Movielens Case Study

Abstract

The GroupLens Research Project is a research group in the Department of Computer Science and Engineering at the University of Minnesota. Members of the GroupLens Research Project are involved in many research projects related to the fields of information filtering, collaborative filtering, and recommender systems.

Presented by: Ankita Agarwal

Problem Statement:

The GroupLens Research Project is a research group in the Department of Computer Science and Engineering at the University of Minnesota. Members of the GroupLens Research Project are involved in many research projects related to the fields of information filtering, collaborative filtering, and recommender systems. The project is led by professors John Riedl and Joseph Konstan. The project began to explore automated collaborative filtering in 1992 but is most well-known for its worldwide trial of an automated collaborative filtering system for Usenet news in 1996. Since then the project has expanded its scope to research overall information by filtering solutions, integrating into content-based methods, as well as, improving current collaborative filtering technology.

Problem Objective: Here, we ask you to perform the analysis using the Exploratory Data Analysis technique. You need to find features affecting the ratings of any particular movie and build a model to predict the movie ratings.

Domain: Entertainment

Detailed description of the given dataset:

These files contain 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000.

Ratings.dat
Format - UserID::MovieID::Rating::Timestamp

Field	**Description**
UserID	Unique identification for each user
MovieID	Unique identification for each movie
Rating	User rating for each movie
Timestamp	Timestamp generated while adding user review

- UserIDs range between 1 and 6040
- The MovieIDs range between 1 and 3952
- Ratings are made on a 5-star scale (whole-star ratings only)
- A timestamp is represented in seconds since the epoch is returned by time(2)
- Each user has at least 20 ratings

Users.dat
Format - UserID::Gender::Age::Occupation::Zip-code

Field	Description
UserID	Unique identification for each user
Genere	Category of each movie

Age	User’s age
Occupation	User’s Occupation
Zip-code	Zip Code for the user’s location

All demographic information is provided voluntarily by the users and is not checked for accuracy. Only users who have provided demographic information are included in this data set.

- Gender is denoted by an "M" for male and "F" for female
- Age is chosen from the following ranges:

Value	**Description**
1	"Under 18"
18	"18-24"
25	"25-34"
35	"35-44"
45	"45-49"
50	"50-55"
56	"56+"

- Occupation is chosen from the following choices:

Value	**Description**
0	"other" or not specified
1	"academic/educator"
2	"artist”
3	"clerical/admin"
4	"college/grad student"
5	"customer service"
6	"doctor/health care"
7	"executive/managerial"
8	"farmer"
9	"homemaker"
10	"K-12 student"
11	"lawyer"
12	"programmer"
13	"retired"
14	"sales/marketing"
15	"scientist"
16	"self-employed"
17	"technician/engineer"
18	"tradesman/craftsman"
19	"unemployed"
20	"writer”

Movies.dat
Format - MovieID::Title::Genres

Field	Description
MovieID	Unique identification for each movie
Title	A title for each movie
Genres	Category of each movie

- Titles are identical to titles provided by the IMDB (including year of release)
- Genres are pipe-separated and are selected from the following genres:

1. Action
2. Adventure
3. Animation
4. Children's
5. Comedy
6. Crime
7. Documentary
8. Drama
9. Fantasy
10. Film-Noir
11. Horror
12. Musical
13. Mystery
14. Romance
15. Sci-Fi
16. Thriller
17. War
18. Western

- Some MovieIDs do not correspond to a movie due to accidental duplicate entries and/or test entries
- Movies are mostly entered by hand, so errors and inconsistencies may exist

To Analyze:

Exploratory Data Analysis:

Import the three datasets

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)

Explore the datasets using visual representations (graphs or tables), also include your comments on the following:

1. User Age Distribution
2. User rating of the movie “Toy Story”
3. Top 25 movies by viewership rating
4. Find the ratings for all the movies reviewed by for a particular user of user id = 2696

Feature Engineering:

Use column genres:

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)
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.
3. Determine the features affecting the ratings of any particular movie.
4. Develop an appropriate model to predict the movie ratings

Analysis and Interpretations:

Exploratory Data Analysis: -

Import the three datasets. 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)

```
#merging the three datasets to create Master_Data

temp_data= pd.merge(ratings,users, on= 'UserID', how='left')
master_data = pd.merge(temp_data,movies, on= 'MovieID',how='left')
```

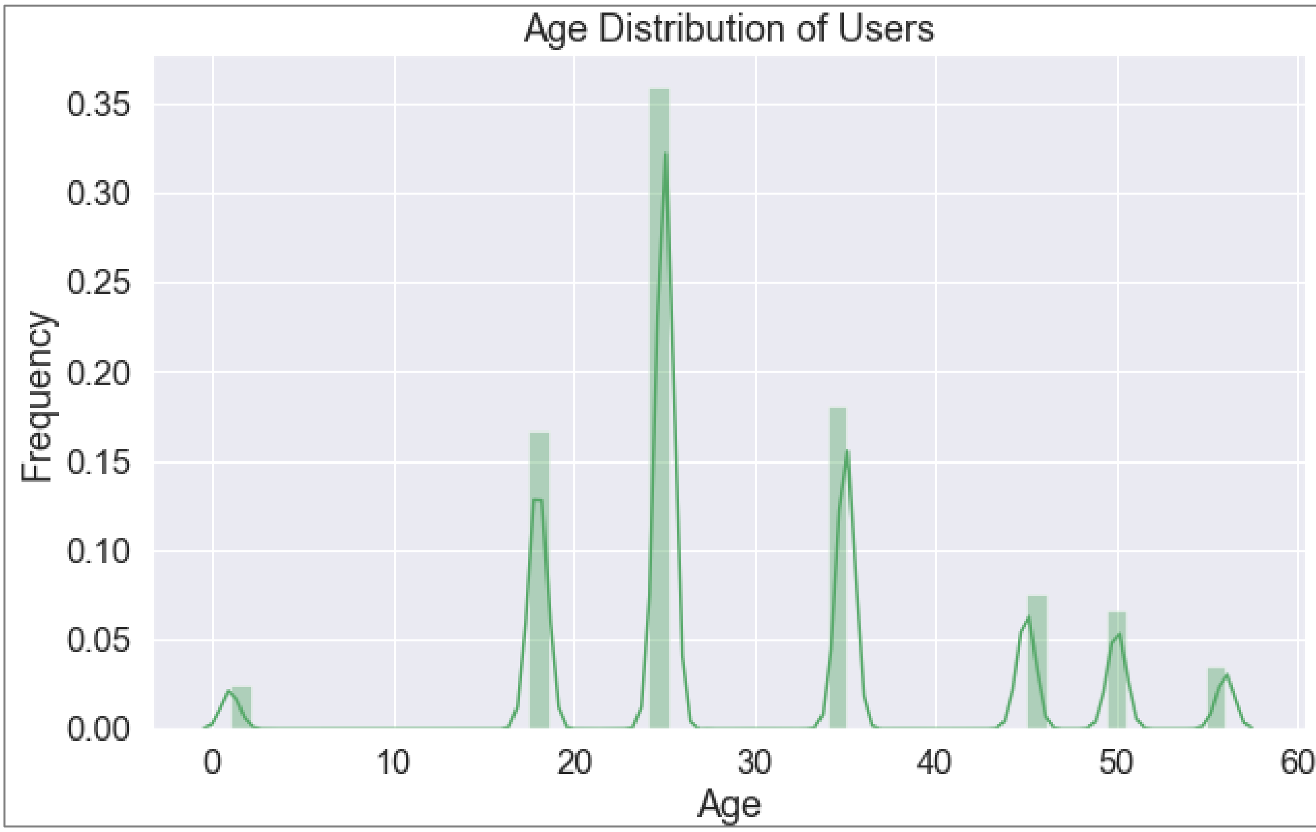
```
print(master_data.head(),'\n\n')
print(master_data.info())
```

```
   UserID  MovieID  Rating  Timestamp Gender  Age  Occupation Zip-Code  \
0       1     1193       5  978300760      F    1          10    48067
1       1      661       3  978302109      F    1          10    48067
2       1      914       3  978301968      F    1          10    48067
3       1     3408       4  978300275      F    1          10    48067
4       1     2355       5  978824291      F    1          10    48067

                                    Title                        Genres
0  One Flew Over the Cuckoo's Nest (1975)                         Drama
1        James and the Giant Peach (1996)  Animation|Children's|Musical
2                     My Fair Lady (1964)               Musical|Romance
3                  Erin Brockovich (2000)                         Drama
4                    Bug's Life, A (1998)   Animation|Children's|Comedy


<class 'pandas.core.frame.DataFrame'>
```

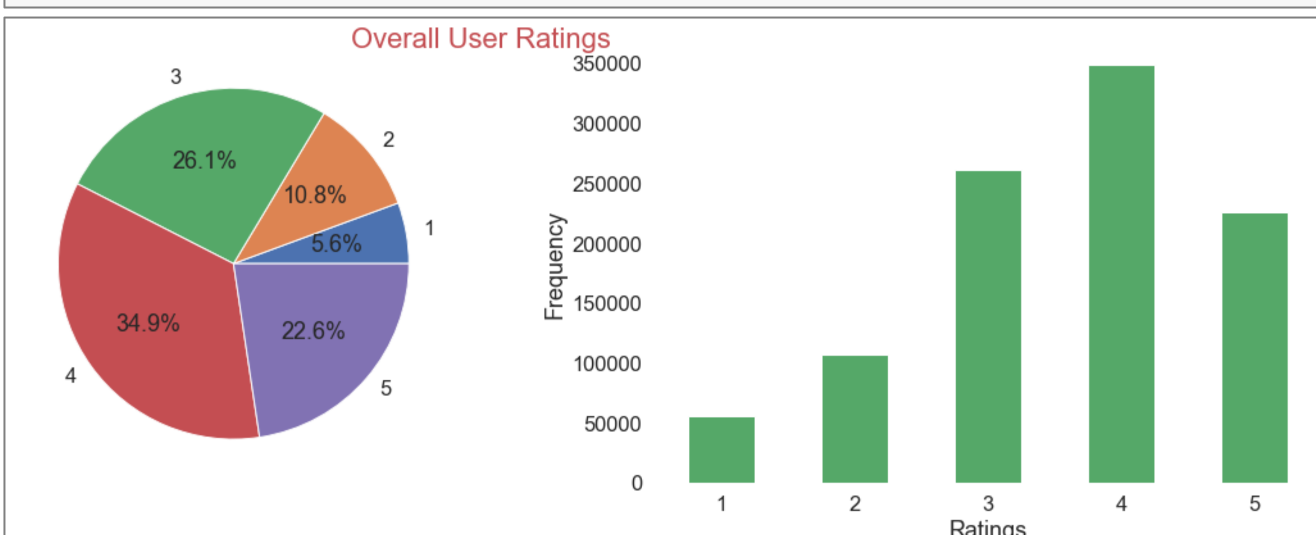
Explore the datasets using visual representations (graphs or tables), also include your comments on the following:

1. User Age Distribution

```
#Explore the datasets using visual representations (graphs or tables), also include your comments on the following:
#1. User Age Distribution

plt.figure(figsize=[10,6])
sns.distplot(master_data.Age,color='g')
plt.title('Age Distribution of Users')
plt.ylabel('Frequency')
plt.xlabel('Age');
```




```
#Overall ratings by users

fig,ax= plt.subplots(1,2,figsize=[16,6])
pd.value_counts(master_data.Rating).sort_index().plot.pie(autopct='%.1f%%',ax=ax[0],labels=[1,2,3,4,5])
ax[0].set(ylabel= '')

pd.value_counts(master_data.Rating).sort_index().plot(kind='bar',color='g',rot=0,ax=ax[1])
ax[1].set(xlabel='Ratings', ylabel='Frequency')
plt.tight_layout(pad=1.2)
plt.box(False)
fig.suptitle('Overall User Ratings',ha='right',color='r',fontsize=22);
```

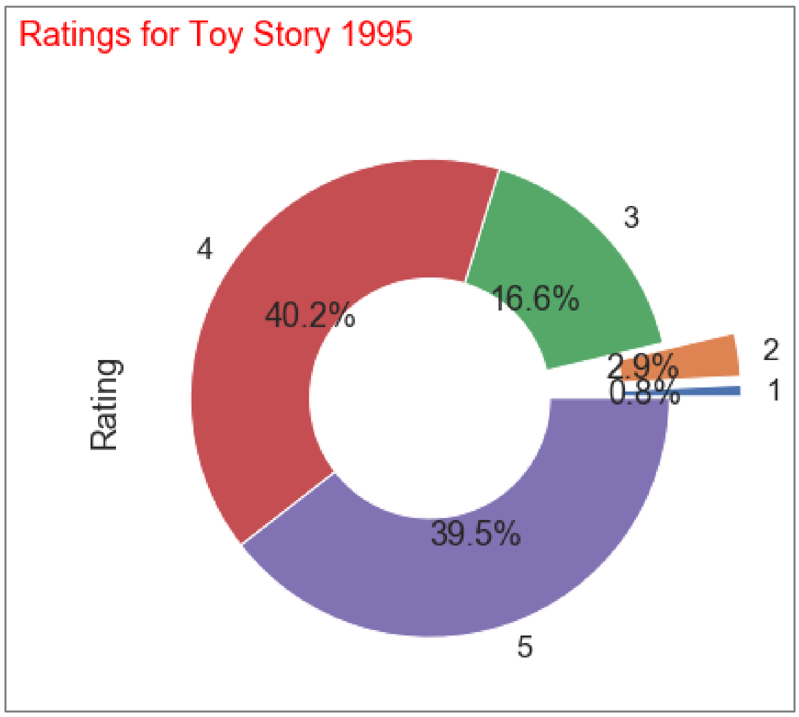


Interpretation:

We can clearly see that most users are 20-30years old and have mostly rated 4 for the movies in the dataset.

2. User rating of the movie “Toy Story”

```
#2. User rating of the movie "Toy Story"

plt.figure(figsize=[10,6])

Toy_Story_Ratings = master_data[master_data.MovieID.isin([1])].Rating.value_counts()

plt.suptitle('Ratings for Toy Story 1995', fontsize=18, color='red', ha='right')
Toy_Story_Ratings.sort_index().plot(kind='pie',autopct='%.1f%%',labels=[1,2,3,4,5],explode=(0.3,0.3,0,0,0),wedgeprops=dict(wi
plt.rcParams['font.size'] = 10
```

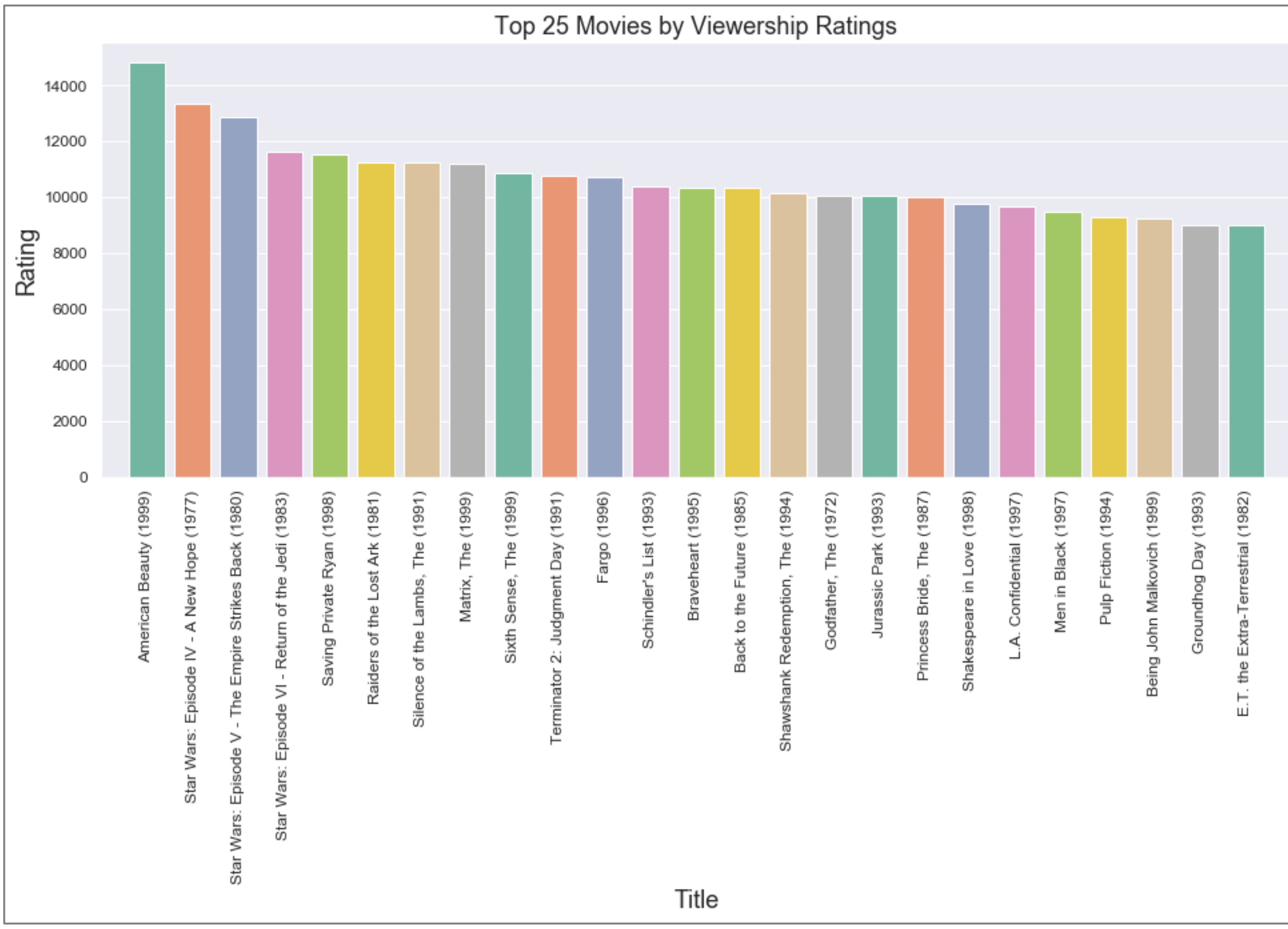


Interpretation:

Most users have rated Toy Story, 1995 with a rating of 4.

3. Top 25 movies by viewership rating

```
#3. Top 25 movies by viewership rating

top_25_movies_by_user_ratings = \
master_data.groupby('Title').agg({\
                                  'Rating':'sum'\
                                 }).sort_values(by='Rating',ascending=False).reset_index()

plt.figure(figsize=[16,6])
sns.barplot('Title','Rating',data=top_25_movies_by_user_ratings[:25],palette='Set2')
plt.xticks(rotation=90, size = 12)
plt.yticks(size = 12)
plt.title('Top 25 Movies by Viewership Ratings')
plt.xlim((-1,25));
```

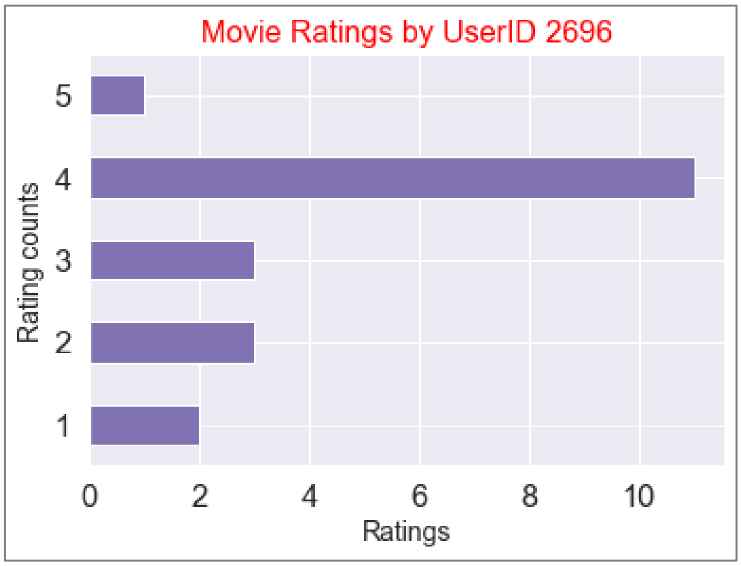


Interpretation:

Top 25 movies can be seen in the graph above with American Beauty 1999 taking the first spot.

4. Find the ratings for all the movies reviewed by for a particular user of user id = 2696

```
#4. Find the ratings for all the movies reviewed by for a particular user of user id = 2696

rating_by_user_2696 = master_data[master_data.UserID == 2696][['UserID','Rating']].reset_index(drop=True)

rating_by_user_2696.Rating.value_counts().sort_index().plot(kind='barh',rot=0,color='m')
plt.title('Movie Ratings by UserID 2696', size=16, color='red')
plt.xlabel('Ratings', size=14)
plt.ylabel('Rating counts', size=14)
```



Interpretation:

UserID 2696 have rated most movies a 4.

Feature Engineering: -

Use column genres:

5. **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)**

```
#Feature Engineering:

#Use column genres:

#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)

master_data['Genres'] = master_data.Genres.apply(lambda x: x.split('|'))
```

```
%%time
genre_list= []
for genres in master_data.Genres.values:
    for genre in genres:
        genre_list.append(genre)
```

```
Wall time: 928 ms
```

```
unique_genre = set(genre_list)
print('The unique Genres are: [{}]'.format(unique_genre))
```

```
The unique Genres are: [{"Children's", 'Drama', 'Fantasy', 'Mystery', 'Musical', 'Documentary', 'Romance', 'Animation', 'We
stern', 'Thriller', 'Sci-Fi', 'Action', 'War', 'Crime', 'Film-Noir', 'Adventure', 'Horror', 'Comedy'}]
```

6. 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.

```
#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.
```

```
%%time
for genre in unique_genre:
    master_data[genre] = master_data.Genres.apply(lambda x: 1 if genre in x else 0)
```

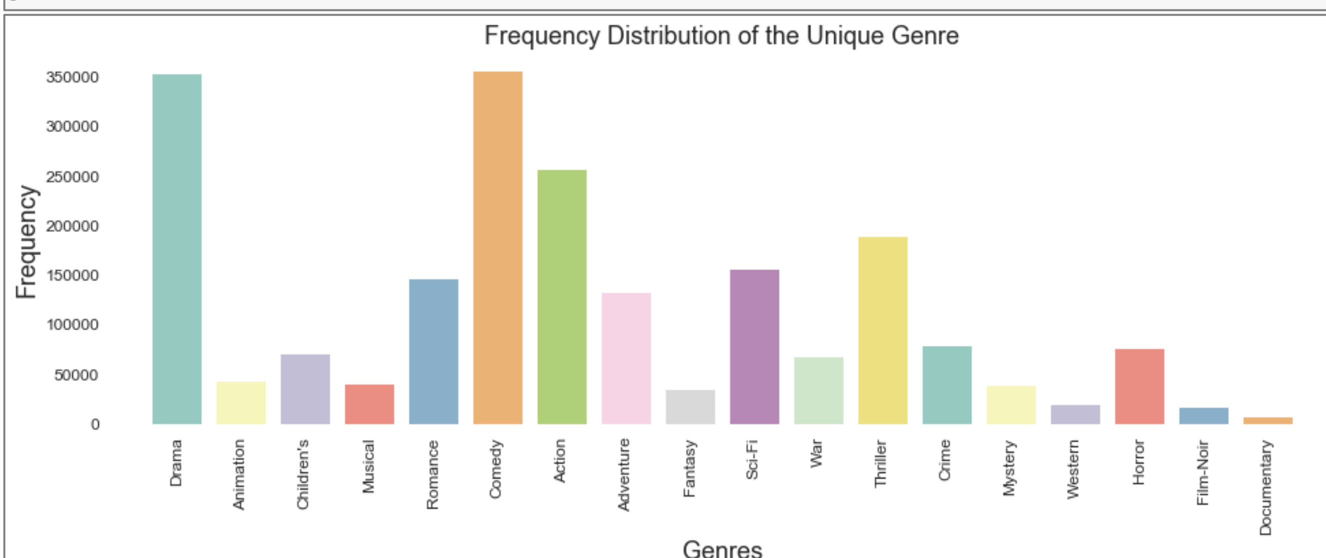
```
Wall time: 21.3 s
```

```
master_data.head(2)
```

	UserID	MovieID	Rating	Timestamp	Gender	Age	Occupation	Zip-Code	Title	Genres	...	Western	Thriller	Sci-Fi	Action	War	Crime	Film-Noir	Adventur
0	1	1193	5	978300760	F	1	10	48067	One Flew Over the Cuckoo's Nest (1975)	[Drama]	...	0	0	0	0	0	0	0	
1	1	661	3	978302109	F	1	10	48067	James and the Giant Peach (1996)	[Animation, Children's, Musical]	...	0	0	0	0	0	0	0	

2 rows × 28 columns

```
genres = pd.Series(np.array(genre_list)).to_frame(name='Genres')
plt.figure(figsize=[16,5])
sns.countplot('Genres',data=genres,palette='Set3')
plt.xticks(rotation=90, size = 12)
plt.yticks(size=12)
plt.xlim(-1,18)
plt.title('Frequency Distribution of the Unique Genre')
plt.ylabel('Frequency');
plt.box(False)
```



Interpretation:

Most movies are either of Drama or Comedy genre.

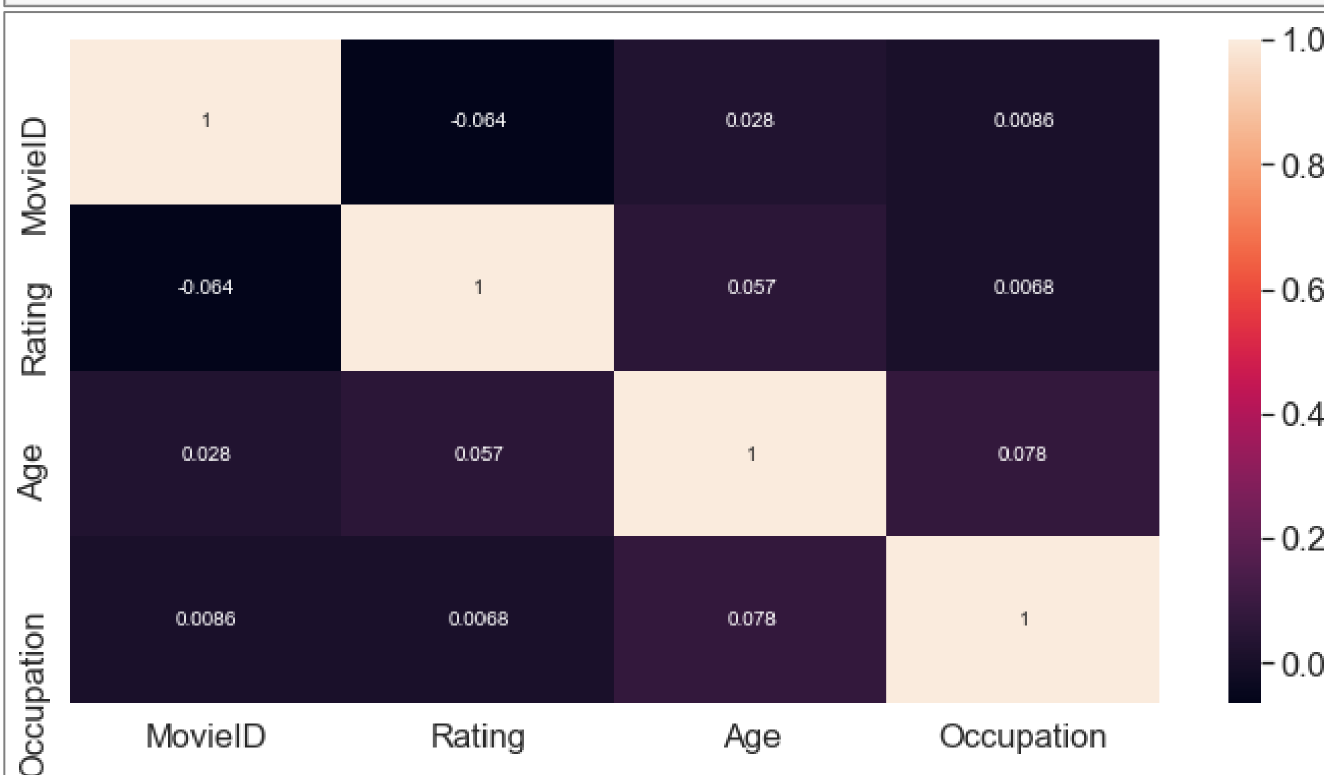
7. Determine the features affecting the ratings of any particular movie.

```
#3. Determine the features affecting the ratings of any particular movie. - will use MovieID, Age and Occupation

from scipy.stats import chi2_contingency

ctTitle = pd.crosstab(master_data.Title,master_data.Rating)
ctGender = pd.crosstab(master_data.Gender,master_data.Rating)
ctAge = pd.crosstab(master_data.Age,master_data.Rating)
ctOccupation = pd.crosstab(master_data.Occupation,master_data.Rating)
ctZipCode = pd.crosstab(master_data['Zip-Code'],master_data.Rating)
```

```
from scipy.stats import chi2_contingency

list1 = [ctTitle,ctGender,ctAge,ctOccupation,ctZipCode]

for i in list1:
    stat,pvalue,dof,expected_R = chi2_contingency(i)
    if pvalue <= 0.05:
        print("Alternate Hypothesis passed. {} and Rating have Relationship; pvalue = {:.5e}".format(i.index.name,pvalue))
    else:
        print("Null hypothesis passed. {} and Rating doesnot have  Relationship".format(i.index.name))
```

```
Alternate Hypothesis passed. Title and Rating have Relationship; pvalue = 0.00000e+00
Alternate Hypothesis passed. Gender and Rating have Relationship; pvalue = 2.34856e-97
Alternate Hypothesis passed. Age and Rating have Relationship; pvalue = 0.00000e+00
Alternate Hypothesis passed. Occupation and Rating have Relationship; pvalue = 0.00000e+00
Alternate Hypothesis passed. Zip-Code and Rating have Relationship; pvalue = 0.00000e+00
```

```
df = master_data[['MovieID','Rating', 'Gender','Age','Occupation', 'Zip-Code']]


plt.figure(figsize=(12,6))
corr=df.corr()
sns.heatmap(corr,xticklabels=corr.columns.values,yticklabels=corr.columns.values,annot=True,annot_kws={'size':10})
```



Interpretation:

As the P-value is very small as compared to alpha 0.05, we can determine that Title, MovieID, Age, Occupation, Age, Gender, Zip-Code all are correlated to Rating of the movies of the dataset.

8. Develop an appropriate model to predict the movie ratings

I have developed a few models such as Linear Regression, Logistic Regression, XGBoost, KNeighbors, Gaussian NB, Decision Tree, Random Forest, Linear Discriminant Analysis.

```
#4. Develop an appropriate model to predict the movie ratings - Will use the above information to do this

feature_cols = ['MovieID',
                'Age',
                'Occupation']

response_col = ['Rating']

X = master_data[feature_cols].values
y = master_data[response_col].values.ravel()

# Split into train and test sets.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state =0)
```

```
# Linear Regression

linreg=LinearRegression()
linreg.fit(X_train, y_train)

# make predictions on the testing set
y_pred = linreg.predict(X_test)

# compute the RMSE of our predictions

print(np.sqrt(mean_squared_error(y_test, y_pred)))
print(linreg.score(X_test, y_test))
```

```
1.1102472527032017
0.007323560847342536
```

```
#KNN

knn = KNeighborsClassifier(n_neighbors = 8).fit(X_train, y_train)
knn_predictions = knn.predict(X_test)

# accuracy on X_test
accuracy = knn.score(X_test, y_test)

# creating a confusion matrix
#cm = confusion_matrix(y_test, knn_predictions)

accuracy
```

```
0.3458223773007668
```

```
#Naive Bayes classifier

GN = GaussianNB().fit(X_train, y_train)
GN_predictions = GN.predict(X_test)

# accuracy on X_test
accuracy = GN.score(X_test, y_test)

# creating a confusion matrix
#cm = confusion_matrix(y_test, GN_predictions)

accuracy
```

```
0.3479769248457824
```

```python
#Random Forest

rf = RandomForestClassifier(n_estimators=200).fit(X_train,y_train)
rf_predictions = rf.predict(X_test)

# accuracy on X_test
accuracy = rf.score(X_test, y_test)

# creating a confusion matrix
#cm = confusion_matrix(y_test, rf_predictions)

accuracy
```

```
0.35666010137871046
```

```python
#XGBoost

xgb = XGBRFClassifier(n_estimators=200).fit(X_train,y_train)
xgb_predictions = xgb.predict(X_test)

# accuracy on X_test
accuracy = xgb.score(X_test, y_test)

# creating a confusion matrix
#cm = confusion_matrix(y_test, xgb_predictions)

accuracy
```

```
0.35855470351226243
```

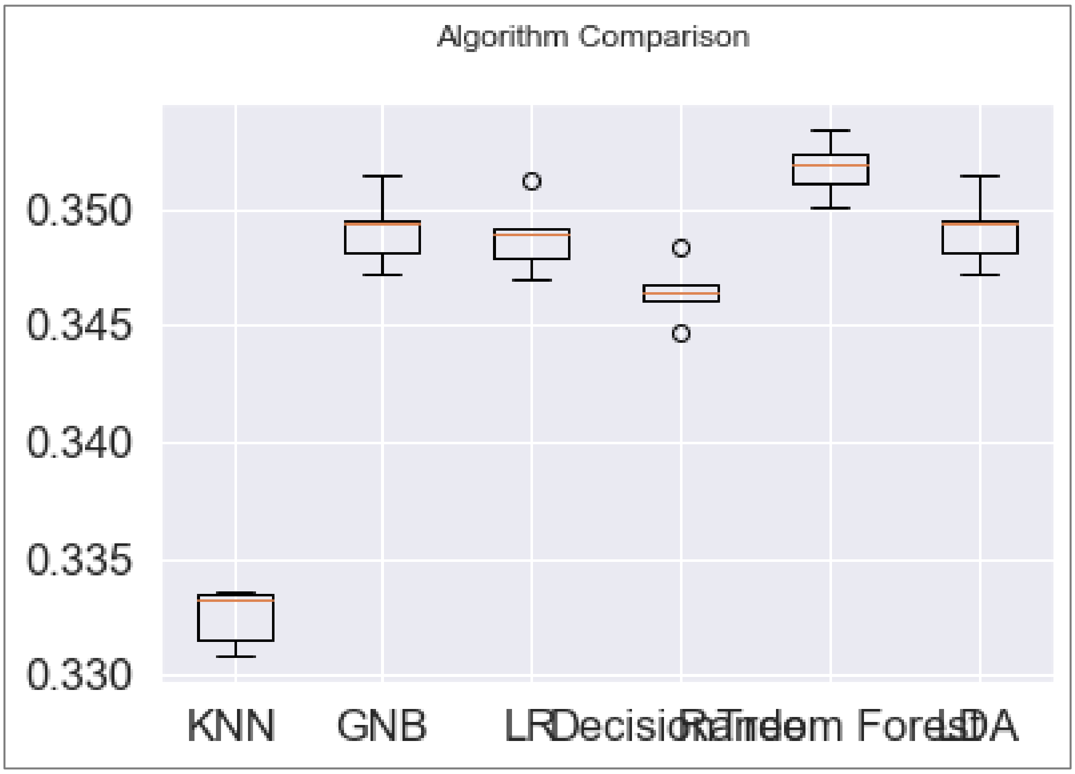
```python
# Create objects of required models.
models = []
models.append(("KNN",           KNeighborsClassifier()))
models.append(("GNB",           GaussianNB()))
models.append(("LR",            LogisticRegression()))
models.append(("DecisionTree",  DecisionTreeClassifier()))
models.append(("Random Forest", RandomForestClassifier()))
models.append(('LDA',           LinearDiscriminantAnalysis()))
```

```python
# Find accuracy of models.
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=5, random_state=0)

    cv_result = cross_val_score(model, X_train, y_train, cv = kfold, scoring = "accuracy")

    results.append(tuple([name, cv_result.mean(), cv_result.std()]))
```

```python
results.sort(key=lambda x: x[1], reverse = True)

for i in range(len(results)):
    print('{:20s} {:2.2f} (+/-) {:2.2f} '.format(results[i][0] , results[i][1] * 100, results[i][2] * 100))
```

```
Random Forest        35.20 (+/-) 0.12
GNB                  34.91 (+/-) 0.15
LDA                  34.91 (+/-) 0.15
LR                   34.88 (+/-) 0.14
DecisionTree         34.66 (+/-) 0.13
KNN                  33.25 (+/-) 0.11
```

```python
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results1)
ax.set_xticklabels(names)
plt.show()
```



Interpretation:

From above we can safely say that XGBoost or Random Forest are the best models recommended for prediction of movie ratings.

Programming Codes:

Adobe Acrobat Document

---The End---