In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy
from datetime import timedelta
from pylab import rcParams
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics import tsaplots
```

In [3]:
```
df_ratings = pd.read_csv('/Users/Ramv/Downloads/zee-ratings.dat', delimiter=
df_users = pd.read_csv('/Users/Ramv/Downloads/zee-users.dat', delimiter = ':
df_movies = pd.read_csv('/Users/Ramv/Downloads/zee-movies.dat', delimiter =
```

```
/var/folders/fx/1km2ndm10xxcmn5xy9fsdksm0000gn/T/ipykernel_85204/2542404622.
py:1: ParserWarning: Falling back to the 'python' engine because the 'c' eng
ine does not support regex separators (separators > 1 char and different fro
m '\s+' are interpreted as regex); you can avoid this warning by specifying
engine='python'.
  df_ratings = pd.read_csv('/Users/Ramv/Downloads/zee-ratings.dat', delimite
r="::")
/var/folders/fx/1km2ndm10xxcmn5xy9fsdksm0000gn/T/ipykernel_85204/2542404622.
py:2: ParserWarning: Falling back to the 'python' engine because the 'c' eng
ine does not support regex separators (separators > 1 char and different fro
m '\s+' are interpreted as regex); you can avoid this warning by specifying
engine='python'.
  df_users = pd.read_csv('/Users/Ramv/Downloads/zee-users.dat', delimiter =
'::')
/var/folders/fx/1km2ndm10xxcmn5xy9fsdksm0000gn/T/ipykernel_85204/2542404622.
py:3: ParserWarning: Falling back to the 'python' engine because the 'c' eng
ine does not support regex separators (separators > 1 char and different fro
m '\s+' are interpreted as regex); you can avoid this warning by specifying
engine='python'.
  df_movies = pd.read_csv('/Users/Ramv/Downloads/zee-movies.dat', delimiter
= '::',encoding='ISO-8859-1')
```

In [4]:
```
print(f'Shape for Ratings DF: {df_ratings.shape}')
print(f'Shape for Users DF: {df_users.shape}')
print(f'Shape for Movies DF: {df_movies.shape}')
```

```
Shape for Ratings DF: (1000209, 4)
Shape for Users DF: (6040, 5)
Shape for Movies DF: (3883, 3)
```

In [5]:
```
df_ratings
```

Out[5]:

	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
...	...	...	...	...
1000204	6040	1091	1	956716541
1000205	6040	1094	5	956704887
1000206	6040	562	5	956704746
1000207	6040	1096	4	956715648
1000208	6040	1097	4	956715569

1000209 rows × 4 columns

In [6]:
```
df_users
```

Out[6]:

	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
...	...	...	...	...	...
6035	6036	F	25	15	32603
6036	6037	F	45	1	76006
6037	6038	F	56	1	14706
6038	6039	F	45	0	01060
6039	6040	M	25	6	11106

6040 rows × 5 columns

In [7]:
```
df_movies
```

Out[7]:

	Movie ID	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
...	...	...	...
3878	3948	Meet the Parents (2000)	Comedy
3879	3949	Requiem for a Dream (2000)	Drama
3880	3950	Tigerland (2000)	Drama
3881	3951	Two Family House (2000)	Drama
3882	3952	Contender, The (2000)	Drama|Thriller

3883 rows × 3 columns

In [8]:
```
# Checking for NaN's
print(f'NaNs in Ratings DF: {df_ratings.isna().sum().sum()}')
print(f'Nans in Users DF: {df_users.isna().sum().sum()}')
print(f'Nans in Movies DF: {df_movies.isna().sum().sum()}')
```

```
NaNs in Ratings DF: 0
Nans in Users DF: 0
Nans in Movies DF: 0
```

In [9]:
```
# Dtypes in Datasets
print(f'Datatypes in Ratings DF:\n{df_ratings.dtypes}')
print(f'\nDatatypes in Users DF:\n{df_users.dtypes}')
print(f'\nDatatypes in Movies DF:\n{df_movies.dtypes}')
```

```
Datatypes in Ratings DF:
UserID       int64
MovieID      int64
Rating       int64
Timestamp    int64
dtype: object

Datatypes in Users DF:
UserID         int64
Gender        object
Age            int64
Occupation     int64
Zip-code      object
dtype: object

Datatypes in Movies DF:
Movie ID     int64
Title       object
Genres      object
dtype: object
```

In [10]:
```
# Duplicates in Datasets
print(f'Duplicates in Ratings DF: {df_ratings.duplicated().sum()}')
print(f'Duplicates in Users DF: {df_users.duplicated().sum()}')
print(f'Duplicates in Movies DF: {df_movies.duplicated().sum()}')
```

```
Duplicates in Ratings DF: 0
Duplicates in Users DF: 0
Duplicates in Movies DF: 0
```

Preparing Data

In [11]:
```
# Renaming Columns to maintain consistency
df_movies.rename(columns = {'Movie ID':'MovieID'}, inplace = True)
```

In [12]:
```
df_merged = pd.merge(left = df_ratings, right = df_movies, on = "MovieID", h
df_merged = pd.merge(left = df_merged, right = df_users, on = 'UserID', how
df_merged.head()
```

Out[12]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Ag
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama	F	
1	1	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical	F	
2	1	914	3	978301968	My Fair Lady (1964)	Musical|Romance	F	
3	1	3408	4	978300275	Erin Brockovich (2000)	Drama	F	
4	1	2355	5	978824291	Bug's Life, A (1998)	Animation|Children's|Comedy	F	

In [13]:
```
# Extracting Years

import re

def get_year(title):
    year = re.search('\(\d*\)', title)
    return year[0][1:5]

df_merged['Year'] = df_merged['Title'].map(get_year)
```

In [14]:
```
# Cleaning Title
def clean_title(title):
    year = re.search('\(\d*\)', title)[0]
    title =  re.search('.*\(\d\d\d\d\)', title)[0][:-6]

    parts = title.split(', ')
    if len(parts) == 2 and parts[1].lower().strip() in ["the", "a", "an"]:
        title = parts[1] + " " + parts[0]
    return title + " " + year

# Apply the function to the 'Title' column
df_merged['Title'] = df_merged['Title'].map(clean_title)
```

In [15]:
```
# Handling Timestamp
df_merged['Timestamp'] = pd.to_datetime(df_merged['Timestamp'], unit = 's')
```

In [16]:
```
# Extractng Rating Year, Month, Date
df_merged['RatingDate'] = df_merged['Timestamp'].dt.day
df_merged['RatingMonth'] = df_merged['Timestamp'].dt.month
df_merged['RatingYear'] = df_merged['Timestamp'].dt.year

# Dropping Timestamp
df_merged.drop(columns=['Timestamp'], inplace = True)
```

In [17]:
```
# Dataset Shape
df_merged.shape
```

Out[17]:
```
(1000209, 13)
```

In [18]:
```
# Dataset Nan's
df_merged.isna().sum()
```

Out[18]:
```
UserID         0
MovieID        0
Rating         0
Title          0
Genres         0
Gender         0
Age            0
Occupation     0
Zip-code       0
Year           0
RatingDate     0
RatingMonth    0
RatingYear     0
dtype: int64
```

In [19]:
```
# DataTypes
df_merged.dtypes
```

Out[19]:
```
UserID          int64
MovieID         int64
Rating          int64
Title          object
Genres         object
Gender         object
Age             int64
Occupation      int64
Zip-code       object
Year           object
RatingDate      int64
RatingMonth     int64
RatingYear      int64
dtype: object
```

In [20]:
```
# Handling Zipcodes
df_merged[~df_merged['Zip-code'].str.isnumeric()].head()
```

Out[20]:

	UserID	MovieID	Rating	Title	Genres	Gender	Age	Occupation
21847	161	2988	4	Melvin and Howard (1980)	Drama	M	45	16
21848	161	1179	4	The Grifters (1990)	Crime|Drama|Film-Noir	M	45	16
21849	161	3860	4	The Opportunists (1999)	Crime	M	45	16
21850	161	1252	5	Chinatown (1974)	Film-Noir|Mystery|Thriller	M	45	16
21851	161	1253	5	The Day the Earth Stood Still (1951)	Drama|Sci-Fi	M	45	16

Changing US format Zip Code to normal on

In [21]:
```
def get_zipcode(zip):
    code = re.search('^\d*-', zip)
    if code:
        return code[0][:-1]
    else:
        return zip

df_merged['Zip-code'] = df_merged['Zip-code'].map(get_zipcode)

