Recommendations_with_IBM

October 3, 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 [125]: 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[125]:    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 [127]:
```python
# Show df_content to get an idea of the data
df_content.head()
```

Out[127]:
```
                                            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 [129]:
```python
print(df.isnull().sum())
print(df_content.isnull().sum())
```

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

```
doc_full_name       0
doc_status          0
article_id          0
dtype: int64
```

In [131]:
```python
df.groupby('email')['article_id'].describe(include='all')
```

Out[131]:
```
                                          count         mean         std  \
email
0000b6387a0366322d7fbfc6434af145adf7fed1   13.0   674.538462  537.898475
001055fc0bb67f71e8fa17002342b256a30254cd    4.0   538.500000  575.343086
00148e4911c7e04eeff8def7bbbdaf1c59c2c621    3.0   858.666667  567.564387
001a852ecbd6cc12ab77a785efa137b2646505fe    6.0   809.833333  495.809406
001fc95b90da5c3cb12c501d201a915e4f093290    2.0   871.500000  696.500179
0042719415c4fca7d30bd2d4e9d17c5fc570de13    2.0   540.000000  735.391052
00772abe2d0b269b2336fc27f0f4d7cb1d2b65d7    3.0  1195.333333  401.258437
008ba1d5b4ebf54babf516a2d5aa43e184865da5   10.0  1077.000000  394.747176
008ca24b82c41d513b3799d09ae276d37f92ce72    1.0   146.000000         NaN
008dfc7a327b5186244caec48e0ab61610a0c660   13.0   663.307692  486.963275
009af4e0537378bf8e8caf0ad0e2994f954d822e    1.0  1299.000000         NaN
00bda305223d05f6df5d77de41abd2a0c7d895fe    4.0  1041.000000  332.859330
00c2d5190e8c6b821b0e3848bf56f6e47e428994    3.0   423.333333  221.590463
00ced21f957bbcee5edf7b107b2bd05628b04774    4.0   590.250000  454.642992
00d9337ecd5f70fba1c4c7a78e21b3532e0112c4    3.0   241.000000    0.000000
00e524e4f13137a6fac54f9c71d7769c6507ecde   11.0   953.545455  533.066293
00f8341cbecd6af00ba8c78b3bb6ec49adf83248    3.0   893.333333  766.721157
00f946b14100f0605fa25089437ee9486378872c    1.0  1364.000000         NaN
01041260c97ab9221d923b0a2c525437f148d589    2.0  1424.000000    8.485281
0108ce3220657a9a89a85bdec959b0f2976dd51c    4.0  1327.500000   76.019734
011455e91a24c1fb815a4deac6b6eaf5ad16819e    9.0   903.888889  583.737836
01198c58d684d79c9026abe355cfb532cb524dc5    1.0  1183.000000         NaN
011ae4de07ffb332b0f51c155a35c23c80294962   35.0   854.685714  472.957949
011fcfb582be9534e9a275336f7e7c3717100381   11.0  1238.272727  410.745686
0129dfcdb701b6e1d309934be6393004c6683a2d   15.0  1015.533333  547.825424
01327bbc4fd7bfe8ad62e599453d2876b928e725    3.0   686.666667  685.219186
01455f0ab0a5a22a93d94ad35f6e78431aa90625    7.0   707.000000  212.370745
014dedab269f1453c647598c92a3fa37b39eed97    2.0   646.000000  726.905771
014e4fe6e6c5eb3fe5ca0b16c16fb4599df6375c    1.0    12.000000         NaN
01560f88312a91894d254e6406c25df19f0ad5e8   11.0   924.454545  432.383248
...                                         ...          ...         ...
fe5396e3762c36767c9c915f7ed1731691d7e4b4    1.0   910.000000         NaN
fe5480ff15f0ac51eeb2314a192351f168d7aad7    1.0   600.000000         NaN
fe56a49b62752708ed2f6e30677c57881f7b78d1   15.0  1010.266667  480.582008
fe5885b80e91be887510a0b6dd04e011178d6364    3.0  1213.000000  239.539141
fe5f9d7528518e00b0a73c7a3994afc335496961    3.0  1133.333333  160.855007
fe66aa534c7824eca663b84b99a437a98a9b026e    2.0  1127.000000  427.092496
fe69c72c964a8346dbc7763309c4e07d818d360f    4.0  1249.750000   52.500000
```

```
fe88d1f683f308b32fb3d7554f007cc55cc48df5    1.0   390.000000         NaN
fe8c1cb974e39d8ea8c005044e927b3f0de8acd0    3.0  1027.333333  463.612266
fe90d98b0287090fe8e653bafba6ed3eff19331e    1.0   162.000000         NaN
fe9327be39fd457df70e83d3fc8cba9b8b3f95b1    1.0   213.000000         NaN
feaea388105a4ccc48795b191bbf0c26a23b1356    4.0   502.500000  257.942500
fef0c6be3a2ed226e1fb8a811b0ee68a389f6f3c   13.0   368.384615  294.231637
fef28e45f7217026b2684d1783a2e18b061bdffb    3.0   424.333333  255.684050
fef3bc88def1aa787c99957ded7d5b2c0edc040e    4.0  1275.000000  236.668545
ff27ffd93e21154b8a9cf2722f2cc0f75dc39eff    1.0  1420.000000         NaN
ff288722b76eba5209cdbf9158c6dfbf229b9129    1.0   270.000000         NaN
ff452614b91f4c9bd965150b1a82e7bf18f59334    2.0   644.500000  232.638131
ff4d3e1c359cfbb73bcae07fa1eb62c45da2b161    3.0   650.000000  508.506637
ff55d0c0b2a4f56aae87c2a21afb7070ab34383d    1.0   809.000000         NaN
ff6e82c763fe2443643e48a03e239eb635f406dc   14.0   941.928571  401.796057
ff7a0f59ba022102ad22981141a7182c4d8273c3    7.0  1027.571429  578.435781
ff833869969184d86f870f98405e7988eccc2309    9.0   356.444444  338.520720
ff979e07f9d906a32ba35a9b75fd9585f6306dbc   38.0  1164.921053  302.728616
ffaefa3a1bc2d074d9a14c9924d4e67a46c35410    1.0  1299.000000         NaN
ffc6cfa435937ca0df967b44e9178439d04e3537    2.0  1053.000000    0.000000
ffc96f8fbb35aac4cb0029332b0fc78e7766bb5d    4.0  1201.500000  461.000000
ffe3d0543c9046d35c2ee3724ea9d774dff98a32   32.0   845.625000  463.342189
fff9fc3ec67bd18ed57a34ed1e67410942c4cd81   10.0   585.400000  399.498908
fffb93a166547448a0ff0232558118d59395fecd   13.0  1224.692308  289.649324

