Recommendation System Project: IBM Community

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

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

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

Table of Contents

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

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import project_tests as t


df = pd.read_csv(
    'data/user-item-interactions.csv',
    dtype={'article_id': int, 'title': str, 'email': str}
)
# Show df to get an idea of the data
df.head()
```

Out[1]:

	Unnamed: 0	article_id	title	email
0	0	1430	using pixiedust for fast, flexible, and easier...	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1	1	1314	healthcare python streaming application demo	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2	2	1429	use deep learning for image classification	b96a4f2e92d8572034b1e9b28f9ac673765cd074
3	3	1338	ml optimization using cognitive assistant	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4	4	1276	deploy your python model as a restful api	f01220c46fc92c6e6b161b1849de11faacd7ccb2

Part I : Exploratory Data Analysis

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

`1.` Are there any missing values? If so, provide a count of missing values. If there are missing values in `email`, assign it the same id value `"unknown_user"`.

In [55]:
```
# Some interactions do not have a user associated with it, assume the same user.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45993 entries, 0 to 45992
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  45993 non-null  int64 
 1   article_id  45993 non-null  int64 
 2   title       45993 non-null  object
 3   email       45976 non-null  object
dtypes: int64(2), object(2)
memory usage: 1.4+ MB
```

In [3]:
```
print(f"Number of Null email values is: ")
```

```
Number of Null email values is:
```

In [11]:
```
df[df.email.isna()]
```

Out[11]:

	Unnamed: 0	article_id	title	email
25131	25146	1016	why you should master r (even if it might even...	NaN
29758	30157	1393	the nurse assignment problem	NaN
29759	30158	20	working interactively with rstudio and noteboo...	NaN
29760	30159	1174	breast cancer wisconsin (diagnostic) data set	NaN
29761	30160	62	data visualization: the importance of excludin...	NaN
35264	36016	224	using apply, sapply, lapply in r	NaN
35276	36029	961	beyond parallelize and collect	NaN
35277	36030	268	sector correlations shiny app	NaN
35278	36031	268	sector correlations shiny app	NaN
35279	36032	268	sector correlations shiny app	NaN
35280	36033	268	sector correlations shiny app	NaN
35281	36034	415	using machine learning to predict value of hom...	NaN
35282	36035	846	pearson correlation aggregation on sparksql	NaN
35283	36036	268	sector correlations shiny app	NaN
35284	36037	162	an introduction to stock market data analysis ...	NaN
42749	44112	647	getting started with apache mahout	NaN
42750	44113	965	data visualization playbook: revisiting the ba...	NaN

In [2]:
```
# Fill email NaNs with "unknown_user"
df['email'] = df['email'].fillna("unknown_user")
```

In [4]:
```
# Check if no more NaNs
df[df.email.isna()]
```

Out[4]:

	Unnamed: 0	article_id	title	email

In [5]:
```
df.iloc[35278:35284,:]
```

Out[5]:

	Unnamed: 0	article_id	title	email
35278	36031	268	sector correlations shiny app	unknown_user
35279	36032	268	sector correlations shiny app	unknown_user
35280	36033	268	sector correlations shiny app	unknown_user
35281	36034	415	using machine learning to predict value of hom...	unknown_user
35282	36035	846	pearson correlation aggregation on sparksql	unknown_user
35283	36036	268	sector correlations shiny app	unknown_user

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

In [6]:
```
df.head()
```

Out[6]:

	Unnamed: 0	article_id	title	email
0	0	1430	using pixiedust for fast, flexible, and easier...	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1	1	1314	healthcare python streaming application demo	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2	2	1429	use deep learning for image classification	b96a4f2e92d8572034b1e9b28f9ac673765cd074
3	3	1338	ml optimization using cognitive assistant	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4	4	1276	deploy your python model as a restful api	f01220c46fc92c6e6b161b1849de11faacd7ccb2

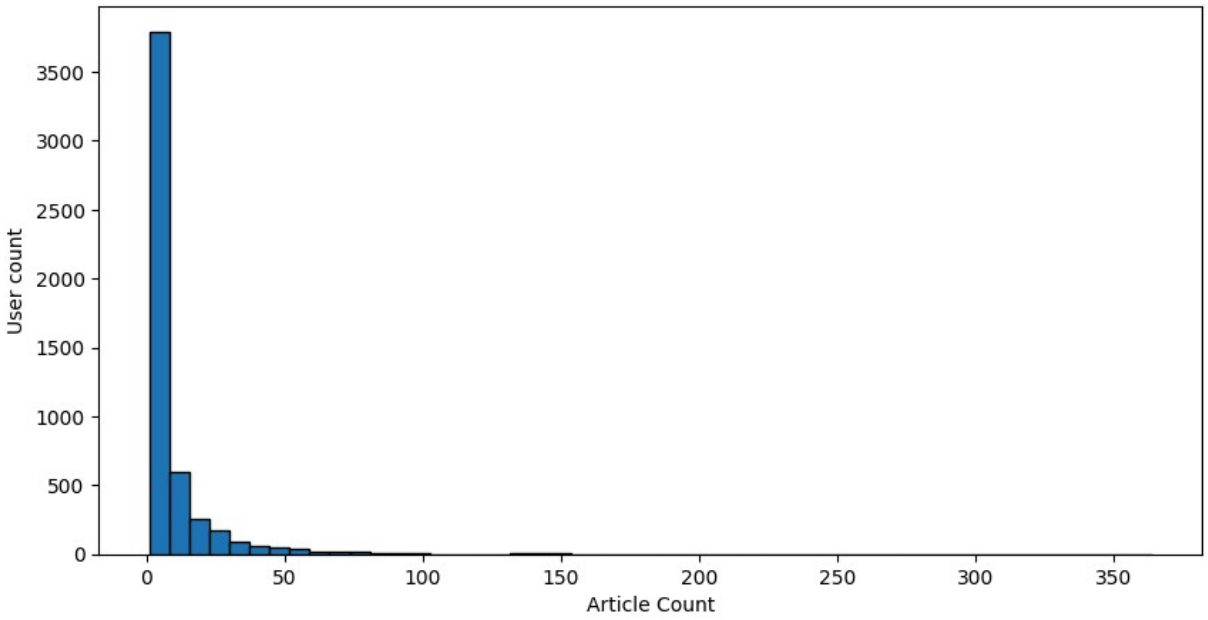
In [5]:
```
len( df['email'].unique() )
```

Out[5]: 5149

In [7]:
```
len( df['article_id'].unique()  )
```

Out[7]: 714

In [11]:
```
plt.figure(figsize=(10,5))
plt.hist(df['email'].value_counts(), bins=50, edgecolor='black')
plt.xlabel("Article Count")
plt.ylabel("User count")

