```
# Load necessary libraries

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

import warnings

warnings.filterwarnings('ignore')
%matplotlib inline
from surprise import KNNBasic, SVD, NormalPredictor, KNNBaseline,KNNWithMeans, KNNWithZScore, BaselineOnly, CoClustering, Reader, dataset, accuracy
# Read and explore the dataset  ( Rename column/add headers, plot histograms, find data characteristics)
columns = ['userID', 'productID', 'ratings','timestamp']

recomm_df = pd.read_csv('ratings_Electronics.csv',names=columns)
recomm_df.info()
recomm_df.head()
recomm_df.shape
recomm_df.describe().T
#  Dropping the "timestamp" as it is not a needed field
recomm_df = recomm_df.drop('timestamp', axis=1)
# Missing Value
recomm_df.isna().sum()
recomm_df.shape
# plot histograms
```

```
recomm_df.hist('ratings',bins = 10)
popular = recomm_df[['userID','ratings']].groupby('userID').sum().reset_index()
popular_20 = popular.sort_values('ratings', ascending=False).head(n=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('Most popular')

plt.show()
# find unique users
recomm_df.userID.value_counts()
print('Number of unique users', len(recomm_df['userID'].unique()))
print('Number of unique products', len(recomm_df['productID'].unique()))
print('Unique Ratings', recomm_df['ratings'].unique())
min_ratings1 = recomm_df[(recomm_df['ratings'] < 2.0)]
print('Number of unique products rated low',len(min_ratings1['productID'].unique()))
med_ratings1 = recomm_df[(recomm_df['ratings'] > 2.0) & (recomm_df['ratings'] < 4.0)]
print('Number of unique products rated medium',len(med_ratings1['productID'].unique()))
max_ratings1 = recomm_df[recomm_df['ratings'] >= 4.0]
print('Number of unique products rated high',len(max_ratings1['productID'].unique()))
avg_rating_prod = recomm_df.groupby('productID').sum() / recomm_df.groupby('productID').count()
```

```
avg_rating_prod.drop('userID', axis=1,inplace =True)

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

# Take a subset of the dataset to make it less sparse/ denser. ( For example, keep the users only who has given 50 or more number of ratings )

userID = recomm_df.groupby('userID').count()

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


topuser_ratings_df = recomm_df[recomm_df['userID'].isin(top_user)]

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

topuser_ratings_df.shape

topuser_ratings_df.head()

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


# Keep data only for products that have 50 or more ratings

prodID = recomm_df.groupby('productID').count()

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

top_ratings_df = topuser_ratings_df[topuser_ratings_df['productID'].isin(top_prod)]

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

top_ratings_df.shape

# Split the data randomly into train and test dataset. ( For example, split it in 70/30 ratio)
```

```
from sklearn.model_selection import train_test_split

train_data, test_data = train_test_split(top_ratings_df, test_size = 0.30, random_state=0)

train_data.head()

test_data.head()

# Build Popularity Recommender model.

#Building the recommendations based on the average of all user ratings for each product.
```

```
train_data_grouped = train_data.groupby('productID').mean().reset_index()

train_data_grouped.head()

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

train_data_sort.head()

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

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()

pred_df = test_data[['userID', 'productID', 'ratings']]

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

pred_df = pred_df.merge(train_data_sort, left_on='productID', right_on = 'productID')

pred_df.head(3)

pred_df.rename(columns = {'ratings' : 'predicted_ratings'}, inplace = True)

pred_df.head()

import sklearn.metrics as metric

from math import sqrt

MSE = metric.mean_squared_error(pred_df['true_ratings'], pred_df['predicted_ratings'])

print('The RMSE value for Popularity Recommender model is', sqrt(MSE))
```

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 format
```

data = Dataset.load_from_df(top_ratings_df[['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)

Training the model

KNNWithMeans

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

algo_user.fit(trainset)

SVD

svd_model = SVD(n_factors=50,reg_all=0.02)

svd_model.fit(trainset)

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.) You can also use a different method to evaluate the models.

Popularity Recommender Model (RMSE)

MSE = metric.mean_squared_error(pred_df['true_ratings'], pred_df['predicted_ratings'])

print('The RMSE value for Popularity Recommender model is', sqrt(MSE))

Collaborative Filtering Recommender Model (RMSE)

print(len(testset))

type(testset)

KNNWithMeans

Evalute on test set

test_pred = algo_user.test(testset)

test_pred[0]

compute RMSE

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

SVD

```
test_pred = svd_model.test(testset)

# compute RMSE

accuracy.rmse(test_pred)
```

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

# Use the "best model" for prediction

gs.test(testset)

accuracy.rmse(gs.test(testset))
```

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. Since our goal is to recommend new products to each user based on his/her habits, 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)
```

.