Recommendations with IBM

In this notebook, you will be putting your recommendation skills to use on real data from the IBM Watson Studio platform.

You may either submit your notebook through the workspace here, or you may work from your local machine and submit through the next page. Either way assure that your code passes the project RUBRIC. **Please save regularly.**

By following the table of contents, you will build out a number of different methods for making recommendations that can be used for different situations.

Table of Contents

At the end of the notebook, you will find directions for how to submit your work. Let's get started by importing the necessary libraries and reading in the data.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import project_tests as t
import pickle

%matplotlib inline

df = pd.read_csv('data/user-item-interactions.csv')
df_content = pd.read_csv('data/articles_community.csv')
del df['Unnamed: 0']
del df_content['Unnamed: 0']

# Show df to get an idea of the data
df.head()
```

Out[2]:

	article_id	title	email
0	1430.0	using pixiedust for fast, flexible, and easier...	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1	1314.0	healthcare python streaming application demo	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2	1429.0	use deep learning for image classification	b96a4f2e92d8572034b1e9b28f9ac673765cd074
3	1338.0	ml optimization using cognitive assistant	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4	1276.0	deploy your python model as a restful api	f01220c46fc92c6e6b161b1849de11faacd7ccb2

In [3]:
```
df.columns
```

Out[3]:
```
Index(['article_id', 'title', 'email'], dtype='object')
```

In [4]:
```
df_content.columns
```

Out[4]:
```
Index(['doc_body', 'doc_description', 'doc_full_name', 'doc_status',
       'article_id'],
      dtype='object')
```

In [5]:
```
# Show df_content to get an idea of the data
df_content.head()
```

Out[5]:

	doc_body	doc_description	doc_full_name	doc_status	article_id
0	Skip navigation Sign in SearchLoading...\r\n\r...	Detect bad readings in real time using Python ...	Detect Malfunctioning IoT Sensors with Streami...	Live	0
1	No Free Hunch Navigation * kaggle.com\r\n\r\n ...	See the forest, see the trees. Here lies the c...	Communicating data science: A guide to present...	Live	1
2	≡ * Login\r\n * Sign Up\r\n\r\n * Learning Pat...	Here's this week's news in Data Science and Bi...	This Week in Data Science (April 18, 2017)	Live	2
3	DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...	Learn how distributed DBs solve the problem of...	DataLayer Conference: Boost the performance of...	Live	3
4	Skip navigation Sign in SearchLoading...\r\n\r...	This video demonstrates the power of IBM DataS...	Analyze NY Restaurant data using Spark in DSX	Live	4

Part I : Exploratory Data Analysis

Use the dictionary and cells below to provide some insight into the descriptive statistics of the data.

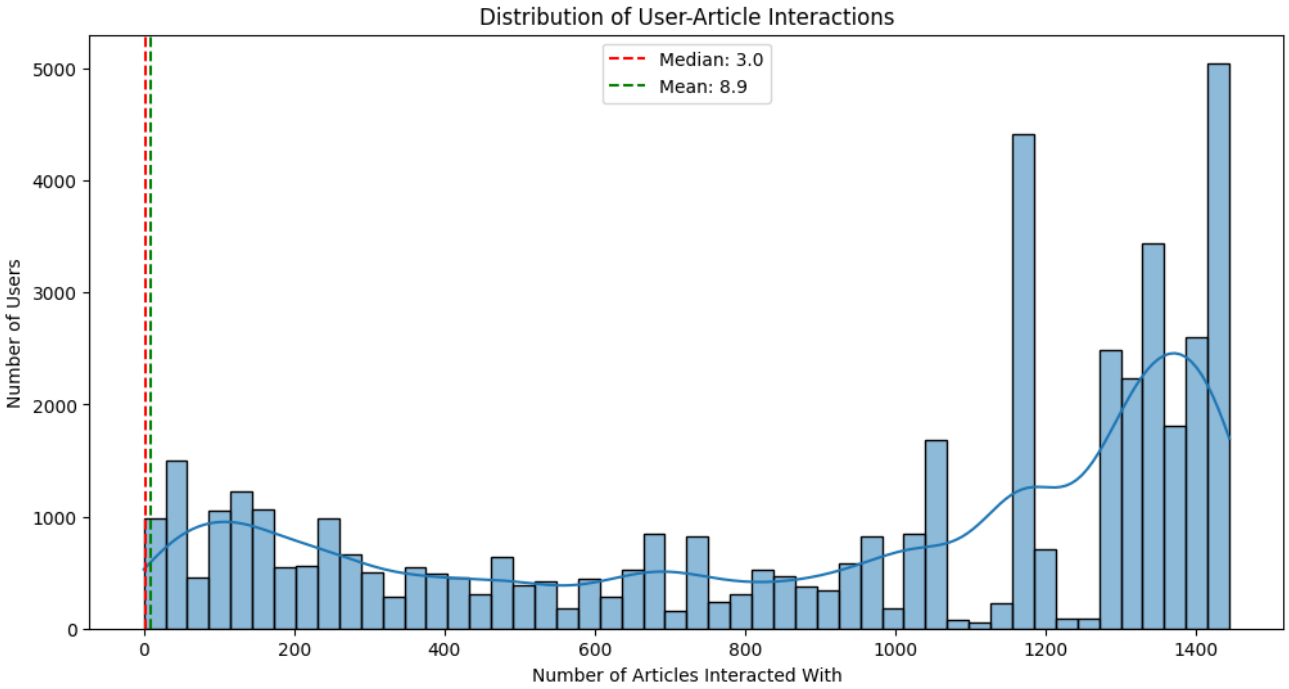
`1.` What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

In [6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def user_article_interactions(df):
    # Count interactions per user
    user_interactions = df.groupby('email')['article_id'].count()

    # Descriptive statistics
    desc_stats = user_interactions.describe()
    print("Descriptive Statistics of User-Article Interactions:")
    print(desc_stats)
```

```
    # Create visualization
    plt.figure(figsize=(12, 6))

    # Plot histogram with KDE
    sns.histplot(data=df,x='article_id',bins=50, kde=True)
    plt.title('Distribution of User-Article Interactions')
    plt.xlabel('Number of Articles Interacted With')
    plt.ylabel('Number of Users')

    # Add median and mean lines
    plt.axvline(user_interactions.median(), color='red', linestyle='--', lab
    plt.axvline(user_interactions.mean(), color='green', linestyle='--', lab

    plt.legend()
    plt.show()

    # Additional insights
    print("\nAdditional Insights:")
    print(f"Total number of unique users: {len(user_interactions)}")
    print(f"Number of users with single interaction: {len(user_interactions[
    print(f"Maximum interactions by a single user: {user_interactions.max()}

    # Calculate percentiles
    percentiles = [25, 50, 75, 90, 95, 99]
    print("\nPercentile Analysis:")
    for p in percentiles:
        print(f"{p}th percentile: {np.percentile(user_interactions, p):.1f}

    return user_interactions
```

In [7]:
```
interaction_distribution = user_article_interactions(df)
```

```
Descriptive Statistics of User-Article Interactions:
count    5148.000000
mean        8.930847
std        16.802267
min         1.000000
25%         1.000000
50%         3.000000
75%         9.000000
max       364.000000
Name: article_id, dtype: float64
```




```
Additional Insights:
Total number of unique users: 5148
Number of users with single interaction: 1416
Maximum interactions by a single user: 364

Percentile Analysis:
25th percentile: 1.0 interactions
50th percentile: 3.0 interactions
75th percentile: 9.0 interactions
90th percentile: 22.0 interactions
95th percentile: 35.0 interactions
99th percentile: 80.0 interactions
```

In [8]:
```
# Fill in the median and maximum number of user_article interactios below

median_val = interaction_distribution.median() # 50% of individuals interact
max_views_by_user = interaction_distribution.max() # The maximum number of u
print(median_val)
print(max_views_by_user)
```

```
3.0
364
```

2. Explore and remove duplicate articles from the **df_content** dataframe.

In [17]:
```
# Find and explore duplicate articles
df_content_duplicated = df_content.duplicated(subset=['article_id'], keep='f
print(df_content_duplicated.sum())
print(df_content[df_content_duplicated])
```

```
5
                                                 doc_body  \
365  Follow Sign in / Sign up Home About Insight Da...
692  Homepage Follow Sign in / Sign up Homepage * H...
761  Homepage Follow Sign in Get started Homepage *...
970  This video shows you how to construct queries ...
971  Homepage Follow Sign in Get started * Home\r\n...

                                          doc_description  \
365  During the seven-week Insight Data Engineering...
692  One of the earliest documented catalogs was co...
761  Today's world of data science leverages data f...
970  This video shows you how to construct queries ...
971  If you are like most data scientists, you are ...

                                       doc_full_name doc_status  article_i
d
365                        Graph-based machine learning       Live          5
0
692  How smart catalogs can turn the big data flood...       Live         22
1
761  Using Apache Spark as a parallel processing fr...       Live         39
8
970                               Use the Primary Index       Live         57
7
971  Self-service data preparation with IBM Data Re...       Live         23
2
```

In [23]:
```
# Remove any rows that have the same article_id - only keep the first
df_content_nodup = df_content.drop_duplicates(subset=['article_id'], keep='f
df_content_nodup.shape[0]
```

Out[23]: 1051

`3.` Use the cells below to find:

a. The number of unique articles that have an interaction with a user.

b. The number of unique articles in the dataset (whether they have any interactions or not).

c. The number of unique users in the dataset. (excluding null values)

d. The number of user-article interactions in the dataset.

In [31]:
```
### This gives the number of unique articles in the interactions dataframe
print('The number of unique articles that have an interaction with a user.')
print(df['article_id'].nunique())
print('The number of unique articles in the dataset (whether they have any i
print(df_content['article_id'].nunique())
print('The number of unique users in the dataset. (excluding null values)')
```

```
print(df[df['email'].notnull()]['email'].nunique())
print('The number of user-article interactions in the dataset.')
print(len(df))
```

```
The number of unique articles that have an interaction with a user.
714
The number of unique articles in the dataset (whether they have any interact
ions or not).
1051
The number of unique users in the dataset. (excluding null values)
5148
The number of user-article interactions in the dataset.
45993
```

In [32]:
```
unique_articles = df['article_id'].nunique() # The number of unique articles
total_articles = df_content['article_id'].nunique() # The number of unique a
unique_users = df[df['email'].notnull()]['email'].nunique() # The number of
user_article_interactions = len(df) # The number of user-article interactior
```

`4.` Use the cells below to find the most viewed **article_id**, as well as how often it was viewed. After talking to the company leaders, the `email_mapper` function was deemed a reasonable way to map users to ids. There were a small number of null values, and it was found that all of these null values likely belonged to a single user (which is how they are stored using the function below).

In [33]:
```
df[['article_id', 'email']].groupby(['article_id']).count().sort_values(['em
```

Out[33]:

article_id	email
1429.0	937
1330.0	927
1431.0	671
1427.0	643
1364.0	627
1314.0	614
1293.0	572
1170.0	565
1162.0	512
1304.0	483

In [38]:
```
most_viewed_article_id = '1429.0' # The most viewed article in the dataset a
max_views = 937 # The most viewed article in the dataset was viewed how many
```

In [34]:
```
## No need to change the code here - this will be helpful for later parts of
# Run this cell to map the user email to a user_id column and remove the ema

def email_mapper():
    coded_dict = dict()
    cter = 1
    email_encoded = []

    for val in df['email']:
        if val not in coded_dict:
            coded_dict[val] = cter
            cter+=1

        email_encoded.append(coded_dict[val])
    return email_encoded

email_encoded = email_mapper()
del df['email']
df['user_id'] = email_encoded

# show header
df.head()
```

Out[34]:

	article_id	title	user_id
0	1430.0	using pixiedust for fast, flexible, and easier...	1
1	1314.0	healthcare python streaming application demo	2
2	1429.0	use deep learning for image classification	3
3	1338.0	ml optimization using cognitive assistant	4
4	1276.0	deploy your python model as a restful api	5

In [39]:
```
## If you stored all your results in the variable names above,
## you shouldn't need to change anything in this cell

sol_1_dict = {
    '`50% of individuals have _____ or fewer interactions.`': median_val,
    '`The total number of user-article interactions in the dataset is _____
    '`The maximum number of user-article interactions by any 1 user is _____
    '`The most viewed article in the dataset was viewed _____ times.`': max_
    '`The article_id of the most viewed article is _____.`': most_viewed_ar
    '`The number of unique articles that have at least 1 rating _____.`': u
    '`The number of unique users in the dataset is _____`': unique_users,
    '`The number of unique articles on the IBM platform`': total_articles
```

```
}

# Test your dictionary against the solution
t.sol_1_test(sol_1_dict)
```

```
It looks like you have everything right here! Nice job!
```

Part II: Rank-Based Recommendations

Unlike in the earlier lessons, we don't actually have ratings for whether a user liked an article or not. We only know that a user has interacted with an article. In these cases, the popularity of an article can really only be based on how often an article was interacted with.

`1.` Fill in the function below to return the **n** top articles ordered with most interactions as the top. Test your function using the tests below.

In [53]:
```
def get_top_articles(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    '''
    # Your code here
    top_n_articles = df['article_id'].value_counts().head(n).index.to_list()
    top_articles = df[df['article_id'].isin(top_n_articles)]['title'].unique
    return top_articles # Return the top article titles from df (not df_cont

def get_top_article_ids(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    '''
    # Your code here
    top_articles = df['article_id'].value_counts().head(n).index.to_list()


    return top_articles # Return the top article ids
```

In [54]:
```
print(get_top_articles(10))
print(get_top_article_ids(10))
```

```
['healthcare python streaming application demo', 'use deep learning for imag
e classification', 'apache spark lab, part 1: basic concepts', 'predicting c
hurn with the spss random tree algorithm', 'analyze energy consumption in bu
ildings', 'visualize car data with brunel', 'use xgboost, scikit-learn & ibm
watson machine learning apis', 'gosales transactions for logistic regression
model', 'insights from new york car accident reports', 'finding optimal loca
tions of new store using decision optimization']
[1429.0, 1330.0, 1431.0, 1427.0, 1364.0, 1314.0, 1293.0, 1170.0, 1162.0, 130
4.0]
```

In [55]:
```
# Test your function by returning the top 5, 10, and 20 articles
top_5 = get_top_articles(5)
top_10 = get_top_articles(10)
top_20 = get_top_articles(20)

# Test each of your three lists from above
t.sol_2_test(get_top_articles)
```

```
Your top_5 looks like the solution list! Nice job.
Your top_10 looks like the solution list! Nice job.
Your top_20 looks like the solution list! Nice job.
```

Part III: User-User Based Collaborative Filtering

`1.` Use the function below to reformat the **df** dataframe to be shaped with users as the rows and articles as the columns.

- Each **user** should only appear in each **row** once.
- Each **article** should only show up in one **column**.
- **If a user has interacted with an article, then place a 1 where the user-row meets for that article-column**. It does not matter how many times a user has interacted with the article, all entries where a user has interacted with an article should be a 1.
- **If a user has not interacted with an item, then place a zero where the user-row meets for that article-column**.

Use the tests to make sure the basic structure of your matrix matches what is expected by the solution.

In [59]:
```
# create the user-article matrix with 1's and 0's

```

```
def create_user_item_matrix(df):
    '''
    INPUT:
    df - pandas dataframe with article_id, title, user_id columns

    OUTPUT:
    user_item - user item matrix

    Description:
    Return a matrix with user ids as rows and article ids on the columns wit
    an article and a 0 otherwise
    '''
    # Fill in the function here
    user_item = df.pivot_table(index='user_id',
                          columns='article_id',
                          values='article_id',
                          aggfunc='size',
                          fill_value=0)
    user_item = (user_item > 0).astype(int)

    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [60]:
```
## Tests: You should just need to run this cell.  Don't change the code.
assert user_item.shape[0] == 5149, "Oops!  The number of users in the user-a
assert user_item.shape[1] == 714, "Oops!  The number of articles in the user
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen b
print("You have passed our quick tests!  Please proceed!")
```

```
You have passed our quick tests!  Please proceed!
```

`2.` Complete the function below which should take a user_id and provide an ordered list of the most similar users to that user (from most similar to least similar). The returned result should not contain the provided user_id, as we know that each user is similar to him/herself. Because the results for each user here are binary, it (perhaps) makes sense to compute similarity as the dot product of two users.

Use the tests to test your function.

In [79]:
```
def find_similar_users(user_id, user_item=user_item):
    '''
    INPUT:
    user_id - (int) a user_id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

    OUTPUT:
```

```
    similar_users - (list) an ordered list where the closest users (largest
                    are listed first

    Description:
    Computes the similarity of every pair of users based on the dot product
    Returns an ordered

    '''
    # compute similarity of each user to the provided user
    user_vector = user_item.loc[user_id]
    similarities = user_item.dot(user_vector)
    #print(similarities)

    # sort by similarity
    # create list of just the ids
    similar_users = similarities.sort_values(ascending=False).index.tolist()
    #print(similar_users)

    # remove the own user's id
    #print(type(similar_users))
    similar_users.remove(user_id)
    most_similar_users = similar_users


    return most_similar_users # return a list of the users in order from mos
```

In [80]:
```
# Do a spot check of your function
print("The 10 most similar users to user 1 are: {}".format(find_similar_user
print("The 5 most similar users to user 3933 are: {}".format(find_similar_us
print("The 3 most similar users to user 46 are: {}".format(find_similar_user
```

```
The 10 most similar users to user 1 are: [3933, 23, 3782, 4459, 203, 131, 38
70, 4201, 46, 395]
The 5 most similar users to user 3933 are: [1, 3782, 23, 203, 4459]
The 3 most similar users to user 46 are: [4201, 3782, 23]
```

`3.` Now that you have a function that provides the most similar users to each user, you will want to use these users to find articles you can recommend. Complete the functions below to return the articles you would recommend to each user.

In [117…
```
def get_article_names(article_ids, df=df):
    '''
    INPUT:
    article_ids - (list) a list of article ids
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    article_names - (list) a list of article names associated with the list
```

```
                    (this is identified by the title column)
    '''
    # Your code here
    float_article_ids = [float(art_id) for art_id in article_ids]
    article_names = df[df['article_id'].isin(float_article_ids)]['title'].ur
    #article_names = []
    #for i in article_ids:
    #    name = df[df['article_id'] == i]['title'].iloc[0]
    #
    #    article_names.append(name)


    return article_names # Return the article names associated with list of


def get_user_articles(user_id, user_item=user_item):
    '''
    INPUT:
    user_id - (int) a user id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

    OUTPUT:
    article_ids - (list) a list of the article ids seen by the user
    article_names - (list) a list of article names associated with the list
                    (this is identified by the doc_full_name column in df_co

    Description:
    Provides a list of the article_ids and article titles that have been see
    '''
    # Your code here
    user_articles = user_item.loc[user_id]
    article_ids = user_articles[user_articles == 1].index.tolist()
    #print('user_artiles:')
    #print(article_ids.dtype())
    #print(article_ids)
    article_names = get_article_names(article_ids)
    article_ids = [str(id) for id in article_ids]


    return article_ids, article_names # return the ids and names


def user_user_recs(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user

    OUTPUT:
```

```
    recs - (list) a list of recommendations for the user

    Description:
    Loops through the users based on closeness to the input user_id
    For each user - finds articles the user hasn't seen before and provides
    Does this until m recommendations are found

    Notes:
    Users who are the same closeness are chosen arbitrarily as the 'next' us

    For the user where the number of recommended articles starts below m
    and ends exceeding m, the last items are chosen arbitrarily

    '''
    # Your code here
    user_articles, _ = get_user_articles(user_id)
    similar_users = find_similar_users(user_id)

    recs = []

    for similar_user in similar_users:
        similar_user_articles, _ = get_user_articles(similar_user)
        new_recs = [article for article in similar_user_articles if article
        recs.extend(new_recs)

        if len(recs) >= m:
            break
    recs = recs[:m]

    return recs # return your recommendations for this user_id
```

In [118…
```
get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.0
get_user_articles(20)[0]
```

Out[118… ['232.0', '844.0', '1320.0']

In [119…
```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for use
```

Out[119…
```
['got zip code data? prep it for analytics. – ibm watson data lab – mediu
m',
 'timeseries data analysis of iot events by using jupyter notebook',
 'graph-based machine learning',
 'using brunel in ipython/jupyter notebooks',
 'experience iot with coursera',
 'the 3 kinds of context: machine learning and the art of the frame',
 'deep forest: towards an alternative to deep neural networks',
 'this week in data science (april 18, 2017)',
 'higher-order logistic regression for large datasets',
 'using machine learning to predict parking difficulty']
```

In [120…
```
# Test your functions here - No need to change this code - just run this cel
assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing
assert set(get_user_articles(20)[0]) == set(['1320.0', '232.0', '844.0'])
assert set(get_user_articles(20)[1]) == set(['housing (2015): united states
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstr
print("If this is all you see, you passed all of our tests!  Nice job!")
```

```
If this is all you see, you passed all of our tests!  Nice job!
```

`4.` Now we are going to improve the consistency of the **user_user_recs** function from above.

- Instead of arbitrarily choosing when we obtain users who are all the same closeness to a given user - choose the users that have the most total article interactions before choosing those with fewer article interactions.
- Instead of arbitrarily choosing articles from the user where the number of recommended articles starts below m and ends exceeding m, choose articles with the articles with the most total interactions before choosing those with fewer total interactions. This ranking should be what would be obtained from the **top_articles** function you wrote earlier.

In [123…
```
def get_top_sorted_users(user_id, df=df, user_item=user_item):
    '''
    INPUT:
    user_id - (int)
    df - (pandas dataframe) df as defined at the top of the notebook
    user_item - (pandas dataframe) matrix of users by articles:
            1's when a user has interacted with an article, 0 otherwise


    OUTPUT:
    neighbors_df - (pandas dataframe) a dataframe with:
```

```
                    neighbor_id - is a neighbor user_id
                    similarity - measure of the similarity of each user to t
                    num_interactions - the number of articles viewed by the

    Other Details - sort the neighbors_df by the similarity and then by numb
                    highest of each is higher in the dataframe

    '''
    # Calculate similarity
    target_user = user_item.loc[user_id]
    similarities = user_item.dot(target_user)


    # Create dataframe for neighbors
    neighbors_df = pd.DataFrame({
        'neighbor_id': similarities.index,
        'similarity': similarities.values,
        'num_interactions': user_item.sum(axis=1).values
    })

    # Exclude the target user from neighbors
    neighbors_df = neighbors_df[neighbors_df['neighbor_id'] != user_id]

    # Sort by similarity and then by number of interactions
    neighbors_df = neighbors_df.sort_values(
        by=['similarity', 'num_interactions'],
        ascending=[False, False]
    ).reset_index(drop=True)


    return neighbors_df # Return the dataframe specified in the doc_string


def user_user_recs_part2(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user

    OUTPUT:
    recs - (list) a list of recommendations for the user by article id
    rec_names - (list) a list of recommendations for the user by article tit

    Description:
    Loops through the users based on closeness to the input user_id
    For each user - finds articles the user hasn't seen before and provides
    Does this until m recommendations are found

    Notes:
    * Choose the users that have the most total article interactions
```

```
    before choosing those with fewer article interactions.

    * Choose articles with the articles with the most total interactions
    before choosing those with fewer total interactions.

    '''
# Get the sorted neighbors
    neighbors_df = get_top_sorted_users(user_id)

    # Articles seen by the target user
    user_articles, _ = get_user_articles(user_id)

    # Initialize recommendations
    recs = []
    for neighbor_id in neighbors_df['neighbor_id']:
        neighbor_articles, _ = get_user_articles(neighbor_id)
        # Get new articles not seen by the user
        new_recs = [article for article in neighbor_articles if article not

        # Rank new recommendations by total interactions
        ranked_recs = sorted(
            new_recs,
            key=lambda art: df[df['article_id'] == float(art)].shape[0],
            reverse=True
        )
        recs.extend(ranked_recs)

        if len(recs) >= m:
            break

    # Trim to top m recommendations
    recs = recs[:m]

    # Get article names
    rec_names = get_article_names(recs, df)
    return recs, rec_names
```

In [124…

```
# Quick spot check - don't change this code - just use it to test your funct
rec_ids, rec_names = user_user_recs_part2(20, 10)
print("The top 10 recommendations for user 20 are the following article ids:
print(rec_ids)
print()
print("The top 10 recommendations for user 20 are the following article name
print(rec_names)
```

```
The top 10 recommendations for user 20 are the following article ids:
['1429.0', '1330.0', '1314.0', '1293.0', '1162.0', '1271.0', '43.0', '1351.
0', '1368.0', '1305.0']

The top 10 recommendations for user 20 are the following article names:
['healthcare python streaming application demo', 'use deep learning for imag
e classification', 'analyze energy consumption in buildings', 'putting a hum
an face on machine learning', 'gosales transactions for naive bayes model',
'insights from new york car accident reports', 'model bike sharing data with
spss', 'finding optimal locations of new store using decision optimization',
'deep learning with tensorflow course by big data university', 'customer dem
ographics and sales']
```

`5.` Use your functions from above to correctly fill in the solutions to the dictionary below. Then test your dictionary against the solution. Provide the code you need to answer each following the comments below.

In [164…
```
### Tests with a dictionary of results
user1_most_sim = get_top_sorted_users(1).iloc[0]['neighbor_id'] # Find the u
user131_10th_sim = get_top_sorted_users(131)['neighbor_id'].iloc[10] # Find
```

In [165…
```
## Dictionary Test Here
sol_5_dict = {
    'The user that is most similar to user 1.': user1_most_sim,
    'The user that is the 10th most similar to user 131': user131_10th_sim,
}

t.sol_5_test(sol_5_dict)
```

```
This all looks good!  Nice job!
```

`6.` If we were given a new user, which of the above functions would you be able to use to make recommendations? Explain. Can you think of a better way we might make recommendations? Use the cell below to explain a better method for new users.

With new user, we don't know anything about their preference and it won't make sense to find any similar user as we don't have any data about them, rather it would make sense to recommend the top 10 popular articles from the inventory.

`7.` Using your existing functions, provide the top 10 recommended articles you would provide for the a new user below. You can test your function against our thoughts to make sure we are all on the same page with how we might make a recommendation.

In [186…
```
new_user = '0.0'

# What would your recommendations be for this new user '0.0'?  As a new user
# Provide a list of the top 10 article ids you would give to
```

```
new_user_recs = df[['article_id','user_id']].groupby(['article_id']).count()
```

In [187…
```
assert set(new_user_recs) == set(['1314.0','1429.0','1293.0','1427.0','1162.

print("That's right!  Nice job!")
```

```
That's right!  Nice job!
```

Part IV: Content Based Recommendations (EXTRA - NOT REQUIRED)

Another method we might use to make recommendations is to perform a ranking of the highest ranked articles associated with some term. You might consider content to be the **doc_body**, **doc_description**, or **doc_full_name**. There isn't one way to create a content based recommendation, especially considering that each of these columns hold content related information.

`1.` Use the function body below to create a content based recommender. Since there isn't one right answer for this recommendation tactic, no test functions are provided. Feel free to change the function inputs if you decide you want to try a method that requires more input values. The input values are currently set with one idea in mind that you may use to make content based recommendations. One additional idea is that you might want to choose the most popular recommendations that meet your 'content criteria', but again, there is a lot of flexibility in how you might make these recommendations.

This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

In [ ]:
```
def make_content_recs():
    '''
    INPUT:

    OUTPUT:

    '''
```

`2.` Now that you have put together your content-based recommendation system, use the cell below to write a summary explaining how your content based recommender works. Do you see any possible improvements that could be made to your function? Is there anything novel about your content based recommender?

This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

Write an explanation of your content based recommendation system here.

`3.` Use your content-recommendation system to make recommendations for the below scenarios based on the comments. Again no tests are provided here, because there isn't one right answer that could be used to find these content based recommendations.

This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

In [ ]:
```
# make recommendations for a brand new user


# make a recommendations for a user who only has interacted with article id
```

Part V: Matrix Factorization

In this part of the notebook, you will build use matrix factorization to make article recommendations to the users on the IBM Watson Studio platform.

`1.` You should have already created a **user_item** matrix above in **question 1** of **Part III** above. This first question here will just require that you run the cells to get things set up for the rest of **Part V** of the notebook.

In [188…
```
# Load the matrix here
user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

In [189…
```
# quick look at the matrix
user_item_matrix.head()
```

Out[189…

article_id user_id	0.0	100.0	1000.0	1004.0	1006.0	1008.0	101.0	1014.0	1015.0	1016.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 714 columns

`2.` In this situation, you can use Singular Value Decomposition from numpy on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

In [190…
```
# Perform SVD on the User-Item Matrix Here
import numpy as np
from scipy.linalg import svd

u, s, vt = svd(user_item_matrix) # use the built in to get the three matrice
```

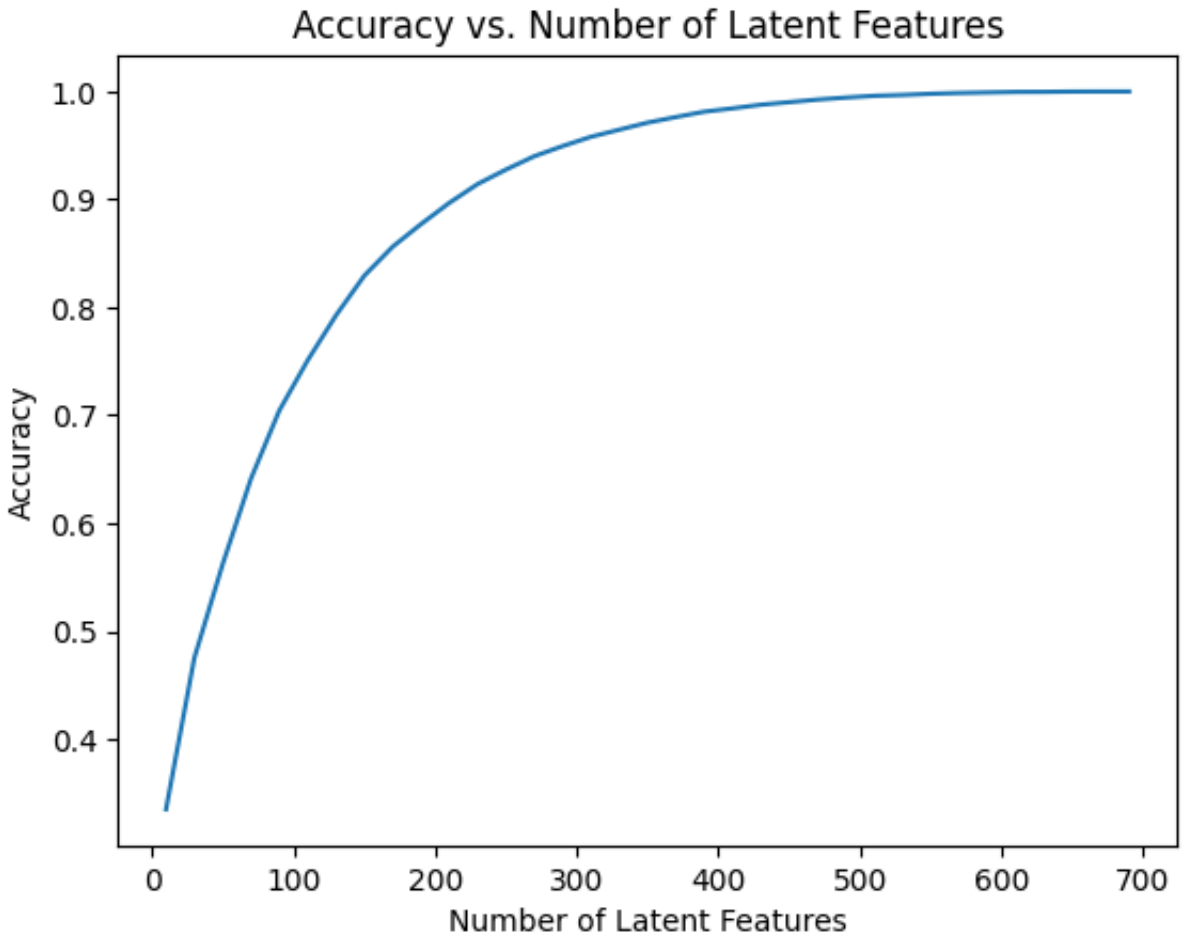
Provide your response here.

`3.` Now for the tricky part, how do we choose the number of latent features to use? Running the below cell, you can see that as the number of latent features increases, we obtain a lower error rate on making predictions for the 1 and 0 values in the user-item matrix. Run the cell below to get an idea of how the accuracy improves as we increase the number of latent features.

In [191…
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs = []

for k in num_latent_feats:
    # restructure with k latent features
    s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]

    # take dot product
    user_item_est = np.around(np.dot(np.dot(u_new, s_new), vt_new))

    # compute error for each prediction to actual value
    diffs = np.subtract(user_item_matrix, user_item_est)
```

```
    # total errors and keep track of them
    err = np.sum(np.sum(np.abs(diffs)))
    sum_errs.append(err)


plt.plot(num_latent_feats, 1 - np.array(sum_errs)/df.shape[0]);
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/numpy/_core/fromnumeric.py:84: FutureWarning: The behavior of DataFrame.sum with axis=None is deprecated, in a future version this will reduce over both axes and return a scalar. To retain the old behavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```



4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are able to make good recommendations. Instead, we might split our dataset into a training and test set of data, as shown in the cell below.

Use the code from question 3 to understand the impact on accuracy of the training and test sets of data with different numbers of latent features. Using the split below:

- How many users can we make predictions for in the test set?
- How many users are we not able to make predictions for because of the cold start problem?
- How many articles can we make predictions for in the test set?
- How many articles are we not able to make predictions for because of the cold start problem?

In [193…
```
df_train = df.head(40000)
df_test = df.tail(5993)

def create_test_and_train_user_item(df_train, df_test):
    '''
    INPUT:
    df_train - training dataframe
    df_test - test dataframe

    OUTPUT:
    user_item_train - a user-item matrix of the training dataframe
                      (unique users for each row and unique articles for eac
    user_item_test - a user-item matrix of the testing dataframe
                    (unique users for each row and unique articles for each
    test_idx - all of the test user ids
    test_arts - all of the test article ids

    '''
    user_item_train = create_user_item_matrix(df_train)
    user_item_test = create_user_item_matrix(df_test)

    test_idx = user_item_test.index.values
    test_arts = user_item_test.columns.values
    return user_item_train, user_item_test, test_idx, test_arts

user_item_train, user_item_test, test_idx, test_arts = create_test_and_trair
```

In [200…
```
#How many users can we make predictions for in the test set?
len(np.intersect1d(df_train['user_id'].unique(),df_test['user_id'].unique())
```

Out[200… 20

In [201…
```
# How many users in the test set are we not able to make predictions for bec
len(df_test['user_id'].unique()) - len(np.intersect1d(df_train['user_id'].ur
```

Out[201… 662

In [206…
```
#How many articles can we make predictions for in the test set?
len(np.intersect1d(df_train['article_id'].unique(),df_test['article_id'].uni
```

Out[206… 574

In [202…
```
# How many articles in the test set are we not able to make predictions for
len(df_test['article_id'].unique()) - len(np.intersect1d(df_train['article_i
```

Out[202… 0

In [208…
```
# Replace the values in the dictionary below
a = 662
b = 574
c = 20
d = 0


sol_4_dict = {
    'How many users can we make predictions for in the test set?': c, # lett
    'How many users in the test set are we not able to make predictions for
    'How many articles can we make predictions for in the test set?': b, # l
    'How many articles in the test set are we not able to make predictions f
}

t.sol_4_test(sol_4_dict)
```

```
Awesome job!  That's right!  All of the test articles are in the training da
ta, but there are only 20 test users that were also in the training set.  Al
l of the other users that are in the test set we have no data on.  Therefor
e, we cannot make predictions for these users using SVD.
```

`5.` Now use the **user_item_train** dataset from above to find U, S, and V transpose using SVD. Then find the subset of rows in the **user_item_test** dataset that you can predict using this matrix decomposition with different numbers of latent features to see how many features makes sense to keep based on the accuracy on the test data. This will require combining what was done in questions `2` - `4`.

Use the cells below to explore how well SVD works towards making predictions for recommendations on the test data.

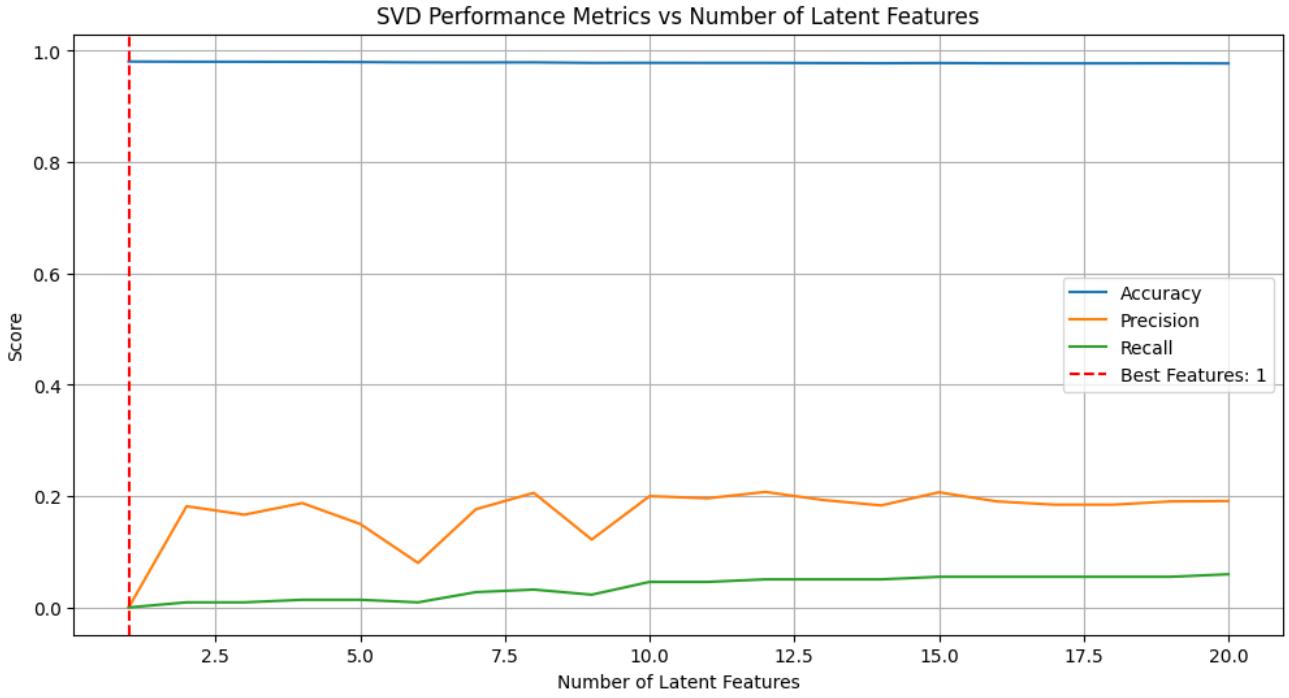
In [211…
```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = svd(user_item_train.values) # fit svd similar t
```

In [213…
```
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score
import pandas as pd
from scipy.linalg import svd
```

```
def perform_svd_analysis(user_item_train, user_item_test, max_features=20):
    """
    Perform SVD analysis and evaluate performance on test data with differer

    Args:
        user_item_train: Training dataset
        user_item_test: Test dataset
        max_features: Maximum number of latent features to test

    Returns:
        tuple: (metrics_df, best_n_features)
    """
    # Convert to numpy arrays for SVD
    train_matrix = user_item_train.values

    # Perform SVD
    U, S, Vt = svd(train_matrix)

    # Store metrics for different numbers of features
    metrics = []

    # Get common users and items between train and test
    common_users = set(user_item_train.index) & set(user_item_test.index)
    common_items = set(user_item_train.columns) & set(user_item_test.columns

    user_positions = [user_item_train.index.get_loc(user) for user in commor
    item_positions = [user_item_train.columns.get_loc(item) for item in comm

    # Filter test data to only include common users and items
    test_filtered = user_item_test.loc[list(common_users), list(common_items

    # Try different numbers of features
    for n_features in range(1, min(max_features + 1, len(S))):
        # Reconstruct matrix with n features
        U_reduced = U[:, :n_features]
        S_reduced = np.diag(S[:n_features])
        Vt_reduced = Vt[:n_features, :]

        predicted_matrix = U_reduced @ S_reduced @ Vt_reduced

        # Convert predictions to binary (0 or 1)
        predicted_binary = (predicted_matrix > 0.5).astype(int)

        # Calculate metrics only for common users/items
        y_true = test_filtered.values.flatten()
        y_pred = predicted_binary[np.ix_(user_positions, item_positions)].fl
        #y_pred = predicted_binary[np.ix_(list(common_users), list(common_it

        # Calculate metrics
        accuracy = accuracy_score(y_true, y_pred)
```

```
        precision = precision_score(y_true, y_pred)
        recall = recall_score(y_true, y_pred)

        metrics.append({
            'n_features': n_features,
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall
        })

    # Convert results to DataFrame
    metrics_df = pd.DataFrame(metrics)

    # Find optimal number of features based on accuracy
    best_n_features = metrics_df.loc[metrics_df['accuracy'].idxmax(), 'n_fea

    return metrics_df, best_n_features

# Run the analysis
metrics_df, best_n_features = perform_svd_analysis(user_item_train, user_ite

print("Metrics for different numbers of latent features:")
print(metrics_df)
print(f"\nBest number of features based on accuracy: {best_n_features}")

# Plot the results
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(metrics_df['n_features'], metrics_df['accuracy'], label='Accuracy')
plt.plot(metrics_df['n_features'], metrics_df['precision'], label='Precisior
plt.plot(metrics_df['n_features'], metrics_df['recall'], label='Recall')
plt.axvline(x=best_n_features, color='r', linestyle='--', label=f'Best Featu
plt.xlabel('Number of Latent Features')
plt.ylabel('Score')
plt.title('SVD Performance Metrics vs Number of Latent Features')
plt.legend()
plt.grid(True)
plt.show()
```

```
Metrics for different numbers of latent features:
    n_features  accuracy  precision    recall
0            1  0.980749   0.000000  0.000000
1            2  0.980401   0.181818  0.009174
2            3  0.980314   0.166667  0.009174
3            4  0.980139   0.187500  0.013761
4            5  0.979791   0.150000  0.013761
5            6  0.979181   0.080000  0.009174
6            7  0.979094   0.176471  0.027523
7            8  0.979268   0.205882  0.032110
8            9  0.978310   0.121951  0.022936
9           10  0.978397   0.200000  0.045872
10          11  0.978310   0.196078  0.045872
11          12  0.978310   0.207547  0.050459
12          13  0.977962   0.192982  0.050459
13          14  0.977700   0.183333  0.050459
14          15  0.978049   0.206897  0.055046
15          16  0.977613   0.190476  0.055046
16          17  0.977439   0.184615  0.055046
17          18  0.977439   0.184615  0.055046
18          19  0.977613   0.190476  0.055046
19          20  0.977352   0.191176  0.059633

Best number of features based on accuracy: 1
```



`6.` Use the cell below to comment on the results you found in the previous question. Given the circumstances of your results, discuss what you might do to determine if the recommendations you make with any of the above recommendation systems are an improvement to how users currently find articles?

Your response here.

We can see that the precision tops at 20% for 8-12 features and recall stays flat when number of features are over 12. This proves that when the number of features go over 12 there isn't any imporvement in recall (users interest) and precision (% of articles users are reading from our recommendations).

Extras

Using your workbook, you could now save your recommendations for each user, develop a class to make new predictions and update your results, and make a flask app to deploy your results. These tasks are beyond what is required for this project. However, from what you learned in the lessons, you certainly capable of taking these tasks on to improve upon your work here!

Conclusion

> Congratulations! You have reached the end of the Recommendations with IBM project!

> **Tip**: Once you are satisfied with your work here, check over your report to make sure that it is satisfies all the areas of the rubric. You should also probably remove all of the "Tips" like this one so that the presentation is as polished as possible.

Directions to Submit

> Before you submit your project, you need to create a .html or .pdf version of this notebook in the workspace here. To do that, run the code cell below. If it worked correctly, you should get a return code of 0, and you should see the generated .html file in the workspace directory (click on the orange Jupyter icon in the upper left).

> Alternatively, you can download this report as .html via the **File** > **Download as** submenu, and then manually upload it into the workspace directory by clicking on the orange Jupyter icon in the upper left, then using the Upload button.

> Once you've done this, you can submit your project by clicking on the

> "Submit Project" button in the lower right here. This will create and submit a zip file with this .ipynb doc and the .html or .pdf version you created. Congratulations!

In [ ]:
```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```