plt.show()
```



In [59]:
```
# What are the descriptive statistics of the number of articles a user interacts wi
df.describe()
```

Out[59]:

	Unnamed: 0	article_id
count	45993.000000	45993.000000
mean	23362.180745	908.846477
std	13717.943019	486.647866
min	0.000000	0.000000
25%	11498.000000	460.000000
50%	22996.000000	1151.000000
75%	35207.000000	1336.000000
max	47581.000000	1444.000000

In [60]:
```
df.describe(include='object')
```

Out[60]:

	title	email
count	45993	45993
unique	714	5149
top	use deep learning for image classification	2b6c0f514c2f2b04ad3c4583407dccd0810469ee
freq	937	364

This shows us of the ~46k rows, there are ~5k unique users (assuming each email is a unique user). Or on average a user interacts with 8.9 articles (assuming that they interact with each article only once)

From this we see the actual counts of user to article counts. And we see how quickly it drops from the most active users at 364 or 363 articles down to less than half that at 170 articles.

In [61]:
```
df['email'].value_counts()
```

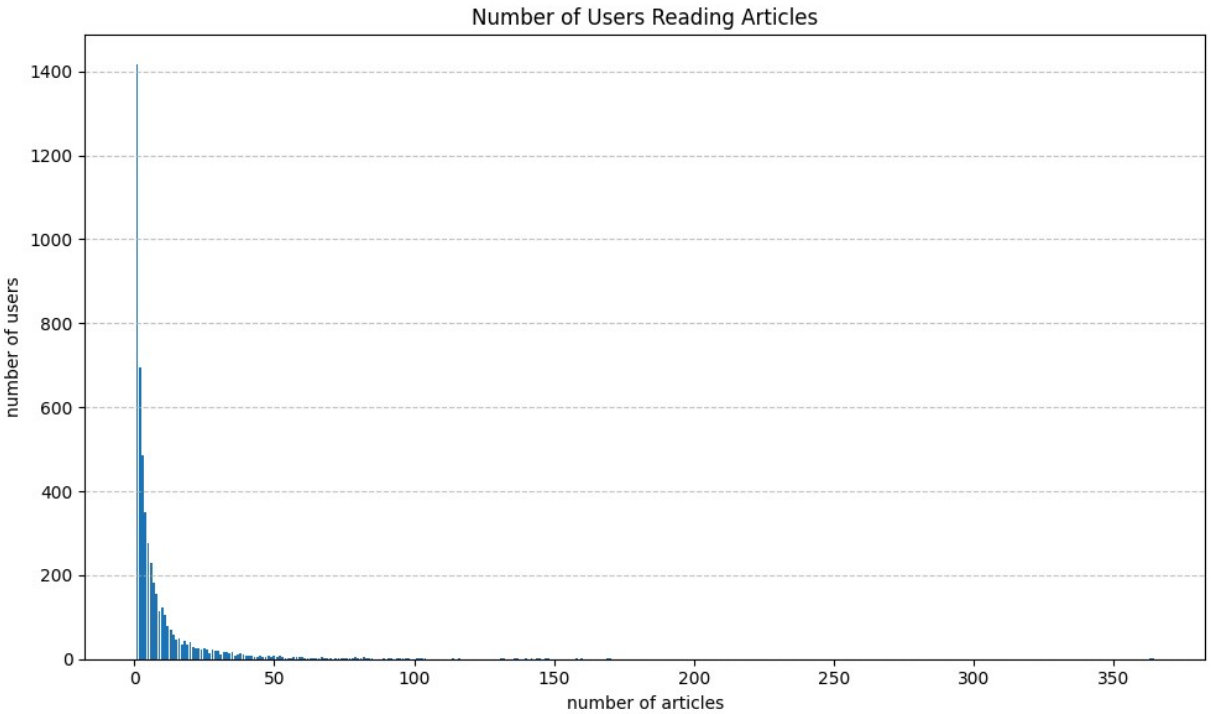
Out[61]:
```
email
2b6c0f514c2f2b04ad3c4583407dccd0810469ee    364
77959baaa9895a7e2bdc9297f8b27c1b6f2cb52a    363
2f5c7feae533ce046f2cb16fb3a29fe00528ed66    170
a37adec71b667b297ed2440a9ff7dad427c7ac85    169
8510a5010a5d4c89f5b07baac6de80cd12cfaf93    160
                                           ...
2cc5d8687503301ff6c01687dd695df0c0817b05      1
8c42f13e549b6223e5e5b6387eab12366f10994b      1
30b880bc19a812122e4905ff8bc896580187bc1c      1
bd7f37b6d6bd5eca7c2823b97dce1acc968672e8      1
488f60c5d90235787048281672d3a023bb809396      1
Name: count, Length: 5149, dtype: int64
```

I was surprised to not see 'unknown_user' at the top of the list. So looking at that email specifically, we only see 17 articles. A much smaller amount of 'missing' data than I expected.

In [12]:
```
distribution = df['email'].value_counts()
print(distribution['unknown_user'])

print( df[df['email'] == 'unknown_user'].shape )
print(df[df['email'] == 'unknown_user'].head())
```

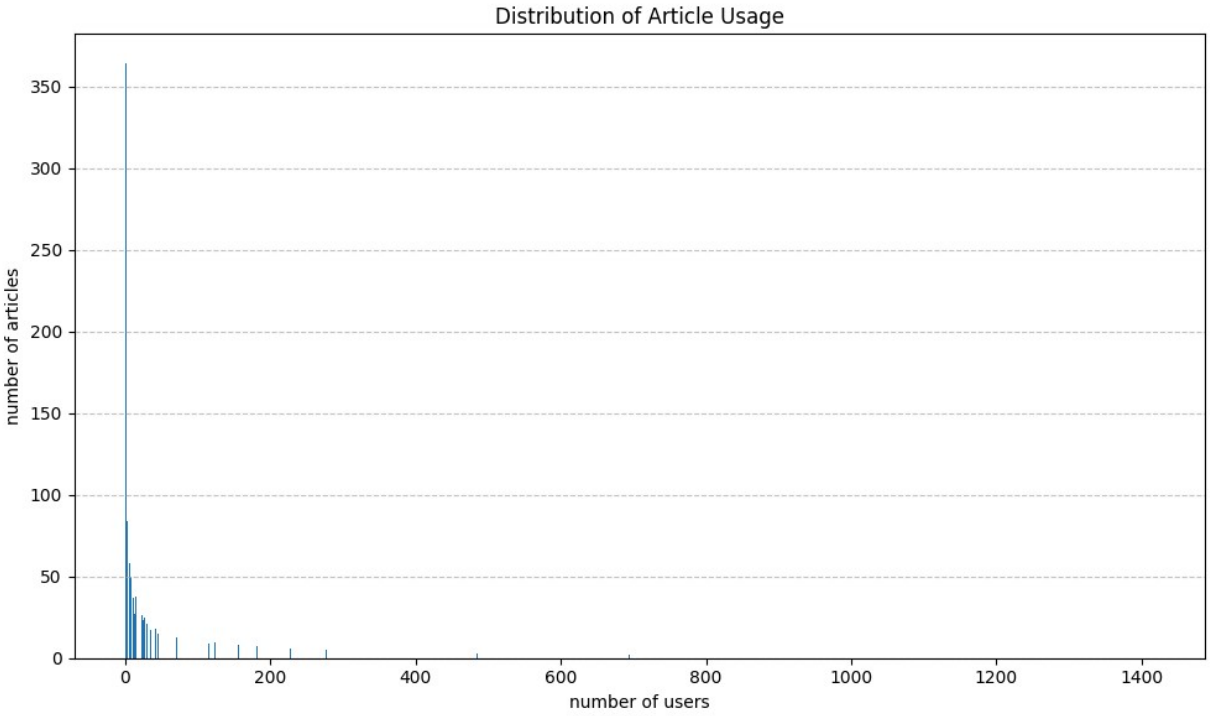
```
17
(17, 4)
       Unnamed: 0  article_id  \
25131       25146        1016
29758       30157        1393
29759       30158          20
29760       30159        1174
29761       30160          62

                                                   title         email
25131  why you should master r (even if it might even...  unknown_user
29758                        the nurse assignment problem  unknown_user
29759  working interactively with rstudio and noteboo...  unknown_user
29760      breast cancer wisconsin (diagnostic) data set  unknown_user
29761  data visualization: the importance of excludin...  unknown_user
```

In [14]:
```
# Create a plot of the number of articles read by each user
user_counts = df['email'].value_counts()
# Count how many users have each count
count_of_counts = user_counts.value_counts().sort_index()

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(count_of_counts.index, count_of_counts.values)
plt.xlabel('number of articles')
plt.ylabel('number of users')
plt.title('Number of Users Reading Articles')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



In [15]:
```
# Create a plot of the number of times each article was read
user_counts = df['email'].value_counts()
# Count how many users have each count
```

```
count_of_counts = user_counts.value_counts().sort_index()

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(count_of_counts.values, count_of_counts.index)
plt.ylabel('number of articles')
plt.xlabel('number of users')
plt.title('Distribution of Article Usage')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



In [16]:
```
# Fill in the median and maximum number of user_article interactions below

median_val =  float(user_counts.median()) # 50% of individuals interact with ____ n
print(f" # 50% of individuals interact with {median_val} number of articles or fewe
max_views_by_user = int(user_counts.max()) # The maximum number of user-article int
print(f"The maximum number of user-article interactions by any 1 user is {max_views
```

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

`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 [15]:
```
df.head()
```

Out[15]:

	Unnamed: 0	article_id	title	email
0	0	1430	using pixiedust for fast, flexible, and easier...	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1	1	1314	healthcare python streaming application demo	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2	2	1429	use deep learning for image classification	b96a4f2e92d8572034b1e9b28f9ac673765cd074
3	3	1338	ml optimization using cognitive assistant	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4	4	1276	deploy your python model as a restful api	f01220c46fc92c6e6b161b1849de11faacd7ccb2

In [17]:
```
unique_articles = df[df['email'] != 'unknown_user']['article_id'].drop_duplicates()
total_articles = df['article_id'].drop_duplicates().shape[0]    # The number of uni
unique_users = df['email'].drop_duplicates().shape[0]     # The number of unique us
user_article_interactions =   df.shape[0]  # The number of user-article interaction

print( df[df['email'] != 'unknown_user'][['article_id', 'email']].drop_duplicates()
print( df[['article_id', 'email']].drop_duplicates().shape[0]  )
print( df[df['email'] != 'unknown_user'][['article_id', 'email']].shape[0]  )

df.shape[0]
#   df[['article_id', 'email']].drop_duplicates().shape[0]
```

```
33669
33682
45976
```

Out[17]: 45993

`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 [41]:
```
df.head()
```

Out[41]:

	Unnamed: 0	article_id	title	email
0	0	1430	using pixiedust for fast, flexible, and easier...	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1	1	1314	healthcare python streaming application demo	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2	2	1429	use deep learning for image classification	b96a4f2e92d8572034b1e9b28f9ac673765cd074
3	3	1338	ml optimization using cognitive assistant	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4	4	1276	deploy your python model as a restful api	f01220c46fc92c6e6b161b1849de11faacd7ccb2

In [18]:
```
most_viewed_article_id = int(df[['article_id']].value_counts().reset_index().iloc[0
max_views = int(df[['article_id']].value_counts().reset_index().iloc[0, 1]) # The m
```

In [19]:
```
## No need to change the code here - this will be helpful for later parts of the no
# Run this cell to map the user email to a user_id column and remove the email colu

def email_mapper(df=df):
    coded_dict = {
        email: num
        for num, email in enumerate(df['email'].unique(), start=1)
    }
    return [coded_dict[val] for val in df['email']]

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

# show header
df.head()
```

Out[19]:

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

In [20]:
```
## 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 ______.``: us
    ``The maximum number of user-article interactions by any 1 user is ______.``: m
    ``The most viewed article in the dataset was viewed _____ times.``: max_views,
    ``The article_id of the most viewed article is ______.``: most_viewed_article_i
    ``The number of unique articles that have at least 1 rating ______.``: unique_a
    ``The number of unique users in the dataset is ______``: unique_users,
    ``The number of unique articles on the IBM platform``: total_articles
}

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

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

In [21]:
```
import copy
df_orig = copy.deepcopy(df)
```

Part II: Rank-Based Recommendations

In this project, 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 [79]:
```
df.head()
```

Out[79]:

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

In [22]:
```
'''
max_views = int(df[['article_id']].value_counts().reset_index().iloc[0, 1])
'''


ranked_df_article = df['title'].value_counts().reset_index()
list(ranked_df_article['title'])[:5]
```

Out[22]:
```
['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']
```

In [23]:
```
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

    """
    ranked_df_article = df['title'].value_counts().reset_index()
    top_articles = list(ranked_df_article['title'])[:n]


    return top_articles # Return the top article titles from df

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

    """
    ranked_df_id = df['article_id'].value_counts().reset_index()
    top_articles = list(ranked_df_id['article_id'])[:n]


    return top_articles # Return the top article ids
```

In [84]:
```
print(get_top_articles(10))
print(get_top_article_ids(10))
```

```
['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', 'hea
lthcare python streaming application demo', 'finding optimal locations of new store
using decision optimization', 'apache spark lab, part 1: basic concepts', 'analyze e
nergy consumption in buildings', 'gosales transactions for logistic regression mode
l']
[1429, 1330, 1431, 1427, 1364, 1314, 1293, 1170, 1162, 1304]
```

In [13]:
```
# Test your function by returning the top 5, 10, and 20 articles
top_5 = get_top_articles(5)
top_10 = get_top_articles(10)
top_20 = get_top_articles(20)
```

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

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

Part III: User-User Based Collaborative Filtering

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

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

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

In [24]:
```
import copy
df_orig = copy.deepcopy(df)
df.head()
```

Out[24]:

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

In [25]:
```
# create the user-article matrix with 1's and 0's

def create_user_item_matrix(df, fill_value=0):
    """
    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 val
    an article and a 0 otherwise
    """
    # Fill in the function here
    user_item = df[['user_id', 'article_id', 'Unnamed: 0']].groupby(['user_id', 'ar
    user_item = user_item.where(user_item.isna(),1)
    user_item = user_item.fillna(fill_value)

    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [16]:
```
user_item
```

Out[16]:

article_id	0	2	4	8	9	12	14	15	16	18	...	1434	1435	1436	1437	1439
user_id																
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	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	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	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	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	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	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	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	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	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	0.0	...	0.0	0.0	0.0	0.0	0.0

5149 rows × 714 columns

In [17]:
```
## 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
assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-articl
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user
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 [26]:
```
# Lets use the cosine_similarity function from sklearn
from sklearn.metrics.pairwise import cosine_similarity
```

In [19]:
```
user_item.index.get_loc(5)
```

Out[19]: 4

In [27]:
```
def find_similar_users(user_id, user_item=user_item, include_similarity=False):
    """
    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
    include_similarity - (bool) whether to include the similarity in the output

    OUTPUT:
    similar_users - (list) an ordered list where the closest users (largest dot pro
                    are listed first

    Description:
    Computes the similarity of every pair of users based on the dot product
    Returns an ordered list of user ids. If include_similarity is True, returns a l
    where the first element is the user id and the second the similarity.

    """

    # compute similarity of each user to the provided user
    user_index = user_item.index.get_loc(user_id)
    similarity_matrix = cosine_similarity(user_item, user_item)
    scores = list(enumerate(similarity_matrix[user_index]))


    # sort by similarity
    scores = sorted(scores, key = lambda x: x[1], reverse=True)
    # remove the own user's id

    scores_list = []
    for x in scores:
        if x[0] == user_index:
            continue
        else:
            scores_list.append(x)
    # create list of just the ids
    user_ids = [i[0] for i in scores_list]

    # create list of just the similarities
    sim_value_list = [i[1] for i in scores_list]
    most_similar_users = user_item.index[user_ids]

    if include_similarity:
        return [[a, b] for a, b in zip(most_similar_users, sim_value_list)]     # r
```

```
    return list(most_similar_users) # return a list of the users in order from most
```

In [40]:
```
find_similar_users(1, include_similarity = True)[:10]
```

Out[40]:
```
[[3933, np.float64(0.9860132971832691)],
 [46, np.float64(0.34258007985157446)],
 [4201, np.float64(0.34258007985157446)],
 [253, np.float64(0.3333333333333333)],
 [824, np.float64(0.3333333333333333)],
 [5034, np.float64(0.3333333333333333)],
 [5041, np.float64(0.30151134457776363)],
 [136, np.float64(0.2981423969999719)],
 [2305, np.float64(0.2981423969999719)],
 [395, np.float64(0.29488391230979427)]]
```

In [41]:
```
# Do a spot check of your function
print("The 10 most similar users to user 1 are: {}".format(find_similar_users(1)[:1
print("The 5 most similar users to user 3933 are: {}".format(find_similar_users(393
print("The 3 most similar users to user 46 are: {}".format(find_similar_users(46)[:
```

```
The 10 most similar users to user 1 are: [3933, 46, 4201, 253, 824, 5034, 5041, 136,
2305, 395]
The 5 most similar users to user 3933 are: [1, 46, 4201, 253, 824]
The 3 most similar users to user 46 are: [4201, 790, 5077]
```

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 [28]:
```
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 arti
                    (this is identified by the title column in df)
    """
    article_df = df[df['article_id'].isin(article_ids)][['article_id', 'title']].dr
    article_names = list( article_df['title'] )

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

def get_ranked_article_unique_counts(article_ids, 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_counts - (list) a list of tuples with article_id and number of
                     unique users that have interacted with the article, sorted
```

```
                    by the number of unique users in descending order

    Description:
    Provides a list of the article_ids and the number of unique users that have
    interacted with the article using the user_item matrix, sorted by the number
    of unique users in descending order
    """
    # Your code here

    article_counts = user_item[article_ids].sum(axis=0)
    article_counts = article_counts.reset_index(drop = False)
    ranked_article_unique_counts = article_counts.values.tolist()

    return ranked_article_unique_counts


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 arti
                    (this is identified by the title column in df)

    Description:
    Provides a list of the article_ids and article titles that have been seen by a
    """
    # Your code here
    user_dataset = user_item.loc[user_id]
    article_ids = list(user_dataset[user_dataset==1].index)
    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
    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
    user_viewed_list = get_user_articles(user_id)[0]
    user_recommend_list = []


    similar_users = find_similar_users(user_id)
    #similar_users
    for similar_iteration in similar_users:
        similar_user_articles = get_user_articles(similar_iteration)[0]
        similar_user_articles = [x for x in similar_user_articles if x not in user_
        user_recommend_list.extend(similar_user_articles)
        user_viewed_list.extend(similar_user_articles)
        if len(user_recommend_list) >= m:
            break

    recs = user_recommend_list[:m]


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

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

Out[119…
```
['leverage python, scikit, and text classification for behavioral profiling',
 'challenges in deep learning',
 'awesome deep learning papers',
 'this week in data science (april 18, 2017)',
 'improving the roi of big data and analytics through leveraging new sources of da
ta',
 'this week in data science (may 2, 2017)',
 'using apply, sapply, lapply in r',
 'how can data scientists collaborate to build better business',
 'top 20 r machine learning and data science packages',
 'do i need to learn r?']
```

In [120…
```
get_ranked_article_unique_counts([1320, 232, 844])
```

Out[120…
```
[[1320.0, 123.0], [232.0, 62.0], [844.0, 78.0]]
```

In [121…
```
# Test your functions here - No need to change this code - just run this cell
assert set(get_article_names([1024, 1176, 1305, 1314, 1422, 1427])) == set(['using
assert set(get_article_names([1320, 232, 844])) == set(['housing (2015): united sta
assert set(get_user_articles(20)[0]) == set([1320, 232, 844])
assert set(get_user_articles(20)[1]) == set(['housing (2015): united states demogra
assert set(get_user_articles(2)[0]) == set([1024, 1176, 1305, 1314, 1422, 1427])
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct hig
assert get_ranked_article_unique_counts([1320, 232, 844])[0] == [1320, 123], "Oops!
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 [122…
```
print(get_top_articles(10))
#print(get_top_article_ids(10))
```

```
['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', 'hea
lthcare python streaming application demo', 'finding optimal locations of new store
using decision optimization', 'apache spark lab, part 1: basic concepts', 'analyze e
nergy consumption in buildings', 'gosales transactions for logistic regression mode
l']
```

In [29]:
```
def get_top_sorted_users(user_id, user_item=user_item):
    """
    INPUT:
    user_id - (int)
    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 prov
                    num_interactions - the number of articles viewed by the user

    Other Details - sort the neighbors_df by the similarity and then by number of i
                    highest of each is higher in the dataframe, i.e. Descending ord

    """
    # Your code here
    # Hint: find similar users, but include their similarity, from there we can fin


    neighbors_df = find_similar_users(user_id, user_item=user_item, include_similar
    neighbors_df = pd.DataFrame(neighbors_df, columns=['neighbor_id', 'similarity']
    interaction_list = []
    for neighbor_id_iter, similarity_value_iter in neighbors_df.values.tolist():
        interaction_list.append(len(get_user_articles(neighbor_id_iter)[0]) )
    neighbors_df['num_interactions'] = interaction_list
    neighbors_df.sort_values(by=['similarity', 'num_interactions'], ascending=[Fals
```

```
    neighbors_df.reset_index(drop=True, inplace=True)

    return neighbors_df # Return the dataframe specified in the doc_string


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

    OUTPUT:
    recs - (list) a list of recommendations for the user by article id
    rec_names - (list) a list of recommendations for the user by article 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
    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
    user_viewed_list = get_user_articles(user_id)[0]
    user_recommend_list = []

    similar_users = get_top_sorted_users(user_id)
    similar_users = list(similar_users['neighbor_id'])

    for similar_iteration in similar_users:
        similar_user_articles = get_user_articles(similar_iteration)[0]
        similar_user_articles = [x for x in similar_user_articles if x not in user_
        user_recommend_list.extend(similar_user_articles)
        user_viewed_list.extend(similar_user_articles)
        if len(user_recommend_list) >= m:
            break

    recs = user_recommend_list[:m]

    return recs, get_article_names(recs) # return your recommendations for this use
```

In [163…

```
# 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:
[1162, 1165, 1185, 1293, 254, 40, 1271, 1328, 1402, 1410]

The top 10 recommendations for user 20 are the following article names:
['classify tumors with machine learning', 'analyze energy consumption in buildings',
'uci: sms spam collection', 'income (2015): united states demographic measures', 'an
alyze precipitation data', 'finding optimal locations of new store using decision op
timization', 'uci: adult - predict income', 'customer demographics and sales', 'ense
mble learning to improve machine learning results', 'apple, ibm add machine learning
to partnership with watson-core ml coupling']
```

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 [158…
```
print(get_top_sorted_users(1, user_item=user_item).head(n=1))
print(get_top_sorted_users(2, user_item=user_item).head(n=10))
print(get_top_sorted_users(131, user_item=user_item).head(n=10))
```

```
   neighbor_id  similarity  num_interactions
0         3933    0.986013                35
   neighbor_id  similarity  num_interactions
0         5083    0.730297                 5
1         1552    0.577350                 2
2         1890    0.577350                 2
3         1372    0.471405                 3
4         2941    0.433013                 8
5         3586    0.408248                 4
6          331    0.408248                 1
7          348    0.408248                 1
8          378    0.408248                 1
9          496    0.408248                 1
   neighbor_id  similarity  num_interactions
0         3870    0.986667                75
1          203    0.388909                96
2         4459    0.388909                96
3         3782    0.387585               135
4           40    0.384308                52
5         4932    0.384308                52
6           23    0.377647               135
7          242    0.375823                59
8         3910    0.372678                60
9          383    0.367423                32
```

In [151…
```
### Tests with a dictionary of results
user1_most_sim =  3933 # Find the user that is most similar to user 1
user2_6th_sim =  3586 # Find the 6th most similar user to user 2
user131_10th_sim =  383 # Find the 10th most similar user to user 131
```

In [152…
```
## Dictionary Test Here
sol_5_dict = {
    'The user that is most similar to user 1.': user1_most_sim,
    'The user that is the 6th most similar to user 2.': user2_6th_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.

Answer:

Top articles as no user history, only can use user-user history till they start having user-item iteractions

For a new user, we'd have to use get_top_articles() for recommendations. Since they have no history we can't compare them to any other 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 [30]:
```
# What would your recommendations be for this new user 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)   # Your recommendations here
```

In [31]:
```
assert set(new_user_recs) == {1314, 1429, 1293, 1427, 1162, 1364, 1304, 1170, 1431,

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

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

Part IV: Content Based Recommendations

Another method we might use to make recommendations is to recommend similar articles that are possibly related. One way we can find article relationships is by clustering text about those articles. Let's consider content to be the article **title**, as it is the only text we have available. One point to highlight, there isn't one way to create a content based recommendation, especially considering that text information can be processed in many ways.

`1.` Use the function bodies below to create a content based recommender function `make_content_recs`. We'll use TF-IDF to create a matrix based off article titles, and use this matrix to create clusters of related articles. You can use this function to make recommendations of new articles.

In [170…
```
df.head()
```

Out[170…

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

In [32]:
```
df_orig = copy.deepcopy(df)
```

In [33]:
```
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn.decomposition import TruncatedSVD
```

In [34]:
```
# unique articles
df_unique_articles = df[ 'title'].drop_duplicates()
#df_unique_articles = df[['article_id', 'title']].drop_duplicates()
#df_unique_articles.set_index('article_id', inplace=True)
#df_unique_articles.index.name = None
```

