Recommendations with IBM

In this notebook, I will be putting my recommendation skills to use on real data from the IBM Watson Studio platform. I will analyse the interactions that users have with articles on the IBM Watson Studio platofrm and make recommendations to them about new articles I think they will like.

By following the table of contents, I will build out a number of different methods for making recommendations that can be used for different situations. I will first explore the data i'm working with and then first do rank based recommendations. To improve upon this I will then use user-user based collaborative filtering. Finally, I will complete a machine learning appraoch to building recommendations by using the user-item interactions I will build out a matrix decomposition.

Table of Contents

I. Exploratory Data Analysis
II. Rank Based Recommendations
III. User-User Based Collaborative Filtering
IV. Matrix Factorization
V. Extras & Concluding

Let's get started by importing the necessary libraries and reading in the data.

In [1]:
```
# import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
import project_tests as t
import pickle
import seaborn as sns
sns.set()
%matplotlib inline

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

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

Out[1]:

	article_id	title	email
0	1430.0	using pixiedust for fast, flexible, and easier...	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1	1314.0	healthcare python streaming application demo	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2	1429.0	use deep learning for image classification	b96a4f2e92d8572034b1e9b28f9ac673765cd074
3	1338.0	ml optimization using cognitive assistant	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4	1276.0	deploy your python model as a restful api	f01220c46fc92c6e6b161b1849de11faacd7ccb2

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

Out[2]:

	doc_body	doc_description	doc_full_name	doc_status	article_id
0	Skip navigation Sign in SearchLoading...\r\n\r...	Detect bad readings in real time using Python ...	Detect Malfunctioning IoT Sensors with Streami...	Live	0
1	No Free Hunch Navigation * kaggle.com\r\n\r\n ...	See the forest, see the trees. Here lies the c...	Communicating data science: A guide to present...	Live	1
2	☰ * Login\r\n * Sign Up\r\n\r\n * Learning Pat...	Here's this week's news in Data Science and Bi...	This Week in Data Science (April 18, 2017)	Live	2
3	DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...	Learn how distributed DBs solve the problem of...	DataLayer Conference: Boost the performance of...	Live	3
4	Skip navigation Sign in SearchLoading...\r\n\r...	This video demonstrates the power of IBM DataS...	Analyze NY Restaurant data using Spark in DSX	Live	4

In [3]:
```
df_content.shape
```
Out[3]:
```
(1056, 5)
```

In [4]:
```
df_content.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1056 entries, 0 to 1055
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   doc_body         1042 non-null   object
 1   doc_description  1053 non-null   object
 2   doc_full_name    1056 non-null   object
 3   doc_status       1056 non-null   object
 4   article_id       1056 non-null   int64 
dtypes: int64(1), object(4)
memory usage: 41.4+ KB
```

In [5]:
```
df_content['article_id'].nunique()
```
Out[5]:
```
1051
```

In [6]:
```
df.shape
```
Out[6]:
```
(45993, 3)
```

Part I : Exploratory Data Analysis

In this section of the notebook I will conduct exploratory data analysis.

`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 [7]:
```
# find number of columns and rows
df.shape
```
Out[7]:
```
(45993, 3)
```

In [8]:
```
df.dtypes
```
Out[8]:
```
article_id    float64
title          object
```