                                             min      25%     50%      75%  \
email
0000b6387a0366322d7fbfc6434af145adf7fed1    43.0   173.00   618.0  1232.00
001055fc0bb67f71e8fa17002342b256a30254cd   124.0   221.50   322.0   639.00
00148e4911c7e04eeff8def7bbbdaf1c59c2c621   258.0   595.00   932.0  1159.00
001a852ecbd6cc12ab77a785efa137b2646505fe   232.0   410.00   775.0  1262.25
001fc95b90da5c3cb12c501d201a915e4f093290   379.0   625.25   871.5  1117.75
0042719415c4fca7d30bd2d4e9d17c5fc570de13    20.0   280.00   540.0   800.00
00772abe2d0b269b2336fc27f0f4d7cb1d2b65d7   732.0  1079.50  1427.0  1427.00
008ba1d5b4ebf54babf516a2d5aa43e184865da5   315.0   827.25  1241.0  1388.75
008ca24b82c41d513b3799d09ae276d37f92ce72   146.0   146.00   146.0   146.00
008dfc7a327b5186244caec48e0ab61610a0c660    34.0   131.00   669.0  1044.00
009af4e0537378bf8e8caf0ad0e2994f954d822e  1299.0  1299.00  1299.0  1299.00
00bda305223d05f6df5d77de41abd2a0c7d895fe   547.0  1015.00  1171.5  1197.50
00c2d5190e8c6b821b0e3848bf56f6e47e428994   250.0   298.50   347.0   510.00
00ced21f957bbcee5edf7b107b2bd05628b04774    40.0   316.75   636.5   910.00
00d9337ecd5f70fba1c4c7a78e21b3532e0112c4   241.0   241.00   241.0   241.00
00e524e4f13137a6fac54f9c71d7769c6507ecde   151.0   447.00  1271.0  1383.50
00f8341cbecd6af00ba8c78b3bb6ec49adf83248     8.0   672.00  1336.0  1336.00
00f946b14100f0605fa25089437ee9486378872c  1364.0  1364.00  1364.0  1364.00
01041260c97ab9221d923b0a2c525437f148d589  1418.0  1421.00  1424.0  1427.00
0108ce3220657a9a89a85bdec959b0f2976dd51c  1276.0  1276.00  1298.5  1350.00
011455e91a24c1fb815a4deac6b6eaf5ad16819e    51.0   314.00  1330.0  1362.00
01198c58d684d79c9026abe355cfb532cb524dc5  1183.0  1183.00  1183.0  1183.00
```

011ae4de07ffb332b0f51c155a35c23c80294962	108.0	310.00	1158.0	1174.00
011fcfb582be9534e9a275336f7e7c3717100381	12.0	1314.00	1314.0	1430.50
0129dfcdb701b6e1d309934be6393004c6683a2d	43.0	778.50	1293.0	1392.50
01327bbc4fd7bfe8ad62e599453d2876b928e725	92.0	312.00	532.0	984.00
01455f0ab0a5a22a93d94ad35f6e78431aa90625	390.0	569.00	761.0	860.50
014dedab269f1453c647598c92a3fa37b39eed97	132.0	389.00	646.0	903.00
014e4fe6e6c5eb3fe5ca0b16c16fb4599df6375c	12.0	12.00	12.0	12.00
01560f88312a91894d254e6406c25df19f0ad5e8	151.0	673.00	958.0	1286.00
...	...	...	...	...
fe5396e3762c36767c9c915f7ed1731691d7e4b4	910.0	910.00	910.0	910.00
fe5480ff15f0ac51eeb2314a192351f168d7aad7	600.0	600.00	600.0	600.00
fe56a49b62752708ed2f6e30677c57881f7b78d1	2.0	1112.50	1170.0	1243.00
fe5885b80e91be887510a0b6dd04e011178d6364	943.0	1119.50	1296.0	1348.00
fe5f9d7528518e00b0a73c7a3994afc335496961	959.0	1062.00	1165.0	1220.50
fe66aa534c7824eca663b84b99a437a98a9b026e	825.0	976.00	1127.0	1278.00
fe69c72c964a8346dbc7763309c4e07d818d360f	1171.0	1249.75	1276.0	1276.00
fe88d1f683f308b32fb3d7554f007cc55cc48df5	390.0	390.00	390.0	390.00
fe8c1cb974e39d8ea8c005044e927b3f0de8acd0	492.0	893.50	1295.0	1295.00
fe90d98b0287090fe8e653bafba6ed3eff19331e	162.0	162.00	162.0	162.00
fe9327be39fd457df70e83d3fc8cba9b8b3f95b1	213.0	213.00	213.0	213.00
feaea388105a4ccc48795b191bbf0c26a23b1356	310.0	349.00	411.0	564.50
fef0c6be3a2ed226e1fb8a811b0ee68a389f6f3c	8.0	124.00	237.0	547.00
fef28e45f7217026b2684d1783a2e18b061bdffb	131.0	336.50	542.0	571.00
fef3bc88def1aa787c99957ded7d5b2c0edc040e	928.0	1222.00	1373.0	1426.00
ff27ffd93e21154b8a9cf2722f2cc0f75dc39eff	1420.0	1420.00	1420.0	1420.00
ff288722b76eba5209cdbf9158c6dfbf229b9129	270.0	270.00	270.0	270.00
ff452614b91f4c9bd965150b1a82e7bf18f59334	480.0	562.25	644.5	726.75
ff4d3e1c359cfbb73bcae07fa1eb62c45da2b161	143.0	395.00	647.0	903.50
ff55d0c0b2a4f56aae87c2a21afb7070ab34383d	809.0	809.00	809.0	809.00
