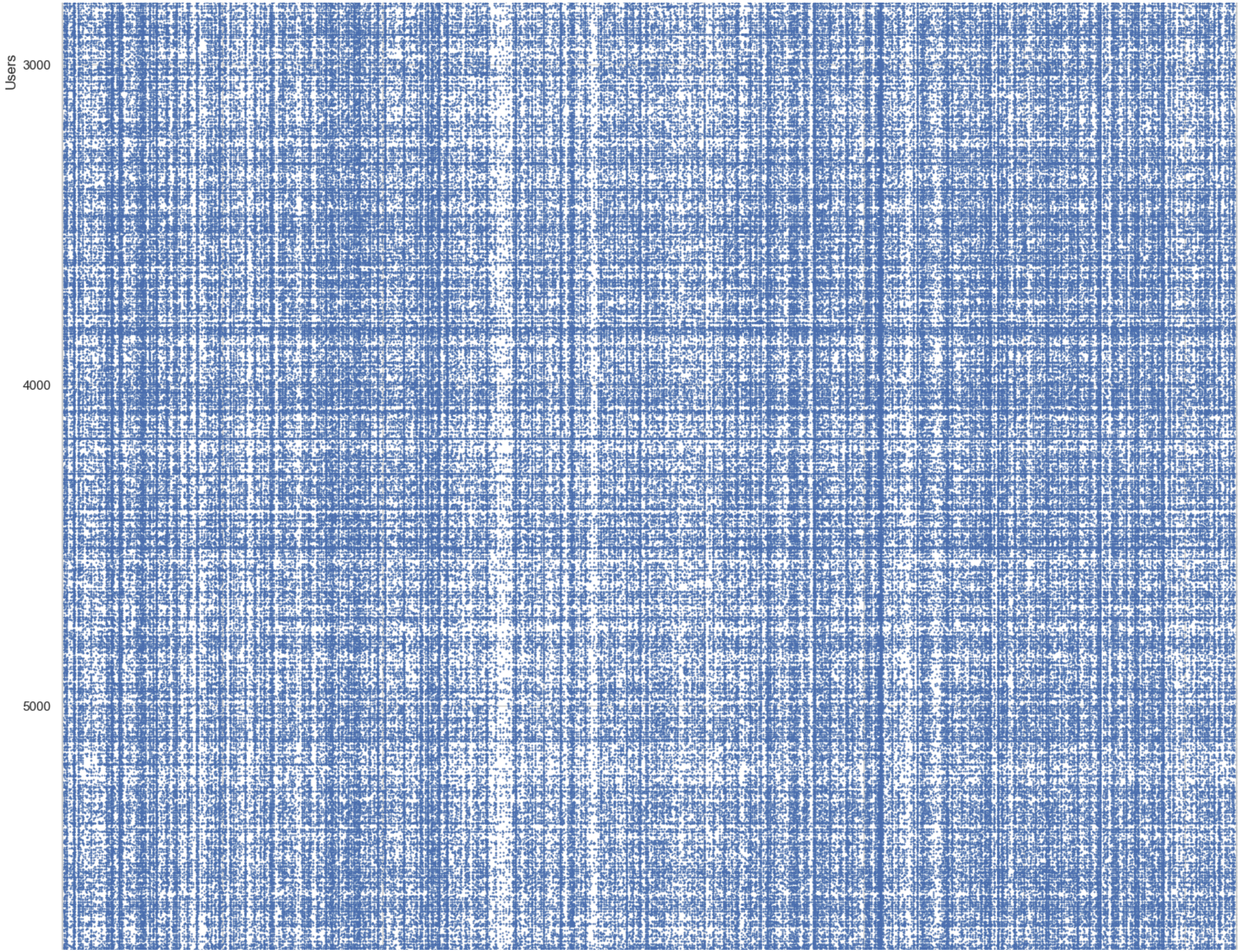
5 rows × 3664 columns

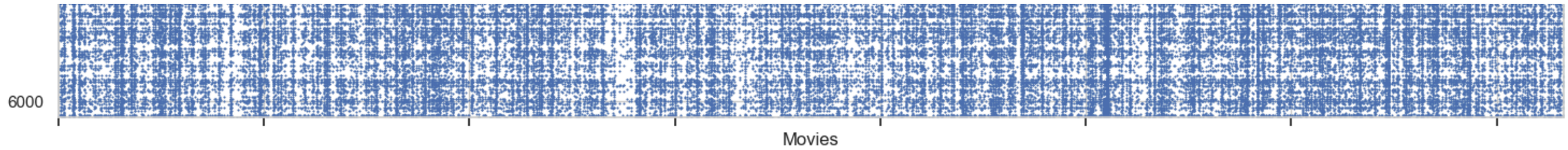
In [40]:
```
user_item_matrix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6040 entries, 1 to 6040
Columns: 3664 entries, $1,000,000 Duck to eXistenZ
dtypes: float64(3664)
memory usage: 168.9 MB
```

In [41]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(20,30))
plt.spy(user_item_matrix, markersize=0.5)
plt.title('Sparsity of User-Item Matrix')
plt.xlabel('Movies')
plt.ylabel('Users')
plt.show()
```


Sparsity of User-Item Matrix
0
500
1000
1500
2000
2500
3000
3500
0
1000
2000


Users
3000
4000
5000



In [42]:
```python
# Total cells in the matrix
total_cells = user_item_matrix.shape[0] * user_item_matrix.shape[1]

# Number of NaN (missing) values
num_missing = user_item_matrix.isna().sum().sum()

# Number of filled (non-NaN) ratings
num_filled = total_cells - num_missing

# Sparsity percentage
sparsity = (num_missing / total_cells) * 100

print(f"Total Cells        : {total_cells:,}")
print(f"Missing Values     : {num_missing:,}")
print(f"Filled Ratings     : {num_filled:,}")
print(f"Sparsity Percentage: {sparsity:.2f}%")
```

```
Total Cells        : 22,130,560
Missing Values     : 21,133,475
Filled Ratings     : 997,085
Sparsity Percentage: 95.49%
```

📌 Business Insight: User–Item Interaction Matrix

- The Dataset exhibits a **Sparsity of 95.49%**, meaning **only `~4.5%` of all User–Movie Combinations have an Actual Rating** — a typical scenario in real-world Recommendation systems.
- Despite the High Sparsity, the **997,000+ Filled Ratings** still offer a **Solid Base for Modeling**, especially across Frequently-Rated Movies and Active Users.
- **Collaborative Filtering techniques** like **User-Based**, **Item-Based**, and **Matrix Factorization** are still viable due to the **Long-Tail of High-Engagement Users and Popular Titles**.
- This level of Sparsity Highlights the importance of:
  - Using **Matrix Compression Techniques** (e.g., CSR)
  - Applying **Hybrid Models** or **Content-Based Fallback** for Cold-Start Items or Users
- The Platform can **Confidently Apply Advanced Recommenders** for its engaged users, while simultaneously investing in **Metadata Enrichment** and **User Onboarding Flows** to reduce Sparsity over time.

5.2 - Item-Based Collaborative Filtering

5.2.1 - Pearson Correlation Based Recommender

In [43]:
```python
# Get titles with at least 50 ratings
popular_movies = df['Title'].value_counts()
popular_movies = popular_movies[popular_movies >= 50].index

# Filter the user-item matrix to only include popular movies
filtered_user_item_matrix = user_item_matrix[popular_movies]

# Recompute correlation matrix with filtered data
item_corr_matrix = filtered_user_item_matrix.corr(method='pearson', min_periods=50)
```

```
item_corr_matrix.head()
```

Out[43]:

Title	American Beauty	Star Wars: Episode IV - A New Hope	Star Wars: Episode V - The Empire Strikes Back	Star Wars: Episode VI - Return of the Jedi	Jurassic Park	Saving Private Ryan	Terminator 2: Judgment Day	The Matrix	Back to the Future	The Silence of the Lambs	...	Blow-Out (La Grande Bouffe)	Vibes	Love Is a Many-Splendored Thing	The Crossing Guard	Audrey Rose	Heidi Fleiss: Hollywood Madam	Hear My Song	A Thousand Acres	House Arrest	The Myth of Fingerprints
American Beauty	1.000000	0.068348	0.089290	0.103226	-0.003588	0.154980	0.055629	0.142432	0.032069	0.155786	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Star Wars: Episode IV - A New Hope	0.068348	1.000000	0.661552	0.574808	0.240746	0.146365	0.191322	0.234341	0.259374	0.121831	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Star Wars: Episode V - The Empire Strikes Back	0.089290	0.661552	1.000000	0.631437	0.201458	0.120312	0.218605	0.208029	0.273120	0.114281	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Star Wars: Episode VI - Return of the Jedi	0.103226	0.574808	0.631437	1.000000	0.307364	0.169816	0.256786	0.217053	0.288792	0.122195	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Jurassic Park	-0.003588	0.240746	0.201458	0.307364	1.000000	0.228763	0.308324	0.163542	0.313988	0.206499	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 2481 columns

In [44]:
```
item_corr_matrix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2481 entries, American Beauty to The Myth of Fingerprints
Columns: 2481 entries, American Beauty to The Myth of Fingerprints
dtypes: float64(2481)
memory usage: 47.0+ MB
```

5.2.1.1 - Fuzzy Matching for Movie Title Search

In [45]:
```
# !pip install fuzzywuzzy[speedup]
# !pip install ipywidgets
```

In [46]:
```
from fuzzywuzzy import process

def fuzzy_search_movie(keyword, movie_list, limit=10, score_threshold=70):
    """
    Fuzzy search for movie titles using fuzzywuzzy.