```
email           object
dtype: object
```

In [9]:
```
# Show df to get an idea of the data
df.head()
```

Out[9]:

	article_id	title	email
0	1430.0	using pixiedust for fast, flexible, and easier...	ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1	1314.0	healthcare python streaming application demo	083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2	1429.0	use deep learning for image classification	b96a4f2e92d8572034b1e9b28f9ac673765cd074
3	1338.0	ml optimization using cognitive assistant	06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4	1276.0	deploy your python model as a restful api	f01220c46fc92c6e6b161b1849de11faacd7ccb2

In [10]:
```
# df_content['article_id'].isin([1024, 1176, 1305, 1314, 1422, 1427]).sum()
set(df[df['article_id'].isin([1024, 1176, 1305, 1314, 1422, 1427])]['title'])
```

Out[10]:
```
{'build a python app on the streaming analytics service',
 'gosales transactions for naive bayes model',
 'healthcare python streaming application demo',
 'use r dataframes & ibm watson natural language understanding',
 'use xgboost, scikit-learn & ibm watson machine learning apis',
 'using deep learning to reconstruct high-resolution audio'}
```

In [11]:
```
# Show df_content to get an idea of the data
df_content.head()
```

Out[11]:

	doc_body	doc_description	doc_full_name	doc_status	article_id
0	Skip navigation Sign in SearchLoading...\r\n\r...	Detect bad readings in real time using Python ...	Detect Malfunctioning IoT Sensors with Streami...	Live	0
1	No Free Hunch Navigation * kaggle.com\r\n\r\n ...	See the forest, see the trees. Here lies the c...	Communicating data science: A guide to present...	Live	1
2	☰ * Login\r\n * Sign Up\r\n\r\n * Learning Pat...	Here’s this week’s news in Data Science and Bi...	This Week in Data Science (April 18, 2017)	Live	2
3	DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...	Learn how distributed DBs solve the problem of...	DataLayer Conference: Boost the performance of...	Live	3
4	Skip navigation Sign in SearchLoading...\r\n\r...	This video demonstrates the power of IBM DataS...	Analyze NY Restaurant data using Spark in DSX	Live	4

In [12]:
```
# median
df.groupby('email')['article_id'].count().median()
```

Out[12]:
```
3.0
```

In [13]:
```
# max
df.groupby('email')['article_id'].count().max()
```

Out[13]:
```
364
```

In [14]:
```
# median and maximum number of user_article interactions

median_val = df.groupby('email')['article_id'].count().median() # 50% of individuals int
max_views_by_user = df.groupby('email')['article_id'].count().max() # The maximum number
```

In [15]:
```
user_interacts = df.groupby('email')['article_id'].count()
```

```
user_interacts
```

Out[15]:

```
email
0000b6387a0366322d7fbfc6434af145adf7fed1    13
001055fc0bb67f71e8fa17002342b256a30254cd     4
00148e4911c7e04eeff8def7bbbdaf1c59c2c621     3
001a852ecbd6cc12ab77a785efa137b2646505fe     6
001fc95b90da5c3cb12c501d201a915e4f093290     2
                                            ..
ffc6cfa435937ca0df967b44e9178439d04e3537     2
ffc96f8fbb35aac4cb0029332b0fc78e7766bb5d     4
ffe3d0543c9046d35c2ee3724ea9d774dff98a32    32
fff9fc3ec67bd18ed57a34ed1e67410942c4cd81    10
fffb93a166547448a0ff0232558118d59395fecd    13
Name: article_id, Length: 5148, dtype: int64
```

In [16]:

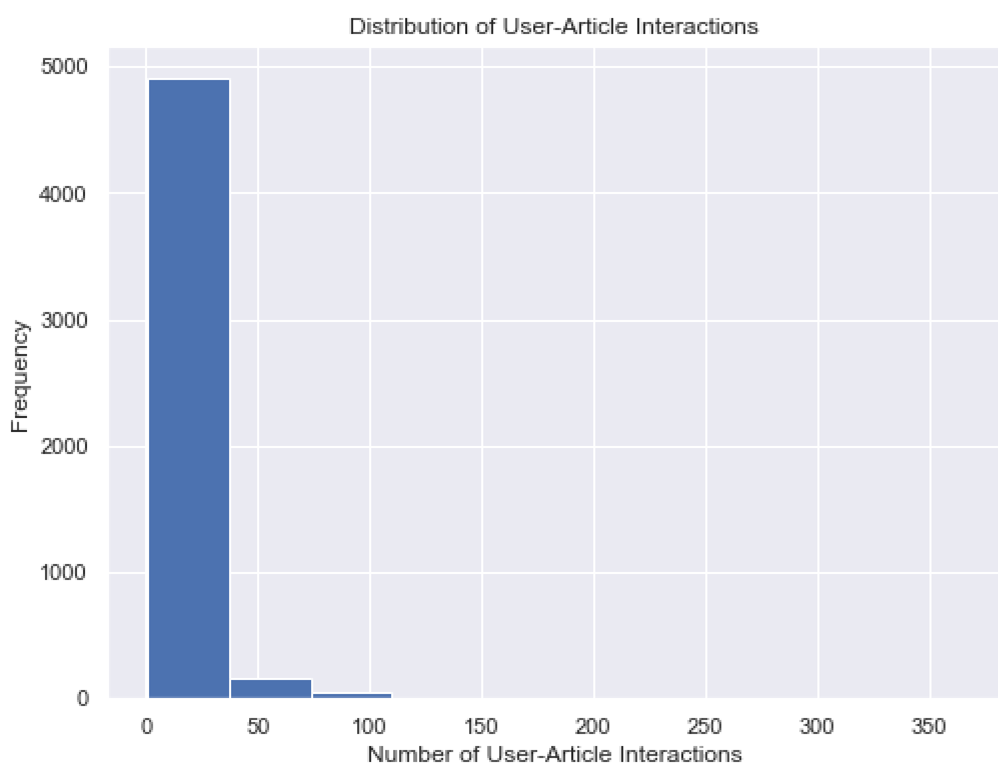
```
# summary stats
user_interacts.describe()
```

Out[16]:

```
count    5148.000000
mean        8.930847
std        16.802267
min         1.000000
25%         1.000000
50%         3.000000
75%         9.000000
max       364.000000
Name: article_id, dtype: float64
```

In [17]:

```
# plot graph
plt.figure(figsize=(8,6))
user_interacts.plot(kind='hist')
plt.title('Distribution of User-Article Interactions')
plt.xlabel('Number of User-Article Interactions');
```



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

In [18]:

```
# Find and explore duplicate articles
df_content.head(3)
```

Out[18]:

	doc_body	doc_description	doc_full_name	doc_status	article_id
0	Skip navigation Sign in SearchLoading...\r\n\r...	Detect bad readings in real time using Python ...	Detect Malfunctioning IoT Sensors with Streami...	Live	0
1	No Free Hunch Navigation * kaggle.com\r\n\r\n ...	See the forest, see the trees. Here lies the c...	Communicating data science: A guide to present...	Live	1
2	☰ * Login\r\n * Sign Up\r\n\r\n * Learning Pat...	Here's this week's news in Data Science and Bi...	This Week in Data Science (April 18, 2017)	Live	2

In [19]:
```python
# Remove any rows that have the same article_id - only keep the first
df_content.shape
```

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

In [20]:
```python
# find duplicate articles
df_content.article_id.duplicated().sum()
```

Out[20]:
```
5
```

In [21]:
```python
ids = df_content['article_id']
```

In [22]:
```python
# explore duplicate articles
df_content[ids.isin(ids[ids.duplicated()])]
```

Out[22]:

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

In [23]:
```python
# Remove any rows that have the same article_id - only keep the first
df_content.drop_duplicates(subset=['article_id'], keep='first', inplace=True)
```

In [24]:
```python
# check if all duplicated rows have been removed
df_content.article_id.duplicated().sum()
```

Out[24]:
```
0
```

In [25]:
```python
# check this works by pulling out an example
df_content.iloc[971]
```

Out[25]:
```
doc_body           Cloudant allows custom Javascript to be run se...
doc_description    Cloudant allows custom Javascript to be run se...
doc_full_name               Defensive coding in Map/Index functions
doc_status                                                     Live
article_id                                                      971
Name: 976, dtype: object
```

`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 [26]:
```python
print(df.shape)
print(df_content.shape)
```

```
(45993, 3)
(1051, 5)
```

In [27]:
```python
# a.
# The number of unique articles that have at least one interaction
len(df['article_id'].unique())
```

Out[27]:
```
714
```

In [28]:
```python
# b.
# The number of unique articles on the IBM platform
len(df_content['article_id'].unique())
```

Out[28]:
```
1051
```

In [29]:
```python
# c.
# The number of unique users
len(df.email.unique())
```

Out[29]:
```
5149
```

In [30]:
```python
# check if we have email address that is left empty (null) in the email column.
df.email.isna().unique().sum()

# we have null email; therefore the unique users we have is 5149-1 = 5148
```

Out[30]:
```
1
```

In [31]:
```python
df.email.nunique()
```

Out[31]:
```
5148
```

In [32]:
```python
# d.
len(df)
```

Out[32]:
```
45993
```

In [33]:
```
# The number of user-article interactions
df.shape[0]
```

Out[33]:
```
45993
```

In [34]:
```
unique_articles = df.article_id.nunique() # The number of unique articles that have at l
total_articles =  df_content.article_id.nunique() # The number of unique articles on the
unique_users = df.email.nunique() # The number of unique users
user_article_interactions = df.shape[0] # The number of user-article interactions
```

`4.` Use the cells below to find the most viewed **article_id**, as well as how often it was viewed. After talking to the company leaders, the `email_mapper` function was deemed a reasonable way to map users to ids. There were a small number of null values, and it was found that all of these null values likely belonged to a single user (which is how they are stored using the function below).

In [35]:
```
# most viewed article_id
df.article_id.value_counts().head(2)
```

Out[35]:
```
1429.0    937
1330.0    927
Name: article_id, dtype: int64
```

In [36]:
```
# show top 10 article_id most viewed by users.
df[['article_id', 'email']].groupby(['article_id']).count().sort_values(['email'], ascen
```

Out[36]:

	email
article_id	
1429.0	937
1330.0	927
1431.0	671
1427.0	643
1364.0	627
1314.0	614
1293.0	572
1170.0	565
1162.0	512
1304.0	483

In [37]:
```
most_viewed_article_id = str(df.article_id.value_counts().index[0]) # The most viewed ar
max_views = df.article_id.value_counts().iloc[0] # The most viewed article in the datase
```

In [38]:
```
## No need to change the code here - this will be helpful for later parts of the noteboo
# 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[38]:

	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 [39]:

```
## 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_ar
    '`The maximum number of user-article interactions by any 1 user is ______.`': max_vi
    '`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_articl
    '`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!
```

Part II: Rank-Based Recommendations

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.` The function below returns the **n** top articles ordered with most interactions as the top.

In [40]:

```
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

    '''
    top_articles = df['title'].value_counts().index.tolist()[:n]
    top_articles = [str(i) for i in top_articles]
```

```
    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 ids

    '''
    top_articles = df['article_id'].value_counts().index.tolist()[:n]
    top_articles = [str(i) for i in top_articles]

    return top_articles # Return the top article ids
```

In [41]:
```
print(get_top_articles(10))
print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new york car accident repo
rts', 'visualize car data with brunel', 'use xgboost, scikit-learn & ibm watson machine
learning apis', 'predicting churn with the spss random tree algorithm', 'healthcare pyth
on streaming application demo', 'finding optimal locations of new store using decision o
ptimization', 'apache spark lab, part 1: basic concepts', 'analyze energy consumption in
buildings', 'gosales transactions for logistic regression model']
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '1170.0', '1162.
0', '1304.0']
```

In [42]:
```
# 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.` The function below reformats 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**.

In [43]:
```
# 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 w
    an article and a 0 otherwise
    '''
    df_count = df.groupby(['user_id', 'article_id']).count().reset_index() # create a ne
    user_item = df_count.pivot_table(values='title', index='user_id', columns='article_i
    user_item.replace(np.nan, 0, inplace=True) # replace nulls with 0s
    user_item=user_item.applymap(lambda x: 1 if x > 0 else x) # entries should be a 1 or

    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
```

In [44]:
```
user_item.sum(axis=1)[1]
```

Out[44]:
```
36.0
```

In [45]:
```
## 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 matri
assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article mat
assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 doe
print("You have passed our quick tests!  Please proceed!")
```

```
You have passed our quick tests!  Please proceed!
```

`2.` The function below takes a user_id and provides an ordered list of the most similar users to that user (from most similar to least similar). The returned result does 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 makes sense to compute similarity as the dot product of two users.

In [46]:
```
user_item.head()
```

Out[46]:

article_id	0.0	2.0	4.0	8.0	9.0	12.0	14.0	15.0	16.0	18.0	...	1434.0	1435.0	1436.0	1437.0	1439.0	144
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

In [47]:
```
def find_similar_users(user_id, user_item=user_item):
    '''
    INPUT:
    user_id - (int) a user_id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

    OUTPUT:
    similar_users - (list) an ordered list where the closest users (largest dot product
```

```
                    are listed first

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

    '''
    # compute similarity of each user to the provided user
    dot_prod_users = user_item.dot(np.transpose(user_item))

    # sort by similarity
    sim_users = dot_prod_users[user_id].sort_values(ascending = False)

    # create list of just the ids
    most_similar_users = sim_users.index.tolist()

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

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

In [48]:

```
# 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)[:5
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, 4
6, 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 we have a function that provides the most similar users to each user, we want to use these users to find articles we can recommend. The functions below return the articles we would recommend to each user.

In [49]:

```
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 i
                    (this is identified by the title column)
    '''
    article_names = []

    for idx in article_ids:
        article_names.append(df[df['article_id']==float(idx)].max()['title'])

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


def get_user_articles(user_id, user_item=user_item):
    '''
    INPUT:
    user_id - (int) a user id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

    OUTPUT:
    article_ids - (list) a list of the article ids seen by the user
    article_names - (list) a list of article names associated with the list of article i
```

```
    Description:
    Provides a list of the article_ids and article titles that have been seen by a user
    '''
    article_ids = user_item.loc[user_id][user_item.loc[user_id] == 1].index.astype('str'

    article_names = []

    for idx in article_ids:
        article_names.append(df[df['article_id']==float(idx)].max()['title']) # need to

    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 recs
    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

    '''
    recs = np.array([]) # recommendations to be made

    user_articles_seen = get_user_articles(user_id)[0] #seen by our user
    closest_users = find_similar_users(user_id) # users closest to our user

    for others in closest_users:

        others_articles_seen = get_user_articles(others)[0] # articles seen by others li
        new_recs = np.setdiff1d(others_articles_seen, user_articles_seen, assume_unique=
        recs = np.unique(np.concatenate([new_recs, recs], axis = 0)) # concate arrays an

        if len(recs) > m-1:
            break

    recs = recs[:m]
    recs.tolist()

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

In [50]:

```
# Check Results
get_article_names(user_user_recs(1, 10)) # Return 10 recommendations for user 1
```

Out[50]:

```
['recommender systems: approaches & algorithms',
 '1448    i ranked every intro to data science course on...\nName: title, dtype: objec
t',
 'data tidying in data science experience',
 'a tensorflow regression model to predict house values',
 '520    using notebooks with pixiedust for fast, flexi...\nName: title, dtype: object',
 'airbnb data for analytics: mallorca reviews',
 'airbnb data for analytics: vancouver listings',
 'analyze facebook data using ibm watson and watson studio',
```

```
 'analyze accident reports on amazon emr spark',
 'analyze energy consumption in buildings']
```

In [51]:
```
# Test your functions here - No need to change this code - just run this cell
assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.0'
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): uni
assert set(get_user_articles(20)[0]) == set(['1320.0', '232.0', '844.0'])
assert set(get_user_articles(20)[1]) == set(['housing (2015): united states demographic 
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.0', '142
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-res
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 we wrote earlier.

In [52]:
```
def get_top_sorted_users(user_id, df=df, user_item=user_item):
    '''
    INPUT:
    user_id - (int)
    df - (pandas dataframe) df as defined at the top of the notebook
    user_item - (pandas dataframe) matrix of users by articles:
            1's when a user has interacted with an article, 0 otherwise


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

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

    '''
    # create neighbors dataframe with empty columns
    neighbors_df = pd.DataFrame(columns=['neighbor_id', 'similarity'])
    # set neighbor_id column equal to user_item index starting from 1
    neighbors_df['neighbor_id'] = user_item.index-1
    # make similarity column equal to most similar using dot product
    dot_prod_users = user_item.dot(np.transpose(user_item))
    neighbors_df['similarity'] = dot_prod_users[user_id]
    # create new df based on number of interactions of users
    interacts_df = df.user_id.value_counts().rename_axis('neighbor_id').reset_index(name
    # merge dataframes which creates number of interactions column from interacts_df
    neighbors_df = pd.merge(neighbors_df, interacts_df, on='neighbor_id', how='outer')
    # sortvalues on similarity and then number of interactions
    neighbors_df = neighbors_df.sort_values(by=['similarity', 'num_interactions'], ascen
    # reset index
    neighbors_df = neighbors_df.reset_index(drop=True)
    # drop row with the user_id as itself will be most similar
    neighbors_df = neighbors_df[neighbors_df.neighbor_id != user_id]

    return neighbors_df # Return the dataframe specified in the doc_string
```

```python
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 recs
    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.

    '''
    recs = np.array([]) # recommendations to be made

    user_articles_ids_seen, user_articles_names_seen = get_user_articles(user_id, user_i
    closest_neighs = get_top_sorted_users(user_id, df, user_item).neighbor_id.tolist() #

    for neighs in closest_neighs:

        neigh_articles_ids_seen, neigh_articles_names_seen = get_user_articles(neighs, u
        new_recs = np.setdiff1d(neigh_articles_ids_seen, user_articles_ids_seen, assume_
        recs = np.unique(np.concatenate([new_recs, recs], axis = 0)) # concate arrays an

        if len(recs) > m-1:
            break

    recs = recs[:m]
    recs = recs.tolist() # convert to a list

    rec_names = get_article_names(recs, df=df)

    return recs, rec_names
```

In [53]:

```python
# Quick spot check - don't change this code - just use it to test your functions
rec_ids, rec_names = user_user_recs_part2(20, 10)
print("The top 10 recommendations for user 20 are the following article ids:")
print(rec_ids)
print()
print("The top 10 recommendations for user 20 are the following article names:")
print(rec_names)
```

```
The top 10 recommendations for user 20 are the following article ids:
['1024.0', '1085.0', '109.0', '1150.0', '1151.0', '1152.0', '1153.0', '1154.0', '1157.
0', '1160.0']

The top 10 recommendations for user 20 are the following article names:
['using deep learning to reconstruct high-resolution audio', 'airbnb data for analytics:
chicago listings', 'tensorflow quick tips', 'airbnb data for analytics: venice calenda
r', 'airbnb data for analytics: venice listings', 'airbnb data for analytics: venice rev
iews', 'airbnb data for analytics: vienna calendar', 'airbnb data for analytics: vienna
listings', 'airbnb data for analytics: washington d.c. listings', 'analyze accident repo
rts on amazon emr spark']
```

`5.` The functions from above are now used to correctly fill in the solutions to the dictionary below.

In [54]:
```
# # Find the user that is most similar to user 1
get_top_sorted_users(1)[:10]

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

Out[54]:

	neighbor_id	similarity	num_interactions
1	3870	74.0	144.0
2	3782	39.0	363.0
3	23	38.0	364.0
4	203	33.0	160.0
5	4459	33.0	158.0
6	98	29.0	170.0
7	3764	29.0	169.0
8	49	29.0	147.0
9	3697	29.0	145.0
11	3910	25.0	147.0

In [55]:
```
### Tests with a dictionary of results

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

In [56]:
```
## 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 given a new user, it would make sense to use Rank Based Recommendations and the `get_top_articles` function to make recommendations. We would just recommend the most popular articles since we do not have any information about the user or their interactions so cannot tell which other users they are most similar to. Once we have more information about the user we could a blended approach of 3 types of recommendation techniques; Rank, Content, and Collaborative.

`7.` Using our existing functions, we can provide the top 10 recommended articles we would provide for the a new user below.

In [57]:
```
new_user = '0.0'

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

In [58]:
```
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!
```

Part IV: Matrix Factorization

In this part of the notebook, we will use matrix factorization to make article recommendations to the users on the IBM Watson Studio platform.

`1.` We 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 IV** of the notebook.

In [59]:
```
# Load the matrix here
user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

In [60]:
```
# quick look at the matrix
user_item_matrix.head()
```

Out[60]:

article_id	0.0	100.0	1000.0	1004.0	1006.0	1008.0	101.0	1014.0	1015.0	1016.0	...	977.0	98.0	981.0	98
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	
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	
3	0.0	0.0	0.0	0.0	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	
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	

5 rows × 714 columns

`2.` In this situation, we can use Singular Value Decomposition from numpy on the user-item matrix. We perform SVD, and explain why this is different than in the lessons.

In [61]:
```
# Perform SVD on the User-Item Matrix Here

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

In [62]:
```
s.shape, u.shape, vt.shape
```

Out[62]:
```
((714,), (5149, 5149), (714, 714))
```

We have no missing values in this matrix therefore we can perform SVD. In the classroom, our matrix had missing values which meant that we had to use FunkSVD.

`3.` Now for the tricky part, how do we choose the number of latent features to use? Running the below cell, we 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. We get an idea of how the accuracy improves as we increase the number of latent features.

In [63]:
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs = []

for k in num_latent_feats:
```

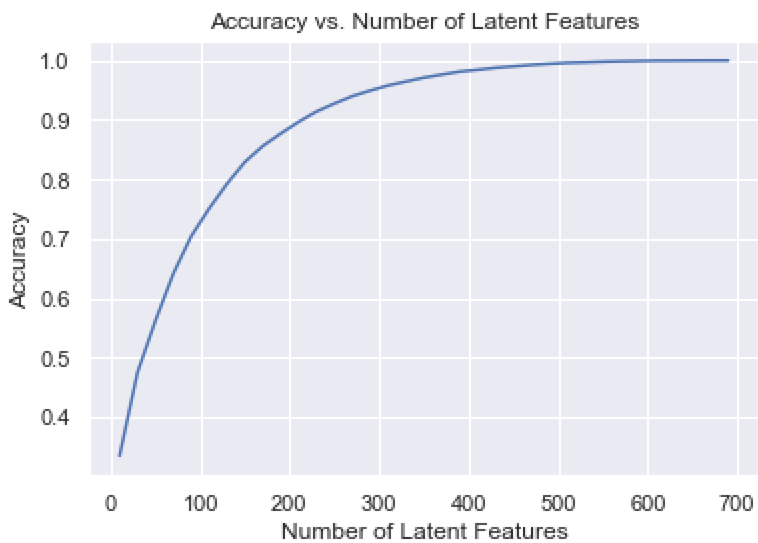
```
    # restructure with k latent features
    s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]

    # 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.

Using the code from question 3 we can 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 [64]:
```
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

    '''
    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(d
```

In [65]:

```
test_idx # 682 users in test set
```

Out[65]:

```
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)
```

In [66]:

```
train_idx = user_item_train.index # 4487 users in training set
train_idx
```

Out[66]:

```
Int64Index([   1,    2,    3,    4,    5,    6,    7,    8,    9,   10,
            ...
            4478, 4479, 4480, 4481, 4482, 4483, 4484, 4485, 4486, 4487],
           dtype='int64', name='user_id', length=4487)
```

In [67]:

```
test_idx.difference(train_idx) # of 682 users in test set, only 20 of them are in traini
```

Out[67]:

```
Int64Index([4488, 4489, 4490, 4491, 4492, 4493, 4494, 4495, 4496, 4497,
            ...
            5140, 5141, 5142, 5143, 5144, 5145, 5146, 5147, 5148, 5149],
           dtype='int64', name='user_id', length=662)
```

In [68]:

```
test_arts #574 movies in test set
```

Out[68]:

```
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 [69]:

```
train_arts = user_item_train.columns #714 movies in train set
train_arts
```

Out[69]:

```
Float64Index([   0.0,    2.0,    4.0,    8.0,    9.0,   12.0,   14.0,   15.0,
                16.0,   18.0,
              ...
              1434.0, 1435.0, 1436.0, 1437.0, 1439.0, 1440.0, 1441.0, 1442.0,
              1443.0, 1444.0],
             dtype='float64', name='article_id', length=714)
```

In [70]:

```
test_arts.difference(train_arts) # all articles in test set are in training set too
```

Out[70]:

```
Float64Index([], dtype='float64', name='article_id')
```

In [71]:

```
# 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 t
    'How many movies can we make predictions for in the test set?': b,
    'How many movies in the test set are we not able to make predictions for because of
}

t.sol_4_test(sol_4_dict)
```

```
Awesome job!  That's right!  All of the test movies are in the training data, but there
are only 20 test users that were also in the training set.  All of the other users that
are in the test set we have no data on.  Therefore, we cannot make predictions for these
users using SVD.
```

`5.` Now we can 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 we 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`.

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

In [72]:
```
# 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 t
```

In [73]:
```
# Use these cells to see how well you can use the training
# decomposition to predict on test data
s_train.shape, u_train.shape, vt_train.shape
```

Out[73]:
```
((714,), (4487, 4487), (714, 714))
```

In [74]:
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs_train = []
sum_errs_test = []


#Decomposition
row_idx = user_item_train.index.isin(test_idx)
col_idx = user_item_train.columns.isin(test_arts)

u_test = u_train[row_idx, :]
vt_test = vt_train[:, col_idx]

# test users that we can predict for
users_can_predict = np.intersect1d(list(user_item_train.index),list(user_item_test.index

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

    # take dot product
    user_item_train_preds = np.around(np.dot(np.dot(u_train_new, s_train_new), vt_train_
    user_item_test_preds = np.around(np.dot(np.dot(u_test_new, s_train_new), vt_test_new

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

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

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

    sum_errs_train.append(err_train)
    sum_errs_test.append(err_test)
```

In [75]:

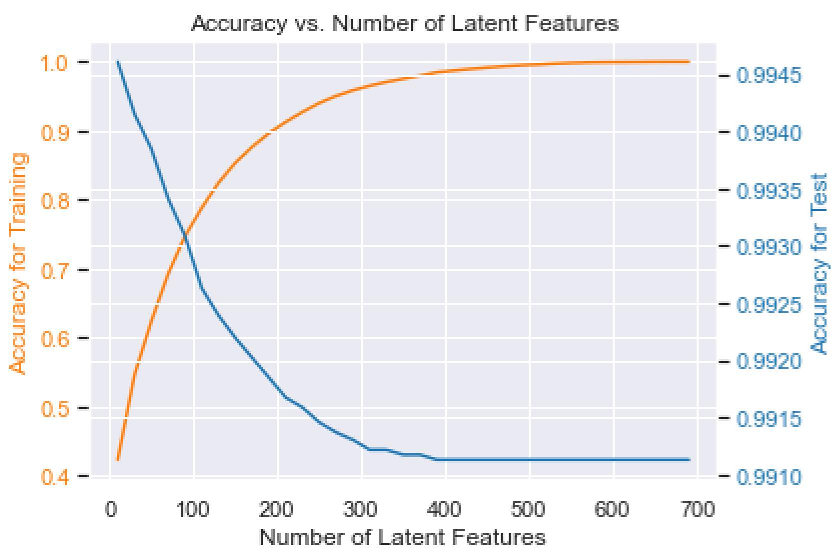
```
# plotting the training and test accuracies
fig, ax1 = plt.subplots()

color = 'tab:orange'
ax1.set_xlabel('Number of Latent Features')
ax1.set_ylabel('Accuracy for Training', color=color)
ax1.plot(num_latent_feats, 1 - np.array(sum_errs_train)/df.shape[0], color=color)
ax1.tick_params(axis='y', labelcolor=color)
ax1.set_title('Accuracy vs. Number of Latent Features')

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('Accuracy for Test', color=color)  # we already handled the x-label with
ax2.plot(num_latent_feats, 1 - np.array(sum_errs_test)/df.shape[0], color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```



From above, we can see that the accuracy for the training data increases with an increase in the number of latent features, however the opposite is true for the accuracy of the test data. This is most likely due to overfitting of the data with the increase in latent features, therefore the number of latent features should be kept relatively low. It is important to note that using SVD here we can only actually make recommendations for the 20 users in both the training and test dataset, and we have a very sparse matrix which is likely why the test data accuracy is so high at >99%. It would be interesting to look at results if we had more users that appeared in both the test and training data. We can see that at approximately 80 features there is a cross over point when the accuracy for test data begins to drop, therefore this would be a good number of latent features to include, since beyond that our accuracy for training increases but testing decreases. To test how well our recommendation engine works in practice, we could conduct an A/B test for new users to help solve the cold start problem. An example would be to recommend articles to one group using our recommendation engine and then to recommend just the most popular articles to the other group of users. We would then compare the click through rates to effectively measure if our recommendation engine leads to an increase in clicks. If we saw a significant rise in clicks by using our recommendation engine then we could conclude this works well and should be deployed.

Conclusion

When using SVD we experience the cold start problem. That is, we can only make predictions for articles and users that exist in both the training and test sets. For users in the test dataset that are not in the training set, we cannot predict articles to recommend to that user. To improve upon our recommendation engine for new users, we could use a blended techniques of Knowledge based, collaborative filtering based, and content based recommendations. We could also conduct an A/B test to best understand which recommendation technique should be employed.

References

https://stackoverflow.com/questions/14657241/how-do-i-get-a-list-of-all-the-duplicate-items-using-pandas-in-python
https://stackoverflow.com/questions/35523635/extract-values-in-pandas-value-counts
https://knowledge.udacity.com/questions/140813
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.difference.html
https://knowledge.udacity.com/questions/387214
https://matplotlib.org/gallery/api/two_scales.html

In [76]:
```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

Out[76]:
```
1
```

In [78]:
```
!jupyter nbconvert --to webpdf --allow-chromium-download Recommendations_with_IBM.ipynb
```

```
[NbConvertApp] Converting notebook Recommendations_with_IBM.ipynb to webpdf
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 444183 bytes to Recommendations_with_IBM.pdf
```

In [ ]: