```
# import liraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import packages from surprise library
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import train_test_split
from surprise.model_selection import cross_validate
from surprise.model_selection import GridSearchCV
from surprise import Dataset
from surprise import SVD

# import dataset

ratings = pd.read_csv('Amazon - Movies and TV Ratings.csv')
ratings.head()
```

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

   Movie8  Movie9  ...  Movie197  Movie198  Movie199  Movie200
Movie201  \
0     NaN     NaN  ...       NaN       NaN       NaN       NaN
NaN
1     NaN     NaN  ...       NaN       NaN       NaN       NaN
NaN
2     NaN     NaN  ...       NaN       NaN       NaN       NaN
NaN
3     NaN     NaN  ...       NaN       NaN       NaN       NaN
NaN
4     NaN     NaN  ...       NaN       NaN       NaN       NaN
NaN

   Movie202  Movie203  Movie204  Movie205  Movie206
0       NaN       NaN       NaN       NaN       NaN
1       NaN       NaN       NaN       NaN       NaN
```

```
2       NaN       NaN       NaN       NaN       NaN
3       NaN       NaN       NaN       NaN       NaN
4       NaN       NaN       NaN       NaN       NaN

[5 rows x 207 columns]
```

```
ratings.shape
```

```
(4848, 207)
```

There are 4848 users and 206 movies.

Task 1: Which movies have maximum views/ratings?

Movie with highest number of ratings:

```
movie_ratings_count = (~ratings.isna()).sum()[1:]
movie_ratings_count[movie_ratings_count == max(movie_ratings_count)]
```

```
Movie127    2313
dtype: int64
```

Movie 'Movie127' has high number of ratings.

Top 10 highly rated movies:

```
top_10_highly_rated_movies =
ratings.describe().loc['count', :].sort_values(ascending=False)
[:10].to_frame().astype('int')
```

```
top_10_highly_rated_movies
```

```
          count
Movie127   2313
Movie140    578
Movie16     320
Movie103    272
Movie29     243
Movie91     128
Movie92     101
Movie89      83
Movie158     66
Movie108     54
```

Above are the top 10 highly rated movies (movies with maximum number of ratings from the users). So these are the most watched movies.

```
ratings.drop('user_id', axis=1).sum().sort_values(ascending=False)
[:10].to_frame().astype('int')
```

```
             0
Movie127  9511
Movie140  2794
```

```
Movie16   1446
Movie103  1241
Movie29   1168
Movie91    586
Movie92    482
Movie89    380
Movie158   318
Movie108   252
```

So the movies we got with highly watched movies, are also highly rated/loved movies.

Task 2: What is the average rating for each movie? Define the top 5 movies with the maximum ratings.

```
movies_max_avg_ratings = ratings.drop('user_id',
axis=1).mean().sort_values(ascending=False)
[:5].to_frame().astype('int')
movies_max_avg_ratings
```

```
          0
Movie1    5
Movie55   5
Movie131  5
Movie132  5
Movie133  5
```

Above movies are having highest average rating (5). So these are the good movies which mostly get rating as 5. There will be other movies as well with average rating as 5. But these are the first 5 movies.

```
top5_movies =
movie_ratings_count[movies_max_avg_ratings.index].sort_values(ascendin
g=False).index[:5].tolist()

top5_movies
```

```
['Movie131', 'Movie132', 'Movie133', 'Movie55', 'Movie1']
```

Above are the top 5 movies with highest average ratings cosidering the rating counts.

Task 3: Define the top 5 movies with the least audience.

```
top5_movies_max_avg_rating_least_audience =
(movie_ratings_count[movies_max_avg_ratings.index]
                                              .sort_values().index[:5].
tolist())

top5_movies_max_avg_rating_least_audience
```

```
['Movie1', 'Movie55', 'Movie133', 'Movie132', 'Movie131']
```

Above are the top 5 movies with highest average ratings cosidering the least rating counts.

```
top5_movies_least_audience = movie_ratings_count.sort_values()
[:5].index.tolist()
top5_movies_least_audience
```

```
['Movie1', 'Movie71', 'Movie145', 'Movie69', 'Movie68']
```

Above are the top 5 movies with least rating counts/audieces.

Task 4: Recommendation Model

We have users and movies with the ratings. So we need to build a recommendation model that will predict the rating for the unwatched/unrated movies for the users (Replacing NaN values with predicted values).

Split the data into train and test

```
# Convert the dataset into standard format (with columns user_id,
movie_id, rating)

ratings_df = pd.melt(ratings, id_vars='user_id', value_name='rating',
var_name='movie_id')
ratings_df.head()
```

```
          user_id movie_id  rating
0  A3R5OBKS7OM2IR   Movie1     5.0
1   AH3QC2PC1VTGP   Movie1     NaN
2  A3LKP6WPMP9UKX   Movie1     NaN
3   AVIY68KEPQ5ZD   Movie1     NaN
4  A1CV1WROP5KTTW   Movie1     NaN
```

```
ratings_df['user_id'].nunique()
```

```
4848
```

```
ratings_df['movie_id'].nunique()
```

```
206
```

```
ratings_df['rating'].unique()
```

```
array([ 5., nan,  2.,  1.,  4.,  3.])
```

We have ratings like 1,2,3,4,5.

```
# Read the dataset
reader = Reader(rating_scale=(1,5))

rating_data = Dataset.load_from_df(ratings_df.fillna(0),
reader=reader)

# train test split
```

```
rating_train, rating_test = train_test_split(rating_data,
test_size=0.25)
```

Use recommendation model (SVD)

```
reco_alg = SVD()
reco_alg.fit(rating_train)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at
0x7f8be8592f10>
```

```
rating_pred = reco_alg.test(rating_test)
```

```
accuracy.rmse(rating_pred)
```

```
RMSE: 1.0258
```

```
1.0258158265299586
```

Predict using SVD for a user and a movie

```
uid = 'A3R5OBKS7OM2IR'
mid = 'Movie1'
rid = 5.0

reco_alg.predict(uid, mid, rid, verbose=True)
```

```
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 1.00
{'was_impossible': False}
```

```
Prediction(uid='A3R5OBKS7OM2IR', iid='Movie1', r_ui=5.0, est=1,
details={'was_impossible': False})
```

From the test, it seems that the predictions are not good enough. We can try out different NaN replacing methods like mean, median, mode etc.

So we can use cross_validate.

```
cross_validate(reco_alg, rating_data, measures=['rmse', 'mae'], cv= 3,
verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0273  1.0255  1.0256  1.0261  0.0008
MAE (testset)     1.0125  1.0118  1.0119  1.0121  0.0003
Fit time          36.39   36.70   36.79   36.63   0.17
Test time         3.13    3.16    3.14    3.14    0.01
```

```
{'test_rmse': array([1.02726558, 1.02552331, 1.02561488]),
 'test_mae': array([1.01254178, 1.01180508, 1.01189133]),
 'fit_time': (36.39019012451172, 36.70391631126404,
36.79219460487366),
```

```
 'test_time': (3.1296591758728027, 3.1619858741760254,
3.1400773525238037)}
```

Lets repeat the process for multiple options of filling NaN value and also with a subset of dataset.

```
def algo_train(data, algo):
    reader = Reader(rating_scale=(1,5))
    data_df = Dataset.load_from_df(data, reader=reader)

    print(cross_validate(algo, data_df, measures=['rmse', 'mae'], cv=
3, verbose=True))
    print('====================================')
    uid = 'A3R5OBKS7OM2IR'
    mid = 'Movie1'
    rid = 5.0

    algo.predict(uid, mid, rid, verbose=True)
    print('************************************')

# Take a small portion of data

rating_small = ratings.iloc[:1500, :100]
rating_small_df = pd.melt(rating_small, id_vars='user_id',
value_name='rating', var_name='movie_id')

algo_train(rating_small_df.fillna(0), SVD())
algo_train(rating_small_df.fillna(rating_small_df.mean()), SVD())
algo_train(rating_small_df.fillna(rating_small_df.median()), SVD())
```

```
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0434  1.0505  1.0452  1.0464  0.0030
MAE (testset)     1.0196  1.0222  1.0200  1.0206  0.0012
Fit time          5.28    5.30    5.32    5.30    0.02
Test time         0.36    0.34    0.66    0.45    0.15
{'test_rmse': array([1.04336264, 1.0505115 , 1.04519189]), 'test_mae':
array([1.01957481, 1.02221962, 1.02000606]), 'fit_time':
(5.276042222976685, 5.3037450313568115, 5.3193395137786865),
'test_time': (0.35703301429748535, 0.3377223014831543,
0.6606464385986328)}
====================================
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 1.00
{'was_impossible': False}
************************************
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0808  0.0902  0.0937  0.0882  0.0054
```

```
MAE (testset)     0.0152  0.0157  0.0155  0.0155  0.0002
Fit time          5.24    5.29    5.35    5.29    0.04
Test time         0.34    0.65    0.34    0.44    0.15
{'test_rmse': array([0.0808228 , 0.09016064, 0.09367273]), 'test_mae':
array([0.01518428, 0.01574488, 0.01550985]), 'fit_time':
(5.238287448883057, 5.285528182983398, 5.34529447555542), 'test_time':
(0.3425281047821045, 0.652195930480957, 0.33948421478271484)}
========================================
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 4.55
{'was_impossible': False}
************************************
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.1092  0.0998  0.0784  0.0958  0.0129
MAE (testset)     0.0111  0.0104  0.0104  0.0106  0.0003
Fit time          5.28    5.32    5.31    5.30    0.02
Test time         0.65    0.34    0.65    0.55    0.15
{'test_rmse': array([0.10915164, 0.09978587, 0.07838541]), 'test_mae':
array([0.01108948, 0.01044131, 0.01041359]), 'fit_time':
(5.279655933380127, 5.3221352100372314, 5.313053131103516),
'test_time': (0.6503043174743652, 0.336200475692749,
0.6541712284088135)}
========================================
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 5.00
{'was_impossible': False}
************************************
```

Here we can see, with median replacement of NaN gave better result than others.

Now we can train the SVD model with proper parameter settings.

```
# GridSearchCV param

params = {'n_epochs': [20,30], 'lr_all': [0.005, 0.05], 'n_factors':
[100, 150]}

# GridSearchCV

gsc = GridSearchCV(SVD, params, measures=['rmse', 'mae'], cv= 3,
refit=True)
gsc.fit(rating_data)

gsc.best_score

{'rmse': 0.09108262495583558, 'mae': 0.006491875481785602}

gsc.best_params

{'rmse': {'n_epochs': 30, 'lr_all': 0.05, 'n_factors': 150},
 'mae': {'n_epochs': 30, 'lr_all': 0.05, 'n_factors': 150}}
```

Above are the best parameters for best scores.

Final SVD Recommendation Algorithm

We can use rec_algo_rsme or rec_algo_mae algorithm instances, or create a new one with best parameters.

```
rating_data =
Dataset.load_from_df(ratings_df.fillna(ratings_df.median()),
reader=reader)
rating_train, rating_test = train_test_split(rating_data,
test_size=0.25)

rec_algo = SVD(n_epochs= 20, lr_all= 0.05, n_factors= 100)
rec_algo.fit(rating_train)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at
0x7f8bed0a3690>
```

```
rating_pred = rec_algo.test(rating_test)
rating_pred1 = gsc.test(rating_test)

print(accuracy.rmse(rating_pred))
print(accuracy.rmse(rating_pred1))
```

```
RMSE: 0.0892
0.08916853549396428
RMSE: 0.0780
0.0780056426437694
```

Predict rating for a userid, movieid and known rating.

```
uid = 'A3R5OBKS7OM2IR'
mid = 'Movie1'
rid = 5.0

print(reco_alg.predict(uid, mid, rid, verbose=False))
print(gsc.predict(uid, mid, rid))
```

```
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 1.00
{'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 5.00
{'was_impossible': False}
```

Predict rating for a userid, movieid and unknown rating.

```
uid = 'AH3QC2PC1VTGP'
mid = 'Movie1'

print(reco_alg.predict(uid=uid, iid=mid, verbose=False))
print(gsc.predict(uid, mid))
```

```
user: AH3QC2PC1VTGP item: Movie1     r_ui = None   est = 1.00
{'was_impossible': False}
user: AH3QC2PC1VTGP item: Movie1     r_ui = None   est = 5.00
{'was_impossible': False}
```

We can use above recommendation algorithm (model from GridSearchCV) and predict rating for a user and a movie.