    Args:
    - keyword (str): Input string (possibly misspelled)
    - movie_list (iterable): List of movie titles to search
    - limit (int): Max number of matches to return
    - score_threshold (int): Minimum score to accept a match (0-100)

    Returns:
    - List of (title, score) tuples
    """
    results = process.extract(keyword, movie_list, limit=limit)
```

```
    filtered_results = [r for r in results if r[1] >= score_threshold]
    return filtered_results

# Example usage
fuzzy_search_movie("Star Wars", item_corr_matrix.columns)
```

Out[46]:
```
[('Star Wars: Episode IV - A New Hope', 90),
 ('Star Wars: Episode V - The Empire Strikes Back', 90),
 ('Star Wars: Episode VI - Return of the Jedi', 90),
 ('Star Wars: Episode I - The Phantom Menace', 90),
 ('Star Trek: The Wrath of Khan', 86),
 ('Star Trek: First Contact', 86),
 ('Star Trek IV: The Voyage Home', 86),
 ('Star Trek VI: The Undiscovered Country', 86),
 ('Star Trek: Generations', 86),
 ('Star Trek III: The Search for Spock', 86)]
```

5.2.1.2 - Pearson-Based Movie Recommender Function

In [47]:
```
def recommend_similar_movies_pearson(movie_title, corr_matrix, n=5):
    """
    Recommend top N similar movies using Pearson correlation.

    Args:
    - movie_title (str): Title of the input movie
    - corr_matrix (pd.DataFrame): Precomputed Pearson correlation matrix
    - n (int): Number of similar movies to return

    Returns:
    - pd.Series: Top N similar movie titles with correlation scores
    """
    if movie_title not in corr_matrix.columns:
        print(f"'{movie_title}' not found in correlation matrix.")
        return pd.Series(dtype='float64')

    similar_movies = corr_matrix[movie_title].dropna()
    similar_movies = similar_movies.drop(labels=[movie_title], errors='ignore')
    top_n = similar_movies.sort_values(ascending=False).head(n)

    return top_n
```

5.2.1.3 - Smart Pearson-Based Recommender with Fuzzy Matching

In [48]:
```
def smart_recommend_pearson(input_title, corr_matrix, n=5):
    """
    Fuzzy-matches the input title to the best match in correlation matrix,
    then returns top N Pearson-similar movies.
    """
    all_titles = corr_matrix.columns
    match = process.extractOne(input_title, all_titles, score_cutoff=70)

    if match:
        corrected_title = match[0]
        print(f"\n✅ Using matched title: '{corrected_title}' (Score: {match[1]})")
        recommendations = recommend_similar_movies_pearson(corrected_title, corr_matrix, n=n)

        if not recommendations.empty:
            print(f"\n📌 Top {n} recommendations similar to '{corrected_title}':\n")
            print(recommendations)
        else:
            print("\n⚠️ No recommendations found (not enough overlap or data).")
    else:
        print(f"\n❌ No close match found for: '{input_title}'. Try refining the name.")
```

```
# Example usage
smart_recommend_pearson("Star Wars", item_corr_matrix)
```

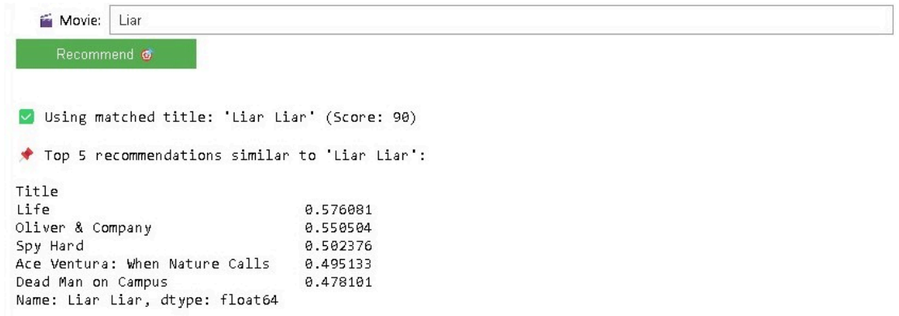
```
✅ Using matched title: 'Star Wars: Episode IV - A New Hope' (Score: 90)

📌 Top 5 recommendations similar to 'Star Wars: Episode IV - A New Hope':

Title
Star Wars: Episode V - The Empire Strikes Back    0.661552
Star Wars: Episode VI - Return of the Jedi        0.574808
Sanjuro                                           0.430860
Raiders of the Lost Ark                           0.421425
42 Up                                             0.371750
Name: Star Wars: Episode IV - A New Hope, dtype: float64
```

5.2.1.4 - Interactive Pearson Recommender Widget (Fuzzy-Aware)

In [ ]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output

# Input widget
movie_input = widgets.Text(
    value='',
    placeholder='Type a movie name (e.g. Star Wars)',
    description='🎬 Movie:',
    disabled=False,
    layout=widgets.Layout(width='60%')
)

# Button widget
recommend_button = widgets.Button(
    description='Recommend 🎯',
    button_style='success',
    layout=widgets.Layout(width='170px')
)

# Output area
output = widgets.Output()

# Function triggered by button
def on_button_click(b):
    with output:
        clear_output()
        user_input = movie_input.value
        if not user_input.strip():
            print("⚠️ Please enter a movie title.")
            return
        smart_recommend_pearson(user_input, item_corr_matrix)

# Bind button to function
recommend_button.on_click(on_button_click)

# Display widgets
display(widgets.VBox([movie_input, recommend_button, output]))
```

```
VBox(children=(Text(value='', description='🎬 Movie:', layout=Layout(width='60%'), placeholder='Type a movie na…
```

🎯 Demo: Interactive Recommender (Screenshot)



Pearson Correlation -

- Captures Movies that were Rated Similarly by the **Same Users**. Focuses on **Co-rated Patterns**, even if Users didn't Rate many movies overall.

📝Top Recommendations for "Liar Liar":

- Life (0.576)
- Oliver & Company (0.550)
- Spy Hard (0.502)
- Ace Ventura: When Nature Calls (0.495)
- Dead Man on Campus (0.478)

📌 Observation:

- Mostly `Comedies` with Similar Audience Preferences and Release period. *__Works well with Users who Rate Consistently__*.

5.2.2 - Item-Based Collaborative Filtering using Cosine Similarity (KNN)

5.2.2.1 - Train KNN Model for Item-Based Collaborative Filtering (Cosine Similarity)

In [50]:
```
from sklearn.neighbors import NearestNeighbors

# Use filtered matrix (only popular movies)
movie_user_matrix = filtered_user_item_matrix.fillna(0).T  # Transpose to get movies as rows

# Fit KNN model
knn_model = NearestNeighbors(metric='cosine', algorithm='brute')
knn_model.fit(movie_user_matrix)

print(f"✅ KNN Model trained on shape: {movie_user_matrix.shape}")
```

```
✅ KNN Model trained on shape: (2481, 6040)
```

5.2.2.2 - Recommend Similar Movies using KNN (Cosine Similarity)

In [51]:
```
def recommend_similar_movies_knn(movie_title, matrix, knn_model, n=5):
    """
    Recommend top N similar movies using KNN (Cosine Similarity).

    Args:
    - movie_title (str): Target movie name
    - matrix (pd.DataFrame): Movie-user matrix (rows = movies)
    - knn_model: Trained sklearn NearestNeighbors model
    - n (int): Number of similar movies to return
```

```
    Returns:
    - pd.DataFrame: Top N similar movies with similarity scores
    """
    if movie_title not in matrix.index:
        print(f"❌ '{movie_title}' not found in matrix.")
        return pd.DataFrame()

    idx = matrix.index.get_loc(movie_title)
    distances, indices = knn_model.kneighbors(matrix.iloc[idx, :].values.reshape(1, -1), n_neighbors=n+1)

    similar_movies = matrix.index[indices.flatten()[1:]]
    similarity_scores = 1 - distances.flatten()[1:]  # convert cosine distance to similarity

    return pd.DataFrame({
        'Movie': similar_movies,
        'Cosine Similarity': similarity_scores
    })
```

In [52]:
```
recommend_similar_movies_knn("The Matrix", movie_user_matrix, knn_model)
```

Out[52]:

	Movie	Cosine Similarity
0	Terminator 2: Judgment Day	0.745532
1	Total Recall	0.703265
2	Star Wars: Episode V - The Empire Strikes Back	0.689459
3	Men in Black	0.684763
4	Star Wars: Episode IV - A New Hope	0.680378

