Recommender Systems - Movies

Movie Recommender Systems

** Objective:** To enhance user experience on a movie streaming platform by developing a personalized recommender system. This system will leverage user ratings and similarities among users to provide tailored movie recommendations.

Project Description:

The challenge is structured into distinct phases, each contributing towards building an efficient and effective recommender system:

- ** Data Preparation and Integration:**
  - **Source Data**: Ratings, Users, and Movies datasets are provided.
  - **Goal**: Format and merge these datasets into a single comprehensive dataframe for analysis.
- ** Exploratory Data Analysis (EDA) and Data Cleaning:**
  - **Focus**: Examine dataset structure, clean data, and perform feature engineering, including type conversions and deriving new attributes like 'Release Year'.
- ** Building the Recommender System:**
  - **Approaches**:
    1. **Item-based Collaborative Filtering**: Using Pearson Correlation to recommend movies.
    2. **User-based Collaborative Filtering** (Optional): Utilizing user ratings to find similar users and recommend movies.
    3. **Cosine Similarity**: Implementing a Nearest Neighbors algorithm to identify similar movies.
    4. **Matrix Factorization**: Using libraries like cmfrec/Surprise for advanced recommendations.
- ** Model Evaluation and Optimization:**
  - **Metrics**: RMSE (Root Mean Squared Error) and MAPE (Mean Absolute Percentage Error).
  - **Visualization**: Embedding techniques for item-item and user-user similarity analysis.

- ** Advanced Techniques (Bonus):**
    - Train-test split strategies for Matrix Factorization.
    - Embedding-based similarity functions.

In [2]:
```
#!pip install surprise
```

In [3]:
```
# Initializing necessary libraries
import numpy as np
import pandas as pd

# Handling warnings
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
# Initializinf DF's
df_ratings = pd.read_csv("zee-ratings.dat", delimiter="::")
df_users = pd.read_csv('zee-users.dat', delimiter = '::')
df_movies = pd.read_csv("zee-movies.dat", delimiter = '::',encoding='ISO-8859-1')
```

In [5]:
```
# Dastaset shape
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 [6]:
```
# 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 [7]:
```python
# 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 [8]:
```python
# 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
```

Data Preparation

In [9]:
```python
# Renaming Columns to maintain consistency
```

```
df_movies.rename(columns = {'Movie ID':'MovieID'}, inplace = True)
```

In [10]:
```
df_merged = pd.merge(left = df_ratings, right = df_movies, on = "MovieID", how = "left")
df_merged = pd.merge(left = df_merged, right = df_users, on = 'UserID', how = 'left')
df_merged.head()
```

Out[10]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama	F	1	10	48067
1	1	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical	F	1	10	48067
2	1	914	3	978301968	My Fair Lady (1964)	Musical|Romance	F	1	10	48067
3	1	3408	4	978300275	Erin Brockovich (2000)	Drama	F	1	10	48067
4	1	2355	5	978824291	Bug's Life, A (1998)	Animation|Children's|Comedy	F	1	10	48067

In [11]:
```
# 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 [12]:
```
# 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
```

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

In [13]:
```python
# Handling Timestamp
df_merged['Timestamp'] = pd.to_datetime(df_merged['Timestamp'], unit = 's')
```

In [14]:
```python
# 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 [15]:
```python
# Dataset Shape
df_merged.shape
```

Out[15]:
```
(1000209, 13)
```

In [16]:
```python
# Dataset Nan's
df_merged.isna().sum()
```

Out[16]:
```
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 [17]:
```python
# DataTypes
df_merged.dtypes
```

Out[17]:
```
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 [18]:
```python
# Handling Zipcodes
df_merged[~df_merged['Zip-code'].str.isnumeric()].head()
```

Out[18]:

	UserID	MovieID	Rating	Title	Genres	Gender	Age	Occupation	Zip-code	Year	RatingDate	RatingMonth	Ra
21847	161	2988	4	Melvin and Howard (1980)	Drama	M	45	16	98107-2117	1980	19	12	
21848	161	1179	4	The Grifters (1990)	Crime|Drama|Film-Noir	M	45	16	98107-2117	1990	19	12	
21849	161	3860	4	The Opportunists (1999)	Crime	M	45	16	98107-2117	1999	19	12	
21850	161	1252	5	Chinatown (1974)	Film-Noir|Mystery|Thriller	M	45	16	98107-2117	1974	19	12	
21851	161	1253	5	The Day the Earth Stood Still (1951)	Drama|Sci-Fi	M	45	16	98107-2117	1951	19	12	

These Zip Codes follow US Standards where digits after - describe exact locations, Let's handle this by imputing them to traditional

In [19]:

```
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 [20]:

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

```
In [21]: # 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))
```

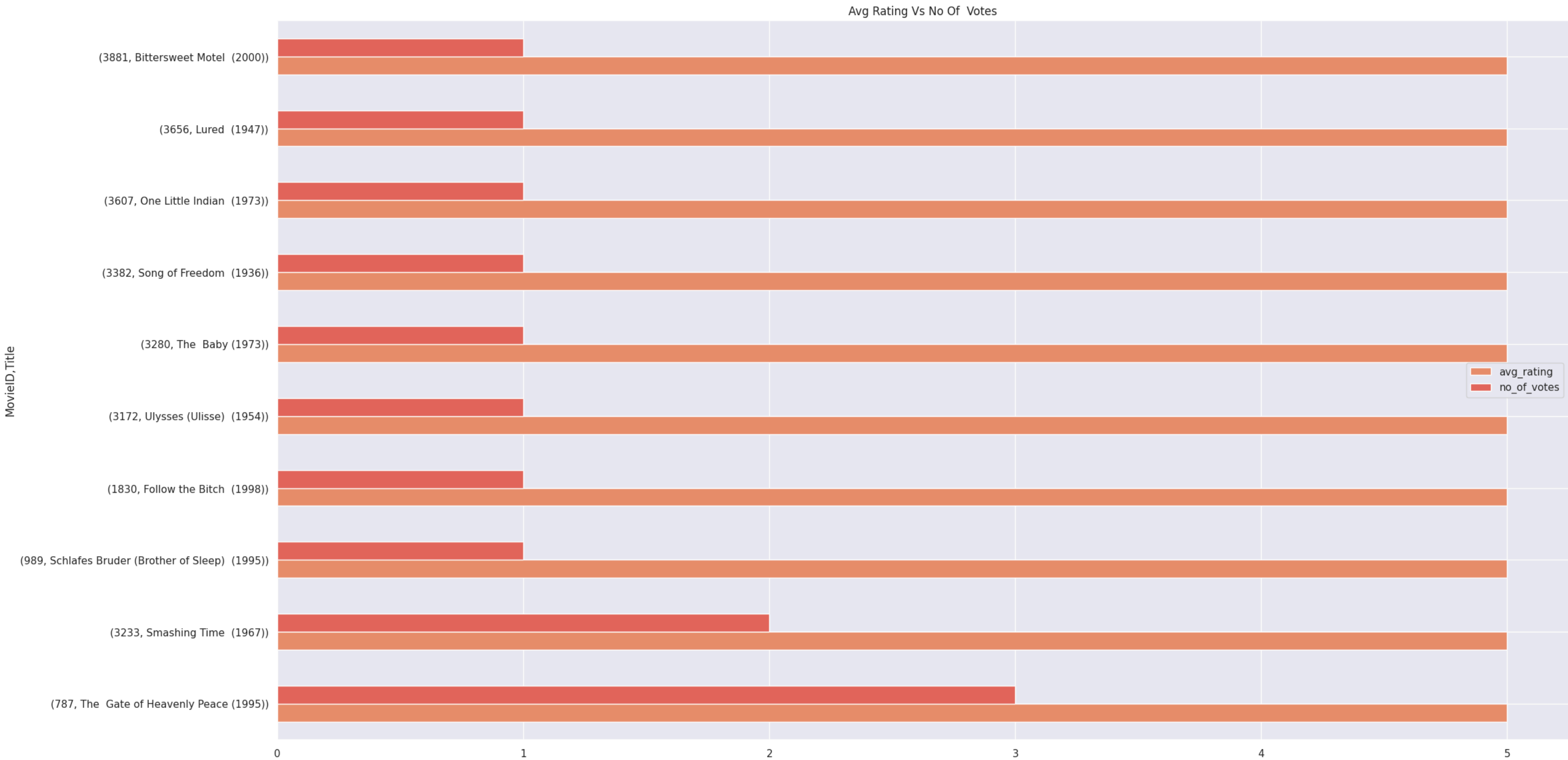
We have not removed year from Movie Title, because many movies in the dataset are observed with same name, Without year it is difficult to analyze

