Product popularity based recommendation system targeted at new customers

Popularity based are a great strategy to target the new customers with the most popular products sold on a business website and is very useful to cold start a recommendation engine.

importing libraries

In [1]:

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

plt.style.use("ggplot")

import sklearn
from sklearn.decomposition import TruncatedSVD
```

Loading the dataset

In [5]:

```
amazon_ratings = pd.read_csv('ratings_Beauty.csv')
amazon_ratings = amazon_ratings.dropna()
amazon_ratings.head()
```

Out[5]:

	UserId	ProductId	Rating	Timestamp
0	A39HTATAQ9V7YF	0205616461	5.0	1369699200
1	A3JM6GV9MNOF9X	0558925278	3.0	1355443200
2	A1Z513UWSAAO0F	0558925278	5.0	1404691200
3	A1WMRR494NWEWV	0733001998	4.0	1382572800
4	A3IAAVS479H7M7	0737104473	1.0	1274227200

In [6]:

```
amazon_ratings.shape
```

Out[6]:

```
(2023070, 4)
```

In [7]:

```
popular_products = pd.DataFrame(amazon_ratings.groupby('ProductId')['Rating'].count())
most_popular = popular_products.sort_values('Rating', ascending=False)
most_popular.head(10)
```

Out[7]:

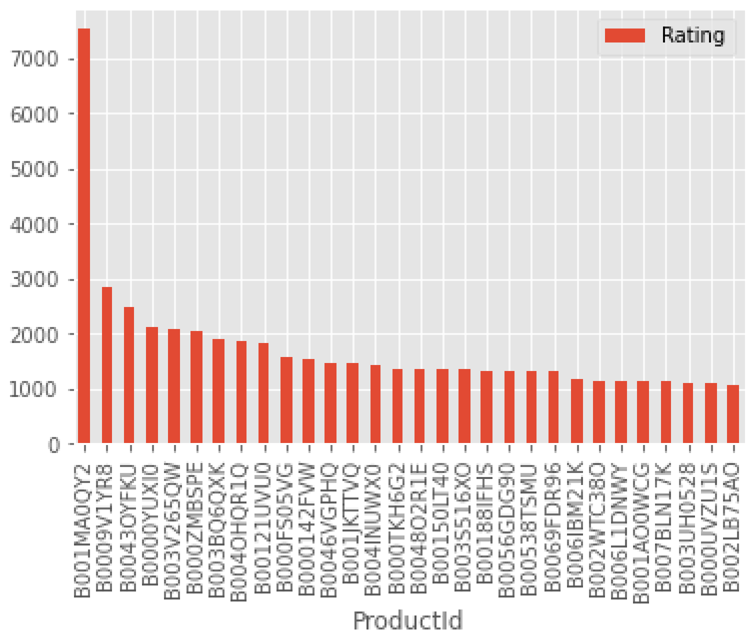
	Rating
ProductId	
B001MA0QY2	7533
B0009V1YR8	2869
B0043OYFKU	2477
B0000YUXI0	2143
B003V265QW	2088
B000ZMBSPE	2041
B003BQ6QXK	1918
B004OHQR1Q	1885
B00121UVU0	1838
B000FS05VG	1589

In [8]:

```
most_popular.head(30).plot(kind = "bar")
```

Out[8]:

<AxesSubplot:xlabel='ProductId'>



Analysis:

The above graph gives us the most popular products (arranged in descending order) sold by the business.

For eaxmple, product, ID # B001MA0QY2 has sales of over 7000, the next most popular product, ID # B0009V1YR8 has sales of 3000, etc.

Model-based collaborative filtering system

Recommend items to users based on purchase history and similarity of ratings provided by other users who bought items to that of a particular customer.

A model based collaborative filtering technique is closen here as it helps in making predictinfg products for a particular user by identifying patterns based on preferences from multiple user data.

In [9]:

```
# Subset of Amazon Ratings
amazon_ratings1 = amazon_ratings.head(10000)
```

Utility Matrix based on products sold and user reviews

Utility Matrix

An utlity matrix is consists of all possible user-item preferences (ratings) details represented as a matrix. The utility matrix is sparce as none of the users would buy all teh items in the list, hence, most of the values are unknown.

In [10]:

```
ratings_utility_matrix = amazon_ratings1.pivot_table(values='Rating', index='UserId', colum
ratings_utility_matrix.head()
```

Out[10]:

ProductId	0205616461	0558925278	0733001998	0737104473	0762451459	1304
UserId						
A00205921JHJK5X9LNP42	0	0	0	0	0	
A024581134CV80ZBLIZTZ	0	0	0	0	0	
A03056581JJIOL5FSKJY7	0	0	0	0	0	
A03099101ZRK4K607JVHH	0	0	0	0	0	
A0505229A7NSH3FRXRR4	0	0	0	0	0	

5 rows × 886 columns

As expected, the utility matrix obtaned above is sparce, I have filled up the unknown values wth 0.

In [11]:

```
ratings_utility_matrix.shape
```

Out[11]:

```
(9697, 886)
```

Transposing the matrix

In [12]:

```
X = ratings_utility_matrix.T
X.head()
```

Out[12]:

UserId ProductId	A00205921JHJK5X9LNP42	A024581134CV80ZBLIZTZ	A03056581JJIOL5FSKJY7	A030
0205616461	0	0	0	
0558925278	0	0	0	
0733001998	0	0	0	
0737104473	0	0	0	
0762451459	0	0	0	

5 rows × 9697 columns

In [13]:

```
X.shape
```

Out[13]:

```
(886, 9697)
```

Unique products in subset of data

In [14]:

```
X1 = X
```

Decomposing the Matrix

In [15]:

```
SVD = TruncatedSVD(n_components=10)
decomposed_matrix = SVD.fit_transform(X)
decomposed_matrix.shape
```

Out[15]:

```
(886, 10)
```

Correlation Matrix

In [18]:

```
correlation_matrix = np.corrcoef(decomposed_matrix)
correlation_matrix.shape
```

Out[18]:

```
(886, 886)
```

Isolating Product ID # 6117036094 from the Correlation Matrix Assuming the customer buys Product ID # 6117036094 (randomly chosen)

In [19]:

```
X.index[99]
```

Out[19]:

```
'6117036094'
```

In [20]:

```
i = "6117036094"

product_names = list(X.index)
product_ID = product_names.index(i)
product_ID
```

Out[20]:

```
99
```

Correlation for all items with the item purchased by this customer based on items rated by other customers people who bought the same product

In [21]:

```
correlation_product_ID = correlation_matrix[product_ID]
correlation_product_ID.shape
```

Out[21]:

```
(886,)
```

Recommending top 10 highly correlated products in sequence

In [22]:

```
Recommend = list(X.index[correlation_product_ID > 0.90])

# Removes the item already bought by the customer
Recommend.remove(i)

Recommend[0:9]
```

Out[22]:

```
['0558925278',
 '0762451459',
 '1304139220',
 '130414674X',
 '1304174778',
 '1304196046',
 '1304196062',
 '1304196135',
 '1304482685']
```

Product Id # Here are the top 10 products to be displayed by the recommendation system to the above customer based on the purchase history of other customers in the website.

For a business without any user-item purchase history, a search engine based recommendation system can be designed for users. The product recommendations can be based on textual clustering analysis given in product description.

In [23]:

```
# Importing libraries

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
```

Item to item based recommendation system based on product description

Applicable when business is setting up its E-commerce website for the first time

In [25]:

```
product_descriptions = pd.read_csv('product_descriptions.csv')
product_descriptions.shape
```

Out[25]:

```
(124428, 2)
```

Checking for missing values

In [26]:

```
# Missing values

product_descriptions = product_descriptions.dropna()
product_descriptions.shape
product_descriptions.head()
```

Out[26]:

	product_uid	product_description
0	100001	Not only do angles make joints stronger, they ...
1	100002	BEHR Premium Textured DECKOVER is an innovativ...
2	100003	Classic architecture meets contemporary design...
3	100004	The Grape Solar 265-Watt Polycrystalline PV So...
4	100005	Update your bathroom with the Delta Vero Singl...

In [27]:

```
product_descriptions1 = product_descriptions.head(500)
# product_descriptions1.iloc[:,1]

product_descriptions1["product_description"].head(10)
```

Out[27]:

```
0    Not only do angles make joints stronger, they ...
1    BEHR Premium Textured DECKOVER is an innovativ...
2    Classic architecture meets contemporary design...
3    The Grape Solar 265-Watt Polycrystalline PV So...
4    Update your bathroom with the Delta Vero Singl...
5    Achieving delicious results is almost effortle...
6    The Quantum Adjustable 2-Light LED Black Emerg...
7    The Teks #10 x 1-1/2 in. Zinc-Plated Steel Was...
8    Get the House of Fara 3/4 in. x 3 in. x 8 ft. ...
9    Valley View Industries Metal Stakes (4-Pack) a...
Name: product_description, dtype: object
```

Feature extraction from product descriptions Converting the text in product description into numerical data for analysis

In [28]:

```
vectorizer = TfidfVectorizer(stop_words='english')
X1 = vectorizer.fit_transform(product_descriptions1["product_description"])
X1
```

Out[28]:

```
<500x8932 sparse matrix of type '<class 'numpy.float64'>'
	with 34817 stored elements in Compressed Sparse Row format>
```

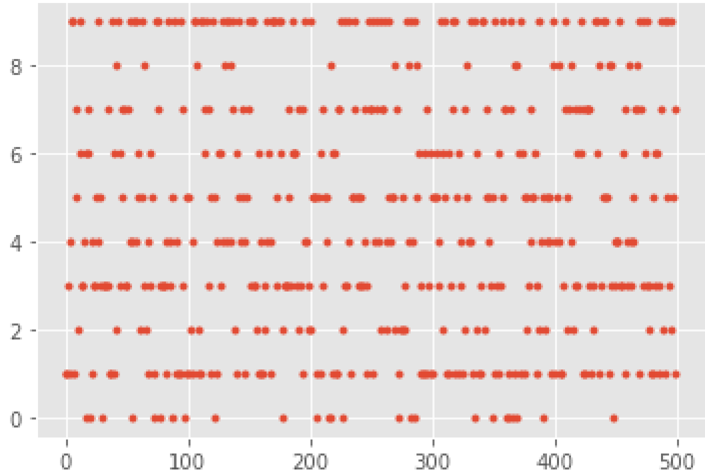
Visualizing product clusters in subset of data

In [29]:

```
# Fitting K-Means to the dataset

X=X1

kmeans = KMeans(n_clusters = 10, init = 'k-means++')
y_kmeans = kmeans.fit_predict(X)
plt.plot(y_kmeans, ".")
plt.show()
```



Top words in each cluster based on product description

In [30]:

```
# # Optimal clusters is

true_k = 10

model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)
model.fit(X1)

print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(true_k):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind]),
    print
```

```
 ft
 color
 natural
 resistant
 patio
Cluster 3:
 storage
 paint
 shelves
 unit
 bracket
 lbs
 wall
 shelving
 tools
 single
Cluster 4:
 insulation
 roof
 ice
```

Predicting clusters based on key search words cutting tool

In [32]:

```
print("Cluster ID:")
Y = vectorizer.transform(["cutting tool"])
prediction = model.predict(Y)
print(prediction)
```

```
Cluster ID:
[8]
```

spray paint

In [33]:

```
print("Cluster ID:")
Y = vectorizer.transform(["spray paint"])
prediction = model.predict(Y)
print(prediction)
```

```
Cluster ID:
[3]
```

steel drill

In [34]:

```
print("Cluster ID:")
Y = vectorizer.transform(["steel drill"])
prediction = model.predict(Y)
print(prediction)
```

```
Cluster ID:
[0]
```

water

In [35]:

```
print("Cluster ID:")
Y = vectorizer.transform(["water"])
prediction = model.predict(Y)
print(prediction)
```

```
Cluster ID:
[1]
```

Once a cluster is identified based on the user's search words, the recommendation system can display items from the corresponding product clusters based on the product descriptions.

Summary:

This works best if a business is setting up its e-commerce website for the first time and does not have user-item purchase/rating history to start with initally. This recommendation system will help the users get a good recommendation to start with and once the buyers have a purchased history, the recommendation engine can use the model based collaborative filtering technique.

In [ ]:

```
```