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

import warnings


warnings.filterwarnings('ignore')
%matplotlib inline
```

```
#!pip install scikit-surprise
from surprise import KNNBasic, SVD, NormalPredictor, KNNBaseline,KNNWithMeans, KNNWithZScore, BaselineOnly, CoClustering, Reader, dataset, ac
```

```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 3.2 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.13.1)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp310-cp310-linux_x86_64.whl size=2357282 sha256=b1fdf9c2454960c160a
  Stored in directory: /root/.cache/pip/wheels/4b/3f/df/6acbf0a40397d9bf3ff97f582cc22fb9ce66adde75bc71fd54
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.4
```

```
columns =['userID','productID','ratings','timestamp']

data = pd.read_csv('/content/ratings_Electronics.csv',names=columns)
data.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

```
data.shape
```

```
(7824482, 4)
```

```
data.describe()
```

	ratings	timestamp
count	7.824482e+06	7.824482e+06
mean	4.012337e+00	1.338178e+09
std	1.380910e+00	6.900426e+07
min	1.000000e+00	9.127296e+08
25%	3.000000e+00	1.315354e+09
50%	5.000000e+00	1.361059e+09
75%	5.000000e+00	1.386115e+09
max	5.000000e+00	1.406074e+09

```
##Dropping the 'timestamp' as it is not needed
data = data.drop('timestamp',axis = 1)
```

Missing value

```
data.isna().sum()
```

```
              0
    userID    0
 productID    0
   ratings    0

dtype: int64
```

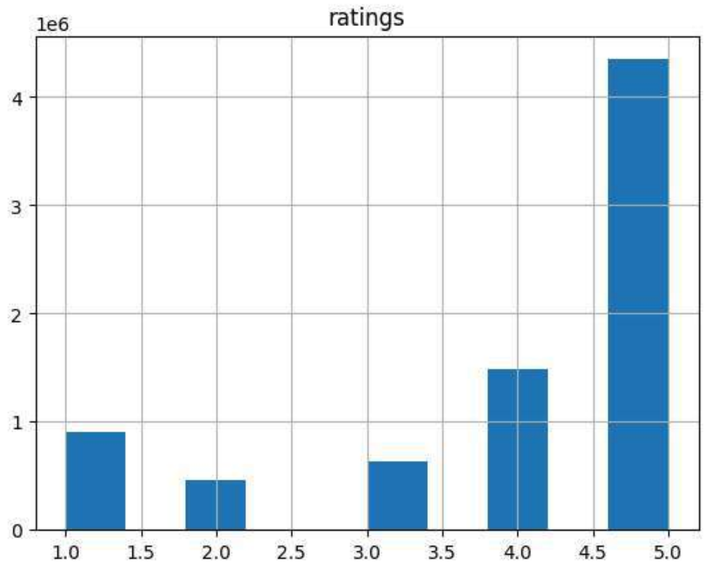
```
data.shape
```

```
(7824482, 3)
```

Histogram plot

```
data.hist('ratings',bins = 10)
```

```
array([[<Axes: title={'center': 'ratings'}>]], dtype=object)
```



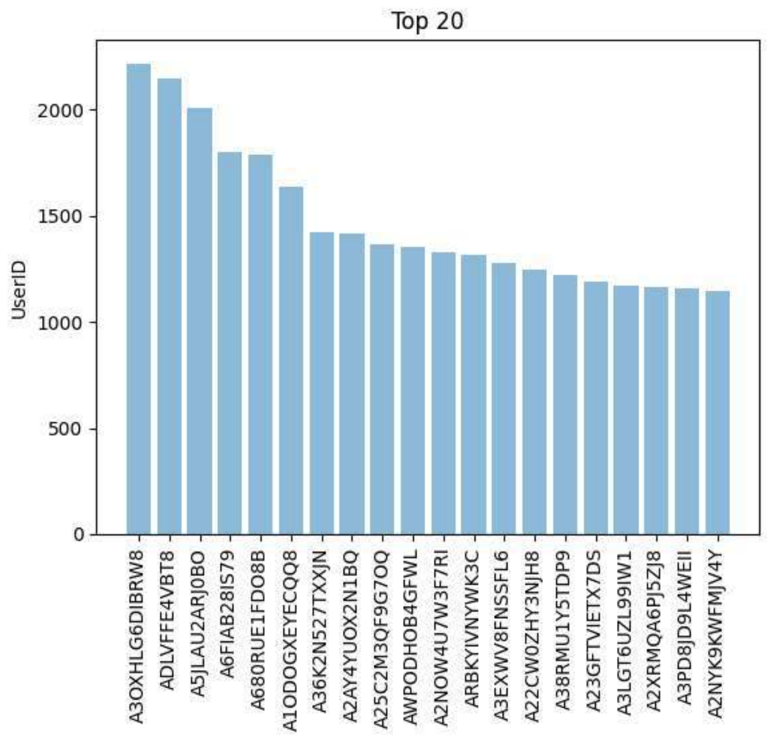
```
popular = data[['userID','ratings']].groupby('userID').sum().reset_index()
popular_20 = popular.sort_values('ratings',ascending = False).head(20)

import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = (list(popular_20['userID']))
y_pos = np.arange(len(objects))
performance = (list(popular_20['ratings']))

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects,rotation = 'vertical')
plt.ylabel('UserID')
plt.title('Top 20')

plt.show()
```



```
# unique users
data.userID.value_counts()
```

```
                  count
         userID
A5JLAU2ARJ0BO       520
ADLVFFE4VBT8        501
A3OXHLG6DIBRW8      498
A6FIAB28IS79        431
A680RUE1FDO8B       406
      ...           ...
A1IUWX30VMVJGP        1
A1WBP7XSZI6AUL        1
A2K7UNJHE9ZR0G        1
A1A6SIW6EWF6FP        1
A10M2KEFPEQDHN        1
4201696 rows × 1 columns

dtype: int64
```

```
print('Number of unique users',len(data['userID'].unique()))
```

```
Number of unique users 4201696
```

```
print('Number of unique products', len(data.productID.unique()))
```

```
Number of unique products 476002
```

```
print('Unique Ratings',data['ratings'].unique())
```

```
Unique Ratings [5. 1. 3. 2. 4.]
```

```
min_ratings1 = data[(data['ratings'] < 2.0)]
```

```
print('Numbeer of unique products rated low',len(min_ratings1['productID'].unique()))
```

```
Numbeer of unique products rated low 176283
```

```
med_ratings1 = data[(data['ratings'] > 2.0) & (data['ratings'] < 4.0)]
```

```
print('Number of unique products rated medium',len(med_ratings1['productID'].unique()))
```

```
Number of unique products rated medium 152827
```

```
max_ratings1 = data[(data['ratings'] >= 4.0)]
```

```
print('Number of unique products rated high',len(max_ratings1['productID'].unique()))
```

```
Number of unique products rated high 410110
```

```
numeric_columns = data.select_dtypes(include='number').columns
avg_rating_prod = data.groupby('productID')[numeric_columns].sum() / data.groupby('productID')[numeric_columns].count()
```

```
print(avg_rating_prod.head())
```

```
             ratings
productID
0132793040  5.000000
0321732944  5.000000
0439886341  1.666667
0511189877  4.500000
0528881469  2.851852
```

```
print('Top 10 highly rated products \n',avg_rating_prod.nlargest(10,'ratings'))
```

```
Top 10 highly rated products
             ratings
productID
0132793040      5.0
0321732944      5.0
059400232X      5.0
0594033934      5.0
0594287995      5.0
0594450209      5.0
0594450705      5.0
0594511488      5.0
0594514789      5.0
0594549558      5.0
```

Take a subset of the dataset to make it less denser

```
userID = data.groupby('userID').count()
```

```
top_user = userID[userID['ratings']>= 50].index
```

```
topuser_ratings_data = data[data['userID'].isin(top_user)]
```

```
#topuser_ratings_data.drop('productID',axis = 1,inplace = True)
```

```
topuser_ratings_data.shape
```

```
(125871, 3)
```

```
topuser_ratings_data.head()
```

	userID	productID	ratings
94	A3BY5KCNQZXV5U	0594451647	5.0
118	AT09WGFUM934H	0594481813	3.0
177	A32HSNCNPRUMTR	0970407998	1.0
178	A17HMM1M7T9PJ1	0970407998	4.0
492	A3CLWR1UUZT6TG	0972683275	5.0

```
topuser_ratings_data.sort_values(by='ratings',ascending = False).head()
```

	userID	productID	ratings
94	A3BY5KCNQZXV5U	0594451647	5.0
4256669	A680RUE1FDO8B	B004M8RWDE	5.0
4258497	AOMEH9W6LHC4S	B004M8SBNE	5.0
4258199	A2GKMXRLI7KLFP	B004M8SBD4	5.0
4258099	A1UNJ46NSB352E	B004M8SBCK	5.0

Keep data only for products that have 50 or more ratings

```
prodID = data.groupby('productID').count()
```

```
top_prod = prodID[prodID['ratings']>= 50].index
```

```
top_ratings_data = topuser_ratings_data[topuser_ratings_data['productID'].isin(top_prod)]
```

```
top_ratings_data.sort_values(by = 'ratings',ascending = False).head()
```

	userID	productID	ratings
492	A3CLWR1UUZT6TG	0972683275	5.0
4220481	A2AY4YUOX2N1BQ	B004KJE8FU	5.0
4217715	A2Q204DY2L7YRP	B004K8WPUQ	5.0
4218748	A87CILADRIZW0	B004KA8Y4U	5.0
4219205	A11KZ906QD08C5	B004KCI80I	5.0

```
top_ratings_data.shape
```

```
(79182, 3)
```

Split the data randomly into train and test dataset

```
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(top_ratings_data, test_size = 0.30,
                                         random_state = 0)
```

```
test_data.head()
```

	userID	productID	ratings
6562653	AWH2AY17ZU7W2	B009A6CZ30	5.0
1001830	A1SHHQSPOWR00F	B000HGIWN4	3.0
3904732	A1PVJICI412IN4	B00466X9SY	5.0
7600678	AGYH5U11ZKPFB	B00F3ZN0CC	4.0
2743475	AMKNPIDFLRFMP	B002O3W2OI	2.0

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

Build Popularity Recommender model

```
## building the recommendations based on the average of all user ratings for each product.
numeric = data.select_dtypes(include='number').columns
train_data_grouped = train_data.groupby('productID')[numeric].mean().reset_index()
```

```
train_data_grouped.head()
```

	productID	ratings
0	0972683275	4.5
1	1400501466	3.0
2	1400501520	5.0
3	1400501776	4.0
4	1400532620	1.0

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

```
train_data_sort = train_data_grouped.sort_values(['ratings','productID'],ascending = False)
```

```
train_data_sort.head()
```

	productID	ratings
14854	B00L3YHF6O	5.0
14851	B00K7O2DJU	5.0
14850	B00K4VQZCM	5.0
14849	B00K0OBEE2	5.0
14845	B00JLADOGW	5.0

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

```
train_data.groupby('productID')['ratings'].count().sort_values(ascending = False).head(10)
```

productID	ratings
B0088CJT4U	135
B003ES5ZUU	128
B007WTAJTO	123
B000N99BBC	122
B00829TIEK	102
B008DWCRQW	102
B00829THK0	98
B002R5AM7C	94
B004CLYEDC	82
B004CLYEFK	76

dtype: int64

```
ratings_mean_count = pd.DataFrame(train_data.groupby('productID')['ratings'].mean())
```

```
ratings_mean_count['rating_counts'] = pd.DataFrame(train_data.groupby('productID')['ratings'].count())
```

```
ratings_mean_count.head()
```

productID	ratings	rating_counts
0972683275	4.5	2
1400501466	3.0	4
1400501520	5.0	1
1400501776	4.0	1
1400532620	1.0	1