Data Visualization

```
In [22]: # Importing Necessary Libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Setting Parametes
sns.set(style = "darkgrid", palette = "flare")
```

```
In [23]: # Grouping data on Number of Votes and Average Rating
df_grouped = df_merged.groupby(['MovieID', 'Title'])['Rating'].agg(avg_rating = 'mean', no_of_votes = 'count')
df_grouped.sort_values(by = ['avg_rating', 'no_of_votes'], ascending = [False, False])[:10].plot(kind='barh',
                        title = "Avg Rating Vs No Of  Votes", figsize = (25,14))
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

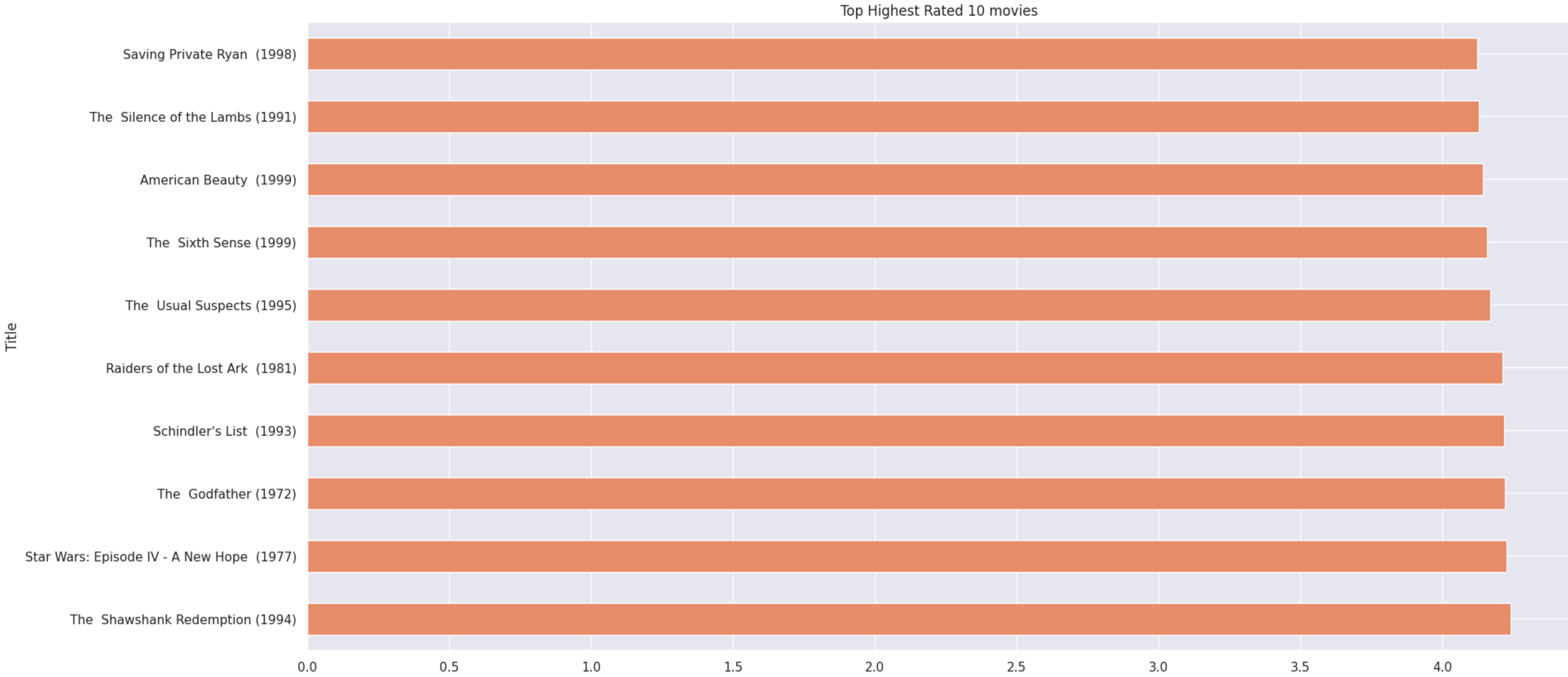
$$\text{Weighted Rating} = \left(\frac{v}{v+m}\cdot\right)R + \left(\frac{m}{v+m}\right)\cdot C$$

- 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 [24]:
```
# Top Highest Rated movies According to ratio of number of votes
C = df_merged['Rating'].mean()
```

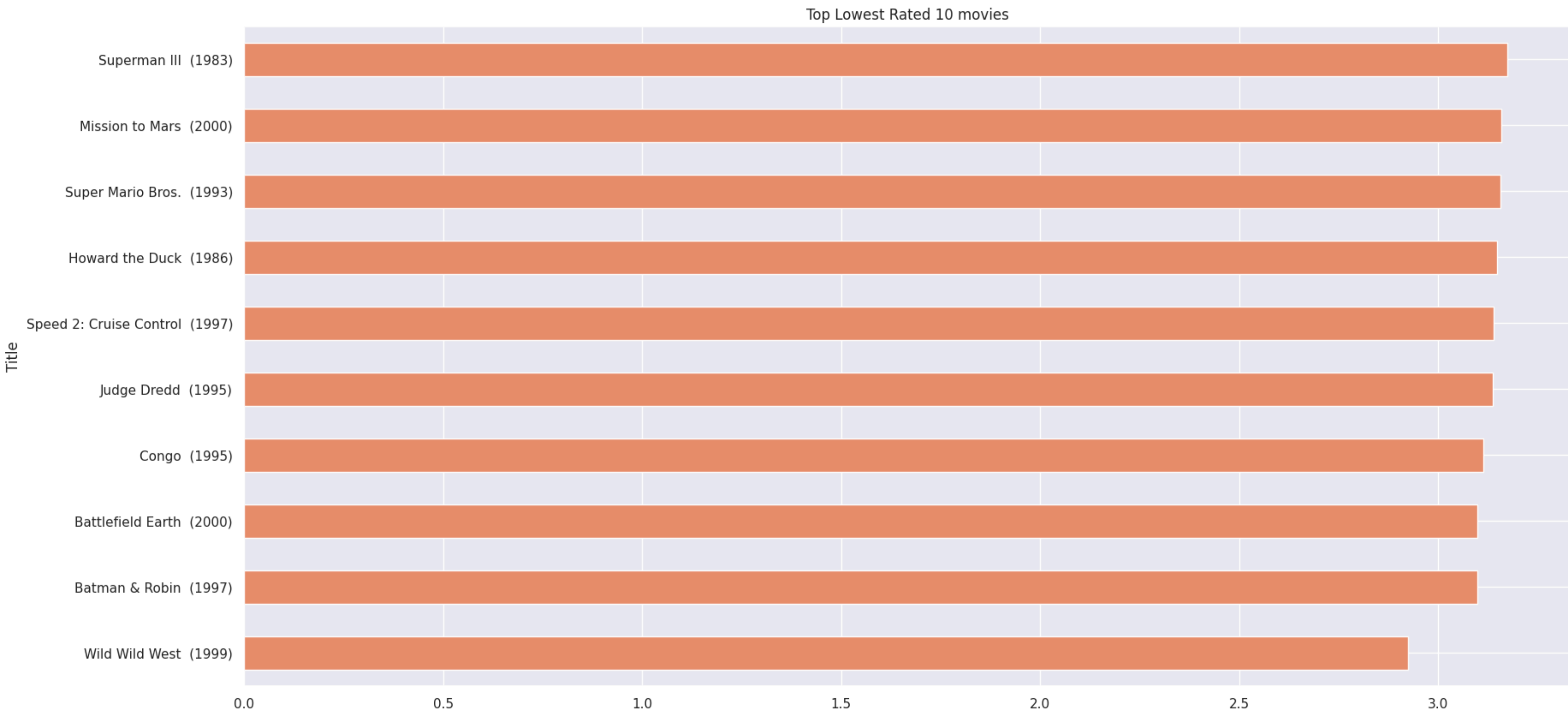
```python
m = df_merged.groupby('Title')['Rating'].count().quantile(0.95)
v = df_merged.groupby('Title')['Rating'].count()
R = df_merged.groupby('Title')['Rating'].mean()
WR = (v/(v+m))*R + (m/(v+m))*C

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



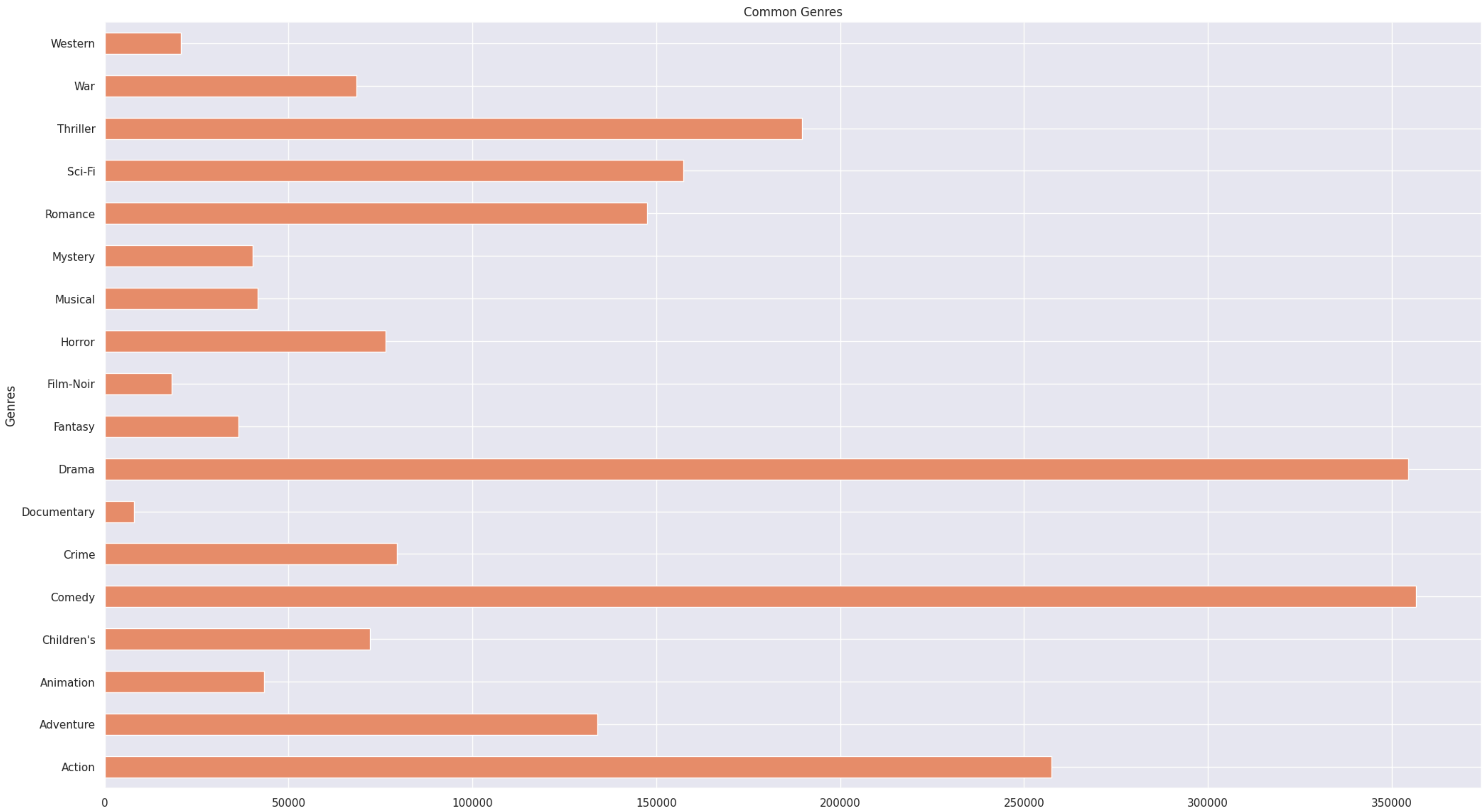
- Top Rated Movies according Bayesian Weighted ratio: The Shawshank Redemption

