```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
!pip install surprise
from surprise import SVD, Reader, Dataset
from surprise.model_selection import cross_validate
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: surprise in /usr/local/lib/python3.9/dist-packages (0.1)
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.9/dist-packages (from surprise) (1.1.3)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise->surprise) (1.22.4)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise->surprise) (1.1.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise->surprise) (1.10.1)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv("/content/drive/My Drive/CSP/ratings_Beauty.csv")
```

```
df.head()
```

	UserId	ProductId	Rating	Timestamp
0	A39HTATAQ9V7YF	205616461	5	1369699200
1	A3JM6GV9MNOF9X	558925278	3	1355443200
2	A1Z513UWSAAO0F	558925278	5	1404691200
3	A1WMRR494NWEWV	733001998	4	1382572800
4	A3IAAVS479H7M7	737104473	1	1274227200

```
df.shape
```

```
(1048575, 4)
```

```
df.info()
```

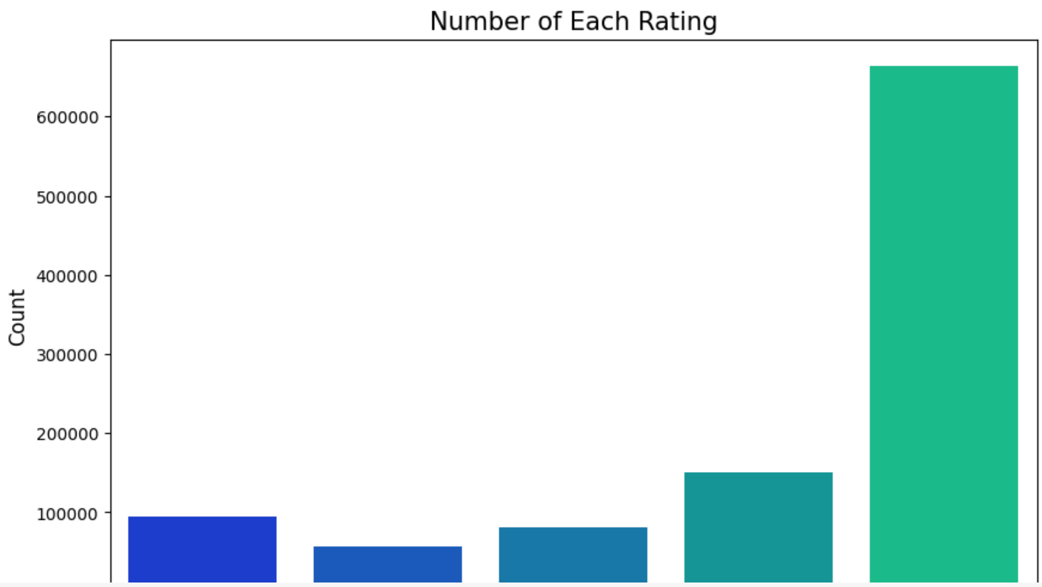
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype 
---  ------     --------------    ----- 
 0   UserId     1048575 non-null  object
 1   ProductId  1048575 non-null  object
 2   Rating     1048575 non-null  int64 
 3   Timestamp  1048575 non-null  int64 
dtypes: int64(2), object(2)
memory usage: 32.0+ MB
```

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

```
UserId       0
ProductId    0
Rating       0
Timestamp    0
dtype: int64
```

We can see that there is no missing data and the data is clean.

```
plt.figure(figsize=(10,6))
sns.countplot(x='Rating', data=df, palette='winter')
plt.xlabel('Rating', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Number of Each Rating', fontsize=15)
plt.show()
```




```
df_rating=pd.DataFrame({'Number of Rating':df.groupby('ProductId').count()['Rating'], 'Mean Rating':df.groupby('ProductId').mean()['Ratir
```

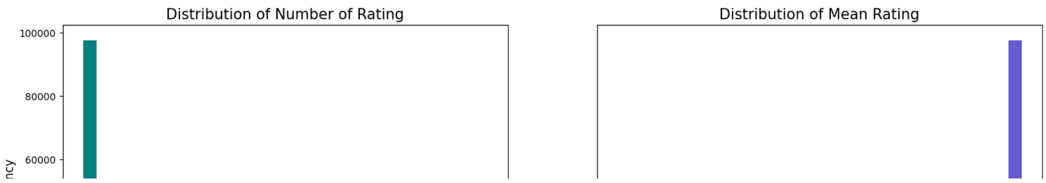
```
df_rating.head()
```

ProductId	Number of Rating	Mean Rating
1304139212	1	5.000000
1304139220	1	5.000000
130414089X	1	5.000000
130414643X	3	4.333333
1304146537	1	5.000000

```
df_rating.shape
```

```
(97987, 2)
```

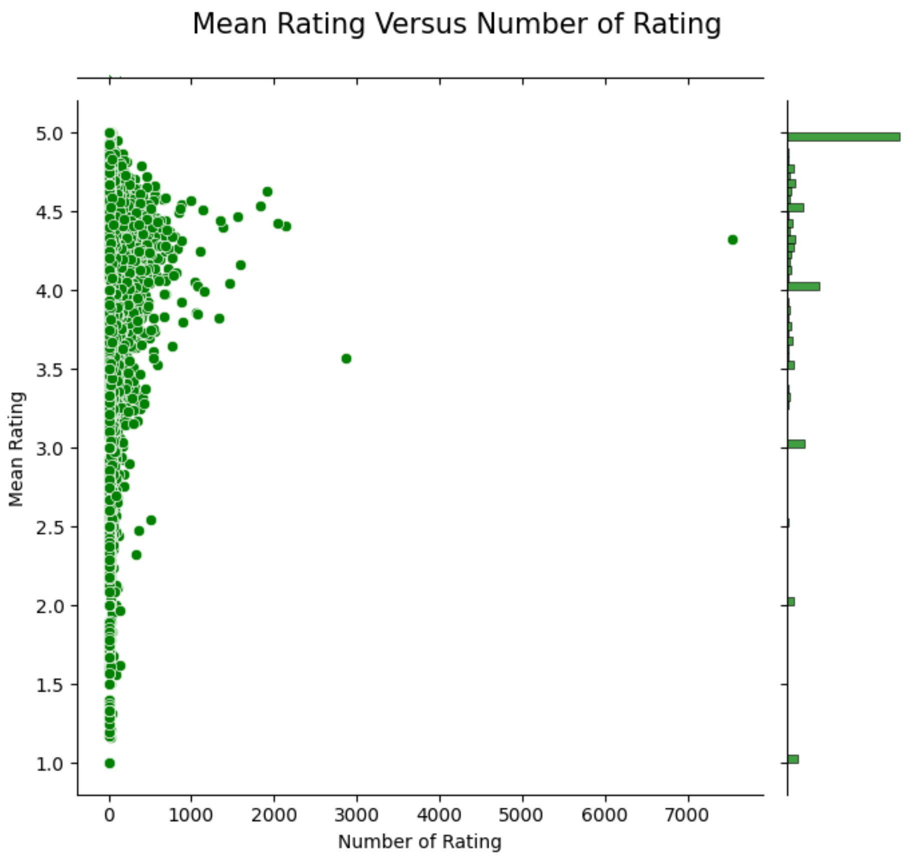
```
plt.figure(figsize=(18,6))

plt.subplot(1,2,1)
plt.hist(x='Number of Rating',data=df_rating,bins=30,color='teal')
plt.title('Distribution of Number of Rating', fontsize=15)
plt.xlabel('Number of Rating', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

plt.subplot(1,2,2)
plt.hist(x='Mean Rating',data=df_rating,bins=30, color='slateblue')
plt.title('Distribution of Mean Rating', fontsize=15)
plt.xlabel('Mean Rating', fontsize=12)
plt.yticks([])
plt.show()
```



From these histograms we can see that most of the number of ratings are between 0 and 1825, and most of the products have a mean rating of 5.

```
plt.figure(figsize=(8,6))
sns.jointplot(x='Number of Rating', y='Mean Rating',data=df_rating,color='g', height=7)
plt.suptitle('Mean Rating Versus Number of Rating', fontsize=15, y=0.92)
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



Popularity-Based Recommender

The implementation of Popularity-Based Filtering is straighforward. All we have to do is sort our products based on ratings, and display the top products of our list. Therefore, we should;

Create a metric to score or rate the products. Calculate the score for every product. Sort the scores and recommend the best rated product to the users. We can use the average ratings of the products as the score but using this will not be fair enough since a product with 5average rating and only43 votes cannot be considered better than the product with 4 as average rating but 40 votes. So, we use IMDB's weighted rating formula to score the products, as follows:

Weighted Rating (WR) = (v/v+m.R)+(m/v+m.C)

v: the number of votes for the product

m: the minimum votes required to be listed in the chart

R: the average rating of the product

C: the mean vote across the whole report

Based on the df_rating dataframe created above, we already have v or the Number of Rating, and R or Mean Rating for each product. So we calculate C.

```
df_rating['Mean Rating'].mean()
```

```
4.182085565717771
```

The mean rating for all the products (C) is approximately 3.9 on a scale of 5.

The next step is to determine an appropriate value for m, the minimum number of votes required for a product to be listed in the chart. We use 90th percentile as our cutoff. In other words, for a product to feature in the charts, the number of its votes should be higher than that of 90% of the products in the list.

```
df_rating['Number of Rating'].quantile(q=0.9)
```

```
21.0
```

Now, we filter the products that qualify for the chart and put them in a new dataframe called df_filtered.

```
df_filtered=df_rating[df_rating['Number of Rating']>df_rating['Number of Rating'].quantile(q=0.89)]
```

```
df_filtered.shape
```

```
(10457, 2)
```

Now, we calculate score for each qualified product. To do this, we define a function, weighted_rating(), and apply this function to the DataFrame of qualified products.

```
def product_score(x):
    v=x['Number of Rating']
    m=df_rating['Number of Rating'].quantile(q=0.9)
    R=x['Mean Rating']
    C=df_rating['Mean Rating'].mean()
    return ((R*v)/(v+m))+((C*m)/(v+m))
```

```
df_filtered['score']=df_filtered.apply(product_score, axis=1)
```

```
<ipython-input-55-f2b417ef47b5>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df_filtered['score']=df_filtered.apply(product_score, axis=1)
```

```
df_filtered.head()
```

	Number of Rating	Mean Rating	score
ProductId			
3227001381	25	4.560000	4.387474
7806397051	35	3.285714	3.621854
9746427962	41	4.609756	4.464900
9759091062	40	3.125000	3.488915
9788071198	36	3.833333	3.961821

Finally, we sort the dataframe based on the score feature, and we output the the top 10 popular products.

```
df_highscore=df_filtered.sort_values(by='score', ascending=False).head(10)
```

```
df_highscore
```

	Number of Rating	Mean Rating	score
ProductId			
B002YFN49I	98	4.948980	4.813645
B001FB5NTG	164	4.865854	4.788237
B000NNDNYY	400	4.792500	4.762052

```
df_highscore.index
```

```
Index(['B002YFN49I', 'B001FB5NTG', 'B000NNDNYY', 'B000127UUA', 'B001EJOPTS',
       'B0027Z2720', 'B000YT5NIG', 'B000YB1XRO', 'B001PX1AIC', 'B001F0RBRE'],
      dtype='object', name='ProductId')
```

B000YT5NIG 95 4.873684 4.748481

So the top 10 popular products that this model will recommend to users include 'B002YFN49I', 'B001FB5NTG', 'B000NNDNYY', 'B000127UUA', 'B001EJOPTS','B0027Z2720', 'B000YT5NIG', 'B000YB1XRO', 'B001PX1AIC', 'B001F0RBRE'.

We should keep in mind that this popularity-based recommender provides a general chart of recommended products to all the users, regardless of the user's personal taste. It is not sensitive to the interests and tastes of a particular user, and it does not give personalized recommendations based on the users.

Collaborative Recommender

SVD: Matrix Factorization Based Algorithm

```
svd = SVD()
```

```
reader = Reader()
```

```
data = Dataset.load_from_df(df[['UserId', 'ProductId', 'Rating']], reader)
```

```
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.2562  1.2569  1.2541  1.2607  1.2574  1.2571  0.0021
MAE (testset)     0.9805  0.9811  0.9791  0.9843  0.9818  0.9813  0.0017
Fit time          26.44   30.55   26.22   27.63   28.11   27.79   1.55
Test time         2.42    2.99    2.51    2.55    2.66    2.63    0.20
{'test_rmse': array([1.25622538, 1.2569127 , 1.25410812, 1.2607116 , 1.25742437]),
 'test_mae': array([0.9804596 , 0.9810761 , 0.97906496, 0.98432047, 0.98179859]),
 'fit_time': (26.442415475845337,
  30.551028966903687,
  26.224515438079834,
  27.633647680282593,
  28.109490871429443),
 'test_time': (2.4183764457702637,
  2.98529314994812,
  2.511044979095459,
  2.547445774078369,
  2.6646618843078613)}
```

We get a mean Root Mean Sqaure Error of 1.25 approx which is good enough for our case. Let us now train on our dataset and arrive at predictions.

```
trainset = data.build_full_trainset()
```

```
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f12690868e0>
```

Let us pick the user with userId of 'A1Z513UWSAAO0F' and check the ratings she/ he has given so far to different products.

```
df[df['UserId'] == 'A1Z513UWSAAO0F']
```

	UserId	ProductId	Rating	Timestamp
2	A1Z513UWSAAO0F	558925278	5	1404691200

As an example, we use the algorithm to predict the score that might be given to the productId of '558925278' by this specific userId.

```
svd.predict(uid='A17HMM1M7T9PJ1', iid='0970407998', r_ui=None)
```

```
Prediction(uid='A17HMM1M7T9PJ1', iid='0970407998', r_ui=None, est=4.176863839019622, details={'was_impossible': False})
```

```
svd.predict(uid='A17HMM1M7T9PJ1', iid='0970407998', r_ui=None).est
```

```
4.176863839019622
```

Our model predicts that userId of 'A17HMM1M7T9PJ1' will give 4.17 as the rating for productId of '0970407998'.

Hybrid Recommender

In this section, we try to build a hybrid recommender that combines corrwith() method which computes the Pearson correlation coefficients with collaborative filtering. This is how it works:

Input: User ID and Product ID

Output: Similar products sorted on the basis of expected ratings by a particular user.

First, we create a pivot table which contains userIds as rows and productIds as columns. However, if we use the whole datafram (df) in the below code, there will be an error because the input data has more than 1 million rows.

The below code will give an error: matrix=pd.pivot_table(data=df, values='rating', index='userId',columns='productId')

To avoid processing difficulties, we filter data which only contains the customers who have given ratings more than 50 times and put them into a dataframe. Since we are providing the recommendation of the products to the customers, it is better to remove data based on the userId rather than productId.

```
df_users=df.groupby('UserId').filter(lambda x: x['Rating'].count()>=50)
```

```
df_users.head()
```

	UserId	ProductId	Rating	Timestamp
184	ACZ94JB8BFMJ9	5357954771	5	1385251200
896	A3KEZLJ59C1JVH	9788073476	5	1335916800
917	A24N4FKHGD7DWT	9788073840	2	1341187200
966	A3KEZLJ59C1JVH	9788074421	5	1351296000
1571	A2C58G8O40YC7T	9790786948	4	1313798400

```
df_users.shape
```

```
(5036, 4)
```

Then we create a pivot table.

```
matrix=pd.pivot_table(data=df_users, values='Rating', index='UserId',columns='ProductId')
```

```
matrix.head()
```

ProductId	5357954771	9788073476	9788073840	9788074421	9790786948	9790794231	979079634X
UserId							
A12PH6L5QSVTYN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
A132ETQPMHQ585	NaN	NaN	NaN	NaN	NaN	NaN	NaN
A19UTUEBWKIZFT	NaN	NaN	NaN	NaN	NaN	NaN	NaN
A1EAX5HVYV96EF	NaN	NaN	NaN	NaN	NaN	NaN	NaN
A1HOPKK8E3MIX6	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 4290 columns

Finally we define a function that takes in productId and useId as input and outputs up to 5 most similar products. For this purpose, we use corrwith() method to compute pairwise correlation between columns of dataFrame and calculate Pearson correlation coefficients.

```
# Function that takes in productId and useId as input and outputs up to 5 most similar products.
def hybrid_recommendations(UserId, ProductId):

    # Get the Id of the top five products that are correlated with the ProductId chosen by the user.
    top_five=matrix.corrwith(matrix[ProductId]).sort_values(ascending=False).head(5)

    # Predict the ratings the user might give to these top 5 most correlated products.
    est_rating=[]
    for x in list(top_five.index):
        if str(top_five[x])!='nan':
            est_rating.append(svd.predict(userId, iid=x, r_ui=None).est)

    return pd.DataFrame({'productId':list(top_five.index)[:len(est_rating)], 'estimated_rating':est_rating}).sort_values(by='estimated_ra
```

```
hybrid_recommendations('A1Z513UWSAAO0F', 'B003G1CE5Y')
```

```
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2821: RuntimeWarning: Degrees of free
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.9/dist-packages/numpy/lib/function_base.py:2680: RuntimeWarning: divide by zero
  c *= np.true_divide(1, fact)
```

productId	estimated_rating