Recommendations_with_IBM

January 31, 2020

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

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
        from matplotlib import rcParams
        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']
```

```
In [2]: # df_content['article_id'].isin([1024, 1176, 1305, 1314, 1422, 1427]).sum()
        set(df[df['article_id'].isin([1024, 1176, 1305, 1314, 1422, 1427])]['title'])
```

```
Out[2]: {'build a python app on the streaming analytics service',
         'gosales transactions for naive bayes model',
         'healthcare python streaming application demo',
         'use r dataframes & ibm watson natural language understanding',
         'use xgboost, scikit-learn & ibm watson machine learning apis',
         'using deep learning to reconstruct high-resolution audio'}
```

In [3]:
```
plt.style.use('seaborn-white')
rcParams['axes.labelsize'] = 'x-large'
rcParams['axes.edgecolor'] = 'black'
rcParams['axes.facecolor'] = 'white'
rcParams['axes.titlesize'] = 'x-large'
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
rcParams['axes.xmargin'] = 0.02
rcParams['axes.ymargin'] = 0.02

rcParams['axes.grid'] = True
rcParams['grid.linestyle'] = ':'
rcParams['grid.alpha'] = 0.2
rcParams['grid.color'] = 'black'

rcParams['figure.titlesize'] = 'x-large'
rcParams['figure.edgecolor']= 'black'
rcParams['figure.facecolor'] = 'white'
rcParams['figure.figsize'] = [10, 6]

rcParams['ytick.labelsize'] = 'large'
rcParams['xtick.labelsize'] = 'large'
```

In [4]:
```
# Show df to get an idea of the data
df.head()
```

Out[4]:
```
   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 [5]:
```
# Show df_content to get an idea of the data
df_content.head()
```

Out[5]:
```
                                            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
```

In [6]:
```
df_content.shape
```

Out[6]: (1056, 5)

In [7]:
```
df_content['article_id'].nunique()
```

Out[7]: 1051

In [8]:
```
df.shape
```

Out[8]: (45993, 3)

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.

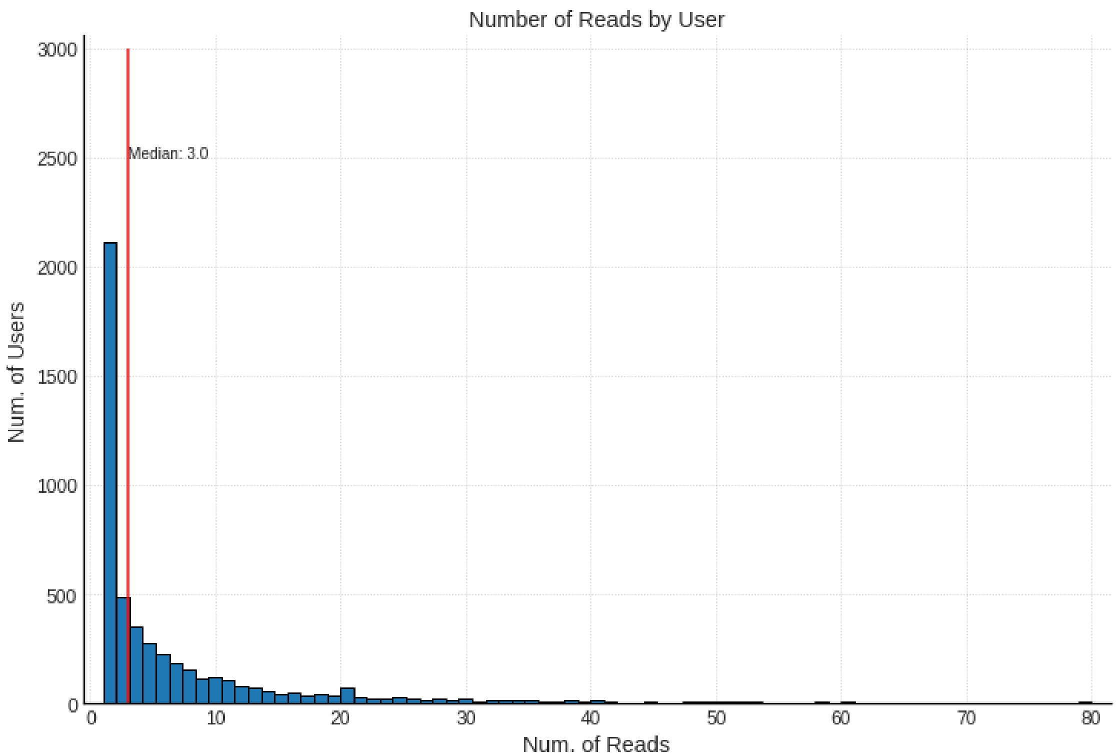
1. What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

In [9]:
```
# df['article_id'] = df['article_id'].astype(np.int).astype('str')
```