5.2.2.3 - Smart KNN-Based Recommender with Fuzzy Title Matching

In [53]:
```
def smart_recommend_knn(input_title, matrix, knn_model, n=5):
    """
    Fuzzy match movie title and return top N KNN-based recommendations.
    """
    all_titles = matrix.index
    match = process.extractOne(input_title, all_titles, score_cutoff=70)

    if match:
        corrected_title = match[0]
        print(f"\n✅ Using matched title: '{corrected_title}' (Score: {match[1]})")
        results = recommend_similar_movies_knn(corrected_title, matrix, knn_model, n=n)

        if not results.empty:
            print(f"\n📌 Top {n} recommendations similar to '{corrected_title}':\n")
            print(results)
        else:
            print("⚠️ No recommendations found (likely due to sparse data).")
    else:
        print(f"❌ No close match found for: '{input_title}'")
```

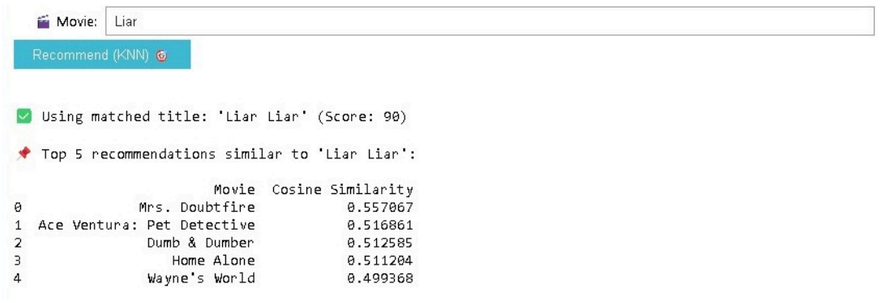
5.2.2.4 - Interactive KNN Recommender Widget (Cosine + Fuzzy Search)

In [ ]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output

# Input widget
movie_input_knn = widgets.Text(
    value='',
    placeholder='Type a movie name (e.g. Star Wars)',
```

```
    description='🎬 Movie:',
    disabled=False,
    layout=widgets.Layout(width='60%')
)

# Button widget
recommend_button_knn = widgets.Button(
    description='Recommend (KNN) 🎯',
    button_style='info',
    layout=widgets.Layout(width='170px')
)

# Output area
output_knn = widgets.Output()

# Function to run when button is clicked
def on_knn_click(b):
    with output_knn:
        clear_output()
        user_input = movie_input_knn.value
        if not user_input.strip():
            print("⚠️ Please enter a movie title.")
            return
        smart_recommend_knn(user_input, movie_user_matrix, knn_model)

# Bind button to handler
recommend_button_knn.on_click(on_knn_click)

# Display the widget
display(widgets.VBox([movie_input_knn, recommend_button_knn, output_knn]))
```

```
VBox(children=(Text(value='', description='🎬 Movie:', layout=Layout(width='60%'), placeholder='Type a movie na…
```

🎯 Demo: Interactive Recommender (Screenshot)



Cosine Similarity (KNN) -

- Measures Vector Similarity in User-Rating Space — Regardless of the Actual Rating Values. Captures Movies Rated by a Similar Set of Users.

📝 Top Recommendations for “Liar Liar”:

- Mrs. Doubtfire (0.557)
- Ace Ventura: Pet Detective (0.516)
- Dumb & Dumber (0.512)
- Home Alone (0.511)
- Wayne’s World (0.499)

📌 **Observation:**

- Strongly Identifies Mainstream `Comedy movies`, *__Emphasizing Popularity Overlap and Shared Audiences__*.

5.3 - Matrix Factorization using SVD (Surprise)

In [55]:
```
from surprise import SVD, Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import accuracy

# Define reader format
reader = Reader(rating_scale=(1, 5))

# Load Surprise dataset
surprise_data = Dataset.load_from_df(df[['UserID', 'Title', 'Rating']], reader)   #? It automatically encodes categorical IDs internally

# Split into train/test
trainset, testset = train_test_split(surprise_data, test_size=0.2, random_state=42)
```

5.3.1 - Train & Evaluate Matrix Factorization (SVD) with RMSE and MAPE

In [56]:
```
from surprise.model_selection import cross_validate
import numpy as np

# Initialize SVD model with d=4 latent factors
svd_model = SVD(n_factors=4, random_state=42)

# Train on training set
svd_model.fit(trainset)

# Predict on test set
predictions = svd_model.test(testset)

# Evaluate using RMSE
rmse = accuracy.rmse(predictions)

# Evaluate using MAPE
def compute_mape(preds):
    ape = [abs((true_r - est_r) / true_r) for (_, _, true_r, est_r, _) in preds if true_r != 0]
    return np.mean(ape) * 100

mape = compute_mape(predictions)

print(f"\n✅ RMSE: {rmse:.4f}")
print(f"✅ MAPE: {mape:.2f}%")
```

```
RMSE: 0.8834

✅ RMSE: 0.8834
✅ MAPE: 26.95%
```

5.3.2 - Extract Item Embeddings from SVD (Matrix Factorization)

In [57]:
```
# Extract Item Embeddings

# Build mapping: Movie name → internal Surprise ID
item_inner_ids = trainset._raw2inner_id_items

# Extract item embeddings from the trained SVD model
item_embeddings = np.array([svd_model.qi[trainset.to_inner_iid(item)] for item in item_inner_ids])

# Create a DataFrame with embeddings and movie names
```

```
item_embed_df = pd.DataFrame(item_embeddings, index=item_inner_ids.keys())
item_embed_df.columns = [f"dim_{i+1}" for i in range(item_embed_df.shape[1])]

item_embed_df.head()
```

Out[57]:

	dim_1	dim_2	dim_3	dim_4
Austin Powers: The Spy Who Shagged Me	-0.011805	0.091893	1.365084	0.311122
Rear Window	-0.156412	-0.159317	-0.320661	0.394828
Gone in 60 Seconds	0.276894	0.244033	0.647698	-0.512856
Titanic	0.346819	-1.043446	-0.037666	-0.750179
Predator 2	0.429259	0.153025	-0.004918	-0.374220

5.3.3 - Recommend Movies using SVD Embeddings + Cosine Similarity

In [58]:
```
from sklearn.metrics.pairwise import cosine_similarity

def get_similar_movies_from_embeddings(movie_name, embed_df, top_n=5):
    """
    Get top N similar movies using cosine similarity on embedding vectors.
    """
    if movie_name not in embed_df.index:
        print(f"❌ '{movie_name}' not found in embeddings.")
        return pd.DataFrame()

    movie_vec = embed_df.loc[movie_name].values.reshape(1, -1)
    all_vecs = embed_df.values
    similarities = cosine_similarity(movie_vec, all_vecs).flatten()

    # Build result DataFrame
    sim_df = pd.DataFrame({
        'Movie': embed_df.index,
        'Cosine Similarity': similarities
    }).sort_values(by='Cosine Similarity', ascending=False)

    # Drop self and return top-N
    sim_df = sim_df[sim_df['Movie'] != movie_name]
    return sim_df.head(top_n)
```

In [59]:
```
get_similar_movies_from_embeddings("Titanic", item_embed_df)
```

Out[59]:

	Movie	Cosine Similarity
467	Awakenings	0.997873
2924	This World, Then the Fireworks	0.994177
1357	And the Band Played On	0.989196
3165	Female Perversions	0.983620
2513	White Sands	0.982609

5.3.4 - Smart SVD-Based Recommender with Fuzzy Title Matching

In [60]:
```
from fuzzywuzzy import process

def smart_recommend_svd_embeddings(input_title, embed_df, top_n=5):
    """
    Fuzzy match a movie name and return top-N similar movies using SVD embeddings.
```

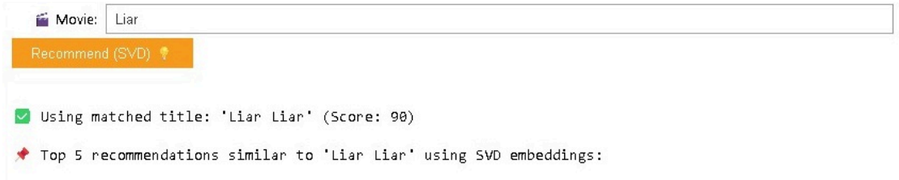
```
    """
    all_titles = embed_df.index
    match = process.extractOne(input_title, all_titles, score_cutoff=70)

    if match:
        corrected_title = match[0]
        print(f"\n✅ Using matched title: '{corrected_title}' (Score: {match[1]})")
        results = get_similar_movies_from_embeddings(corrected_title, embed_df, top_n=top_n)

        if not results.empty:
            print(f"\n📌 Top {top_n} recommendations similar to '{corrected_title}' using SVD embeddings:\n")
            return results
        else:
            print("⚠️ No recommendations found — possibly due to sparse latent representation.")
            return pd.DataFrame()
    else:
        print(f"❌ No close match found for: '{input_title}'")
        return pd.DataFrame()
```

5.3.5 - Interactive SVD Embedding Recommender Widget (Fuzzy + Cosine)

In [61]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output

# Input field
movie_input_embed = widgets.Text(
    value='',
    placeholder='Type a movie name (e.g. Star Wars)',
    description='🎬 Movie:',
    layout=widgets.Layout(width='60%')
)

# Wider recommend button
recommend_button_embed = widgets.Button(
    description='Recommend (SVD) 💡',
    button_style='warning',
    layout=widgets.Layout(width='170px')  # 👈 Wider layout
)

# Output area
output_embed = widgets.Output()

# Click function using fuzzy-matched embedding recommender
def on_embed_click(b):
    with output_embed:
        clear_output()
        movie = movie_input_embed.value.strip()
        if not movie:
            print("⚠️ Please enter a movie title.")
            return
        results = smart_recommend_svd_embeddings(movie, item_embed_df)
        if not results.empty:
            display(results)

# Bind event
recommend_button_embed.on_click(on_embed_click)

# Display UI
display(widgets.VBox([movie_input_embed, recommend_button_embed, output_embed]))
```

```
VBox(children=(Text(value='', description='🎬 Movie:', layout=Layout(width='60%'), placeholder='Type a movie na…
```

🎯 Demo: Interactive Recommender (Screenshot)



	Movie	Cosine Similarity
1802	If Lucy Fell	0.994338
574	Any Given Sunday	0.991745
3213	Mighty Peking Man (Hsing hsing wang)	0.987575
1679	The Hot Spot	0.987551
2808	House Party 3	0.986474

Matrix Factorization (SVD) -

- Learns latent factors from user-movie interactions. Captures deep thematic connections (e.g., humor style, cast influence).

📝 Top Recommendations for "Liar Liar":

- Lucy Fell (0.99)
- Any Given Sunday (0.99)
- Mighty Peking Man (Hsing hsing wang) (0.98)
- The Hot Spot (0.98)
- House Party 3 (0.98)

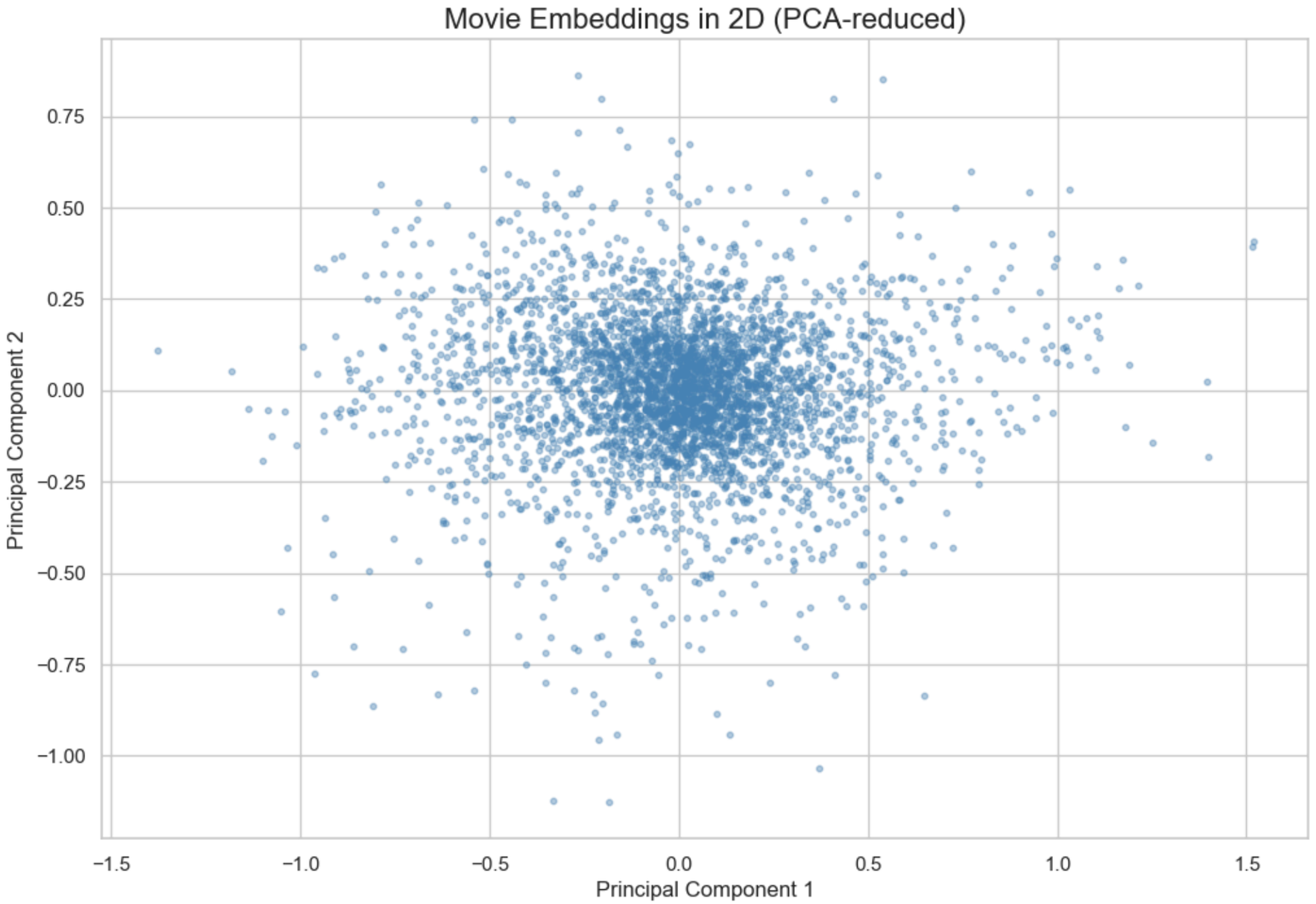
📌 **Observation**:

- The SVD model returns **Less Mainstream, Thematically Distant** Movies — suggesting it's Picking up on `Hidden User Preferences`, `Humor Tones`, **or** `Stylistic` **Overlaps**. While accurate Mathematically, these recommendations may require **Additional Filtering or Blending** for practical use.

5.4 - Bonus: Visualizing Movie Embeddings (SVD)

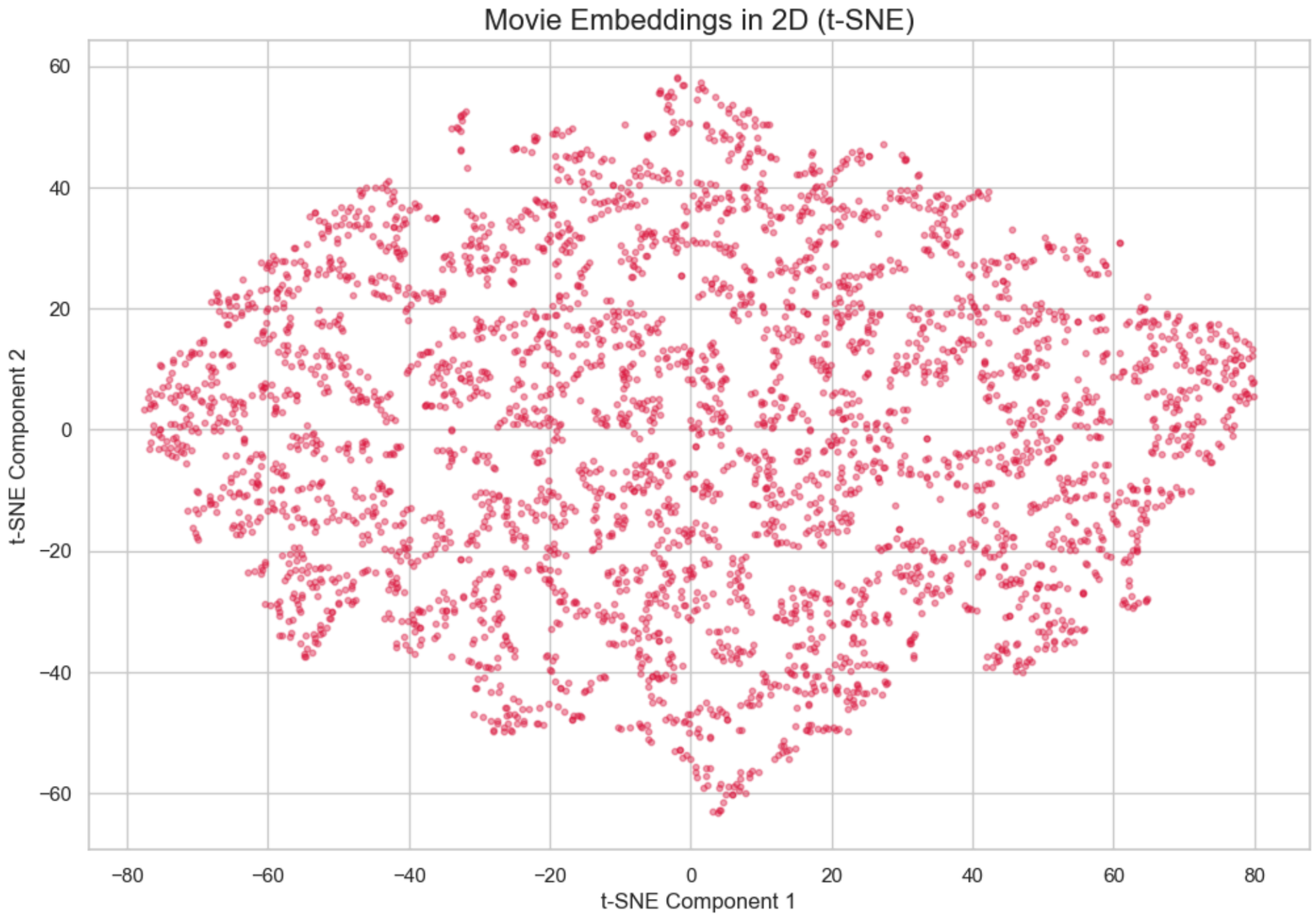
5.4.1 - Bonus: Visualize Movie Embeddings using PCA (2D Projection)