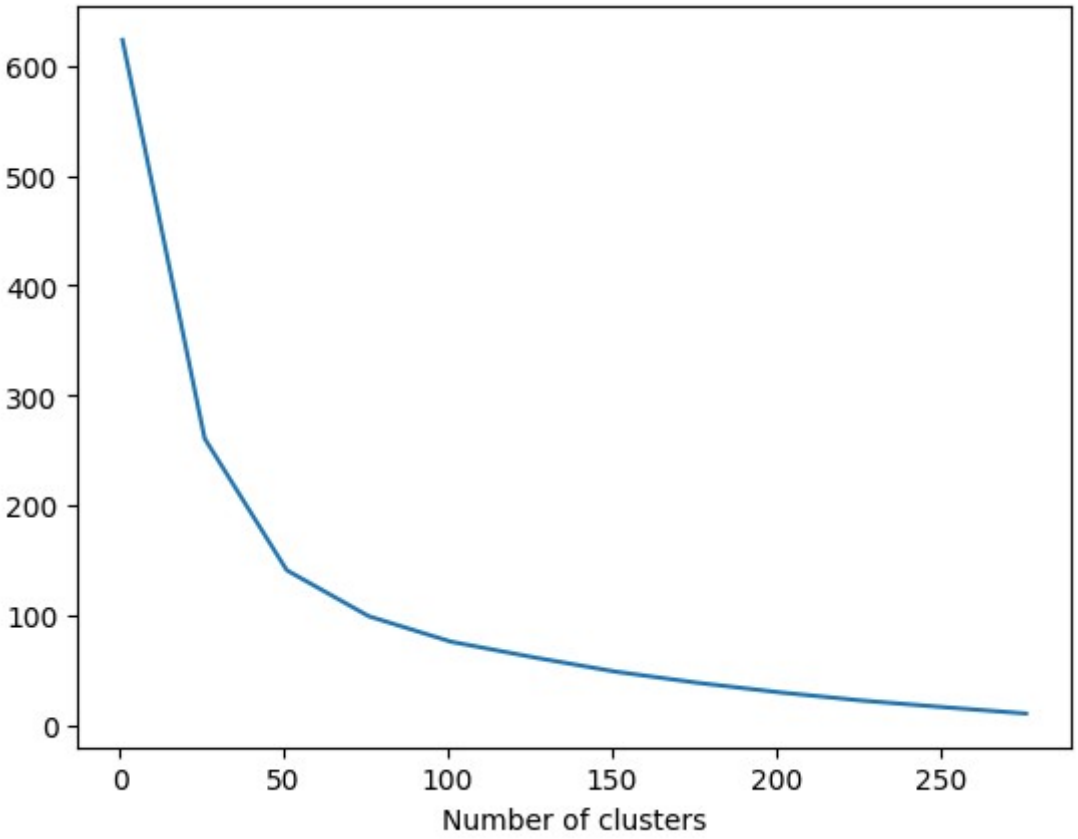
In [35]:
```
# Create a vectorizer using TfidfVectorizer and fit it to the article titles
max_features = 200
max_df = 0.75
min_df = 5

vectorizer = TfidfVectorizer(
    max_df=max_df,
    min_df=min_df,
    stop_words="english",
    max_features=max_features,
)
print("Running TF-IDF")
X_tfidf = vectorizer.fit_transform(df_unique_articles)  # Fit the vectorizer to the

print(f"n_samples: {X_tfidf.shape[0]}, n_features: {X_tfidf.shape[1]}")

lsa = make_pipeline(TruncatedSVD(n_components=50), Normalizer(copy=False))
X_lsa = lsa.fit_transform(X_tfidf)  # Fit the LSA model to the vectorized article t
explained_variance = lsa[0].explained_variance_ratio_.sum()

print(f"Explained variance of the SVD step: {explained_variance * 100:.1f}%")
```

```
Running TF-IDF
n_samples: 714, n_features: 125
Explained variance of the SVD step: 76.0%
```

In [36]:
```
# Let's map the inertia for different number of clusters to find the optimal number
# We'll plot it to see the elbow
inertia = []
```

```
clusters = 300
step = 25
max_iter = 50
n_init = 5
random_state = 42
for k in range(1, clusters, step):
    kmeans = KMeans(
        n_clusters=k,
        max_iter=max_iter,
        n_init=n_init,
        random_state=random_state,
    ).fit(X_lsa)
    # inertia is the sum of squared distances to the closest cluster center
    inertia.append(kmeans.inertia_)
plt.plot(range(1, clusters, step), inertia)
plt.xlabel('Number of clusters')
```

Out[36]: Text(0.5, 0, 'Number of clusters')



There appears to be an elbow about 50, so we'll use 50 clusters.

In [37]:
```
n_clusters = 50 # Number of clusters
kmeans = KMeans(
    # Your code here, same as parameters above
    n_clusters=n_clusters,
    max_iter=max_iter,
    n_init=n_init,
    random_state=random_state
).fit(X_lsa)
```

In [192…
```
df_unique_articles
```

Out[192…
```
0        using pixiedust for fast, flexible, and easier...
1             healthcare python streaming application demo
2               use deep learning for image classification
3                ml optimization using cognitive assistant
4                deploy your python model as a restful api
                              ...
24278    airbnb data for analytics: washington d.c. cal...
24616              build a naive-bayes model with wml & dsx
24726              load and analyze public data sets in dsx
24737    the new builders ep. 13: all the data that’s f...
24827                               create a project in dsx
Name: title, Length: 714, dtype: object
```

In [38]:
```
article_temp_map = pd.DataFrame({'title':df_unique_articles, 'cluster_labels': kmea
df_articles = df[['article_id', 'title']].drop_duplicates()
```

In [39]:
```
article_temp_map = pd.merge(df_articles, article_temp_map, on = 'title', how = 'inn
article_temp_map.head()
```

Out[39]:

	article_id	title	cluster_labels
0	1430	using pixiedust for fast, flexible, and easier...	26
1	1314	healthcare python streaming application demo	29
2	1429	use deep learning for image classification	37
3	1338	ml optimization using cognitive assistant	26
4	1276	deploy your python model as a restful api	41

In [40]:
```
# create a new column `title_cluster` and assign it the kmeans cluster labels
# First we need to map the labels to df_unique_articles article ids and then apply
article_cluster_map = dict(zip(article_temp_map['article_id'], article_temp_map['cl
df['title_cluster'] = df['article_id'].map(article_cluster_map)   # apply map to cr
```

In [241…
```
df.head()
```

Out[241…

	Unnamed: 0	article_id	title	user_id	title_cluster
0	0	1430	using pixiedust for fast, flexible, and easier...	1	33
1	1	1314	healthcare python streaming application demo	2	42
2	2	1429	use deep learning for image classification	3	15
3	3	1338	ml optimization using cognitive assistant	4	33
4	4	1276	deploy your python model as a restful api	5	22

In [ ]:

In [242…
```
# Let's check the number of articles in each cluster
np.array(np.unique(kmeans.labels_, return_counts=True)).T
```

Out[242…
```
array([[ 0, 23],
       [ 1, 10],
       [ 2, 59],
       [ 3, 18],
       [ 4, 29],
       [ 5,  9],
       [ 6, 25],
       [ 7, 37],
       [ 8, 28],
       [ 9, 20],
       [10,  8],
       [11,  7],
       [12, 33],
       [13, 15],
       [14, 21],
       [15, 19],
       [16, 24],
       [17, 10],
       [18, 13],
       [19, 10],
       [20, 10],
       [21,  5],
       [22, 13],
       [23,  8],
       [24, 12],
       [25,  8],
       [26, 20],
       [27,  9],
       [28,  9],
       [29,  8],
       [30,  8],
       [31,  7],
       [32, 11],
       [33, 15],
       [34, 11],
       [35, 12],
       [36,  9],
       [37, 10],
       [38,  7],
       [39, 10],
       [40, 16],
       [41,  7],
       [42,  8],
       [43,  5],
       [44,  9],
       [45, 19],
       [46, 10],
       [47,  3],
       [48,  7],
       [49, 10]])
```

In [46]:
```
def get_similar_articles(article_id, df=df):
    """
    INPUT:
    article_id - (int) an article id
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    article_ids - (list) a list of article ids that are in the same title cluster

    Description:
    Returns a list of the article ids that are in the same title cluster
    """
     # Your code here
    title_cluster = df[df['article_id'] == article_id]['title_cluster'].unique()[0]
    articles_in_cluster = df[df['title_cluster']== title_cluster]['article_id'].uni
    articles_in_cluster.remove(article_id)
    # remove the input article_id from the list

    return articles_in_cluster
```

In [63]:
```
def make_content_recs(article_id, n, df=df):
    """
    INPUT:
    article_id - (int) an article id
    n - (int) the number of recommendations you want similar to the article id
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    n_ranked_similar_articles - (list) a list of article ids that are in the same t
                                by popularity
    n_ranked_article_names - (list) a list of article names associated with the lis

    Description:
    Returns a list of the n most ranked similar articles to a given article_id base
    cluster in df. Rank similar articles using the function get_ranked_article_uniq
    """
    # Your code here
    results = get_similar_articles(article_id)
    results = pd.DataFrame(get_ranked_article_unique_counts( results ), columns=['a
    results = results.sort_values('popularity', ascending=False).reset_index(drop =
    n_ranked_similar_articles = list(results['article_id'][:n])
    n_ranked_article_names = get_article_names(n_ranked_similar_articles, df=df)

    return n_ranked_similar_articles, n_ranked_article_names
```

In [64]:
```
# Test out your content recommendations given artice_id 25
rec_article_ids, rec_article_titles = make_content_recs(25, 10)
print(rec_article_ids)
print(rec_article_titles)
```

```
[1025.0, 101.0, 975.0, 766.0, 508.0, 547.0, 132.0, 878.0, 92.0, 693.0]
['better together: spss and data science experience', 'data science in the cloud',
'collecting data science cheat sheets', 'making data science a team sport', 'how to
choose a project to practice data science', 'data tidying in data science experienc
e', 'trust in data science', 'the data science process', '10 data science podcasts y
ou need to be listening to right now', '9 mistakes to avoid when starting your caree
r in data science']
```

In [65]:
```
assert len({1025, 593, 349, 821, 464, 29, 1042, 693, 524, 352}.intersection(set(rec
```

`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? What other text data would be useful to help make better recommendations besides the article title?

This system clusters articles based on a tfidf vectorization of the article titles. So then when someone views an article, we look at all the other articles in that cluster, ranked by how many other users have read them.

Improvements: This is driven by an individual article, It would be better if it were driven by a collection of articles. Additional attributes beyond the tfidf would be beneficial. (Relative length, author, etc)

Part V: Matrix Factorization

In this part of the notebook, you will build use matrix factorization to make article recommendations to users.

`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 [67]:
```
df_orig = copy.deepcopy(df)
user_item_orig = copy.deepcopy(user_item)
```

In [68]:
```
# quick look at the matrix
user_item.head()
```

Out[68]:

article_id	0	2	4	8	9	12	14	15	16	18	...	1434	1435	1436	1437	1439
user_id																
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	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	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	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	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

5 rows × 714 columns

`2.` In this situation, you can use Singular Value Decomposition from scikit-learn on the user-item matrix. Use the cell to perform SVD.

In [69]:
```
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import precision_score, recall_score, accuracy_score
# Using the full number of components which equals the number of columns
svd = TruncatedSVD(n_components=len(user_item.columns), n_iter=5, random_state=42)

u = svd.fit_transform(user_item)
v = svd.components_
s = svd.singular_values_
print('u', u.shape)
print('s', s.shape)
print('vt', v.shape)
```

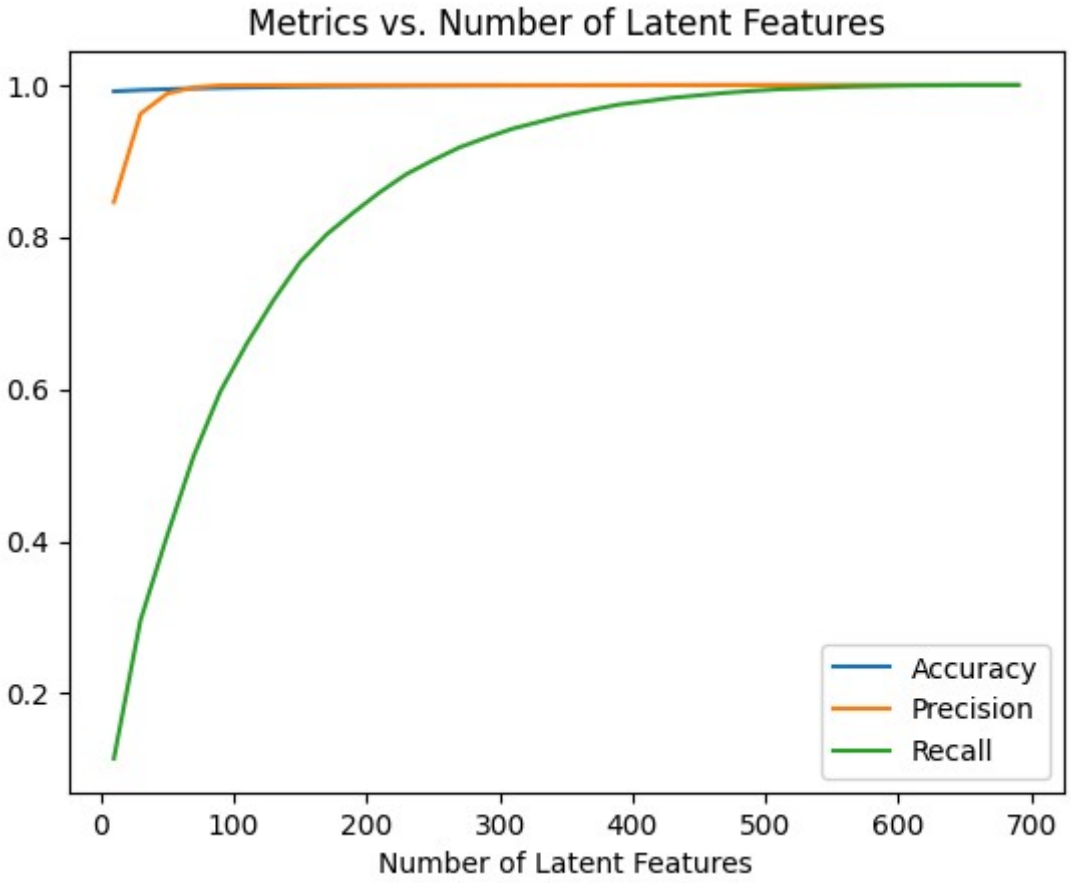
```
u (5149, 714)
s (714,)
vt (714, 714)
```

`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 better metrics when making predictions for the 1 and 0 values in the user-item matrix. Run the cell below to get an idea of how our metrics improve as we increase the number of latent features.

In [70]:
```
num_latent_feats = np.arange(10, 700+10, 20)
metric_scores = []

for k in num_latent_feats:
    # restructure with k latent features
    u_new, vt_new = u[:, :k], v[:k, :]

    # take dot product
    user_item_est = abs(np.around(np.dot(u_new, vt_new))).astype(int)
    # make sure the values are between 0 and 1
```

```
    user_item_est = np.clip(user_item_est, 0, 1)

    # total errors and keep track of them
    acc = accuracy_score(user_item.values.flatten(), user_item_est.flatten())
    precision = precision_score(user_item.values.flatten(), user_item_est.flatten()
    recall = recall_score(user_item.values.flatten(), user_item_est.flatten())
    metric_scores.append([acc, precision, recall])


plt.plot(num_latent_feats, metric_scores, label=['Accuracy', 'Precision', 'Recall']
plt.legend()
plt.xlabel('Number of Latent Features')
plt.title('Metrics vs. Number of Latent Features')
```

Out[70]: Text(0.5, 1.0, 'Metrics 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. Given the plot above, what would you pick for the number of latent features and why?

Accuracy and Precision both hit 100% rather quickly, around 50 features. And Recall maxes out around 500 Features. This indicates that as we increase the features, we improve our False Negative metric. If our run time requirements can support it, I'd look at 500 features. If our computational resources can't handle it, around 200 Features gives us a good balance of Accuracy, Precision, and Recall.

`5.` Using 200 latent features and the values of U, S, and V transpose we calculated above, create an article id recommendation function that finds similar article ids to the one provide.

Create a list of 10 recommendations that are similar to article with id 4. The function should provide these recommendations by finding articles that have the most similar latent features as the provided article.

In [146…
```
def get_svd_similar_article_ids(article_id, vt, user_item=user_item, include_simila
    """
    INPUT:
    article_id - (int) an article id
    vt - (numpy array) vt matrix from SVD
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise
    include_similarity - (bool) whether to include the similarity in the output

    OUTPUT:
    article_ids - (list) a list of article ids that are in the same title cluster

    Description:
    Returns a list of the article ids similar using SVD factorization
    """
    # Find the index of the article_id
    article_idx = list( user_item.columns.get_indexer([article_id]) )[0].item()
    # Find the cosine similarity of all articles
    # Hint: vt should be transposed before passing to cosine_similarity to get a 71
    cos_sim = cosine_similarity(vt.T)
    # Get similarities only for the cos_sim of the article_idx
    article_sim = cos_sim[article_idx]
    # Sort and return the articles, don't include the own article
    article_sim = cos_sim[article_idx]
    article_sim_df = pd.DataFrame({'article_sim': list(article_sim), 'article_id':
    article_sim_df.sort_values('article_sim', ascending=False, inplace=True)
    article_sim_df.drop(article_idx, inplace=True)
    article_sim_df.reset_index(drop=True, inplace=True)
    most_similar_items = list(article_sim_df['article_id'])

    if include_similarity:
        return  article_sim_df.values.tolist()  # return a list of lists with [[sim
    return most_similar_items
```

In [98]:
```
# Create a vt_new matrix with 200 latent features
k = 200

vt_new = v[:k, :]
vt_new
```

Out[98]:
```
array([[ 0.00221822,  0.02604036,  0.00246327, ...,  0.00050441,
         0.00492688,  0.00404343],
       [-0.00207737,  0.0281658 , -0.00051368, ..., -0.00262161,
        -0.01325774, -0.01327298],
       [ 0.00043211, -0.0135172 , -0.00300468, ..., -0.00157638,
        -0.01481189, -0.00798957],
       ...,
       [ 0.00052298,  0.03114128,  0.0412575 , ..., -0.00481181,
         0.02392748,  0.01149674],
       [ 0.01517699,  0.03271468, -0.00647556, ..., -0.00124832,
        -0.0036916 , -0.02407027],
       [ 0.01029133,  0.00532077, -0.01000918, ..., -0.00263044,
         0.04288296,  0.02769194]], shape=(200, 714))
```

In [135…
```
# What is the article name for article_id 4?
print("Current article:", get_article_names([4], df=df)[0])
```

```
Current article: analyze ny restaurant data using spark in dsx
```

In [147…
```
# What are the top 10 most similar articles to article_id 4?
rec_articles = get_svd_similar_article_ids(4, vt_new, user_item=user_item)[:10]
rec_articles
```

Out[147…
```
[1199, 1068, 486, 1202, 176, 1120, 244, 793, 58, 132]
```

In [149…
```
# What are the top 10 most similar articles to article_id 4?
get_article_names(rec_articles, df=df)
```

Out[149…
```
['use spark r to load and analyze data',
 'collecting data science cheat sheets',
 '10 powerful features on watson data platform, no coding necessary',
 'notebooks: a power tool for data scientists',
 'country statistics: crude oil - proved reserves',
 'country statistics: crude oil - exports',
 'top analytics tools in 2016',
 'advancements in the spark community',
 'airbnb data for analytics: paris calendar',
 'airbnb data for analytics: athens reviews']
```

In [148…
```
assert set(rec_articles) == {1199, 1068, 486, 1202, 176, 1120, 244, 793, 58, 132},
print("That's right!  Great job!")
```

```
That's right!  Great job!
```

`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 above are an improvement to how users currently find articles, either by Sections 2, 3, or 4? Add any tradeoffs between each of the methods, and how you could leverage each type for different situations including new users with no history, recently new users with little history, and users with a lot of history.

For how to best compare the quality of the different models, we would use a train/test style approach. We would identify 'test' users. For them we would only see 70% of their articles

read for training. Next we'd train each type of model and see the level of overlap between what each model recommends and what the 30% test articles are.

Tradeoffs:

Rank based recommenders

are the most simple, and can have the shortest predict time, which could be important is certain situations. They also benefit from not being affected by cold starts (no history). That said, they are likely the least accurate fo the systems.

Collaborative recommenders

Are also relatively straightforward, they don't involve involve much feature engineering. They do suffer from cold starts, as if you don't have any history, we can't say which other users you're like

Content recommenders

These are less straightforward. In the example we ran, we had to create some features based on text and then cluster. In other scenareos you may have to identify other features to use (author, publisher, topic, language, etc). These have less of a cold start issue, as we're largely looking at the characteristics of the item you're currently viewing.

Matrix Factorization

These are the least straightforward of the systems. If we were looking at sorting methods,

- rank based would be bubblesort
- collaborative and content recommenders would be binary and merge sort
- Matrix Factorization would be quick sort. Something not easily intuited.

This is a blind version of collaborative and content recommenders, where blind features are created based on the rankings/interactions of other users,
And those other users are also compared to eachother when producing those features.
Of the methods considered, it's the most impacted by cold start (as we have to worry about the user's cold start, as well as if we have enough interactions from other users to generate the features.)

Extras

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

Conclusion

> Congratulations! You have reached the end of the Recommendation Systems project!

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

Directions to Submit

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

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

> Once you've done this, you can submit your project by clicking on the "Submit Project" button in the lower right here. This will create and submit a zip file with this .ipynb doc and the .html or .pdf version you created. Congratulations!

In [150…
```
#from subprocess import call
#call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

Out[150… 1

In [ ]:
```
```