1. Import the libraries

```
!pip install matplotlib-venn
```

```
!apt-get -qq install -y libfluidsynth1
```

```
!apt-get -qq install -y libarchive-dev && pip install -U libarchive
import libarchive
```

```
# https://pypi.python.org/pypi/pydot
!apt-get -qq install -y graphviz && pip install pydot
import pydot
```

```
!pip install cartopy
import cartopy
```

```
!curl https://sdk.cloud.google.com | bash
```

```
#here we have to connect with google cloud with lohin of teamunt2023@gmail.com
from google.colab import auth
auth.authenticate_user()
```

```
!pip install google-cloud-storage
```

```
#here we should call the lib for google cloud for access of storage
from google.cloud import storage
import pandas as pd

client = storage.Client()
bucket = client.get_bucket('teamunt_1')
```

```
blob = bucket.blob('https://storage.cloud.google.com/teamunt_1/ratings_Electronics%20(1).csv')
```

```
print(blob)
```

```
    <Blob: teamunt_1, https://storage.cloud.google.com/teamunt_1/ratings_Electronics%20(1).csv, None>
```

```
#import the reqired libraries
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import numpy as np
import pandas as pd
import math
import json
import time
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors
#from sklearn.externals import joblib
import scipy.sparse
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds
import warnings; warnings.simplefilter('ignore')
%matplotlib inline
```

2. Load the dataset and add headers

```
# Import the dataset and give the column names
columns=['userId', 'productId', 'ratings','timestamp']
electronics_df=pd.read_csv('https://storage.cloud.google.com/teamunt_1/ratings_Electronics%20(1).csv',names=columns)


#electronics_df.head()
```

Dropping the timestamp column

```
electronics_df.drop('timestamp',axis=1,inplace=True)


electronics_df.info()


#Check the number of rows and columns
rows,columns=electronics_df.shape
print('Number of rows: ',rows)
print('Number of columns: ',columns)
```

```
Number of rows:  119
Number of columns:  3
```

```
#Check the datatypes
electronics_df.dtypes


#Taking subset of the dataset
electronics_df1=electronics_df.iloc[:50000,0:]
```

- Since the data is very big. Consider electronics_df1 named dataframe with first 50000 rows and all columns from 0 of dataset.

```
electronics_df1.info()


#Summary statistics of rating variable
electronics_df1['ratings'].describe().transpose()
```

- Rating are on the scale 1 to 5.

Handling Missing values

```
#Check for missing values
print('Number of missing values across columns: \n',electronics_df.isnull().sum())
```

- There are no missing records in the dataset.

Ratings

```
# Check the distribution of the rating
with sns.axes_style('white'):
    g = sns.factorplot("ratings", data=electronics_df1, aspect=2.0,kind='count')
    g.set_ylabels("Total number of ratings")
```

- We can see that more number of users have given the rating of 5.

Users and products

```
# Number of unique user id  in the data
print('Number of unique users in Raw data = ', electronics_df1['userId'].nunique())
# Number of unique product id  in the data
print('Number of unique product in Raw data = ', electronics_df1['productId'].nunique())
```

3. Taking the subset of dataset to make it less sparse/ denser.

```
#Check the top 10 users based on ratings
most_rated=electronics_df1.groupby('userId').size().sort_values(ascending=False)[:10]
print('Top 10 users based on ratings: \n',most_rated)
```

```
counts=electronics_df1.userId.value_counts()
electronics_df1_final=electronics_df1[electronics_df1.userId.isin(counts[counts>=15].index)]
print('Number of users who have rated 25 or more items =', len(electronics_df1_final))
print('Number of unique users in the final data = ', electronics_df1_final['userId'].nunique())
print('Number of unique products in the final data = ', electronics_df1_final['userId'].nunique())
```

- electronics_df1_final has the users who have rated 25 or more items.

ratings analysis in final dataset

```
#constructing the pivot table
final_ratings_matrix = electronics_df1_final.pivot(index = 'userId', columns ='productId', values = 'ratings').fillna(0)
final_ratings_matrix.head()
```

- It shows that it is a sparse matrix. So, many cells are filled with 0 values.

```
print('Shape of final_ratings_matrix: ', final_ratings_matrix.shape)
```

- We can see that there are 7 products and 236 users.

```
#Calucating the density of the rating marix
given_num_of_ratings = np.count_nonzero(final_ratings_matrix)
print('given_num_of_ratings = ', given_num_of_ratings)
possible_num_of_ratings = final_ratings_matrix.shape[0] * final_ratings_matrix.shape[1]
print('possible_num_of_ratings = ', possible_num_of_ratings)
density = (given_num_of_ratings/possible_num_of_ratings)
density *= 100
print ('density: {:4.2f}%'.format(density))
```

- The density value of the matrix also shows that it is a sparse matrix.

4. Splitting the data

```
#Split the data randomnly into train and test datasets into 70:30 ratio
train_data, test_data = train_test_split(electronics_df1_final, test_size = 0.3, random_state=0)
train_data.head()
```

```
print('Shape of training data: ',train_data.shape)
print('Shape of testing data: ',test_data.shape)
```

5. Building Popularity Recommder model

```
#Count of user_id for each unique product as recommendation score
train_data_grouped = train_data.groupby('productId').agg({'userId': 'count'}).reset_index()
train_data_grouped.rename(columns = {'userId': 'score'},inplace=True)
train_data_grouped.head(40)
```

```
#Sort the products on recommendation score
train_data_sort = train_data_grouped.sort_values(['score', 'productId'], ascending = [0,1])

#Generate a recommendation rank based upon score
train_data_sort['rank'] = train_data_sort['score'].rank(ascending=0, method='first')

#Get the top 5 recommendations
popularity_recommendations = train_data_sort.head(5)
popularity_recommendations


# Use popularity based recommender model to make predictions
def recommend(user_id):
    user_recommendations = popularity_recommendations

    #Add user_id column for which the recommendations are being generated
    user_recommendations['userId'] = user_id

    #Bring user_id column to the front
    cols = user_recommendations.columns.tolist()
    cols = cols[-1:] + cols[:-1]
    user_recommendations = user_recommendations[cols]

    return user_recommendations


find_recom = [10,100,150]   # This list is user choice.
for i in find_recom:
    print("The list of recommendations for the userId: %d\n" %(i))
    print(recommend(i))
    print("\n")
```

- Since, it is a Popularity recommender model, so, all the three users are given the same recommendations. Here, we predict the products based on the popularity. It is not personalized to particular user. It is a non-personalized recommender system.

6. Building Collaborative Filtering recommender model.

```
electronics_df_CF = pd.concat([train_data, test_data]).reset_index()
electronics_df_CF.head()
```

User Based Collaborative Filtering model

```
# Matrix with row per 'user' and column per 'item'
pivot_df = electronics_df_CF.pivot(index = 'userId', columns ='productId', values = 'ratings').fillna(0)
pivot_df.head()
```

```
print('Shape of the pivot table: ', pivot_df.shape)
```

```
    Shape of the pivot table:  (11, 186)
```

```
#define user index from 0 to 10
pivot_df['user_index'] = np.arange(0, pivot_df.shape[0], 1)
pivot_df.head()
```

```
pivot_df.set_index(['user_index'], inplace=True)
# Actual ratings given by users
pivot_df.head()
```

- As this is a sparse matrix we will use SVD.

Singular Value Decomposition

```
# Singular Value Decomposition
U, sigma, Vt = svds(pivot_df, k = 10)
```

```
print('Left singular matrix: \n',U)
```

```
print('Sigma: \n',sigma)
```

- As sigma is not a diagonal matrix we have to convert it into diagonal matrix.

```
# Construct diagonal array in SVD
sigma = np.diag(sigma)
print('Diagonal matrix: \n',sigma)
```

```
print('Right singular matrix: \n',Vt)
```

```
#Predicted ratings
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt)
# Convert predicted ratings to dataframe
preds_df = pd.DataFrame(all_user_predicted_ratings, columns = pivot_df.columns)
preds_df.head()
```

```
# Recommend the items with the highest predicted ratings

def recommend_items(userID, pivot_df, preds_df, num_recommendations):
    # index starts at 0
    user_idx = userID-1
    # Get and sort the user's ratings
    sorted_user_ratings = pivot_df.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_ratings
    sorted_user_predictions = preds_df.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_predictions
    temp = pd.concat([sorted_user_ratings, sorted_user_predictions], axis=1)
    temp.index.name = 'Recommended Items'
    temp.columns = ['user_ratings', 'user_predictions']
    temp = temp.loc[temp.user_ratings == 0]
    temp = temp.sort_values('user_predictions', ascending=False)
    print('\nBelow are the recommended items for user(user_id = {}):\n'.format(userID))
    print(temp.head(num_recommendations))
```

```
userID = 4
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

```
userID = 6
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

```
userID = 8
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

- Since, it is a Collaborative recommender model, so, all the three users are given different recommendations based on users past behaviour.

7. Evaluation of Collabrative recommendation model

```
# Actual ratings given by the users
final_ratings_matrix.head()
```

```
# Average ACTUAL rating for each item
final_ratings_matrix.mean().head()
```

```
# Predicted ratings
preds_df.head()
```

```
# Average PREDICTED rating for each item
preds_df.mean().head()
```

```
rmse_df = pd.concat([final_ratings_matrix.mean(), preds_df.mean()], axis=1)
rmse_df.columns = ['Avg_actual_ratings', 'Avg_predicted_ratings']
print(rmse_df.shape)
rmse_df['item_index'] = np.arange(0, rmse_df.shape[0], 1)
rmse_df.head()
```

```
RMSE = round((((rmse_df.Avg_actual_ratings - rmse_df.Avg_predicted_ratings) ** 2).mean() ** 0.5), 5)
print('\nRMSE SVD Model = {} \n'.format(RMSE))
```

8. Getting top - K (K = 5) recommendations.

```
# Enter 'userID' and 'num_recommendations' for the user #
userID = 9
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

Summarising insights.