In [62]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Reduce 4D embeddings to 2D using PCA
pca = PCA(n_components=2)
item_embed_2d = pca.fit_transform(item_embed_df.values)

# Create a DataFrame for plotting
pca_df = pd.DataFrame(item_embed_2d, columns=["PC1", "PC2"])
pca_df["Title"] = item_embed_df.index

# Plot the embeddings
plt.figure(figsize=(12, 8))
plt.scatter(pca_df["PC1"], pca_df["PC2"], alpha=0.4, s=10, color='steelblue')
plt.title("Movie Embeddings in 2D (PCA-reduced)", fontsize=16)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()
```



5.4.2 - Bonus: Visualize Movie Embeddings using t-SNE (Nonlinear 2D Projection)

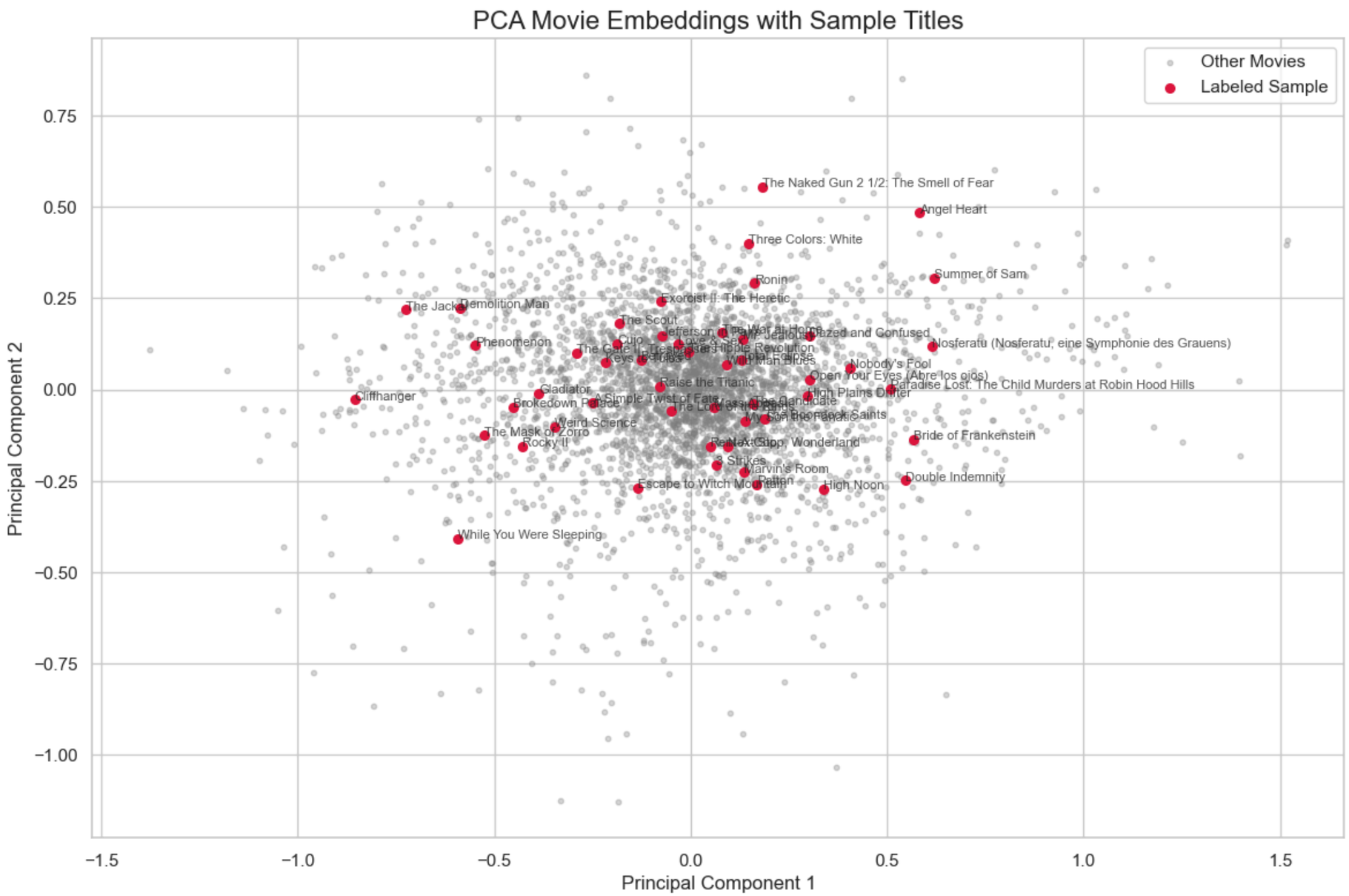
In [63]:
```python
# t-SNE Visualization of Movie Embeddings

from sklearn.manifold import TSNE

# Reduce embeddings to 2D using t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30, learning_rate=200)
item_embed_2d_tsne = tsne.fit_transform(item_embed_df.values)

# Create DataFrame
tsne_df = pd.DataFrame(item_embed_2d_tsne, columns=["x", "y"])
tsne_df["Title"] = item_embed_df.index

# Plot
plt.figure(figsize=(12, 8))
plt.scatter(tsne_df["x"], tsne_df["y"], alpha=0.4, s=10, color='crimson')
plt.title("Movie Embeddings in 2D (t-SNE)", fontsize=16)
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.grid(True)
plt.show()
```



5.4.3 - Bonus: Annotated PCA Plot with Sampled Movie Titles

In [64]:
```python
# Add Titles to PCA Plot (Sampled for Clarity)

import random

# Sample a subset of titles to annotate (e.g. 50 movies max)
sampled = pca_df.sample(n=50, random_state=42)

plt.figure(figsize=(14, 9))

# Scatter all 2481 points in gray (this is the full set of embeddings)
plt.scatter(pca_df["PC1"], pca_df["PC2"], alpha=0.3, s=10, color='gray', label='Other Movies')

# Plot the 50 sampled ones in a different color (e.g., crimson)
plt.scatter(sampled["PC1"], sampled["PC2"], color='crimson', s=30, label='Labeled Sample')

# Then overlay labels for just 50 randomly sampled points
for _, row in sampled.iterrows():
    plt.text(row["PC1"], row["PC2"], row["Title"], fontsize=8, alpha=0.8)
```

```
plt.title("PCA Movie Embeddings with Sample Titles", fontsize=16)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.grid(True)
plt.show()
```



5.4.4 - Bonus: PCA Plot Colored by Dominant Genre

In [65]:
```
print(df[['Title', 'Genres']].head())
```

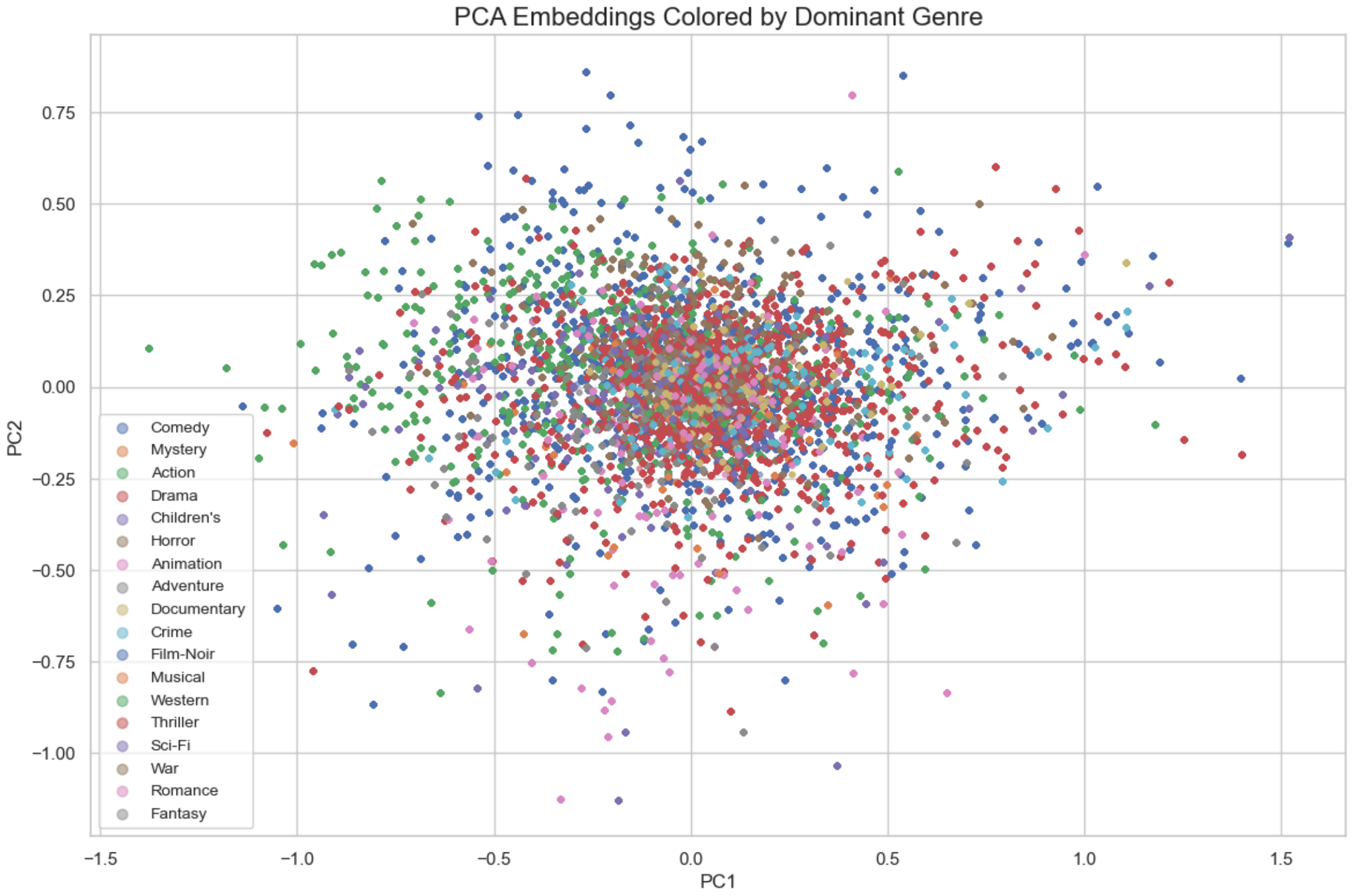
```
                          Title                          Genres
0  One Flew Over the Cuckoo's Nest                       [Drama]
1       James and the Giant Peach  [Animation, Children's, Musical]
2                    My Fair Lady               [Musical, Romance]
3                 Erin Brockovich                       [Drama]
4                    A Bug's Life   [Animation, Children's, Comedy]
```

In [66]:
```
# Fix genre column if it's a list
df["Genres"] = df["Genres"].apply(lambda x: '|'.join(x) if isinstance(x, list) else x)

# Extract dominant genre
df['DominantGenre'] = df['Genres'].apply(
    lambda x: x.split('|')[0] if isinstance(x, str) and '|' in x else x
)

# Clean titles for merge
df["Title_clean"] = df["Title"].str.strip().str.lower()
pca_df["Title_clean"] = pca_df["Title"].str.strip().str.lower()

# Merge genre into PCA data
genre_pca_df = pca_df.merge(df[["Title_clean", "DominantGenre"]], on="Title_clean", how="left")
genre_pca_df["DominantGenre"] = genre_pca_df["DominantGenre"].fillna("Unknown")
```

In [67]:
```
print("Unique Genres Found:", genre_pca_df["DominantGenre"].nunique())
print(genre_pca_df["DominantGenre"].value_counts())
```

```
Unique Genres Found: 18
DominantGenre
Comedy         276919
Action         257454
Drama          208606
Horror          44170
Adventure       43629
Crime           37848
Animation       36936
Children's      21491
Thriller        17851
Sci-Fi          11464
Mystery         10237
Film-Noir        9343
Musical          7112
Documentary      6812
Western          5689
Romance          2831
War               991
Fantasy           790
Name: count, dtype: int64
```

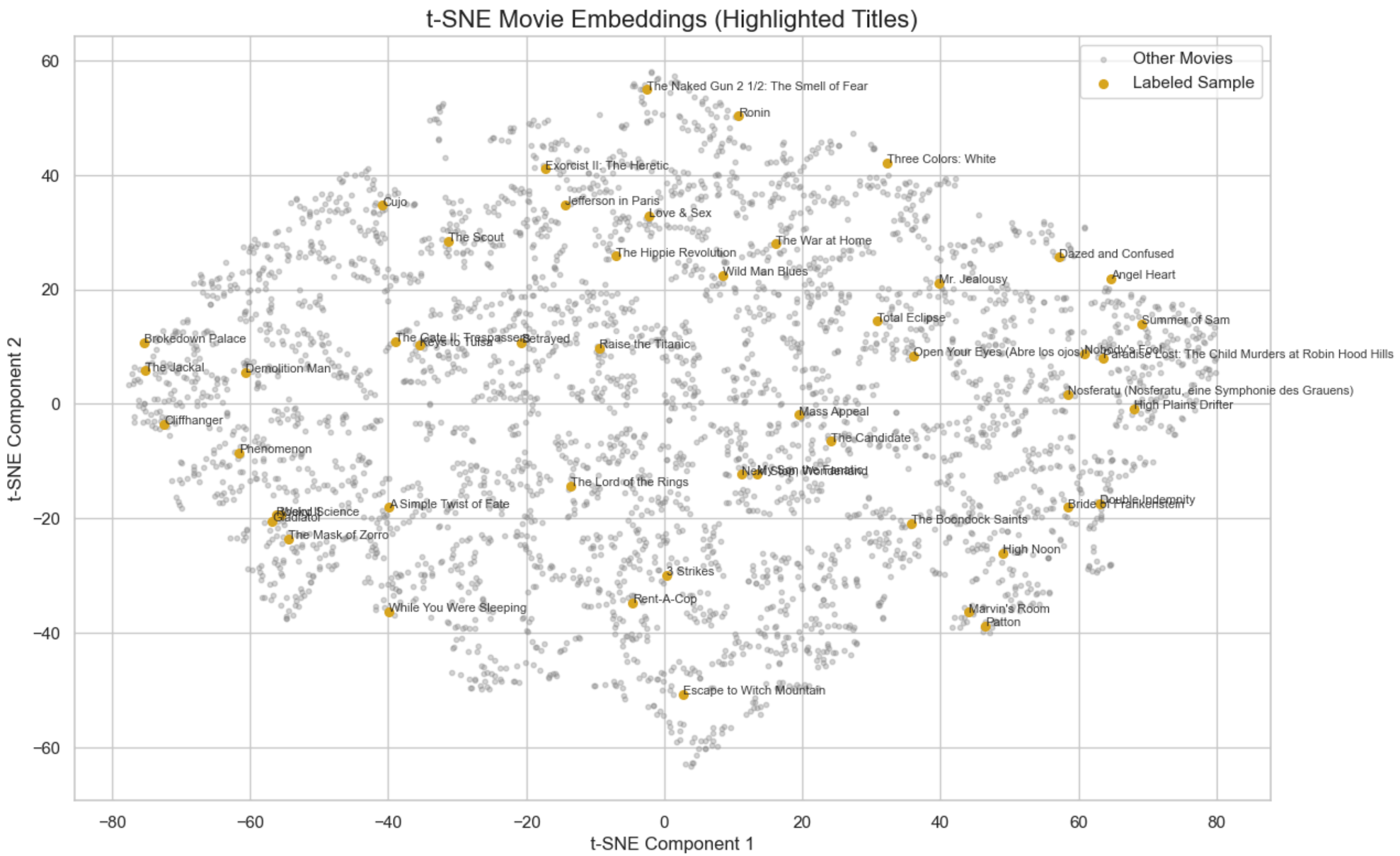
In [68]:
```
# Plot with color by genre
plt.figure(figsize=(14, 9))
for genre in genre_pca_df["DominantGenre"].unique():
    subset = genre_pca_df[genre_pca_df["DominantGenre"] == genre]
    plt.scatter(subset["PC1"], subset["PC2"], label=genre, alpha=0.5, s=10)

plt.title("PCA Embeddings Colored by Dominant Genre", fontsize=16)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(loc='best', fontsize='small', markerscale=2)
plt.grid(True)
plt.show()
```



5.4.5 - Bonus: Annotated t-SNE Plot with Sampled Movie Titles

In [69]:
```python
# Sample 50 movies from the t-SNE result
sampled_tsne = tsne_df.sample(n=50, random_state=42)

# Plot ALL movies in light gray
plt.figure(figsize=(14, 9))
plt.scatter(tsne_df["x"], tsne_df["y"], alpha=0.3, s=10, color='gray', label='Other Movies')

# Highlight the sampled 50 in a different color (e.g., gold)
plt.scatter(sampled_tsne["x"], sampled_tsne["y"], color='goldenrod', s=30, label='Labeled Sample')

