```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt
!pip install scikit-surprise
import seaborn as sns
sns.set_style('white')
%matplotlib inline

# Split
from sklearn.model_selection import train_test_split

from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
Requirement already satisfied: scikit-surprise in
/usr/local/lib/python3.10/dist-packages (1.1.3)
Requirement already satisfied: joblib>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-surprise) (1.3.2)
Requirement already satisfied: numpy>=1.17.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-surprise)
(1.25.2)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/dist-packages (from scikit-surprise)
(1.11.4)
```

```
df = pd.read_csv("ratings_Electronics (1).csv",
                                names=['userId',
'productId','rating','timestamp'])

df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 77618,\n  \"fields\":
[\n    {\n      \"column\": \"userId\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 70713,\n
\"samples\": [\n          \"A3NKX6NIHCC1LS\",\n
\"AFYAUWZ1HKFRJ\",\n          \"A2MTXUIGK2736T\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"productId\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5528,\n        \"samples\": [\n
\"B00004TZKD\",\n          \"B00000J1UW\",\n          \"B00004Z79F\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"rating\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.3391923135876298,\n        \"min\": 1.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1.0,\n
4.0,\n          3.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"timestamp\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 164258524.30111605,\n        \"min\":
912729600.0,\n        \"max\": 1405987200.0,\n
\"num_unique_values\": 5442,\n        \"samples\": [\n
1400544000.0,\n          1059350400.0,\n          1291161600.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
print("Total Reviews:",df.shape[0])
print("Total Columns:",df.shape[1])
```

```
Total Reviews: 77618
Total Columns: 4
```

```
df = df.iloc[:5000,0:]
```

```
print("Total Reviews:",df.shape[0])
print("Total Columns:",df.shape[1])
```

```
Total Reviews: 5000
Total Columns: 4
```

```
print("Total number of ratings :",df.rating.nunique())
print("Total number of users   :", df.userId.nunique())