ff6e82c763fe2443643e48a03e239eb635f406dc	33.0	900.75	1054.0	1168.50
ff7a0f59ba022102ad22981141a7182c4d8273c3	162.0	743.50	1338.0	1402.00
ff833869969184d86f870f98405e7988eccc2309	57.0	120.00	193.0	600.00
ff979e07f9d906a32ba35a9b75fd9585f6306dbc	153.0	1165.00	1267.0	1330.00
ffaefa3a1bc2d074d9a14c9924d4e67a46c35410	1299.0	1299.00	1299.0	1299.00
ffc6cfa435937ca0df967b44e9178439d04e3537	1053.0	1053.00	1053.0	1053.00
ffc96f8fbb35aac4cb0029332b0fc78e7766bb5d	510.0	1201.50	1432.0	1432.00
ffe3d0543c9046d35c2ee3724ea9d774dff98a32	26.0	350.75	831.0	1325.50
fff9fc3ec67bd18ed57a34ed1e67410942c4cd81	116.0	268.00	604.5	684.00
fffb93a166547448a0ff0232558118d59395fecd	329.0	1305.00	1305.0	1305.00

	max
email	
0000b6387a0366322d7fbfc6434af145adf7fed1	1354.0
001055fc0bb67f71e8fa17002342b256a30254cd	1386.0
00148e4911c7e04eeff8def7bbbdaf1c59c2c621	1386.0
001a852ecbd6cc12ab77a785efa137b2646505fe	1364.0
001fc95b90da5c3cb12c501d201a915e4f093290	1364.0
0042719415c4fca7d30bd2d4e9d17c5fc570de13	1060.0

00772abe2d0b269b2336fc27f0f4d7cb1d2b65d7	1427.0
008ba1d5b4ebf54babf516a2d5aa43e184865da5	1431.0
008ca24b82c41d513b3799d09ae276d37f92ce72	146.0
008dfc7a327b5186244caec48e0ab61610a0c660	1393.0
009af4e0537378bf8e8caf0ad0e2994f954d822e	1299.0
00bda305223d05f6df5d77de41abd2a0c7d895fe	1274.0
00c2d5190e8c6b821b0e3848bf56f6e47e428994	673.0
00ced21f957bbcee5edf7b107b2bd05628b04774	1048.0
00d9337ecd5f70fba1c4c7a78e21b3532e0112c4	241.0
00e524e4f13137a6fac54f9c71d7769c6507ecde	1410.0
00f8341cbecd6af00ba8c78b3bb6ec49adf83248	1336.0
00f946b14100f0605fa25089437ee9486378872c	1364.0
01041260c97ab9221d923b0a2c525437f148d589	1430.0
0108ce3220657a9a89a85bdec959b0f2976dd51c	1437.0
011455e91a24c1fb815a4deac6b6eaf5ad16819e	1436.0
01198c58d684d79c9026abe355cfb532cb524dc5	1183.0
011ae4de07ffb332b0f51c155a35c23c80294962	1427.0
011fcfb582be9534e9a275336f7e7c3717100381	1432.0
0129dfcdb701b6e1d309934be6393004c6683a2d	1436.0
01327bbc4fd7bfe8ad62e599453d2876b928e725	1436.0
01455f0ab0a5a22a93d94ad35f6e78431aa90625	939.0
014dedab269f1453c647598c92a3fa37b39eed97	1160.0
014e4fe6e6c5eb3fe5ca0b16c16fb4599df6375c	12.0
01560f88312a91894d254e6406c25df19f0ad5e8	1400.0
...	...
fe5396e3762c36767c9c915f7ed1731691d7e4b4	910.0
fe5480ff15f0ac51eeb2314a192351f168d7aad7	600.0
fe56a49b62752708ed2f6e30677c57881f7b78d1	1416.0
fe5885b80e91be887510a0b6dd04e011178d6364	1400.0
fe5f9d7528518e00b0a73c7a3994afc335496961	1276.0
fe66aa534c7824eca663b84b99a437a98a9b026e	1429.0
fe69c72c964a8346dbc7763309c4e07d818d360f	1276.0
fe88d1f683f308b32fb3d7554f007cc55cc48df5	390.0
fe8c1cb974e39d8ea8c005044e927b3f0de8acd0	1295.0
fe90d98b0287090fe8e653bafba6ed3eff19331e	162.0
fe9327be39fd457df70e83d3fc8cba9b8b3f95b1	213.0
feaea388105a4ccc48795b191bbf0c26a23b1356	878.0
fef0c6be3a2ed226e1fb8a811b0ee68a389f6f3c	855.0
fef28e45f7217026b2684d1783a2e18b061bdffb	600.0
fef3bc88def1aa787c99957ded7d5b2c0edc040e	1426.0
ff27ffd93e21154b8a9cf2722f2cc0f75dc39eff	1420.0
ff288722b76eba5209cdbf9158c6dfbf229b9129	270.0
ff452614b91f4c9bd965150b1a82e7bf18f59334	809.0
ff4d3e1c359cfbb73bcae07fa1eb62c45da2b161	1160.0
ff55d0c0b2a4f56aae87c2a21afb7070ab34383d	809.0
ff6e82c763fe2443643e48a03e239eb635f406dc	1424.0
ff7a0f59ba022102ad22981141a7182c4d8273c3	1402.0
ff833869969184d86f870f98405e7988eccc2309	965.0

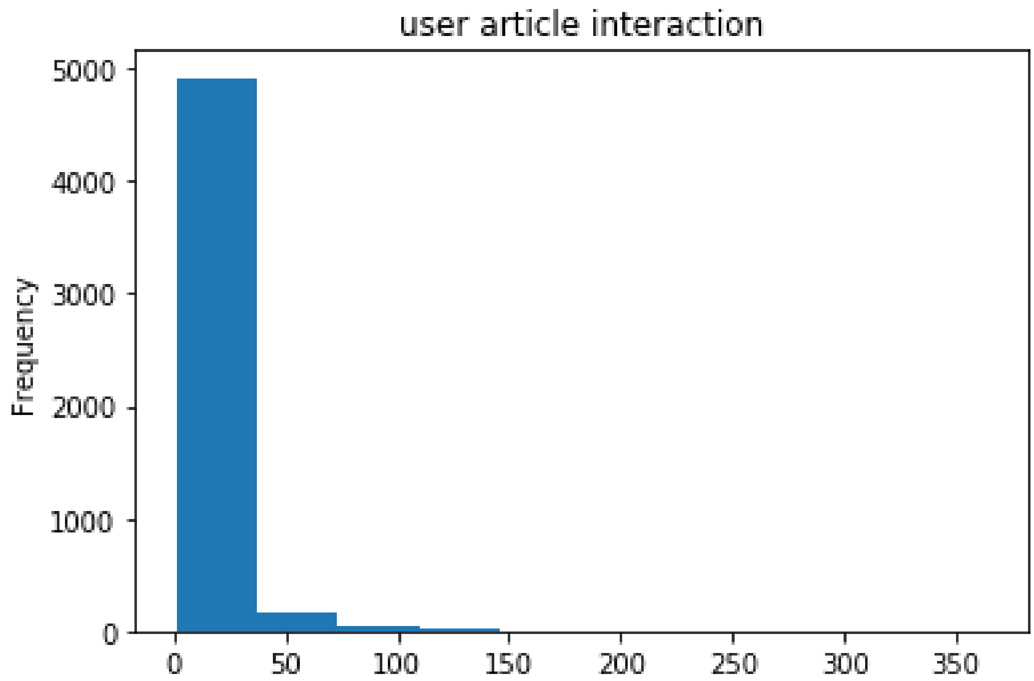
```
ff979e07f9d906a32ba35a9b75fd9585f6306dbc  1425.0
ffaefa3a1bc2d074d9a14c9924d4e67a46c35410  1299.0
ffc6cfa435937ca0df967b44e9178439d04e3537  1053.0
ffc96f8fbb35aac4cb0029332b0fc78e7766bb5d  1432.0
ffe3d0543c9046d35c2ee3724ea9d774dff98a32  1427.0
fff9fc3ec67bd18ed57a34ed1e67410942c4cd81  1431.0
fffb93a166547448a0ff0232558118d59395fecd  1437.0

[5148 rows x 8 columns]
```

In [133]:
```
df.groupby('email')['article_id'].value_counts()
```

Out[133]:
```
email                                     article_id
0000b6387a0366322d7fbfc6434af145adf7fed1  43.0          2
                                          124.0         1
                                          173.0         1
                                          288.0         1
                                          349.0         1
                                          618.0         1
                                          732.0         1
                                          1162.0        1
                                          1232.0        1
                                          1314.0        1
                                          1337.0        1
                                          1354.0        1
001055fc0bb67f71e8fa17002342b256a30254cd  124.0         1
                                          254.0         1
                                          390.0         1
                                          1386.0        1
00148e4911c7e04eeff8def7bbbdaf1c59c2c621  258.0         1
                                          932.0         1
                                          1386.0        1
001a852ecbd6cc12ab77a785efa137b2646505fe  1364.0        2
                                          232.0         1
                                          349.0         1
                                          593.0         1
                                          957.0         1
001fc95b90da5c3cb12c501d201a915e4f093290  379.0         1
                                          1364.0        1
0042719415c4fca7d30bd2d4e9d17c5fc570de13  20.0          1
                                          1060.0        1
00772abe2d0b269b2336fc27f0f4d7cb1d2b65d7  1427.0        2
                                          732.0         1
                                                       ..
ffe3d0543c9046d35c2ee3724ea9d774dff98a32  351.0         1
                                          448.0         1
                                          607.0         1
                                          617.0         1
```

```
                                          701.0         1
                                          727.0         1
                                          782.0         1
                                          784.0         1
                                          878.0         1
                                          943.0         1
                                          986.0         1
                                          1047.0        1
                                          1162.0        1
                                          1165.0        1
                                          1386.0        1
                                          1425.0        1
                                          1427.0        1
fff9fc3ec67bd18ed57a34ed1e67410942c4cd81  684.0         3
                                          268.0         2
                                          116.0         1
                                          232.0         1
                                          525.0         1
                                          962.0         1
                                          1431.0        1
fffb93a166547448a0ff0232558118d59395fecd  1305.0        8
                                          329.0         1
                                          981.0         1
                                          1304.0        1
                                          1430.0        1
                                          1437.0        1
Name: article_id, Length: 33669, dtype: int64
```

In [135]:
```
df.groupby('email')['article_id'].count().plot(kind="hist")
plt.title("user article interaction");
```




```
In [137]: print(df.groupby('email')['article_id'].count().median())
          print(df.groupby('email')['article_id'].count().max())
```

```
3.0
364
```

```
In [139]: # Fill in the median and maximum number of user_article interactios below
          # 50% of individuals interact with ____ number of articles or fewer
          median_val = df.groupby('email')['article_id'].count().median()
          #print('50% of individuals interact with {} number of article or fewer'.format(median_
           # The maximum number of user-article interactions by any 1 user is ______.
          max_views_by_user =df.groupby('email')['article_id'].count().max()
          #print('The maximum number of user-article interactions by any 1 user is {}'.format(ma
```

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

```
In [141]: # Find and explore duplicate articles
          #print('Number of rows in the original df_content: {}'.format(df_content.shape[0]))
          #print('Number of duplicate articles: {}'.format(df_content.duplicated('article_id').s
```

```
In [143]: # Remove any rows that have the same article_id - only keep the first
          df_content = df_content.drop_duplicates('article_id', keep='first')
          df_content.shape
```

```
Out[143]: (1051, 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 [145]:
```
## unique_articles = # The number of unique articles that have at least one interactio
## total_articles = # The number of unique articles on the IBM platform
## unique_users = # The number of unique users
## user_article_interactions = # The number of user-article interactions
```

In [147]:
```
# The number of unique articles that have at least one interaction
unique_articles = df[df.email.isnull() == False].article_id.nunique()
# The number of unique articles in the dataset
total_articles = df_content.article_id.nunique()
# The number of unique users
unique_users = df.email.nunique()
# The number of user-article interactions in the dateset
user_article_interactions = df.shape[0]
#print('The number of unique articles that have at least one interaction: {}'.format(u
#print('The number of unique articles on the IBM platform: {}'.format(total_articles))
#print('The number of unique users: {}'.format(unique_users))
#print('The number of user-article interactions: {}'.format(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 [149]:
```
# The most viewed article in the dataset as a string with one value following the deci
most_viewed_article_id =df[df.email.isnull() == False]['article_id'].value_counts()

most_viewed_article_id=most_viewed_article_id.head(1).index.astype(str)
most_viewed_article_id
#most_viewed_article_id='1429.0'
```

Out[149]:
```
Index(['1429.0'], dtype='object')
```

In [150]:
```
# The most viewed article in the dataset was viewed how many times?
max_views = df[df.email.isnull() == False].groupby('article_id')['email'].count().max(
max_views
```

Out[150]:
```
937
```

In [151]:
```
## No need to change the code here - this will be helpful for later parts of the noteb
# Run this cell to map the user email to a user_id column and remove the email column

def email_mapper():
    coded_dict = dict()
```

```
    cter = 1
    email_encoded = []

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

        email_encoded.append(coded_dict[val])
    return email_encoded

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

# show header
df.head()
```

Out[151]:

	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 [152]:
```
## 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_
    '`The maximum number of user-article interactions by any 1 user is ______.`': max_
    '`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_arti
    '`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 [153]: 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
              articles_desc=df.groupby('title')['article_id'].count().sort_values(ascending=Fals
              top_articles=articles_desc.head(n).index
              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
              article_id_desce=df['article_id'].value_counts().sort_values(ascending=False)
              top_articles_id=article_id_desce.head(n).index

              return top_articles_id # Return the top article ids
```

```
In [154]: print(get_top_articles(10))
          print(get_top_article_ids(10))
```

```
Index(['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'],
```

```
      dtype='object', name='title')
Float64Index([1429.0, 1330.0, 1431.0, 1427.0, 1364.0, 1314.0, 1293.0, 1170.0,
              1162.0, 1304.0],
             dtype='float64')
```

In [155]:
```
# 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 [156]:
```
# 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
    '''
    df_new=df.pivot_table(index='user_id',columns='article_id', values='title', aggfun
    df_new=df_new.replace(np.nan, 0)
    user_item=df_new.applymap(lambda x: 1 if x > 0 else x)
    #user_item=df.drop_duplicates().groupby(['user_id', 'article_id']).size().unstack(
    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [157]:
```
## 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 mat
assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article m
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 d
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 [158]:
```
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 produc
                    are listed first

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

    '''
    # compute similarity of each user to the provided user

    user_item = pd.DataFrame(user_item)

    user_similarity = user_item.dot(np.transpose(user_item))

    # sort by similarity
```

```
        user_similarity = user_similarity.sort_values(by=user_id, ascending=False)

        # remove the own user's id

        user_similarity.drop(user_id, axis=0, inplace=True)

        # create list of just the ids

        most_similar_users = list(user_similarity.index)


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

In [159]:
```
# 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 [160]:
```
user_item.head()
```

Out[160]:
```
article_id  0.0     2.0     4.0     8.0     9.0     12.0    14.0    15.0    \
user_id
1              0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
2              0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
3              0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0
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  16.0    18.0     ...    1434.0  1435.0  1436.0  1437.0  1439.0  \
user_id                      ...
1              0.0     0.0   ...       0.0     0.0     1.0     0.0     1.0
2              0.0     0.0   ...       0.0     0.0     0.0     0.0     0.0
3              0.0     0.0   ...       0.0     0.0     1.0     0.0     0.0
4              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

article_id  1440.0  1441.0  1442.0  1443.0  1444.0
user_id
```

```
1              0.0    0.0    0.0    0.0    0.0
2              0.0    0.0    0.0    0.0    0.0
3              0.0    0.0    0.0    0.0    0.0
4              0.0    0.0    0.0    0.0    0.0
5              0.0    0.0    0.0    0.0    0.0

[5 rows x 714 columns]
```

In [161]:
```
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_id in article_ids:
        article_names.append(df[df['article_id']==float(article_id)]['title'].unique()

    article_names=list(article_names)

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


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 use
    '''
    # Your code here
    article_ids = user_item.loc[user_id][user_item.loc[user_id] == 1].index.astype('st
    article_names=get_article_names(article_ids)

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

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

    OUTPUT:
    recs - (list) a list of recommendations for the user

    Description:
    Loops through the users based on closeness to the input user_id
    For each user - finds articles the user hasn't seen before and provides them as re
    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

    # find the similar users
    similar_user_ids=find_similar_users(user_id)

    # get a list of articles the user has interacted with (to exclude from recs)
    article_ids_seen, article_names_seen = get_user_articles(user_id)
    # get a list of articles unseen by the user that most similar users have interacte
    recs_ids = []
    for similar_user_id in similar_user_ids:
        similar_article_ids, similar_article_names = get_user_articles(similar_user_id
        new_recommed=np.setdiff1d(similar_article_ids, article_ids_seen, assume_unique
        recs_ids.append(new_recommed)
    unique_recommend = np.unique(np.concatenate(recs_ids))
    recs = unique_recommend[:m]

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

In [162]:
```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

Out[162]:
```
['detect malfunctioning iot sensors with streaming analytics',
 'use data assets in a project using ibm data catalog',
 'recommender systems: approaches & algorithms',
 'how to get a job in deep learning',
 'essentials of machine learning algorithms (with python and r codes)',
```

```
 "2875    hugo larochelle's neural network & deep learni...\nName: title, dtype: objec
 'how to choose a project to practice data science',
 '1448    i ranked every intro to data science course on...\nName: title, dtype: objec
 'enhanced color mapping',
 'why you should master r (even if it might eventually become obsolete)']
```

In [163]:
```
# 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.
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): u
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 demographi
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.0', '1
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-r
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 [164]:
```
def get_top_sorted_users(user_id, df=df, user_item=user_item):
    '''
    INPUT:
    user_id - (int)
    df - (pandas dataframe) df as defined at the top of the notebook
    user_item - (pandas dataframe) matrix of users by articles:
            1's when a user has interacted with an article, 0 otherwise


    OUTPUT:
    neighbors_df - (pandas dataframe) a dataframe with:
                    neighbor_id - is a neighbor user_id
                    similarity - measure of the similarity of each user to the provide
                    num_interactions - the number of articles viewed by the user - if

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

    '''
    # Your code here
```

```
    n_users = user_item.shape[0]

    neighbors_ids = []
    similarities = []
    interactions = []

    for i in range(1, n_users+1):
        neighbors_ids.append(i)
        similarities.append(np.dot(user_item.loc[user_id], user_item.loc[i]))
        #print(df[df['user_id'] == i].user_id.value_counts().item())
        interactions.append(df[df['user_id'] == i].user_id.value_counts().item())


    neighbors_data = {'neighbor_id': neighbors_ids, 'similarity': similarities, 'num_i

    neighbors_df = pd.DataFrame.from_dict(neighbors_data)

    neighbors_df.drop(neighbors_df[neighbors_df.neighbor_id == user_id].index[0], inpl
    neighbors_df.sort_values(by=['similarity', 'num_interactions'], ascending=False, i

    return neighbors_df # Return the dataframe specified in the doc_string


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

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

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

    Notes:
    * Choose the users that have the most total article interactions
    before choosing those with fewer article interactions.

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

    '''
    # Your code here
```

```
    recs = []
    rec_names =[]

    # Article ids the user has seen already
    articles_seen = get_user_articles(user_id, user_item=user_item)[0].sort_values(asc
    #articles_seen = get_top_article_ids(m)
    #df['article_id'].value_counts().sort_values(ascending=False)

    # Get a litst of sorted similar users i.e neighbor users by using get_top_sorted_u
    most_similar_users = get_top_sorted_users(user_id).neighbor_id.values.tolist()

    # Find the articles seen by similar users and add them to recs \
    for user in most_similar_users:
        article_ids = get_user_articles(user, user_item=user_item)[0].sort_values(asce
        recs.extend(np.setdiff1d(article_ids, recs, assume_unique=True))

        # Remove the articles the user has seen if it was added to rec
        for r in recs:
            if r in articles_seen:
                recs.remove(r)

        # if the number of recs exceeds m, get the first m articles in rec
        if len(recs) > m-1:
            recs = recs[:m]
            break


    rec_names = get_article_names(recs)


    return recs, rec_names
```

In [165]:
```
# 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:
['981.0', '939.0', '911.0', '880.0', '793.0', '761.0', '730.0', '686.0', '681.0', '651.0']

The top 10 recommendations for user 20 are the following article names:
['super fast string matching in python', 'deep learning from scratch i: computational graphs', '
```

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 [166]: ### Tests with a dictionary of results

          user1_most_sim = get_top_sorted_users(1).iloc[0]['neighbor_id']# Find the user that is
          #print(user1_most_sim)
          user131_10th_sim = get_top_sorted_users(131).iloc[9]['neighbor_id']# Find the 10th mos
          #print(user131_10th_sim)

In [167]: ## 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.

The rank based functions as above such as get_top_articles_ids and get_top_article, which is able to get article id and article title separately. Since the new user would not have any background information which could be used on collabarative filering algorithm, therefore rank based recommendation is the option.

I would thought a sequence-based recommendation system can be a good choice. It could implement rank based recommendation or content based recommendation systems first for the new user, then based on the events the new user provided then to use collabrative recommendations.

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 [168]: new_user = '0.0'

          # What would your recommendations be for this new user '0.0'?  As a new user, they hav
          # Provide a list of the top 10 article ids you would give to
          new_user_recs =  [str(id) for id in get_top_article_ids(10)] # Your recommendations he

In [169]: 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 [170]: 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 [171]: # 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 [172]: # Load the matrix here
          user_item_matrix = pd.read_pickle('user_item_matrix.p')

In [173]: # quick look at the matrix
          user_item_matrix.head()

Out[173]: 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.

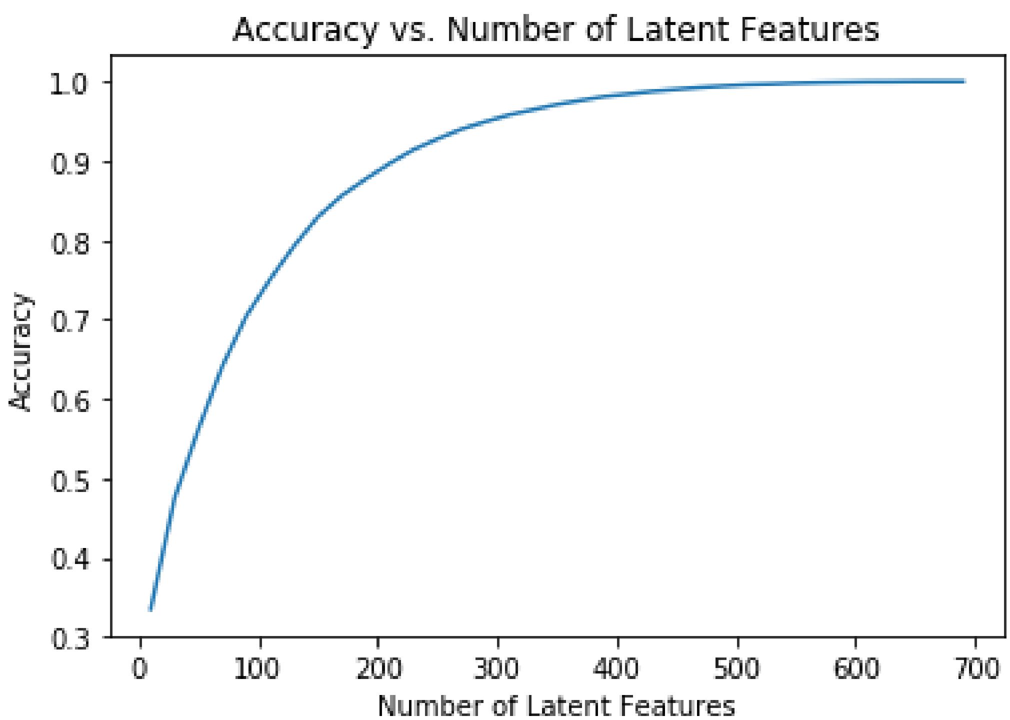
```
In [174]: # Perform SVD on the User-Item Matrix Here

          u, s, vt = np.linalg.svd(user_item_matrix)# use the built in to get the three matrices
```

Provide your response here.

When implement function of create_user_item_matrix missing values has been droped therefore we could use SVD, wherease in the lesson the matrix has missing 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 [175]:
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs = []

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

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

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

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


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



4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are

able to make good recommendations. Instead, we might split our dataset into a training and test set of data, as shown in the cell below.

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

- How many users can we make predictions for in the test set?

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

- How many articles are we not able to make predictions for because of the cold start problem?

In [176]:
```
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=user_item_test.index
    test_arts=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
print(test_idx)
print(test_arts)
```

```
Int64Index([2917, 3024, 3093, 3193, 3527, 3532, 3684, 3740, 3777, 3801,
            ...
            5140, 5141, 5142, 5143, 5144, 5145, 5146, 5147, 5148, 5149],
           dtype='int64', name='user_id', length=682)
Float64Index([   0.0,    2.0,    4.0,    8.0,    9.0,   12.0,   14.0,   15.0,
```

```
          16.0,   18.0,
        ...
        1432.0, 1433.0, 1434.0, 1435.0, 1436.0, 1437.0, 1439.0, 1440.0,
        1441.0, 1443.0],
       dtype='float64', name='article_id', length=574)
```

In [177]:
```python
user_item_test.shape
```

Out[177]: (682, 574)

In [178]:
```python
user_item_train.shape
```

Out[178]: (4487, 714)

In [179]:
```python
len(np.intersect1d(test_idx, list(user_item_train.index)))
```

Out[179]: 20

In [180]:
```python
len(np.intersect1d(test_arts, list(user_item_train.columns)))
```

Out[180]: 574

In [181]:
```python
len(np.setdiff1d(user_item_test.columns, user_item_train.columns))
```

Out[181]: 0

In [182]:
```python
len(np.setdiff1d(user_item_test.index, user_item_train.index))
```

Out[182]: 662

In [183]:
```python
# Replace the values in the dictionary below
a = 662
b = 574
c = 20
d = 0


sol_4_dict = {
    'How many users can we make predictions for in the test set?': c,
    'How many users in the test set are we not able to make predictions for because of
    'How many movies can we make predictions for in the test set?': b,
    'How many movies in the test set are we not able to make predictions for because o
}

t.sol_4_test(sol_4_dict)
```

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

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

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

In [184]:
```
# fit SVD on the user_item_train matrix
u_train, s_train, vt_train = np.linalg.svd(user_item_train) # fit svd similar to above

u_train.shape, s_train.shape, vt_train.shape
```

Out[184]: ((4487, 4487), (714,), (714, 714))

In [185]:
```
# Use these cells to see how well you can use the training
# decomposition to predict on test data
```

In [186]:
```
common_rows = user_item_train.index.isin(test_idx)
common_cols = user_item_train.columns.isin(test_arts)

u_test = u_train[common_rows, :]
s_test = s_train
vt_test = vt_train[:, common_cols]

u_test.shape, s_test.shape, vt_test.shape
```

Out[186]: ((20, 4487), (714,), (714, 574))

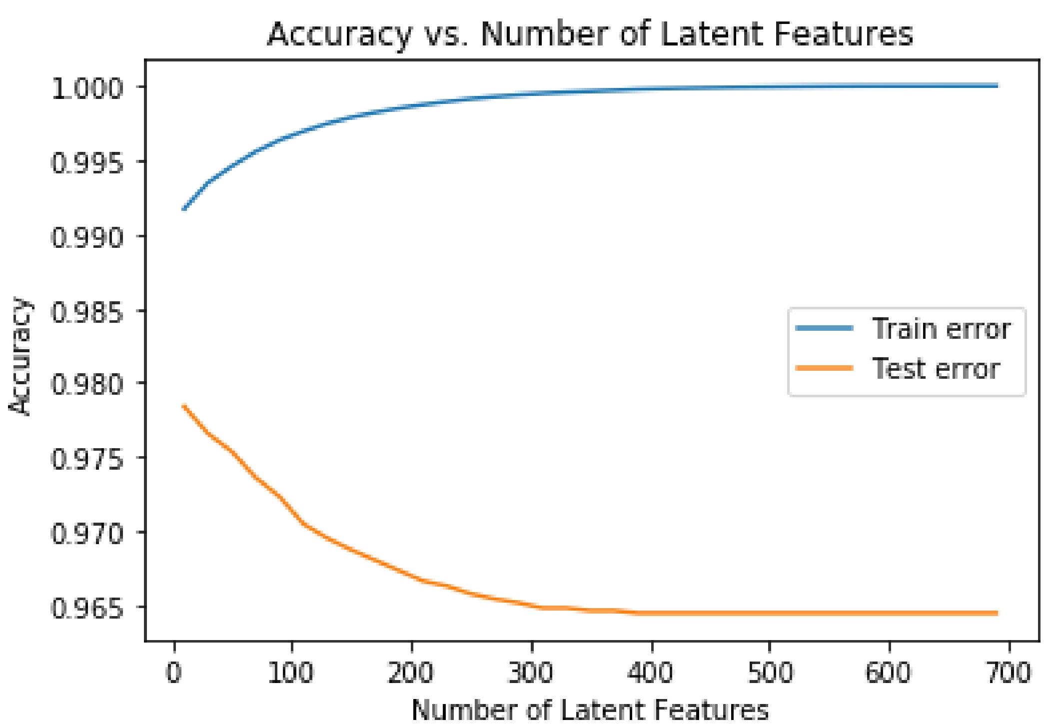
In [187]:
```
train_idx = user_item_train.index
common_idx = user_item_test.index.isin(train_idx)
sub_user_item_test = user_item_test.loc[common_idx]
```

In [188]:
```
latent_feats = np.arange(10, 700+10, 20)
train_errs, test_errs = [], []
sum_errs_train,sum_errs_test=[],[]
for k in latent_feats:
    # restructure with k latent features
    s_new_train, u_new_train, vt_new_train = np.diag(s_train[:k]), u_train[:, :k], vt_
    s_new_test, u_new_test, vt_new_test = np.diag(s_test[:k]), u_test[:, :k], vt_test[

    # take dot product
    user_item_est_train = np.around(np.dot(np.dot(u_new_train, s_new_train), vt_new_tr
    sub_user_item_est_test = np.around(np.dot(np.dot(u_new_test, s_new_test), vt_new_t

    # compute error for each prediction to actual value
    diffs_train = np.subtract(user_item_train, user_item_est_train)
    diffs_test = np.subtract(sub_user_item_test, sub_user_item_est_test)
```

```
        # Calculating the sum square errors from the predictions and actual values from th
        err_train = np.sum(np.sum(np.abs(diffs_train)))
        sum_errs_train.append(err_train)

        err_test = np.sum(np.sum(np.abs(diffs_test)))
        sum_errs_test.append(err_test)
```

```
In [189]: plt.figure()
          plt.plot(latent_feats, 1 - np.array(sum_errs_train)/(user_item_est_train.shape[0]*user
          plt.plot(latent_feats, 1 - np.array(sum_errs_test)/(sub_user_item_est_test.shape[0]*su
          plt.xlabel('Number of Latent Features')
          plt.ylabel('Accuracy')
          plt.title('Accuracy vs. Number of Latent Features')
          plt.legend()
```

```
Out[189]: <matplotlib.legend.Legend at 0x7f882ee87e10>
```



`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.*

Based on the chart above, we could see the accurarcy drops with more latent features which potentially means that these are noise and could not contribute to better accuracy to the predication. Further A/B study could help to identify the latent factors which most contribute to the accuracy of the predication.

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

Out[190]: 0
```