Amazon Movie & TV Ratings

Data Dictionary

UserID – 4848 customers who provided a rating for each movie

Movie 1 to Movie 206 – 206 movies for which ratings are provided by 4848 distinct users

Data Considerations

- All the users have not watched all the movies and therefore, all movies are not rated. These missing values are represented by NA.
- Ratings are on a scale of -1 to 10 where -1 is the least rating and 10 is the best.

Analysis Task

Exploratory Data Analysis:

- Which movies have maximum views/ratings?
- What is the average rating for each movie? Define the top 5 movies with the maximum ratings.
- Define the top 5 movies with the least audience.

Recommendation Model:

Some of the movies hadn't been watched and therefore, are not rated by the users. Netflix would like to take this as an opportunity and build a machine learning recommendation algorithm which provides the ratings for each of the users.

- Divide the data into training and test data
- Build a recommendation model on training data
- Make predictions on the test data

In [1]:
```
import pandas as pd
```

In [2]:
```
data = pd.read_csv('~/MyWorkPython/data/Amazon_Movies_and_TV_Ratings.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movi
0	A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	NaN	N.
1	AH3QC2PC1VTGP	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	N.
2	A3LKP6WPMP9UKX	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	N.
3	AVIY68KEPQ5ZD	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	N.
4	A1CV1WROP5KTTW	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	N.

5 rows × 207 columns

In [4]:
```
data.describe()
```

Out[4]:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	.
count	1.0	1.0	1.0	2.0	29.000000	1.0	1.0	1.0	1.0	1.0	.
mean	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	.
std	NaN	NaN	NaN	0.0	1.496301	NaN	NaN	NaN	NaN	NaN	.
min	5.0	5.0	2.0	5.0	1.000000	4.0	5.0	5.0	5.0	5.0	.
25%	5.0	5.0	2.0	5.0	4.000000	4.0	5.0	5.0	5.0	5.0	.
50%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	.
75%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	.
max	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	.

8 rows × 206 columns

Q1 Which movies have maximum views/ratings?

Logic 1 - getting the count using the describe

In [5]:
```
data.describe().T["count"].sort_values(ascending = False)[:10].to_frame()
```

Out[5]:

	count
Movie127	2313.0
Movie140	578.0
Movie16	320.0
Movie103	272.0
Movie29	243.0
Movie91	128.0
Movie92	101.0
Movie89	83.0
Movie158	66.0
Movie108	54.0

Logic 2 based on the movie column data, we will get the Movie Name, Rating using Mean and No. Of Users

In [6]:
```
data_logic2 = pd.DataFrame(columns=['MovieId','MovieRating','NoOfUserRating'])

for id, value in enumerate(data.iloc[:,1:].columns):
    data_logic2.loc[id,'MovieId'] = value
    data_logic2.loc[id,'MovieRating'] = round(data.loc[:,value].mean(),2)
    data_logic2.loc[id,'NoOfUserRating'] = data.loc[:,value].notnull().sum()

# now we have a new data with the Movie Id, Movie Rating and No Of User Rating
data_logic2.sort_values(by=['NoOfUserRating','MovieRating'], ascending=False)
[:10]
```

Out[6]:

	MovieId	MovieRating	NoOfUserRating
126	Movie127	4.11	2313
139	Movie140	4.83	578
15	Movie16	4.52	320
102	Movie103	4.56	272
28	Movie29	4.81	243
90	Movie91	4.58	128
91	Movie92	4.77	101
88	Movie89	4.58	83
157	Movie158	4.82	66
107	Movie108	4.67	54

Q2 What is the average rating for each movie? Define the top 5 movies with the maximum ratings.

Logic 1

In [7]:
```
data.describe().T["count"].sort_values(ascending = False)[:5].to_frame()
```

Out[7]:

	count
Movie127	2313.0
Movie140	578.0
Movie16	320.0
Movie103	272.0
Movie29	243.0

Logic 2

In [8]:
```
data_logic2.sort_values(by=['NoOfUserRating','MovieRating'], ascending=False).
head()
```

Out[8]:

	MovieId	MovieRating	NoOfUserRating
126	Movie127	4.11	2313
139	Movie140	4.83	578
15	Movie16	4.52	320
102	Movie103	4.56	272
28	Movie29	4.81	243

Q3 Define the top 5 movies with the least audience.

Logic 1

In [9]:
```python
data.describe().T["count"].sort_values(ascending = True)[:5].to_frame()
```

Out[9]:

	count
Movie1	1.0
Movie71	1.0
Movie145	1.0
Movie69	1.0
Movie68	1.0

Logic 2

In [10]:
```python
data_logic2.sort_values(by=['NoOfUserRating']).head()
```

Out[10]:

	MovieId	MovieRating	NoOfUserRating
0	Movie1	5	1
70	Movie71	4	1
144	Movie145	5	1
68	Movie69	1	1
67	Movie68	5	1

Recommendation Model:

Some of the movies hadn't been watched and therefore, are not rated by the users. Netflix would like to take this as an opportunity and build a machine learning recommendation algorithm which provides the ratings for each of the users.

- Divide the data into training and test data
- Build a recommendation model on training data
- Make predictions on the test data

Logic 1 using Surprise Python Scikit Building and Analyzing Recommender Systems

In [11]:
```python
# pip install surprise
# or
# conda install -c conda-forge scikit-surprise
# conda install -c conda-forge/label/gcc7 scikit-surprise
# conda install -c conda-forge/label/cf201901 scikit-surprise
```

In [12]:
```python
from surprise import Reader, accuracy, Dataset
from surprise.model_selection import train_test_split
```

In [33]:
```python
data.head()
```

Out[33]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movi
0	A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	NaN	N
1	AH3QC2PC1VTGP	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	N
2	A3LKP6WPMP9UKX	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	N
3	AVIY68KEPQ5ZD	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	N
4	A1CV1WROP5KTTW	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	N

5 rows × 207 columns

In [34]:
```python
#now we can use the melt function to setup the data

melt_data = data.melt(id_vars = data.columns[0], value_vars= data.columns[1:],
var_name="Movie", value_name="Rating")

melt_data.head()
```

Out[34]:

	user_id	Movie	Rating
0	A3R5OBKS7OM2IR	Movie1	5.0
1	AH3QC2PC1VTGP	Movie1	NaN
2	A3LKP6WPMP9UKX	Movie1	NaN
3	AVIY68KEPQ5ZD	Movie1	NaN
4	A1CV1WROP5KTTW	Movie1	NaN

In [35]:
```python
data.describe().T.head()
```

Out[35]:

	count	mean	std	min	25%	50%	75%	max
Movie1	1.0	5.000000	NaN	5.0	5.0	5.0	5.0	5.0
Movie2	1.0	5.000000	NaN	5.0	5.0	5.0	5.0	5.0
Movie3	1.0	2.000000	NaN	2.0	2.0	2.0	2.0	2.0
Movie4	2.0	5.000000	0.000000	5.0	5.0	5.0	5.0	5.0
Movie5	29.0	4.103448	1.496301	1.0	4.0	5.0	5.0	5.0

In [16]:
```
# set up the rating scale as per the description

ratingReader = Reader(rating_scale=(-1,10))

final_data = Dataset.load_from_df(melt_data.fillna(0), reader=ratingReader)

final_data
```

Out[16]:
```
<surprise.dataset.DatasetAutoFolds at 0x276f135eb88>
```

Q1 set up the train and test dataset, we have setup the test size 0.50

In [17]:
```
trainset, testset = train_test_split(final_data, test_size=0.50)
```

In [18]:
```
# we can use the SVD prediction algo
from surprise import SVD, evaluate
```

Q2 Build a recommendation model on training data

In [19]:
```
# set the SVD predication Algo
algo = SVD()

#The evaluate() method is deprecated. Please use model_selection.cross_validat
e() instead. 'model_selection.cross_validate() instead.', UserWarning)
# evaluate(algo, data, measures=['RMSE', 'MAE'])
```

In [20]:
```
# fit the trainSet
algo.fit(trainset)
```

Out[20]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x27684348308>
```

In [47]:
```
# test the algo
predictions = algo.test(testset)
```

Q3 Make predictions on the test data

So in predict case the estimate was a score of 0. But in order to recommend the best rating to users, we need to find n items that have the highest predicted score.

In [45]:
```python
# let make a predication

userID = 'APUVMJA6Q7YZP'
MovieName = 'Movie1'
MovieRating = 0.0

pred = algo.predict(userID, MovieName, r_ui=MovieRating, verbose=True)

print(pred.est)
```

```
user: APUVMJA6Q7YZP item: Movie1     r_ui = 0.00   est = 0.04   {'was_impossi
ble': False}
0.04134679368893357
```

In [46]:
```python
userID = 'A1IMQ9WMFYKWH5'
MovieName = 'Movie127'
MovieRating = 4.5

print('User ', userID, algo.predict(userID, MovieName, r_ui=MovieRating, verbo
se=True).est)

userID = 'A3R5OBKS7OM2IR'

print('User ', userID, algo.predict(userID, MovieName, r_ui=MovieRating, verbo
se=True).est)
```

```
user: A1IMQ9WMFYKWH5 item: Movie127   r_ui = 4.50   est = 0.43   {'was_imposs
ible': False}
User  A1IMQ9WMFYKWH5 0.4289237183017107
user: A3R5OBKS7OM2IR item: Movie127   r_ui = 4.50   est = 0.33   {'was_imposs
ible': False}
User  A3R5OBKS7OM2IR 0.3292819847646751
```

In [22]:
```python
# calculate the accuracy using RMSE
accuracy.rmse(predictions)
```

```
RMSE: 0.2890
```

Out[22]:
```
0.28896912211758213
```

Repeat it for test_size 0.25

In [41]:
```python
# we can repeat trainset and testset for test size 0.25
trainset, testset = train_test_split(final_data, test_size=0.25)
# fit the trainSet
algo.fit(trainset)
```

Out[41]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x27684348308>
```

In [24]:
```python
# test the algo
predictions = algo.test(testset)
```

In [25]:
```python
# calculate the accuracy using RMSE
accuracy.rmse(predictions)
```

```
RMSE: 0.2895
```

Out[25]:
```
0.28947222967443725
```

In [29]:
```python
# evaluate model
from surprise.model_selection import cross_validate
```

In [30]:
```python
cross_validate(algo, final_data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.2781  0.2783  0.2801  0.2797  0.2749  0.2782  0.0018
MAE (testset)     0.0404  0.0402  0.0410  0.0415  0.0409  0.0408  0.0005
Fit time          66.82   66.77   66.79   68.05   71.64   68.01   1.88
Test time         2.15    2.45    2.82    2.54    2.43    2.48    0.21
```

Out[30]:
```
{'test_rmse': array([0.27808127, 0.27825733, 0.28007764, 0.27972275, 0.274909
79]),
 'test_mae': array([0.04042886, 0.04020518, 0.041034  , 0.04149584, 0.0409266
]),
 'fit_time': (66.81780695915222,
  66.76883506774902,
  66.79082584381104,
  68.05404710769653,
  71.63683652877808),
 'test_time': (2.149674892425537,
  2.451490640640259,
  2.81726336479187,
  2.5394344329833984,
  2.429504156112671)}
```

In [31]:
```python
from surprise import NormalPredictor
from surprise import KNNBasic
from surprise import KNNWithMeans
from surprise import KNNWithZScore
from surprise import KNNBaseline
from surprise import SVD
from surprise import BaselineOnly
from surprise import SVDpp
from surprise import NMF
from surprise import SlopeOne
from surprise import CoClustering
```

In [32]:

```
benchmark = []
# Iterate over all algorithms
for algorithm in [SVD(), SVDpp()]:
    # , SVDpp(), SlopeOne(), NMF(), NormalPredictor(), KNNBaseline(), KNNBasic
(), KNNWithMeans(), KNNWithZScore(), BaselineOnly(), CoClustering()
    # Perform cross validation
    results = cross_validate(algorithm, final_data, measures=['RMSE'], cv=3, v
erbose=False)

    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp = tmp.append(pd.Series([str(algorithm).split(' ')[0].split('.')[-1]],
index=['Algorithm']))
    benchmark.append(tmp)

pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
```

Out[32]:

Algorithm	test_rmse	fit_time	test_time
SVDpp	0.280749	1656.151152	70.517408
SVD	0.282776	55.187311	4.187085

In [ ]: