Recommendations_with_IBM

October 31, 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

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

         %matplotlib inline

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

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

```
Out[35]:    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 [36]:
```
# Show df_content to get an idea of the data
df_content.head()
```

Out[36]:
```
                                            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 [37]:
```
df.shape[0]
```

Out[37]:
```
45993
```

In [38]:
```
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 [39]: df.describe()
```

```
Out[39]:          article_id
         count  45993.000000
         mean     908.846477
         std      486.647866
         min        0.000000
         25%      460.000000
         50%     1151.000000
         75%     1336.000000
         max     1444.000000
```

```
In [40]: df_content.shape
```

```
Out[40]: (1056, 5)
```

```
In [41]: 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 [42]: df_content.describe()
```

```
Out[42]:         article_id
         count  1056.000000
         mean    523.913826
         std     303.480641
         min       0.000000
         25%     260.750000
         50%     523.500000
         75%     786.250000
         max    1050.000000
```

```
In [43]: df.groupby(['email']).count().median()[0]
```

Out[43]: 3.0

In [44]:
```
df.groupby(['email']).count().sort_values('title').tail(1)
```

Out[44]:
```
                                          article_id  title
email
2b6c0f514c2f2b04ad3c4583407dccd0810469ee         364    364
```

In [45]:
```
# Fill in the median and maximum number of user_article interactios below

median_val = df.groupby(['email']).count().median()[0] # 50% of individuals interact wi
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 [46]:
```
# Find and explore duplicate articles
```

In [47]:
```
# Remove any rows that have the same article_id - only keep the first
#df_content = df_content.drop_duplicates()
#df_content.head()
```

In [48]:
```
df_content = df_content.drop_duplicates()
df_content.head()
```

Out[48]:
```
                                            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 [49]:
```
df_content.shape
```

Out[49]: (1056, 5)

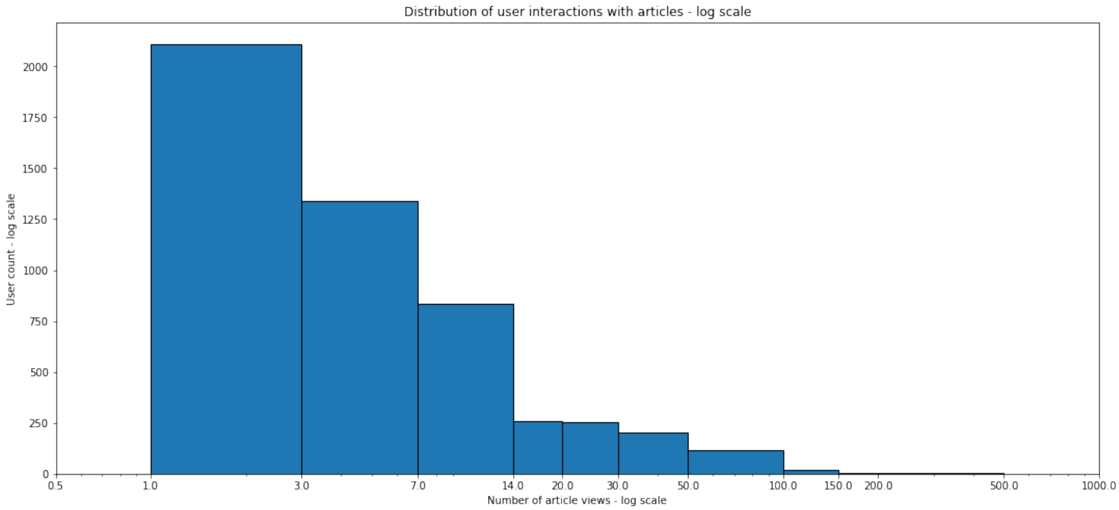
3. Use the cells below to find:
a. The number of unique articles that have an interaction with a user.
b. The number of unique articles in the dataset (whether they have any interactions or not). **c.** The number of unique users in the dataset. (excluding null values) **d.** The number of user-article interactions in the dataset.

```
In [50]: df.nunique()[0]

Out[50]: 714

In [51]: df_content.nunique()[2]

Out[51]: 1051

In [73]: unique_articles = df.nunique()[0] # The number of unique articles that have at least on
         total_articles = df_content.nunique()[2] # The number of unique articles on the IBM pla
         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 [53]: df.groupby(['article_id']).count().sort_values('title').tail()

Out[53]:             title  email
         article_id
         1364.0        627    627
         1427.0        643    643
         1431.0        671    671
         1330.0        927    927
         1429.0        937    937

In [72]: df.groupby(['article_id']).count().sort_values('title', ascending=False).head(1)

Out[72]:             title  email
         article_id
         1429.0        937    937

In [61]: df.groupby('article_id').count().max().title

Out[61]: 937

In [58]: df[df.article_id == 1330.0].shape

Out[58]: (927, 3)

In [75]: most_viewed_article_id = "1429.0" # The most viewed article in the dataset as a string
         max_views = df[['email','article_id']].groupby(['email']).count().describe().max() # Th
```

```
In [77]: # Histogram for distribution of user interaction with articles
         plt.figure(figsize=(18,8))
         histogram_bins = [0,1,3,7,14,20,30,50,100,150,200,500]
         histogram_ticks = np.array([0.5, 1,3,7,14,20,30,50,100,150,200,500,1000])
         plt.hist(df[['email','article_id']].groupby(['email']).count()['article_id'],bins=histo
         plt.yscale('linear')
         plt.xscale('log')
         plt.xticks(histogram_ticks,histogram_ticks.astype(str))
         plt.title('Distribution of user interactions with articles - log scale')
         plt.xlabel('Number of article views - log scale')
         plt.ylabel('User count - log scale')
         plt.show()
```




```
In [24]: ## 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[24]:

	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 [25]:
```
## 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 [26]:
```
df_title = df.groupby(['article_id', 'title']).count().reset_index().sort_values('user_
list(df_title['title'])
```

Out[26]:
```
['use deep learning for image classification',
 'insights from new york car accident reports',
 'visualize car data with brunel',
 'use xgboost, scikit-learn & ibm watson machine learning apis',
 'predicting churn with the spss random tree algorithm',
```

```
'healthcare python streaming application demo',
'finding optimal locations of new store using decision optimization',
'apache spark lab, part 1: basic concepts',
'analyze energy consumption in buildings',
'gosales transactions for logistic regression model',
'welcome to pixiedust',
'customer demographics and sales',
'total population by country',
'deep learning with tensorflow course by big data university',
'model bike sharing data with spss',
'the nurse assignment problem',
'classify tumors with machine learning',
'analyze accident reports on amazon emr spark',
'movie recommender system with spark machine learning',
'putting a human face on machine learning',
'gosales transactions for naive bayes model',
'ml optimization using cognitive assistant',
'learn basics about notebooks and apache spark',
'analyze precipitation data',
'apache spark lab, part 3: machine learning',
'jupyter notebook tutorial',
'deploy your python model as a restful api',
'visualize data with the matplotlib library',
'using pixiedust for fast, flexible, and easier data analysis and experimentation',
'access db2 warehouse on cloud and db2 with python',
'python machine learning: scikit-learn tutorial',
'maximize oil company profits',
'analyze open data sets with spark & pixiedust',
'uci ml repository: chronic kidney disease data set',
'introducing ibm watson studio ',
'analyze open data sets with pandas dataframes',
'working interactively with rstudio and notebooks in dsx',
'rapidly build machine learning flows with dsx',
'the pandas data analysis library',
'the machine learning database',
'learn tensorflow and deep learning together and now!',
'real-time sentiment analysis of twitter hashtags with spark (+ pixiedust)',
'access mysql with r',
'what caused the challenger disaster?',
'access mysql with python',
'pixieapp for outlier detection',
'breast cancer wisconsin (diagnostic) data set',
'apache spark lab, part 2: querying data',
'access ibm analytics for apache spark from rstudio',
'times world university ranking analysis',
'intents & examples for ibm watson conversation',
'data model with streaming analytics and python',
'tensorflow quick tips',
```

```
'deep learning with data science experience',
'fortune 100 companies',
'analyzing data by using the sparkling.data library features',
'employed population by occupation and age',
'sudoku',
'small steps to tensorflow',
'520    using notebooks with pixiedust for fast, flexi...\nName: title, dtype: object'
'programmatic evaluation using watson conversation',
'ibm watson facebook posts for 2015',
'the nurse assignment problem data',
'data tidying in data science experience',
'getting started with python',
'build a python app on the streaming analytics service',
'i am not a data scientist  ibm watson data lab',
'categorize urban density',
'use r dataframes & ibm watson natural language understanding',
'housing (2015): united states demographic measures',
'practical tutorial on random forest and parameter tuning in r',
'using deep learning with keras to predict customer churn',
'timeseries data analysis of iot events by using jupyter notebook',
'building your first machine learning system ',
'use sql with data in hadoop python',
'how to map usa rivers using ggplot2',
'using machine learning to predict value of homes on airbnb',
'sector correlations shiny app',
'machine learning exercises in python, part 1',
'an introduction to stock market data analysis with r (part 1)',
'income (2015): united states demographic measures',
'connect to db2 warehouse on cloud and db2 using scala',
'developing for the ibm streaming analytics service',
'using brunel in ipython/jupyter notebooks',
'transfer learning for flight delay prediction via variational autoencoders',
'flightpredict ii: the sequel   ibm watson data lab',
'use spark for scala to load data and run sql queries',
'a comparison of logistic regression and naive bayes ',
'uci: heart disease - cleveland',
'deep learning from scratch i: computational graphs',
'using github for project control in dsx',
'use spark for python to load data and run sql queries',
'super fast string matching in python',
'a dynamic duo  inside machine learning  medium',
'upload files to ibm data science experience using the command line',
'statistics for hackers',
'discover hidden facebook usage insights',
'working with ibm cloud object storage in python',
'spark 2.1 and job monitoring available in dsx',
'perform sentiment analysis with lstms, using tensorflow',
'modern machine learning algorithms',
```

```
'analyze traffic data from the city of san francisco',
'overlapping co-cluster recommendation algorithm (ocular)',
'the unit commitment problem',
'pixiedust 1.0 is here!  ibm watson data lab',
'use decision optimization to schedule league games',
'watson assistant workspace analysis with user logs',
'car performance data',
'1448    i ranked every intro to data science course on...\nName: title, dtype: object
'data science for real-time streaming analytics',
'introducing streams designer',
'this week in data science (may 30, 2017)',
'modeling energy usage in new york city',
'use apache systemml and spark for machine learning',
'how smart catalogs can turn the big data flood into an ocean of opportunity',
'7292    a dramatic tour through pythons data visualiz...\nName: title, dtype: object'
'watson machine learning for developers',
'uci: sms spam collection',
'use the machine learning library',
'analyzing streaming data from kafka topics',
'1357    what i learned implementing a classifier from ...\nName: title, dtype: object
'visualize the 1854 london cholera outbreak',
'flexdashboard: interactive dashboards for r',
'predict chronic kidney disease using spss modeler flows',
'workflow in r',
'develop a scala spark model on chicago building violations',
'probabilistic graphical models tutorial\u200a\u200apart 1  stats and bots',
'access postgresql with python',
'how to perform a logistic regression in r',
'united states demographic measures: population and age',
'machine learning and the science of choosing',
'brunel interactive visualizations in jupyter notebooks',
'use the cloudant-spark connector in python notebook',
"i'd rather predict basketball games than elections: elastic nba rankings",
'house building with worker skills',
'10 essential algorithms for machine learning engineers',
'upload data and create data frames in jupyter notebooks',
'top 10 machine learning use cases: part 1',
'markdown for jupyter notebooks cheatsheet',
'10 must attend data science, ml and ai conferences in 2018',
'leverage python, scikit, and text classification for behavioral profiling',
'how to choose a project to practice data science',
'got zip code data? prep it for analytics.  ibm watson data lab  medium',
'using machine learning to predict parking difficulty',
'analyze open data sets using pandas in a python notebook',
'graph-based machine learning',
'jupyter notebooks with scala, python, or r kernels',
'predicting flight cancellations using weather data, part 3',
'ibm data science experience white paper - sparkr transforming r into a tool for big d
```

```
'challenges in deep learning',
'data science bowl 2017',
'using bigdl in dsx for deep learning on spark',
'new shiny cheat sheet and video tutorial',
'time series prediction using recurrent neural networks (lstms)',
'70 amazing free data sources you should know',
'using rstudio in ibm data science experience',
'using dsx notebooks to analyze github data',
'automating web analytics through python',
'why you should master r (even if it might eventually become obsolete)',
'dsx: hybrid mode',
'adolescent fertility rate (births per 1,000 women ages 15-19), worldwide',
'calls by customers of a telco company',
'the greatest public datasets for ai  startup grind',
'common excel tasks demonstrated in\xa0pandas',
'how to scale your analytics using r',
'use spark r to load and analyze data',
'experience iot with coursera',
'simple graphing with ipython and\xa0pandas',
'best packages for data manipulation in r',
'the data science process',
'54174    detect potentially malfunctioning sensors in r...\nName: title, dtype: objec
'using deep learning to reconstruct high-resolution audio',
'city population by sex, city and city type',
'data structures related to machine learning algorithms',
'easy json loading and social sharing in dsx notebooks',
'use spark for r to load data and run sql queries',
'ensemble learning to improve machine learning results',
'process events from the watson iot platform in a streams python application',
'the 3 kinds of context: machine learning and the art of the frame',
'uci: car evaluation',
'pulling and displaying etf data',
'tidy up your jupyter notebooks with scripts',
'quick guide to build a recommendation engine in python',
'self-service data preparation with ibm data refinery',
'top 10 machine learning algorithms for beginners',
'15 page tutorial for r',
'sparklyr  r interface for apache spark',
'7 types of job profiles that makes you a data scientist',
'airbnb data for analytics: amsterdam calendar',
'understanding empirical bayes estimation (using baseball statistics)',
'introduction to market basket analysis in\xa0python',
'pixiedust gets its first community-driven feature in 1.0.4',
"a kaggler's guide to model stacking in practice",
'making data science a team sport',
'pixiedust: magic for your python notebook',
'neural language modeling from scratch (part 1)',
"a beginner's guide to variational methods",
```

```
'5 practical use cases of social network analytics: going beyond facebook and twitter'
'from scikit-learn model to cloud with wml client',
'higher-order logistic regression for large datasets',
'data science in the cloud',
'whats new in the streaming analytics service on bluemix',
'neurally embedded emojis',
'deep learning trends and an example',
'working with ibm cloud object storage in r',
'apple, ibm add machine learning to partnership with watson-core ml coupling',
'how to solve 90% of nlp problems',
'uci: iris',
'ibm cloud sql query',
'excel files: loading from object storage  python',
'this week in data science (april 18, 2017)',
'access postgresql with r',
'an introduction to scientific python (and a bit of the maths behind it)  numpy',
'annual precipitation by country 1990-2009',
'this week in data science (may 16, 2017)',
'this week in data science (may 2, 2017)',
'airbnb data for analytics: amsterdam listings',
'this week in data science (april 4, 2017)',
'use ibm data science experience to detect time series anomalies',
'variational auto-encoder for "frey faces" using keras',
'predicting gentrification using longitudinal census data',
'breast cancer detection with xgboost, wml and scikit',
'getting started with apache mahout',
'airbnb data for analytics: washington d.c. reviews',
'uci: red wine quality',
'trust in data science',
'an attempt to understand boosting algorithm(s)',
"feature importance and why it's important",
'declarative machine learning',
'this week in data science (february 14, 2017)',
'mapping points with folium',
'uci: adult - predict income',
'hurricane how-to',
'using machine learning to predict baseball injuries',
'working with sqlite databases using python and pandas',
'data visualization with r: scrum metrics',
'some random weekend reading',
'a moving average trading strategy',
'uci: forest fires',
'this week in data science (april 11, 2017)',
'spark 1.4 for rstudio',
'python for loops explained (python for data science basics #5)',
'introduction to neural networks, advantages and applications',
'this week in data science (may 23, 2017)',
'brunel 2.0 preview',
```

```
'health insurance (2015): united states demographic measures',
'recommendation system algorithms  stats and bots',
'56594    lifelong (machine) learning: how automation ca...\nName: title, dtype: objec
'spark-based machine learning tools for capturing word meanings',
'tidy data in python',
'502    forgetting the past to learn the future: long ...\nName: title, dtype: object'
'twelve\xa0ways to color a map of africa using brunel',
'use ibm data science experience to read and write data stored on amazon s3',
'ml algorithm != learning machine',
"december '16 rstudio tips and tricks",
'how to use version control (github) in rstudio within dsx?',
'collecting data science cheat sheets',
'what is text analytics?',
'aspiring data scientists! start to learn statistics with these 6 books!',
'simple linear regression? do it the bayesian way',
'united states demographic measures: zip code tabulation areas (zctas)',
'collect your own fitbit data with python',
'model a golomb ruler',
'8170    data science expert interview: dez blanchfield...\nName: title, dtype: object
'uci: white wine quality',
'the t-distribution: a key statistical concept discovered by a beer brewery',
'a tensorflow regression model to predict house values',
'using apply, sapply, lapply in r',
'neural networks for beginners: popular types and applications',
'airbnb data for analytics: washington d.c. listings',
'wages',
'deep forest: towards an alternative to deep neural networks',
'occupation (2015): united states demographic measures',
'improving real-time object detection with yolo',
'awesome deep learning papers',
'10 data science podcasts you need to be listening to right now',
'what is smote in an imbalanced class setting (e.g. fraud detection)?',
'this week in data science (march 7, 2017)',
'from spark ml model to online scoring with scala',
'this week in data science (january 24, 2017)',
'improving the roi of big data and analytics through leveraging new sources of data',
'generative adversarial networks (gans)',
'ingest data from message hub in a streams flow',
'fashion-mnist',
'get social with your notebooks in dsx',
'overfitting in machine learning: what it is and how to prevent it',
'9 mistakes to avoid when starting your career in data science',
'cifar-100 - python version',
'working with db2 warehouse on cloud in data science experience',
'd3heatmap: interactive heat maps',
'this week in data science (january 31, 2017)',
'analyze facebook data using ibm watson and watson studio',
'10 powerful features on watson data platform, no coding necessary',
```

```
'this week in data science (february 7, 2017)',
'notebooks: a power tool for data scientists',
'life expectancy at birth by country in total years',
'better together: spss and data science experience',
'this week in data science (february 21, 2017)',
'apache spark as the new engine of genomics',
'enjoy python 3.5 in jupyter notebooks',
'leaflet: interactive web maps with r',
'0 to life-changing app: new apache systemml api on spark shell',
'pearson correlation aggregation on sparksql',
'generalization in deep learning',
'probabilistic graphical models tutorial\u200a\u200apart 2  stats and bots',
'accelerate your workflow with dsx',
'interactive web apps with shiny cheat sheet',
'getting started with graphframes in apache spark',
'web picks (week of 23 january 2017)',
'this week in data science (march 28, 2017)',
'optimizing a marketing campaign: moving from predictions to actions',
'how to use db2 warehouse on cloud in data science experience notebooks',
'country statistics: unemployment rate',
'contraceptive prevalence (% women 15-49) by country',
'births attended by skilled health staff (% of total) by country',
'how to write the first for loop in r',
'word2vec in data products',
'10 tips on using jupyter notebook',
'ibm data catalog is now generally available',
'web picks (december 2017)',
'from python nested lists to multidimensional numpy arrays',
'the power of machine learning in spark',
'a visual explanation of the back propagation algorithm for neural networks',
'unstructured and structured data versus repetitive and non-repetitive',
'country statistics: health expenditures',
'environment statistics database - water',
'rstudio ide  cheat sheet',
'dry bulb temperature, by country, station and year',
'analyze starcraft ii replays with jupyter notebooks',
'interconnect with us',
'a classification problem',
'effectively using\xa0matplotlib',
'how to ease the strain as your data volumes rise',
'intentional homicide, number and rate per 100,000 population, by country',
'annual % population growth by country',
'this week in data science (february 28, 2017)',
'imitation learning in tensorflow (hopper from openai gym)',
'when machine learning matters û erik bernhardsson',
'shiny 0.12: interactive plots with ggplot2',
'dimensionality reduction algorithms',
'data visualization playbook: revisiting the basics',
```

```
'python if statements explained (python for data science basics #4)',
'cleaning the swamp: turn your data lake into a source of crystal-clear insight',
'tidyverse practice: mapping large european cities',
'0 to life-changing app: scala first steps and an interview with jakob odersky',
'apache spark 2.0: extend structured streaming for spark ml',
'airbnb data for analytics: sydney calendar',
'social media insights with watson developer cloud & watson studio',
'airbnb data for analytics: vienna reviews',
'time series anomaly detection algorithms  stats and bots',
'this week in data science (january 10, 2017)',
'shiny 0.13.0',
'uci: wine recognition',
'fighting gerrymandering: using data science to draw fairer congressional districts',
'why even a moths brain is smarter than an ai',
'making sense of the bias / variance trade-off in (deep) reinforcement learning',
'10 data science, machine learning and ai podcasts you must listen to',
'education (2015): united states demographic measures',
'statistical bias types explained (with examples)',
'visualising data the node.js way',
'transform anything into a vector',
'top 20 r machine learning and data science packages',
'customers of a telco including services used',
'total employment, by economic activity (thousands)',
'apache spark 2.0: machine learning. under the hood and over the rainbow.',
'the random forest algorithm ',
'analyze data, build a dashboard with spark and pixiedust',
'mobile-cellular telephone subscriptions per 100 inhabitants, worldwide',
'web picks (week of 4 september 2017)',
'country statistics: gdp - per capita (ppp)',
'airbnb data for analytics: venice calendar',
'the million dollar question: where is my data?',
'country statistics: commercial bank prime lending rate',
'brunel in jupyter',
'make machine learning a reality for your enterprise',
'this week in data science (april 25, 2017)',
'breaking the 80/20 rule: how data catalogs transform data scientists productivity',
'worldwide electricity demand and production 1990-2012',
'data science platforms are on the rise and ibm is leading the way',
'agriculture, value added (% of gdp) by country',
'airbnb data for analytics: amsterdam reviews',
'68879    dont throw more data at the problem! heres h...\nName: title, dtype: object'
'uci: poker hand - testing data set',
'what is hadoop?',
'discover, catalog and govern data with ibm data catalog',
'predict loan applicant behavior with tensorflow neural networking',
'announcing dsx environments in beta!',
'artificial intelligence, ethically speaking  inside machine learning  medium',
'data wrangling with dplyr and tidyr cheat sheet',
```

```
'cifar-10 - python version',
'htmlwidgets: javascript data visualization for r',
'visual information theory ',
'three reasons machine learning models go out of sync',
'51822    using apache spark as a parallel processing fr...\nName: title, dtype: objec
'data visualization: the importance of excluding unnecessary details',
'random forest interpretation  conditional feature contributions',
'3 scenarios for machine learning on multicloud',
'can a.i. be taught to explain itself?',
'developing ibm streams applications with the python api (version 1.6)',
'8 ways to turn data into value with apache spark machine learning',
'building custom machine learning algorithms with apache systemml',
'web picks - dataminingapps',
'configuring the apache spark sql context',
'part-time employment rate, worldwide, by country and year',
'statistical bias types explained',
'gradient boosting explained',
'how ibm builds an effective data science team',
'deep learning achievements over the past year ',
'get started with streams designer by following this roadmap',
'leverage scikit-learn models with core ml',
'how the circle line rogue train was caught with data',
'data visualization playbook: the right level of detail',
'score a predictive model built with ibm spss modeler, wml & dsx',
'electric power consumption (kwh per capita) by country',
'machine learning for the enterprise.',
'this week in data science (july 26, 2016)',
'data visualization playbook: telling the data story',
'how can data scientists collaborate to build better business',
'manage object storage in dsx',
'blogging with brunel',
'a guide to receptive field arithmetic for convolutional neural networks',
'join and enrich data from multiple sources',
'interactive time series with dygraphs',
'hyperparameter optimization: sven hafeneger',
'mycheatsheets.com',
"2875    hugo larochelle's neural network & deep learni...\nName: title, dtype: object
'share the (pixiedust) magic  ibm watson data lab  medium',
'drowning in data sources: how data cataloging could fix your findability problems',
'how open api economy accelerates the growth of big data and analytics',
'optimization for deep learning highlights in 2017',
'poverty (2015): united states demographic measures',
'readr 1.0.0',
'uci: poker hand - training data set',
'country population by gender 1985-2005',
'recommender systems: approaches & algorithms',
'airbnb data for analytics: venice reviews',
'calculate moving averages on real time data with streams designer',
```

```
'finding the user in data science',
'learning statistics on youtube',
'bayesian regularization for #neuralnetworks  autonomous agents\u200a\u200a#ai',
'bayesian nonparametric models  stats and bots',
'using shell scripts to control data flows created in watson applications',
'what is machine learning?',
'migrating to python 3 with pleasure',
'best practices for custom models in watson visual recognition',
'find airbnb deals in portland with machine learning using r',
'creating the data science experience',
'open sourcing 223gb of driving data  udacity inc',
'airbnb data for analytics: vienna listings',
'airbnb data for analytics: vancouver reviews',
'detect malfunctioning iot sensors with streaming analytics',
'work with data connections in dsx',
'data visualization with ggplot2 cheat sheet',
'essentials of machine learning algorithms (with python and r codes)',
"let's have some fun with nfl data",
'making data cleaning simple with the sparkling.data library',
'adoption of machine learning to software failure prediction',
'analyze ny restaurant data using spark in dsx',
'percentage of internet users by country',
'deep learning, structure and innate priors',
'data science of variable selection',
'20405    how to tame the valley  hessian-free hacks fo...\nName: title, dtype: object
'r for data science',
'spark sql - rapid performance evolution',
'external debt stocks, total (dod, current us$) by country',
'shiny: a data scientists best friend',
'population below national poverty line, total, percentage',
'time series analysis using max/min and neuroscience',
'web picks by dataminingapps',
'top analytics tools in 2016',
'country statistics: life expectancy at birth',
'the difference between ai, machine learning, and deep learning?',
'build a logistic regression model with wml & dsx',
"for ai to get creative, it must learn the rules--then how to break 'em",
'airbnb data for analytics: toronto reviews',
'predicting the 2016 us presidential election',
'are your predictive models like broken clocks?',
'country statistics: telephones - mobile cellular',
'airbnb data for analytics: trentino listings',
'web picks (week of 2 october 2017)',
'clustering: a guide for the perplexed',
'working with notebooks in dsx',
'united states demographic measures: income',
'airbnb data for analytics: toronto calendar',
'apache spark sql analyzer resolves order-by column',
```

```
'pseudo-labeling a simple semi-supervised learning method',
'what is spark?',
'apache spark @scale: a 60 tb+ production use case',
'advancements in the spark community',
'data science experience documentation',
'roads paved as % of total roads by country',
'what is systemml? why is it relevant to you?',
'airbnb data for analytics: vienna calendar',
'data science experience demo: modeling energy usage in nyc',
'read and write data to and from amazon s3 buckets in rstudio',
'ratio (% of population) at national poverty line by country',
'country statistics - europe - population and society',
'talent vs luck: the role of randomness in success and failure',
'watson speech-to-text services  tl;dr need not apply',
'world marriage data',
'3992    using apache spark to predict attack vectors a...\nName: title, dtype: object
'build a predictive analytic model',
'apache spark 2.0: migrating applications',
'this week in data science (november 01, 2016)',
'forest area by country in sq km',
'ibm watson machine learning: get started',
'xml2 1.0.0',
'backpropagation  how neural networks learn complex behaviors',
'analyze db2 warehouse on cloud data in rstudio in dsx',
'empirical bayes for multiple sample sizes',
'the two phases of gradient descent in deep learning',
'a guide to convolution arithmetic for deep learning',
'an interview with pythonista katharine jarmul',
'intelligent applications - apache spark',
'recent trends in recommender systems',
'create a project for watson machine learning in dsx',
'airbnb data for analytics: antwerp reviews',
'use data assets in a project using ibm data catalog',
'airbnb data for analytics: barcelona reviews',
'brunel: imitation is a sincere form of flattery',
'airbnb data for analytics: portland listings',
'data science expert interview: holden karau',
'airbnb data for analytics: berlin reviews',
'airbnb data for analytics: toronto listings',
'apache systemml',
'airbnb data for analytics: portland reviews',
'airbnb data for analytics: madrid listings',
'dplyr 0.5.0',
'r markdown reference guide',
'shaping data with ibm data refinery',
'government consumption expenditure',
'country statistics: gross national saving',
'country statistics: airports',
```

```
'airbnb data for analytics: paris calendar',
'you could be looking at it all wrong',
'the art of side effects: curing apache spark streamings amnesia (part 1/2)',
'airbnb data for analytics: new york city calendar',
'interest rates',
'airbnb data for analytics: boston listings',
'airbnb data for analytics: trentino reviews',
'airbnb data for analytics: vancouver listings',
'country statistics: telephones - fixed lines',
'airbnb data for analytics: antwerp calendar',
'world tourism data by the world tourism organization',
'airbnb data for analytics: venice listings',
'reducing overplotting in scatterplots',
'laplace noising versus simulated out of sample methods (cross frames)',
'whats new in data refinery?',
'improving quality of life with spark-empowered machine learning',
'country statistics: roadways',
'foundational methodology for data science',
'fertility rate by country in total births per woman',
'seti data, publicly available, from ibm',
'jupyter (ipython) notebooks features',
'uci: abalone',
'persistent changes to spark config in dsx',
'how to get a job in deep learning',
'mobile cellular subscriptions per 100 people by country',
'airbnb data for analytics: antwerp listings',
'country statistics: stock of domestic credit',
'country statistics: stock of broad money',
'airbnb data for analytics: austin listings',
'a fast on-disk format for data frames for r and python, powered by apache arrow',
'airbnb data for analytics: sydney reviews',
'a survey of books about apache spark',
'this week in data science',
'airbnb data for analytics: sydney listings',
'unmet need for family planning, spacing, percentage, worldwide, by country',
'co2 emissions (metric tons per capita) by country',
'this week in data science (august 02, 2016)',
'airbnb data for analytics: portland calendar',
'consumer prices',
'load data into rstudio for analysis in dsx',
'household consumption expenditure',
'households by type of household, age and sex of head of household',
'tidyr 0.6.0',
'do i need to learn r?',
'dt: an r interface to the datatables library',
'run shiny applications in rstudio in dsx',
'publish notebooks to github in dsx',
'airbnb data for analytics: athens calendar',
```

```
'worldwide fuel oil consumption by household (in 1000 metric tons)',
'enhanced color mapping',
'country statistics: reserves of foreign exchange and gold',
'country statistics: infant mortality rate',
'airbnb data for analytics: barcelona listings',
'machine learning for everyone',
'airbnb data for analytics: chicago listings',
'airbnb data for analytics: boston reviews',
'country statistics: crude oil - exports',
'country populations 15 years of age and over, by educational attainment, age and sex'
'which one to choose for your problem',
'high-tech exports as % of manufactured exports by country',
'airbnb data for analytics: new york city reviews',
'back to basics  jupyter notebooks',
'refugees',
'united states demographic measures: education',
'build a naive-bayes model with wml & dsx',
'machine learning for the enterprise',
'apache spark 2.0: impressive improvements to spark sql',
'greenhouse gas emissions worldwide',
'learn about data science in world of watson',
'airbnb data for analytics: vancouver calendar',
'use iot data in streams designer for billing and alerts',
'dont overlook simpler techniques and algorithms',
'cache table in apache spark sql',
'why relational databases and r?',
'a day in the life of a data engineer',
'gross national income per capita, atlas method (current us$) by country',
'apache spark: upgrade and speed-up your analytics',
'airbnb data for analytics: austin reviews',
'country statistics: railways',
'country statistics: maternal mortality rate',
'from local spark mllib model to cloud with watson machine learning',
'missing data conundrum: exploration and imputation techniques',
'foreign direct investment, net inflows (bop, current us$) by country',
'airbnb data for analytics: chicago calendar',
'10 pieces of advice to beginner data scientists',
'airbnb data for analytics: athens reviews',
'worldwide county and region - national accounts - gross national income 1948-2010',
'natural gas production, 1995 - 2012, worldwide',
'beyond parallelize and collect',
'environment statistics database - waste',
'airbnb data for analytics: boston calendar',
'airbnb data for analytics: antwerp listings test',
'airbnb data for analytics: nashville calendar',
'run dsx notebooks on amazon emr',
'country statistics: distribution of family income - gini index',
'airbnb data for analytics: montreal listings',
```

```
'airbnb data for analytics: san diego reviews',
'use pmml to predict iris species',
'66855    migration from ibm bluemix data connect api (a...\nName: title, dtype: objec
'this week in data science (january 17, 2017)',
'airbnb data for analytics: santa cruz county reviews',
'using the maker palette in the ibm data science experience',
'airbnb data for analytics: athens listings',
'refugees, worldwide, 2003 - 2013',
'roads, paved (% of total roads), worldwide, 1990-2011',
'the new builders podcast ep 3: collaboration',
'style transfer experiments with watson machine learning',
'airbnb data for analytics: mallorca reviews',
'airbnb data for analytics: washington d.c. calendar',
'airbnb data for analytics: austin calendar',
'airbnb data for analytics: london listings',
'airbnb data for analytics: paris listings',
'ggplot2 2.2.0 coming soon!',
'airbnb data for analytics: paris reviews',
'airbnb data for analytics: dublin reviews',
'airbnb data for analytics: berlin calendar',
'airbnb data for analytics: chicago reviews',
'airbnb data for analytics: trentino calendar',
'this week in data science (october 18, 2016)',
'airbnb data for analytics: seattle reviews',
'airbnb data for analytics: san francisco listings',
'continuous learning on watson',
'airbnb data for analytics: brussels reviews',
'airbnb data for analytics: seattle calendar',
'airbnb data for analytics: new orleans listings',
'big data is better data',
'load db2 warehouse on cloud data with apache spark in dsx',
'country statistics: industrial production growth rate',
'improved water source by country: % population with access',
'building a business that combines human experts and data science',
'country statistics: electricity - from fossil fuels',
'governance overview for ibm data catalog',
'one year as a data scientist at stack overflow',
'country statistics: imports',
"h2o with ibm's data science experience (dsx)",
'geographic coordinates of world locations',
'country statistics: internet users',
'country statistics: current account balance',
'package development with devtools  cheat sheet',
'country statistics: merchant marine',
'energy use (kg of oil equivalent per capita) by country',
'load and analyze public data sets in dsx',
'country statistics: population',
'country statistics: refined petroleum products - production',
```

```
 'working with on-premises databases  step by step',
 'a glimpse inside the mind of a data scientist',
 'the new builders ep. 13: all the data thats fit to analyze',
 'country statistics: electricity - consumption',
 'consumption of ozone-depleting cfcs in odp metric tons',
 'country statistics: central bank discount rate',
 'this week in data science (december 27, 2016)',
 'let data dictate the visualization',
 'a new version of dt (0.2) on cran',
 'military expenditure as % of gdp by country',
 'annual % inflation by country',
 'country statistics: budget surplus or deficit',
 'marital status of men and women',
 'create a project in dsx',
 'country statistics: children under the age of 5 years underweight',
 'labor',
 'primary school completion rate % of relevant age group by country',
 'international liquidity',
 'stacking multiple custom models in watson visual recognition',
 'nips 2016  day 2 highlights',
 'country statistics: market value of publicly traded shares',
 'airbnb data for analytics: barcelona calendar',
 'this week in data science (november 22, 2016)',
 'country statistics: natural gas - consumption',
 'ibm data catalog overview',
 'country statistics: crude oil - proved reserves',
 'webinar: april 11 - thinking inside the box: you can do that inside a data frame?!',
 'build deep learning architectures with neural network modeler',
 'country statistics: crude oil - imports',
 'airbnb data for analytics: london reviews',
 'create a connection and add it to a project using ibm data refinery',
 'measles immunization % children 12-23 months by country',
 'airbnb data for analytics: new orleans reviews',
 'airbnb data for analytics: oakland reviews',
 'the data processing inequality',
 'airbnb data for analytics: san diego listings',
 'country surface area (sq. km)']
```

```
In [27]: df_ids = df.groupby(['article_id', 'title']).count().reset_index().sort_values('user_id
         list(df_ids['article_id'])
```

```
Out[27]: [1429.0,
          1330.0,
          1431.0,
          1427.0,
          1364.0,
          1314.0,
          1293.0,
```

```
1170.0,
1162.0,
1304.0,
1436.0,
1271.0,
1398.0,
43.0,
1351.0,
1393.0,
1185.0,
1160.0,
1354.0,
1368.0,
1305.0,
1338.0,
1336.0,
1165.0,
1172.0,
151.0,
1276.0,
1432.0,
1430.0,
1052.0,
124.0,
1343.0,
1163.0,
1400.0,
390.0,
1164.0,
20.0,
732.0,
1017.0,
260.0,
164.0,
681.0,
1054.0,
1437.0,
1053.0,
1360.0,
1174.0,
1171.0,
600.0,
1396.0,
1332.0,
1274.0,
109.0,
237.0,
1296.0,
```

```
1166.0,
1282.0,
1391.0,
1386.0,
108.0,
1367.0,
1324.0,
1394.0,
1025.0,
542.0,
1176.0,
241.0,
1183.0,
1422.0,
1320.0,
116.0,
482.0,
12.0,
250.0,
1423.0,
98.0,
415.0,
268.0,
812.0,
162.0,
1328.0,
1186.0,
730.0,
33.0,
57.0,
969.0,
1426.0,
1047.0,
1405.0,
939.0,
409.0,
1424.0,
981.0,
120.0,
593.0,
125.0,
943.0,
910.0,
193.0,
641.0,
51.0,
865.0,
1357.0,
```

```
1395.0,
729.0,
1428.0,
1435.0,
1181.0,
1014.0,
194.0,
53.0,
510.0,
213.0,
1420.0,
221.0,
795.0,
122.0,
1410.0,
809.0,
651.0,
1048.0,
1433.0,
684.0,
673.0,
933.0,
1278.0,
880.0,
1055.0,
645.0,
1416.0,
833.0,
957.0,
844.0,
462.0,
1317.0,
723.0,
369.0,
34.0,
477.0,
173.0,
316.0,
101.0,
14.0,
26.0,
202.0,
50.0,
1050.0,
153.0,
349.0,
336.0,
8.0,
```

```
223.0,
525.0,
310.0,
669.0,
821.0,
958.0,
658.0,
1016.0,
362.0,
1057.0,
1180.0,
18.0,
189.0,
103.0,
486.0,
29.0,
131.0,
138.0,
975.0,
1277.0,
1024.0,
1184.0,
379.0,
682.0,
1425.0,
40.0,
1366.0,
74.0,
1403.0,
39.0,
111.0,
788.0,
232.0,
495.0,
692.0,
353.0,
1018.0,
1059.0,
959.0,
329.0,
617.0,
32.0,
766.0,
110.0,
898.0,
205.0,
996.0,
1298.0,
```

```
16.0,
508.0,
470.0,
315.0,
278.0,
1439.0,
254.0,
152.0,
1406.0,
1321.0,
695.0,
2.0,
1056.0,
68.0,
1169.0,
78.0,
76.0,
1060.0,
559.0,
464.0,
761.0,
64.0,
1175.0,
647.0,
1158.0,
1409.0,
547.0,
903.0,
130.0,
444.0,
210.0,
1044.0,
1402.0,
749.0,
911.0,
1028.0,
962.0,
314.0,
485.0,
1404.0,
448.0,
240.0,
494.0,
465.0,
491.0,
422.0,
1313.0,
607.0,
```

```
253.0,
744.0,
764.0,
887.0,
283.0,
1042.0,
967.0,
263.0,
930.0,
132.0,
382.0,
843.0,
609.0,
1418.0,
350.0,
1350.0,
251.0,
1411.0,
566.0,
1051.0,
224.0,
142.0,
1157.0,
1434.0,
28.0,
1356.0,
236.0,
295.0,
878.0,
825.0,
725.0,
1299.0,
440.0,
184.0,
813.0,
1329.0,
1290.0,
665.0,
871.0,
92.0,
1178.0,
634.0,
158.0,
288.0,
1159.0,
793.0,
136.0,
244.0,
```

```
1337.0,
693.0,
864.0,
977.0,
855.0,
955.0,
411.0,
846.0,
337.0,
311.0,
302.0,
891.0,
239.0,
230.0,
191.0,
359.0,
146.0,
1263.0,
1189.0,
1173.0,
868.0,
705.0,
528.0,
427.0,
606.0,
727.0,
721.0,
225.0,
936.0,
1225.0,
1286.0,
563.0,
1279.0,
858.0,
65.0,
381.0,
782.0,
102.0,
1331.0,
1168.0,
113.0,
667.0,
107.0,
951.0,
297.0,
965.0,
768.0,
861.0,
```

```
585.0,
143.0,
15.0,
1138.0,
1387.0,
1155.0,
928.0,
990.0,
968.0,
1412.0,
346.0,
952.0,
735.0,
784.0,
1280.0,
273.0,
367.0,
81.0,
89.0,
1273.0,
1397.0,
284.0,
905.0,
1161.0,
1349.0,
252.0,
1219.0,
1150.0,
583.0,
1198.0,
299.0,
455.0,
348.0,
892.0,
1443.0,
524.0,
1058.0,
1061.0,
145.0,
1407.0,
468.0,
659.0,
1363.0,
347.0,
195.0,
215.0,
1177.0,
622.0,
```

```
722.0,
616.0,
398.0,
62.0,
599.0,
234.0,
618.0,
266.0,
54.0,
112.0,
460.0,
993.0,
1358.0,
876.0,
475.0,
352.0,
656.0,
339.0,
80.0,
680.0,
36.0,
686.0,
1281.0,
1035.0,
87.0,
720.0,
569.0,
323.0,
882.0,
291.0,
298.0,
785.0,
632.0,
324.0,
1008.0,
522.0,
479.0,
30.0,
480.0,
1362.0,
60.0,
1408.0,
1190.0,
1000.0,
1152.0,
670.0,
115.0,
270.0,
```

```
181.0,
233.0,
319.0,
313.0,
426.0,
853.0,
1292.0,
25.0,
258.0,
1154.0,
1149.0,
0.0,
277.0,
544.0,
1006.0,
557.0,
935.0,
412.0,
4.0,
1359.0,
356.0,
157.0,
59.0,
986.0,
678.0,
1289.0,
668.0,
1361.0,
862.0,
492.0,
176.0,
1232.0,
500.0,
82.0,
715.0,
1143.0,
399.0,
759.0,
1261.0,
1145.0,
881.0,
366.0,
373.0,
1415.0,
1141.0,
949.0,
932.0,
463.0,
```

```
77.0,
58.0,
48.0,
1377.0,
948.0,
1153.0,
428.0,
400.0,
1369.0,
1191.0,
926.0,
919.0,
1440.0,
9.0,
751.0,
117.0,
134.0,
1295.0,
437.0,
256.0,
303.0,
626.0,
781.0,
532.0,
896.0,
383.0,
473.0,
973.0,
655.0,
1065.0,
100.0,
1074.0,
330.0,
1124.0,
183.0,
1077.0,
1142.0,
375.0,
1125.0,
1097.0,
534.0,
857.0,
517.0,
1306.0,
1221.0,
1192.0,
1120.0,
567.0,
```

```
515.0,
1114.0,
1333.0,
1079.0,
1146.0,
1148.0,
1260.0,
1062.0,
1441.0,
1151.0,
1038.0,
1030.0,
446.0,
96.0,
1253.0,
188.0,
1291.0,
760.0,
906.0,
1401.0,
404.0,
1004.0,
1348.0,
1063.0,
1257.0,
1254.0,
1070.0,
941.0,
1140.0,
714.0,
679.0,
1139.0,
1419.0,
1179.0,
502.0,
1123.0,
1187.0,
763.0,
1318.0,
1319.0,
588.0,
351.0,
564.0,
355.0,
474.0,
1066.0,
1444.0,
1015.0,
```

```
1252.0,
1228.0,
1073.0,
805.0,
1085.0,
1080.0,
1199.0,
1267.0,
553.0,
1315.0,
1116.0,
610.0,
1371.0,
1414.0,
555.0,
800.0,
389.0,
1307.0,
521.0,
1147.0,
1043.0,
884.0,
420.0,
429.0,
985.0,
1308.0,
997.0,
1071.0,
1247.0,
1234.0,
1297.0,
701.0,
1294.0,
1084.0,
740.0,
1068.0,
1442.0,
1355.0,
961.0,
1285.0,
1078.0,
1064.0,
1108.0,
395.0,
1206.0,
1106.0,
1128.0,
1421.0,
```

```
363.0,
631.0,
1134.0,
416.0,
1067.0,
1372.0,
1378.0,
430.0,
1390.0,
1101.0,
1156.0,
1069.0,
1091.0,
1121.0,
376.0,
1122.0,
1089.0,
1075.0,
1086.0,
1144.0,
364.0,
1137.0,
1130.0,
384.0,
1083.0,
1135.0,
1112.0,
947.0,
443.0,
1227.0,
1326.0,
586.0,
1210.0,
778.0,
636.0,
1226.0,
644.0,
1303.0,
1230.0,
1203.0,
677.0,
1235.0,
1283.0,
708.0,
1244.0,
1251.0,
757.0,
758.0,
```

```
 575.0,
 1208.0,
 1188.0,
 1196.0,
 504.0,
 499.0,
 940.0,
 1346.0,
 1167.0,
 1195.0,
 1340.0,
 972.0,
 1197.0,
 1335.0,
 1365.0,
 1334.0,
 870.0,
 724.0,
 1233.0,
 1072.0,
 974.0,
 1237.0,
 417.0,
 1202.0,
 675.0,
 662.0,
 1200.0,
 1092.0,
 653.0,
 1344.0,
 1113.0,
 1119.0,
 984.0,
 1127.0,
 1266.0]
```

```
In [28]: 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
             df_title = df.groupby(['article_id', 'title']).count().reset_index().sort_values('u
```

```
        top_articles = list(df_title['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
        df_ids = df.groupby(['article_id', 'title']).count().reset_index().sort_values('use
        articles = list(df_ids['article_id'])

        top_articles = []
        for article in articles:
            top_articles.append(str(article))

        return top_articles # Return the top article ids
```

