```
'''
Steps -
1. Read and explore the given dataset. ( Rename column/add headers, plot histograms,find data
2. Take a subset of the dataset to make it less sparse/ denser. ( For example, keep the users
3. Split the data randomly into train and test dataset. ( For example, split it in 70/30 rati
4. Build Popularity Recommender model.
5. Build Collaborative Filtering model.
6. Evaluate both the models. ( Once the model is trained on the training data, it can be used
also use a different method to evaluate the models.
7. Get top - K ( K = 5) recommendations. Since our goal is to recommend new products to each
8. Summarise your insights.
'''
```

```
'\nSteps -\n1. Read and explore the given dataset. ( Rename column/add headers, plot his
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io
from datetime import datetime
import time
```

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /content/drive
```

```
df_raw = pd.read_csv('/content/drive/My Drive/ratings_Electronics.csv',names=['userId', 'prod
```

```
df_raw.shape
```

```
(7824482, 4)
```

```
ratings_Electronics_200K = df_raw.head(200000)
ratings_Electronics_200K.shape
```

```
(200000, 4)
```

```
df = ratings_Electronics_200K.copy(deep=True)
```

```
df.head()
```

	userId	productId	ratings	timestamp
0	AKM1MP6P0OYPR	0132793040	5.0	1365811200
1	A2CX7LUOHB2NDG	0321732944	5.0	1341100800
2	A2NWSAGRHCP8N5	0439886341	1.0	1367193600
3	A2WNBOD3WNDNKT	0439886341	3.0	1374451200
4	A1GI0U4ZRJA8WN	0439886341	1.0	1334707200

```
df.drop(columns=['timestamp'],inplace=True)
```

```
df.head(4)
```

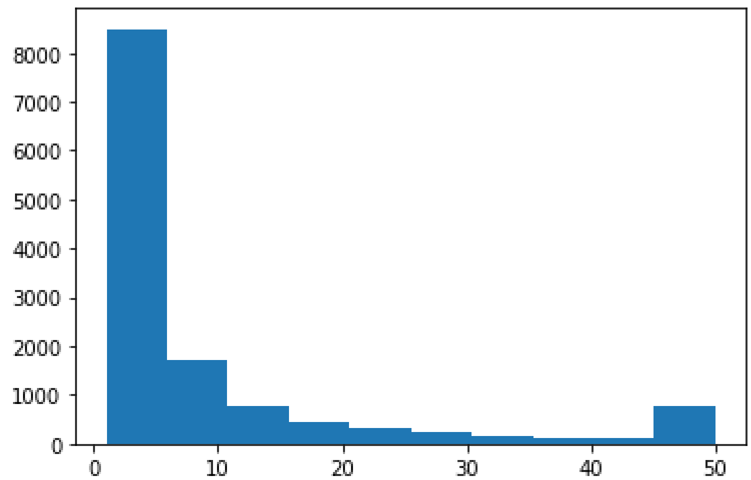
	userId	productId	ratings
0	AKM1MP6P0OYPR	0132793040	5.0
1	A2CX7LUOHB2NDG	0321732944	5.0
2	A2NWSAGRHCP8N5	0439886341	1.0
3	A2WNBOD3WNDNKT	0439886341	3.0

```
df.shape
```

```
(200000, 3)
```

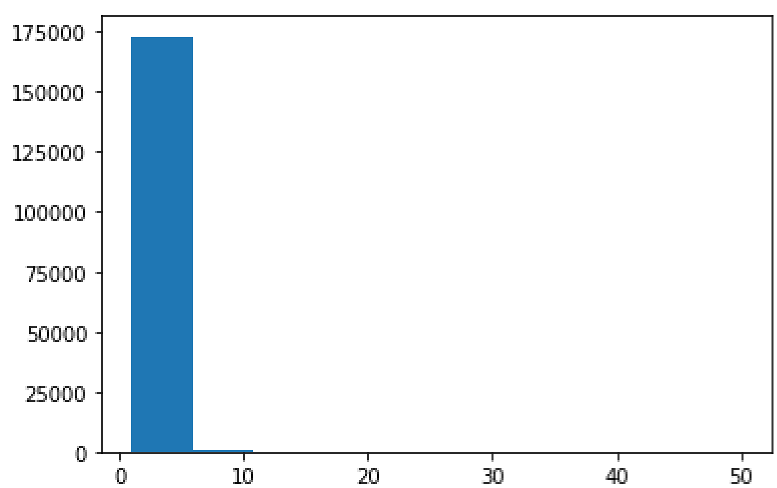
```
#####################################################  EDA ###################################
```