print("Total number of products  :", df.productId.nunique())
```

```
Total number of ratings : 5
Total number of users   : 4929
Total number of products  : 299
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   userId     5000 non-null   object
 1   productId  5000 non-null   object
 2   rating     5000 non-null   float64
 3   timestamp  5000 non-null   float64
dtypes: float64(2), object(2)
memory usage: 156.4+ KB
```

```
# Check missing value
df.isnull().sum()
```

```
userId       0
productId    0
rating       0
timestamp    0
dtype: int64
```

```
# Check Duplicate data
df[df.duplicated()].any()
```

```
userId       False
productId    False
rating       False
timestamp    False
dtype: bool
```

```
# rating describe summary
df.describe()['rating']
```

```
count    5000.000000
mean        3.986600
std         1.411814
min         1.000000
25%         3.000000
50%         5.000000
75%         5.000000
max         5.000000
Name: rating, dtype: float64
```

```
print("Unique value of Rating:",df.rating.unique())
```

```
Unique value of Rating: [5. 1. 3. 2. 4.]
```

```
# Find the minimum and maximum ratings
print('Minimum rating is: %d' %(df.rating.min()))
print('Maximum rating is: %d' %(df.rating.max()))
```

```
Minimum rating is: 1
Maximum rating is: 5
```

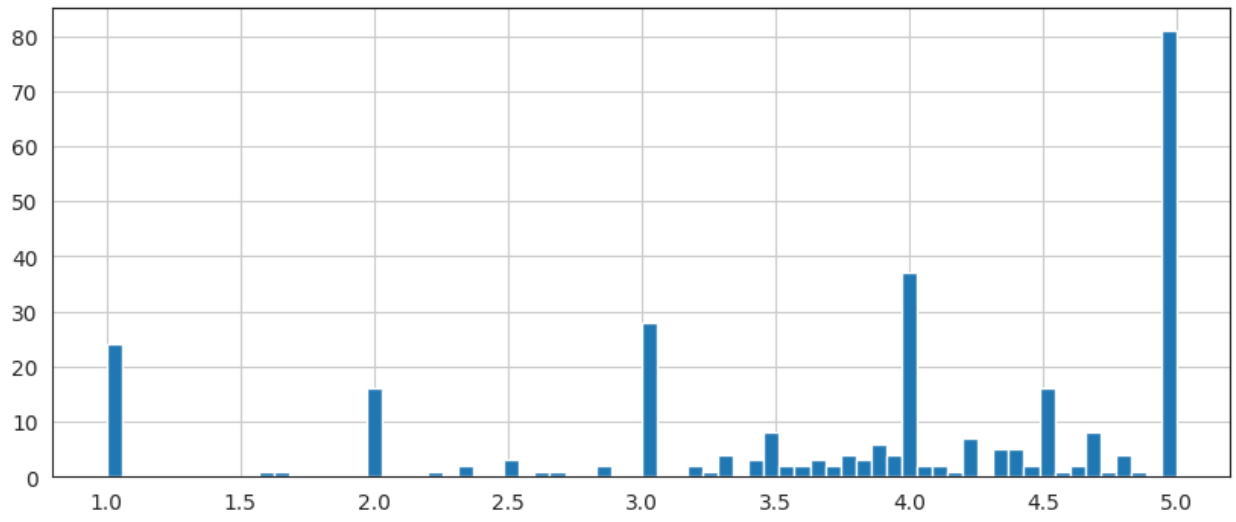
```
# Average rating of products
ratings = pd.DataFrame(df.groupby('productId')['rating'].mean())
ratings['ratings_count'] = pd.DataFrame(df.groupby('productId')
['rating'].count())
ratings['ratings_average'] = pd.DataFrame(df.groupby('productId')
['rating'].mean())
ratings.head(10)
```

```
{"summary":"{\n  \"name\": \"ratings\",\n  \"rows\": 299,\n
\"fields\": [\n    {\n      \"column\": \"productId\",\n
```

```
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 299,\n        \"samples\": [\n
\"9043413585\",\n          \"8862936826\",\n          \"6000012217\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"rating\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.205496806239334,\n        \"min\": 1.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 78,\n        \"samples\": [\n
3.490990990990991,\n          5.0,\n          4.246376811594203\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"ratings_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
76,\n        \"min\": 1,\n        \"max\": 1051,\n
\"num_unique_values\": 46,\n        \"samples\": [\n          19,\n
222,\n          138\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ratings_average\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.205496806239334,\n        \"min\":
1.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 78,\n
\"samples\": [\n          3.490990990990991,\n          5.0,\n
4.246376811594203\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"ratings"}
```

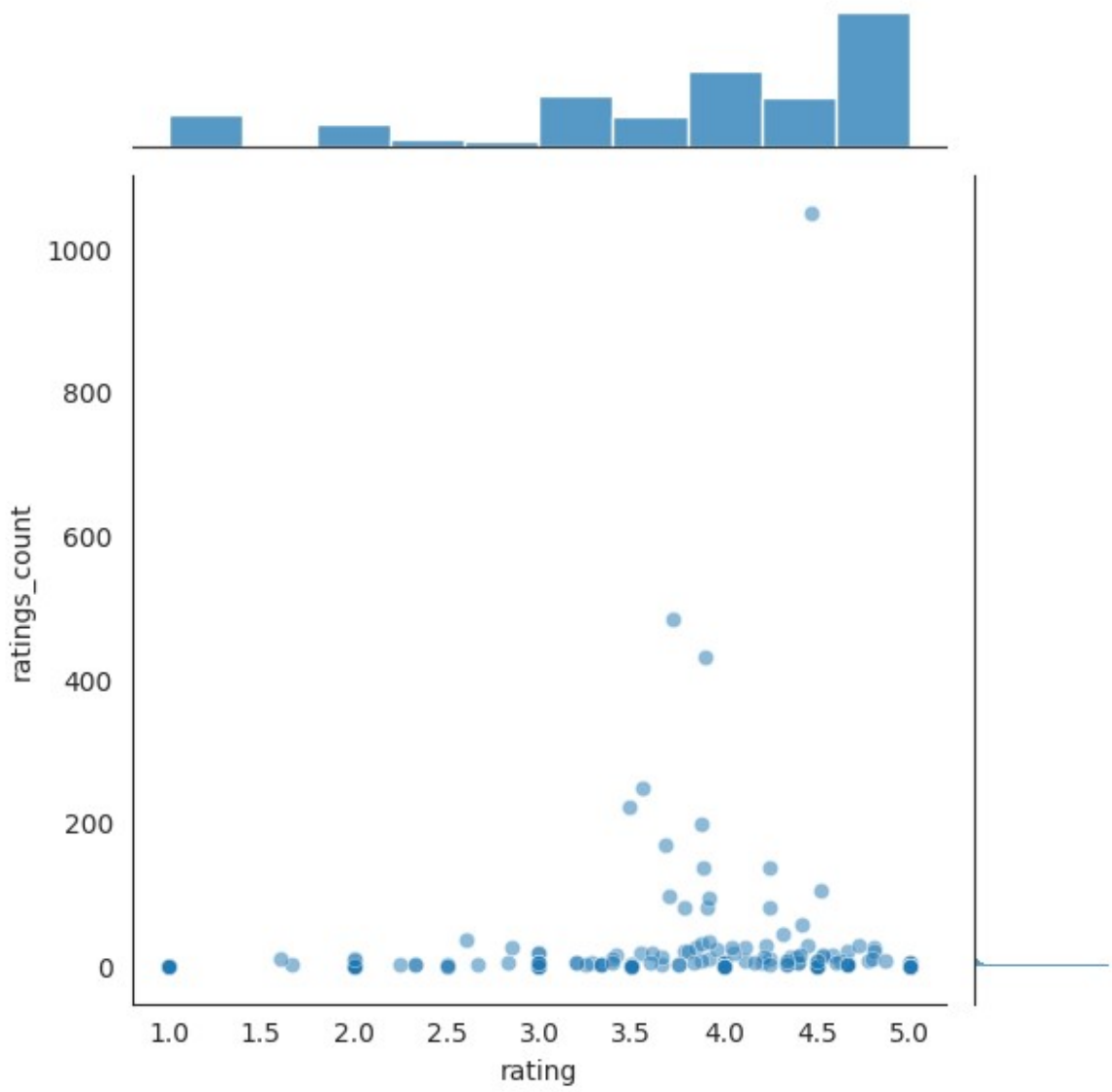

```
plt.figure(figsize=(10,4))
ratings['rating'].hist(bins=70)
```

```
<Axes: >
```



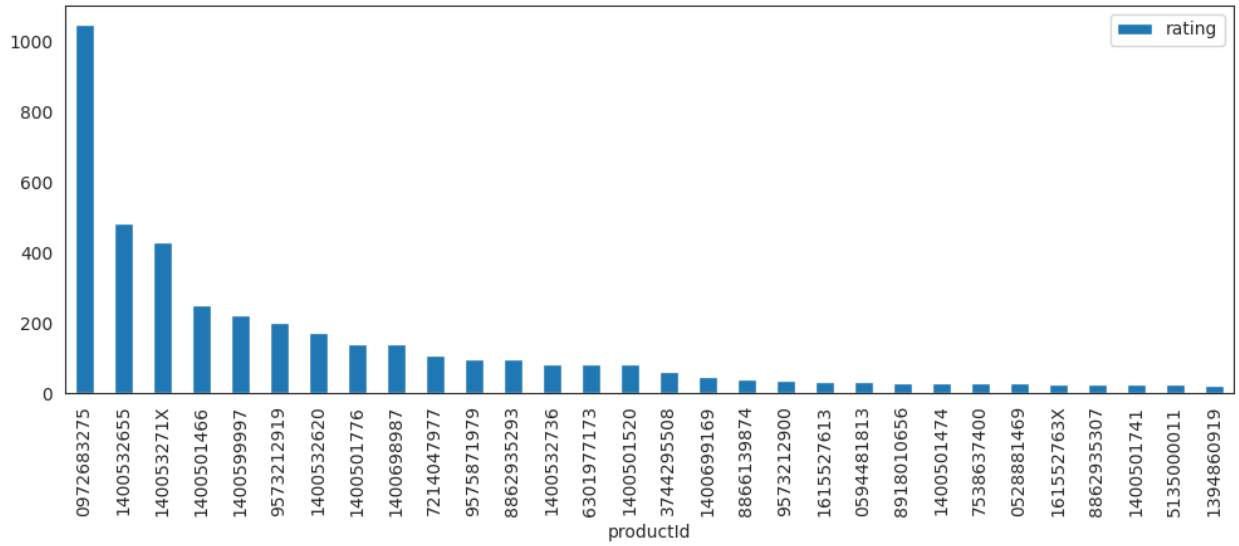

```
sns.jointplot(x='rating',y='ratings_count',data=ratings,alpha=0.5)
```

```
<seaborn.axisgrid.JointGrid at 0x7967b0ceb880>
```




```
# Most top 30 products
popular_products = pd.DataFrame(df.groupby('productId')
['rating'].count())
most_popular = popular_products.sort_values('rating', ascending=False)
most_popular.head(30).plot(kind = "bar",figsize=(12, 4))
```

```
<Axes: xlabel='productId'>
```




```
vote_counts = ratings[ratings['ratings_count'].notnull()]
['ratings_count'].astype('int')
vote_averages = ratings[ratings['ratings_average'].notnull()]
['ratings_average'].astype('int')
C = vote_averages.mean()
print("Average rating of product across the whole dataset is",C)
```

```
Average rating of product across the whole dataset is
3.5953177257525084
```

```
m = vote_counts.quantile(0.95)
print("Minimum votes required to be listed in the chart is",m)
```

```
Minimum votes required to be listed in the chart is 62.19999999999925
```

```
ratings.head()
```


```
{"summary":"{\n  \"name\": \"ratings\",\n  \"rows\": 299,\n
\"fields\": [\n    {\n      \"column\": \"productId\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 299,\n        \"samples\": [\n
\"9043413585\",\n          \"8862936826\",\n          \"6000012217\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"rating\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.205496806239334,\n        \"min\": 1.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 78,\n        \"samples\": [\n
3.490990990990991,\n          5.0,\n          4.246376811594203\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"ratings_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
76,\n        \"min\": 1,\n        \"max\": 1051,\n
\"num_unique_values\": 46,\n        \"samples\": [\n          19,\n
```

```
222,\n          138\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ratings_average\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.205496806239334,\n        \"min\":
1.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 78,\n
\"samples\": [\n          3.490990990990991,\n          5.0,\n
4.246376811594203\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"ratings"}
```

```
qualified = ratings[(ratings['ratings_count'] >= m) &
(ratings['ratings_count'].notnull()) &
(ratings['ratings_average'].notnull())][['ratings_count',
'ratings_average']]
```

```
https://www.kaggle.com/code/farizhaykal/recommendation-system-for-
amazon-products?scriptVersionId=81129622&cellId=39
```

```
  File "<ipython-input-28-151a85bb64aa>", line 1
    https://www.kaggle.com/code/farizhaykal/recommendation-system-for-
amazon-products?scriptVersionId=81129622&cellId=39
          ^
SyntaxError: invalid syntax
```

```
qualified['ratings_count'] = qualified['ratings_count'].astype('int')
qualified['ratings_average'] =
qualified['ratings_average'].astype('int')
qualified.head().sort_values(by='ratings_count', ascending=False)
```

```
{"summary":"{\n  \"name\": \"qualified\",\n  \"rows\": 15,\n
\"fields\": [\n    {\n      \"column\": \"productId\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15,\n        \"samples\": [\n
\"1400698987\",\n          \"7214047977\",\n          \"0972683275\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"ratings_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
255,\n        \"min\": 82,\n        \"max\": 1051,\n
\"num_unique_values\": 15,\n        \"samples\": [\n          138,\n
107,\n          1051\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ratings_average\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 3,\n
\"max\": 4,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          3,\n          4\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"qualified"}
```

```
qualified.shape
```

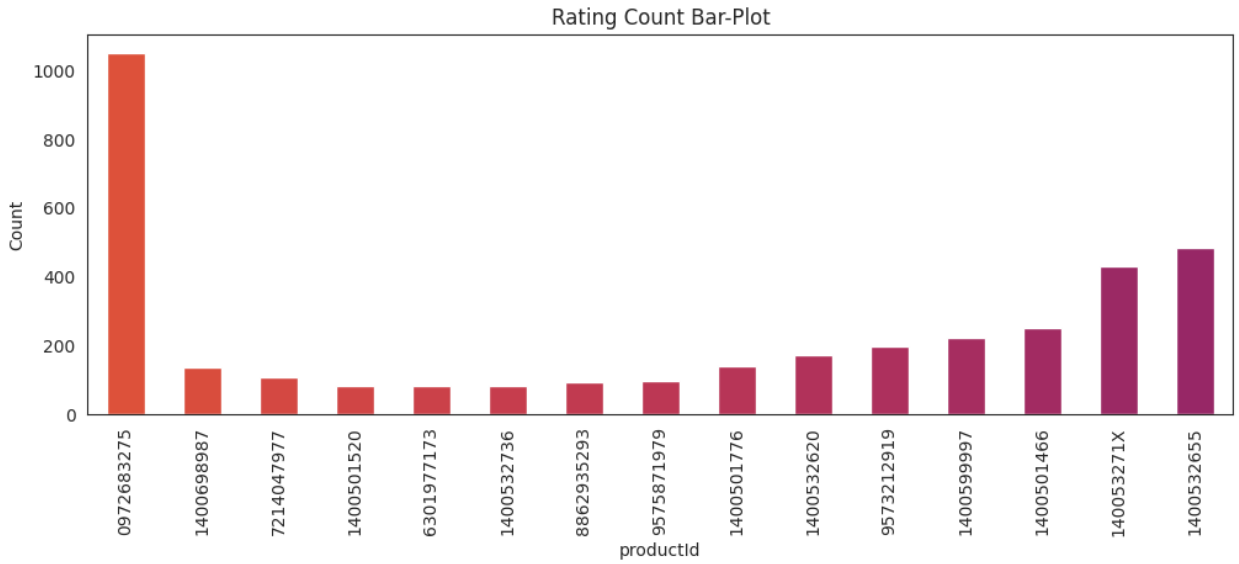
```
(15, 2)
```

```
def weighted_rating(x):
    v = x['ratings_count']
    R = x['ratings_average']
    return (v/(v+m) * R) + (m/(m+v) * C)

qualified['wr'] = qualified.apply(weighted_rating, axis=1)

qualified = qualified.sort_values('wr', ascending=False).head(20)

qualified.head(10)
```

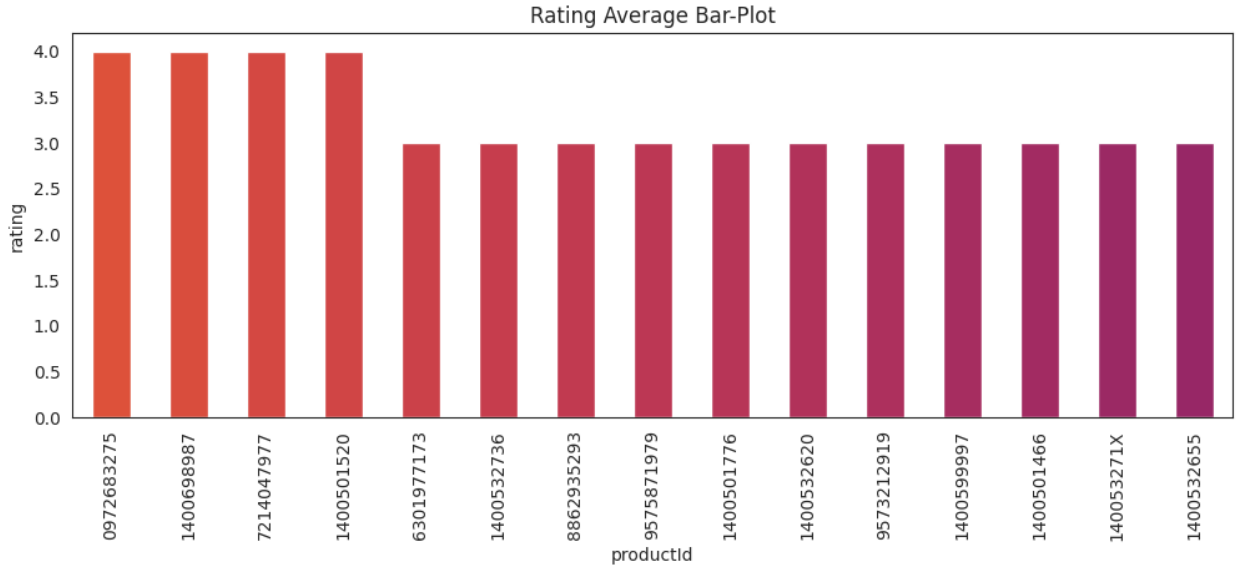

```
{"summary":"{\n  \"name\": \"qualified\",\n  \"rows\": 15,\n
\"fields\": [\n    {\n      \"column\": \"productId\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15,\n        \"samples\": [\n
\"1400532620\",\n          \"1400599997\",\n          \"0972683275\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"ratings_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
255,\n        \"min\": 82,\n        \"max\": 1051,\n
\"num_unique_values\": 15,\n        \"samples\": [\n          171,\n
222,\n          1051\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ratings_average\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 3,\n
\"max\": 4,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          3,\n          4\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"wr\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.33324625223611076,\n        \"min\":
3.0677934136613065,\n        \"max\": 3.9773883961029517,\n
\"num_unique_values\": 15,\n        \"samples\": [\n
3.158785431139819,\n          3.130291212321624\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"qualified"}
```


```
# Add color
from matplotlib import cm
color = cm.inferno_r(np.linspace(.4, .8, 30))

rating_plot_count = qualified['ratings_count'].plot.bar(figsize=(12,
4),color=color)
rating_plot_count.set_title("Rating Count Bar-Plot")
rating_plot_count.set_xlabel("productId")
rating_plot_count.set_ylabel("Count")
```

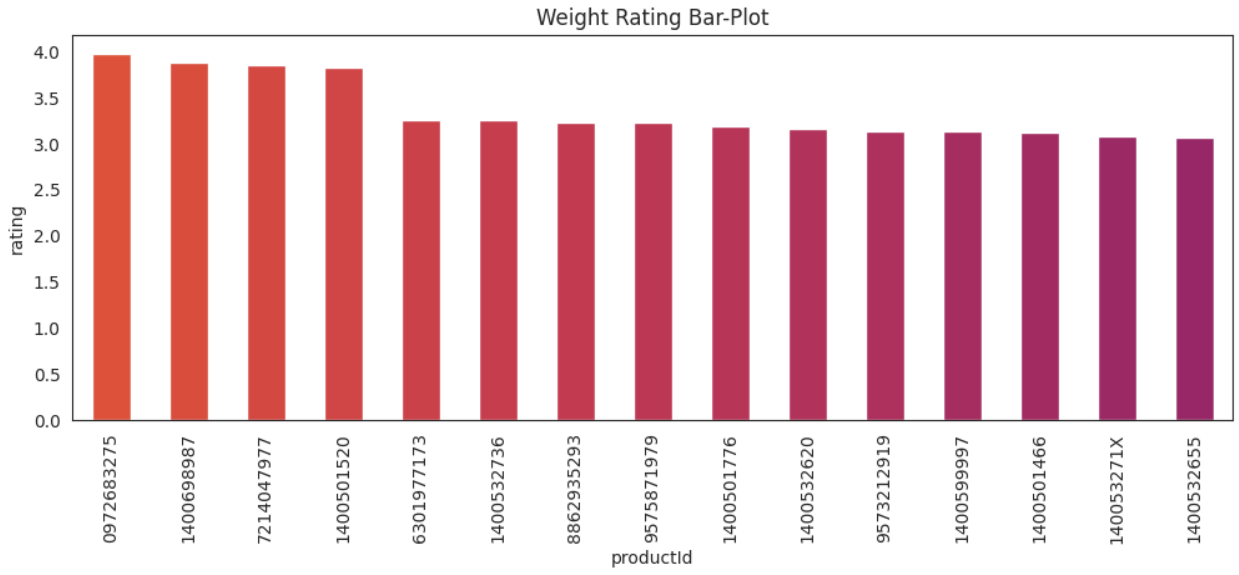
```
Text(0, 0.5, 'Count')
```




```
rating_plot_avg = qualified['ratings_average'].plot.bar(figsize=(12,
4),color=color)
rating_plot_avg.set_title("Rating Average Bar-Plot")
rating_plot_avg.set_xlabel("productId")
rating_plot_avg.set_ylabel("rating")

Text(0, 0.5, 'rating')
```




```
wr_plot = qualified['wr'].plot.bar(figsize=(12, 4),color=color)
wr_plot.set_title("Weight Rating Bar-Plot")
wr_plot.set_xlabel("productId")
wr_plot.set_ylabel("rating")

Text(0, 0.5, 'rating')
```




```
reader = Reader()

df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 5000,\n  \"fields\":
[\n    {\n      \"column\": \"userId\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 4929,\n
\"samples\": [\n          \"A29GBBAGFT2BM\",\n
\"A2WZKJ2TO6JP7M\",\n          \"A2HI35H2BOCV9R\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"productId\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 299,\n        \"samples\": [\n
\"9043413585\",\n          \"8862936826\",\n          \"6000012217\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"rating\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.411814110928476,\n        \"min\": 1.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1.0,\n
4.0,\n          3.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"timestamp\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 46498783.1041773,\n        \"min\":
940896000.0,\n        \"max\": 1405987200.0,\n
\"num_unique_values\": 1555,\n        \"samples\": [\n
1274745600.0,\n          1385337600.0,\n          1384214400.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
data = Dataset.load_from_df(df[['userId', 'productId', 'rating']],
reader)
```

```
# Use the famous SVD algorithm
svd = SVD()

# Run 5-fold cross-validation and then print results
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)    1.3416  1.3540  1.3711  1.3699  1.3875  1.3649
0.0157
MAE (testset)     1.0942  1.1060  1.1231  1.1006  1.1276  1.1103
0.0129
Fit time          0.15    0.09    0.07    0.07    0.07    0.09    0.03

Test time         0.01    0.00    0.00    0.00    0.00    0.01    0.00

{'test_rmse': array([1.34164426, 1.35404719, 1.3711482 , 1.36991713,
1.38753544]),
 'test_mae': array([1.09415458, 1.10595558, 1.12310671, 1.10064175,
1.12761081]),
 'fit_time': (0.14507818222045898,
  0.09437751770019531,
  0.07236981391906738,
  0.06802940368652344,
  0.07015824317932129),
 'test_time': (0.008614540100097656,
  0.004225730895996094,
  0.004988908767700195,
  0.004572629928588867,
  0.004108428955078125)}

trainset = data.build_full_trainset()
svd.fit(trainset)

<surprise.prediction_algorithms.matrix_factorization.SVD at
0x7967a8a243d0>

df.head()

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 5000,\n  \"fields\":
[\n    {\n      \"column\": \"userId\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 4929,\n
\"samples\": [\n          \"A29GBBAGFT2BM\",\n
\"A2WZKJ2TO6JP7M\",\n          \"A2HI35H2BOCV9R\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"productId\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 299,\n        \"samples\": [\n
```

\"9043413585\",\n   \"8862936826\",\n   \"6000012217\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"rating\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
1.411814110928476,\n   \"min\": 1.0,\n   \"max\": 5.0,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   1.0,\n
4.0,\n   3.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"timestamp\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 46498783.1041773,\n   \"min\":
940896000.0,\n   \"max\": 1405987200.0,\n
\"num_unique_values\": 1555,\n   \"samples\": [\n
1274745600.0,\n   1385337600.0,\n   1384214400.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe","variable_name":"df"}

```
df['userId'].value_counts()
```

```
A3LDPF5FMB782Z    5
A36V8NDDRZYRYO    3
A3E7PG9CHDBICA    3
AOY9SZTMNQWAW     3
A2FHM5FB0BXKGA    3
                 ..
A1W92X1R9QNM2C    1
A1KW4AGRCOIWI2    1
A1B6WHCBJSNO6J    1
A3BEVLI33Q0ZF4    1
A1TYKVIT4FTMRO    1
Name: userId, Length: 4929, dtype: int64
```

```
# Check specific userId review
df[df['userId'] == 'A3LDPF5FMB782Z']
```

{"summary":"{\n \"name\": \"df[df['userId'] == 'A3LDPF5FMB782Z']\",\n
\"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"userId\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n
\"A3LDPF5FMB782Z\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"productId\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 5,\n   \"samples\":
[\n   \"1400501520\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"rating\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.5477225575051662,\n   \"min\":
4.0,\n   \"max\": 5.0,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   4.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"timestamp\",\n   \"properties\": {\n

```
\"dtype\": \"number\",\n        \"std\": 31514487.347377237,\n
\"min\": 1310515200.0,\n        \"max\": 1392854400.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
1362873600.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

# predict based on this data
svd.predict('A3LDPF5FMB782Z', '140053271X', 5.0)

Prediction(uid='A3LDPF5FMB782Z', iid='140053271X', r_ui=5.0,
est=3.770977429976071, details={'was_impossible': False})
```