In [10]:
```
df.head()
```

Out[10]:
```
   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 [13]:
```
median = df['email'].value_counts().median()
plt.figure(figsize=(12,8))
df['email'].value_counts()[50:].plot(kind='hist', bins=75, edgecolor='k');
plt.title("Number of Reads by User");
plt.ylabel('Num. of Users');
plt.xlabel('Num. of Reads');
plt.vlines(x=median, ymin=0, ymax=3000, color='red');
plt.annotate(xy=(median, 2500), s=f'Median: {median}');
```

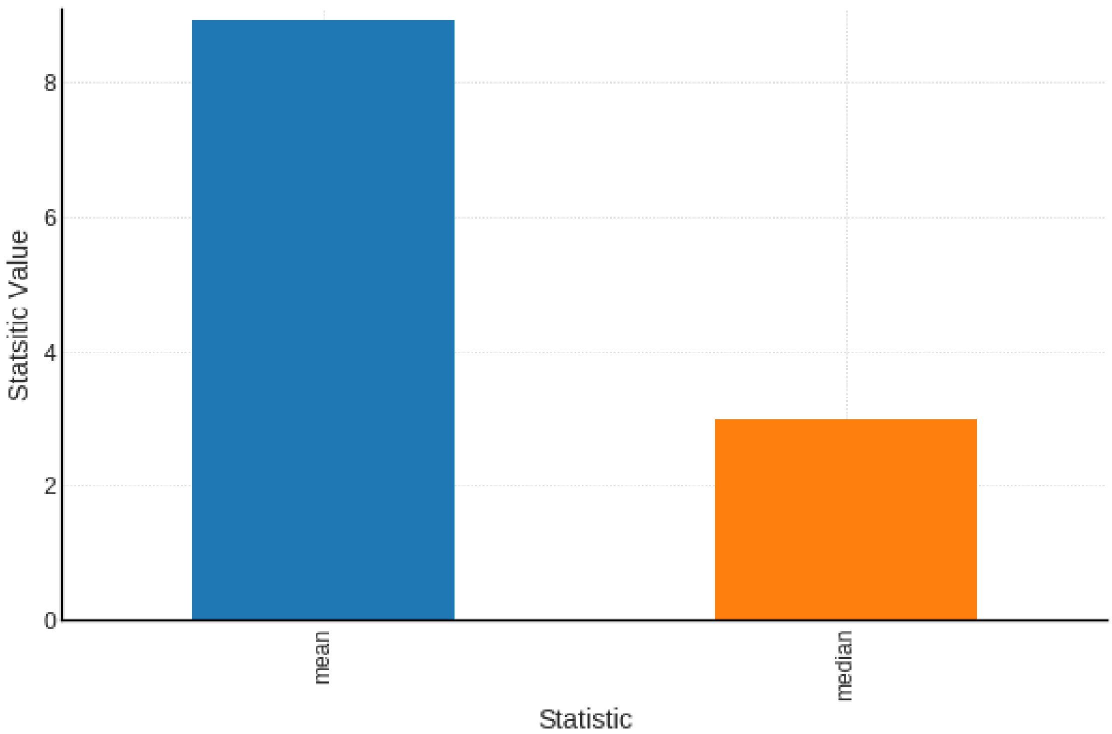


In [14]:
```
# df.groupby("email").count().sort_values(by='article_id', ascending=False)
```

In [15]:
```
df['email'].value_counts().agg(['mean', 'median'])
```

Out[15]:
```
mean      8.930847
median    3.000000
Name: email, dtype: float64
```

In [16]:
```
# df['email'].value_counts().agg(['mean', 'median']).plot(kind='bar', edgecolor='k');
df['email'].value_counts().agg(['mean', 'median']).plot.bar();
plt.ylabel("Statsitic Value");
plt.xlabel('Statistic');
```




```
In [17]: # Fill in the median and maximum number of user_article interactios below

         median_val = df['email'].value_counts().median() # 50% of individuals interact with ___
         max_views_by_user = df['email'].value_counts().max() # The maximum number of user-artic
```

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

```
In [18]: # explore duplicate articles
         df_content[df_content.duplicated(subset='article_id', keep=False)]
```

```
Out[18]:                                              doc_body  \
         50   Follow Sign in / Sign up Home About Insight Da...
         221  * United States\r\n\r\nIBMő * Site map\r\n\r\n...
         232  Homepage Follow Sign in Get started Homepage *...
         365  Follow Sign in / Sign up Home About Insight Da...
         399  Homepage Follow Sign in Get started * Home\r\n...
         578  This video shows you how to construct queries ...
         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  \
         50                  Community Detection at Scale
```

```
221    When used to make sense of huge amounts of con...
232    If you are like most data scientists, you are ...
365    During the seven-week Insight Data Engineering...
399    Todays world of data science leverages data f...
578    This video shows you how to construct queries ...
692    One of the earliest documented catalogs was co...
761    Todays 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_id
50                            Graph-based machine learning       Live          50
221   How smart catalogs can turn the big data flood...       Live         221
232   Self-service data preparation with IBM Data Re...       Live         232
365                           Graph-based machine learning       Live          50
399   Using Apache Spark as a parallel processing fr...       Live         398
578                                  Use the Primary Index       Live         577
692   How smart catalogs can turn the big data flood...       Live         221
761   Using Apache Spark as a parallel processing fr...       Live         398
970                                  Use the Primary Index       Live         577
971   Self-service data preparation with IBM Data Re...       Live         232
```

In [19]:
```
# check shape before sroppign duplicates
df_content.shape
```

Out[19]:
```
(1056, 5)
```

In [20]:
```
# check shape with duplicates removed
df_content.drop_duplicates(keep=False)
```

Out[20]:
```
                                               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...
5      Compose is all about immediacy. You want a new...
6      UPGRADING YOUR POSTGRESQL TO 9.5Share on Twitt...
7      Follow Sign in / Sign up 135 8 * Share\r\n * 1...
8      * Host\r\n * Competitions\r\n * Datasets\r\n *...
9      THE GRADIENT FLOW\r\nDATA / TECHNOLOGY / CULTU...
10     OFFLINE-FIRST IOS APPS WITH SWIFT & PART 1: TH...
11     Warehousing data from Cloudant to dashDB great...
12     Skip to main content IBM developerWorks / Deve...
13     Maureen McElaney Blocked Unblock Follow Follow...
14     Raj Singh Blocked Unblock Follow Following Dev...
15     * Home\r\n * Community\r\n * Projects\r\n * Bl...
16     * Home\r\n * Research\r\n * Partnerships and C...
17     Enterprise Pricing Articles Sign in Free 30-Da...
```