In [25]:
```python
# 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()
```



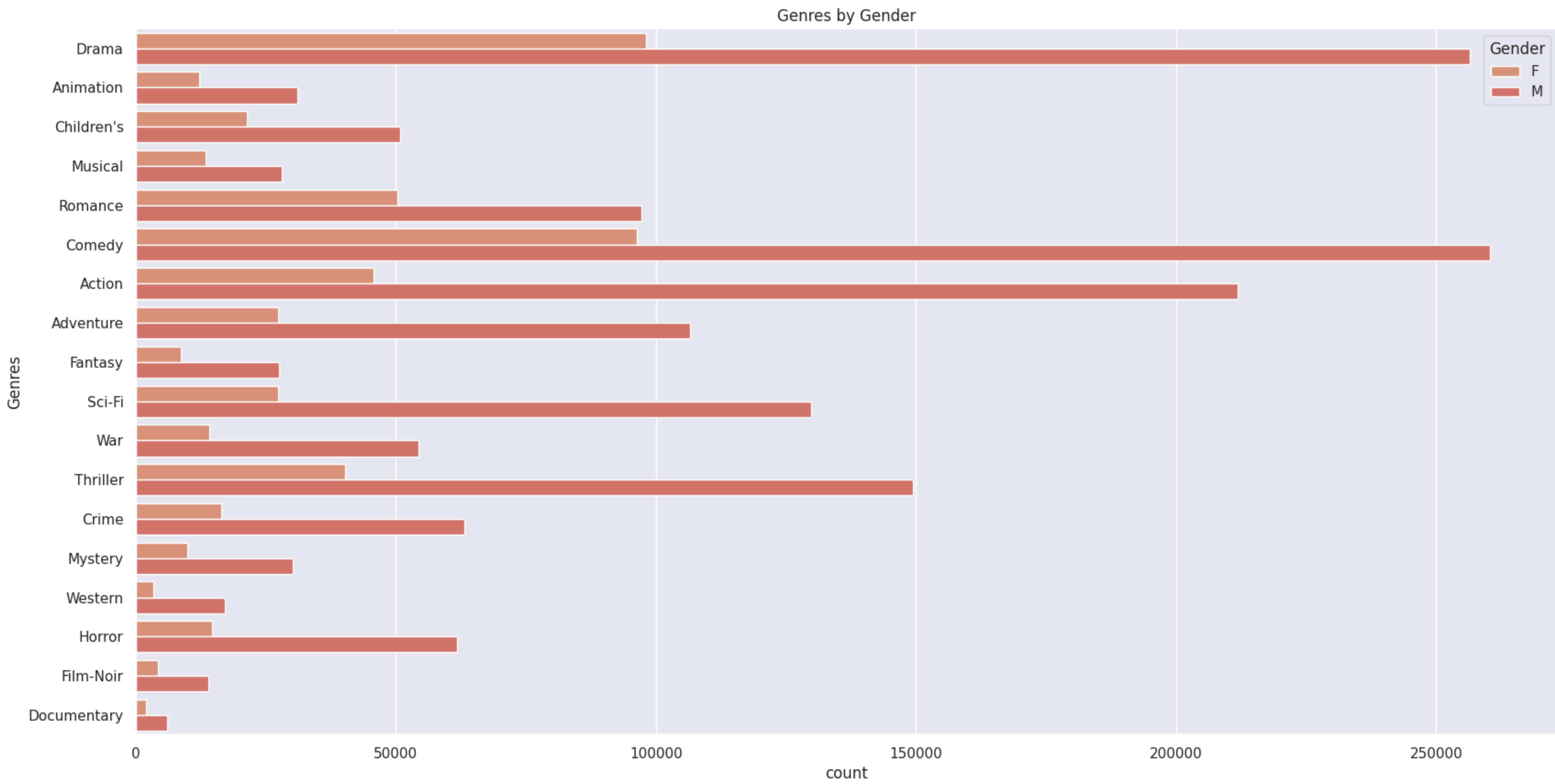
- Lowest Rated movie according to Bayesian Weighted Ratio: Wild Wild West

In [26]:
```
# Common Genres Movies
df_exploded.groupby('Genres').size().plot(kind = 'barh', figsize=(25,14), title='Common Genres')
plt.show()
```



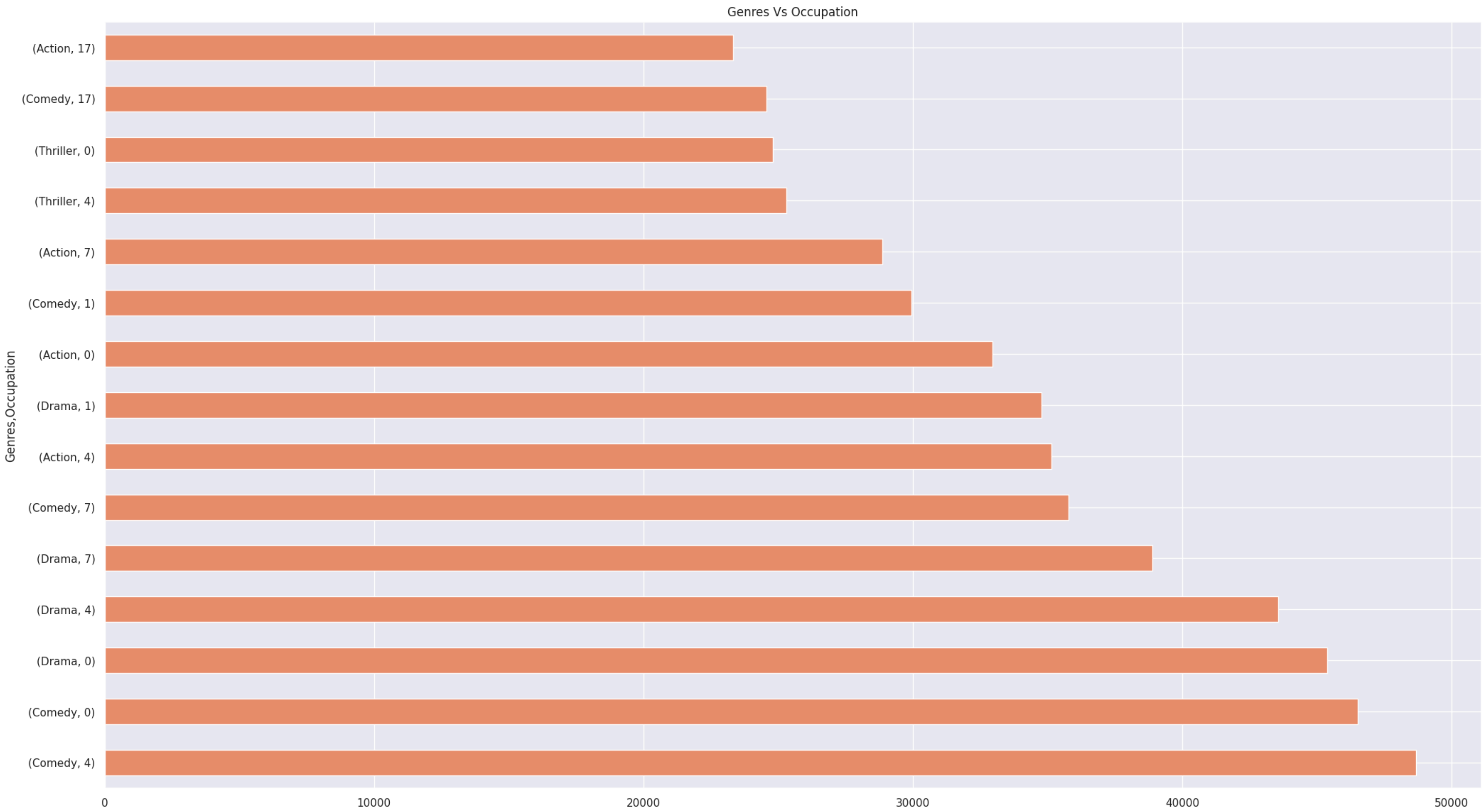
- Drama is most common genre

In [27]:
```
# 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()
```



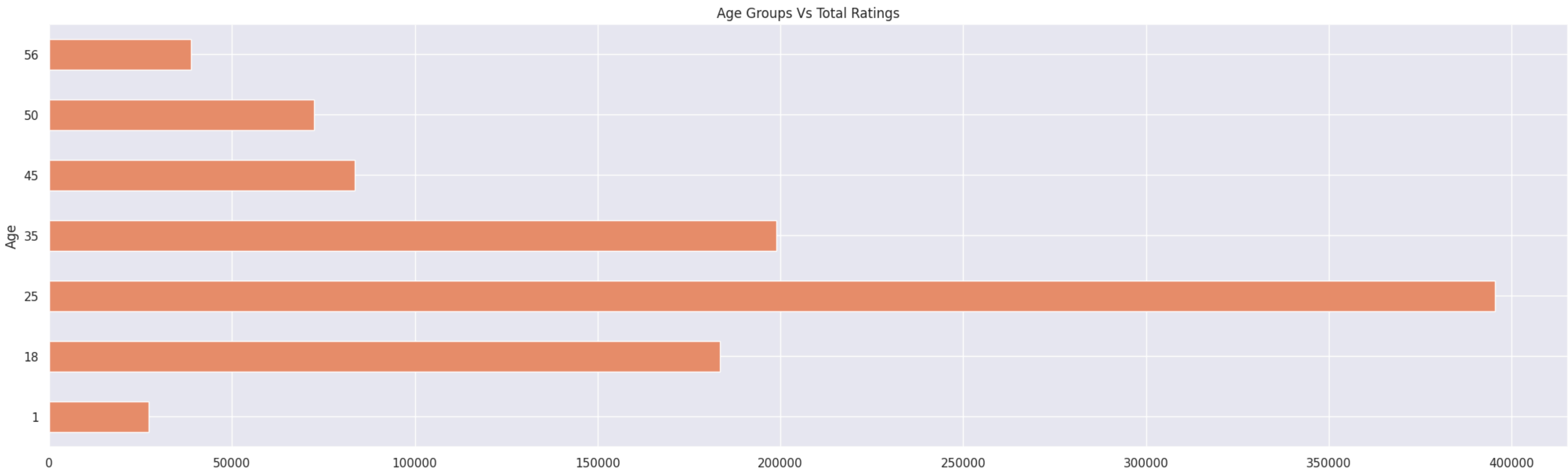
- Drama and Comedy is most popular genre among both gendres
- While Action is highly preferred by Males

In [28]:
```
# Top 15 Genres Watched according to profesison
df_exploded.groupby(['Genres', 'Occupation']).size().sort_values(ascending=False)[:15
                ].plot(kind = 'barh', figsize=(25,14), title='Genres Vs Occupation')
plt.show()
```



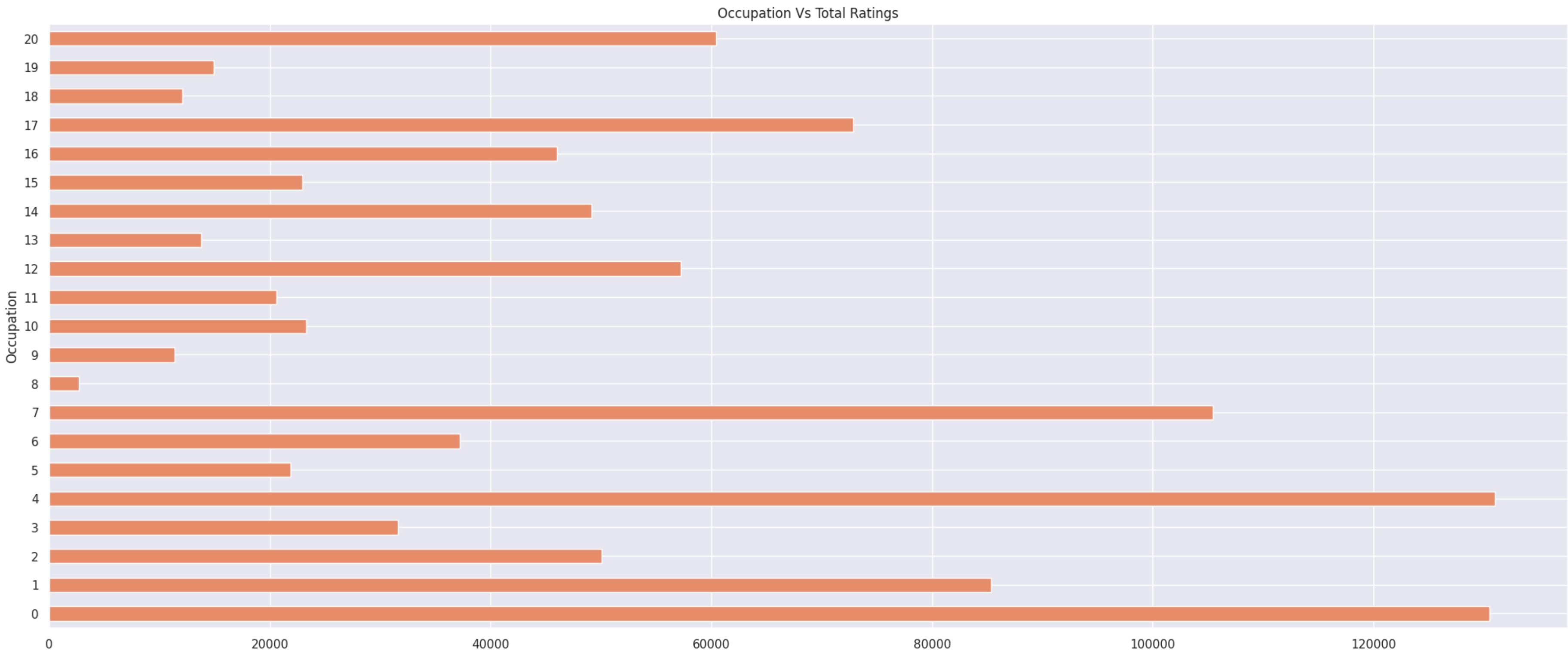
- Most viewers are College Grad student watching Comedy

In [29]:
```
df_merged.groupby("Age")['Rating'].count().plot(kind='barh',figsize=(25,7), title='Age Groups Vs Total Ratings')
plt.show()
```



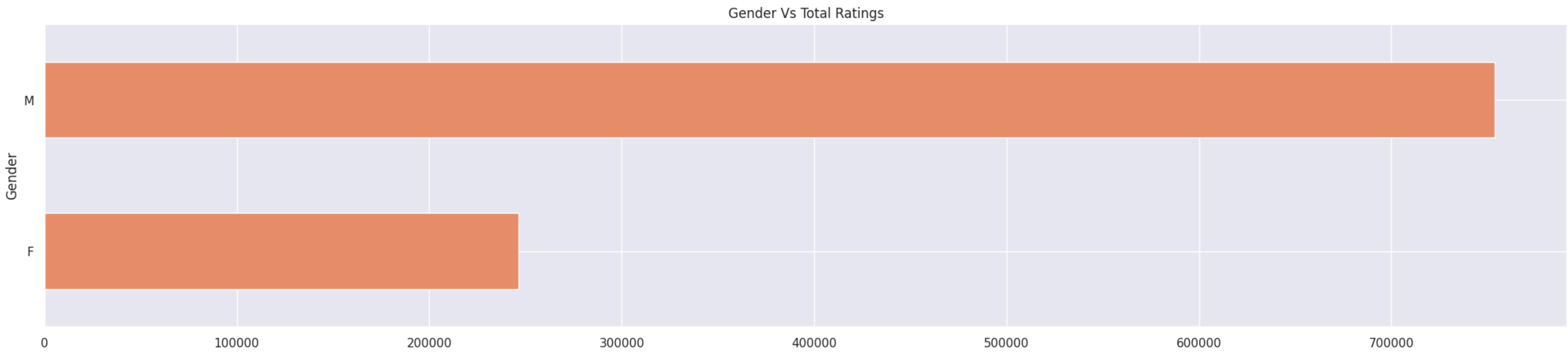
- Age group of "25-34" have watched and rated most movies

In [30]:
```
df_merged.groupby("Occupation")['Rating'].count().plot(kind='barh',figsize=(25,10), title='Occupation Vs Total Ratings')
plt.show()
```