```
# Number of ratings per book
data = df.groupby('productId')['ratings'].count().clip(upper=50)# Number of ratings per book
data = df.groupby('productId')['ratings'].count().clip(upper=50)



# Create trace
trace = plt.hist(x = data.values)
```




```
# Number of ratings per book
data = df.groupby('userId')['productId'].count().clip(upper=50)
```

```
# Create trace
trace = plt.hist(x = data.values)
```




```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 3 columns):
userId       200000 non-null object
productId    200000 non-null object
ratings      200000 non-null float64
dtypes: float64(1), object(2)
memory usage: 4.6+ MB
```

```
df.shape[0]
```

```
200000
```

```
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
ratings	200000.0	4.013895	1.373682	1.0	3.0	5.0	5.0	5.0

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

```
userId       0
productId    0
ratings      0
dtype: int64
```

```
print(df['userId'].nunique())
```

```
173349
```

```
print(df['productId'].nunique())
```

```
13131
```

```
##                          P O P U L A R I T Y                    B A S E D
```

```
product_unique_count = df.productId.value_counts().to_frame()
```

```
product_unique_count.reset_index(inplace=True)
```

```
product_unique_count.head(5)
```

	index	productId
0	B00004ZCJE	2547
1	B00001P4ZH	2075
2	B000065BP9	1714
3	B00004T8R2	1692
4	B00001WRSJ	1586

```
##### We want to know which pruduct are sold most as sold_count
```

```
product_unique_count.rename(columns={'index':'productId','productId':'sold_count'},inplace=Tr

product_unique_count = product_unique_count.sort_values(by='productId',ascending=False)

product_unique_count.head()
```

	productId	sold_count
60	B00006JN3G	389
2349	B00006JN2R	14
6359	B00006JM74	3
8954	B00006JM73	2
7382	B00006JM72	3

```
product_unique_count.sold_count.sum()
```

```
200000
```

```
###################### CHECKING FOR THE AVERAGE RATINGS ###################################

product_unique_count[(product_unique_count.sold_count==7)==True]
```

	productId	sold_count
3964	B00006JJPP	7
3926	B00006JHYW	7
3819	B00006JBKN	7
3921	B00006J09F	7
3870	B00006IS63	7
...	...	...
3900	9985558065	7
4009	9981724742	7
4058	9876050621	7
3700	6000008775	7
3800	1615513388	7

402 rows × 2 columns

```
##     CHECKING ANY ONE PRODUCT SOLD COUNT = 7 and ratings

df[(df['productId']=='9985558065')==True]
```

```
df[(df['productId']=='9985558065')==True]
```

	userId	productId	ratings
6879	A3P1UWQ4NWEYMX	9985558065	2.0
6880	A36HT2ITEIAJXQ	9985558065	5.0
6881	AGSXTHPNGNA16	9985558065	5.0
6882	A3G37IM6Z8ZNB4	9985558065	5.0
6883	A1NUHPQ47DKGW7	9985558065	5.0
6884	A2Z123EZCA9177	9985558065	5.0
6885	AX2O4I0TZ7STS	9985558065	5.0

```
#################  Calculating the average product ratings ##################################

product_avg_ratings = np.round(df.groupby(df['productId'])['ratings'].sum()/df.groupby(df['pr

product_avg_ratings = product_avg_ratings.to_frame().reset_index()

product_avg_ratings.head(5)
```