```
18      Homepage Follow Sign in / Sign up * Home\r\n *...
19      METRICS MAVEN: MODE D'EMPLOI - FINDING THE MOD...
20      Homepage Follow Sign in / Sign up Homepage * H...
21      Raj Singh Blocked Unblock Follow Following Dev...
22      IMPORTING JSON DOCUMENTS WITH NOSQLIMPORT\r\nG...
23      This video shows you how to build and query a ...
24      THE CONVERSATIONAL INTERFACE IS THE NEW PARADI...
25      Skip navigation Upload Sign in SearchLoading...
26      GOOGLE RESEARCH BLOG The latest news from Rese...
27      Skip navigation Upload Sign in SearchLoading...
28      ACCESS DENIED\r\nSadly, your client does not s...
29      Homepage Follow Sign in / Sign up Homepage * H...
...                                                   ...
1026    Enterprise Pricing Articles Sign in Free 30-Da...
1027    Skip navigation Sign in SearchLoading...\r\n\r...
1028    Compose The Compose logo Articles Sign in Free...
1029    Follow Sign in / Sign up * Home\r\n * About In...
1030    Homepage Follow Sign in / Sign up Homepage * H...
1031    Develop in the cloud at the click of a button!...
1032    BLAZINGLY FAST GEOSPATIAL QUERIES WITH REDIS\r...
1033    Blog Home Dataquest.io Learn Data Science in Y...
1034    DATALAYER: MANAGING (OR NOT) THE DATA IN IMMUT...
1035    Skip to contentWin-Vector Blog\r\n\r\nThe Win-...
1036    This work is licensed under a Creative Commons...
1037                                                  NaN
1038    The relational database has been the dominant ...
1039    Skip to main content IBM developerWorks / Deve...
1040    Skip to contentDinesh Nirmal's Blog\r\n\r\nA b...
1041    Compose The Compose logo Articles Sign in Free...
1042    Glynn Bird Blocked Unblock Follow Following De...
1043    MENU\r\nClose\r\nSubscribe SubscribeREDUCING O...
1044    Homepage IBM Watson Data Lab Follow Sign in / ...
1045    Although it is built around a JavaScript engin...
1046    Margriet Groenendijk Blocked Unblock Follow Fo...
1047    Homepage Follow Sign in / Sign up Homepage * H...
1048    Homepage Follow Sign in Get started * Home\r\n...
1049    * \r\n * \r\n * \r\n * \r\n * \r\n * \r\n * \r...
1050    1A SPEED GUIDE TO REDIS LUA SCRIPTING\r\nShare...
1051    PouchDB-find is a new API and syntax that allo...
1052    We compare discriminative and generative learn...
1053    Essays about data, building products and boots...
1054                                                  NaN
1055    Homepage Follow Sign in / Sign up Homepage * H...

                                          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...
5                 Using Compose's PostgreSQL data browser.
6       Upgrading your PostgreSQL deployment to versio...
7       For a company like Slack that strives to be as...
8       Kaggle is your home for data science. Learn ne...
9       [A version of this post appears on the OReill...
10      Apple's sample app, Food Tracker, taught you i...
11      Replicating data to a relational dashDB databa...
12      This recipe showcases how one can analyze the ...
13      Theres a reason youve been hearing a lot abo...
14      Who are those people lurking behind the statis...
15      Early methods to integrate machine learning us...
16      The performance of supervised predictive model...
17      We've always considered MySQL as a potential C...
18      It has never been easier to build AI or machin...
19      In our Metrics Maven series, Compose's data sc...
20      It is often useful to use RStudio for one piec...
21      Youre doing your data a disservice if you don...
22      Introducing nosqlimport, an npm module to help...
23      This video shows you how to build and query a ...
24      Botkit provides a simple framework to handle t...
25      Want to learn more about how we created the Da...
26      Much of driving is spent either stuck in traff...
27      This talk assumes you have a basic understandi...
28      In this paper, we propose gcForest, a decision...
29      Im very happy and proud to announce that IBM ...
...                                                   ...
1026    Varun Singh, a software engineer at IBM's Wats...
1027    This video shows you how to create and adminis...
1028    With the latest 0.2.1 version of Transporter, ...
1029    Audio super-resolution aims to reconstruct a h...
1030    Since then, this metric has been ubiquitously ...
1031    Build a word game app and see how to manage an...
1032    Use Redis and and Python scripts to speed your...
1033    In this post, youll learn to query, update, a...
1034    Adron Hall of Thrashing Code and Home Depot, t...
1035    Describes the use of Laplace noise in machine ...
1036    A full guide to Elasticsearch, the real-time d...
1037    See how quick and easy it is to set up a dashD...
1038    The relational database has been the dominant ...
1039    Building your first data warehouse doesnt hav...
1040    In my last blog Business differentiation thro...
1041    MongoDB's aggregation pipeline makes finding d...
1042    Which write API endpoint is the right write ca...
1043            Nothing spoils a plot like (too much) data.
1044    Getting started with custom visualizations, si...
1045    Although it is built around a JavaScript engin...
```

```
1046  Last week I attended the GeoPython conference ...
1047  In this post, we will go through how to read a...
1048  As more devices become internet enabled, harne...
1049  Continuing my previous work on exploring Arlin...
1050  Lua is a compact language which can be embedde...
1051  PouchDB uses MapReduce as its default search m...
1052  We compare discriminative and generative learn...
1053  In order to demystify some of the magic behind...
1054  Learn how to use IBM dashDB as data store for ...
1055  Once you get used to developing in a Notebook ...
```

```
                                          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
5                 Browsing PostgreSQL Data with Compose       Live           5
6                     Upgrading your PostgreSQL to 9.5       Live           6
7                               Data Wrangling at Slack       Live           7
8                                Data Science Bowl 2017       Live           8
9     Using Apache Spark to predict attack vectors a...       Live           9
10    Offline-First iOS Apps with Swift & Cloudant S...       Live          10
11                          Warehousing GeoJSON documents       Live          11
12    Timeseries Data Analysis of IoT events by usin...       Live          12
13    Bridging the Gap Between Python and Scala Jupy...       Live          13
14    Got zip code data? Prep it for analytics.  IB...      Live          14
15    Apache Spark 2.0: Extend Structured Streaming...      Live          15
16    Higher-order Logistic Regression for Large Dat...       Live          16
17                          Compose for MySQL now for you       Live          17
18    The Greatest Public Datasets for AI  Startup ...      Live          18
19                        Finding the Mode in PostgreSQL       Live          19
20    Working interactively with RStudio and noteboo...       Live          20
21    Mapping for Data Science with PixieDust and Ma...       Live          21
22              Move CSVs into different JSON doc stores       Live          22
23    Tutorial: How to build and query a Cloudant ge...       Live          23
24     The Conversational Interface is the New Paradigm       Live          24
25                 Creating the Data Science Experience       Live          25
26    Using Machine Learning to predict parking diff...       Live          26
27             Getting The Best Performance With PySpark       Live          27
28    Deep Forest: Towards An Alternative to Deep Ne...       Live          28
29                           Experience IoT with Coursera       Live          29
...                                                 ...        ...         ...
1026          Redis and MongoDB in the biomedical domain       Live        1021
1027  Create and administer a data catalog using IBM...       Live        1022
1028  How to move data with Compose Transporter - Fr...       Live        1023
1029  Using Deep Learning to Reconstruct High-Resolu...       Live        1024
1030             Data tidying in Data Science Experience       Live        1025
```

```
1031  Build a simple word game app using Cloudant on...       Live         1026
1032       Blazingly Fast Geospatial Queries with Redis       Live         1027
1033  Working with SQLite Databases using Python and...       Live         1028
1034  DataLayer Conference: Managing (or not) the Da...       Live         1029
1035  Laplace noising versus simulated out of sample...       Live         1030
1036                                The Definitive Guide       Live         1031
1037                      Get started with dashDB on Bluemix       Live         1032
1038       The Many Flavors of NoSQL at That Conference       Live         1033
1039   Your First Data Warehouse Is Easy. Meet the ODS.       Live         1034
1040                  Machine Learning for the Enterprise.       Live         1035
1041                Finding Duplicate Documents in MongoDB       Live         1036
1042  Piecemeal, Bulk, or Batch?  IBM Watson Data L...       Live         1037
1043                 Reducing overplotting in scatterplots       Live         1038
1044  You Too Can Make Magic (in Jupyter Notebooks w...       Live         1039
1045  How I Stopped Worrying & Learned to Love the M...       Live         1040
1046  Mapping All the Things with Python  IBM Watso...       Live         1041
1047  Use IBM Data Science Experience to Read and Wr...       Live         1042
1048  Use IoT data in Streams Designer for billing a...       Live         1043
1049                           Mapping Points with Folium       Live         1044
1050                  A Speed Guide To Redis Lua Scripting       Live         1045
1051               A look under the covers of PouchDB-find       Live         1046
1052  A comparison of logistic regression and naive ...       Live         1047
1053  What I Learned Implementing a Classifier from ...       Live         1048
1054                                  Use dashDB with Spark       Live         1049
1055  Jupyter Notebooks with Scala, Python, or R Ker...       Live         1050

[1056 rows x 5 columns]
```

In [21]:
```
# Remove any rows that have the same article_id - only keep the first
df_content.drop_duplicates(subset='article_id', inplace=True)
```

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 [22]:
```
df.groupby(['article_id']).count().sum()
```

Out[22]:
```
title    45993
email    45976
dtype: int64
```

In [23]:
```
unique_articles = df['article_id'].nunique() # The number of unique articles that have
total_articles = df_content['article_id'].shape[0] # The number of unique articles on t
unique_users = df['email'].nunique() # The number of unique users
user_article_interactions = df.shape[0] # 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 [24]:
```
# number of max views
df['article_id'].value_counts()[1429]
```

Out[24]: 937

In [25]:
```
# most viewed article id
df['article_id'].value_counts().index[0]
```

Out[25]: 1429.0

In [26]:
```
most_viewed_article_id = str(df['article_id'].value_counts().index[0]) # The most viewe
max_views = df['article_id'].value_counts()[1429] # The most viewed article in the data
```

In [27]:
```
## No need to change the code here - this will be helpful for later parts of the notebo
# 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[27]:

	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 [28]:
```
## 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!
```

In [29]:
```
sol_1_dict
```

Out[29]:
```
{'`50% of individuals have _____ or fewer interactions.`': 3.0,
 '`The total number of user-article interactions in the dataset is ______.`': 45993,
 '`The maximum number of user-article interactions by any 1 user is ______.`': 364,
 '`The most viewed article in the dataset was viewed _____ times.`': 937,
 '`The article_id of the most viewed article is ______.`': '1429.0',
 '`The number of unique articles that have at least 1 rating ______.`': 714,
 '`The number of unique users in the dataset is ______`': 5148,
 '`The number of unique articles on the IBM platform`': 1051}
```

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 [30]:
```
# convert dtype
df['article_id'] = df['article_id'].astype(np.int32)
```

In [31]:
```
# map counts of articles read in to `df` DataFrame
df['article_counts'] = df['article_id'].map(df['article_id'].value_counts())
```

In [32]:
```
# sort values, then drop duplicates articles
top_df = df.sort_values(by='article_counts', ascending=False).drop_duplicates(subset='a
top_df.iloc[:10, :]
```

Out[32]:
```
       article_id                                        title  user_id  \
15728        1429      use deep learning for image classification       72
41229        1330      insights from new york car accident reports     4637
```

```
6078         1431                       visualize car data with brunel     1427
18447        1427  use xgboost, scikit-learn & ibm watson machine...      639
29909        1364  predicting churn with the spss random tree alg...     3477
37039        1314       healthcare python streaming application demo     4145
5300         1293  finding optimal locations of new store using d...       23
10764        1170           apache spark lab, part 1: basic concepts       67
23798        1162             analyze energy consumption in buildings     1223
9566         1304  gosales transactions for logistic regression m...     1481

       article_counts
15728             937
41229             927
6078              671
18447             643
29909             627
37039             614
5300              572
10764             565
23798             512
9566              483
```

In [33]:
```python
# top_df.iloc[-50:-1, :]
```

In [34]:
```python
top_df.iloc[:10, :]['article_id'].tolist()
```

Out[34]:
```
[1429, 1330, 1431, 1427, 1364, 1314, 1293, 1170, 1162, 1304]
```

In [35]:
```python
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

    '''
    # sort values, then drop duplicate articles
    top_df = df.sort_values(by='article_counts', ascending=False).drop_duplicates(subse
    top_articles = top_df.iloc[:n, :]['title'].tolist()

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

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

        '''
        top_df = df.sort_values(by='article_counts', ascending=False).drop_duplicates(subse
        top_article_ids = top_df.iloc[:n, :]['article_id'].tolist()

        return top_article_ids # Return the top article ids
```

```
In [36]: print(get_top_articles(10))
         print('\n')
         print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new york car accident reports', 'v


[1429, 1330, 1431, 1427, 1364, 1314, 1293, 1170, 1162, 1304]
```

```
In [37]: # 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 [38]: df.head()
```

```
Out[38]:    article_id                                              title  user_id  \
         0        1430  using pixiedust for fast, flexible, and easier...        1
         1        1314       healthcare python streaming application demo        2
         2        1429         use deep learning for image classification        3
         3        1338          ml optimization using cognitive assistant        4
         4        1276          deploy your python model as a restful api        5

            article_counts
         0             336
         1             614
         2             937
         3             382
         4             347
```

```
In [39]: user_item_df = df.groupby(['user_id','article_id'])['title'].count().unstack()
         user_item_df.head()
```

```
Out[39]: article_id  0     2     4     8     9     12    14    15    16    18    ...   \
         user_id                                                              ...
         1            NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
         2            NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
         3            NaN   NaN   NaN   NaN   NaN   1.0   NaN   NaN   NaN   NaN  ...
         4            NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
         5            NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...

         article_id  1434  1435  1436  1437  1439  1440  1441  1442  1443  1444
         user_id
         1            NaN   NaN   1.0   NaN   1.0   NaN   NaN   NaN   NaN   NaN
         2            NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
         3            NaN   NaN   1.0   NaN   NaN   NaN   NaN   NaN   NaN   NaN
         4            NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
         5            NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

         [5 rows x 714 columns]
```

```
In [40]: # np.where(user_item_df.isnull(), 0, 1)
         # user_item_df.replace(np.nan, 0)
```

```
In [41]: user_item_df.shape
```

```
Out[41]: (5149, 714)
```

```
In [42]: user_item_df.loc[1, :].notnull().sum()
```

```
Out[42]: 36
```

```
In [43]: user_item_matrix = np.where(user_item_df.notnull(), 1, 0)
         # user_item_matrix = user_item_df.replace(np.nan, 0).values
```

In [44]:
```
# create user_item_matrix_df
user_item_matrix_df = pd.DataFrame(user_item_matrix)
# define the index and columns
user_item_matrix_df.columns = user_item_df.columns
user_item_matrix_df.index = user_item_df.index
```

In [45]:
```
# 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
    '''
    # groupby and unstack
    user_item_df = df.groupby(['user_id','article_id'])['title'].count().unstack()

    # where notnull, place a 1, else 0
    user_item_matrix = np.where(user_item_df.isnull(), 0, 1)

    # create user_item_matrix_df
    user_item_matrix_df = pd.DataFrame(user_item_matrix)
    # define the index and columns
    user_item_matrix_df.columns = user_item_df.columns
    user_item_matrix_df.index = user_item_df.index

    return user_item_matrix_df # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [46]:
```
user_item.sum(axis=1)[1]
```

Out[46]: 36

In [47]:
```
## 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 [48]:
```
user_item_df.head()
```

Out[48]:
```
article_id  0     2     4     8     9     12    14    15    16    18    ...   \
user_id                                                               ...
1           NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   ...
2           NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   ...
3           NaN   NaN   NaN   NaN   NaN   1.0   NaN   NaN   NaN   NaN   ...
4           NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   ...
5           NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   ...

article_id  1434  1435  1436  1437  1439  1440  1441  1442  1443  1444
user_id
1           NaN   NaN   1.0   NaN   1.0   NaN   NaN   NaN   NaN   NaN
2           NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
3           NaN   NaN   1.0   NaN   NaN   NaN   NaN   NaN   NaN   NaN
4           NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
5           NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

[5 rows x 714 columns]
```

In [53]:
```
# user_item.values
```

In [54]:
```
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_idx = np.where(user_item_df.index == user_id)[0][0]
    dot_prod = np.dot(user_item.values[user_idx], user_item.values.T)
```

```
        # sort by similarity, `[::-1]`
        # greatest to least similar, exclude first item, `[1:]`
        sorted_users_idx = np.argsort(dot_prod)[::-1][1:]

        # get list of user_ids
        most_similar_users = user_item_df.iloc[sorted_users_idx].index.tolist()

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

```
In [56]: # 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, 5041]
The 5 most similar users to user 3933 are: [3933, 23, 3782, 4459, 203]
The 3 most similar users to user 46 are: [46, 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 [57]: def to_int(X):
             """
             Convert input contents of list `X` to integers.
             """
             return pd.Series(X).astype(np.float).astype(np.int32).tolist()
```

```
In [58]: 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)
             '''
             # check dtype of article_ids
             # convert to integer if not integer
             if pd.Series(article_ids).dtype != np.int:
                 article_ids = to_int(article_ids)

             # get article indices
             df_unique = df.drop_duplicates(subset='article_id')
```

```
    article_names = df_unique[df_unique['article_id'].isin(article_ids)]['title'].tolis

    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
    '''
    # check dtype of input user_id
    # convert to integer if not integer
    if pd.Series(user_id).dtype != np.int:
        user_id = to_int(user_id)

    article_ids = user_item_df.loc[user_id][user_item_df.loc[user_id].notnull()].index.
    article_names = set(df[df['article_id'].isin(article_ids)]['title'].tolist())

    return article_ids, article_names # return the ids and names
```

In [59]:
```
user_id = 2
article_ids = user_item_df.loc[user_id][user_item_df.loc[user_id].notnull()].index.toli
article_names = df_content[df_content['article_id'].isin(article_ids)]['doc_full_name']

article_ids
# df_content[df_content['article_id'].isin(article_ids)]['doc_full_name']

set(df[df['article_id'].isin([1024, 1176, 1305, 1314, 1422, 1427])]['title'])
```

Out[59]:
```
{'build a python app on the streaming analytics service',
 'gosales transactions for naive bayes model',
 'healthcare python streaming application demo',
 'use r dataframes & ibm watson natural language understanding',
 'use xgboost, scikit-learn & ibm watson machine learning apis',
 'using deep learning to reconstruct high-resolution audio'}
```

In [60]:
```
def user_user_recs(user_id, m=10):
    '''
    INPUT:
```

```python
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

'''
# get list of similar users to given user_id, outputs user_ids
similar_users = find_similar_users(user_id)[:m]

# get list of articles seen by `user_id`
given_user_articles = get_user_articles(user_id)[0]

recs = np.array([])
for i in np.arange(m):

    # check shape of array,
    # if number of recs < `m`, then obtain more recs
    if recs.shape[0] < m:

        for sim_user in similar_users:
            # compare each similar user to given user_id

            # get article_ids for `sim_user`
            sim_user_articles = get_user_articles(sim_user)[0]

            # compare article_ids between the two users
            # return array of ids not present for `user_id`
            diff_ids_array = np.setdiff1d(sim_user_articles, given_user_articles, a

            # take first `m` items from array
            recs = diff_ids_array[:m]

# return your recommendations for this user_id
# return a list of article_ids
return recs
```

```
In [61]: # Check Results
         get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1

Out[61]: ['got zip code data? prep it for analytics.  ibm watson data lab  medium',
          'dsx: hybrid mode',
          'data science for real-time streaming analytics',
          'python machine learning: scikit-learn tutorial',
          'how smart catalogs can turn the big data flood into an ocean of opportunity',
          'learn tensorflow and deep learning together and now!',
          'using brunel in ipython/jupyter notebooks',
          'getting started with graphframes in apache spark',
          'this week in data science (april 4, 2017)',
          'apple, ibm add machine learning to partnership with watson-core ml coupling']

In [62]: get_user_articles(20)[0]

Out[62]: [232, 844, 1320]

In [63]: # NOTE:
         # Below assert statements appear to expect an output in form of a string of floats.
         def to_float_str(x):
             return pd.Series(x).astype(np.float).astype(str).tolist()

In [64]: # 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(to_float_str(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(to_float_str(get_user_articles(2)[0])) == set(['1024.0', '1176.0', '1305.0',
         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 [65]: df.head()
```

```
Out[65]:    article_id                                              title  user_id  \
        0         1430  using pixiedust for fast, flexible, and easier...        1
        1         1314       healthcare python streaming application demo        2
        2         1429         use deep learning for image classification        3
        3         1338          ml optimization using cognitive assistant        4
        4         1276          deploy your python model as a restful api        5

           article_counts
        0             336
        1             614
        2             937
        3             382
        4             347
```

```
In [66]: def get_top_sorted_users(user_id, df=df, user_item=user_item_matrix_df):
             '''
             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 provided
                             num_interactions - the number of articles viewed by the user - if a

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

             '''
             # find most similar users, outputs array of user ids
             # sorted from most to least similar
             similar_users_arr = find_similar_users(user_id)

             # similarity, dot-product between given user and all others
             similarity = np.dot(user_item.loc[user_id].values, user_item.loc[similar_users_arr]

             # number of interactions for each user, `neighbor_id`
             # within `user_item` dataframe, count non-zero terms
             user_mapping = user_item.apply(np.count_nonzero, axis=1)

             # define DataFrame
             neighbors_df = pd.DataFrame({

                 'neighbor_id': similar_users_arr,
```

```
            'similarity': similarity
        })
        # apply a mapping to each user `neighbor_id`,
        # interaction count is mapped to specifed user
        neighbors_df['num_interactions'] = neighbors_df['neighbor_id'].map(user_mapping)

        # sort by interactions
        return neighbors_df.sort_values(by=['similarity', 'num_interactions',], ascending=F
```

In [67]:
```
get_top_sorted_users(55).head()
```

Out[67]:

	neighbor_id	similarity	num_interactions
0	3417	28	29
2	3782	14	135
3	23	14	135
1	203	14	96
4	4459	14	96

In [68]:
```
get_top_sorted_users(10).head(10)
```

Out[68]:

	neighbor_id	similarity	num_interactions
0	3354	17	17
1	49	15	101
2	3697	15	100
3	3764	14	97
4	98	14	97
6	322	13	53
5	3622	13	52
7	23	12	135
8	3782	12	135
10	4785	11	62

In [69]:
```
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 rec
    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 list of similar users to given user_id, outputs user_ids
            similar_users = get_top_sorted_users(user_id)['neighbor_id'].values

            # get list of articles seen by `user_id`
            given_user_articles = get_user_articles(user_id)[0]

            # store article_id recs in array
            recs = []

            # loop over the chosen users
            for sim_user in similar_users:
                # compare each similar user to given user_id

                # get article_ids for `sim_user`
                sim_user_articles = get_user_articles(sim_user)[0]

                # compare article_ids between the two users
                # return array of ids not present for `user_id`
                diff_article_ids_arr = np.setdiff1d(sim_user_articles, given_user_articles, ass

                # loop over diff_articles
                for article in diff_article_ids_arr:
                    # if < `m` recommendations, append article_id to list
                    if len(recs) < m:
                        recs.append(article)


            # get names for article_ids
            rec_names = get_article_names(recs)

            return recs, rec_names
```

In [70]:
```
# get_article_names??
```

In [71]:
```
# Quick spot check - don't change this code - just use it to test your functions
rec_ids, rec_names = user_user_recs_part2(1, 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:
```

```
[2, 12, 14, 16, 26, 28, 29, 33, 50, 74]

The top 10 recommendations for user 20 are the following article names:
['got zip code data? prep it for analytics.  ibm watson data lab  medium', 'timeseries data anal
```

In [72]:
```python
# 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, 14, 29, 33, 43, 51, 109, 111, 130, 142]

The top 10 recommendations for user 20 are the following article names:
['got zip code data? prep it for analytics.  ibm watson data lab  medium', 'timeseries data anal
```

`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 [73]:
```python
# # Find the user that is most similar to user 1
get_top_sorted_users(1)[:10]

# Find the 10th most similar user to user 131
get_top_sorted_users(131).sort_values('similarity', ascending=False)[:10]
```

Out[73]:

	neighbor_id	similarity	num_interactions
0	3870	74	75
1	3782	39	135
2	23	38	135
3	4459	33	96
4	203	33	96
8	49	29	101
7	3697	29	100
5	98	29	97
6	3764	29	97
9	242	25	59

In [74]:
```python
### Tests with a dictionary of results

user1_most_sim = 3933 # Find the user that is most similar to user 1
user131_10th_sim = 242 # Find the 10th most similar user to user 131
```

In [75]:
```python
## 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.

Given a new user, we would have no information about about their preferences. Therefore, it would be best to begin with recommending the **most popular articles** and gather information about this user. Once we have more information about their interactions with articles, we can attempt to use **Collaborative Filtering** to match with the most similar users.

Additionally, they could fill out a small questionaire about specific topics that they might prefer. Then use this information to recommend articles based on **content** of the article and popularity.

`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 [76]: # df_content['article_id'].value_counts(ascending=False)
         df_content.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1051 entries, 0 to 1055
Data columns (total 5 columns):
doc_body           1037 non-null object
doc_description    1048 non-null object
doc_full_name      1051 non-null object
doc_status         1051 non-null object
article_id         1051 non-null int64
dtypes: int64(1), object(4)
memory usage: 49.3+ KB
```

```
In [78]: to_float_str(get_top_article_ids(10))
```

```
Out[78]: ['1429.0',
          '1330.0',
          '1431.0',
          '1427.0',
          '1364.0',
          '1314.0',
          '1293.0',
          '1170.0',
          '1162.0',
          '1304.0']
```

```
In [79]: new_user = '0.0'

         # What would your recommendations be for this new user '0.0'?  As a new user, they have
         # Provide a list of the top 10 article ids you would give to
         new_user_recs = to_float_str(get_top_article_ids(10)) # Your recommendations here
```

```
In [80]: 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 [81]: 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 [82]: # 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 [83]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')

In [84]: # quick look at the matrix
         user_item_matrix.head()

Out[84]: 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 [85]: user_item_matrix.shape
```

Out[85]: (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 [86]: df.shape
```

```
Out[86]: (45993, 4)
```

```
In [87]: # Perform SVD on the User-Item Matrix Here
         u, s, vt = np.linalg.svd(user_item_matrix) # use the built in to get the three matrices
```

```
In [88]: u.shape
```

```
Out[88]: (5149, 5149)
```

```
In [89]: s.shape
```

```
Out[89]: (714,)
```

```
In [90]: vt.shape
```

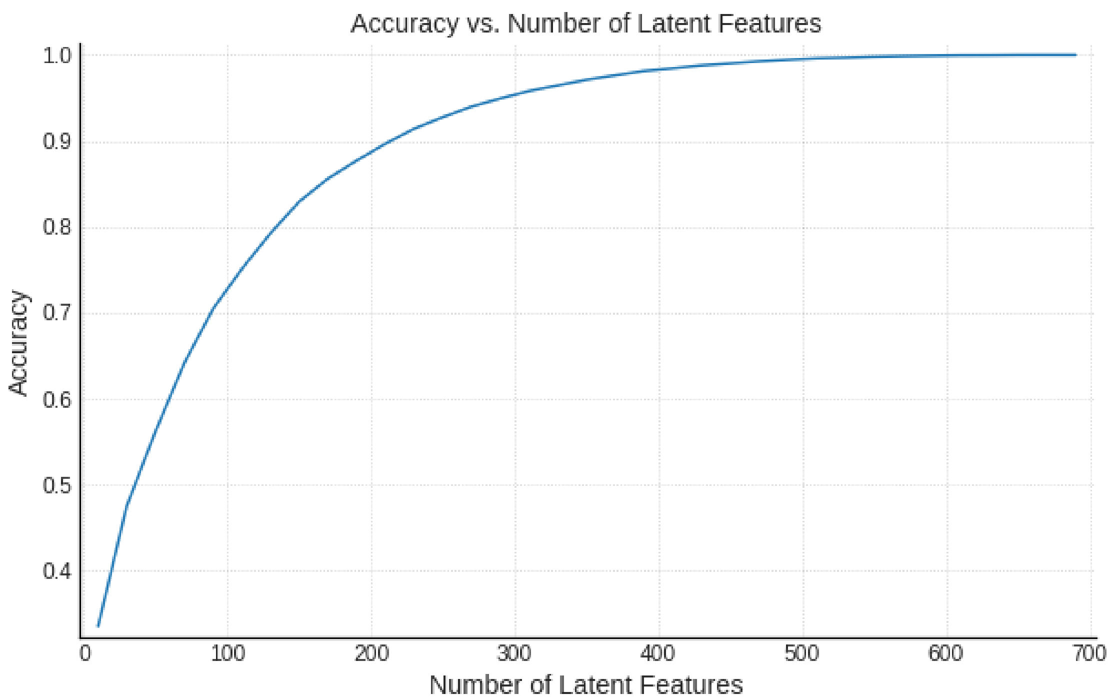
```
Out[90]: (714, 714)
```

We can use SVD here because there are no missing values. If the user-item matrix had any missing values then SVD would throw an error.

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 [91]: u, s, vt = np.linalg.svd(user_item_matrix)

         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 [92]:
```
df_train = df.head(40000)
df_test = df.tail(5993)

df_train.head()
```

Out[92]:
```
   article_id                                             title  user_id  \
0        1430  using pixiedust for fast, flexible, and easier...        1
```

```
1        1314      healthcare python streaming application demo        2
2        1429        use deep learning for image classification        3
3        1338         ml optimization using cognitive assistant        4
4        1276          deploy your python model as a restful api        5

   article_counts
0             336
1             614
2             937
3             382
4             347
```

In [93]:
```python
df.head(40000).shape
```

Out[93]:
```
(40000, 4)
```

In [94]:
```python
def make_user_item_df(df):

    # build user-item DataFrame
    user_item_df = df.groupby(['user_id','article_id'])['title'].count().unstack()

    # where notnull, place a 1, else 0
    user_item_matrix = np.where(user_item_df.isnull(), 0, 1)

    # create user_item_matrix_df
    user_item_matrix_df = pd.DataFrame(user_item_matrix)

    # define the index and columns
    user_item_matrix_df.columns = user_item_df.columns
    user_item_matrix_df.index = user_item_df.index

    return user_item_matrix_df

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

    '''
```

```python
    # train set
    user_item_train = make_user_item_df(df_train)
    # test set
    user_item_test = make_user_item_df(df_test)

    # extract user_ids
    test_idx = user_item_test.index.tolist()

    # extract article_ids
    test_arts = user_item_test.columns.tolist()

    return user_item_train, user_item_test, test_idx, test_arts
```

In [95]:
```python
df_train = df.head(40000)
df_test = df.tail(5993)

user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_user_item(
```

In [96]:
```python
# count of `test users` in training set
print(user_item_train.index.isin(test_idx).sum())

# count of `test articles` in training set
print(user_item_train.columns.isin(test_arts).sum())
```

```
20
574
```

In [97]:
```python
# 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,
    '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,
    'How many movies in the test set are we not able to make predictions for because of
}

t.sol_4_test(sol_4_dict)
# NOTE:
# `movies` should be replaced with `articles` in the `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 [98]:
```
def get_common_users_artcl(user_item_train, user_test_ids=test_idx, article_test_ids=te
    """
    Finds the indices of users and articles which appear in both,
    train and test, splits.

    Params:
    -------
        user_item_train: (pd.DataFrame)
            Training set of user-item matrix

        user_test_ids: (list, array)
            Users' ids which appear in the test set split

        article_test_ids: list, array
            Articles' ids which appear in the test set split

    Returns:
    --------
        (numpy array)
        Indices of articles and users which appear in both splits,
        train and test sets.

    """

    # users common to both training and test data sets
    # these are the users we can make predictions for
    users_both_sets = user_item_train.index[user_item_train.index.isin(user_test_ids)].

    # find user indices within U and V matrices to make predictions
    user_bool = pd.Series(user_item_train.index).isin(user_test_ids)
    user_idx = user_bool[user_bool].index.values

    # get article indices common to both training and test data set
    articles_bool = pd.Series(user_item_train.columns).isin(article_test_ids)
    articles_idx = articles_bool[articles_bool].index.values

    return user_idx, articles_idx
```