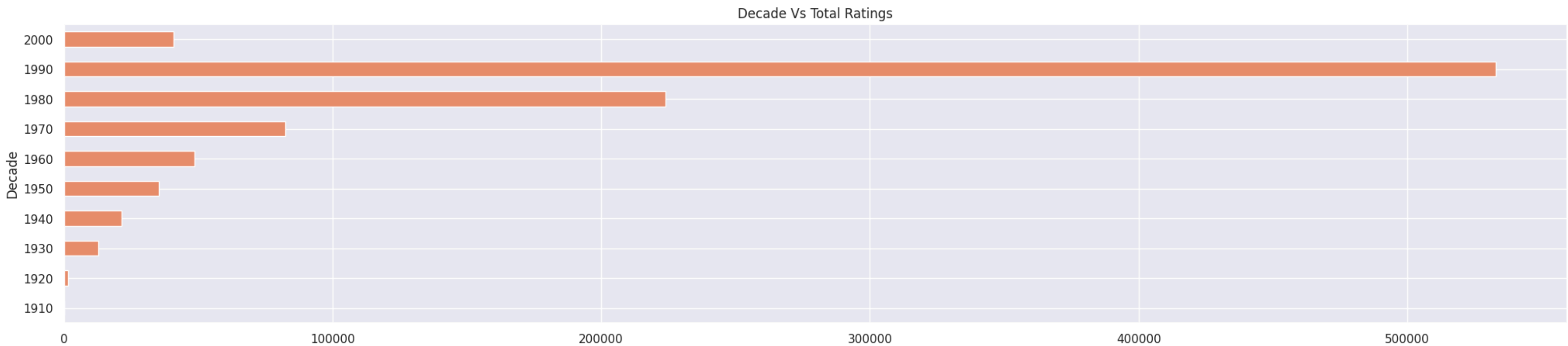
- User beong to Others or College/Grad Student have watched and rated most movies

In [31]:
```
df_merged.groupby("Gender")['Rating'].count().plot(kind='barh',figsize=(25,5), title='Gender Vs Total Ratings')
plt.show()
```



- Most of the movies are watched and rated by males

In [32]:
```
df_decade = df_merged.copy()
df_decade['Decade'] = (df_decade['Year']//10)*10
df_decade.groupby("Decade")['Rating'].count().plot(kind='barh',figsize=(25,5), title='Decade Vs Total Ratings')
plt.show()
```



- Most of the movies were released in 90's

In [33]:
```
# Maximum No of Ratings
df_merged.groupby('Title')['Rating'].count().sort_values(ascending=False)[:1]
```

Out[33]:
```
Title
American Beauty  (1999)    3428
Name: Rating, dtype: int64
```

- American Beauty has maximum No of Ratings

Modeling

Pearson Correlation Model

In [34]:
```
df_pivot = df_merged.pivot(index = 'UserID', columns = 'Title', values = 'Rating')
```

In [35]:
```
# Fillling NaN'

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

1: We ll first find popular 100 movies according to Bayesian Weighted Rating( on a scale of 5)
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 [36]:
```
# Item Similarity
item_similarity_df =  df_pivot.corr(method = 'pearson')
```

In [37]:
```
# Recommending similar items
def get_similar_movies(movie_title):
    similar_score = item_similarity_df.loc[movie_title].sort_values(ascending=False)
    return similar_score.loc[similar_score.index!=movie_title]

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

Out[37]:
```
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 [38]:
```
get_similar_movies('Liar Liar  (1997)')[:3]
```

Out[38]:
```
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: Pet Detective, Dumb & Dumber, Mrs. Doubtfire are the most similar movies to Liar Liar

In [39]:
```
# Pearson User Similarity
user_similarity_df =  df_pivot.T.corr(method = 'pearson')
```

In [40]:
```
# Recommending similar Users
def get_similar_movies(user_id):
    similar_score = user_similarity_df.loc[user_id].sort_values(ascending=False)
    return similar_score.loc[similar_score.index!=user_id][:10]
```

```
get_similar_movies(1)
```

Out[40]:

```
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
```

Cosine Similarity

In [41]:

```
df_pivot
```

Out[41]:

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)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	...	Young Guns (1988)	Young Guns II (1990)
UserID													
1	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	3.845286	3.540005	...	3.524645	3.406558
2	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	3.845286	3.540005	...	3.524645	3.406558
3	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	3.845286	3.540005	...	5.000000	4.000000
4	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	3.845286	3.540005	...	3.524645	3.406558
5	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	3.845286	3.540005	...	3.524645	3.406558
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6036	3.562715	3.000000	3.53966	3.602571	3.579511	2.000000	4.000000	3.444023	3.845286	3.540005	...	3.524645	3.406558
6037	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	4.000000	3.540005	...	3.524645	3.406558
6038	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	3.845286	3.540005	...	3.524645	3.406558
6039	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	3.845286	3.540005	...	3.524645	3.406558
6040	3.562715	3.568449	3.53966	3.602571	3.579511	3.518136	3.586771	3.444023	5.000000	3.540005	...	3.524645	3.406558

6040 rows × 3706 columns

In [42]:

```python
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_pivot.columns , columns = df_pivot.columns)
user_similarity_df = pd.DataFrame(cosine_similarity(df_pivot), index = df_pivot.index , columns = df_pivot.index)
```

In [43]:
```
# 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)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	...	Young Guns (1988)	Yo Gu (1
Title													
$1,000,000 Duck (1971)	1.000000	0.999073	0.999067	0.998657	0.999632	0.994960	0.996346	0.996193	0.996648	0.999017	...	0.995802	0.99
'Night Mother (1986)	0.999073	1.000000	0.998797	0.998495	0.999403	0.994878	0.996011	0.995903	0.996654	0.998775	...	0.995551	0.99
'Til There Was You (1997)	0.999067	0.998797	1.000000	0.998430	0.999396	0.994806	0.996059	0.996158	0.996605	0.998800	...	0.995505	0.99
...And Justice for All (1979)	0.998657	0.998495	0.998430	1.000000	0.998996	0.994480	0.995668	0.995471	0.996444	0.998409	...	0.995066	0.99
1-900 (1994)	0.999632	0.999403	0.999396	0.998996	1.000000	0.995367	0.996501	0.996516	0.997161	0.999357	...	0.996074	0.99
...	...	...	...	...	...	...	...	...	...	...	...	...	
Zachariah (1971)	0.999621	0.999392	0.999374	0.998985	0.999953	0.995356	0.996490	0.996506	0.997149	0.999342	...	0.996042	0.99
Zero Effect (1998)	0.997560	0.997357	0.997345	0.997014	0.997902	0.993547	0.994586	0.994372	0.995204	0.997248	...	0.994275	0.99
Zero Kelvin (Kjærlighetens kjøtere) (1995)	0.999647	0.999418	0.999400	0.999019	0.999978	0.995382	0.996518	0.996531	0.997181	0.999368	...	0.996092	0.99
Zeus and Roxanne (1997)	0.999325	0.999078	0.999093	0.998673	0.999639	0.995130	0.996320	0.996402	0.996849	0.999026	...	0.995747	0.99
eXistenZ (1999)	0.995586	0.995417	0.995465	0.995004	0.995934	0.991565	0.992495	0.992482	0.993361	0.995421	...	0.992275	0.99

3706 rows × 3706 columns

In [44]:
```python
# User-User Cosine Matrix
display(user_similarity_df)
```

UserID	1	2	3	4	5	6	7	8	9	10	...	6031	6032	6033
UserID														
1	1.000000	0.998529	0.999140	0.999407	0.996507	0.999141	0.999399	0.998539	0.999139	0.996154	...	0.998567	0.998549	0.999231
2	0.998529	1.000000	0.998362	0.998605	0.995586	0.998214	0.998625	0.997568	0.998289	0.995198	...	0.997708	0.997694	0.998483
3	0.999140	0.998362	1.000000	0.999277	0.996410	0.998898	0.999261	0.998270	0.998903	0.995843	...	0.998377	0.998393	0.999016
4	0.999407	0.998605	0.999277	1.000000	0.996674	0.999153	0.999452	0.998623	0.999173	0.996005	...	0.998633	0.998598	0.999308
5	0.996507	0.995586	0.996410	0.996674	1.000000	0.996350	0.996567	0.995714	0.996247	0.993246	...	0.995927	0.995871	0.996468
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6036	0.990504	0.989994	0.990390	0.990760	0.987999	0.990155	0.990510	0.989780	0.990180	0.986901	...	0.989696	0.990025	0.990396
6037	0.998194	0.997485	0.998039	0.998321	0.995433	0.997931	0.998189	0.997317	0.997971	0.994979	...	0.997480	0.997525	0.998099
6038	0.999399	0.998567	0.999284	0.999450	0.996710	0.999119	0.999486	0.998589	0.999179	0.996206	...	0.998602	0.998562	0.999337
6039	0.999010	0.998260	0.998896	0.999102	0.996368	0.998782	0.999098	0.998201	0.998776	0.995886	...	0.998227	0.998195	0.998914
6040	0.995106	0.994186	0.994837	0.995266	0.992729	0.994965	0.995234	0.994427	0.994896	0.991457	...	0.994163	0.994551	0.995041

6040 rows × 6040 columns

In [45]:
```python
# Model Fit
model = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
model.fit(df_pivot.T)
```

Out[45]:

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

In [46]:

```
# 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, -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) 0.006305739637256778
Foreign Student  (1994) 0.006429487323369787
The  Man from Down Under (1943) 0.006440697641172499
Message to Love: The Isle of Wight Festival  (1996) 0.006449437417632597
Detroit 9000  (1973) 0.006451987589464858
Kestrel's Eye (Falkens öga)  (1998) 0.006453937949174038
Get Over It  (1996) 0.006460142127208912
Dadetown  (1995) 0.006460329570545986
Hangmen Also Die  (1943) 0.00646101744283023
Spirits of the Dead (Tre Passi nel Delirio)  (1968) 0.006463890330667965
```

In [47]:

```
get_similar_movies('Liar Liar  (1997)', df_pivot, 3)
```

```
The  Last Time I Committed Suicide (1997) 0.0035862824012093952
Unhook the Stars  (1996) 0.003597535117602302
The  Leading Man (1996) 0.003599562049961258
```

- The Last Time I Committed Suicide, Unhook the Stars, The Leading Man are most similar items to Liar Liar using Cosine Similarity Approach

Matrix Factorization

In [52]:
```python
# Creating Dataset
from surprise import SVD, Dataset, Reader

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

In [53]:
```python
from surprise.model_selection import train_test_split

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

In [54]:
```python
# Fitting model
model = SVD()
model.fit(trainset)
```

Out[54]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7b825509d7b0>
```

In [55]:
```python
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 predictions if pred.r_ui > 0]
    return sum(errors) / len(errors) * 100

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

```
RMSE: 0.8753
MAPE: 26.481380843212047%
```

- RMSE and MAPE for Matrix Factorization model is 0.8785 and 26.612%

Embeddings

In [56]:
```
# 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 [57]:
```
# Matrix with Embedding 4
display(item_similarity.head())
display(user_similarity.head())
```

	0	1	2	3	4	5	6	7	8	9	...	3696	3697	369
0	1.000000	0.791374	0.426905	-0.124354	0.440514	0.532764	0.325651	0.801991	0.615457	0.689092	...	0.095535	-0.633560	-0.0412
1	0.791374	1.000000	-0.037741	-0.571559	0.093719	0.542384	-0.151460	0.471854	0.037136	0.106145	...	-0.452741	-0.283553	0.2180
2	0.426905	-0.037741	1.000000	0.773477	0.485794	0.353873	0.646385	0.689516	0.750651	0.793304	...	0.590195	-0.135533	-0.4536
3	-0.124354	-0.571559	0.773477	1.000000	0.043514	-0.259854	0.342354	0.104024	0.431611	0.503945	...	0.831972	-0.055485	-0.7410
4	0.440514	0.093719	0.485794	0.043514	1.000000	0.756890	0.948281	0.844341	0.800799	0.638886	...	0.013336	-0.051266	0.4812

5 rows × 3706 columns

	0	1	2	3	4	5	6	7	8	9	...	6030	6031	
0	1.000000	0.278054	0.362066	0.226506	-0.343729	0.580129	0.711956	-0.017147	-0.282703	0.257036	...	0.832630	-0.777970	0.33
1	0.278054	1.000000	0.352007	-0.868484	-0.811124	0.425622	-0.476371	-0.457857	0.240524	0.756282	...	0.744543	-0.768579	0.93
2	0.362066	0.352007	1.000000	-0.092753	-0.682759	0.961057	0.076580	-0.735197	-0.722447	0.821173	...	0.532001	-0.628078	0.62
3	0.226506	-0.868484	-0.092753	1.000000	0.598981	-0.071047	0.842662	0.407353	-0.445605	-0.585533	...	-0.317816	0.365031	-0.74
4	-0.343729	-0.811124	-0.682759	0.598981	1.000000	-0.669066	0.270315	0.404865	-0.002527	-0.767336	...	-0.767823	0.733381	-0.95

5 rows × 6040 columns

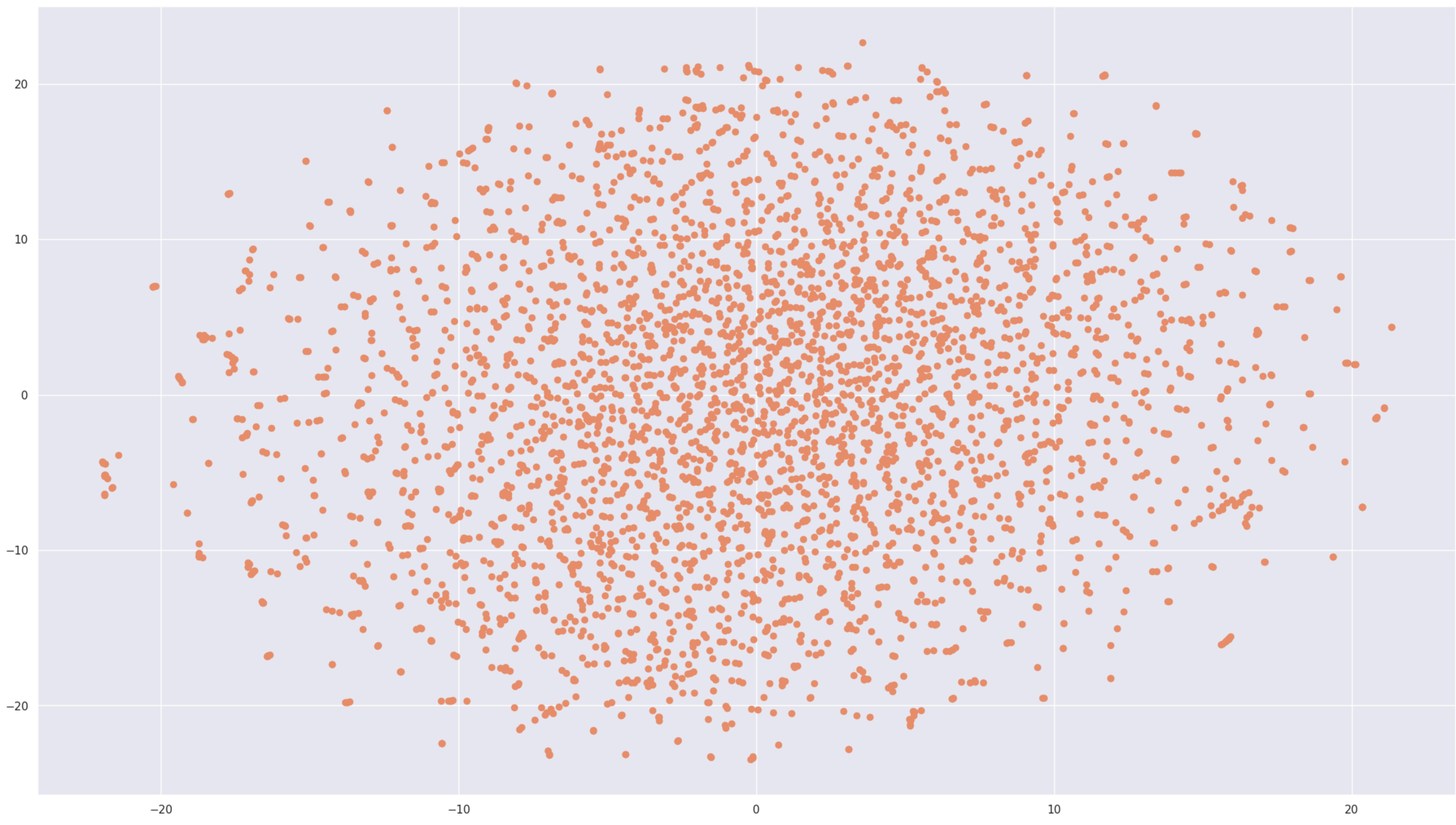
In [58]:
```python
# Embeddings Visualization

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

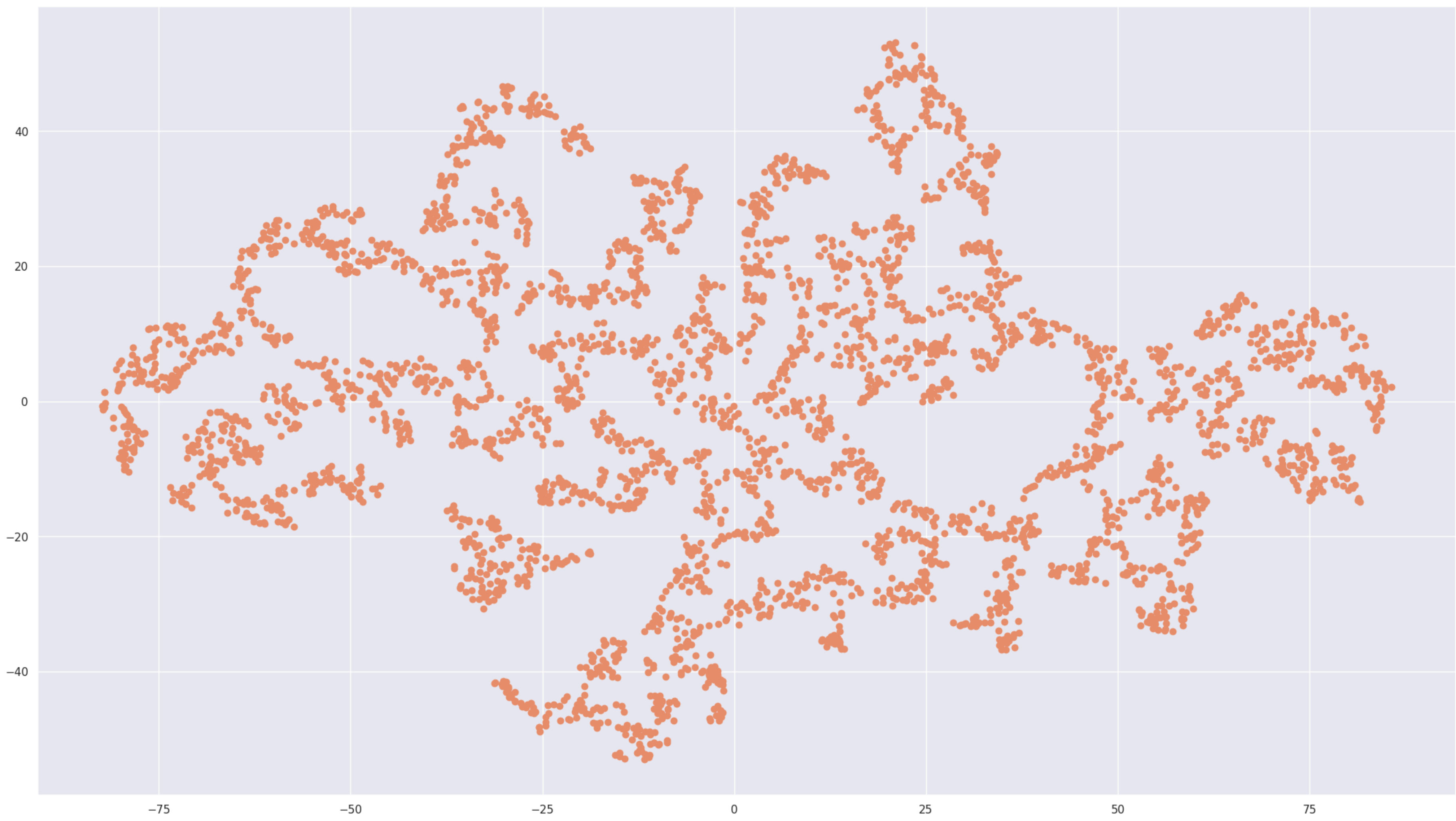
In [59]:
```
# Embeddings Visualization with 2 Factors

from sklearn.manifold import TSNE

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

```python
# 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()
```




```
In [ ]:
```

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

User Based Approach

In [60]:
```python
# Creating User Dataframe
df_user = df_merged.copy()
```

In [61]:
```python
# Getting New User Input
new_user_ratings = {'MovieID': [1, 5, 20], 'Rating': [4, 5, 3]}
new_user_df = pd.DataFrame(new_user_ratings)
```

In [62]:
```python
# 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', values='Rating')
```

In [63]:
```python
# 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_values(ascending=False)[1:101]
```

In [64]:
```python
# Weighted Ratings
weighted_ratings = pivot_table.T.copy()
for user in similar_users.index:
    weighted_ratings.loc[user] = weighted_ratings.loc[user] *  similar_users[user]
```

In [65]:
```python
# Top 10 movies
recommended_movies = weighted_ratings.sum(axis=0) / similar_users.sum()
top_movies = recommended_movies.sort_values(ascending=False)[:10]
```

In [66]:
```python
# Display Top Movies
top_movies
```

```
Out[66]: 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

Insights

1. **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.

2. **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.

3. **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.

4. **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.

5. **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

1. **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.

2. **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.

3. **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.

4. **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.

5. **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.