	productId	ratings
0	0132793040	5.0
1	0321732944	5.0
2	0439886341	2.0
3	0511189877	4.0
4	0528881469	3.0

```
product_avg_ratings = product_avg_ratings.sort_values(by='productId',ascending=False)
product_avg_ratings.head(4)
```

	productId	ratings
13130	B00006JN3G	4.0
13129	B00006JN2R	4.0
13128	B00006JM74	1.0
13127	B00006JM73	4.0

```
product_avg_ratings[(product_avg_ratings['productId']=='9985558065')==True]
```

	productId	ratings
915	9985558065	5.0

```
product_recommend_pop = pd.merge(product_unique_count,product_avg_ratings, on='productId',how
```

```
product_recommend_pop.head()
```

	productId	sold_count	ratings
0	B00006JN3G	389	4.0
1	B00006JN2R	14	4.0
2	B00006JM74	3	1.0
3	B00006JM73	2	4.0
4	B00006JM72	3	2.0

```
################## Recommending the product with average rating as 5 and sold maximum #######
```

```
product_recommend_pop.sort_values(by=['ratings','sold_count'],ascending=False).head(5)
```

	productId	sold_count	ratings
10882	B00001WRSJ	1586	5.0
5026	B00005T3G0	1287	5.0
6042	B00005LEN4	1107	5.0
5689	B00005NIMJ	884	5.0
8465	B00004Z5M1	815	5.0

```
from sklearn.model_selection import train_test_split
```

```
df1 = df.groupby('userId').count()
```

```
df1.reset_index(inplace=True)
```

```
df1.head()
```

	userId	productId	ratings
0	A001944026UMZ8T3K5QH1	1	1
1	A00570163ATHRHPDG3GKN	1	1
2	A00625243BI8W1SSZNLMD	1	1
3	A00766851QZZUBOVF4JFT	1	1
4	A00995931BE16NG4F52QC	1	1

```
print(df1.shape)
print(df1['userId'].nunique())
```

```
(173349, 3)
173349
```

```
## Identifying the users  who rated more than and equal to 50 products
```

```
df_user_10 = df1[(df1['productId']>=10)==True].reset_index()
```

```
df_user_10.shape
```

```
(177, 4)
```

```
df_user_10_ratings = pd.DataFrame.merge(df,df_user_10,on='userId',how='inner',sort=True)
```

```
print(df_user_10_ratings.shape)
print(df_user_10_ratings['userId'].nunique())
```

```
(3187, 6)
177
```

```
df_user_10_ratings.head(3)
```

	userId	productId_x	ratings_x	index	productId_y	ratings_y
0	A10C84Y38RT22P	B000023VUL	5.0	518	13	13
1	A10C84Y38RT22P	B00003CW9Q	5.0	518	13	13
2	A10C84Y38RT22P	B00003CWBX	5.0	518	13	13

```
df_user_10_ratings.drop(columns=['productId_y','ratings_y','index'],inplace=True)
```

```
df_user_10_ratings.head(3)
```

	userId	productId_x	ratings_x
0	A10C84Y38RT22P	B000023VUL	5.0
1	A10C84Y38RT22P	B00003CW9Q	5.0
2	A10C84Y38RT22P	B00003CWBX	5.0

```
print(df_user_10_ratings['userId'].nunique())
```

```
177
```

```
print(df_user_10_ratings['productId_x'].nunique())
```

```
2092
```

```
####################################################  INSTALLING THE SURPRISE LIBRARY #######
```

```
pip install surprise
```

```
Collecting surprise
  Downloading https://files.pythonhosted.org/packages/61/de/e5cba8682201fcf9c3719a6fdda9
Collecting scikit-surprise
  Downloading https://files.pythonhosted.org/packages/f5/da/b5700d96495fb4f092be497f0249
     |████████████████████████████████| 6.5MB 3.5MB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (fr
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.0-cp36-cp36m-linux_x86
  Stored in directory: /root/.cache/pip/wheels/cc/fa/8c/16c93fccce688ae1bde7d979ff102f7b
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.0 surprise-0.1
```

```
'''
                          C o l l a b                  B A S E D                      R
'''
```

```
'\n                          C o l l a b                  B A S E D
```

```
#################################################################### RECOMMENDATION ALGORITHM
```

```
## Use user_based true/false to switch between user-based or item-based collaborative filteri
```

```
from surprise import KNNWithMeans,Reader,Dataset,SVD,SVDpp,SlopeOne, NMF, NormalPredictor, KN
```

```
from sklearn.model_selection import train_test_split , cross_val_score ,GridSearchCV,cross_va


from surprise.model_selection.validation import cross_validate
from surprise.model_selection.search import GridSearchCV


bsl_options = {'method': 'als',
               'learning_rate': .00005,
               'n_epochs': 5,
               'reg_u': 12,
               'reg_i': 5
               }


MLA =[   KNNBasic()
        ,SVD()
      , SVDpp()
      , SlopeOne()
      , NMF()
      , NormalPredictor()
      ,KNNWithZScore()
      , KNNBaseline()
      , BaselineOnly()
      , CoClustering()
     ,KNNWithMeans(k=5, bsl_options=bsl_options,sim_options={'name': 'cosine','user_based': T
     ,KNNWithMeans(k=5, sim_options={'name': 'cosine','user_based': False})
     ,KNNWithMeans(k=5, bsl_options=bsl_options,sim_options={'name': 'pearson_baseline','shri
      ,KNNWithMeans(k=5, bsl_options=bsl_options,sim_options={'name': 'pearson_baseline','shr
]


print(df.head(4))
print(df.shape)
```

```
            userId   productId  ratings
0   AKM1MP6P0OYPR  0132793040      5.0
1  A2CX7LUOHB2NDG  0321732944      5.0
2  A2NWSAGRHCP8N5  0439886341      1.0
3  A2WNBOD3WNDNKT  0439886341      3.0
(200000, 3)
```

```
data_recomm_raw = df_user_10_ratings.copy(deep=True)


data_recomm = data_recomm_raw.rename(columns={'userId':'uid','productId_x':'iid','ratings_x':
data_recomm.head()
```

	uid	iid	rating
0	A10C84Y38RT22P	B000023VUL	5.0
1	A10C84Y38RT22P	B00003CW9Q	5.0
2	A10C84Y38RT22P	B00003CWBX	5.0
3	A10C84Y38RT22P	B00004RBR6	5.0
4	A10C84Y38RT22P	B00004Z5A5	4.0

```
############################### MATRIX FACTORIZATION BASED REDOMMENDATIONS ##################
```

```
data_recom_mat = data_recomm.copy(deep=True)
```

```
reader = Reader(rating_scale=(1,5))
data_recom_mat_reader = Dataset.load_from_df(data_recom_mat, reader)
```

```
param_grid = {'lr_all':[.001,.01],'reg_all':[0.1,0.5]}
```

```
gs = GridSearchCV(SVDpp,param_grid,cv=3)
```

```
gs.fit(data_recom_mat_reader)
```

```
print(gs.best_params['mae'])
```

```
{'lr_all': 0.01, 'reg_all': 0.1}
```

```
print(gs.best_params['rmse'])
```

```
{'lr_all': 0.01, 'reg_all': 0.5}
```

```
benchmark = []
# Iterate over all algorithms
for algorithm in MLA:
    # Perform cross validation
    results = cross_validate(algorithm, data_recom_mat_reader , cv=10, verbose=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=['Algorit
    benchmark.append(tmp)

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

```
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

	test_rmse	test_mae	fit_time	test_time
Algorithm				
SVD	1.006040	0.755627	0.184034	0.002495
BaselineOnly	1.006772	0.765564	0.007813	0.001700
SVDpp	1.008800	0.749677	1.412097	0.009424
KNNWithMeans	1.053471	0.791651	0.009103	0.003193
KNNBaseline	1.057756	0.778046	0.009173	0.003454
KNNWithMeans	1.075866	0.804466	0.006316	0.003723

KNNWithZScore	1.085930	0.807261	0.011918	0.003819
SlopeOne	1.101901	0.824963	0.099108	0.005037
KNNBasic	1.123505	0.830128	0.001728	0.003538
CoClustering	1.136705	0.825918	0.223642	0.001774
KNNWithMeans	1.166904	0.849177	0.216933	0.007714
KNNWithMeans	1.184388	0.853791	0.150901	0.006234
NMF	1.196855	0.931455	0.302693	0.002162
NormalPredictor	1.354858	1.012515	0.003857	0.002548

```
## we see that SVDpp has minimum RMSE  so building the model on same
```

```
trainset_mat = data_recom_mat_reader.build_full_trainset()
```

```
trainset_mat.ur[1][1]
```

```
(14, 1.0)
```

```
algo = SVDpp()
algo.fit(trainset_mat)
```

```
algo.fit(trainset_mat)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVDpp at 0x7f219f547a20>
```

```
predict = algo.predict(uid='A00038802J7X43YTW44TD',iid= 'B0000645RH')
score=predict.est
print(score)
```

```
4.262629432067776
```

```
# Than predict ratings for all pairs (u, i) that are NOT in the training set.
testset_mat = trainset_mat.build_anti_testset()
```

```
testset_mat[9:23]
```

```
[('A10C84Y38RT22P', 'B00004TZK6', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00004Z0BN', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00004Z0C2', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00004Z672', 4.262629432067776),
 ('A10C84Y38RT22P', 'B000059L44', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00005B8SF', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00005B9W6', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00005QBUR', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00005QBUU', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00005QT5J', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00005T39Y', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00005T3SP', 4.262629432067776),
 ('A10C84Y38RT22P', 'B00005V54U', 4.262629432067776),
 ('A10C84Y38RT22P', 'B000060OEO', 4.262629432067776)]
```

```
predictions_mat = algo.test(testset_mat)
```

```
predictions_mat[:9]
```

```
[Prediction(uid='A10C84Y38RT22P', iid='B000001OMN', r_ui=4.262629432067776, est=4.046859
 Prediction(uid='A10C84Y38RT22P', iid='B00000J1G6', r_ui=4.262629432067776, est=3.723717
 Prediction(uid='A10C84Y38RT22P', iid='B00000J9Z7', r_ui=4.262629432067776, est=4.309784
 Prediction(uid='A10C84Y38RT22P', iid='B00000JBYW', r_ui=4.262629432067776, est=3.602165
 Prediction(uid='A10C84Y38RT22P', iid='B00000JCTD', r_ui=4.262629432067776, est=4.285002
 Prediction(uid='A10C84Y38RT22P', iid='B00001P3XM', r_ui=4.262629432067776, est=4.172571
 Prediction(uid='A10C84Y38RT22P', iid='B00001W0D4', r_ui=4.262629432067776, est=4.253347
 Prediction(uid='A10C84Y38RT22P', iid='B00001ZWRV', r_ui=4.262629432067776, est=4.177548
 Prediction(uid='A10C84Y38RT22P', iid='B00004T8R2', r_ui=4.262629432067776, est=3.876421
```

```
from surprise import accuracy
```

```
# get RMSE
print("User-based Model : Test Set")
accuracy.rmse(predictions_mat, verbose=True)
```

```
User-based Model : Test Set
RMSE: 0.4014
0.4013974860307046
```

```
for uid, iid, r_ui, est, _ in predictions_mat[:9]:
  print(uid, iid, r_ui, est, _)
```

```
A10C84Y38RT22P B000001OMN 4.262629432067776 4.046859001665824 {'was_impossible': False}
A10C84Y38RT22P B00000J1G6 4.262629432067776 3.723717665281609 {'was_impossible': False}
A10C84Y38RT22P B00000J9Z7 4.262629432067776 4.309784754799142 {'was_impossible': False}
A10C84Y38RT22P B00000JBYW 4.262629432067776 3.6021651223330626 {'was_impossible': False}
A10C84Y38RT22P B00000JCTD 4.262629432067776 4.285002191448045 {'was_impossible': False}
A10C84Y38RT22P B00001P3XM 4.262629432067776 4.172571949859146 {'was_impossible': False}
A10C84Y38RT22P B00001W0D4 4.262629432067776 4.253347047690503 {'was_impossible': False}
A10C84Y38RT22P B00001ZWRV 4.262629432067776 4.177548644778244 {'was_impossible': False}
A10C84Y38RT22P B00004T8R2 4.262629432067776 3.8764219541131215 {'was_impossible': False}
```

```
def get_top_n(predictions,userid, n=5):
  # First map the predictions to each user.
  from collections import defaultdict
  top_n =  defaultdict(list)
  for uid, iid, r_ui, est, _ in predictions:
     ## we can use r_ui also instead of true_r and we used _ to resolve error "too many value
     ###  ## A00038802J7X43YTW44TD B0000645RH 4.446808510638298 4.07584978867076 {'was_imposs
     top_n[uid].append((iid, est)) ##  {'A00038802J7X43YTW44TD': [('B0000645RH', 4.0758497886

        # Then sort the predictions for each user and retrieve the k highest ones.
        # for uid, user_ratings in top_n.items():
     userid_list = top_n[userid]
     userid_list.sort(key=lambda x: x[1], reverse=True)
        # L = [(1,2), (2,3), (4,5), (3,4), (6,7), (6,7), (3,8)]
        #[x[1] for x in L]
     items_sorted_list = [x[0] for x in userid_list]
     top_n_recomm =[]
     top_n_recomm = items_sorted_list[:n]
  return 'Top {} recommended Items are -  {}'.format(n,top_n_recomm)
```

```
top_n = get_top_n(predictions_mat,'A10C84Y38RT22P', n=10)
top_n
```

```
"Top 10 recommended Items are -  ['B00004WHF9', 'B00006HZ0L', 'B000031KIM', 'B00005Q5U5'
```

```
############################################################### USER-USER RECOMMENDATION ####
```

```
from surprise.model_selection import train_test_split
```

```
reader = Reader(rating_scale=(1,5))
dataset_reader = Dataset.load_from_df(data_recom_mat, reader)


trainset_70, testset_30 = train_test_split(dataset_reader, test_size=.3)


MLA_columns = ['MLA_Name']
MLA_compare_user = pd.DataFrame(columns = MLA_columns)
MLA_compare_user
```

MLA_Name

```
MLA_user =[   KNNBasic()
        ,SVD()
      , SVDpp()
      , SlopeOne()
      , NMF()
      , NormalPredictor()
      ,KNNWithZScore()
      , KNNBaseline()
      , BaselineOnly()
      , CoClustering()
     ,KNNWithMeans(k=5, bsl_options=bsl_options,sim_options={'name': 'cosine','user_based': T
     ,KNNWithMeans(k=5, bsl_options=bsl_options,sim_options={'name': 'pearson_baseline','shri
]


row_index = 0
for alg in MLA_user:

    from datetime import datetime
    import time

    #set name and parameters
    #print(alg.__class__.__name__)
    MLA_compare_user.loc[row_index, 'MLA_Name'] = alg.__class__.__name__
    #print(str(alg.get_params()))
   # Use user_based true/false to switch between user-based or item-based collaborative filte
    algo = alg
    algo.fit(trainset_70)
    test_pred = algo.test(testset_30)
    MLA_compare_user.loc[row_index, 'RMSE'] =  accuracy.rmse(test_pred, verbose=True)
    MLA_compare_user.loc[row_index, 'Timestamp'] = str(datetime.now().strftime('%Y-%m-%d %H:%

    row_index+=1
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.0658
RMSE: 0.9840
RMSE: 0.9829
RMSE: 1.0509
RMSE: 1.1399
RMSE: 1.3401
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.0442
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.0078
Estimating biases using als...
RMSE: 0.9891
RMSE: 1.0743
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 1.0331
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
```

```
MLA_compare_user
```

	MLA_Name	RMSE	Timestamp
0	KNNBasic	1.065833	2019-11-22 05:19:09
1	SVD	0.983984	2019-11-22 05:19:09
2	SVDpp	0.982897	2019-11-22 05:19:10
3	SlopeOne	1.050913	2019-11-22 05:19:10
4	NMF	1.139885	2019-11-22 05:19:10
5	NormalPredictor	1.340074	2019-11-22 05:19:10
6	KNNWithZScore	1.044199	2019-11-22 05:19:10
7	KNNBaseline	1.007778	2019-11-22 05:19:10
8	BaselineOnly	0.989079	2019-11-22 05:19:10
9	CoClustering	1.074277	2019-11-22 05:19:10
10	KNNWithMeans	1.033132	2019-11-22 05:19:10
11	KNNWithMeans	1.032294	2019-11-22 05:19:10

```
fmt = '%Y-%m-%d %H:%M:%S'
max_tstmp = datetime.strptime(MLA_compare_user['Timestamp'].max(), fmt)
min_tstmp = datetime.strptime(MLA_compare_user['Timestamp'].min(), fmt)


td = max_tstmp - min_tstmp
td_mins = int(round(td.total_seconds() / 60))

print('The model performance is approx. %s minutes' % td_mins)
```

```
The model performance is approx. 0 minutes
```

```
MLA_compare_user.sort_values(by='RMSE',ascending=True,inplace=True)
```

```
############################################################### ITEM - ITEM RECOMMENDATION ##
```

```
from surprise.model_selection import train_test_split
```

```
reader = Reader(rating_scale=(1,5))
dataset_reader = Dataset.load_from_df(data_recom_mat, reader)
```

```
trainset_70, testset_30 = train_test_split(dataset_reader, test_size=.3)
```

```
MLA_columns = ['MLA_Name']
MLA_compare_item = pd.DataFrame(columns = MLA_columns)
MLA_compare_item
```

MLA_Name

```
MLA_item =[   KNNBasic()
        ,SVD()
      , SVDpp()
      , SlopeOne()
      , NMF()
      , NormalPredictor()
      ,KNNWithZScore()
```

```
       , KNNBaseline()
       , BaselineOnly()
       , CoClustering()
      ,KNNWithMeans(k=5, bsl_options=bsl_options,sim_options={'name': 'cosine','user_based': F
      ,KNNWithMeans(k=5, bsl_options=bsl_options,sim_options={'name': 'pearson_baseline','shri
]


row_index = 0
for alg in MLA_item:

    from datetime import datetime
    import time

    #set name and parameters
    #print(alg.__class__.__name__)
    MLA_compare_item.loc[row_index, 'MLA_Name'] = alg.__class__.__name__
    #print(str(alg.get_params()))
   # Use user_based true/false to switch between user-based or item-based collaborative filte
    algo = alg
    algo.fit(trainset_70)
    test_pred = algo.test(testset_30)
    MLA_compare_item.loc[row_index, 'RMSE'] =  accuracy.rmse(test_pred, verbose=True)
    MLA_compare_item.loc[row_index, 'Timestamp'] = str(datetime.now().strftime('%Y-%m-%d %H:%

    row_index+=1
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.0987
RMSE: 1.0167
RMSE: 1.0200
RMSE: 1.0715
RMSE: 1.2128
RMSE: 1.3629
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.0674
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.0519
Estimating biases using als...
RMSE: 1.0161
RMSE: 1.1449
Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 1.1653
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
RMSE: 1.1773
```