```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px


import warnings
# Set the warning filter to 'ignore'
warnings.filterwarnings('ignore')


# read data set

movies = pd.read_csv("/content/movies.dat", sep='::', engine='python', encoding='latin1')
movies.head()
```

	1	Toy Story (1995)	Animation|Children's|Comedy
0	2	Jumanji (1995)	Adventure|Children's|Fantasy
1	3	Grumpier Old Men (1995)	Comedy|Romance
2	4	Waiting to Exhale (1995)	Comedy|Drama
3	5	Father of the Bride Part II (1995)	Comedy
4	6	Heat (1995)	Action|Crime|Thriller

Next steps: Generate code with movies | View recommended plots | New interactive sheet

```
movies.columns =['MovieID', 'Title', 'Genres']
movies.dropna(inplace=True)
movies.head()
```

	MovieID	Title	Genres
0	2	Jumanji (1995)	Adventure|Children's|Fantasy
1	3	Grumpier Old Men (1995)	Comedy|Romance
2	4	Waiting to Exhale (1995)	Comedy|Drama
3	5	Father of the Bride Part II (1995)	Comedy
4	6	Heat (1995)	Action|Crime|Thriller

Next steps: Generate code with movies | View recommended plots | New interactive sheet

```
movies.shape
```

```
(3882, 3)
```

```
movies.describe()
```

	MovieID
count	3882.000000
mean	1986.560793
std	1146.483260
min	2.000000
25%	983.250000
50%	2010.500000
75%	2980.750000
max	3952.000000

```
movies.isnull().sum()
```

	0
MovieID	0
Title	0
Genres	0

dtype: int64

```
#Input ratings dataset
ratings = pd.read_csv("/content/ratings.dat",sep='::', engine='python')
ratings.columns =['UserID', 'MovieID', 'Rating', 'Timestamp']
ratings.dropna(inplace=True)

#Read the sample ratings dataset
ratings.head()
```

	UserID	MovieID	Rating	Timestamp
0	1	661	3	978302109
1	1	914	3	978301968
2	1	3408	4	978300275
3	1	2355	5	978824291
4	1	1197	3	978302268

```
ratings.shape
```

```
(1000208, 4)
```

```
ratings.describe()
```

	UserID	MovieID	Rating	Timestamp
count	1.000208e+06	1.000208e+06	1.000208e+06	1.000208e+06
mean	3.024515e+03	1.865541e+03	3.581563e+00	9.722437e+08
std	1.728411e+03	1.096041e+03	1.117102e+00	1.215256e+07
min	1.000000e+00	1.000000e+00	1.000000e+00	9.567039e+08
25%	1.506000e+03	1.030000e+03	3.000000e+00	9.653026e+08
50%	3.070000e+03	1.835000e+03	4.000000e+00	9.730180e+08
75%	4.476000e+03	2.770000e+03	4.000000e+00	9.752209e+08
max	6.040000e+03	3.952000e+03	5.000000e+00	1.046455e+09

```
ratings.isnull().sum()
```

	0
UserID	0
MovieID	0
Rating	0
Timestamp	0

dtype: int64

```
#Input users dataset
users = pd.read_csv("/content/ratings.dat",sep='::',engine='python')
users.columns =['UserID', 'Gender', 'Age',  'Zip-code']
users.dropna(inplace=True)

#Read the sample users dataset
users.head()
```

	UserID	Gender	Age	Zip-code
0	1	661	3	978302109
1	1	914	3	978301968
2	1	3408	4	978300275
3	1	2355	5	978824291
4	1	1197	3	978302268

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the data
users['Gender'] = label_encoder.fit_transform(users['Gender'])
users.head()
```

	UserID	Gender	Age	Zip-code
0	1	639	3	978302109
1	1	853	3	978301968
2	1	3177	4	978300275
3	1	2162	5	978824291
4	1	1107	3	978302268

```
users.shape
```

```
(1000208, 4)
```

```
users.describe()
```

	UserID	Gender	Age	Zip-code
count	1.000208e+06	1.000208e+06	1.000208e+06	1.000208e+06
mean	3.024515e+03	1.730481e+03	3.581563e+00	9.722437e+08
std	1.728411e+03	1.017027e+03	1.117102e+00	1.215256e+07
min	1.000000e+00	0.000000e+00	1.000000e+00	9.567039e+08
25%	1.506000e+03	9.650000e+02	3.000000e+00	9.653026e+08
50%	3.070000e+03	1.657000e+03	4.000000e+00	9.730180e+08
75%	4.476000e+03	2.565000e+03	4.000000e+00	9.752209e+08
max	6.040000e+03	3.705000e+03	5.000000e+00	1.046455e+09

```
users.isnull().sum()
```

	0
UserID	0
Gender	0
Age	0
Zip-code	0

dtype: int64

Data Cleaning :-

Concatenating the Datasets

```
df=pd.concat([movies,ratings,users],axis=1)
df.dropna()
df.head(5)
```

	MovieID	Title	Genres	UserID	MovieID	Rating	Timestamp	UserID	Gender	Age	Zip-code
0	2.0	Jumanji (1995)	Adventure|Children's|Fantasy	1	661	3	978302109	1	639	3	978302109
1	3.0	Grumpier Old Men (1995)	Comedy|Romance	1	914	3	978301968	1	853	3	978301968
2	4.0	Waiting to Exhale (1995)	Comedy|Drama	1	3408	4	978300275	1	3177	4	978300275
3	5.0	Father of the Bride Part II (1995)	Comedy	1	2355	5	978824291	1	2162	5	978824291

```
df.shape
```

```
(1000208, 11)
```

Removing unnecessary columns

```
df=df.drop(["Timestamp","Zip-code","MovieID","UserID"],axis=1)
df.head()
```

	Title	Genres	Rating	Gender	Age
0	Jumanji (1995)	Adventure|Children's|Fantasy	3	639	3
1	Grumpier Old Men (1995)	Comedy|Romance	3	853	3
2	Waiting to Exhale (1995)	Comedy|Drama	4	3177	4
3	Father of the Bride Part II (1995)	Comedy	5	2162	5
4	Heat (1995)	Action|Crime|Thriller	3	1107	3

```
df.describe()
```

	Rating	Gender	Age
count	1.000208e+06	1.000208e+06	1.000208e+06
mean	3.581563e+00	1.730481e+03	3.581563e+00
std	1.117102e+00	1.017027e+03	1.117102e+00
min	1.000000e+00	0.000000e+00	1.000000e+00
25%	3.000000e+00	9.650000e+02	3.000000e+00
50%	4.000000e+00	1.657000e+03	4.000000e+00
75%	4.000000e+00	2.565000e+03	4.000000e+00
max	5.000000e+00	3.705000e+03	5.000000e+00

```
df.isnull().sum()
```

	0
Title	996326
Genres	996326
Rating	0
Gender	0
Age	0

dtype: int64

Handling Missing values

```
df=df.dropna()
df.shape
```

```
(3882, 5)
```

```
# all 5 rating movies list count = 840
df[df['Rating'] == 5]
```

	Title	Genres	Rating	Gender	Age
3	Father of the Bride Part II (1995)	Comedy	5	2162	5
5	Sabrina (1995)	Comedy|Romance	5	1195	5
6	Tom and Huck (1995)	Adventure|Children's	5	2599	5
9	American President, The (1995)	Comedy|Drama|Romance	5	581	5
13	Cutthroat Island (1995)	Action|Adventure|Romance	5	970	5
...	...	...	...	...	...
3860	Giant Gila Monster, The (1959)	Horror|Sci-Fi	5	1186	5
3865	Phantom of the Opera, The (1943)	Drama|Thriller	5	1192	5
3866	Runaway (1984)	Sci-Fi|Thriller	5	1193	5
3870	Sorority House Massacre (1986)	Horror	5	841	5
3880	Two Family House (2000)	Drama	5	851	5

840 rows × 5 columns

```
# all 5 rating movies list and Age Less Then 25 count = 208

df[(df['Rating'] == 5) & (df['Age'] < 25 ) ]
```

	Title	Genres	Rating	Gender	Age
3	Father of the Bride Part II (1995)	Comedy	5	2162	5
5	Sabrina (1995)	Comedy|Romance	5	1195	5
6	Tom and Huck (1995)	Adventure|Children's	5	2599	5
9	American President, The (1995)	Comedy|Drama|Romance	5	581	5
13	Cutthroat Island (1995)	Action|Adventure|Romance	5	970	5
...	...	...	...	...	...
3860	Giant Gila Monster, The (1959)	Horror|Sci-Fi	5	1186	5
3865	Phantom of the Opera, The (1943)	Drama|Thriller	5	1192	5
3866	Runaway (1984)	Sci-Fi|Thriller	5	1193	5
3870	Sorority House Massacre (1986)	Horror	5	841	5
3880	Two Family House (2000)	Drama	5	851	5

840 rows × 5 columns

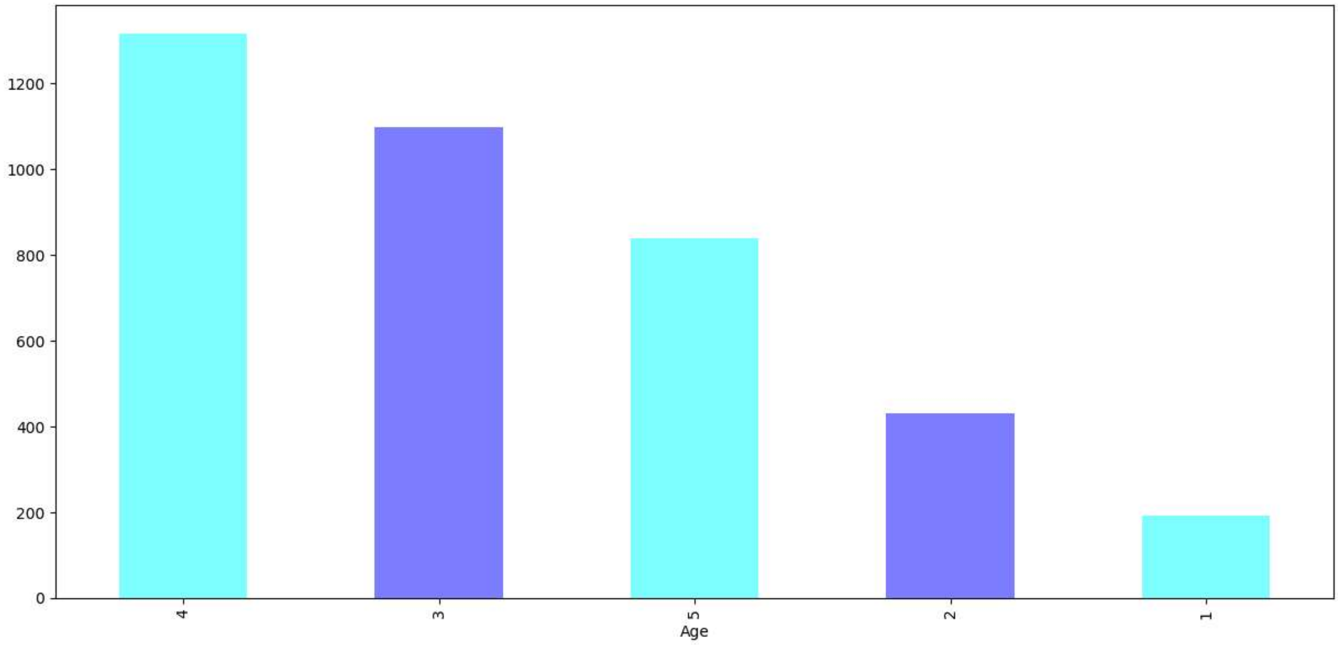
```
# all  movies rating less than 3 list and Age Lass Then 25 count = 47163

df[(df['Rating'] < 3) & (df['Age'] < 25 )]
```

	Title	Genres	Rating	Gender	Age
66	French Twist (Gazon maudit) (1995)	Comedy|Romance	2	1123	2
72	Bed of Roses (1996)	Drama|Romance	2	420	2
74	Screamers (1995)	Sci-Fi|Thriller	2	2891	2
82	Last Summer in the Hamptons (1995)	Comedy|Drama	2	841	2
90	Vampire in Brooklyn (1995)	Comedy|Romance	2	3032	2
...	...	...	...	...	...
3833	Uninvited Guest, An (2000)	Drama	2	1919	2
3837	Urban Legends: Final Cut (2000)	Horror	2	1157	2
3841	Beautiful (2000)	Comedy|Drama	2	138	2
3867	Slumber Party Massacre, The (1982)	Horror	2	2885	2
3878	Requiem for a Dream (2000)	Drama	2	259	2

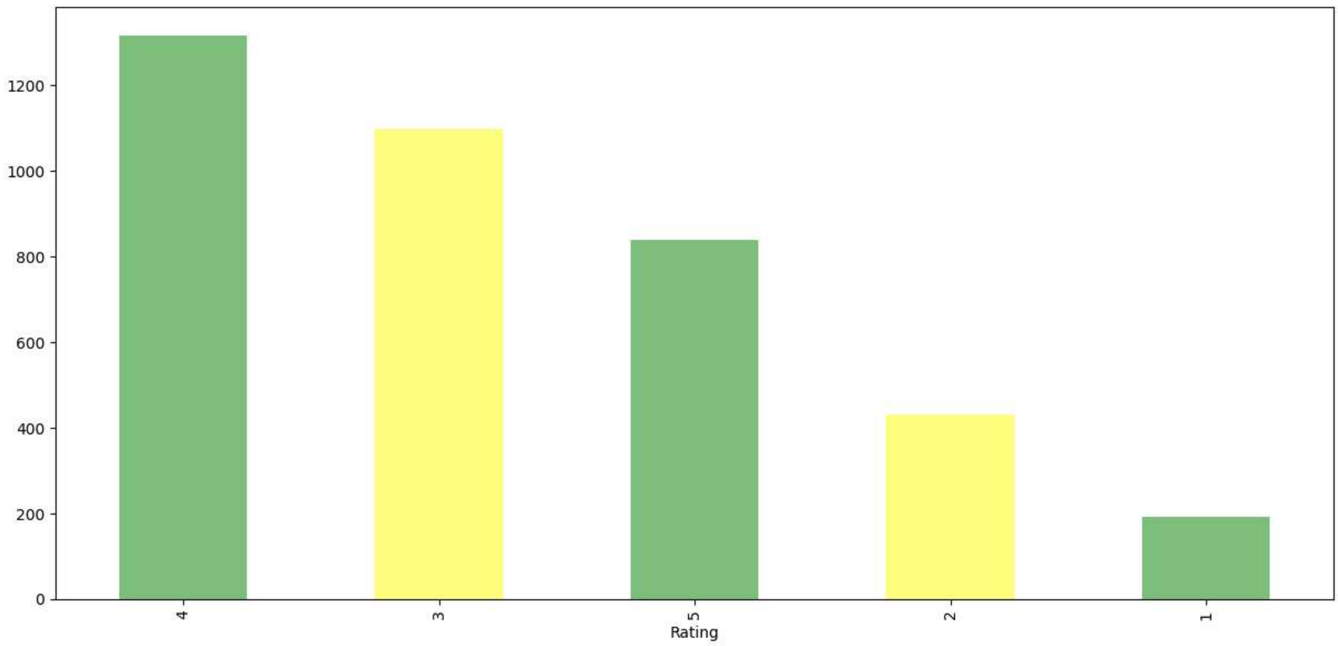
625 rows × 5 columns

Data Visualization

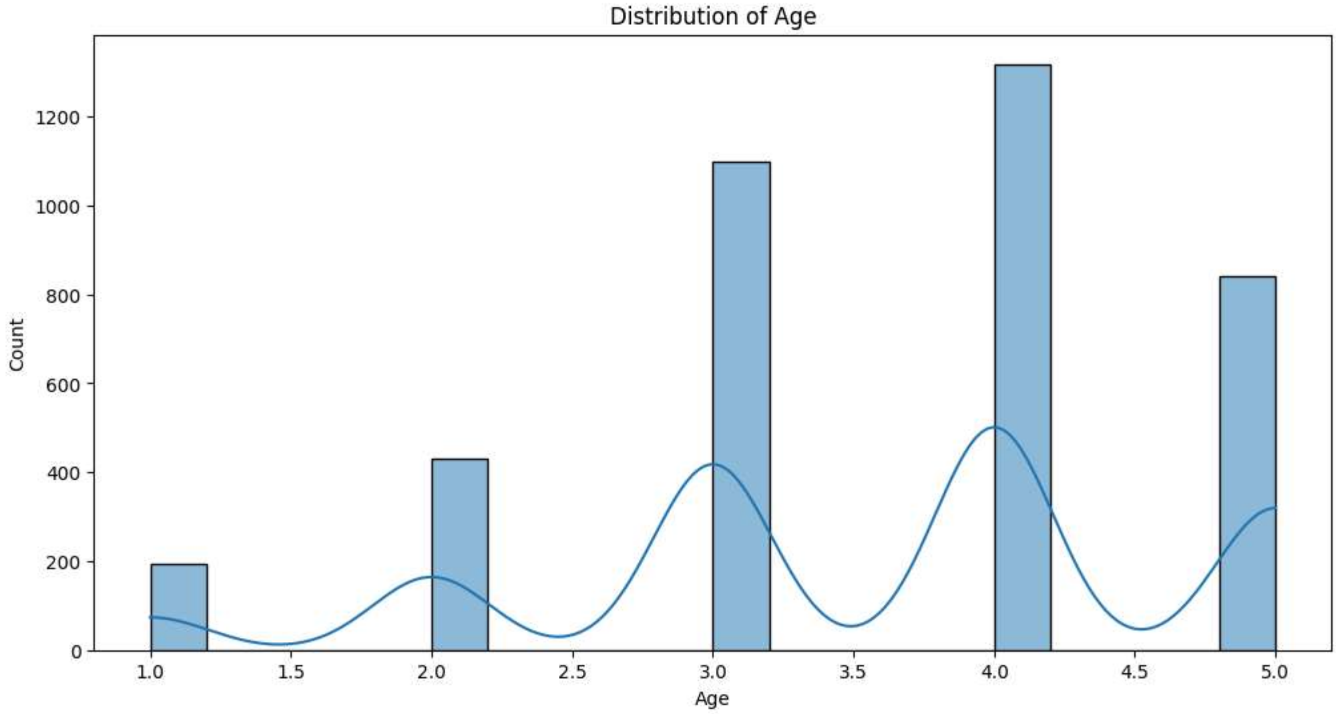
```
df['Age'].value_counts().plot(kind='bar', color= ['cyan', 'blue'],alpha=0.5,figsize=(15,7))
plt.show()
```



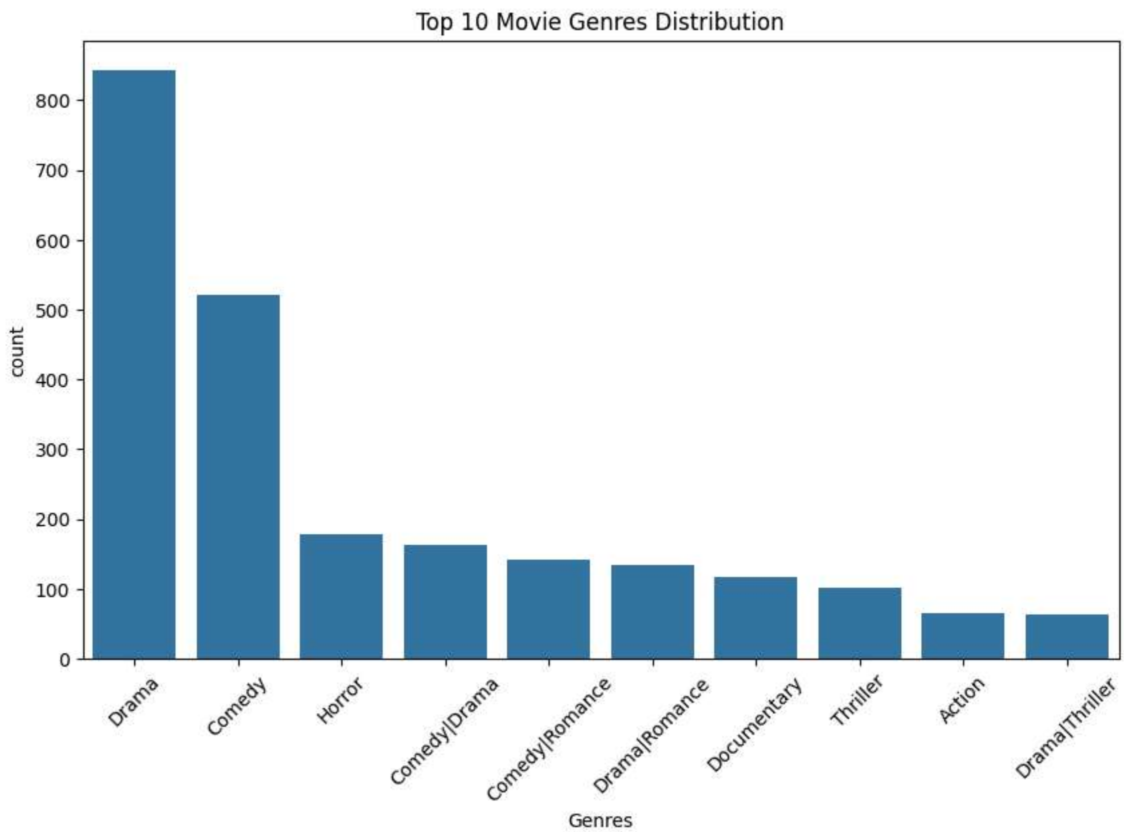

```
df['Rating'].value_counts().plot(kind='bar', color=['green', 'yellow'],alpha=0.5,figsize=(15,7))
plt.show()
```




```
# 2. Histogram for 'Age'
plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='Age', bins=20, kde=True)
plt.title('Distribution of Age')
plt.show()
```

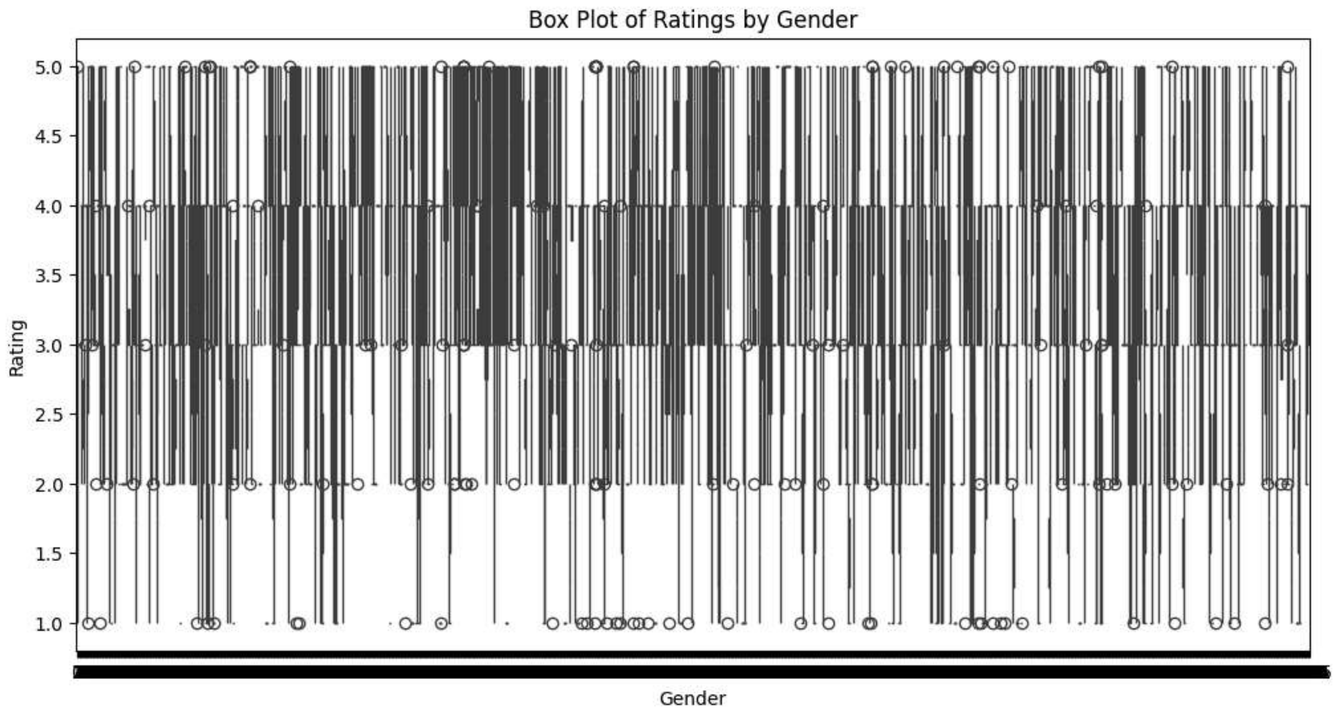



```
# Get the top 10 genres by count
top_genres = df['Genres'].value_counts().nlargest(10).index

# Filter the DataFrame to include only the top 10 genres
df_top_genres = df[df['Genres'].isin(top_genres)]


# Plot the count plot for the top 10 genres
plt.figure(figsize=(10, 6))
sns.countplot(x='Genres', data=df_top_genres, order=top_genres)
plt.title('Top 10 Movie Genres Distribution')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```



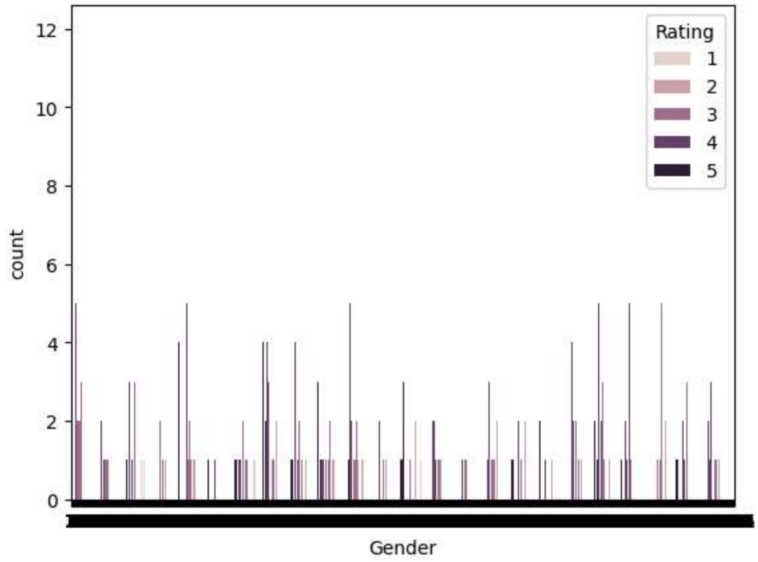

```
# 3. Box plot for 'Rating' by 'Gender'
plt.figure(figsize=(12, 6))
```

```
sns.boxplot(data=df, x='Gender', y='Rating')
plt.title('Box Plot of Ratings by Gender')
plt.show()
```

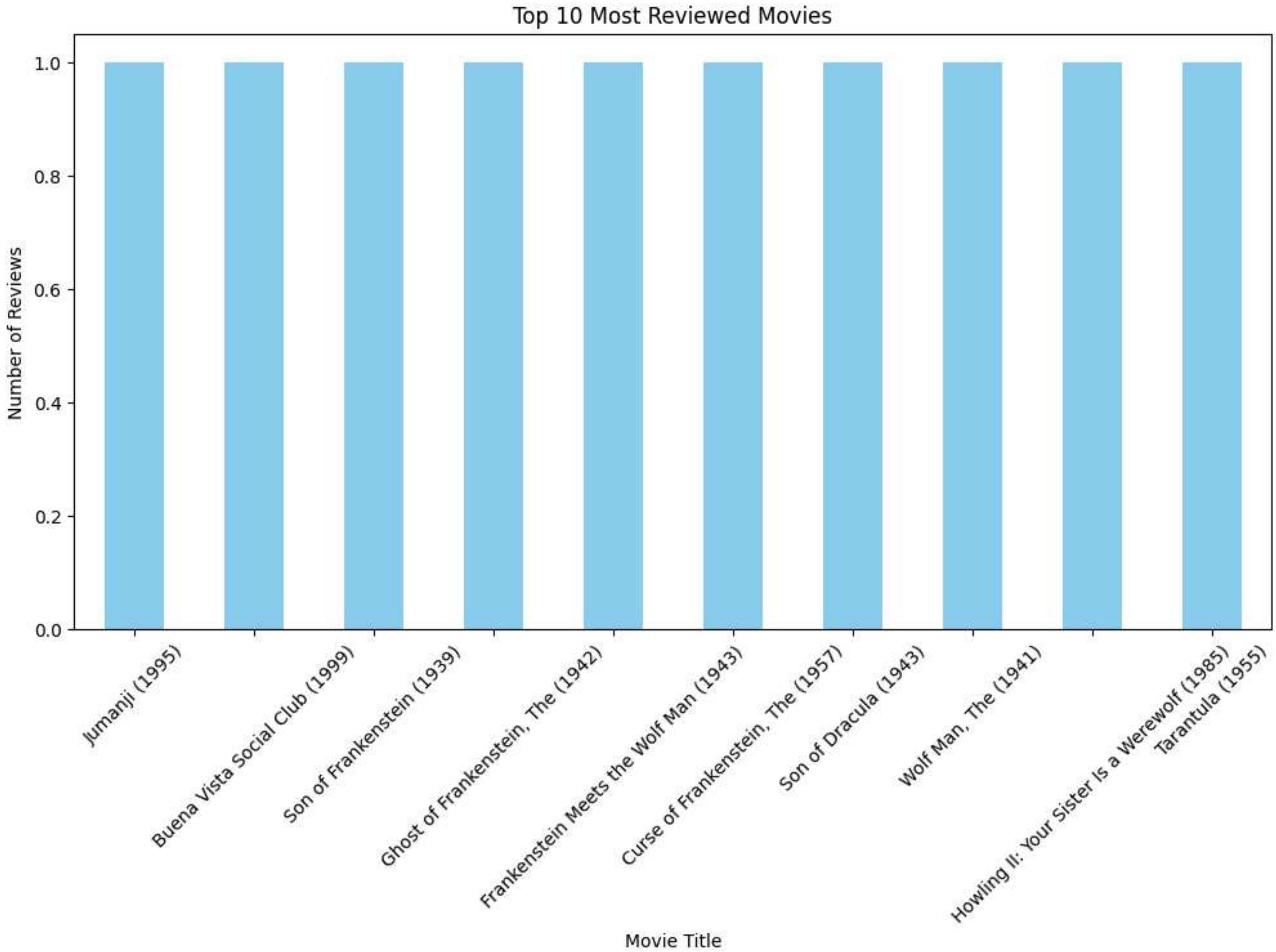



```
sns.countplot(x=df['Gender'],hue=df['Rating'])
```

<Axes: xlabel='Gender', ylabel='count'>

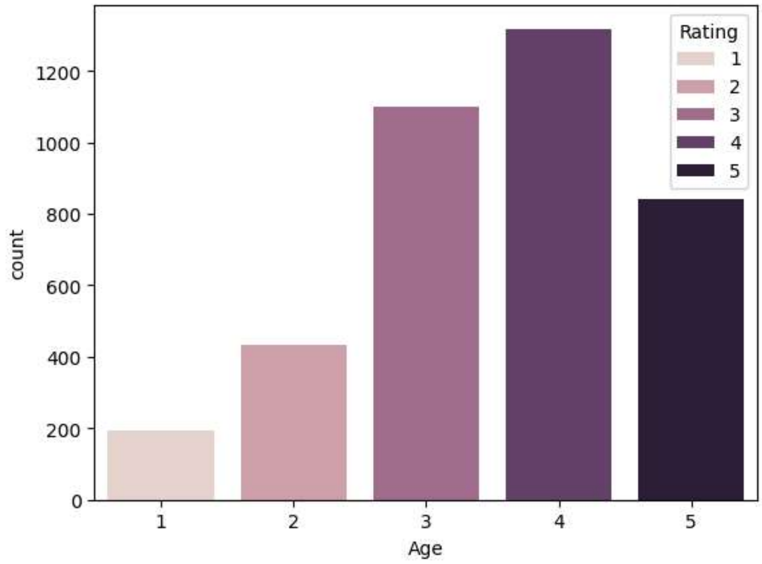



```
# 4. Bar chart for 'Title'
top_titles = df['Title'].value_counts().nlargest(10)
plt.figure(figsize=(12, 6))
top_titles.plot(kind='bar', color='skyblue')
plt.title('Top 10 Most Reviewed Movies')
plt.xlabel('Movie Title')
plt.ylabel('Number of Reviews')
plt.xticks(rotation=45)
plt.show()
```




```
sns.countplot(x=df['Age'],hue=df['Rating'])
```

```
<Axes: xlabel='Age', ylabel='count'>
```



!image.png

```
# Splitting the features and targets

x=df.drop(['Rating','Genres','Title'],axis=1)
y=df['Rating']
```

```
x.head()
```

	Gender	Age
0	639	3
1	853	3
2	3177	4
3	2162	5
4	1107	3

Next steps: Generate code with x View recommended plots New interactive sheet

```
### Importing the dependencies

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

```
### Machine Learning models Libraries:
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold,cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=3)
```

```
print(x.shape,x_train.shape,x_test.shape)
```

```
(3882, 2) (3105, 2) (777, 2)
```

Accuracy Score

```
models = [LogisticRegression(max_iter=1000),DecisionTreeClassifier(),RandomForestClassifier(),KNeighborsClassifier()]
```

```
def compare_models_train_test():
    for model in models:
        model.fit(x_train,y_train)
        y_predicted = model.predict(x_test)
        accuracy = accuracy_score(y_test,y_predicted)
        print("Accuracy of the ",model,"=",accuracy)
        print("="*100)
```

```
compare_models_train_test()
```

```
Accuracy of the  LogisticRegression(max_iter=1000) = 0.9678249678249679
====================================================================================================
Accuracy of the  DecisionTreeClassifier() = 1.0
====================================================================================================
Accuracy of the  RandomForestClassifier() = 1.0
====================================================================================================
Accuracy of the  KNeighborsClassifier() = 0.4594594594594595
====================================================================================================
```

Cross Validation

```
models = [LogisticRegression(max_iter=1000),DecisionTreeClassifier(),RandomForestClassifier(),KNeighborsClassifier()]
```

```
def compare_models_cv():
    for model in models:
        cv_score =cross_val_score(model,x,y,cv=5)
        mean_accuracy = sum(cv_score)/len(cv_score)
        mean_accuracy= mean_accuracy*100
        mean_accuracy = round(mean_accuracy,2)
        print("cv_score of the",model,"=",cv_score)
        print("mean_accuracy % of the",model,"=",mean_accuracy,"%")
        print("="*100)
```

```
compare_models_cv()
```

```
cv_score of the LogisticRegression(max_iter=1000) = [1.         1.         0.94974227 0.95618557 0.9871134 ]
mean_accuracy % of the LogisticRegression(max_iter=1000) = 97.86 %
==================================================================================================
cv_score of the DecisionTreeClassifier() = [1. 1. 1. 1. 1.]
mean_accuracy % of the DecisionTreeClassifier() = 100.0 %
==================================================================================================
cv_score of the RandomForestClassifier() = [1. 1. 1. 1. 1.]
mean_accuracy % of the RandomForestClassifier() = 100.0 %
==================================================================================================
cv_score of the KNeighborsClassifier() = [0.45173745 0.44401544 0.45231959 0.46778351 0.46391753]
mean_accuracy % of the KNeighborsClassifier() = 45.6 %
==================================================================================================
```

```
# Sample data (replace with your actual data)
# X = your feature matrix, y = your target variable
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

models = [
    LogisticRegression(max_iter=1000),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    KNeighborsClassifier()
]

# Hyperparameter grids for each model
param_grids = [
    {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]},
    {'max_depth': [None, 10, 20, 30, 40, 50],
     'min_samples_split': [2, 5, 10],
     'min_samples_leaf': [1, 2, 4]},
    {'n_estimators': [50, 100, 200],
     'max_depth': [None, 10, 20, 30, 40, 50],
     'min_samples_split': [2, 5, 10],
     'min_samples_leaf': [1, 2, 4]},
    {'n_neighbors': [3, 5, 7, 9],
     'weights': ['uniform', 'distance'],
     'metric': ['euclidean', 'manhattan']}
]

best_models = []

for i, model in enumerate(models):
    grid_search = GridSearchCV(model, param_grids[i], cv=5, scoring='accuracy')
    grid_search.fit(x_train, y_train)

    best_model = grid_search.best_estimator_
    best_models.append(best_model)

    print(f"Best hyperparameters for {type(model).__name__}: {grid_search.best_params_}")
    print(f"Best cross-validated accuracy: {grid_search.best_score_:.4f}")

    y_pred = best_model.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Test accuracy for {type(model).__name__}: {accuracy:.4f}\n")

# You can now use best_models for further analysis or predictions.
```

```
Best hyperparameters for LogisticRegression: {'C': 10}
Best cross-validated accuracy: 0.9903
Test accuracy for LogisticRegression: 1.0000

Best hyperparameters for DecisionTreeClassifier: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best cross-validated accuracy: 1.0000
Test accuracy for DecisionTreeClassifier: 1.0000

Best hyperparameters for RandomForestClassifier: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
Best cross-validated accuracy: 1.0000
Test accuracy for RandomForestClassifier: 1.0000

Best hyperparameters for KNeighborsClassifier: {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
Best cross-validated accuracy: 0.6264
Test accuracy for KNeighborsClassifier: 0.6680
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

tuned_results = []
```

```
for idx, model in enumerate(best_models):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Specify average='micro' for multiclass classification
    precision = precision_score(y_test, y_pred, average='micro')
    recall = recall_score(y_test, y_pred, average='micro')
    f1 = f1_score(y_test, y_pred, average='micro')

    # Specify either 'ovo' (one-vs-one) or 'ovr' (one-vs-rest) for multi_class
    roc_auc = roc_auc_score(y_test, model.predict_proba(x_test), multi_class='ovr')

    tuned_results.append([f'Model_{idx}', accuracy, precision, recall, f1, roc_auc])


columns = ['Models', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC AUC']


# Step 8: Compare Tuned Models
tuned_results_df = pd.DataFrame(tuned_results, columns=columns)

print(tuned_results_df)
```

```
    Models  Accuracy  Precision    Recall  F1 Score   ROC AUC
0  Model_0  1.000000   1.000000  1.000000  1.000000  1.000000
1  Model_1  1.000000   1.000000  1.000000  1.000000  1.000000
2  Model_2  1.000000   1.000000  1.000000  1.000000  1.000000
3  Model_3  0.667954   0.667954  0.667954  0.667954  0.808534
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.42      0.19      0.26        26
           2       0.60      0.32      0.42       101
           3       0.64      0.68      0.66       222
           4       0.65      0.80      0.72       276
           5       0.82      0.71      0.76       152

    accuracy                           0.67       777
   macro avg       0.63      0.54      0.56       777
weighted avg       0.67      0.67      0.66       777
```