# Handling Zip Code Datatype
df_merged['Zip-code'] = df_merged['Zip-code'].astype(int)
```

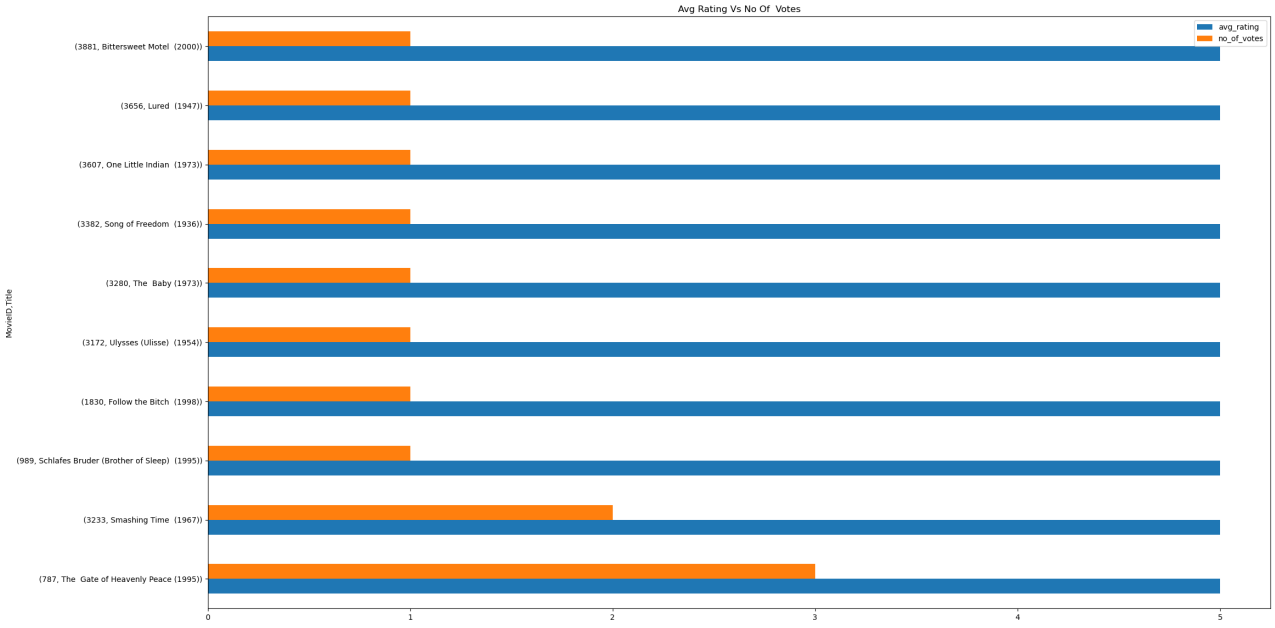
In [22]:
```
# Handling Data Types
df_merged['Year'] = df_merged['Year'].astype('int')
```

In [23]:
```
# Handling Genres
df_merged['Genres'] = df_merged['Genres'].str.split('|')
df_exploded = df_merged.explode('Genres')

# Rejoining Genres
df_merged['Genres'] = df_merged['Genres'].apply(lambda x : ', '.join(x))
```

Data Visualization

In [24]:
```
# Grouping data on Number of Votes and Average Rating
df_grouped = df_merged.groupby(['MovieID', 'Title'])['Rating'].agg(avg_ratin
df_grouped.sort_values(by = ['avg_rating', 'no_of_votes'], ascending = [Fals
                       title = "Avg Rating Vs No Of  Votes", figsize = (25,
plt.show()
```



Above method is not robust enough to draw Higher Sense in Analysis. We will find High rated movies on the basis of Bayesian Weighted Ratio

Weighted Rating =(v v + m ⋅) R + (m v + m) ⋅ C

v is the number of the votes

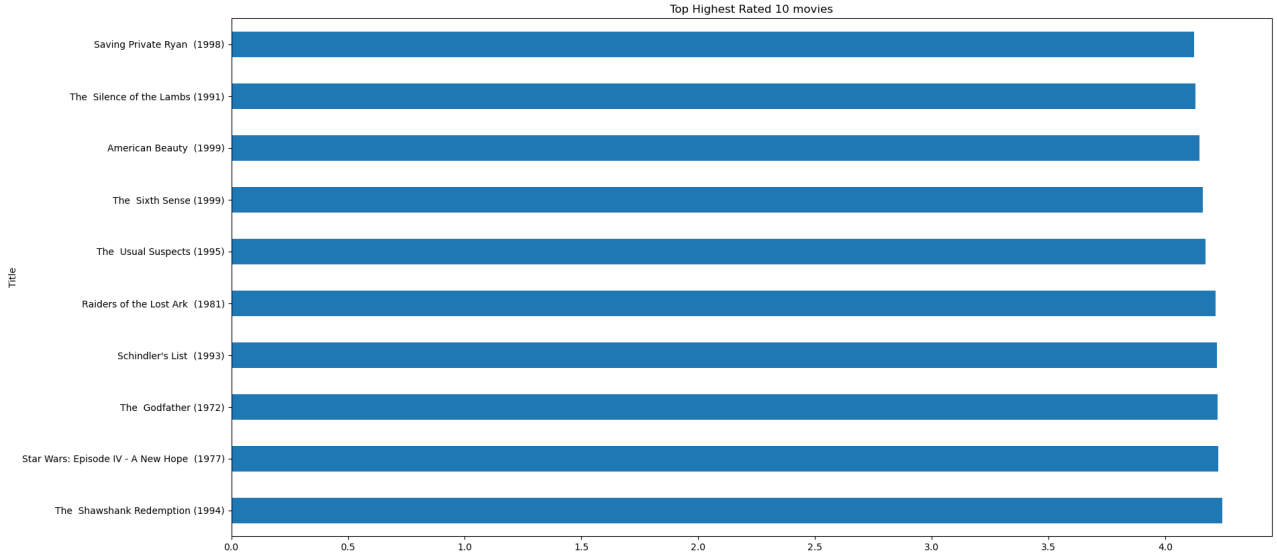
m is the minimum votes required to be listed

R is the average rating for the movie

C is the mean vote across the whole report

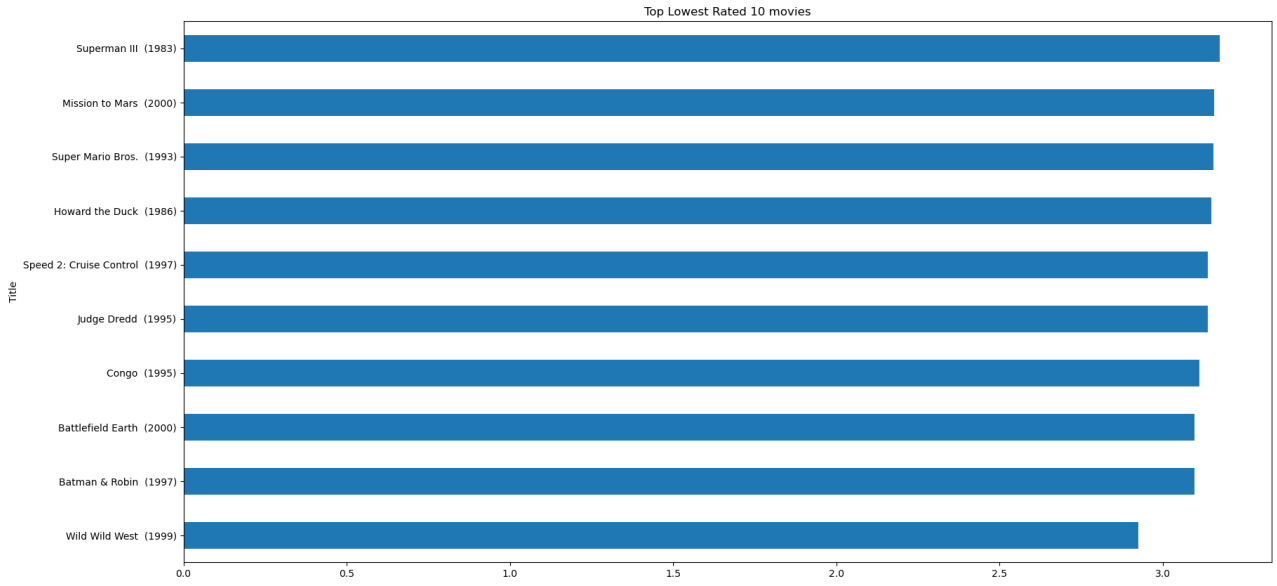
In [25]:
```
# Top Highest Rated movies According to ratio of number of votes
C = df_merged['Rating'].mean()
m = df_merged.groupby('Title')['Rating'].count().quantile(0.95)
v = df_merged.groupby('Title')['Rating'].count()
R = df_merged.groupby('Title')['Rating'].mean()
WR = (v/(v+m))*R + (m/(v+m))*C

WR.sort_values(ascending=False)[:10].plot(kind = 'barh', figsize=(20,10))
plt.title("Top Highest Rated 10 movies")
plt.show()
```



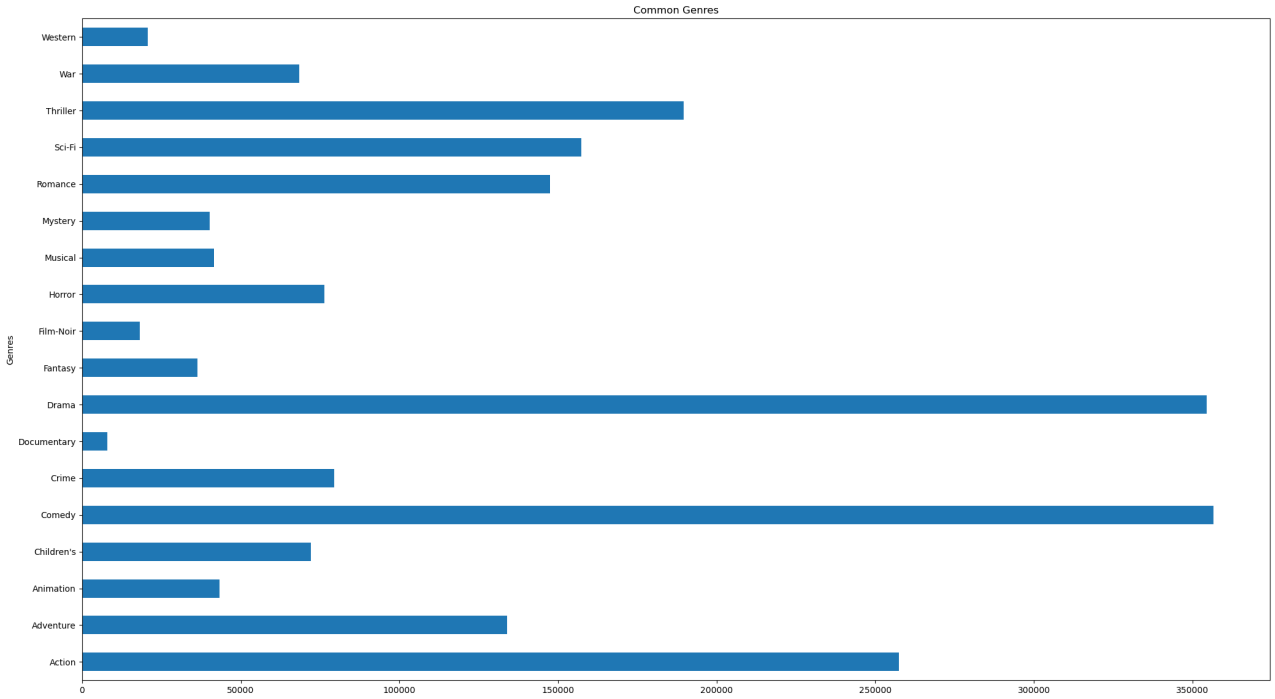
Shawshank Redemption is the top rated movie after applying Bayesian Weighted Ratio

In [26]:
```
# Top Lowest Rated movies According to ratio of number of votes
WR.sort_values(ascending=True)[:10].plot(kind = 'barh', figsize=(20 ,10))
plt.title("Top Lowest Rated 10 movies")
plt.show()
```



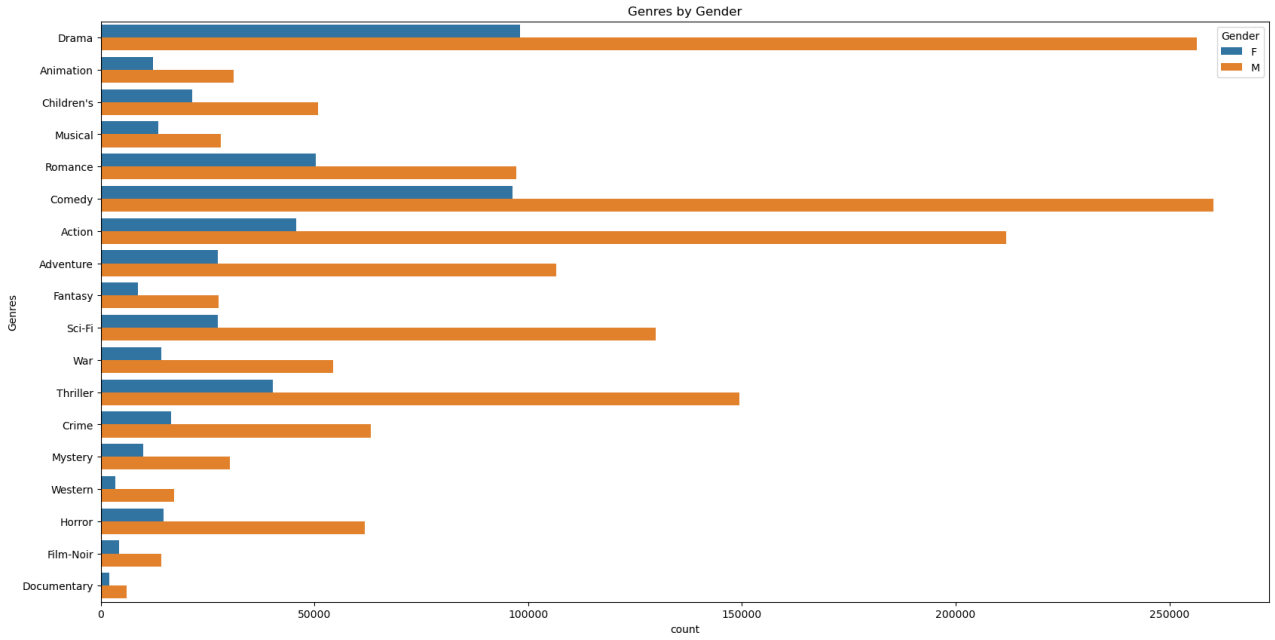
Wild Wild West is the bottom rated movie after applying Bayesian Weighted Ratio

In [27]:
```
# Common Genres Movies
df_exploded.groupby('Genres').size().plot(kind = 'barh', figsize=(25,14), ti
plt.show()
```



Drama is the most common genre of all

In [28]:
```
# Genres watched acccording to Gender
plt.figure(figsize=(20,10))
plt.title('Genres by Gender')
sns.countplot(data = df_exploded, y = 'Genres', hue='Gender')
plt.show()
```



Males prefer Action movies.

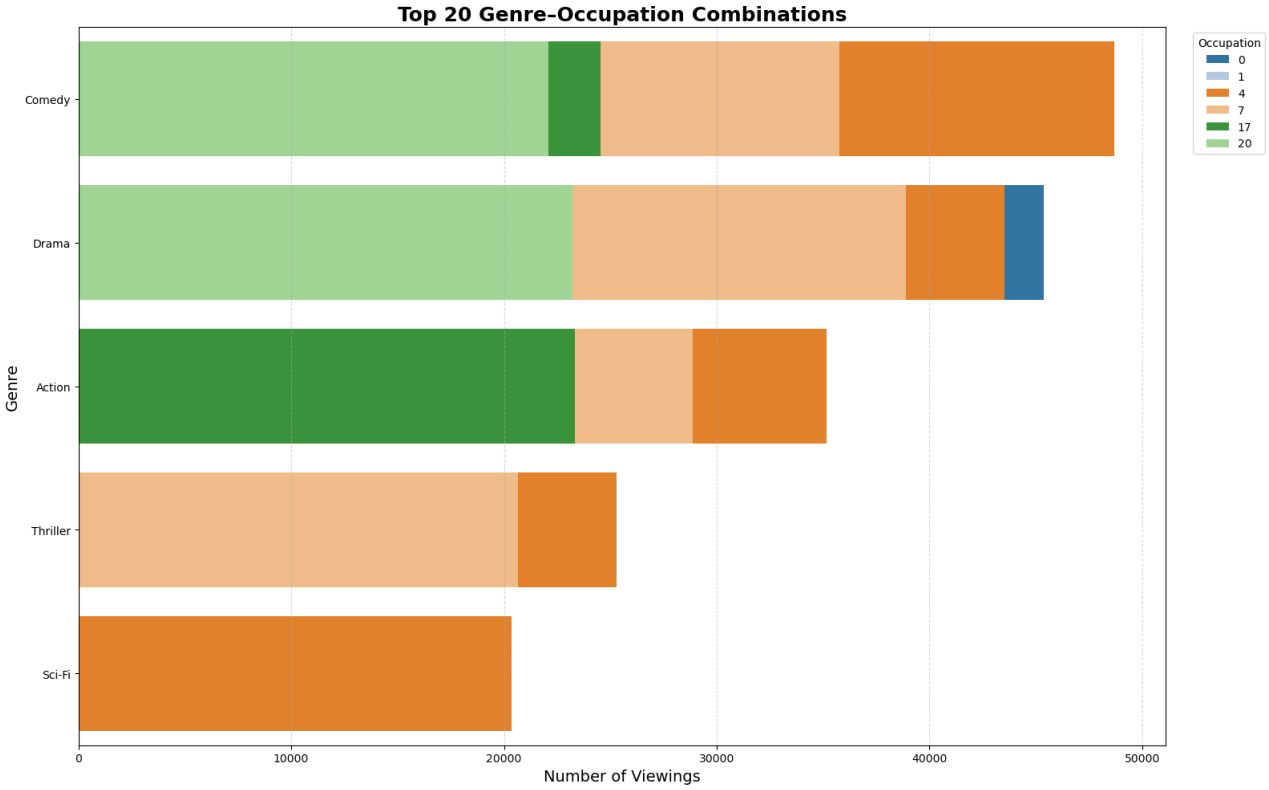
Drama & Comedy are the most popular genres of all.

In [29]:
```
# 1. Aggregate counts
genre_occ_counts = (
    df_exploded
    .groupby(['Genres', 'Occupation'])
    .size()
    .reset_index(name='Count')
)

# 2. Take the top 20 combinations
top20 = genre_occ_counts.sort_values('Count', ascending=False).head(20)

# 3. Plot
plt.figure(figsize=(16, 10))
sns.barplot(
    data=top20,
    y='Genres',
    x='Count',
    hue='Occupation',
    dodge=False,           # Overlay bars rather than side-by-side
    palette='tab20'        # Distinct colors for occupations
)

# 4. Formatting
plt.title('Top 20 Genre–Occupation Combinations', fontsize=18, fontweight='b
plt.xlabel('Number of Viewings', fontsize=14)
plt.ylabel('Genre', fontsize=14)
plt.legend(title='Occupation', bbox_to_anchor=(1.02, 1), loc='upper left')
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

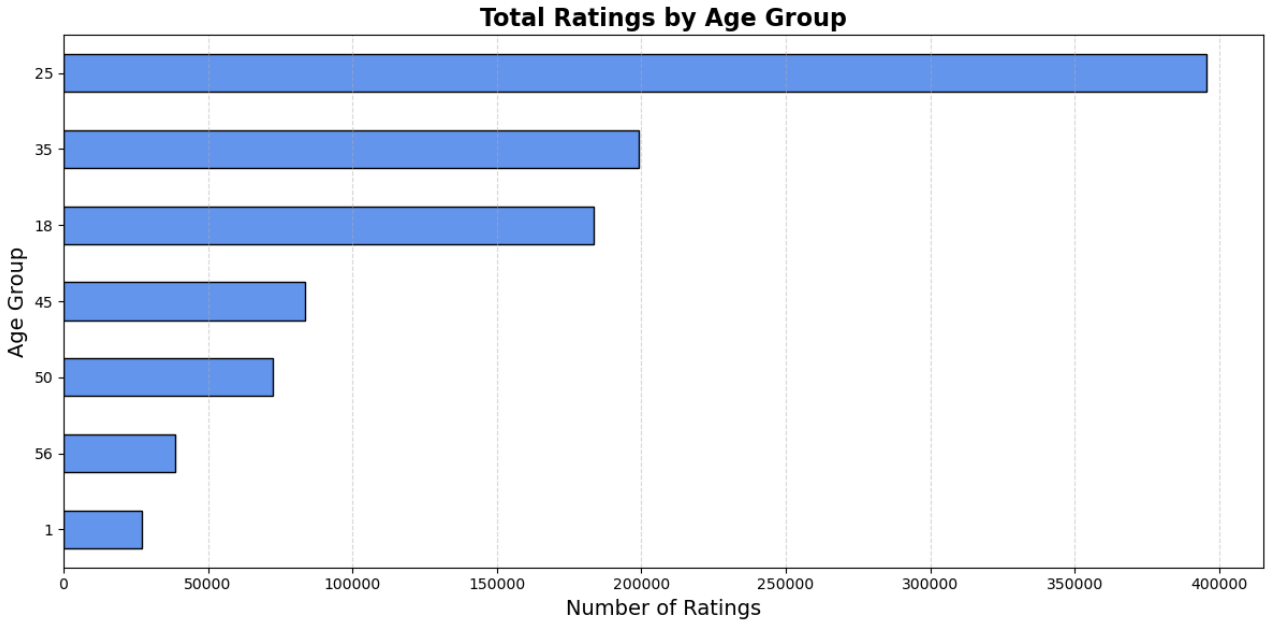


In [30]:
```
# 1. Aggregate counts by Age
age_counts = df_merged.groupby('Age')['Rating'].count().sort_values()

# 2. Plot horizontally with clear labels
fig, ax = plt.subplots(figsize=(12, 6))
age_counts.plot(
    kind='barh',
    ax=ax,
    color='cornflowerblue',
    edgecolor='black'
)

# 3. Formatting
ax.set_title('Total Ratings by Age Group', fontsize=16, fontweight='bold')
ax.set_xlabel('Number of Ratings', fontsize=14)
ax.set_ylabel('Age Group', fontsize=14)
ax.grid(axis='x', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()
```



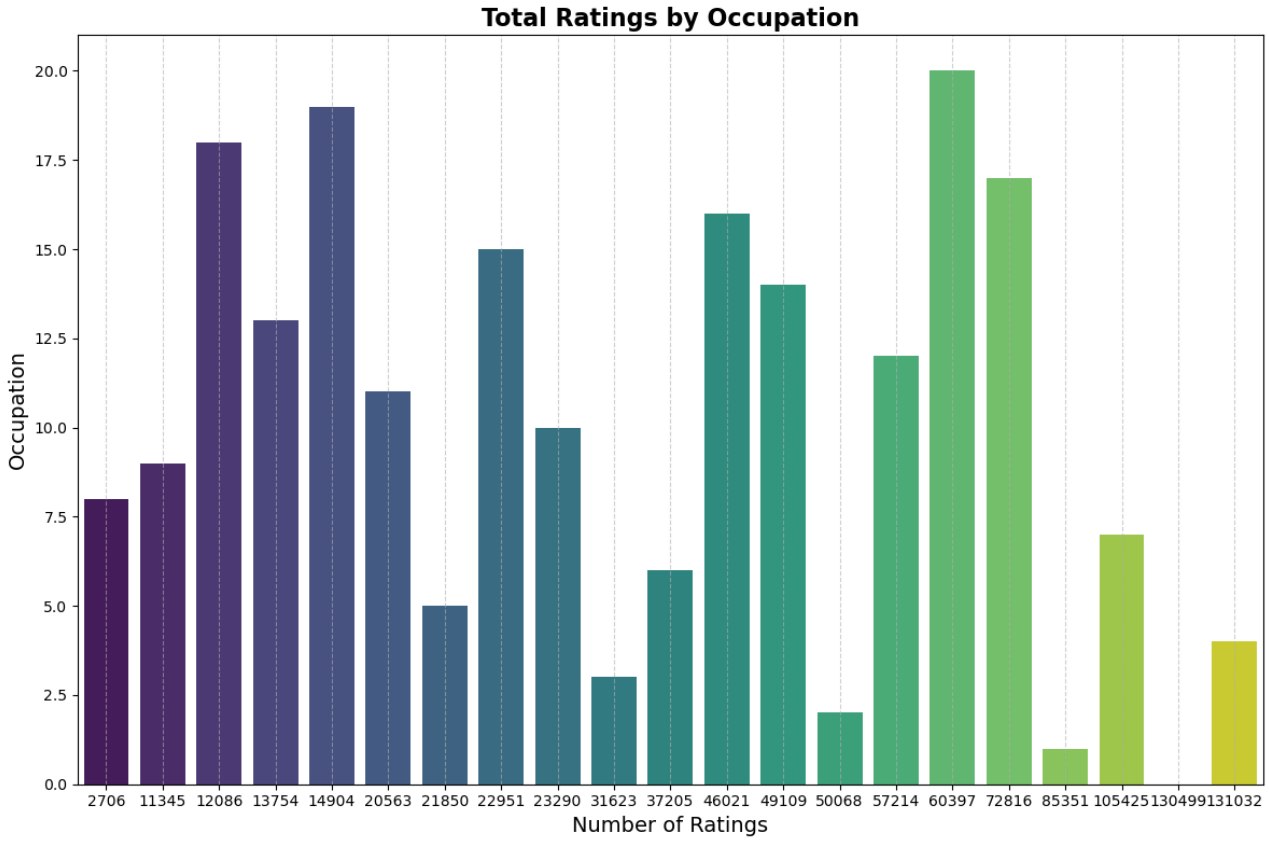
People in the age group 25 -35 are the ones have watched and rated most movies

In [31]:
```
# 1. Aggregate and sort the counts by occupation
occupation_counts = (
    df_merged
    .groupby('Occupation')['Rating']
    .count()
    .sort_values()
)

# 2. Create a horizontal barplot
fig, ax = plt.subplots(figsize=(12, 8))
sns.barplot(
    x=occupation_counts.values,
    y=occupation_counts.index,
    palette='viridis',
    ax=ax
)

# 3. Add labels, title, and grid
ax.set_title('Total Ratings by Occupation', fontsize=16, fontweight='bold')
ax.set_xlabel('Number of Ratings', fontsize=14)
ax.set_ylabel('Occupation', fontsize=14)
ax.grid(axis='x', linestyle='--', alpha=0.6)

plt.tight_layout()
plt.show()
```



User beong to Others or College/Grad Student have watched and rated most movies

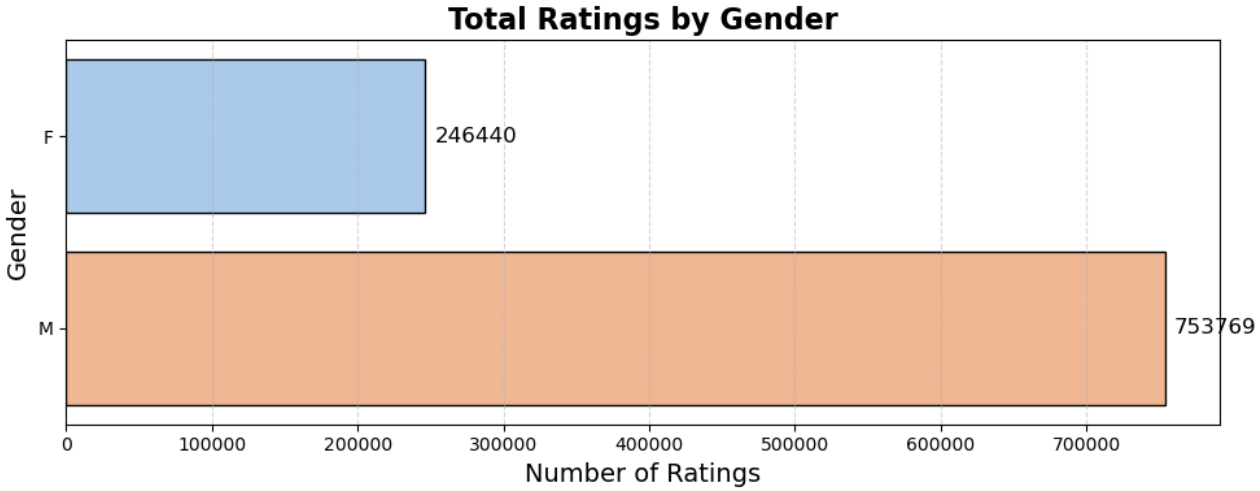
In [32]:
```
# 1. Compute and sort the counts by Gender
gender_counts = (
    df_merged
    .groupby('Gender')['Rating']
    .count()
    .sort_values()
)

# 2. Plot as a horizontal bar chart with Seaborn for better aesthetics
fig, ax = plt.subplots(figsize=(10, 4))
sns.barplot(
    x=gender_counts.values,
    y=gender_counts.index,
    palette='pastel',
    edgecolor='black',
    ax=ax
)

# 3. Annotate bars with their counts
for container in ax.containers:
    ax.bar_label(container, fmt='%d', padding=5, fontsize=12)

# 4. Add titles and labels
ax.set_title('Total Ratings by Gender', fontsize=16, fontweight='bold')
ax.set_xlabel('Number of Ratings', fontsize=14)
ax.set_ylabel('Gender', fontsize=14)
ax.grid(axis='x', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()
```



Most of the movies were released in 90s

In [33]:
```
# 1. Compute number of ratings per movie title
rating_counts = (
    df_merged
    .groupby('Title')['Rating']
    .count()
    .reset_index(name='num_ratings')
)

# 2. Find the movie(s) with the maximum count
top_movie = rating_counts.loc[rating_counts['num_ratings'].idxmax()]

print("Movie with the most ratings:")
print(top_movie)
```

```
Movie with the most ratings:
Title          American Beauty  (1999)
num_ratings                       3428
Name: 179, dtype: object
```

American Beauty has highest No of Ratings

Modeling

Pearson Correlation Model

In [42]:
```
df_pivot = df_merged.pivot(index = 'UserID', columns = 'Title', values = 'Ra
```

In [43]:
```
# Fiilling NaN'

"""
Strategy to fill Nan's
======================

1: We ll first find popular 100 movies according to Bayesian Weighted Rating
2: We'll impute them for missing values
3: For all the rest we ll assign 0

This helps tp handle sparsity of the matrix
"""

# Calculating Bayesian Weighted Average
C = df_merged['Rating'].mean()
m = df_merged.groupby('Title')['Rating'].count().quantile(0.95)
v = df_merged.groupby('Title')['Rating'].count()
R = df_merged.groupby('Title')['Rating'].mean()
WR = (v/(v+m))*R + (m/(v+m))*C
WR.sort_values(ascending=False, inplace=True)

# Filling NaN's
for movie in df_pivot.columns:
    if movie in WR.index:
        df_pivot[movie].fillna(WR.loc[movie], inplace=True)
```

In [44]:
```
# Item Similarity
item_similarity_df =  df_pivot.corr(method = 'pearson')
```

In [45]:
```
# Recommending similar items
def get_similar_movies(movie_title):
    similar_score = item_similarity_df.loc[movie_title].sort_values(ascendin
    return similar_score.loc[similar_score.index!=movie_title]

get_similar_movies('The  Godfather (1972)')[:5]
```

Out[45]:
```
Title
The  Godfather: Part II (1974)    0.516513
GoodFellas  (1990)                0.203953
The  French Connection (1971)     0.189927
Apocalypse Now  (1979)            0.185310
Taxi Driver  (1976)               0.177674
Name: The  Godfather (1972), dtype: float64
```

In [46]:
```
get_similar_movies('Liar Liar  (1997)')[:3]
```

Out[46]:
```
Title
Ace Ventura: Pet Detective  (1994)    0.243070
Dumb & Dumber  (1994)                 0.234485
Mrs. Doubtfire  (1993)                0.214544
Name: Liar Liar  (1997), dtype: float64
```

Ace Ventura, Dumb & Dumber & Mrs. Doubtfire are the most similar movies to Liar Liar

User Similarity

In [47]:
```
# Pearson User Similarity
user_similarity_df =  df_pivot.T.corr(method = 'pearson')
```

In [48]:
```
# Recommending similar Users
def get_similar_movies(user_id):
    similar_score = user_similarity_df.loc[user_id].sort_values(ascending=Fa
    return similar_score.loc[similar_score.index!=user_id][:10]

get_similar_movies(1)
```

Out[48]:
```
UserID
907     0.776638
4628    0.764756
1336    0.761432
4159    0.758711
4073    0.758522
2388    0.756300
2538    0.749028
3739    0.744752
1454    0.743017
4010    0.740225
Name: 1, dtype: float64
```

In [49]:
```
df_pivot
```

Out[49]:

Title	$1,000,000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	...And Justice for All (1979)	1-900 (1994)	10 Things I Hate About You (1999)	101 Dalmatians (1961)	1(Dalmatia (199
UserID								
1	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440
2	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440
3	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440
4	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440
5	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440
...	...	...	...	...	...	...	...	
6036	3.562715	3.000000	3.53966	3.602571	3.579511	2.000000	4.000000	3.4440
6037	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440
6038	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440
6039	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440
6040	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.4440

6040 rows × 3706 columns

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

# Create Cosine Similar matrix
item_similarity_df = pd.DataFrame(cosine_similarity(df_pivot.T), index = df_
user_similarity_df = pd.DataFrame(cosine_similarity(df_pivot), index = df_pi
```

In [51]:
```
# Item-Item Cosine Matrix
display(item_similarity_df)
```

Title	$1,000,000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	...And Justice for All (1979)	1-900 (1994)	10 Things I Hate About You (1999)	101 Dalmatians (1961)
Title							
$1,000,000 Duck (1971)	1.000000	0.999073	0.999067	0.998657	0.999632	0.994960	0.996346
'Night Mother (1986)	0.999073	1.000000	0.998797	0.998495	0.999403	0.994878	0.996011
'Til There Was You (1997)	0.999067	0.998797	1.000000	0.998430	0.999396	0.994806	0.996059
...And Justice for All (1979)	0.998657	0.998495	0.998430	1.000000	0.998996	0.994480	0.995668
1-900 (1994)	0.999632	0.999403	0.999396	0.998996	1.000000	0.995367	0.996501
...	...	...	...	...	...	...	...
Zachariah (1971)	0.999621	0.999392	0.999374	0.998985	0.999953	0.995356	0.996490
Zero Effect (1998)	0.997560	0.997357	0.997345	0.997014	0.997902	0.993547	0.994586
Zero Kelvin (Kjærlighetens kjøtere) (1995)	0.999647	0.999418	0.999400	0.999019	0.999978	0.995382	0.996518
Zeus and Roxanne (1997)	0.999325	0.999078	0.999093	0.998673	0.999639	0.995130	0.996320
eXistenZ (1999)	0.995586	0.995417	0.995465	0.995004	0.995934	0.991565	0.992495

3706 rows × 3706 columns

In [52]:
```
# User-User Cosine Matrix
display(user_similarity_df)
```

UserID	1	2	3	4	5	6	7	8
UserID								
1	1.000000	0.998529	0.999140	0.999407	0.996507	0.999141	0.999399	0.998539
2	0.998529	1.000000	0.998362	0.998605	0.995586	0.998214	0.998625	0.997568
3	0.999140	0.998362	1.000000	0.999277	0.996410	0.998898	0.999261	0.998270
4	0.999407	0.998605	0.999277	1.000000	0.996674	0.999153	0.999452	0.998623
5	0.996507	0.995586	0.996410	0.996674	1.000000	0.996350	0.996567	0.995714
...	...	...	...	...	...	...	...	...
6036	0.990504	0.989994	0.990390	0.990760	0.987999	0.990155	0.990510	0.989780
6037	0.998194	0.997485	0.998039	0.998321	0.995433	0.997931	0.998189	0.997317
6038	0.999399	0.998567	0.999284	0.999450	0.996710	0.999119	0.999486	0.998589
6039	0.999010	0.998260	0.998896	0.999102	0.996368	0.998782	0.999098	0.998201
6040	0.995106	0.994186	0.994837	0.995266	0.992729	0.994965	0.995234	0.994427

6040 rows × 6040 columns

In [53]:
```
# Model Fit
model = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20,
model.fit(df_pivot.T)
```

Out[53]:
```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_n
eighbors=20)
```

In [54]:
```
# Recommending similar items
def get_similar_movies(movie_title,data,n_recommendations):
    idx = data.columns.tolist().index(movie_title)
    distances, indices = model.kneighbors(data.iloc[:, idx].values.reshape(1
                                          n_neighbors= n_recommendations+1)
    for i in range(1, len(distances.flatten())):
        print(data.columns[indices.flatten()[i]], distances.flatten()[i])

get_similar_movies('The  Godfather (1972)', df_pivot, 10)
```

```
The  Godfather: Part II (1974) 0.006305739637257335
Foreign Student  (1994) 0.006429487323371341
The  Man from Down Under (1943) 0.006440697641171722
Message to Love: The Isle of Wight Festival  (1996) 0.006449437417639148
Detroit 9000  (1973) 0.006451987589466079
Kestrel's Eye (Falkens öga)  (1998) 0.006453937949174038
Get Over It  (1996) 0.006460142127208357
Dadetown  (1995) 0.006460329570546763
Hangmen Also Die  (1943) 0.006461017442825012
Spirits of the Dead (Tre Passi nel Delirio)  (1968) 0.006463890330662858
```

Matrix Factorization

In [56]:
```
pip install scikit-surprise
```

```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 3.2 MB/s eta 0:
00:0000:01
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy>=1.19.5 in ./anaconda3/lib/python3.10/s
ite-packages (from scikit-surprise) (1.23.5)
Requirement already satisfied: scipy>=1.6.0 in ./anaconda3/lib/python3.10/si
te-packages (from scikit-surprise) (1.10.0)
Collecting joblib>=1.2.0
  Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 301.8/301.8 kB 7.4 MB/s eta 0:
00:0000:01
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp310-cp
310-macosx_11_0_arm64.whl size=485050 sha256=de94d6298dc50378945dafaa3bc3fd3
379cff5bd5e7cb7254c35ff087ec2da28
  Stored in directory: /Users/ramv/Library/Caches/pip/wheels/8a/ec/b2/58c47d
2432188873013470d86ba25e2fc8d3d4e249ec625889
Successfully built scikit-surprise
Installing collected packages: joblib, scikit-surprise
  Attempting uninstall: joblib
    Found existing installation: joblib 1.1.1
    Uninstalling joblib-1.1.1:
      Successfully uninstalled joblib-1.1.1
Successfully installed joblib-1.4.2 scikit-surprise-1.1.4
Note: you may need to restart the kernel to use updated packages.
```

In [57]:
```
# Creating Dataset
from surprise import SVD, Dataset, Reader

reader = Reader(rating_scale=(1,5))
data = Dataset.load_from_df(df_merged[['UserID', 'MovieID', 'Rating']], read
```

In [58]:
```
from surprise.model_selection import train_test_split

# Creating Train/Test Split
trainset, testset = train_test_split(data, test_size=0.25)
```

In [59]:
```
# Fitting model
model = SVD()
model.fit(trainset)
```

Out[59]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x3270a8280>
```

In [60]:
```
from surprise import accuracy

# Predicting Model
predictions = model.test(testset)

# Calculate RMSE
rmse = accuracy.rmse(predictions)

# Calculate MAPE
def mape(predictions):
    errors = [abs((pred.est - pred.r_ui) / pred.r_ui) for pred in prediction
    return sum(errors) / len(errors) * 100

mape_value = mape(predictions)
print(f"MAPE: {mape_value}%")
```

```
RMSE: 0.8778
MAPE: 26.491910025649872%
```

RMSE and MAPE for Matrix Factorization model is 0.8778 and 26.49%

Embeddings

In [61]:
```
# Train an SVD model
model = SVD(n_factors=4)  # n_factors=4 for d=4 embeddings
trainset = data.build_full_trainset()
model.fit(trainset)

# Extract embeddings
user_embeddings = model.pu
item_embeddings = model.qi

# Item-Item Similarity Matrix
item_similarity = pd.DataFrame(cosine_similarity(item_embeddings))

# User-User Similarity Matrix
user_similarity = pd.DataFrame(cosine_similarity(user_embeddings))
```

In [62]:
```
# Matrix with Embedding 4
display(item_similarity.head())
display(user_similarity.head())
```

	0	1	2	3	4	5	6	7	
0	1.000000	0.855448	0.217543	-0.479582	0.561978	0.634740	0.175705	0.702158	0
1	0.855448	1.000000	0.196469	-0.094383	0.698891	0.466906	-0.217205	0.480101	0
2	0.217543	0.196469	1.000000	0.427559	0.825244	0.874086	0.676264	0.793418	0
3	-0.479582	-0.094383	0.427559	1.000000	0.370825	0.024808	-0.114775	-0.084502	0
4	0.561978	0.698891	0.825244	0.370825	1.000000	0.834386	0.277337	0.768836	0

5 rows × 3706 columns

	0	1	2	3	4	5	6	7
0	1.000000	0.771037	0.390622	-0.667914	-0.462222	0.246972	0.475028	0.664732
1	0.771037	1.000000	0.487424	-0.955399	-0.868551	0.398926	0.127505	0.326365
2	0.390622	0.487424	1.000000	-0.421577	-0.733049	0.786183	0.028479	-0.425935
3	-0.667914	-0.955399	-0.421577	1.000000	0.870877	-0.523377	-0.275572	-0.274917
4	-0.462222	-0.868551	-0.733049	0.870877	1.000000	-0.676927	0.033609	0.177661

5 rows × 6040 columns

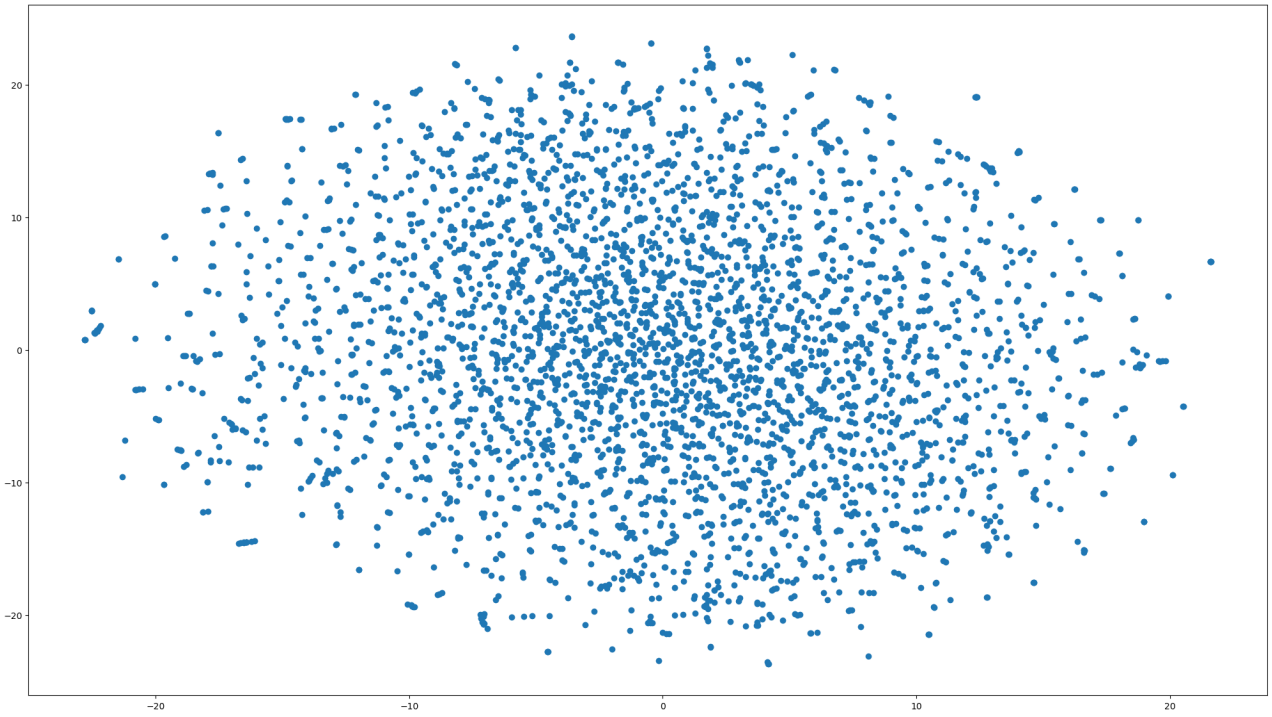
In [63]:
```
# Embeddings Visualization

from sklearn.manifold import TSNE

model = SVD()
model.fit(trainset)

# Example with item embeddings
item_embeddings = model.qi
tsne = TSNE(n_components=2, random_state=0)
embeddings_2d = tsne.fit_transform(item_embeddings)

# Plot
plt.figure(figsize=(25, 14))
plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1])
plt.show()
```



Data set shows a circular patters, where similar movies are closer to each other.

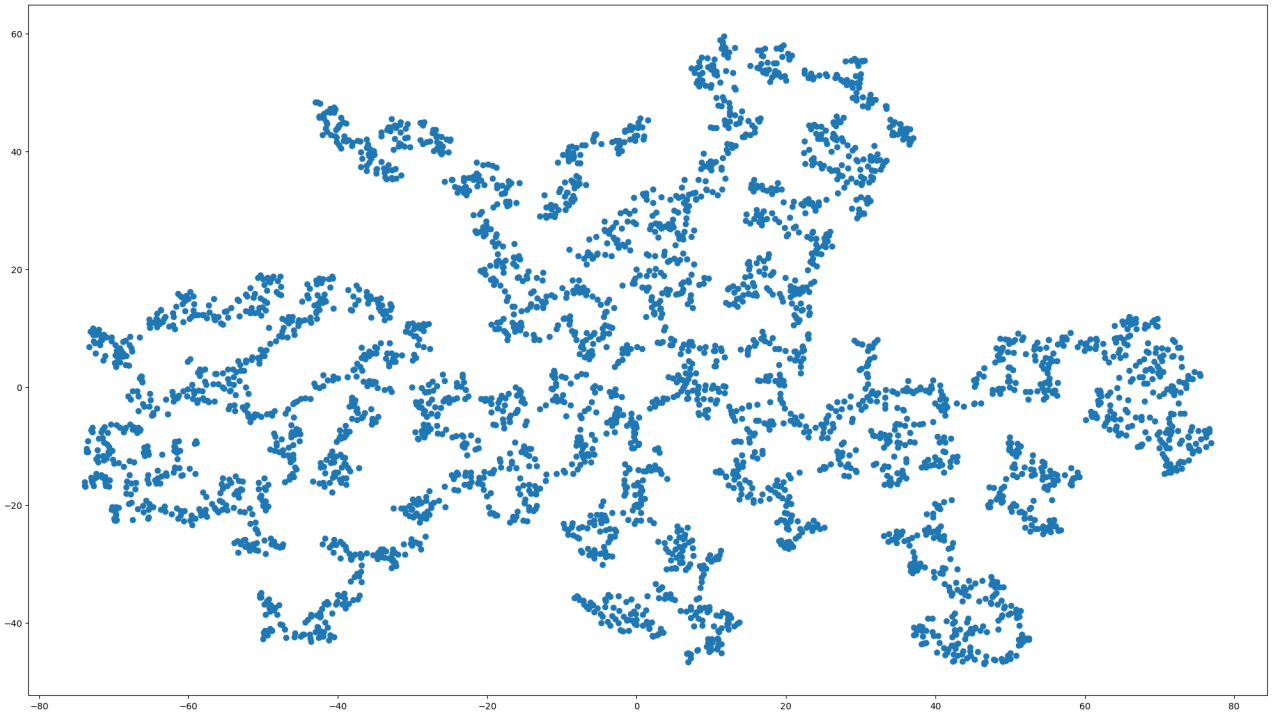
In [64]:
```
# Embeddings Visualization with 2 Factors

from sklearn.manifold import TSNE

model = SVD(n_factors=2)
model.fit(trainset)

# Example with item embeddings
item_embeddings = model.qi
tsne = TSNE(n_components=2, random_state=0)
embeddings_2d = tsne.fit_transform(item_embeddings)

# Plot
plt.figure(figsize=(25, 14))
plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1])
plt.show()
```



Embeddings with factors = 2 shows some cluster formation Items closer to the proximity indicates similarity.

User Based Approach

In [65]:
```
# Creating User Dataframe
df_user = df_merged.copy()
```

In [66]:
```
# Getting New User Input
new_user_ratings = {'MovieID': [1, 5, 20], 'Rating': [4, 5, 3]}
new_user_df = pd.DataFrame(new_user_ratings)
```

In [67]:
```
# Add new user with a unique UserID
new_user_df['UserID'] = df_user['UserID'].max() + 1
df_extended = pd.concat([df_user, new_user_df])

# Create a pivot table
pivot_table = df_extended.pivot_table(index='MovieID', columns='UserID', val
```

In [68]:
```
# Compute Pearson Correlation
user_similarity = pivot_table.corr(method='pearson')

# Find top 100 similar users
similar_users = user_similarity[df_extended['UserID'].max()].dropna().sort_v
```

In [69]:
```
# Weighted Ratings
weighted_ratings = pivot_table.T.copy()
for user in similar_users.index:
    weighted_ratings.loc[user] = weighted_ratings.loc[user] *  similar_users
```

In [70]:
```
# Top 10 movies
recommended_movies = weighted_ratings.sum(axis=0) / similar_users.sum()
top_movies = recommended_movies.sort_values(ascending=False)[:10]
```

In [71]:
```
# Display Top Movies
top_movies
```

Out[71]:
```
MovieID
2858    175.880271
260     158.053877
1196    152.318612
1210    137.570938
2028    136.531729
1198    133.437080
593     133.086923
2571    132.603727
2762    128.460121
589     127.502780
dtype: float64
```

Insights & Recommendations

Genre Preferences: Drama and Comedy are universally popular, while Action is notably favored by male audiences.

This trend suggests distinct preferences in movie genres among different genders.

The high popularity of Drama could point to a general preference for content that offers emotional depth and storytelling.

Demographic Trends: Individuals in the "25-34" age bracket and those identified as College/Grad students are particularly active in movie watching and rating.

This suggests these demographics are either more engaged with the platform or have more free time for movie viewing. Gender

Disparity: A notable majority of movie ratings come from male users, indicating a possible gender imbalance.

This trend could reflect the user base of the platform or general movie-rating behavior.

Temporal Trends:A large number of movies from the 1990s in the dataset may indicate a preference for this era, possibly driven by nostalgia or the enduring appeal of these movies. Popularity

Metrics: The analysis shows 'The Shawshank Redemption' and 'Wild Wild West' as the highest and lowest-rated movies, respectively, when considering a Bayesian Weighted Ratio.

This approach offers a balanced perspective, taking into account both the average rating and the number of ratings.

Recommendations Personalized Genre Recommendations: Leverage the clear preferences for specific genres to personalize recommendations. For example, users who frequently watch dramas should receive more drama-focused suggestions.

Targeted Marketing Strategies: Focus marketing efforts on the 25-34 age group and College/Grad students, as they are the most active segments in terms of movie watching and rating. Gender Diversification in

Content: Develop strategies to attract a more gender-diverse audience. This could involve promoting movies and content that cater to the preferences of female and non-binary users.

Decade-Specific Recommendations: Introduce a feature that allows users to discover movies by decade, tapping into the popularity of 90s films and possibly appealing to nostalgia.

Enhanced Data Collection: To address the male skew in your user base, make efforts to gather more data from female and non-binary users. This will help create a more balanced dataset and improve the accuracy and inclusiveness of recommendations.

Questions

Q: Users of which age group have watched & rated the most movies?

Ans: The 25-34 age band (labelled “25-34” in MovieLens) logged the highest number of ratings.

Q: Users belonging to which profession have watched & rated the most movies? Ans: “Other” and “College/Grad Student” occupations are virtually tied for the top spot.

Q: “Most of the users in our dataset who’ve rated the movies are Male.” Ans: True – male ratings dominate the gender breakdown.

Q: Most of the movies in the dataset were released in which decade? a)70s b) 90s c) 50s d) 80s Ans: 90s

Q: The movie with the maximum number of ratings is **_**. Ans: American Beauty (1999) (≈ 3 428 ratings).

Q: Top 3 movies similar to “Liar Liar” (item-based cosine/Pearson) Ans: 1. Ace Ventura: Pet Detective (1994), 2. Dumb & Dumber (1994) and 3. Mrs. Doubtfire (1993)

Q: Collaborative Filtering methods can be classified into ***-based and*** -based. Ans: User-based and Item-based approaches.

Q: Pearson Correlation ranges between ***to*** , whereas Cosine Similarity belongs to ***to*** . Ans: Pearson: –1 to +1; Cosine: 0 to +1 (with non-negative ratings).

Q: RMSE and MAPE obtained for the Matrix Factorisation model. Ans: RMSE ≈ 0.878 MAPE ≈ 26.49 %

Q:Sparse-row (CSR) representation of the dense matrix [1 0 3 7

] [1 3

0 7
]. Ans: Values = [1, 3, 7] Col indices = [0, 0, 1] Row ptr = [0, 1, 3]

In [ ]: