Recommendations_with_IBM

January 19, 2022

1 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.

1.1 Table of Contents

I. Section **??** II. Section **??** III. Section **??** IV. Section **??** V. Section **??** VI. Section **??**

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 [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import project_tests as t
        import pickle
        from scipy.stats.stats import pearsonr

        %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[1]:    article_id                                              title  \
        0      1430.0  using pixiedust for fast, flexible, and easier...
        1      1314.0       healthcare python streaming application demo
        2      1429.0         use deep learning for image classification
        3      1338.0          ml optimization using cognitive assistant
```

```
4      1276.0          deploy your python model as a restful api

                                          email
0  ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1  083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2  b96a4f2e92d8572034b1e9b28f9ac673765cd074
3  06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4  f01220c46fc92c6e6b161b1849de11faacd7ccb2
```

```
In [2]: # Show df_content to get an idea of the data
        df_content.head()
```

```
Out[2]:                                              doc_body  \
0  Skip navigation Sign in SearchLoading...\r\n\r...
1  No Free Hunch Navigation * kaggle.com\r\n\r\n ...
2   * Login\r\n * Sign Up\r\n\r\n * Learning Pat...
3  DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...
4  Skip navigation Sign in SearchLoading...\r\n\r...

                                       doc_description  \
0  Detect bad readings in real time using Python ...
1  See the forest, see the trees. Here lies the c...
2  Heres this weeks news in Data Science and Bi...
3  Learn how distributed DBs solve the problem of...
4  This video demonstrates the power of IBM DataS...

                                        doc_full_name doc_status  article_id
0  Detect Malfunctioning IoT Sensors with Streami...       Live           0
1  Communicating data science: A guide to present...       Live           1
2         This Week in Data Science (April 18, 2017)       Live           2
3  DataLayer Conference: Boost the performance of...       Live           3
4      Analyze NY Restaurant data using Spark in DSX       Live           4
```

1.1.1 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 [3]: df.shape,  df_content.shape
```

```
Out[3]: ((45993, 3), (1056, 5))
```

```
In [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45993 entries, 0 to 45992
```

```
Data columns (total 3 columns):
article_id    45993 non-null float64
title         45993 non-null object
email         45976 non-null object
dtypes: float64(1), object(2)
memory usage: 1.1+ MB
```

```
In [5]: df_content.info()
```

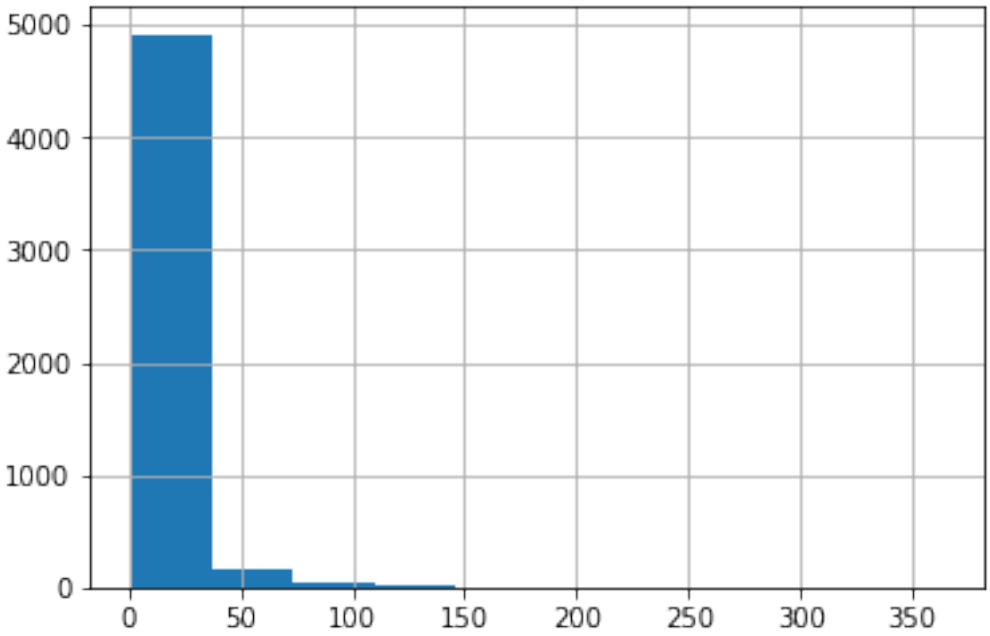
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1056 entries, 0 to 1055
Data columns (total 5 columns):
doc_body           1042 non-null object
doc_description    1053 non-null object
doc_full_name      1056 non-null object
doc_status         1056 non-null object
article_id         1056 non-null int64
dtypes: int64(1), object(4)
memory usage: 41.3+ KB
```

```
In [6]: print(df.isnull().sum())
        print('-')
        print(df_content.isnull().sum())
```

```
article_id     0
title          0
email         17
dtype: int64
-
doc_body           14
doc_description     3
doc_full_name       0
doc_status          0
article_id          0
dtype: int64
```

```
In [7]: user_by_article = df.groupby('email').article_id.count()
        user_by_article.hist()
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7faac8ec7e10>
```




```
In [8]: np.sort(user_by_article)[-1]
```

```
Out[8]: 364
```

```
In [9]: np.median(user_by_article)
```

```
Out[9]: 3.0
```

```
In [10]: # Fill in the median and maximum number of user_article interactios below

         median_val = 3.0 # 50% of individuals interact with _3___ number of articles or fewer.
         max_views_by_user = 364 # The maximum number of user-article interactions by any 1 user
```

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

```
In [11]: # Find and explore duplicate articles
         #df_content.duplicated(subset=['doc_body', 'doc_description', 'doc_full_name']).sum()
         print(df_content[df_content.duplicated(subset=[  'article_id'], keep=False)].count())
         df_content[df_content.duplicated(subset=[  'article_id'], keep=False)]
```

```
doc_body           10
doc_description    10
doc_full_name      10
doc_status         10
article_id         10
dtype: int64
```

Out[11]:

	doc_body	doc_description	doc_full_name	doc_status	article_id
50	Follow Sign in / Sign up Home About Insight Da...	Community Detection at Scale	Graph-based machine learning	Live	50
221	* United States\r\n\r\nIBMő * Site map\r\n\r\n...	When used to make sense of huge amounts of con...	How smart catalogs can turn the big data flood...	Live	221
232	Homepage Follow Sign in Get started Homepage *...	If you are like most data scientists, you are ...	Self-service data preparation with IBM Data Re...	Live	232
365	Follow Sign in / Sign up Home About Insight Da...	During the seven-week Insight Data Engineering...	Graph-based machine learning	Live	50
399	Homepage Follow Sign in Get started * Home\r\n...	Todays world of data science leverages data f...	Using Apache Spark as a parallel processing fr...	Live	398
578	This video shows you how to construct queries ...	This video shows you how to construct queries ...	Use the Primary Index	Live	577
692	Homepage Follow Sign in / Sign up Homepage * H...	One of the earliest documented catalogs was co...	How smart catalogs can turn the big data flood...	Live	221
761	Homepage Follow Sign in Get started Homepage *...	Todays world of data science leverages data f...	Using Apache Spark as a parallel processing fr...	Live	398
970	This video shows you how to construct queries ...	This video shows you how to construct queries ...	Use the Primary Index	Live	577
971	Homepage Follow Sign in Get started * Home\r\n...	If you are like most data scientists, you are ...	Self-service data preparation with IBM Data Re...	Live	232

In [12]:
```
# Remove any rows that have the same article_id - only keep the first
df_content = df_content.drop_duplicates(subset=['article_id'], keep='first')
```

In [13]:
```
df_content[df_content.duplicated(subset=[  'article_id'], keep=False)].count()
```

Out[13]:
```
doc_body           0
doc_description    0
doc_full_name      0
doc_status         0
article_id         0
dtype: int64
```

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 [14]: len(list(df.groupby('article_id').title.count()))
```

```
Out[14]: 714
```

```
In [15]: df_content.shape[0]
```

```
Out[15]: 1051
```

```
In [16]: len(list(df.groupby('email').title.count()))
```

```
Out[16]: 5148
```

```
In [17]: df.shape[0]
```

```
Out[17]: 45993
```

```
In [18]: unique_articles = 714 # The number of unique articles that have at least one interactio
         total_articles = 1051 # The number of unique articles on the IBM platform
         unique_users = 5148 # The number of unique users
         user_article_interactions = 45993 # The number of user-article interactions
```

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 [19]: #findmost viewd article
         df.groupby(['article_id', 'title']).title.count().reset_index(name='count').sort_values
```

```
Out[19]:      article_id                                              title  count
         699      1429.0         use deep learning for image classification    937
         625      1330.0        insights from new york car accident reports    927
         701      1431.0                      visualize car data with brunel    671
         697      1427.0  use xgboost, scikit-learn & ibm watson machine...    643
         652      1364.0  predicting churn with the spss random tree alg...    627
```

```
In [20]: most_viewed_article_id = '1429.0' # The most viewed article in the dataset as a string
         max_views = 937 # The most viewed article in the dataset was viewed how many times?
```

```
In [21]: ## No need to change the code here - this will be helpful for later parts of the noteb
         # Run this cell to map the user email to a user_id column and remove the email column

         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[21]:

	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 [22]:
```
    ## 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 ______.`': user_a
        '`The maximum number of user-article interactions by any 1 user is ______.`': max_v
        '`The most viewed article in the dataset was viewed _____ times.`': max_views,
        '`The article_id of the most viewed article is ______.`': most_viewed_article_id,
        '`The number of unique articles that have at least 1 rating ______.`': unique_artic
        '`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!
```

1.1.2 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 [23]: def get_top_df(n, df=df):
             top_df = df.groupby(['article_id', 'title']).title.count().reset_index(name='count'

             return top_df

         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_articles = list(get_top_df(n, df).title)

             return top_articles # Return the top article titles from df (not df_content)

         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 = list(get_top_df(n,df).article_id.astype('str'))

             return top_articles # Return the top article ids
```

```
In [24]: print(get_top_articles(10))
         print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new york car accident reports', 'v
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '1170.0', '1162.0', '1304
```

```
In [25]: # 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.
```

1.1.3 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 [26]:
```
f = lambda x: 1
test = df.groupby(['user_id', 'article_id']).apply(f).unstack()
print(test.iloc[0].sum())
print(test.iloc[1].sum())
test
```

```
36.0
6.0
```

Out[26]:

article_id	0.0	2.0	4.0	8.0	9.0	12.0	14.0	15.0	\
user_id									
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
8	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	
9	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

11	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN
13	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN
14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
19	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
21	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN
22	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN
23	NaN	1.0	NaN	NaN	NaN	1.0	1.0	NaN
24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
26	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
27	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
28	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN
29	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
30	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
5120	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5121	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5122	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
5123	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5124	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5125	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5126	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5127	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5128	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5129	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5130	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5131	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5132	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5133	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5134	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5135	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5136	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5137	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5138	NaN	NaN	NaN	NaN	NaN	1.0	1.0	NaN
5139	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0
5140	NaN	1.0	NaN	NaN	NaN	1.0	NaN	NaN
5141	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5142	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN
5143	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5144	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5145	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5146	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
5147           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
5148           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
5149           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN

article_id  16.0    18.0     ...    1434.0  1435.0  1436.0  1437.0  1439.0  \
user_id                      ...
1              NaN     NaN   ...       NaN     NaN     1.0     NaN     1.0
2              NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
3              NaN     NaN   ...       NaN     NaN     1.0     NaN     NaN
4              NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5              NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
6              NaN     NaN   ...       NaN     NaN     1.0     NaN     NaN
7              NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
8              NaN     NaN   ...       NaN     NaN     1.0     NaN     NaN
9              1.0     NaN   ...       NaN     NaN     NaN     NaN     NaN
10             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
11             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
12             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
13             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
14             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
15             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
16             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
17             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
18             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
19             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
20             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
21             NaN     NaN   ...       NaN     NaN     1.0     1.0     NaN
22             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
23             1.0     NaN   ...       NaN     NaN     1.0     NaN     1.0
24             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
25             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
26             NaN     NaN   ...       NaN     NaN     1.0     NaN     NaN
27             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
28             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
29             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
30             NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
...            ...     ...   ...       ...     ...     ...     ...     ...
5120           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5121           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5122           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5123           NaN     NaN   ...       NaN     NaN     1.0     1.0     NaN
5124           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5125           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5126           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5127           NaN     NaN   ...       1.0     NaN     NaN     NaN     NaN
5128           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5129           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
5130           NaN     NaN   ...       NaN     NaN     NaN     NaN     NaN
```

5131	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5132	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5133	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5134	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5135	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5136	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5137	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5138	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5139	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5140	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5141	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5142	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5143	NaN	NaN	...	NaN	NaN	1.0	NaN	NaN
5144	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5145	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5146	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5147	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5148	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
5149	1.0	NaN	...	NaN	NaN	NaN	NaN	NaN

article_id	1440.0	1441.0	1442.0	1443.0	1444.0
user_id					
1	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN	NaN
17	NaN	NaN	NaN	NaN	NaN
18	NaN	NaN	NaN	NaN	NaN
19	NaN	NaN	NaN	NaN	NaN
20	NaN	NaN	NaN	NaN	NaN
21	NaN	NaN	NaN	NaN	NaN
22	NaN	NaN	NaN	NaN	NaN
23	NaN	NaN	NaN	NaN	NaN
24	1.0	NaN	NaN	NaN	NaN
25	NaN	NaN	NaN	NaN	NaN
26	NaN	NaN	NaN	NaN	NaN

```
27              NaN     NaN     NaN     NaN     NaN
28              NaN     NaN     NaN     NaN     NaN
29              NaN     NaN     NaN     NaN     NaN
30              NaN     NaN     NaN     NaN     NaN
...             ...     ...     ...     ...     ...
5120            NaN     NaN     NaN     NaN     NaN
5121            NaN     NaN     NaN     NaN     NaN
5122            NaN     NaN     NaN     NaN     NaN
5123            NaN     NaN     NaN     NaN     NaN
5124            NaN     NaN     NaN     NaN     NaN
5125            NaN     NaN     NaN     NaN     NaN
5126            NaN     NaN     NaN     NaN     NaN
5127            NaN     NaN     NaN     NaN     NaN
5128            NaN     NaN     NaN     NaN     NaN
5129            NaN     NaN     NaN     NaN     NaN
5130            NaN     NaN     NaN     NaN     NaN
5131            NaN     NaN     NaN     NaN     NaN
5132            NaN     NaN     NaN     NaN     NaN
5133            NaN     NaN     NaN     NaN     NaN
5134            NaN     NaN     NaN     NaN     NaN
5135            NaN     NaN     NaN     NaN     NaN
5136            NaN     NaN     NaN     NaN     NaN
5137            NaN     NaN     NaN     NaN     NaN
5138            NaN     NaN     NaN     NaN     NaN
5139            NaN     NaN     NaN     NaN     NaN
5140            NaN     NaN     NaN     NaN     NaN
5141            NaN     NaN     NaN     NaN     NaN
5142            NaN     NaN     NaN     NaN     NaN
5143            NaN     NaN     NaN     NaN     NaN
5144            NaN     NaN     NaN     NaN     NaN
5145            NaN     NaN     NaN     NaN     NaN
5146            NaN     NaN     NaN     NaN     NaN
5147            NaN     NaN     NaN     NaN     NaN
5148            NaN     NaN     NaN     NaN     NaN
5149            NaN     NaN     NaN     NaN     NaN

[5149 rows x 714 columns]
```

In [27]:
```
# 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 with 1 values
        an article and a 0 otherwise
        '''
        # Fill in the function here
        f = lambda x: 1

        user_item_df = df.groupby(['user_id', 'article_id']).apply(f).unstack().fillna(0)

        user_item = user_item_df

        return user_item # return the user_item matrix

    user_item = create_user_item_matrix(df)
```

```
In [28]: ## 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-article matr
         assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article ma
         assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 do
         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 [29]: 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 dot product
                             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_item_array = np.array(user_item)
```

```
    dot_prod_user_item = user_item_array.dot(np.transpose(user_item_array))

    user_idx = np.where(user_item.index == user_id)[0][0]
    similar_vals = dot_prod_user_item[user_idx]

    idxs = list(range(0, len(similar_vals)-1))
    similar_items = zip(idxs, similar_vals)

    # sort by similarity
    similar_items = sorted(similar_items, key=lambda x: x[1], reverse=True)

    # create list of just the ids
    most_similar_users = []
    for idx, val in similar_items:
        if 0 < val:
            most_similar_users.append( user_item.index[idx])

    # remove the own user's id
    if user_id in most_similar_users:
        most_similar_users.remove(user_id)

    return most_similar_users # return a list of the users in order from most to least
```

```
In [30]: # Do a spot check of your function
         print("The 10 most similar users to user 1 are: {}".format(find_similar_users(1)[:10]))
         print("The 5 most similar users to user 3933 are: {}".format(find_similar_users(3933)[:
         print("The 3 most similar users to user 46 are: {}".format(find_similar_users(46)[:3]))
```

```
The 10 most similar users to user 1 are: [3933, 23, 3782, 203, 4459, 131, 3870, 46, 4201, 49]
The 5 most similar users to user 3933 are: [1, 23, 3782, 203, 4459]
The 3 most similar users to user 46 are: [4201, 23, 3782]
```

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 [31]: user_item.head(3)
```

```
Out[31]: article_id  0.0     2.0     4.0     8.0     9.0     12.0    14.0    15.0    \
         user_id
         1              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
         3              0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0

         article_id  16.0    18.0     ...    1434.0  1435.0  1436.0  1437.0  1439.0  \
         user_id                      ...
         1              0.0     0.0   ...       0.0     0.0     1.0     0.0     1.0
```

```
2              0.0     0.0   ...       0.0     0.0     0.0     0.0     0.0
3              0.0     0.0   ...       0.0     0.0     1.0     0.0     0.0

article_id  1440.0  1441.0  1442.0  1443.0  1444.0
user_id
1              0.0     0.0     0.0     0.0     0.0
2              0.0     0.0     0.0     0.0     0.0
3              0.0     0.0     0.0     0.0     0.0

[3 rows x 714 columns]
```

In [32]:
```
user_item_row = user_item[user_item.index == 20]
article_ids = user_item_row.loc[:, user_item_row.any()].columns.astype(str)
article_ids
```

Out[32]:
```
Index(['232.0', '844.0', '1320.0'], dtype='object', name='article_id')
```

In [ ]:

In [33]:
```
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 of article
                    (this is identified by the title column)
    '''
    # Your code here
    article_names = list(df[df.article_id.isin(article_ids)].title.unique())

    return article_names # Return the article names associated with list of article ids


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 of article
                    (this is identified by the doc_full_name column in df_content)

    Description:
    Provides a list of the article_ids and article titles that have been seen by a user
```

```
    '''
    # Your code here
    user_item_row = user_item[user_item.index == user_id]
    article_ids = user_item_row.loc[:, user_item_row.any()].columns.astype(str)

    article_names = []
    for article_id in article_ids:
        article_names.extend(get_article_names([article_id]))


    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 them as rec
    Does this until m recommendations are found

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

    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
    similar_users = find_similar_users(user_id)

    seen_artcle_ids, _ = get_user_articles(user_id)

    recs = []
    flg = False
    for similar_user_id in similar_users:
        similar_user_seen_article_ids, _ =  get_user_articles(similar_user_id)

        for article_id in similar_user_seen_article_ids:

            if article_id not in recs:# and article_id not in seen_artcle_ids:
                recs.append(article_id)
```

```
                    if m <= len(recs):
                        flg=True
                        break
                if flg:
                    break
            if flg:
                break

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

```
In [34]: # Check Results
         get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

```
Out[34]: ['time series prediction using recurrent neural networks (lstms)',
          'deep learning with tensorflow course by big data university',
          'tensorflow quick tips',
          'sector correlations shiny app',
          'new shiny cheat sheet and video tutorial',
          'introduction to market basket analysis in\xa0python',
          'tidyverse practice: mapping large european cities',
          'fighting gerrymandering: using data science to draw fairer congressional districts',
          'python for loops explained (python for data science basics #5)',
          'introducing ibm watson studio ']
```

```
In [35]: # Test your functions here - No need to change this code - just run this cell
         assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.0
         assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): un
         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 demographic
         assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.0', '14
         assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-re
         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 [36]: get_user_articles(4201)
```

```
Out[36]: (Index(['2.0', '43.0', '76.0', '89.0', '109.0', '184.0', '224.0', '295.0',
                 '316.0', '336.0', '351.0', '525.0', '569.0', '692.0', '705.0', '868.0',
                 '933.0', '962.0', '967.0', '1014.0', '1017.0', '1054.0', '1162.0',
                 '1164.0', '1170.0', '1185.0', '1192.0', '1293.0', '1305.0', '1351.0',
                 '1360.0', '1364.0', '1368.0', '1393.0', '1423.0', '1427.0', '1429.0',
                 '1430.0', '1436.0', '1439.0'],
                dtype='object', name='article_id'),
         ['this week in data science (april 18, 2017)',
          'deep learning with tensorflow course by big data university',
          'this week in data science (may 2, 2017)',
          'top 20 r machine learning and data science packages',
          'tensorflow quick tips',
          'improving the roi of big data and analytics through leveraging new sources of data',
          'using apply, sapply, lapply in r',
          'awesome deep learning papers',
          'leverage python, scikit, and text classification for behavioral profiling',
          'challenges in deep learning',
          'do i need to learn r?',
          'new shiny cheat sheet and video tutorial',
          'how can data scientists collaborate to build better business',
          '15 page tutorial for r',
          'word2vec in data products',
          'how to write the first for loop in r',
          'workflow in r',
          'data visualization with r: scrum metrics',
          'ml algorithm != learning machine',
          '1448    i ranked every intro to data science course on...\nName: title, dtype: objec
          'the pandas data analysis library',
          'access mysql with r',
          'analyze energy consumption in buildings',
          'analyze open data sets with pandas dataframes',
          'apache spark lab, part 1: basic concepts',
          'classify tumors with machine learning',
          'country statistics: airports',
          'finding optimal locations of new store using decision optimization',
          'gosales transactions for naive bayes model',
          'model bike sharing data with spss',
          'pixieapp for outlier detection',
          'predicting churn with the spss random tree algorithm',
          'putting a human face on machine learning',
          'the nurse assignment problem',
          'use sql with data in hadoop python',
          'use xgboost, scikit-learn & ibm watson machine learning apis',
          'use deep learning for image classification',
          'using pixiedust for fast, flexible, and easier data analysis and experimentation',
          'welcome to pixiedust',
          'working with ibm cloud object storage in r'])
```

In [100]:
```
user_sim=(user_item.loc[1,:].dot(user_item.T))
user_sim = user_sim.drop(1).reset_index()
user_articles=user_item.sum(axis=1).reset_index()
user_sim=user_sim.merge(user_articles, how='left',on='user_id')
user_sim.columns=['neightbor_id','similarity', 'num_interactions']
user_sim.sort_values(['similarity','num_interactions'], ascending=False)
user_sim.head()
```

Out[100]:

	neightbor_id	similarity	num_interactions
0	2	2.0	6.0
1	3	6.0	40.0
2	4	3.0	26.0
3	5	0.0	3.0
4	6	4.0	18.0

In [101]:
```
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 the provide
                    num_interactions - the number of articles viewed by the user - if

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

    '''
    # Your code here

    user_sim=(user_item.loc[user_id,:].dot(user_item.T))
    user_sim = user_sim.drop(user_id).reset_index()
    user_articles=user_item.sum(axis=1).reset_index()
    user_sim=user_sim.merge(user_articles, how='left',on='user_id')
    user_sim.columns=['neightbor_id','similarity', 'num_interactions']
    neighbors_df = user_sim.sort_values(['similarity','num_interactions'], ascending=F


    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 title

    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 them as re
    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.

    '''
    # Your code here
    similar_users = get_top_sorted_users(user_id)

    recs = []
    rec_names = []
    flg = False
    for similar_user_id in similar_users.neightbor_id:
        similar_user_seen_article_ids, similar_user_seen_article_titles =  get_user_ar

        for idx, article_id in enumerate(similar_user_seen_article_ids):

            if article_id not in recs:# and article_id not in seen_artcle_ids:
                recs.append(article_id)
                rec_names.append(similar_user_seen_article_titles[idx])

                if m <= len(recs):
                    flg=True
                    break
            if flg:
                break
        if flg:
            break

    return recs, rec_names
```

In [102]:
```
# Quick spot check - don't change this code - just use it to test your functions
```

```
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 names:")
print(rec_names)
```

```
The top 10 recommendations for user 20 are the following article ids:
['12.0', '14.0', '29.0', '33.0', '43.0', '51.0', '109.0', '111.0', '130.0', '142.0']

The top 10 recommendations for user 20 are the following article names:
['timeseries data analysis of iot events by using jupyter notebook', 'got zip code data? prep it
```

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 [103]: get_top_sorted_users(1).head()
```

Out[103]:

	neightbor_id	similarity	num_interactions
3931	3933	35.0	35.0
21	23	17.0	135.0
3780	3782	17.0	135.0
201	203	15.0	96.0
4457	4459	15.0	96.0

```
In [84]:  get_top_sorted_users(131).head(10)
```

Out[84]:

	neightbor_id	similarity	num_interactions
3868	3870	74.0	75.0
3780	3782	39.0	135.0
22	23	38.0	135.0
201	203	33.0	96.0
4457	4459	33.0	96.0
48	49	29.0	101.0
3695	3697	29.0	100.0
97	98	29.0	97.0
3762	3764	29.0	97.0
3908	3910	25.0	60.0

```
In [109]: ### Tests with a dictionary of results

          user1_most_sim = get_top_sorted_users(1).iloc[0].neightbor_id # Find the user that is
          user131_10th_sim = get_top_sorted_users(131).iloc[10].neightbor_id # Find the 10th mos

In [112]: ## 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.

For new user we can use rank base recommendation, because it's not able to use above functions that is user-user based collaborative filtering recommend without any interaction.

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 [113]: new_user = '0.0'

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


          new_user_recs = get_top_article_ids(10)
```

```
In [114]: assert set(new_user_recs) == set(['1314.0','1429.0','1293.0','1427.0','1162.0','1364.0

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

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

1.1.4 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.

1.1.5 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 [45]: 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?

1.1.6 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.

1.1.7 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 [46]: # make recommendations for a brand new user


         # make a recommendations for a user who only has interacted with article id '1427.0'
```

1.1.8 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 [47]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')

In [48]: # quick look at the matrix
         user_item_matrix.head()
```

Out[48]:

article_id	0.0	100.0	1000.0	1004.0	1006.0	1008.0	101.0	1014.0	1015.0	\
user_id										
1	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	
3	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	
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

```
article_id  1016.0  ...    977.0  98.0  981.0  984.0  985.0  986.0  990.0  \
user_id             ...
1              0.0  ...      0.0   0.0    1.0    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
3              0.0  ...      1.0   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
5              0.0  ...      0.0   0.0    0.0    0.0    0.0    0.0    0.0

article_id  993.0  996.0  997.0
user_id
1             0.0    0.0    0.0
2             0.0    0.0    0.0
3             0.0    0.0    0.0
4             0.0    0.0    0.0
5             0.0    0.0    0.0

[5 rows x 714 columns]
```

```
In [127]: user_item_matrix.shape
```

```
Out[127]: (5149, 714)
```

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 [132]: # Perform SVD on the User-Item Matrix Here

          u, s, vt = np.linalg.svd(user_item_matrix)

          print(u.shape, s.shape, vt.shape, user_item_matrix.shape)
```

```
(5149, 5149) (714,) (714, 714) (5149, 714)
```

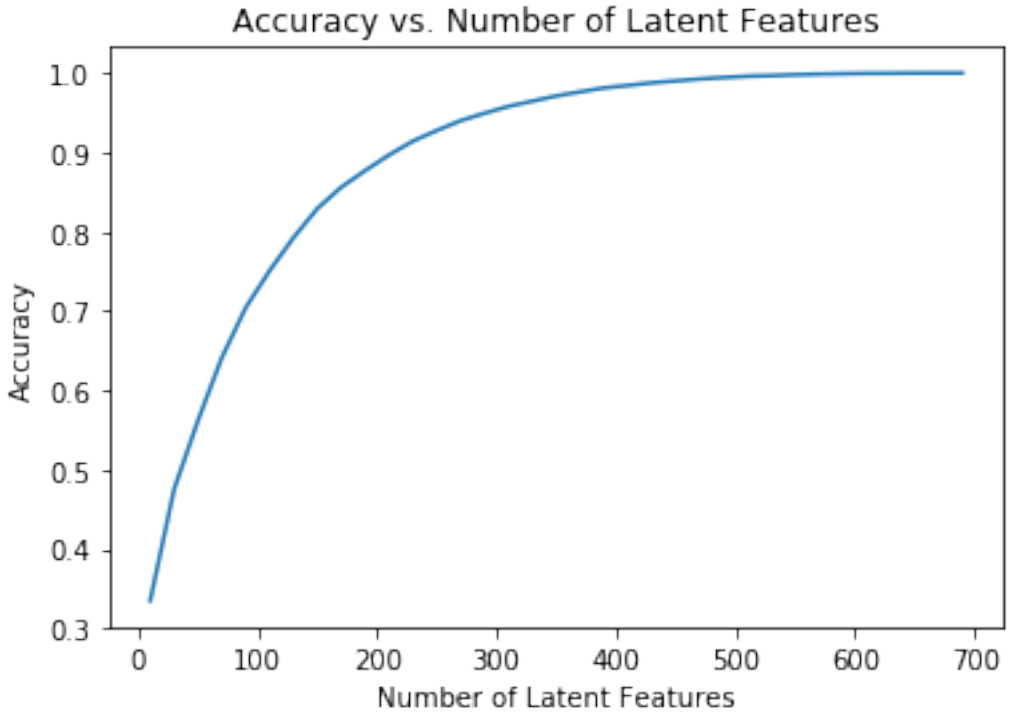
Using SVD

There is no missing value in this user-item matrix so we can use SVD, if we want to use SVD with missing values we need to use another algoritm like Funk SVD.

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 [131]: 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');
```



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 [103]:
```
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 each column)
    user_item_test - a user-item matrix of the testing dataframe
                     (unique users for each row and unique articles for each column)
    test_idx - all of the test user ids
    test_arts - all of the test article ids

    '''
    # Your code here
    user_item_train = create_user_item_matrix(df_train)
    user_item_test = create_user_item_matrix(df_test)

    train_idx = list(user_item_train.index)
    test_idx = list(user_item_test.index)

    intersect_idx = np.intersect1d(train_idx, test_idx)

    train_arts = list(user_item_train.columns)
    test_arts = list(user_item_test.columns)
    intersect_arts = np.intersect1d(train_arts, test_arts)

    user_item_test = user_item_test.loc[intersect_idx, intersect_arts]

    return user_item_train, user_item_test, test_idx, test_arts

user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_user_item

len(test_idx), len(test_arts), user_item_test.shape, user_item_train.shape
```

Out[103]:
```
(682, 574, (20, 574), (4487, 714))
```

In [107]:
```
# 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, # letter here,
    'How many users in the test set are we not able to make predictions for because of
    'How many movies can we make predictions for in the test set?': b, # letter here,
    'How many movies in the test set are we not able to make predictions for because o
}

t.sol_4_test(sol_4_dict)
```

```
Awesome job!  That's right!  All of the test movies are in the training data, but there are only
```

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 [133]:
```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(user_item_train) # fit svd similar to above
u_train.shape, s_train.shape, vt_train.shape, user_item_train.shape
```

Out[133]:
```
((4487, 4487), (714,), (714, 714), (4487, 714))
```

In [119]:
```
# Use these cells to see how well you can use the training
# decomposition to predict on test data
np.array(user_item_test).shape
```

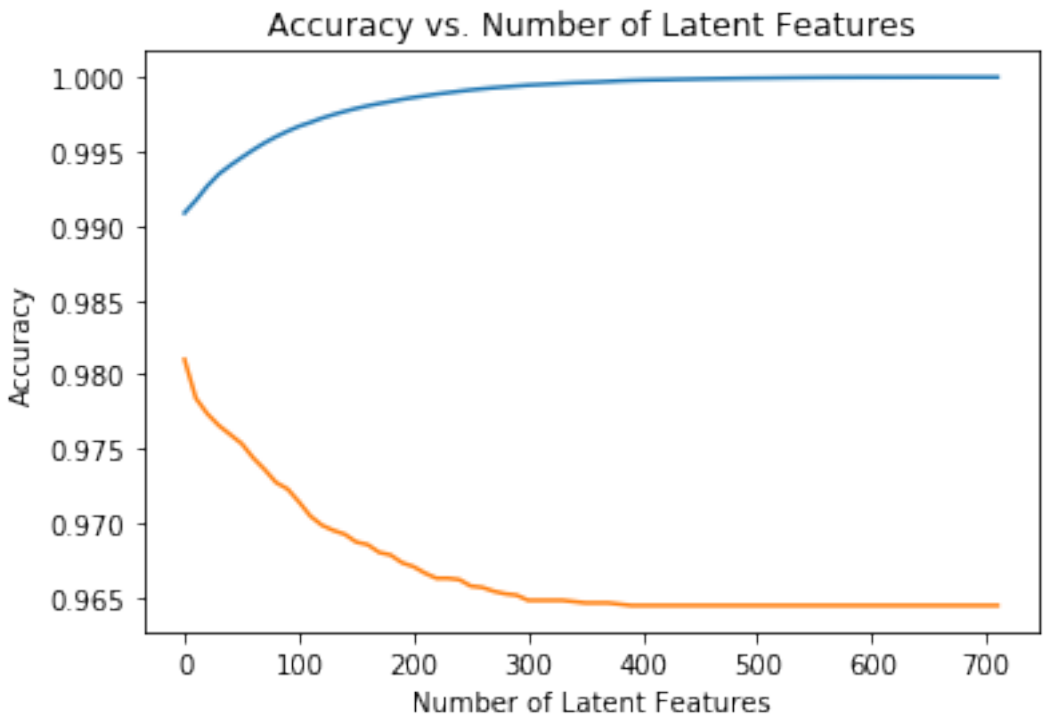
Out[119]:
```
(20, 574)
```

In [140]:
```
u_test, s_test, vt_test = np.linalg.svd(user_item_test)
u_test.shape, s_test.shape, vt_test.shape
```

Out[140]:
```
((20, 20), (20,), (574, 574))
```

In [165]:
```
num_latent_feats = np.arange(0,714,10)

#create test data set
row_idx = user_item_train.index.isin(test_idx)
col_idx = user_item_train.columns.isin(test_arts)
u_test = u_train[row_idx, :]
```

```
vt_test = vt_train[:, col_idx]
s_test= np.diag(s_train[:u_test.shape[0]])

s_test.shape, u_test.shape, vt_test.shape

sum_errs_train = []
sum_errs_test = []

for k in num_latent_feats:
    # restructure with k latent features
    s_new, u_train_new, vt_train_new = np.diag(s_train[:k]), u_train[:, :k], vt_train[
    u_test_new, vt_test_new = u_test[:, :k], vt_test[:k, :]

    # take dot product
    train_est = np.around(np.dot(np.dot(u_train_new, s_new), vt_train_new))
    test_est = np.around(np.dot(np.dot(u_test_new, s_new), vt_test_new))


    # compute error for each prediction to actual value
    diffs_train = np.subtract(user_item_train, train_est)
    diffs_test = np.subtract(user_item_test, test_est)


    # total errors and keep track of them
    sum_errs_train.append( np.sum(np.sum(np.abs(diffs_train))))
    sum_errs_test.append( np.sum(np.sum(np.abs(diffs_test))))


plt.plot(num_latent_feats, 1 - np.array(sum_errs_train)/user_item_train.size, label='t
plt.plot(num_latent_feats, 1 - np.array(sum_errs_test)/user_item_test.size, label='tes

plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
```



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?

Comment on the results

Using more Latent Features seems to be overfitting the SVD model. There are not enough data between train and test dataset (only 20 users) to create good generalized recommendation model. Looks user's view count is not enogh to create acculate model.

We can cerate A/B test between SVD recommendation group and Rank base recommendation group to find better performance.

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!

1.2 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.

1.3 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 [115]: from subprocess import call
          call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

```
Out[115]: 0
```

```
In [ ]:
```

```
In [ ]:
```