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

```
pred_data = test_data[['userID','productID','ratings']]
```

```
pred_data.rename(columns = {'ratings':'true_ratings'},inplace = True)
```

```
pred_data = pred_data.merge(train_data_sort, left_on='productID', right_on = 'productID')
```

```
pred_data.rename(columns = {'ratings' : 'predicted_ratings'}, inplace = True)
pred_data.head()
```

	userID	productID	true_ratings	predicted_ratings
0	AWH2AY17ZU7W2	B009A6CZ30	5.0	4.5
1	A316XO4RWX21YN	B009A6CZ30	4.0	4.5
2	A1UNJ46NSB352E	B009A6CZ30	5.0	4.5
3	A1CMD08Z49PGKQ	B009A6CZ30	5.0	4.5
4	A1F9Z42CFF9IAY	B009A6CZ30	5.0	4.5

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

```
import sklearn.metrics as metric
from math import sqrt
MSE = metric.mean_squared_error(pred_data['true_ratings'], pred_data['predicted_ratings'])
print('The RMSE value for Popularity Recommender model is', sqrt(MSE))
```

```
The RMSE value for Popularity Recommender model is 1.0914119718039657
```

The RMSE value for Popularity Recommender model is 1.091

Build Collaborative Filtering model

```
import surprise
from surprise import KNNWithMeans
from surprise.model_selection import GridSearchCV
from surprise import Dataset
from surprise import accuracy
from surprise import Reader
from surprise.model_selection import train_test_split
```

```
reader = Reader(rating_scale=(0.5, 5.0))
```

** Converting Pandas Dataframe to Surpise formatNew Section**

```
data = Dataset.load_from_df(top_ratings_data[['userID', 'productID', 'ratings']],reader)
```

```
# Split data to train and test
from surprise.model_selection import train_test_split
trainset, testset = train_test_split(data, test_size=.3,random_state=0)
```

```
type(trainset)
```

```
surprise.trainset.Trainset
def __init__(ur, ir, n_users, n_items, n_ratings, rating_scale, raw2inner_id_users,
raw2inner_id_items)
    ur(:obj:`defaultdict` of :obj:`list`): The users ratings. This is a
        dictionary containing lists of tuples of the form ``(item_inner_id,
        rating)``. The keys are user inner ids.
    ir(:obj:`defaultdict` of :obj:`list`): The items ratings. This is a
        dictionary containing lists of tuples of the form ``(user_inner_id,
        rating)``. The keys are item inner ids.
    n_users: Total number of users :math:`|U|`.
```

Training the model

KNNWithMeans

```
algo_user = KNNWithMeans(k=10, min_k=6, sim_options={'name': 'pearson_baseline', 'user_based': True})
algo_user.fit(trainset)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNWithMeans at 0x7bb7d2133580>
```

SVD

```
svd_model = SVD(n_factors=50,reg_all=0.02)
svd_model.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7bb7d2131c00>
```

*Evaluate both the models.(Once the model is trained on the training data, it can be used to compute the error (like RMSE)on predictions made on the test data.) We can also use a different method to evaluate the models. *

Popularity Recommender Model RMSE

```
MSE = metric.mean_squared_error(pred_data['true_ratings'], pred_data['predicted_ratings'])
print('The RMSE value for Popularity Recommender model is', sqrt(MSE))
```

```
The RMSE value for Popularity Recommender model is 1.0914119718039657
```

Collaborative Filtering Recommender Model (RMSE)

```
print(len(testset))
type(testset)
```

```
23755
list
```

KNNWithMeans

```
# Evalute on test set
test_pred = algo_user.test(testset)
test_pred[0]
```

```
Prediction(uid='A28UMA3GW9L124', iid='B001GX6MJ8', r_ui=3.0, est=3.3492063492063493, details={'actual_k': 2, 'was_impossible': False})
```

```
# compute RMSE
accuracy.rmse(test_pred) #range of value of error
```

```
RMSE: 0.9941
0.9940800621800723
```

SVD

```
test_pred = svd_model.test(testset)


# compute RMSE
accuracy.rmse(test_pred)
```

```
RMSE: 0.9609
0.9609361199857981
```

Parameter tuning of SVD Recommendation system

```
from surprise.model_selection import GridSearchCV
param_grid = {'n_factors' : [5,10,15], "reg_all":[0.01,0.02]}
gs = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=3,refit = True)


gs.fit(data)


# get best parameters
gs.best_params
```

```
{'rmse': {'n_factors': 5, 'reg_all': 0.01}}
```

```
# Use the "best model" for prediction
gs.test(testset)
accuracy.rmse(gs.test(testset))
```

```
RMSE: 0.8565
0.8564850433826154
```

The RMSE value for Collaborative Filtering model, byKNNWithMeans is 0.9941 and SVD is 0.9606. After parameter tuning of SVD it is 0.858

Get top -K(K= 5)recommendations. We will recommend 5 new products.

```
from collections import defaultdict
def get_top_n(predictions, n=5):

    # First map the predictions to each user.
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the k highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n


top_n = get_top_n(test_pred, n=5)



 #Print the recommended items for each user
for uid, user_ratings in top_n.items():
    print(uid, [iid for (iid, _) in user_ratings])
```

```
A28UMA3GW9L124 ['B008JJLW4M', 'B000VX6XL6', 'B0002D6QJO', 'B008HO9DK4', 'B00HFRWWAM']
A38NHXL5257E3B ['B004Q3R9AQ', 'B00BOHNYU6', 'B00109Y2DQ', 'B003SGCO3E', 'B00CB2F65O']
A36IHC0K68NS2 ['B00HMREOLK', 'B00ATM1MGA', 'B004ING996', 'B001FAACHK', 'B005SXT6TA']
A231WM2Z2JL0U3 ['B00008OE6I', 'B00005LB8P', 'B00004RC2D', 'B00008OE5G', 'B00006HZ0L']
A2AC6GQ24S45GA ['B001S2RCWI', 'B009E6J1BU', 'B009VV56TY', 'B001TOD7ME', 'B000PGHCG4']
A3OXHLG6DIBRW8 ['B001T9NUJE', 'B004CLYEFK', 'B002VPE1X4', 'B001ID829O', 'B001TH7GVE']
A2XA8CW5DF4MNZ ['B000TKHBDK', 'B00387EW1K', 'B001TOD7ME', 'B005BCCML2', 'B000AP05BO']
AG35JCCQWDRCS ['B000068O16', 'B0036Q7MV0', 'B0045TYDNC', 'B001MSVPM6', 'B00AXTQQDS']
A2X3L31KCXBHCL ['B00829THEG', 'B0044DEDC0', 'B000B63KSM', 'B006WHPQE0', 'B002V1APJ2']
A33YZNZIRA3H97 ['B00BOHNYTW', 'B00HFRWWAM', 'B00AXTQQDS', 'B00BFO14W8', 'B00E8CF0CE']
AX05DBU8IRUWY ['B007WTAJTO', 'B006W8U2MU', 'B008D6YZXG', 'B00IKCQ0EK', 'B00FISD9DO']
A296QED1MV1V0J ['B007VGGFZU', 'B008AST7R6', 'B0058G40O8', 'B005BOMTT0', 'B00AXTQQDS']
A1CPRP3VFJRS1R ['B003CFATT2', 'B008C1JC4O', 'B00080O0UI', 'B002HWRJY4', 'B004HO58SO']
AB094YABX21WQ ['B001TH7GUU', 'B00483WRZ6', 'B000O5N5AI', 'B000JMJWV2', 'B0023Y9EQC']
A1MEISNED4NP7U ['B007R5YDYA', 'B00834SJSK', 'B00483WRZ6', 'B007PTCFFW', 'B009NHAEXE']
```

```
A3LGT6UZL99IW1 ['B004W2JKWG', 'B000VUIXOO', 'B004CLYEH8', 'B008HK50ZA', 'B004CLYEDC']
A1PPS91NLI7KEH ['B00AAHT8JC', 'B00CD8ADKO', 'B0064EL2DK', 'B0098PRKA6', 'B00825BZUY']
A1VJ0V58N0698J ['B000QY9KIS', 'B004TB70Y0', 'B0075SUK14', 'B004VM0SE6', 'B000MUV6BA']
A196JN53PG0C7R ['B003ES5ZUU', 'B008JJLW4M', 'B001TH7T2U', 'B001FVI91U', 'B001AYGDCE']
A1901NTE8LFJF6 ['B000M2TAN4', 'B0011U65F2', 'B005DKZTMG', 'B007ABANFY', 'B002HWRJBM']
A3TAS1AG6FMBQW ['B00COF7DGS', 'B009NHWVIA', 'B002WE6D44', 'B001TH7GSW', 'B00D6XW62I']
A362FM6FYA1SYS ['B007WTAJTO', 'B005ES0YYA', 'B0099XGZXA', 'B000VDCT3C', 'B00E3W15P0']
A3D0UM4ZD2CMAW ['B004W2T2TM', 'B009O7YU56', 'B004FA8NOQ', 'B005J7YA4G', 'B003N8GVUY']
A6FIAB28IS79 ['B004YKXGIK', 'B00006I5WJ', 'B0048IATQ0', 'B004VM1T5S', 'B0097BEF1S']
A2X695AM08AIN1 ['B002WE6D44', 'B004C3AW40', 'B007WTAJTO', 'B0075SUK14', 'B003YKG2UK']
A2NOW4U7W3F7RI ['B004IK2EAW', 'B00017LSPI', 'B007WTAJTO', 'B0027Q4HXG', 'B00AE0IEHM']
A2W2ODLRQ2L8LE ['B000V7AF8E', 'B0009PTBZ6', 'B006FNCWSY', 'B004JQN670', 'B000LD14PQ']
AR6APXLK7TJU2 ['B000AZ57M6', 'B003SX0P1A', 'B0011TM19C', 'B002J9HBSE', 'B004FEEZHQ']
A1V4A5U5O3TMMD ['B000F2BLTM', 'B00009XVCZ', 'B001EAQTRI', 'B007FGYZFI', 'B0002E1RZQ']
A110PC8C5Y7MQD ['B005EOWBKE', 'B005DQG5SC', 'B005J4C820', 'B002ZVCGXQ', 'B0013J0502']
A5MCDQ60DWUEV ['B002WE6D44', 'B001MSU1FS', 'B0007U00X0', 'B002ZIMEMW', 'B000VZS2EU']
ASCBJEPXTOU0V ['B004W2JKWG', 'B003ES5ZUU', 'B000QUUFRW', 'B0088PUEPK', 'B00212NO6W']
A2294LS59GC5K7 ['B002V88HFE', 'B008FJJ66C', 'B001J8BPYM', 'B0093HGD2K', 'B006G5ZVA2']
A34UVV757IKPVB ['B003XM1WE0', 'B000EPHR0C', 'B008MF3X9K', 'B005I6EU48', 'B000F49RAA']
A10O7THJ2O20AG ['B007PJ4P4G', 'B000ABB4HC', 'B00005N6KG', 'B00017LSPI', 'B000QUUFRW']
A2NYK9KWFMJV4Y ['B000VX6XL6', 'B00HG1L334', 'B00DTZYHX4', 'B009UNZ5WQ', 'B009ZIILLI']
A3TBMAWIIHKHFN ['B000WOVD1Y', 'B000MVBHRW', 'B004YIFKRM', 'B000167OAC', 'B0015HYPOO']
A1ODOGXEYECQQ8 ['B000233WJ6', 'B00109Y2DQ', 'B001RCTA8I', 'B008AST7R6', 'B001TH7GUK']
A1MFVAHTT2BHM0 ['B007WTAJTO', 'B005NGKR54', 'B000S5Q9CA', 'B00005T3G0', 'B0015DYMVO']
A18HE80910BTZI ['B00EO302SO', 'B002WE4HE2', 'B0001G6U4S', 'B004G6002M', 'B003LPUWT0']
AEZJTA4KDIWY8 ['B00E3W15P0', 'B000HPV3RW', 'B001ID829O', 'B0044XTJ10', 'B000O8I474']
A19EKT8H85AKO5 ['B004CLYEFK', 'B00834SJSK', 'B0079UAT0A', 'B005XPFYBM', 'B0013VFI34']
A141OPVE376YFI ['B0000BZL1P', 'B000067RT6', 'B003YKFKR6', 'B0056YNA1Q', 'B000EUFJXE']
A17HMM1M7T9PJ1 ['B0000BZL1P', 'B0010Z28XG', 'B001KELVS0', 'B0012IJYZ6', 'B0049CPQ36']
A28RSMADFCBJDT ['B000M2TAN4', 'B000M2GYF6', 'B000N7VPRW', 'B005DKZTMG', 'B001XCX9V6']
AV9PIER7NE448 ['B00E3W15P0', 'B0045TYDNC', 'B003X26VV4', 'B009NHAEXE', 'B000HPV3RW']
ADOR3TR7GDF68 ['B007WTAJTO', 'B002YIG9AQ', 'B008MUDGBA', 'B009XN9GQY', 'B00FFJ0HUE']
A316XO4RWX21YN ['B00D5Q75RC', 'B00FPKDPZC', 'B002S53LJ2', 'B00ATM1MVU', 'B001CROHX6']
ARBKYIVNYWK3C ['B0052SCU8U', 'B00006HSML', 'B006EKIQYM', 'B000TKHBDK', 'B00007IFED']
A1AQ8JT2A3UWMY ['B000NP3DJW', 'B00E3W15P0', 'B000NP46K2', 'B003VAK16O', 'B00004RC2D']
A680RUE1FDO8B ['B004CLYEH8', 'B005LDLP8W', 'B00E0HITQ6', 'B00119T6NQ', 'B0019EHU8G']
A38NELQT98S4H8 ['B00DT04I9W', 'B004MF11MU', 'B0041D81WQ', 'B006BUN5YQ', 'B008DWCRQW']
A26SO3TOT2TLJE ['B00BOHNYU6', 'B008PQAFR4', 'B00ATM1MHO', 'B00CD1FB26', 'B00EZ9XG5I']
A3TR3KLL5PXSZ8 ['B004WB8EYM', 'B000A1SZ2Y', 'B005KEL4NI', 'B0048IW030', 'B003F2DA2A']
A3AYSYSLHU26U9 ['B0019EHU8G', 'B00IX2VGFA', 'B001U3ZH7W', 'B00B9KOCYA', 'B001FA1NZK']
A13BX9O5UDBILC ['B002V88HFE', 'B00015GYU4', 'B002K9IHGI', 'B001XHBNN2', 'B009YQ8BTI']
AVFJ327UXPXLF ['B00BOHNYU6', 'B006K553LU', 'B0039BPG1U', 'B006LW0W5Y', 'B0036VO6IC']
```

Summary

Build Popularity Recommender model and found the RMSE value for Popularity Recommender model as 1.091

Build Collaborative Filtering model.The RMSE value for Collaborative Filtering model, byKNNWithMeans is 0.9941 and SVD is 0.9606. After parameter tuning of SVD it is 0.858

Between RMSE of Popularity and Collaborative filtering , Collaborative fitering fares better with 0.86 scores.