In [29]:
```
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 [30]:
```
# 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 [31]: # Create user-by-item matrix
         user_by_item = df.groupby(['user_id', 'article_id']).count().unstack()

         user_by_item[np.isnan(user_by_item)] = 0
         user_by_item[(user_by_item)>0] = 1

         print(user_by_item)
         print(user_by_item.shape)
```

```
           title                                                        \
article_id 0.0    2.0    4.0    8.0    9.0    12.0   14.0   15.0   16.0
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    1.0    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
6            0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7            0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
8            0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0
9            0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    1.0
10           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
11           0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0    0.0
12           0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0
13           0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0
14           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
15           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
16           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
17           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
18           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
19           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
20           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
21           0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0
22           0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0    0.0
23           0.0    1.0    0.0    0.0    0.0    1.0    1.0    0.0    1.0
```

```
24             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
25             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
26             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
27             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
28             0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0    0.0
29             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
30             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
...            ...    ...    ...    ...    ...    ...    ...    ...    ...
5120           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5121           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5122           0.0    0.0    0.0    1.0    0.0    0.0    0.0    0.0    0.0
5123           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5124           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5125           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5126           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5127           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5128           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5129           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5130           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5131           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5132           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5133           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5134           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5135           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5136           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5137           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5138           0.0    0.0    0.0    0.0    0.0    1.0    1.0    0.0    0.0
5139           0.0    0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0
5140           0.0    1.0    0.0    0.0    0.0    1.0    0.0    0.0    0.0
5141           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5142           0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0
5143           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5144           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5145           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5146           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5147           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5148           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5149           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    1.0

                      ...                                                   \
article_id 18.0       ...   1434.0 1435.0 1436.0 1437.0 1439.0 1440.0 1441.0
user_id               ...
1              0.0    ...      0.0    0.0    1.0    0.0    1.0    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    1.0    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
6              0.0    ...      0.0    0.0    1.0    0.0    0.0    0.0    0.0
```

7	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0
9	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
21	0.0	...	0.0	0.0	1.0	1.0	0.0	0.0	0.0
22	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23	0.0	...	0.0	0.0	1.0	0.0	1.0	0.0	0.0
24	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0
25	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0
27	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
28	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
5120	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5121	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5122	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5123	0.0	...	0.0	0.0	1.0	1.0	0.0	0.0	0.0
5124	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5125	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5126	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5127	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0
5128	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5129	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5130	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5131	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5132	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5133	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5134	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5135	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5136	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5137	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5138	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5139	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5140	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5141	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5142	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
5143          0.0  ...      0.0    0.0    1.0    0.0    0.0    0.0    0.0
5144          0.0  ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
5145          0.0  ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
5146          0.0  ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
5147          0.0  ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
5148          0.0  ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
5149          0.0  ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0


article_id 1442.0 1443.0 1444.0
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
6             0.0    0.0    0.0
7             0.0    0.0    0.0
8             0.0    0.0    0.0
9             0.0    0.0    0.0
10            0.0    0.0    0.0
11            0.0    0.0    0.0
12            0.0    0.0    0.0
13            0.0    0.0    0.0
14            0.0    0.0    0.0
15            0.0    0.0    0.0
16            0.0    0.0    0.0
17            0.0    0.0    0.0
18            0.0    0.0    0.0
19            0.0    0.0    0.0
20            0.0    0.0    0.0
21            0.0    0.0    0.0
22            0.0    0.0    0.0
23            0.0    0.0    0.0
24            0.0    0.0    0.0
25            0.0    0.0    0.0
26            0.0    0.0    0.0
27            0.0    0.0    0.0
28            0.0    0.0    0.0
29            0.0    0.0    0.0
30            0.0    0.0    0.0
...           ...    ...    ...
5120          0.0    0.0    0.0
5121          0.0    0.0    0.0
5122          0.0    0.0    0.0
5123          0.0    0.0    0.0
5124          0.0    0.0    0.0
5125          0.0    0.0    0.0
```

```
5126          0.0    0.0    0.0
5127          0.0    0.0    0.0
5128          0.0    0.0    0.0
5129          0.0    0.0    0.0
5130          0.0    0.0    0.0
5131          0.0    0.0    0.0
5132          0.0    0.0    0.0
5133          0.0    0.0    0.0
5134          0.0    0.0    0.0
5135          0.0    0.0    0.0
5136          0.0    0.0    0.0
5137          0.0    0.0    0.0
5138          0.0    0.0    0.0
5139          0.0    0.0    0.0
5140          0.0    0.0    0.0
5141          0.0    0.0    0.0
5142          0.0    0.0    0.0
5143          0.0    0.0    0.0
5144          0.0    0.0    0.0
5145          0.0    0.0    0.0
5146          0.0    0.0    0.0
5147          0.0    0.0    0.0
5148          0.0    0.0    0.0
5149          0.0    0.0    0.0

[5149 rows x 714 columns]
(5149, 714)
```

```
In [32]: data = user_by_item['title']
         data[0]
```

```
Out[32]: user_id
         1        0.0
         2        0.0
         3        0.0
         4        0.0
         5        0.0
         6        0.0
         7        0.0
         8        0.0
         9        0.0
         10       0.0
         11       0.0
         12       0.0
         13       0.0
         14       0.0
         15       0.0
```

```
16      0.0
17      0.0
18      0.0
19      0.0
20      0.0
21      0.0
22      0.0
23      0.0
24      0.0
25      0.0
26      0.0
27      0.0
28      0.0
29      0.0
30      0.0
       ...
5120    0.0
5121    0.0
5122    0.0
5123    0.0
5124    0.0
5125    0.0
5126    0.0
5127    0.0
5128    0.0
5129    0.0
5130    0.0
5131    0.0
5132    0.0
5133    0.0
5134    0.0
5135    0.0
5136    0.0
5137    0.0
5138    0.0
5139    0.0
5140    0.0
5141    0.0
5142    0.0
5143    0.0
5144    0.0
5145    0.0
5146    0.0
5147    0.0
5148    0.0
5149    0.0
Name: 0.0, Length: 5149, dtype: float64
```

In [33]:
```python
# You need to group the data by ['user_id', 'article_id']
# and aggregation should by on ['title'] using the aggregation function count().notnull

# Create user-by-item matrix
user_by_item = df.groupby(['user_id', 'article_id']).agg({'title': "count"}).unstack()

user_by_item[np.isnan(user_by_item)] = 0
user_by_item[(user_by_item)>0] = 1

user_by_item
```

Out[33]:

	title								\
article_id	0.0	2.0	4.0	8.0	9.0	12.0	14.0	15.0	16.0
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	1.0	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
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
21	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
22	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
23	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0
24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
28	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
5120	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5121	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5122	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

```
5123          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5124          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5125          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5126          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5127          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5128          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5129          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5130          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5131          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5132          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5133          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5134          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5135          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5136          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5137          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5138          0.0    0.0    0.0    0.0    0.0    1.0    1.0    0.0    0.0
5139          0.0    0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0
5140          0.0    1.0    0.0    0.0    0.0    1.0    0.0    0.0    0.0
5141          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5142          0.0    0.0    0.0    0.0    0.0    0.0    1.0    0.0    0.0
5143          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5144          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5145          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5146          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5147          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5148          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5149          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    1.0

                  ...                                                      \
article_id 18.0   ...   1434.0 1435.0 1436.0 1437.0 1439.0 1440.0 1441.0
user_id           ...
1             0.0 ...      0.0    0.0    1.0    0.0    1.0    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    1.0    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
6             0.0 ...      0.0    0.0    1.0    0.0    0.0    0.0    0.0
7             0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
8             0.0 ...      0.0    0.0    1.0    0.0    0.0    0.0    0.0
9             0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
10            0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
11            0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
12            0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
13            0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
14            0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
15            0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
16            0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
17            0.0 ...      0.0    0.0    0.0    0.0    0.0    0.0    0.0
```

18	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
21	0.0	...	0.0	0.0	1.0	1.0	0.0	0.0	0.0
22	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23	0.0	...	0.0	0.0	1.0	0.0	1.0	0.0	0.0
24	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0
25	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0
27	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
28	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
5120	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5121	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5122	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5123	0.0	...	0.0	0.0	1.0	1.0	0.0	0.0	0.0
5124	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5125	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5126	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5127	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0
5128	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5129	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5130	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5131	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5132	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5133	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5134	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5135	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5136	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5137	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5138	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5139	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5140	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5141	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5142	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5143	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0
5144	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5145	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5146	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5147	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5148	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5149	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
article_id 1442.0 1443.0 1444.0
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
6	0.0	0.0	0.0
7	0.0	0.0	0.0
8	0.0	0.0	0.0
9	0.0	0.0	0.0
10	0.0	0.0	0.0
11	0.0	0.0	0.0
12	0.0	0.0	0.0
13	0.0	0.0	0.0
14	0.0	0.0	0.0
15	0.0	0.0	0.0
16	0.0	0.0	0.0
17	0.0	0.0	0.0
18	0.0	0.0	0.0
19	0.0	0.0	0.0
20	0.0	0.0	0.0
21	0.0	0.0	0.0
22	0.0	0.0	0.0
23	0.0	0.0	0.0
24	0.0	0.0	0.0
25	0.0	0.0	0.0
26	0.0	0.0	0.0
27	0.0	0.0	0.0
28	0.0	0.0	0.0
29	0.0	0.0	0.0
30	0.0	0.0	0.0
...	...	...	...
5120	0.0	0.0	0.0
5121	0.0	0.0	0.0
5122	0.0	0.0	0.0
5123	0.0	0.0	0.0
5124	0.0	0.0	0.0
5125	0.0	0.0	0.0
5126	0.0	0.0	0.0
5127	0.0	0.0	0.0
5128	0.0	0.0	0.0
5129	0.0	0.0	0.0
5130	0.0	0.0	0.0
5131	0.0	0.0	0.0
5132	0.0	0.0	0.0
5133	0.0	0.0	0.0
5134	0.0	0.0	0.0
5135	0.0	0.0	0.0
5136	0.0	0.0	0.0

```
5137          0.0    0.0    0.0
5138          0.0    0.0    0.0
5139          0.0    0.0    0.0
5140          0.0    0.0    0.0
5141          0.0    0.0    0.0
5142          0.0    0.0    0.0
5143          0.0    0.0    0.0
5144          0.0    0.0    0.0
5145          0.0    0.0    0.0
5146          0.0    0.0    0.0
5147          0.0    0.0    0.0
5148          0.0    0.0    0.0
5149          0.0    0.0    0.0

[5149 rows x 714 columns]
```