# Annotate sampled movies
for _, row in sampled_tsne.iterrows():
    plt.text(row["x"], row["y"], row["Title"], fontsize=8, alpha=0.9)

plt.title("t-SNE Movie Embeddings (Highlighted Titles)", fontsize=16)
plt.xlabel("t-SNE Component 1")
```

```
plt.ylabel("t-SNE Component 2")
plt.legend()
plt.grid(True)
plt.show()
```



📌 Bonus Insight: Embedding Visualization Analysis

- The **PCA plot** shows a **Dense Central Cluster** with some scattered outliers, indicating that linear components capture overlapping variance across movie embeddings — a sign of **Linear Factor Overlap** among genres.
- The **t-SNE plot**, by contrast, reveals **Well-Separated, Loosely Packed Clusters**, highlighting that **t-SNE Preserves Local Similarity Structures** more effectively than PCA.
- Across both Visualizations, **Labeled Movies tend to Cluster by Genre or Thematic similarity**:
  - For example, action-oriented films like *"Gladiator"*, *"Demolition Man"*, and *"The Mask of Zorro"* are consistently positioned close together — reflecting latent semantic proximity.
  - Similarly, films like *"While You Were Sleeping"* and *"A Simple Twist of Fate"* (romantic/drama) form nearby groupings.

- In the **Genre-Colored PCA plot**, genres like **Drama**, **Comedy**, and **Action** dominate the central region, while niche genres like **Film-Noir**, **Fantasy**, and **War** scatter more sparsely — indicating either embedding sparsity or unique stylistic signatures.
- **Cluster Density in the t-SNE plot** varies across regions. Certain tight clusters may represent movies with strong semantic coherence (e.g., series, sequels, or same-director films), while dispersed areas might represent genre hybrids or niche productions.
- These visuals together support the interpretation that **Matrix Factorization uncovers Latent Themes** and **Semantic Relationships** that go beyond explicit genre labels — capturing similarities based on storyline, tone, or audience engagement.

📝 Note -

- `PCA` is **faster and linear** → Good for **Variance-Based Structure**.
- `t-SNE` is **non-linear** → Better for Visualizing **Semantic Groupings**.

5.5 - (Optional) User-Based Collaborative Filtering

5.5.1 - Creating a New User Profile & Finding Overlapping Users

In [70]:
```
# Create New User Ratings

# Create new user with manual ratings
new_user_ratings = pd.DataFrame({
    'UserID': [9999]*5,
    'Title': [
        "The Matrix",
        "Titanic",
        "Toy Story (1995)",
        "Star Wars: Episode IV - A New Hope (1977)",
        "The Godfather"
    ],
    'Rating': [5, 4, 5, 4, 5]
})

new_user_ratings
```

Out[70]:

	UserID	Title	Rating
0	9999	The Matrix	5
1	9999	Titanic	4
2	9999	Toy Story (1995)	5
3	9999	Star Wars: Episode IV - A New Hope (1977)	4
4	9999	The Godfather	5

In [71]:
```
# Filter Overlapping Users

# Get movie titles rated by new user
target_movies = new_user_ratings["Title"].tolist()

# Get all ratings from users who rated at least one of these movies
overlap_users = df[df["Title"].isin(target_movies)]

# Exclude the new user if already appended to df
overlap_users = overlap_users[overlap_users["UserID"] != 9999]

print(f"🎯 Found {overlap_users['UserID'].nunique()} users who rated at least one of the same movies.")
overlap_users.head()
```

🎯 Found 4013 users who rated at least one of the same movies.

Out[71]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Year	Gender	Age	Occupation	Zip-code	WatchDate	WatchYear	WatchMonth	WatchHour	WatchDay	ReleaseDecade	WatchMonthName	DominantGenre	T
27	1	1721	4	2000-12-31 22:00:55	Titanic	Drama|Romance	1997	F	Under 18	K-12 Student	48067	2000-12-31 22:00:55	2000	12	22	Sunday	1990s	December	Drama	
135	2	2571	4	2000-12-31 21:56:13	The Matrix	Action|Sci-Fi|Thriller	1999	M	56+	Self-Employed	70072	2000-12-31 21:56:13	2000	12	21	Sunday	1990s	December	Action	
326	5	2571	5	2000-12-31 06:34:53	The Matrix	Action|Sci-Fi|Thriller	1999	M	25-34	Writer	55455	2000-12-31 06:34:53	2000	12	6	Sunday	1990s	December	Action	
375	5	1721	1	2000-12-31 06:56:03	Titanic	Drama|Romance	1997	M	25-34	Writer	55455	2000-12-31 06:56:03	2000	12	6	Sunday	1990s	December	Drama	
536	7	2571	5	2000-12-31 03:53:06	The Matrix	Action|Sci-Fi|Thriller	1999	M	35-44	Academic/Educator	06810	2000-12-31 03:53:06	2000	12	3	Sunday	1990s	December	Action	

5.5.2 - Calculating User Similarities using Pearson Correlation

In [72]:
```
# Pearson Similarity Between Users

# Append new user to overlap users
user_cf_data = pd.concat([overlap_users, new_user_ratings], ignore_index=True)

# Create pivot: Users × Movies
user_movie_matrix = user_cf_data.pivot_table(index='UserID', columns='Title', values='Rating')

# Compute Pearson correlation of new user with others
user_similarities = user_movie_matrix.T.corr(method='pearson')

# Extract similarity scores for the new user (UserID 9999)
new_user_sim_scores = user_similarities[9999].drop(labels=[9999]).dropna().sort_values(ascending=False)

# Top 10 most similar users
new_user_sim_scores.head(10)
```

Out[72]:
```
UserID
5015    1.0
2050    1.0
770     1.0
3413    1.0
720     1.0
5094    1.0
5090    1.0
1880    1.0
3471    1.0
4382    1.0
Name: 9999, dtype: float64
```

5.5.3 - Generating User-Based Recommendations (Weighted by Similarity)

In [73]:
```
# Generate Weighted Recommendations

# Limit to top N similar users
top_sim_users = new_user_sim_scores.head(100)

# Get ratings from top similar users
similar_users_ratings = df[df["UserID"].isin(top_sim_users.index)]

# Exclude movies already rated by new user
already_rated = new_user_ratings["Title"].tolist()
```

```
candidate_ratings = similar_users_ratings[~similar_users_ratings["Title"].isin(already_rated)]

# Merge similarity scores with ratings
candidate_ratings = candidate_ratings.merge(top_sim_users.rename("Similarity"), left_on="UserID", right_index=True)

# Compute weighted rating
candidate_ratings["WeightedRating"] = candidate_ratings["Rating"] * candidate_ratings["Similarity"]

# Compute final score: sum(weighted ratings) / sum(similarities)
recommend_df = candidate_ratings.groupby("Title").agg(
    TotalWeightedRating=("WeightedRating", "sum"),
    TotalSimilarity=("Similarity", "sum"),
    NumRatings=("UserID", "count")
)

recommend_df["FinalScore"] = recommend_df["TotalWeightedRating"] / recommend_df["TotalSimilarity"]

# Sort by final recommendation score
# OPTIONAL FILTER: Only include movies rated by at least 5 similar users
top_recommendations = recommend_df[recommend_df["NumRatings"] >= 5]

# Sort and select top 10
top_recommendations = top_recommendations.sort_values(by="NumRatings", ascending=False).head(10)
# top_recommendations = top_recommendations.sort_values(by="FinalScore", ascending=False).head(10)

top_recommendations[["FinalScore", "NumRatings"]]
```

Out[73]:

Title	FinalScore	NumRatings
Star Wars: Episode V - The Empire Strikes Back	4.445652	92
Star Wars: Episode IV - A New Hope	4.444444	90
Fargo	4.261364	88
Men in Black	3.511364	88
The Silence of the Lambs	4.413793	87
Pulp Fiction	4.534884	86
Terminator 2: Judgment Day	4.176471	85
American Beauty	4.517647	85
Saving Private Ryan	4.352941	85
The Fugitive	4.036145	83

5.6 - Final Top-N Comparison & Wrap-up

5.6.1 - Comparing Top-N Recommendations Across Models

In [74]:
```
# Recommendation Comparison

# Pearson
pearson_recs = recommend_similar_movies_pearson("The Matrix", item_corr_matrix).reset_index()
pearson_recs.columns = ['Title', 'Pearson_Score']

# Cosine (KNN)
cosine_recs = recommend_similar_movies_knn("The Matrix", movie_user_matrix, knn_model).reset_index(drop=True)
cosine_recs.columns = ['Title', 'Cosine_Score']
```

```
# SVD Embeddings
svd_recs = get_similar_movies_from_embeddings("The Matrix", item_embed_df).reset_index(drop=True)
svd_recs.columns = ['Title', 'SVD_Score']

# Merge into one table
comparison_df = pearson_recs.merge(cosine_recs, on='Title', how='outer') \
                            .merge(svd_recs, on='Title', how='outer') \
                            .fillna('-')

comparison_df
```

Out[74]:

	Title	Pearson_Score	Cosine_Score	SVD_Score
0	Bed of Roses	0.503039	-	-
1	Dead Man on Campus	0.420561	-	-
2	Grace of My Heart	0.412888	-	-
3	In the Line of Duty 2	-	-	0.976307
4	Men in Black	-	0.684763	-
5	Party Girl	-	-	0.997349
6	Payback	-	-	0.976667
7	Queen Margot (La Reine Margot)	0.477035	-	-
8	Star Wars: Episode IV - A New Hope	-	0.680378	-
9	Star Wars: Episode V - The Empire Strikes Back	-	0.689459	-
10	Terminator 2: Judgment Day	-	0.745532	-
11	The Cure	-	-	0.991563
12	The Gods Must Be Crazy II	0.393565	-	-
13	Total Recall	-	0.703265	-
14	White Men Can't Jump	-	-	0.988254

📌 Section 5: Recommender Systems — Final Summary

Purpose

This section explored and compared **three collaborative filtering techniques** for generating movie recommendations:

1. **Pearson Correlation**
2. **Cosine Similarity (KNN)**
3. **Matrix Factorization (SVD)**

We analyzed their behavior using the movie **"The Matrix"** as a reference title.

Model-Wise Behavior

Recommender	What It Measures	Sample Recommendations	Strengths	Limitations
Pearson Correlation	Linear Similarity between Co-Rated Movies	🎬 *Bed of Roses*, *Dead Man on Campus*, *Grace of My Heart*	✅ Easy to explain ✅ Good with Overlap	❌ Sparse for Low-Activity Users/Movies
Cosine Similarity (KNN)	Angular Similarity in Rating Patterns (Ignores Scale)	🎬 *Terminator 2*, *Star Wars*, *Total Recall*	✅ Fast ✅ Effective with Sparse Data	❌ Ignores Rating Values (Just Patterns)

Recommender	What It Measures	Sample Recommendations	Strengths	Limitations
Matrix Factorization (SVD)	Latent Semantic Relationships via Learned Embeddings	🎬 *The Cure, White Men Can't Jump, Payback*	✅ Captures deep themes ✅ Best performance	❌ Requires training ❌ Less interpretable

Results Comparison: Recommendations for `"The Matrix"`

✅ Pearson Correlation

- *Bed of Roses* (0.50)
- *Dead Man on Campus* (0.42)
- *Grace of My Heart* (0.41)
- *Queen Margot* (0.47)
- *The Gods Must Be Crazy II* (0.39)

✅ Cosine Similarity (KNN)

- *Terminator 2: Judgment Day* (0.74)
- *Total Recall* (0.70)
- *Men in Black* (0.68)
- *Star Wars: Episode IV* (0.68)
- *Star Wars: Episode V* (0.68)

✅ Matrix Factorization (SVD)

- *The Cure* (0.99)
- *Party Girl* (0.99)
- *White Men Can't Jump* (0.98)
- *Payback* (0.97)
- *In the Line of Duty 2* (0.97)

📝 Key Takeaways

✅ Each Model views **"Similarity"** Differently:

- **Pearson**: "These movies were *Rated Similarly by the same users*"
- **Cosine**: "These movies *Received Similar rating patterns*"
- **SVD**: "These movies are *Thematically/Semantically close*, even without explicit overlap"

✅ A **Hybrid Strategy** can Leverage the Strengths of Multiple Models:

- Use *__SVD__* for `Deep Structure` & `Cold Start`
- Use *__Cosine__* for `Scalable Pattern Matching`
- Use *__Pearson__* where `Overlap Exists` (Niche Clusters)

✅ When to Use Each Model

Scenario	Recommended Model
Cold-start (new user/movie)	Matrix Factorization (SVD)
Sparse but wide data coverage	Cosine Similarity (KNN)
Strong user-user rating overlap	Pearson Correlation
Want best of both (robust + smart)	Hybrid (SVD + KNN)

📝 Final Recommendation Strategy

- **Filter low-count items** to avoid unreliable suggestions
- **Ensemble recommenders** to balance precision and coverage
- **Visualize embeddings** to understand movie clusters and model behavior

❓ Questionnarie

✅ 1. Users of which age group have watched and rated the most number of movies?

Answer:

- Users in the **25–34 Age group** have watched and rated the most number of movies - `395556` Ratings.
  - With **35-44** `(199003)` & **18-24** `(183536)` Age Groups coming in 2nd & 3rd Places Respectively.

✅ 2. Users belonging to which profession have watched and rated the most movies?

Answer:

- Users with the occupation code **"4 – college/grad student"** have rated the most number of movies in the dataset - `131032`.
  - With **Other** `(130499)` & **Executive/Managerial** `(105425)` Occupation Groups coming in 2nd & 3rd Places Respectively.

✅ 3. Most of the users in our dataset who've rated the movies are Male. (T/F)

Answer:

- **True**. A majority of the users in the dataset are **Male** - `753,769` Users (*~75 %*), as seen from the gender distribution pie chart.
  - While **Female** - `246,440` Users (*~25%*) are Comparitively Smaller in Size.

✅ 4. Most of the movies present in our dataset were released in which decade?

Answer:

- **(b) 1990s** — The majority of movies in the dataset were released in the 1990s - `532843 Movies`.
  - With **1980s** `(224056)` & **1970s** `(82552)` coming in 2nd & 3rd Places Respectively.

✅ 5. The movie with maximum no. of ratings is ___.

Answer:

- The movie with the highest number of ratings is **"American Beauty"** - `3428`.
  - With **Star Wars: Episode IV - A New Hope** `(2991)` & **Star Wars: Episode V - The Empire Strikes Back** `(2990)` Movies coming in 2nd & 3rd Places Respectively.

✅ 6. Name the top 3 movies similar to 'Liar Liar' on the item-based approach.

Answer:

- Based on the Pearson correlation item-based approach, the top 3 similar movies to *Liar Liar* are:

Title - **Correlation Coefficient**

- 1. *Life* - 0.576
- 2. *Oliver & Company* - 0.550
- 3. *Spy Hard* - 0.502
- 4. *Ace Ventura: when Nature Calls* - 0.495
- 5. *Dead man on Campus* - 0.478
- Based on the Cosine Similarity (KNN Model) item-based approach, the top 3 similar movies to *Liar Liar* are:

Title - **Cosine Similarity**

- 1. *Mrs. Doubtfire* - 0.557
- 2. *Ace Ventura: Pet Detective* - 0.516
- 3. *Dumb & Dumber* - 0.512
- 4. *Home Alone* - 0.511
- 5. *Wayne's World* - 0.499

✅ 7. On the basis of approach, Collaborative Filtering methods can be classified into ___-based and ___-based.

Answer:

- Collaborative Filtering can be classified into **user-based** and **item-based** approaches.

✅ 8. Pearson Correlation ranges between ___ to ___ whereas, Cosine Similarity belongs to the interval between ___ to ___.

Answer:

- Pearson Correlation: ranges from **–1 to +1**
- Cosine Similarity: ranges from **0 to 1**

✅ 9. Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.

Answer:

- ▪ RMSE: **0.8834**
- ▪ MAPE: **26.95%**

✅ 10. Give the sparse 'row' matrix representation for the following dense matrix -

`[[1 0], [3 7]]`

Answer:

```
from scipy.sparse import csr_matrix
dense = np.array([[1, 0], [3, 7]])
sparse = csr_matrix(dense)
print(sparse)
```

Output:

```
  (0, 0)        1
  (1, 0)        3
  (1, 1)        7
```

So the sparse row format is:

- **[(0,0)=1, (1,0)=3, (1,1)=7]**

🎯 Key Strategic Recommendations

1. **Adopt a Hybrid Recommender System**
    - Combine **SVD (latent features)** with **KNN (collaborative signals)** to capture both **semantic similarity** and **behavioral overlap**.
    - Use **Pearson** for cases with rich rating history, and **Cosine** for faster cold-start scenarios.
2. **Leverage Viewing Time Patterns**
    - Focus **content drops and notifications** around **evening hours (6–10 PM)** and **early weekdays (Mon–Tue)** to maximize engagement.
    - Use **seasonal spikes** (e.g., November, August) to time promotional campaigns and platform announcements.
3. **Genre & Demographic Targeting**
    - Double down on **high-engagement genres** (Drama, Comedy, Action) for mass appeal.
    - Curate **genre-decade clusters** (e.g., 90s Action, 80s Sci-Fi) to deepen user connection and content discovery.
    - Personalize recommendations by user traits (e.g., age, occupation), which show clear behavioral distinctions.
4. **Address Sparsity and User Diversity**
    - With a sparsity level of **95.5%**, continue using **matrix factorization techniques** and **embedding-based models**.
    - Drive inclusivity by promoting **tailored UI/UX** and onboarding flows for **under-represented segments** (e.g., older users, female users, niche professions).
5. **Content Strategy from Top-Rated & Most-Rated Titles**
    - Use high-engagement movies like *American Beauty*, *Star Wars*, and *The Matrix* as **anchor titles** in carousels or “Because You Watched” lists.
    - Promote **top-rated classics** (e.g., *Seven Samurai*, *Shawshank Redemption*) to boost **credibility and user satisfaction**.