Movielens_Project

November 23, 2023

[2]:
```python
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[3]:
```python
pip install matplotlib seaborn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/site-
packages (3.6.3)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/site-
packages (0.12.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/site-packages (from matplotlib) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/site-
packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/site-packages (from matplotlib) (4.33.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/site-packages (from matplotlib) (1.4.3)
Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.10/site-
packages (from matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/site-packages (from matplotlib) (22.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/site-
packages (from matplotlib) (9.1.1)
Requirement already satisfied: pyparsing>=2.2.1 in
/usr/local/lib/python3.10/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/site-
packages (from seaborn) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/site-
packages (from pandas>=0.25->seaborn) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/site-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
WARNING: There was an error checking the latest version of pip.
Note: you may need to restart the kernel to use updated packages.
```

[4]:
```python
# 1.Import the three datasets

import os

# Get the current working directory
current_directory = os.getcwd()

# Define the file names and extensions
movies_file_name = "movies.dat"
ratings_file_name = "ratings.dat"
users_file_name = "users.dat"

# Create the absolute file paths
movies_file = os.path.join(current_directory, movies_file_name)
ratings_file = os.path.join(current_directory, ratings_file_name)
users_file = os.path.join(current_directory, users_file_name)

# Define the column names for each dataset
movies_columns = ['MovieID', 'Title', 'Genres']
ratings_columns = ['UserID', 'MovieID', 'Rating', 'Timestamp']
users_columns = ['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code']

# Read the datasets into pandas dataframes
movies_df = pd.read_csv(movies_file, sep='::', header=None,␣
 ↪names=movies_columns, encoding='latin-1', engine='python')
ratings_df = pd.read_csv(ratings_file, sep='::', header=None,␣
 ↪names=ratings_columns, encoding='latin-1', engine='python')
users_df = pd.read_csv(users_file, sep='::', header=None, names=users_columns,␣
 ↪encoding='latin-1', engine='python')

# Display the first few rows of each dataframe
print("Movies DataFrame:")
print(movies_df.head())

print("\nRatings DataFrame:")
print(ratings_df.head())

print("\nUsers DataFrame:")
print(users_df.head())
```

```
Movies DataFrame:
   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 DataFrame:
   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

Users DataFrame:
   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
```

[21]:
```
movies_df.shape
```

[21]:
```
(3883, 3)
```

[22]:
```
ratings_df.shape
```

[22]:
```
(1000209, 4)
```

[23]:
```
users_df.shape
```

[23]:
```
(6040, 5)
```

[ ]:
```
# 2.Create a new dataset [Master_Data] with the following columns MovieID Title␣
↪UserID Age Gender Occupation
# Rating. (Hint: (i) Merge two tables at a time. (ii) Merge the tables using␣
↪two primary keys MovieID & UserId)
```

[5]:
```
# Get the current working directory
current_directory = os.getcwd()

# Merge ratings_df and users_df on 'UserID'
merged_ratings_users = pd.merge(ratings_df, users_df, on='UserID')

# Merge merged_ratings_users and movies_df on 'MovieID'
master_data = pd.merge(merged_ratings_users, movies_df, on='MovieID')

# Select the desired columns
```

```
master_data = master_data[['MovieID', 'Title', 'UserID', 'Age', 'Gender',␣
↪'Occupation', 'Rating']]

# Display the first few rows of the Master_Data dataframe
print("Master_Data DataFrame:")
print(master_data.head())
```

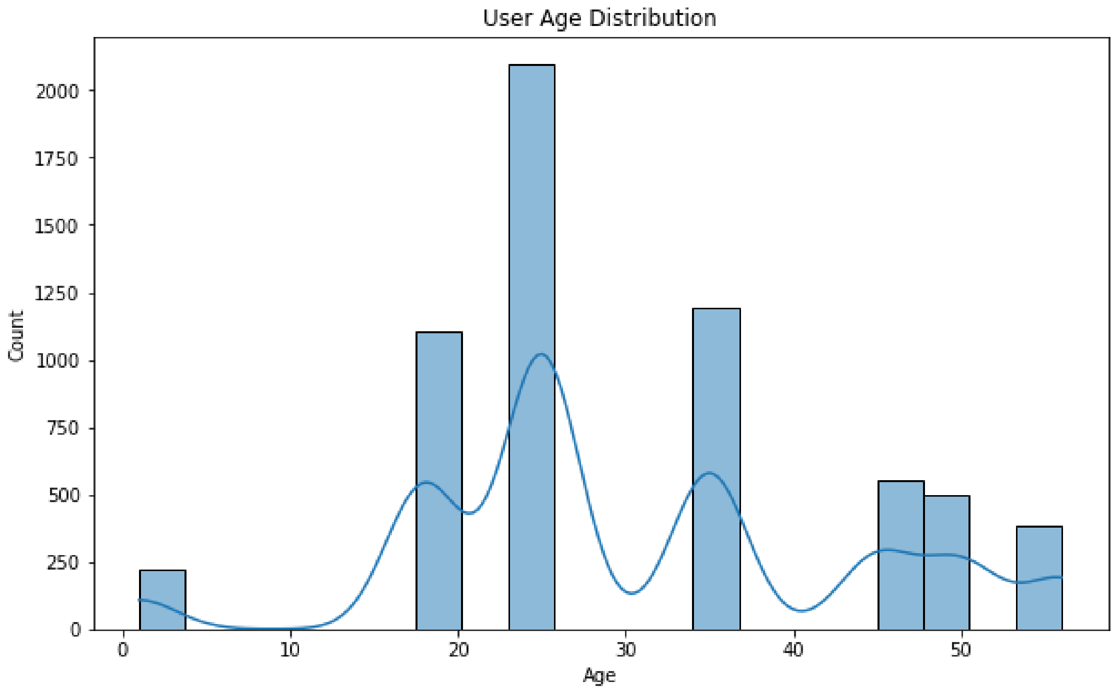
```
Master_Data DataFrame:
   MovieID                                   Title  UserID  Age Gender  \
0     1193  One Flew Over the Cuckoo's Nest (1975)       1    1      F
1     1193  One Flew Over the Cuckoo's Nest (1975)       2   56      M
2     1193  One Flew Over the Cuckoo's Nest (1975)      12   25      M
3     1193  One Flew Over the Cuckoo's Nest (1975)      15   25      M
4     1193  One Flew Over the Cuckoo's Nest (1975)      17   50      M

   Occupation  Rating
0          10       5
1          16       5
2          12       4
3           7       4
4           1       5
```

[31]:
```
# Print the column names of the DataFrame
print(master_data.columns)
```

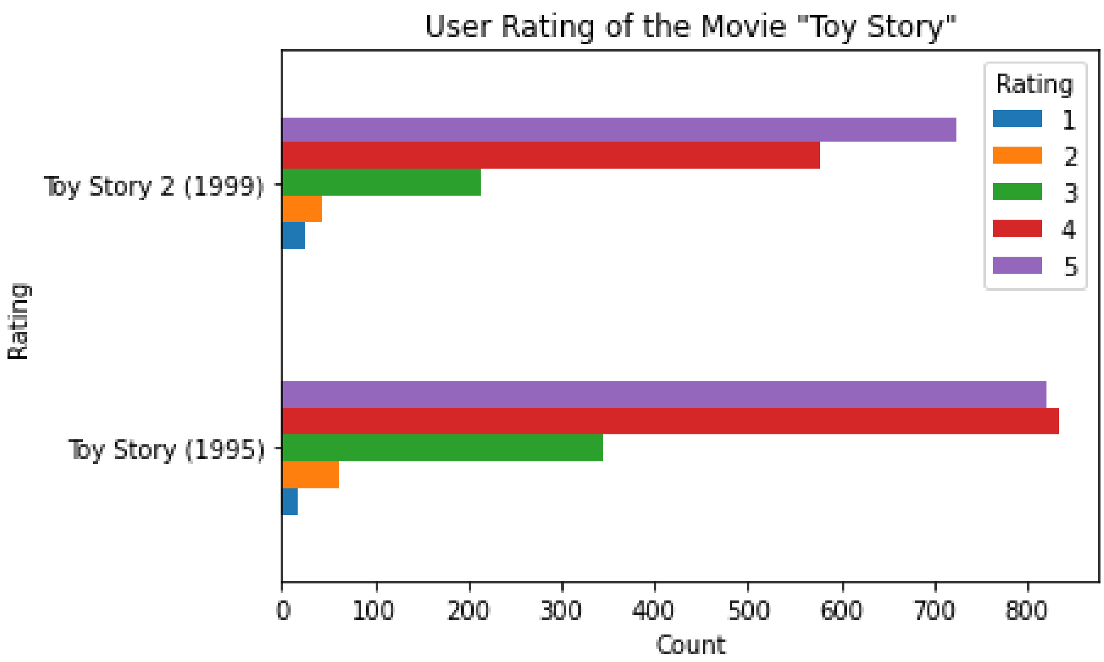
```
Index(['MovieID', 'Title', 'UserID', 'Age', 'Gender', 'Occupation', 'Rating'],
dtype='object')
```

[ ]:
```
# 3.Explore the datasets using visual representations (graphs or tables),
    also include your comments on the following:
```

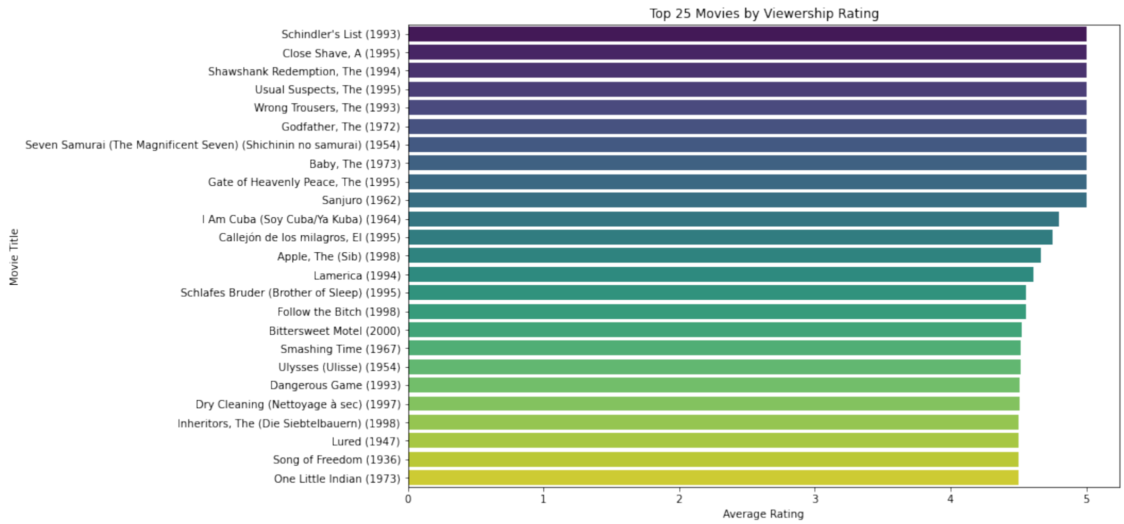
[28]:
```
# 3.1 User Age Distribution
# Plot the User Age Distribution
plt.figure(figsize=(10, 6))
sns.histplot(users_df['Age'], bins=20, kde=True)
plt.title('User Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



[23]:
```
# 3.2 User rating of the movie "Toy Story"

toystoryRating = master_data[master_data['Title'].str.contains('Toy Story')]

# Group by Title and Rating, count occurrences, and plot
toystoryRating.groupby(["Title", "Rating"]).size().unstack().plot(
    kind='barh', stacked=False, legend=True
)

plt.title('User Rating of the Movie "Toy Story"')
plt.xlabel('Count')
plt.ylabel('Rating')
plt.show()
```



[25]:
```
# 3.3 Top 25 movies by viewership rating
# Group by MovieID and calculate the mean rating for each movie
movie_ratings = master_data.groupby('MovieID')['Rating'].mean()

# Sort movies by mean rating in descending order
top_movies = movie_ratings.sort_values(ascending=False).head(25)

# Get the movie titles for the top 25 movies
top_movies_titles = master_data[master_data['MovieID'].isin(top_movies.
 ↪index)]['Title'].unique()

# Plot the top 25 movies by viewership rating
plt.figure(figsize=(12, 8))
sns.barplot(x=top_movies, y=top_movies_titles, palette='viridis')
plt.title('Top 25 Movies by Viewership Rating')
plt.xlabel('Average Rating')
plt.ylabel('Movie Title')
plt.show()
```



[27]:
```python
# 3.4 Find the ratings for all the movies reviewed by for a particular user of␣
↪user id = 2696

# Filter the master_data DataFrame for user ID 2696
user_2696_ratings = master_data[master_data['UserID'] == 2696][['Title',␣
↪'Rating']]

# Display the ratings for the movies reviewed by user 2696 as a table
print(user_2696_ratings.to_markdown(index=False))
```

```
| Title                                         |   Rating |
|:----------------------------------------------|---------:|
| Back to the Future (1985)                     |        2 |
| E.T. the Extra-Terrestrial (1982)             |        3 |
| L.A. Confidential (1997)                      |        4 |
| Lone Star (1996)                              |        5 |
| JFK (1991)                                    |        1 |
| Talented Mr. Ripley, The (1999)               |        4 |
| Midnight in the Garden of Good and Evil (1997) |        4 |
| Cop Land (1997)                               |        3 |
| Palmetto (1998)                               |        4 |
| Perfect Murder, A (1998)                      |        4 |
| Game, The (1997)                              |        4 |
| I Know What You Did Last Summer (1997)        |        2 |
| Devil's Advocate, The (1997)                  |        4 |
| Psycho (1998)                                 |        4 |
| Wild Things (1998)                            |        4 |
| Basic Instinct (1992)                         |        4 |
| Lake Placid (1999)                            |        1 |
| Shining, The (1980)                           |        4 |
```

```
| I Still Know What You Did Last Summer (1998)    |         2 |
| Client, The (1994)                              |         3 |
```

[ ]:
```python
# 4. Feature Engineering:
```

[36]:
```python
# 4.1 Find out all the unique genres (Hint: split the data in column genre␣
↪making a list and
# then process the data to find out only the unique categories of genres)

# Split the genres column into lists of genres
genres_lists = movies_df['Genres'].str.split('|')

# Create a set to store unique genres
unique_genres = set()

# Iterate over the lists of genres and add unique genres to the set
for genres_list in genres_lists:
    unique_genres.update(genres_list)

# Convert the set of unique genres to a list
unique_genres_list = list(unique_genres)

# Display the unique genres
print("Unique Genres:")
for genre in unique_genres_list:
    print(genre)
```

```
Unique Genres:
Crime
Horror
Animation
Children's
Adventure
Musical
Thriller
Documentary
War
Mystery
Comedy
Western
Fantasy
Sci-Fi
Romance
Film-Noir
Action
Drama
```

[34]:
```python
# Split the genres column into lists of genres
genres_lists = movies_df['Genres'].str.split('|')

# Create a set to store unique genres
unique_genres = set()

# Iterate over the lists of genres and add unique genres to the set
for genres_list in genres_lists:
    unique_genres.update(genres_list)

# Convert the set of unique genres to a list
unique_genres_list = list(unique_genres)

# Display the unique genres
print("Unique Genres:")
for genre in unique_genres_list:
    print(genre)
```

```
Unique Genres:
Crime
Horror
Animation
Children's
Adventure
Musical
Thriller
Documentary
War
Mystery
Comedy
Western
Fantasy
Sci-Fi
Romance
Film-Noir
Action
Drama
```

[38]:
```python
# 4.2 Create a separate column for each genre category with a one-hot encoding␣
↪( 1 and 0)
# whether or not the movie belongs to that genre.# If 'master_data' does not␣
↪have 'Genres', merge it with 'movies_df'
if 'Genres' not in master_data.columns:
    master_data = pd.merge(master_data, movies_df[['MovieID', 'Genres']],␣
↪on='MovieID', how='left')

# Assuming master_data has a 'Genres' column
```

```python
genres_lists = master_data['Genres'].str.split('|')

# One-hot encode genres
one_hot_encoded_genres = pd.get_dummies(genres_lists.apply(pd.Series).stack()).
↪groupby(level=0).sum()

# Concatenate one-hot encoded genres with master_data
master_data = pd.concat([master_data, one_hot_encoded_genres], axis=1)

# Display the updated master_data
print(master_data)
```

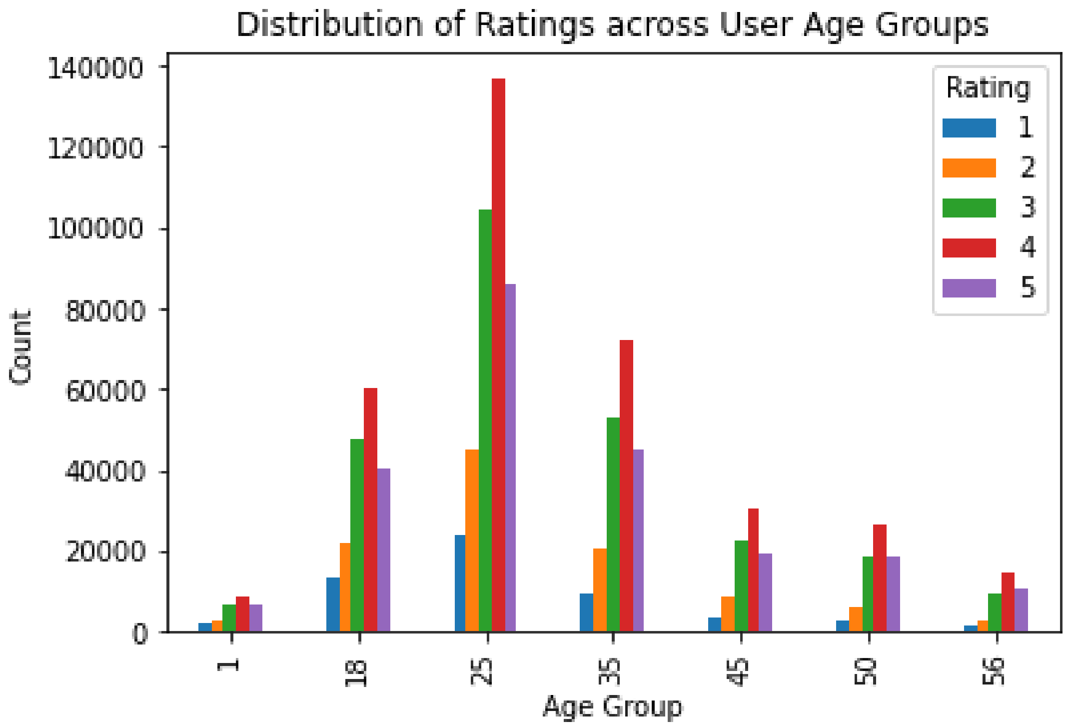
```
         MovieID                                       Title  UserID  Age  \
0           1193      One Flew Over the Cuckoo's Nest (1975)       1    1
1           1193      One Flew Over the Cuckoo's Nest (1975)       2   56
2           1193      One Flew Over the Cuckoo's Nest (1975)      12   25
3           1193      One Flew Over the Cuckoo's Nest (1975)      15   25
4           1193      One Flew Over the Cuckoo's Nest (1975)      17   50
…            …                                           …     …   …
1000204     2198                         Modulations (1998)    5949   18
1000205     2703                      Broken Vessels (1998)    5675   35
1000206     2845                          White Boys (1999)    5780   18
1000207     3607                    One Little Indian (1973)    5851   18
1000208     2909  Five Wives, Three Secretaries and Me (1998)    5938   25

        Gender  Occupation  Rating                Genres  Action  Adventure  \
0            F          10       5                 Drama       0          0
1            M          16       5                 Drama       0          0
2            M          12       4                 Drama       0          0
3            M           7       4                 Drama       0          0
4            M           1       5                 Drama       0          0
…          …           …      …                    …      …         …
1000204      M          17       5           Documentary       0          0
1000205      M          14       3                 Drama       0          0
1000206      M          17       1                 Drama       0          0
1000207      F          20       5  Comedy|Drama|Western       0          0
1000208      M           1       4           Documentary       0          0

         …  Fantasy  Film-Noir  Horror  Musical  Mystery  Romance  Sci-Fi  \
0        …        0          0       0        0        0        0       0
1        …        0          0       0        0        0        0       0
2        …        0          0       0        0        0        0       0
3        …        0          0       0        0        0        0       0
4        …        0          0       0        0        0        0       0
…      …       …          …     …       …        …       …      …
1000204  …        0          0       0        0        0        0       0
1000205  …        0          0       0        0        0        0       0
```

```
1000206  …         0          0        0        0        0        0        0
1000207  …         0          0        0        0        0        0        0
1000208  …         0          0        0        0        0        0        0

         Thriller  War  Western
0               0    0        0
1               0    0        0
2               0    0        0
3               0    0        0
4               0    0        0
…             …   …        …
1000204         0    0        0
1000205         0    0        0
1000206         0    0        0
1000207         0    0        1
1000208         0    0        0

[1000209 rows x 44 columns]
```

[42]:

```
# 4.3 Determine the features affecting the ratings of any particular movie.

# Explore the relationship between user age and ratings
plt.figure(figsize=(12, 6))
ratings_by_age = master_data.groupby(['Age', 'Rating']).size().unstack()
ratings_by_age.plot(kind='bar', stacked=False, legend=True)
plt.title('Distribution of Ratings across User Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.show()
```

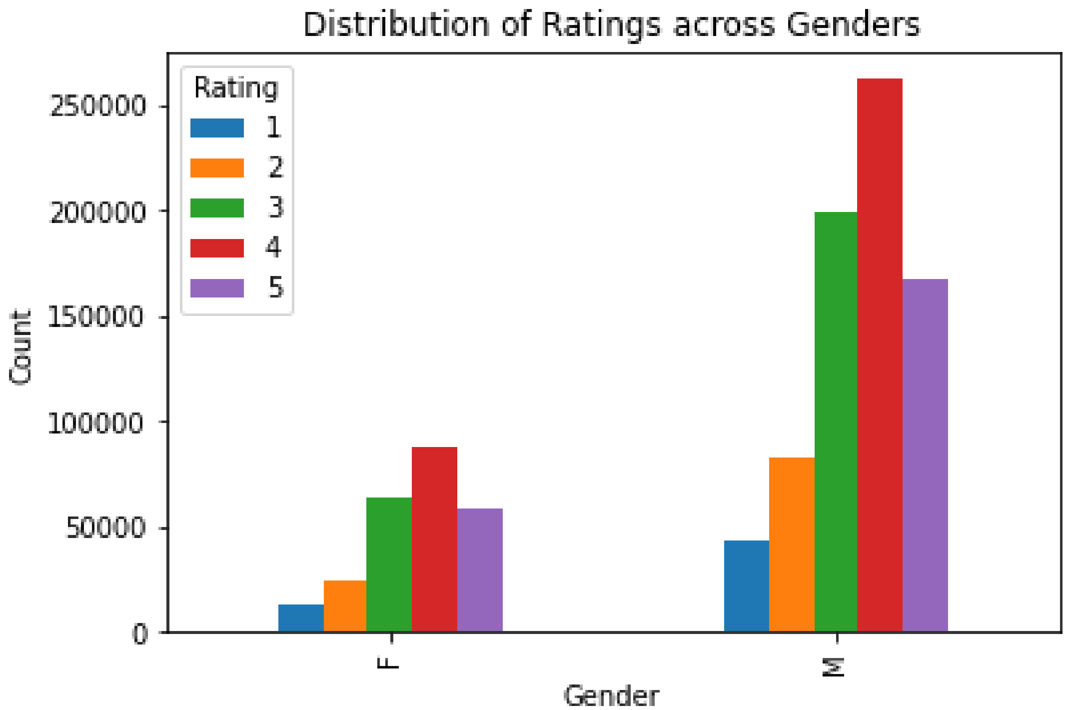
```
<Figure size 864x432 with 0 Axes>
```



[43]:

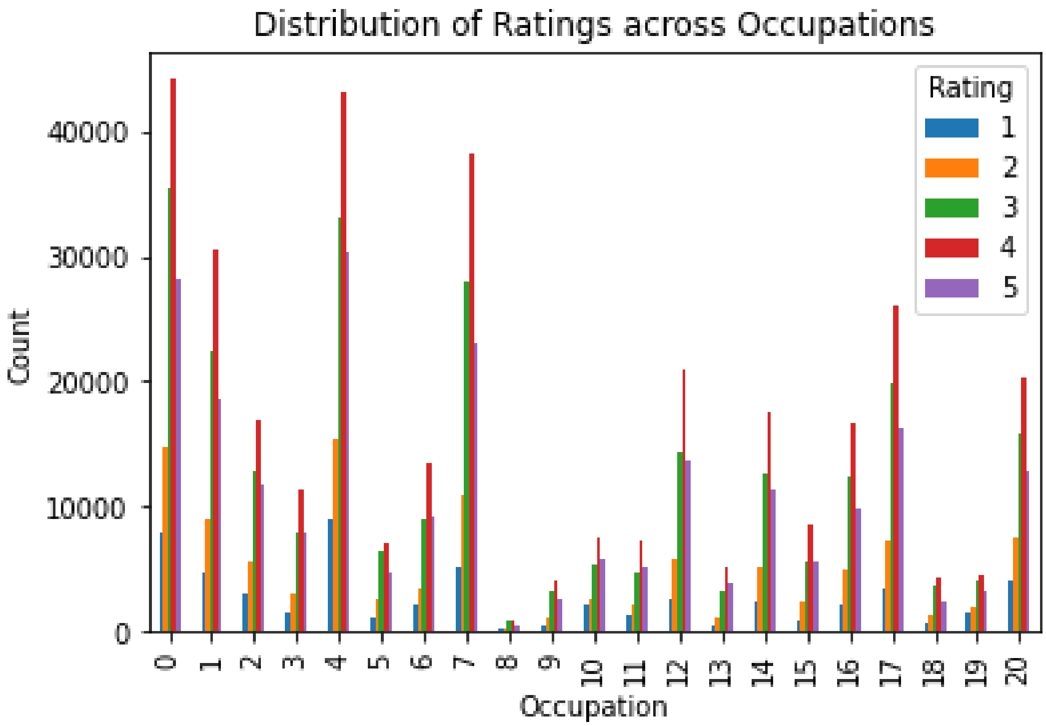
```
# Gender vs. Ratings:
plt.figure(figsize=(10, 6))
ratings_by_gender = master_data.groupby(['Gender', 'Rating']).size().unstack()
ratings_by_gender.plot(kind='bar', stacked=False, legend=True)
plt.title('Distribution of Ratings across Genders')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```

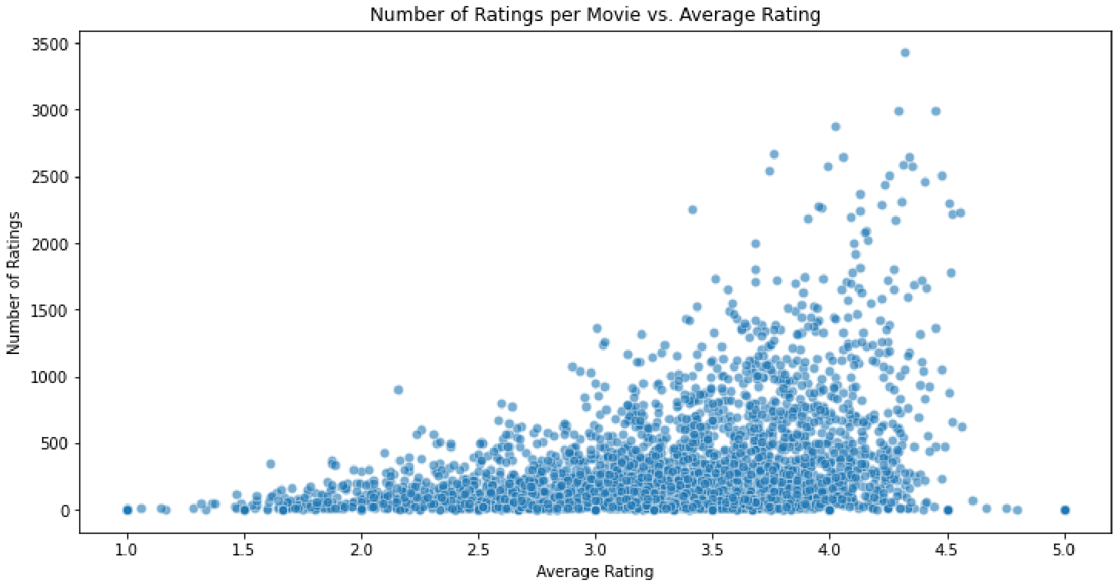


[48]:
```
# Occupation vs. Ratings:
plt.figure(figsize=(12, 10))
ratings_by_occupation = master_data.groupby(['Occupation', 'Rating']).size().
  ↪unstack()
ratings_by_occupation.plot(kind='bar', stacked=False, legend=True)
plt.title('Distribution of Ratings across Occupations')
plt.xlabel('Occupation')
plt.ylabel('Count')
plt.show()
```

```
<Figure size 864x720 with 0 Axes>
```



[7]:
```
# Calculate the number of ratings per movie :
ratings_per_movie = master_data.groupby('Title')['Rating'].count().reset_index()

# Merge with the average rating per movie
movie_ratings_stats = ratings_per_movie.merge(
    master_data.groupby('Title')['Rating'].mean().reset_index(),
    on='Title',
    how='inner',
    suffixes=('_count', '_avg')
)

# Plotting
plt.figure(figsize=(12, 6))

# Scatter plot with size representing the number of ratings
sns.scatterplot(x='Rating_avg', y='Rating_count', data=movie_ratings_stats,␣
 ↪alpha=0.6, s=40)

plt.title('Number of Ratings per Movie vs. Average Rating')
plt.xlabel('Average Rating')
plt.ylabel('Number of Ratings')
plt.show()
```




```
[ ]: # 4.Develop an appropriate model to predict the movie ratings
```

```
[31]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Assuming dfMaster contains your data
first_500 = master_data[:1000]

# Use the following features: 'MovieID', 'Age', 'Occupation'
features = first_500[['MovieID', 'Age', 'Occupation']].values

# Use 'Rating' as the label
labels = first_500['Rating'].values

# Split the data into training and testing sets
features_train, features_test, labels_train, labels_test = train_test_split(
    features, labels, test_size=0.2, random_state=42
)

# Decision Trees
decision_tree_model = DecisionTreeRegressor(random_state=42)
decision_tree_model.fit(features_train, labels_train)
labels_pred_decision_tree = decision_tree_model.predict(features_test)
mse_decision_tree = mean_squared_error(labels_test, labels_pred_decision_tree)
```

```python
print(f'Decision Tree Mean Squared Error: {mse_decision_tree}')

# Random Forest
random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)
random_forest_model.fit(features_train, labels_train)
labels_pred_random_forest = random_forest_model.predict(features_test)
mse_random_forest = mean_squared_error(labels_test, labels_pred_random_forest)
print(f'Random Forest Mean Squared Error: {mse_random_forest}')
```

```
Decision Tree Mean Squared Error: 0.668740168687103
Random Forest Mean Squared Error: 0.6553640841152201
```

[ ]:
```
In both cases:
Lower MSE values are generally better, indicating more accurate predictions on␣
↪average.
Considering the scale of your ratings, an MSE around 0.66 to 0.67 suggests that␣
↪the models are
making reasonably accurate predictions.
```

[14]:
```python
# Ridge Regression
from sklearn.linear_model import Ridge

# Assuming dfMaster contains your data
first_500 = master_data[:1000]

# Use the following features: 'MovieID', 'Age', 'Occupation'
features = first_500[['MovieID', 'Age', 'Occupation']].values

# Use 'Rating' as the label
labels = first_500['Rating'].values

# Split the data into training and testing sets
features_train, features_test, labels_train, labels_test = train_test_split(
    features, labels, test_size=0.2, random_state=42
)

# Ridge Regression without hyperparameter tuning
ridge_model = Ridge(alpha=1.0)  # You can adjust alpha as needed
ridge_model.fit(features_train, labels_train)
labels_pred_ridge = ridge_model.predict(features_test)

# Evaluate the Ridge Regression model
mse_ridge = mean_squared_error(labels_test, labels_pred_ridge)
print(f'Ridge Regression Mean Squared Error: {mse_ridge}')
```

```
Ridge Regression Mean Squared Error: 0.5713180074954795
```

[ ]:

```
Lower MSE values are generally better, indicating more accurate predictions on␣
↪average.
Ridge regression is the best model among the three models we developed here.
```