In [99]:
```
# df_train = df.head(40000)
# df_test = df.tail(5993)
```

```
idx_split = 40000

df_train = df.iloc[:idx_split, :]
df_test = df.iloc[idx_split:, :]

user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_user_item(
```

```
In [100]: user_idx, articles_idx = get_common_users_artcl(user_item_train, user_test_ids=test_id
```

```
In [101]: print('Train set:', df_train.shape[0])
          print('Test set:', df_test.shape[0])
          print('Common Users:', user_idx.shape[0])
          print('Common Articles:', articles_idx.shape[0])
```

```
Train set: 40000
Test set: 5993
Common Users: 20
Common Articles: 574
```

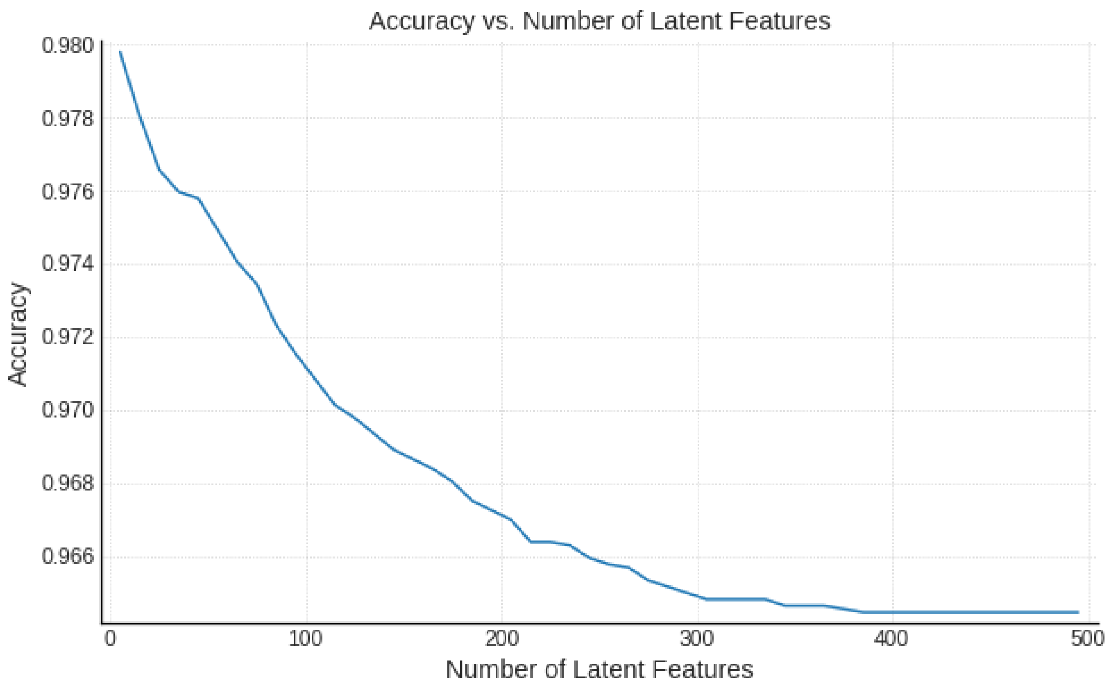
```
In [102]: # users common to both training and test data sets
          # these are the users we can make predictions for
          users_both_sets = user_item_train.index[user_item_train.index.isin(test_idx)].tolist()
          # users_both_sets
```

```
In [103]: # check shape of actual matrix
          user_item_actual = user_item_test.loc[users_both_sets]
          # print(user_item_pred.shape)
          print(user_item_actual.shape)
```

```
(20, 574)
```

```
In [104]: # perform SVD on the training set
          u_train, s_train, v_train = np.linalg.svd(user_item_train)

          num_latent_feats = np.arange(5, 500, 10)
          sum_errs = []

          for k in num_latent_feats:
              # restructure with k latent features
              s_new, u_new, vt_new = np.diag(s_train[:k]), u_train[user_idx, :k], v_train[:k, ar

              # take dot product
              user_item_pred = 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_actual, user_item_pred)
```

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




```
In [105]: def explained_variance(sigma, n_components):
              """
              Computes explained variance number of components
              """
              # explained variance
              total_var = np.sum(sigma**2)
              var_exp = np.sum([np.square(i) for i in sigma[:n_components]])
              perc_exp = (var_exp / total_var) * 100
              return perc_exp

In [106]: n_comp = 300
          v = explained_variance(s_train, n_comp)

          print(f'Number of components: {n_comp}')
          print(f'Variance explained: {np.round(v, 2)}%')
```

```
Number of components: 300
Variance explained: 92.08%
```

`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?

The overall accuracy appears to decline as the number of latent features, *k*, increases. However, the decline does level off and still provides a relatively good accuracy score, ~0.96. The decline in accuracy is small and can be said to be insignificant, although, more analysis would be needed. Additionally, this graph doesn't provide a way to select an optimal amount of latent features.

In order to determine if the above recommendations offer any improvement we could explore a number of options: 1. Perform online testing. Deploy the recommendations (from above) with metrics in mind and monitor the performance. One simple measure of performance would be the interactions between users and articles. If the interactions increase after deployment, then this could be an indication of improved recommendations. 2. A/B Testing. Using results from 1, we could analyze the data to see if the results support our hypothesis. Additionally, we could conduct an experiment to determine if the new recommendations offer any significant improvement. 3. Feedback. We could implement a simple rating system for articles. For example, ask users if they `like` or `dislike` an article after reading it. Or, ask users about the content that they would prefer to see, and then make new recommendations catering to their preferences.

```
In [108]: from subprocess import call
          call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])

Out[108]: 0
```