In [34]:
```
# 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
    user_item = df.groupby(['user_id', 'article_id']).agg({'title': "count"}).unstack()

    user_item[np.isnan(user_item)] = 0
    user_item[(user_item)>0] = 1

    return user_item.title # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [35]:
```
user_by_item.sum(axis=1)[1]
```

Out[35]:
```
36.0
```

In [36]:
```
## 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 [37]:
```python
vector = user_item @ user_item.iloc[1]
vector_sort = vector.sort_values(ascending=False)
lista = list(vector_sort.index)
lista.remove(1)
lista
```

Out[37]:
```
[2,
 3764,
 49,
 98,
 3697,
 10,
 2982,
 21,
 3782,
 290,
 4785,
 3354,
 23,
 3684,
 5083,
 38,
 273,
 4061,
 346,
 262,
 51,
 4098,
 2790,
 3169,
 58,
 3414,
 4778,
 322,
 3082,
 912,
 287,
 72,
```

```
242,
3540,
163,
2908,
3910,
4209,
170,
173,
135,
2941,
131,
4230,
204,
4145,
3870,
765,
111,
4134,
1355,
754,
746,
3783,
1063,
536,
3740,
3621,
35,
5138,
3245,
471,
4,
3622,
3057,
3238,
1262,
4904,
4901,
4900,
4255,
2270,
3243,
211,
213,
4642,
1552,
3727,
214,
763,
```

```
215,
3136,
1198,
3292,
3532,
748,
4876,
3141,
203,
351,
1890,
197,
153,
155,
2912,
4201,
4204,
3500,
1240,
3691,
488,
4619,
1372,
3378,
3376,
4934,
4933,
4932,
785,
3705,
186,
187,
188,
3654,
3879,
783,
4522,
195,
409,
230,
3201,
648,
3586,
313,
644,
641,
639,
2007,
```

```
3615,
3801,
4755,
4404,
333,
371,
4416,
621,
583,
619,
3195,
3784,
343,
4459,
4429,
3310,
3775,
601,
4774,
3763,
304,
4506,
3147,
235,
3230,
4293,
406,
246,
1338,
249,
3548,
403,
712,
3829,
3212,
696,
4494,
4487,
3570,
383,
3578,
3579,
667,
663,
2814,
820,
4595,
69,
```

```
3966,
9,
902,
2318,
60,
40,
3483,
900,
11,
958,
57,
5129,
4130,
4088,
125,
3933,
854,
17,
3968,
1033,
1409,
5118,
121,
3079,
45,
64,
46,
3912,
6,
3061,
37,
2993,
5110,
938,
907,
823,
829,
71,
1405,
94,
5140,
32,
1059,
3485,
88,
148,
2926,
2772,
```

```
607,
4047,
3197,
3069,
927,
926,
596,
3208,
4453,
1163,
2002,
4449,
1170,
3765,
4054,
2005,
4447,
2768,
999,
600,
4442,
4450,
3193,
1984,
612,
1008,
4076,
3180,
1137,
1730,
640,
4079,
4392,
4390,
4389,
3960,
3807,
646,
2994,
4082,
908,
4381,
1640,
1142,
1636,
3211,
923,
3005,
```

```
613,
4065,
1003,
4426,
3196,
918,
4409,
4417,
623,
3190,
4071,
1633,
3072,
4410,
1779,
4186,
3058,
955,
502,
504,
1215,
2693,
511,
3034,
3244,
2059,
521,
4564,
4021,
523,
3724,
980,
3235,
2049,
981,
1199,
501,
496,
4527,
1230,
1234,
4002,
972,
4010,
474,
3687,
476,
4585,
```

```
3693,
3703,
3694,
1226,
974,
490,
1583,
961,
2682,
1585,
4528,
3233,
576,
4484,
3743,
3745,
558,
3986,
4491,
2022,
4037,
1181,
651,
552,
1614,
4038,
4039,
3754,
4480,
570,
3757,
575,
554,
4502,
2043,
3733,
4025,
2042,
535,
4026,
984,
538,
4518,
3026,
948,
4034,
4031,
2721,
```

```
1193,
1192,
4511,
2725,
2727,
946,
3218,
4368,
4379,
3874,
3134,
861,
860,
4248,
1038,
2949,
1685,
4245,
855,
1035,
4242,
4241,
4149,
776,
3876,
3877,
3878,
780,
4254,
4137,
1131,
753,
3943,
3265,
1819,
880,
4272,
751,
752,
1026,
3091,
2956,
755,
757,
758,
759,
761,
2962,
```

```
3935,
866,
1886,
2891,
1040,
4203,
4212,
3890,
2907,
4171,
804,
3909,
4206,
828,
1068,
851,
1700,
809,
810,
4197,
1058,
2913,
3895,
1065,
1875,
2903,
1705,
3888,
1711,
2945,
1841,
3100,
3102,
786,
1075,
843,
3103,
2897,
1073,
3885,
1707,
794,
3123,
4167,
797,
1023,
1674,
3857,
```

```
681,
3820,
675,
4093,
676,
4094,
679,
680,
4345,
1944,
4331,
3077,
2825,
2826,
687,
688,
2829,
2830,
1119,
2820,
3819,
669,
2989,
1128,
2812,
4373,
2813,
4371,
658,
1011,
1698,
4367,
3172,
661,
662,
1126,
665,
666,
2990,
668,
1655,
694,
3856,
3083,
4107,
4108,
720,
1660,
```

```
722,
725,
726,
1663,
1668,
1016,
735,
1098,
4110,
4111,
3947,
3852,
1671,
2975,
719,
3949,
1110,
3159,
2835,
698,
2983,
701,
702,
2838,
1113,
3833,
707,
708,
2842,
3834,
711,
1018,
713,
3835,
2846,
4275,
4577,
3045,
294,
284,
2324,
285,
1317,
3404,
104,
4801,
292,
103,
```

1319,
4795,
2210,
2209,
299,
2208,
4790,
4788,
2206,
3572,
280,
4784,
1512,
260,
261,
118,
4833,
263,
2234,
3474,
1511,
113,
5023,
112,
110,
4820,
3565,
3566,
3402,
1459,
2419,
4786,
2205,
3316,
2187,
1302,
1301,
2407,
85,
3595,
3596,
330,
331,
334,
89,
335,
82,
337,

```
2336,
4741,
1299,
339,
2403,
325,
1406,
305,
97,
307,
2415,
4776,
2202,
4773,
3325,
3324,
4768,
2197,
4759,
3408,
2552,
95,
318,
3466,
92,
3590,
3320,
1331,
256,
4840,
3369,
2467,
191,
149,
146,
196,
3517,
3518,
200,
145,
4925,
4909,
1384,
3366,
3523,
3490,
209,
1386,
```

```
210,
3373,
4929,
3553,
172,
160,
3499,
4958,
4962,
165,
156,
2290,
171,
4968,
185,
176,
4969,
4940,
1483,
2465,
184,
151,
2444,
4898,
4897,
1491,
2430,
237,
3353,
4995,
132,
239,
3352,
4999,
241,
4855,
4892,
126,
4853,
245,
248,
122,
120,
4843,
1334,
3355,
232,
1499,
```

```
4994,
3363,
1387,
3528,
1353,
3486,
4883,
4882,
1350,
4987,
136,
3535,
4877,
4874,
4991,
4872,
4871,
4992,
79,
324,
405,
5111,
411,
1272,
1271,
414,
1270,
417,
418,
420,
2621,
4647,
423,
1264,
1426,
34,
1261,
33,
31,
1549,
3638,
3637,
396,
1444,
52,
3448,
3629,
48,
```

```
2149,
3630,
44,
4735,
3632,
43,
3633,
2608,
1423,
2144,
3636,
2355,
431,
1543,
3653,
1431,
3667,
2646,
3274,
2648,
5135,
1245,
456,
8,
1244,
3435,
3434,
3674,
5145,
3678,
4600,
3,
1566,
1439,
2377,
2117,
3425,
3441,
27,
26,
24,
439,
22,
1254,
18,
2359,
5123,
5124,
```

```
2360,
4623,
444,
445,
2391,
1476,
76,
5053,
379,
3305,
378,
5059,
4715,
5069,
1418,
359,
2174,
62,
5070,
4696,
4697,
63,
3611,
65,
354,
2163,
2572,
3616,
4720,
2176,
3614,
365,
5078,
352,
1281,
3602,
2180,
3625,
2597,
3300,
56,
2595,
5064,
347,
4685,
2579,
4725,
4711,
```

```
5052,
4689,
364,
3301,
5057,
67,
3314,
348,
1747,
1693,
1451,
1470,
1701,
1729,
1455,
1748,
1450,
1699,
1733,
1438,
1471,
1727,
1475,
1754,
1474,
1473,
1694,
1753,
1752,
1695,
1454,
1472,
1696,
1751,
1697,
1453,
1437,
1452,
1702,
1750,
1749,
1726,
1746,
1724,
1449,
1703,
1722,
1737,
```

```
1464,
1463,
1731,
1441,
1732,
1736,
1713,
1462,
1442,
1735,
1446,
1721,
1714,
1461,
1720,
1460,
1715,
1716,
1719,
1443,
1734,
1718,
1717,
1712,
1465,
1466,
1706,
1440,
1745,
1448,
1469,
1704,
1744,
1725,
1445,
1456,
1468,
1743,
1708,
1467,
1447,
1742,
1709,
1457,
1741,
1458,
1740,
1739,
```

```
1723,
1710,
1738,
1728,
1572,
1692,
1602,
1541,
1542,
1618,
1617,
1616,
1615,
1613,
1612,
1611,
1610,
1609,
1544,
1545,
1608,
1607,
1546,
1606,
1605,
1604,
1603,
1547,
1619,
1540,
1539,
1534,
1632,
1631,
1630,
1531,
1532,
1533,
1629,
1628,
1627,
1626,
1538,
1535,
1536,
1537,
1625,
1624,
```

```
        1623,
        1622,
        1621,
        1620,
        1548,
        1601,
        1477,
        1550,
        ...]
```

```
In [38]: 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
             # similarity = user_item @ user_item.iloc[user_id]

             # compute similarity of each user to the provided user
             dot_prod_users = user_item.dot(np.transpose(user_item))
             # sort by similarity and remove the own user's id
             similar_user_matrix = dot_prod_users.sort_values(user_id, ascending=False).drop(use
             # create list of just the ids
             most_similar_users = similar_user_matrix[user_id].index.tolist()

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

```
In [39]: # 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, 3870, 131, 4201, 46, 5041]
The 5 most similar users to user 3933 are: [1, 23, 3782, 203, 4459]
The 3 most similar users to user 46 are: [4201, 3782, 23]
```

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

```
In [40]: df.query('article_id==1314.0')['title'].iloc[0]
```

```
Out[40]: 'healthcare python streaming application demo'
```

```
In [41]: user_item[[2.0]].iloc[22].iloc[0]
```

```
Out[41]: 1.0
```

```
In [42]: for x in range(1,user_item[[2.0]].shape[0]):
             if user_item[[2.0]].iloc[x].iloc[0] == 1.0:
                 print(x)
                 #print(user_item[[2.0]].iloc[x].iloc[0])
```

```
22
45
59
97
183
216
412
459
638
646
664
667
675
788
793
894
1144
1400
1576
2101
3181
3335
3610
3638
3643
3763
3781
3929
4098
4139
4200
4265
```

```
4353
4429
4483
4556
4706
4776
4801
4871
4891
4894
5077
5139
```

In [43]:
```
user_item.shape
```

Out[43]:
```
(5149, 714)
```

In [44]:
```
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 = []

    for article in article_ids:
        #print(article)
        value = df.query('article_id==@article')['title'].iloc[0]
        #value = [df['article_id']==article]['title']
        article_names.append(value)

    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
    list_articles = []
    for x in range(1, user_item[[2.0]].shape[0]):
        if user_item[[2.0]].iloc[x].iloc[0] == 1.0:
            print(x)
            list_articles.append(x)

    article_ids = list_articles
    #print(article_ids)
    article_names = get_article_names(article_ids, df=df)

    return article_ids, article_names # return the ids and names7
    '''
    list_articles = []
    # for user_id in range(1, 5149):
    articles = user_item.iloc[user_id-1]
    for idx, row in articles.iteritems():
        if row == 1.0:
            list_articles.append(str(idx))

    article_ids = list_articles
    # print(article_ids)
    article_names = get_article_names(article_ids, df=df)

    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
    recs = []
    articles_seen_ids, articles_seen_names = get_user_articles(user_id, user_item=user_

    while len(recs) < m:
        for user in find_similar_users(user_id, user_item=user_item):
            aux_id, aux_names = get_user_articles(user, user_item=user_item)
            #print(aux_id)
            for article in aux_id:
                if (article not in articles_seen_ids) and (article not in recs) and len
                    recs.append(article)

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

```
In [45]: a = get_article_names(("1430.0", "1314.0"), df=df)
         a
```

```
Out[45]: ['using pixiedust for fast, flexible, and easier data analysis and experimentation',
          'healthcare python streaming application demo']
```

```
In [46]: b = get_user_articles(1, user_item=user_item)
         b
```

```
Out[46]: (['43.0',
           '109.0',
           '151.0',
           '268.0',
           '310.0',
           '329.0',
           '346.0',
           '390.0',
           '494.0',
           '525.0',
           '585.0',
           '626.0',
           '668.0',
           '732.0',
           '768.0',
           '910.0',
           '968.0',
           '981.0',
           '1052.0',
```

```
 '1170.0',
 '1183.0',
 '1185.0',
 '1232.0',
 '1293.0',
 '1305.0',
 '1363.0',
 '1368.0',
 '1391.0',
 '1400.0',
 '1406.0',
 '1427.0',
 '1429.0',
 '1430.0',
 '1431.0',
 '1436.0',
 '1439.0'],
['deep learning with tensorflow course by big data university',
 'tensorflow quick tips',
 'jupyter notebook tutorial',
 'sector correlations shiny app',
 'time series prediction using recurrent neural networks (lstms)',
 'introduction to market basket analysis in\xa0python',
 'fighting gerrymandering: using data science to draw fairer congressional districts',
 'introducing ibm watson studio ',
 'python for loops explained (python for data science basics #5)',
 'new shiny cheat sheet and video tutorial',
 'tidyverse practice: mapping large european cities',
 'analyze db2 warehouse on cloud data in rstudio in dsx',
 'shiny: a data scientists best friend',
 'rapidly build machine learning flows with dsx',
 'python if statements explained (python for data science basics #4)',
 'working with ibm cloud object storage in python',
 'shiny 0.13.0',
 'super fast string matching in python',
 'access db2 warehouse on cloud and db2 with python',
 'apache spark lab, part 1: basic concepts',
 'categorize urban density',
 'classify tumors with machine learning',
 'country statistics: life expectancy at birth',
 'finding optimal locations of new store using decision optimization',
 'gosales transactions for naive bayes model',
 'predict loan applicant behavior with tensorflow neural networking',
 'putting a human face on machine learning',
 'sudoku',
 'uci ml repository: chronic kidney disease data set',
 'uci: iris',
 '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',
'visualize car data with brunel',
'welcome to pixiedust',
'working with ibm cloud object storage in r'])
```

In [47]:
```python
# Check Results
import time
start = time.time()
print(start)
#get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
end = time.time()
print(end)
print(end - start)
```

```
1667212891.6216347
1667212891.6223028
0.0006680488586425781
```

In [48]:
```python
# 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 [49]:
```python
"""
neighbors_df=pd.DataFrame(columns=['neighbor_id','similarity','num_interactions'])

    user_sim=np.dot(user_item,user_item.T)

    # user_simm=find_similar_users(user_id)
```

```
            index=range(1, user_sim.shape[0]+1)

            user_sim=pd.DataFrame(user_sim, index=index, columns=index)
            user_sim=user_sim.loc[user_id]
            user_sim=user_sim.drop(user_id, axis=0)

            neighbors_df.neighbor_id=user_sim.index
            neighbors_df.index=user_sim.index
            neighbors_df.similarity=user_sim
            neighbors_df.num_interactions=df.groupby('user_id').count().sort_values('title',asc

            return neighbors_df.sort_values(by=['similarity','num_interactions'],ascending=Fals
            # Return the dataframe specified in the doc_string

            user_item.loc[user_id,:].dot(user_item.T)
            """
```

Out[49]: "\nneighbors_df=pd.DataFrame(columns=['neighbor_id','similarity','num_interactions'])\n

```
In [50]: def find_similar_users_and_similarity_and_num_interactions(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
            Returns as well the similarity and the number of articles seen

            '''
            # compute similarity of each user to the provided user
            # similarity = user_item @ user_item.iloc[user_id]

            user_sim = np.dot(user_item,user_item.T)
            user_simm = find_similar_users(user_id)
            index = range(1, user_sim.shape[0]+1)
            user_sim = pd.DataFrame(user_sim, index=index, columns=index)
            user_sim = user_sim.loc[user_id]
            user_sim = user_sim.drop(user_id, axis=0)

            most_similar_users = user_sim.index
```

```
        num_interactions = df.groupby('user_id')['article_id'].count().drop(user_id)
        #num_interactions = user_item.sum(axis=1).drop(user_id)
        #num_interactions = df.groupby('user_id').count().sort_values('title',ascending=Fal

        return most_similar_users, user_sim, num_interactions
```

```
In [51]: a, b, c = find_similar_users_and_similarity_and_num_interactions(2, user_item=user_item
```

```
In [52]: 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 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

             '''
             # Your code here
             # get_top_articles(n, df=df)

             users, similarity, interactions = find_similar_users_and_similarity_and_num_interac

             neighbors_df = pd.DataFrame(
             {'neighbor_id': users,
              'similarity': similarity,
              'num_interactions': interactions
             })

             neighbors_df = neighbors_df.sort_values(['similarity', 'num_interactions'], ascendi

             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 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.

        '''
        # Your code here
        recs = []
        articles_seen_ids, articles_seen_names = get_user_articles(user_id, user_item=user_
        dataframe_aux = get_top_sorted_users(user_id, df=df, user_item=user_item)

        while len(recs) < m:
            for user in dataframe_aux.neighbor_id:
                aux_id, aux_names = get_user_articles(user, user_item=user_item)
                for article in aux_id:
                    if (article not in articles_seen_ids) and (article not in recs) and len
                        recs.append(article)

        rec_names = get_article_names(recs, df=df)

        return recs, rec_names
```

In [53]:
```
aux = get_top_sorted_users(1, df=df, user_item=user_item)
```

In [54]:
```
# 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', '109.0', '125.0', '142.0', '164.0', '205.0', '302.0', '336.0', '362.0', '465.0']
```

```
The top 10 recommendations for user 20 are the following article names:
['timeseries data analysis of iot events by using jupyter notebook', 'tensorflow quick tips', 's
```

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 [55]:
```python
user_to_1 = get_top_sorted_users(1, df=df, user_item=user_item)
print(user_to_1)
```

```
         neighbor_id  similarity  num_interactions
user_id
3933            3933        35.0                45
23                23        17.0               364
3782            3782        17.0               363
203              203        15.0               160
4459            4459        15.0               158
131              131        14.0               145
3870            3870        14.0               144
46                46        13.0                63
4201            4201        13.0                61
49                49        12.0               147
3697            3697        12.0               145
395              395        12.0                69
5041            5041        12.0                67
242              242        11.0               148
3910            3910        11.0               147
322              322        11.0                85
3622            3622        11.0                83
98                98        10.0               170
3764            3764        10.0               169
912              912        10.0               102
3540            3540        10.0               101
290              290        10.0                80
2982            2982        10.0                79
754              754        10.0                60
4642            4642        10.0                58
21                21         9.0               137
4785            4785         9.0               136
52                52         9.0               132
3596            3596         9.0               131
204              204         9.0                97
...              ...         ...               ...
5065            5065         0.0                 1
5068            5068         0.0                 1
5071            5071         0.0                 1
5073            5073         0.0                 1
```

```
5084          5084         0.0                1
5085          5085         0.0                1
5087          5087         0.0                1
5090          5090         0.0                1
5091          5091         0.0                1
5092          5092         0.0                1
5098          5098         0.0                1
5100          5100         0.0                1
5101          5101         0.0                1
5104          5104         0.0                1
5107          5107         0.0                1
5109          5109         0.0                1
5112          5112         0.0                1
5113          5113         0.0                1
5116          5116         0.0                1
5119          5119         0.0                1
5121          5121         0.0                1
5122          5122         0.0                1
5125          5125         0.0                1
5130          5130         0.0                1
5131          5131         0.0                1
5141          5141         0.0                1
5144          5144         0.0                1
5147          5147         0.0                1
5148          5148         0.0                1
5149          5149         0.0                1

[5148 rows x 3 columns]
```

In [56]:
```python
user_to_131 = get_top_sorted_users(131, df=df, user_item=user_item)
print(user_to_131)
```

```
        neighbor_id  similarity  num_interactions
user_id
3870           3870        74.0               144
3782           3782        39.0               363
23               23        38.0               364
203             203        33.0               160
4459           4459        33.0               158
98               98        29.0               170
3764           3764        29.0               169
49               49        29.0               147
3697           3697        29.0               145
242             242        25.0               148
3910           3910        25.0               147
40               40        24.0                78
4932           4932        24.0                76
```

58	58	23.0	142
3740	3740	23.0	140
52	52	23.0	132
3596	3596	23.0	131
290	290	22.0	80
21	21	21.0	137
4785	4785	21.0	136
912	912	21.0	102
3540	3540	21.0	101
2982	2982	21.0	79
754	754	21.0	60
170	170	20.0	116
3169	3169	20.0	114
135	135	20.0	82
3621	3621	20.0	80
4642	4642	20.0	58
184	184	18.0	104
...	...	...	...
5065	5065	0.0	1
5068	5068	0.0	1
5073	5073	0.0	1
5076	5076	0.0	1
5084	5084	0.0	1
5085	5085	0.0	1
5087	5087	0.0	1
5090	5090	0.0	1
5091	5091	0.0	1
5092	5092	0.0	1
5098	5098	0.0	1
5100	5100	0.0	1
5101	5101	0.0	1
5104	5104	0.0	1
5107	5107	0.0	1
5109	5109	0.0	1
5112	5112	0.0	1
5113	5113	0.0	1
5116	5116	0.0	1
5119	5119	0.0	1
5120	5120	0.0	1
5121	5121	0.0	1
5122	5122	0.0	1
5125	5125	0.0	1
5130	5130	0.0	1
5131	5131	0.0	1
5141	5141	0.0	1
5144	5144	0.0	1
5147	5147	0.0	1
5149	5149	0.0	1

```
[5148 rows x 3 columns]
```

```
In [57]: ### 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 [58]: ## 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.

Provide your response here. As a new user does not have any interaction with any user, neither any interaction with any article, we will have to recomend them the top articles. This top articles list will be the same and it is not personalized for the especific new users.

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 [59]: 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 = get_top_article_ids(10, df=df) # Your recommendations here
```

```
In [60]: 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 [61]: 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 [62]: # 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 [63]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

```
In [64]: # quick look at the matrix
         user_item_matrix.head()
```

```
Out[64]: 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]
```

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.

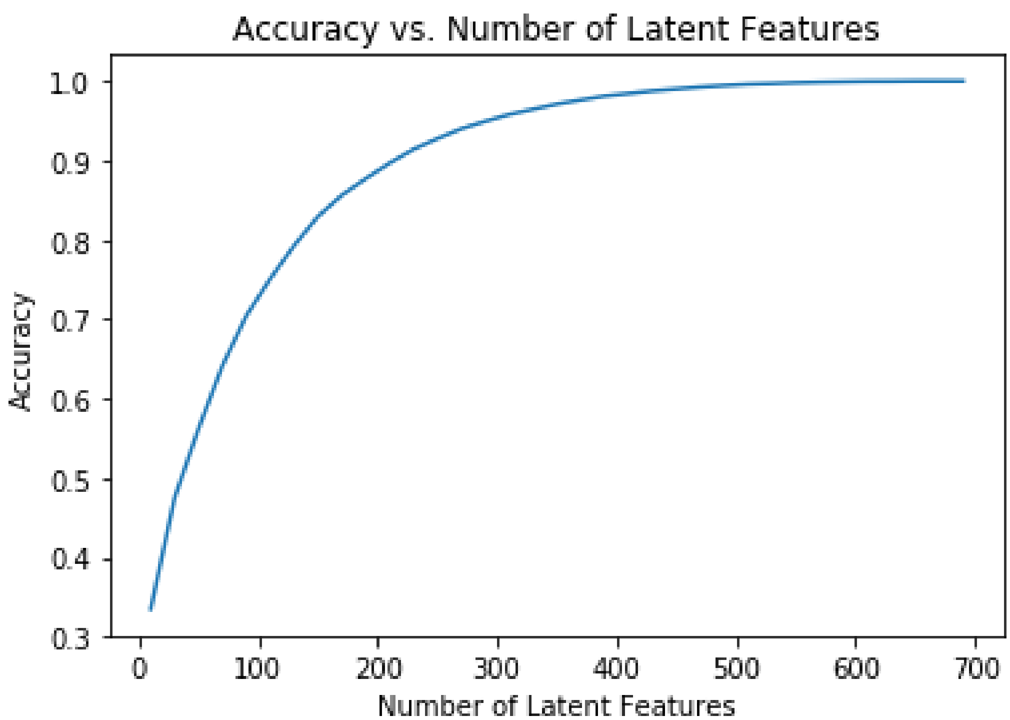
```
In [65]: # Perform SVD on the User-Item Matrix Here

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

Provide your response here. Because there are not null values

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 [66]: num_latent_feats = np.arange(10,700+10,20)
         sum_errs = []

         for k in num_latent_feats:
```

```python
    # 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 [68]:
```
df.shape
```

Out[68]: (45993, 3)

In [69]:
```
user_item_matrix.shape
```

Out[69]: (5149, 714)

In [82]:
```
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)
    test_idx = list(user_item_test.index)
    test_arts = list(user_item_test.columns)

    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(
```

In [83]:
```
user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_user_item(
print('articles of the test set we can make predictions about')
print(len(np.intersect1d(df_train.article_id.unique(),df_test.article_id.unique())))
print('articles of the test set we cant make predictions about')
print(len(df_test.article_id.unique()) - len(np.intersect1d(df_train.article_id.unique(
print('users of the test set we can make predictions about')
```

```
print(len(np.intersect1d(df_train.user_id.unique(),df_test.user_id.unique())))
print('users of the test set we cant make predictions about')
print(len(df_test.user_id.unique()) - len(np.intersect1d(df_train.user_id.unique(),df_t
```

```
articles of the test set we can make predictions about
574
articles of the test set we cant make predictions about
0
users of the test set we can make predictions about
20
users of the test set we cant make predictions about
662
```

In [84]:
```
# 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 of
}

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
```

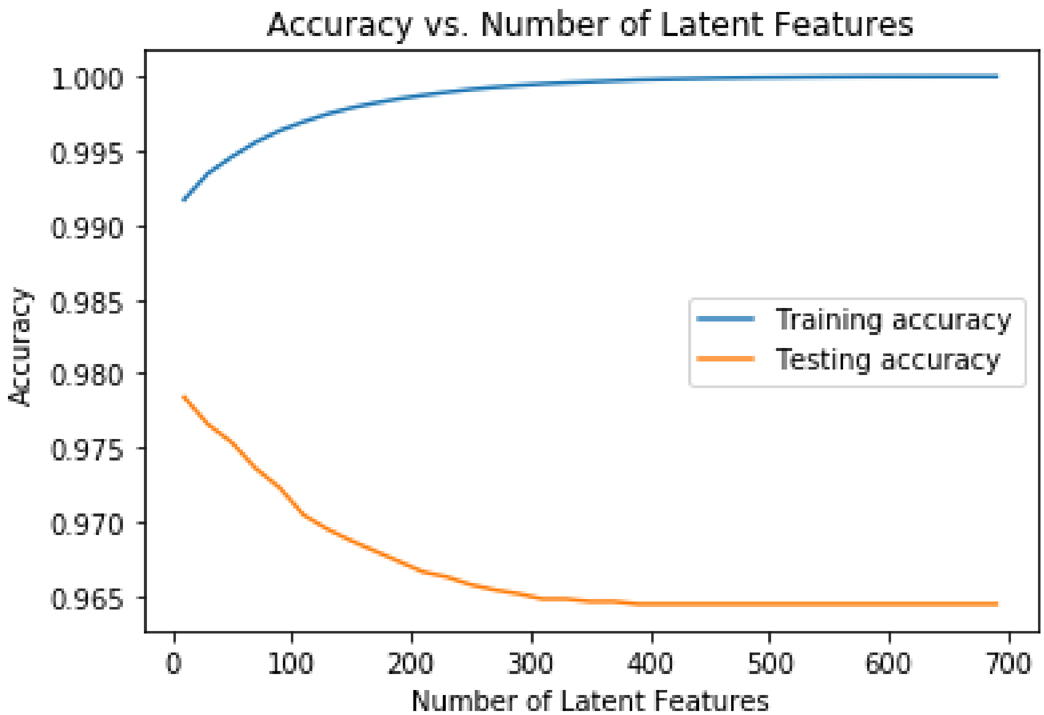
5. Now use the **user_item_train** dataset from above to find U, S, and V transpose using SVD. Then find the subset of rows in the **user_item_test** dataset that you can predict using this matrix decomposition with different numbers of latent features to see how many features makes sense to keep based 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 [87]:
```
# 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
```

In [88]:
```
# Use these cells to see how well you can use the training
# decomposition to predict on test data

num_latent_feats = np.arange(10,700+10,20)
sum_errs_train = []
```

```
sum_errs_test = []


# Decomposition
row_index = user_item_train.index.isin(test_idx)    # Getting index of common users in
col_index = user_item_train.columns.isin(test_arts) # Getting columns ('article_id') of

# Creating u_test and vt_test using the above
u_test = u_train[row_index,:]
vt_test = vt_train[:,col_index]

# subsetting common users from user_item_train
common_users = np.intersect1d(list(user_item_train.index),list(user_item_test.index))


for k in num_latent_feats:

    # restructure with k latent features for training and test set #
    s_train_new, u_train_new, vt_train_new = np.diag(s_train[:k]), u_train[:, :k], vt_t
    u_test_new, vt_test_new = u_test[:, :k], vt_test[:k, :]

    # take dot product
    user_item_est_train = np.around(np.dot(np.dot(u_train_new, s_train_new), vt_train_n
    user_item_est_test = np.around(np.dot(np.dot(u_test_new, s_train_new), vt_test_new)

    # compute error for each prediction to actual value
    diffs_train = np.subtract(user_item_train, user_item_est_train)
    diffs_test = np.subtract(user_item_test.loc[common_users,:], user_item_est_test)

    # total training errors and keep track of them
    err_train = np.sum(np.sum(np.abs(diffs_train)))
    sum_errs_train.append(err_train)

    # total testing errors and keep track of them
    err_test = np.sum(np.sum(np.abs(diffs_test)))
    sum_errs_test.append(err_test)


plt.plot(num_latent_feats, 1 - np.array(sum_errs_train)/(user_item_train.shape[0]*user_
plt.plot(num_latent_feats, 1 - np.array(sum_errs_test)/(user_item_test.loc[common_users
         , label = 'Testing accuracy');
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
plt.legend(loc='best');
```



In [ ]:

In [ ]:

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

Your response here. Is accuracy a good metric to use here in this case? Does it provide us with a fair assessment of the model's performance? Accuracy is not a good metric in this case because the training and testing sets are very imbalanced. F-beta score with an emphasis on precision for minimizing the false positives could be a better metric in this case. Is the current assessment framework robust enough to make conclusive results about the model? Think about the number of train and test users. How many users exist in your test set for whom you can make predictions? Is it sufficient? We can only predict for 20 users in our test ser. Therefore we have the cold-start problem and the current model in not robust. How will you separate the user groups? Will it be based on userIDs, cookies, devices, IP addresses? How long will we run the experiment? What metrics will be tracked during this experiment? We can use A/B tests with a control and an experiment group to see if the users in the experiment group find better articles. We will separate the users in two group: in the control group, the users will use the current way of finding articles, in the experiment group the users will get the recommendation system. We can separate the users by userIDs. We can run the experiment for one month.

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 [78]:
```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

Out[78]: 0

In [ ]: