

Recommendations_with_IBM

December 17, 2023

1 Recommendations with IBM

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

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

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

1.1 Table of Contents

I. Section ?? II. Section ?? III. Section ?? IV. Section ?? V. Section ?? VI. Section ??

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

%matplotlib inline

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

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

Out[394]:

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

```

                                email
0  ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1  083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2  b96a4f2e92d8572034b1e9b28f9ac673765cd074
3  06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4  f01220c46fc92c6e6b161b1849de11faacd7ccb2

```

```

In [395]: # Show df_content to get an idea of the data
df_content.head()

```

```

Out[395]:                                doc_body \
0  Skip navigation Sign in SearchLoading...\r\n\r...
1  No Free Hunch Navigation * kaggle.com\r\n\r\n ...
2  * Login\r\n * Sign Up\r\n\r\n * Learning Pat...
3  DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...
4  Skip navigation Sign in SearchLoading...\r\n\r...

```

```

                                doc_description \
0  Detect bad readings in real time using Python ...
1  See the forest, see the trees. Here lies the c...
2  Heres this weeks news in Data Science and Bi...
3  Learn how distributed DBs solve the problem of...
4  This video demonstrates the power of IBM DataS...

```

	doc_full_name	doc_status	article_id
0	Detect Malfunctioning IoT Sensors with Streami...	Live	0
1	Communicating data science: A guide to present...	Live	1
2	This Week in Data Science (April 18, 2017)	Live	2
3	DataLayer Conference: Boost the performance of...	Live	3
4	Analyze NY Restaurant data using Spark in DSX	Live	4

1.1.1 Part I: Exploratory Data Analysis

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

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

```

In [396]: # Descriptive statistics of how many articles a user interacts with in the dataset
user_interact_article = df.groupby('email')['article_id'].count()
user_interact_article.describe()

```

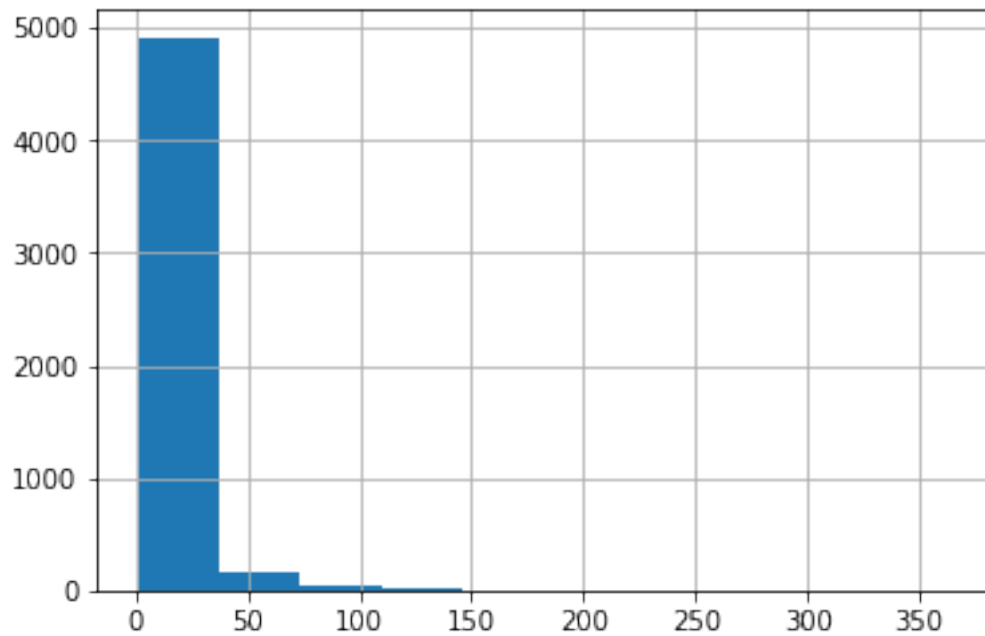
```

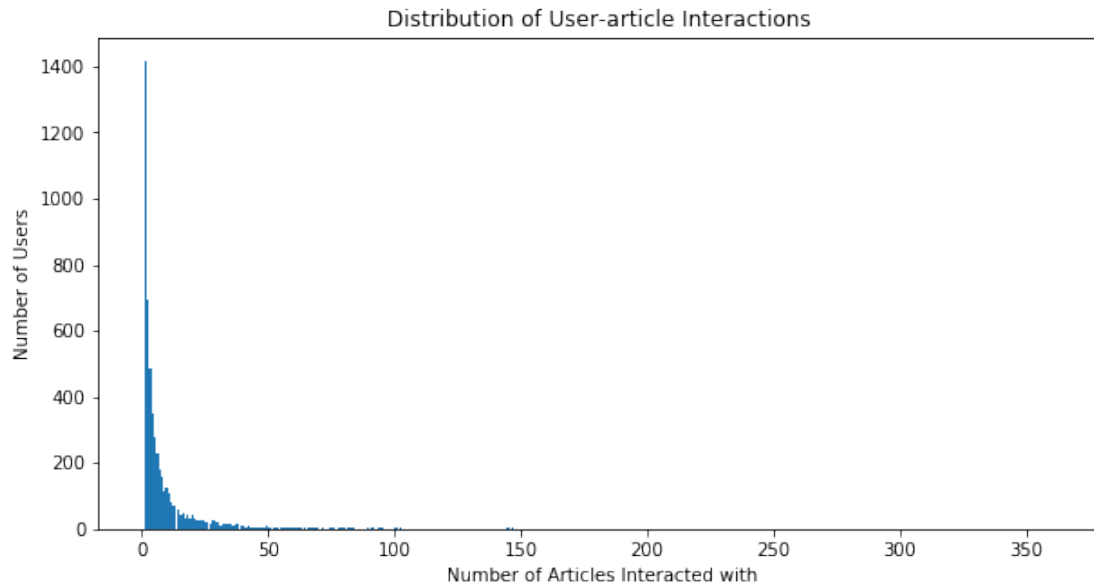
Out[396]: count    5148.000000
          mean      8.930847
          std      16.802267
          min       1.000000
          25%       1.000000

```

```
50%          3.000000
75%          9.000000
max         364.000000
Name: article_id, dtype: float64
```

```
In [397]: #Visual of how many articles a user interacts with in the dataset
user_interact_article.hist()
plt.figure(figsize=(10,5))
plt.hist(user_interact_article,bins = 'auto')
plt.xlabel('Number of Articles Interacted with')
plt.ylabel('Number of Users')
plt.title('Distribution of User-article Interactions ')
plt.show();
```





In [398]: *# Fill in the median and maximum number of user_article interactions below*

```
print(user_interact_article.median())
print(user_interact_article.max())

median_val = user_interact_article.median() # 50% of individuals interact with ____ nu
max_views_by_user = user_interact_article.max() # The maximum number of user-article i
```

3.0

364

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

In [399]: *# Find and explore duplicate articles*

```
df_content.duplicated(subset='article_id').sum()
```

Out[399]: 5

In [400]: *# Find and explore duplicate articles*

```
df_content[df_content.duplicated(['article_id'], keep='first')] # show first value of
```

```
Out[400]:
doc_body \
365 Follow Sign in / Sign up Home About Insight Da...
692 Homepage Follow Sign in / Sign up Homepage * H...
761 Homepage Follow Sign in Get started Homepage *...
970 This video shows you how to construct queries ...
971 Homepage Follow Sign in Get started * Home\r\n...
```

```

doc_description \
365 During the seven-week Insight Data Engineering...
692 One of the earliest documented catalogs was co...
761 Todays world of data science leverages data f...
970 This video shows you how to construct queries ...
971 If you are like most data scientists, you are ...

```

	doc_full_name	doc_status	article_id
365	Graph-based machine learning	Live	50
692	How smart catalogs can turn the big data flood...	Live	221
761	Using Apache Spark as a parallel processing fr...	Live	398
970	Use the Primary Index	Live	577
971	Self-service data preparation with IBM Data Re...	Live	232

```
In [401]: df_content.duplicated("article_id").sum() #number of duplicates
```

```
Out[401]: 5
```

```
In [402]: # Remove any rows that have the same article_id - only keep the first instance of dupl
df_content.drop_duplicates(subset='article_id', keep='first')
```

```

Out[402]: doc_body \
0 Skip navigation Sign in SearchLoading...\r\n\r...
1 No Free Hunch Navigation * kaggle.com\r\n\r\n ...
2 * Login\r\n * Sign Up\r\n\r\n * Learning Pat...
3 DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...
4 Skip navigation Sign in SearchLoading...\r\n\r...
5 Compose is all about immediacy. You want a new...
6 UPGRADING YOUR POSTGRESQL TO 9.5Share on Twitt...
7 Follow Sign in / Sign up 135 8 * Share\r\n * 1...
8 * Host\r\n * Competitions\r\n * Datasets\r\n *...
9 THE GRADIENT FLOW\r\nDATA / TECHNOLOGY / CULTU...
10 OFFLINE-FIRST IOS APPS WITH SWIFT & PART 1: TH...
11 Warehousing data from Cloudant to dashDB great...
12 Skip to main content IBM developerWorks / Deve...
13 Maureen McElaney Blocked Unblock Follow Follow...
14 Raj Singh Blocked Unblock Follow Following Dev...
15 * Home\r\n * Community\r\n * Projects\r\n * Bl...
16 * Home\r\n * Research\r\n * Partnerships and C...
17 Enterprise Pricing Articles Sign in Free 30-Da...
18 Homepage Follow Sign in / Sign up * Home\r\n *...
19 METRICS MAVEN: MODE D'EMPLOI - FINDING THE MOD...
20 Homepage Follow Sign in / Sign up Homepage * H...
21 Raj Singh Blocked Unblock Follow Following Dev...
22 IMPORTING JSON DOCUMENTS WITH NOSQLIMPORT\r\nG...
23 This video shows you how to build and query a ...
24 THE CONVERSATIONAL INTERFACE IS THE NEW PARADI...
25 Skip navigation Upload Sign in SearchLoading...

```

```

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

```

```

                                doc_description \
0  Detect bad readings in real time using Python ...
1  See the forest, see the trees. Here lies the c...
2  Heres this weeks news in Data Science and Bi...
3  Learn how distributed DBs solve the problem of...
4  This video demonstrates the power of IBM DataS...
5      Using Compose's PostgreSQL data browser.
6  Upgrading your PostgreSQL deployment to versio...
7  For a company like Slack that strives to be as...
8  Kaggle is your home for data science. Learn ne...
9  [A version of this post appears on the OReill...
10 Apple's sample app, Food Tracker, taught you i...

```

11 Replicating data to a relational dashDB databa...
 12 This recipe showcases how one can analyze the ...
 13 Theres a reason youve been hearing a lot abo...
 14 Who are those people lurking behind the statis...
 15 Early methods to integrate machine learning us...
 16 The performance of supervised predictive model...
 17 We've always considered MySQL as a potential C...
 18 It has never been easier to build AI or machin...
 19 In our Metrics Maven series, Compose's data sc...
 20 It is often useful to use RStudio for one piec...
 21 Youre doing your data a disservice if you don...
 22 Introducing nosqlimport, an npm module to help...
 23 This video shows you how to build and query a ...
 24 Botkit provides a simple framework to handle t...
 25 Want to learn more about how we created the Da...
 26 Much of driving is spent either stuck in traff...
 27 This talk assumes you have a basic understandi...
 28 In this paper, we propose gcForest, a decision...
 29 Im very happy and proud to announce that IBM ...
 ...
 1026 Varun Singh, a software engineer at IBM's Wats...
 1027 This video shows you how to create and adminis...
 1028 With the latest 0.2.1 version of Transporter, ...
 1029 Audio super-resolution aims to reconstruct a h...
 1030 Since then, this metric has been ubiquitously ...
 1031 Build a word game app and see how to manage an...
 1032 Use Redis and and Python scripts to speed your...
 1033 In this post, youll learn to query, update, a...
 1034 Adron Hall of Thrashing Code and Home Depot, t...
 1035 Describes the use of Laplace noise in machine ...
 1036 A full guide to Elasticsearch, the real-time d...
 1037 See how quick and easy it is to set up a dashD...
 1038 The relational database has been the dominant ...
 1039 Building your first data warehouse doesnt hav...
 1040 In my last blog Business differentiation thro...
 1041 MongoDB's aggregation pipeline makes finding d...
 1042 Which write API endpoint is the right write ca...
 1043 Nothing spoils a plot like (too much) data.
 1044 Getting started with custom visualizations, si...
 1045 Although it is built around a JavaScript engin...
 1046 Last week I attended the GeoPython conference ...
 1047 In this post, we will go through how to read a...
 1048 As more devices become internet enabled, harne...
 1049 Continuing my previous work on exploring Arlin...
 1050 Lua is a compact language which can be embedde...
 1051 PouchDB uses MapReduce as its default search m...
 1052 We compare discriminative and generative learn...
 1053 In order to demystify some of the magic behind...

1054 Learn how to use IBM dashDB as data store for ...
 1055 Once you get used to developing in a Notebook ...

	doc_full_name	doc_status	article_id
0	Detect Malfunctioning IoT Sensors with Streami...	Live	0
1	Communicating data science: A guide to present...	Live	1
2	This Week in Data Science (April 18, 2017)	Live	2
3	DataLayer Conference: Boost the performance of...	Live	3
4	Analyze NY Restaurant data using Spark in DSX	Live	4
5	Browsing PostgreSQL Data with Compose	Live	5
6	Upgrading your PostgreSQL to 9.5	Live	6
7	Data Wrangling at Slack	Live	7
8	Data Science Bowl 2017	Live	8
9	Using Apache Spark to predict attack vectors a...	Live	9
10	Offline-First iOS Apps with Swift & Cloudant S...	Live	10
11	Warehousing GeoJSON documents	Live	11
12	Timeseries Data Analysis of IoT events by usin...	Live	12
13	Bridging the Gap Between Python and Scala Jupy...	Live	13
14	Got zip code data? Prep it for analytics. IB...	Live	14
15	Apache Spark 2.0: Extend Structured Streaming...	Live	15
16	Higher-order Logistic Regression for Large Dat...	Live	16
17	Compose for MySQL now for you	Live	17
18	The Greatest Public Datasets for AI Startup ...	Live	18
19	Finding the Mode in PostgreSQL	Live	19
20	Working interactively with RStudio and noteboo...	Live	20
21	Mapping for Data Science with PixieDust and Ma...	Live	21
22	Move CSVs into different JSON doc stores	Live	22
23	Tutorial: How to build and query a Cloudant ge...	Live	23
24	The Conversational Interface is the New Paradigm	Live	24
25	Creating the Data Science Experience	Live	25
26	Using Machine Learning to predict parking diff...	Live	26
27	Getting The Best Performance With PySpark	Live	27
28	Deep Forest: Towards An Alternative to Deep Ne...	Live	28
29	Experience IoT with Coursera	Live	29
...
1026	Redis and MongoDB in the biomedical domain	Live	1021
1027	Create and administer a data catalog using IBM...	Live	1022
1028	How to move data with Compose Transporter - Fr...	Live	1023
1029	Using Deep Learning to Reconstruct High-Resolu...	Live	1024
1030	Data tidying in Data Science Experience	Live	1025
1031	Build a simple word game app using Cloudant on...	Live	1026
1032	Blazingly Fast Geospatial Queries with Redis	Live	1027
1033	Working with SQLite Databases using Python and...	Live	1028
1034	DataLayer Conference: Managing (or not) the Da...	Live	1029
1035	Laplace noising versus simulated out of sample...	Live	1030
1036	The Definitive Guide	Live	1031
1037	Get started with dashDB on Bluemix	Live	1032
1038	The Many Flavors of NoSQL at That Conference	Live	1033

1039	Your First Data Warehouse Is Easy. Meet the ODS.	Live	1034
1040	Machine Learning for the Enterprise.	Live	1035
1041	Finding Duplicate Documents in MongoDB	Live	1036
1042	Piecemeal, Bulk, or Batch? IBM Watson Data L...	Live	1037
1043	Reducing overplotting in scatterplots	Live	1038
1044	You Too Can Make Magic (in Jupyter Notebooks w...	Live	1039
1045	How I Stopped Worrying & Learned to Love the M...	Live	1040
1046	Mapping All the Things with Python IBM Watso...	Live	1041
1047	Use IBM Data Science Experience to Read and Wr...	Live	1042
1048	Use IoT data in Streams Designer for billing a...	Live	1043
1049	Mapping Points with Folium	Live	1044
1050	A Speed Guide To Redis Lua Scripting	Live	1045
1051	A look under the covers of PouchDB-find	Live	1046
1052	A comparison of logistic regression and naive ...	Live	1047
1053	What I Learned Implementing a Classifier from ...	Live	1048
1054	Use dashDB with Spark	Live	1049
1055	Jupyter Notebooks with Scala, Python, or R Ker...	Live	1050

[1051 rows x 5 columns]

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 [403]: df.article_id.nunique() # The number of unique articles that have at least one interaction
```

```
Out[403]: 714
```

```
In [404]: df_content.article_id.nunique() # The number of unique articles on the IBM platform
```

```
Out[404]: 1051
```

```
In [405]: df.email.nunique() #The number of unique users
```

```
Out[405]: 5148
```

```
In [406]: df.shape[0] #The number of user-article interactions in the dataset.
```

```
Out[406]: 45993
```

```
In [407]: unique_articles = df.article_id.nunique() # The number of unique articles that have at least one interaction
total_articles = df_content.article_id.nunique() # The number of unique articles on the IBM platform
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 [408]: df.article_id.value_counts().index[0]
```

```
Out[408]: 1429.0
```

```
In [409]: df.article_id.value_counts().max()
```

```
Out[409]: 937
```

```
In [410]: most_viewed_article_id = str(df.article_id.value_counts().index[0]) # The most viewed
max_views = df.article_id.value_counts().max() # The most viewed article in the dataset
```

```
In [411]: ## No need to change the code here - this will be helpful for later parts of the notebook
# 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[411]:
```

	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 [412]: ## If you stored all your results in the variable names above,
## you shouldn't need to change anything in this cell
```

```
sol_1_dict = {
    '50% of individuals have ____ or fewer interactions.': median_val,
    'The total number of user-article interactions in the dataset is ____': user_
    'The maximum number of user-article interactions by any 1 user is ____': max_
    'The most viewed article in the dataset was viewed ____ times.': max_views,
```

```

    ``The article_id of the most viewed article is _____.``: most_viewed_article_id,
    ``The number of unique articles that have at least 1 rating _____.``: unique_arti
    ``The number of unique users in the dataset is _____.``: unique_users,
    ``The number of unique articles on the IBM platform``: total_articles
}

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

```

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

1.1.2 Part II: Rank-Based Recommendations

Unlike in the earlier lessons, we don't actually have ratings for whether a user liked an article or not. We only know that a user has interacted with an article. In these cases, the popularity of an article can really only be based on how often an article was interacted with.

1. Fill in the function below to return the **n** top articles ordered with most interactions as the top. Test your function using the tests below.

```

In [413]: def get_top_articles(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    '''
    # Your code here
    top_articles = df['title'].value_counts().head(n).index.tolist()

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

def get_top_article_ids(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    '''
    # Your code here
    top_articles = df['article_id'].value_counts().head(n).index.tolist()
    top_articles=[str(x) for x in top_articles]

```

```
    return top_articles # Return the top article ids
```

```
In [414]: print(get_top_articles(10))
          print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new york car accident reports', 'v
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '1170.0', '1162.0', '1304
```

```
In [415]: # Test your function by returning the top 5, 10, and 20 articles
          top_5 = get_top_articles(5)
          top_10 = get_top_articles(10)
          top_20 = get_top_articles(20)

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

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

1.1.3 Part III: User-User Based Collaborative Filtering

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

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

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

```
In [416]: # create the user-article matrix with 1's and 0's
          def create_user_item_matrix(df):
              '''
              INPUT:
              df - pandas dataframe with article_id, title, user_id columns

              OUTPUT:
              user_item - user item matrix
```

```

    Description:
    Return a matrix with user ids as rows and article ids on the columns with 1 values
    an article and a 0 otherwise
    '''
    # Fill in the function here
    user_item = df.groupby(['user_id', 'article_id'])['title'].max().unstack().isnull()

    return user_item # return the user_item matrix

```

```
user_item = create_user_item_matrix(df)
```

```

In [417]: ## Tests: You should just need to run this cell. Don't change the code.
assert user_item.shape[0] == 5149, "Oops! The number of users in the user-article mat
assert user_item.shape[1] == 714, "Oops! The number of articles in the user-article m
assert user_item.sum(axis=1)[1] == 36, "Oops! The number of articles seen by user 1 d
print("You have passed our quick tests! Please proceed!")

```

You have passed our quick tests! Please proceed!

2. Complete the function below which should take a `user_id` and provide an ordered list of the most similar users to that user (from most similar to least similar). The returned result should not contain the provided `user_id`, as we know that each user is similar to him/herself. Because the results for each user here are binary, it (perhaps) makes sense to compute similarity as the dot product of two users.

Use the tests to test your function.

```
In [418]: user_item.head()
```

```

Out[418]: article_id  0.0      2.0      4.0      8.0      9.0      12.0      14.0      15.0  \
user_id
1           0          0          0          0          0          0          0          0
2           0          0          0          0          0          0          0          0
3           0          0          0          0          0          1          0          0
4           0          0          0          0          0          0          0          0
5           0          0          0          0          0          0          0          0

article_id  16.0      18.0      ...      1434.0  1435.0  1436.0  1437.0  1439.0  \
user_id      ...
1           0          0      ...          0          0          1          0          1
2           0          0      ...          0          0          0          0          0
3           0          0      ...          0          0          1          0          0
4           0          0      ...          0          0          0          0          0
5           0          0      ...          0          0          0          0          0

article_id  1440.0  1441.0  1442.0  1443.0  1444.0
user_id
1           0          0          0          0          0

```

2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

[5 rows x 714 columns]

```
In [419]: def find_similar_users(user_id, user_item=user_item):
    """
    INPUT:
    user_id - (int) a user_id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

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

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

    """
    # compute similarity of each user to the provided user
    user = user_item.dot(user_item.loc[user_id])
    # sort by similarity
    user = user.sort_values(ascending=False)
    # create list of just the ids
    similar_users = list(user.index)
    # remove the own user's id
    similar_users.remove(user_id)
    return similar_users # return a list of the users in order from most to least similar

In [420]: # 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, 131, 3870, 46, 4201, 5041]
The 5 most similar users to user 3933 are: [1, 23, 3782, 4459, 203]
The 3 most similar users to user 46 are: [4201, 23, 3782]
```

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

```
In [421]: 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 ids
                 (this is identified by the title column)
'''
# Your code here

article_names = list(df[df.article_id.isin(article_ids)]['title'].unique())
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 ids
                    (this is identified by the doc_full_name column in df_content)

    Description:
    Provides a list of the article_ids and article titles that have been seen by a user
    '''
    # Your code here

    article_ids = user_item.columns[np.where(user_item.loc[user_id]==1)].astype(str).tolist()
    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 recommendations
    '''

```

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

```
similar_users = find_similar_users(user_id, user_item=user_item)
article_ids= get_user_articles(user_id, user_item=user_item)
recs = []
for similar in similar_users:
    ids = get_user_articles(similar)
    new_rec = np.setdiff1d(ids, article_ids, assume_unique=True)
    recs = np.unique(np.concatenate([new_rec,recs], axis=0))
    if len(recs)>m-1:
        recs = recs[:m]
        break

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

In []:

In [422]: *# Check Results*

```
get_article_names(user_user_rec(1, 10)) # Return 10 recommendations for user 1
```

```
Out[422]: ['analyze energy consumption in buildings',
'analyze accident reports on amazon emr spark',
'520    using notebooks with pixiedust for fast, flexi...\nName: title, dtype: object',
'1448    i ranked every intro to data science course on...\nName: title, dtype: object',
'data tidying in data science experience',
'airbnb data for analytics: vancouver listings',
'recommender systems: approaches & algorithms',
'airbnb data for analytics: mallorca reviews',
'analyze facebook data using ibm watson and watson studio',
'a tensorflow regression model to predict house values']
```

In []:

In [423]: *# 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'])) == set(['housing (2015): united states demography'])
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): united states demography'])
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 demography'])
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.0'])
```



```

assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-r
print("If this is all you see, you passed all of our tests! Nice job!")

```

If this is all you see, you passed all of our tests! Nice job!

4. Now we are going to improve the consistency of the **user_user_recs** function from above.

- Instead of arbitrarily choosing when we obtain users who are all the same closeness to a given user - choose the users that have the most total article interactions before choosing those with fewer article interactions.
- Instead of arbitrarily choosing articles from the user where the number of recommended articles starts below m and ends exceeding m, choose articles with the articles with the most total interactions before choosing those with fewer total interactions. This ranking should be what would be obtained from the **top_articles** function you wrote earlier.

```

In [426]: def get_top_sorted_users(user_id, df=df, user_item=user_item):
    '''
    INPUT:
    user_id - (int)
    df - (pandas dataframe) df as defined at the top of the notebook
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

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

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

    '''
    # Your code here

    similarity = user_item.dot(user_item.loc[user_id])
    neighbor_id = user_item.index.values
    num_interactions = df.user_id.value_counts().sort_index()

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

    neighbors_df = neighbors_df[neighbors_df.neighbor_id != user_id]

    #sort the neighbors_df by the similarity and then by number of interactions in des

```

```

neighbors_df.sort_values(by = ['similarity', 'num_interactions'], ascending=False,

return neighbors_df # Return the dataframe specified in the doc_string

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

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

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

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

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

    """
    # Your code here

    tops = get_top_sorted_users(user_id, df=df, user_item=user_item)['neighbor_id']
    article_ids, article_names = get_user_articles(user_id, user_item=user_item)

    recs = []

    for top in tops:
        ids, names = get_user_articles(top, user_item=user_item)
        new_recs = df[df.article_id.isin(ids)]
        new_recs = get_top_article_ids(len(ids), new_recs)

        for article_id in new_recs:
            if not article_id in article_ids:
                if len(recs) >= m:
                    break
                recs.append(article_id)

    if len(recs) >= m:
        break

```

```
rec_names = get_article_names(recs)
```

```
return recs, rec_names
```

```
In [427]: # 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:

```
['1330.0', '1427.0', '1364.0', '1170.0', '1162.0', '1304.0', '1351.0', '1160.0', '1354.0', '1368.0']
```

The top 10 recommendations for user 20 are the following article names:

```
['apache spark lab, part 1: basic concepts', 'predicting churn with the spss random tree algorithm']
```

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 [428]: get_top_sorted_users(1)['neighbor_id'].iloc[0] # Find the user that is most similar to user 1
```

```
Out[428]: 3933
```

```
In [429]: get_top_sorted_users(131)['neighbor_id'].iloc[9] # Find the 10th most similar user to user 131
```

```
Out[429]: 242
```

```
In [430]: ### Tests with a dictionary of results
```

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

```
In [431]: ## Dictionary Test Here
```

```
sol_5_dict = {
    'The user that is most similar to user 1.': user1_most_sim,
    'The user that is the 10th most similar to user 131': user131_10th_sim,
}
```

```
t.sol_5_test(sol_5_dict)
```

This all looks good! Nice job!

6. If we were given a new user, which of the above functions would you be able to use to make recommendations? Explain. Can you think of a better way we might make recommendations? Use the cell below to explain a better method for new users.

The above functions `get_top_articles` and `get_top_articles_ids` from Part II could be used to get the top articles to recommend to a new user. This problem described above is known as the cold start problem. It will not be possible to use User-User Based Collaborative Filtering Recommendations as information (such as ratings) on user-article interactions are not known for a new user, and this method is based on using the collaboration of user-item interactions.

Rank Based recommendations for new users is a better way of making recommendations.

Content-based similarity method for recommendations could be used for a new user. In this method, recommendations are based using information about the content of the articles and ranking of the highest ranked articles based on specific content or associated with some term. NLP can be used in this method. Content could be from the columns `doc_body`, `doc_description`, or `doc_full_name`.

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 [432]: get_top_article_ids(10)
```

```
Out[432]: ['1429.0',
           '1330.0',
           '1431.0',
           '1427.0',
           '1364.0',
           '1314.0',
           '1293.0',
           '1170.0',
           '1162.0',
           '1304.0']
```

```
In [433]: new_user = '0.0'
```

```
# What would your recommendations be for this new user '0.0'? As a new user, they have no history
# Provide a list of the top 10 article ids you would give to
new_user_recs = get_top_article_ids(10) # Your recommendations here
print(get_top_article_ids(10))
```

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

```
In [434]: assert set(new_user_recs) == set(['1314.0', '1429.0', '1293.0', '1427.0', '1162.0', '1364.0',
                                             '1330.0', '1431.0', '1304.0', '1429.0'])

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

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

1.1.4 Part IV: Content Based Recommendations (EXTRA - NOT REQUIRED)

Another method we might use to make recommendations is to perform a ranking of the highest ranked articles associated with some term. You might consider content to be the **doc_body**, **doc_description**, or **doc_full_name**. There isn't one way to create a content based recommendation, especially considering that each of these columns hold content related information.

1. Use the function body below to create a content based recommender. Since there isn't one right answer for this recommendation tactic, no test functions are provided. Feel free to change the function inputs if you decide you want to try a method that requires more input values. The input values are currently set with one idea in mind that you may use to make content based recommendations. One additional idea is that you might want to choose the most popular recommendations that meet your 'content criteria', but again, there is a lot of flexibility in how you might make these recommendations.

1.1.5 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
In [ ]: def make_content_recs():  
        '''  
        INPUT:  
  
        OUTPUT:  
  
        '''
```

2. Now that you have put together your content-based recommendation system, use the cell below to write a summary explaining how your content based recommender works. Do you see any possible improvements that could be made to your function? Is there anything novel about your content based recommender?

1.1.6 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

Write an explanation of your content based recommendation system here.

3. Use your content-recommendation system to make recommendations for the below scenarios based on the comments. Again no tests are provided here, because there isn't one right answer that could be used to find these content based recommendations.

1.1.7 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
In [ ]: # make recommendations for a brand new user  
  
        # make a recommendations for a user who only has interacted with article id '1427.0'
```

1.1.8 Part V: Matrix Factorization

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

1. You should have already created a **user_item** matrix above in **question 1** of **Part III** above. This first question here will just require that you run the cells to get things set up for the rest of **Part V** of the notebook.

```
In [435]: # Load the matrix here
```

```
user_item_matrix = pd.read_pickle('user_item_matrix.p')
```

```
In [436]: # quick look at the matrix
```

```
user_item_matrix.head()
```

```
Out[436]: article_id  0.0  100.0  1000.0  1004.0  1006.0  1008.0  101.0  1014.0  1015.0  \
user_id
1          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

article_id  1016.0  ...    977.0  98.0  981.0  984.0  985.0  986.0  990.0  \
user_id      ...
1          0.0  ...    0.0  0.0  1.0  0.0  0.0  0.0  0.0
2          0.0  ...    0.0  0.0  0.0  0.0  0.0  0.0  0.0
3          0.0  ...    1.0  0.0  0.0  0.0  0.0  0.0  0.0
4          0.0  ...    0.0  0.0  0.0  0.0  0.0  0.0  0.0
5          0.0  ...    0.0  0.0  0.0  0.0  0.0  0.0  0.0

article_id  993.0  996.0  997.0
user_id
1          0.0    0.0    0.0
2          0.0    0.0    0.0
3          0.0    0.0    0.0
4          0.0    0.0    0.0
5          0.0    0.0    0.0

[5 rows x 714 columns]
```

2. In this situation, you can use Singular Value Decomposition from [numpy](#) on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

```
In [437]: # 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 [438]: s.shape, u.shape, vt.shape
```

```
Out[438]: ((714,), (5149, 5149), (714, 714))
```

The SVD in the lesson did not converge as it had encountered a NaN or missing values in the user_movie_subset matrix. There was a significant missing data. "The number of ratings made for

user-movie pairs that didn't have ratings is 13 835 713". This means it was a very sparse matrix. SVD from NumPy does not work when the matrices do not have a value in every cell and is thus considered not complete. FunkSVD which uses a stochastic gradient descent formula was used in the lesson to address NaN/missing values. SVD in this Project is used on the user_item_matrix which has no missing values.

3. Now for the tricky part, how do we choose the number of latent features to use? Running the below cell, you can see that as the number of latent features increases, we obtain a lower error rate on making predictions for the 1 and 0 values in the user-item matrix. Run the cell below to get an idea of how the accuracy improves as we increase the number of latent features.

```
In [439]: num_latent_feats = np.arange(10,700+10,20)
          sum_errs = []

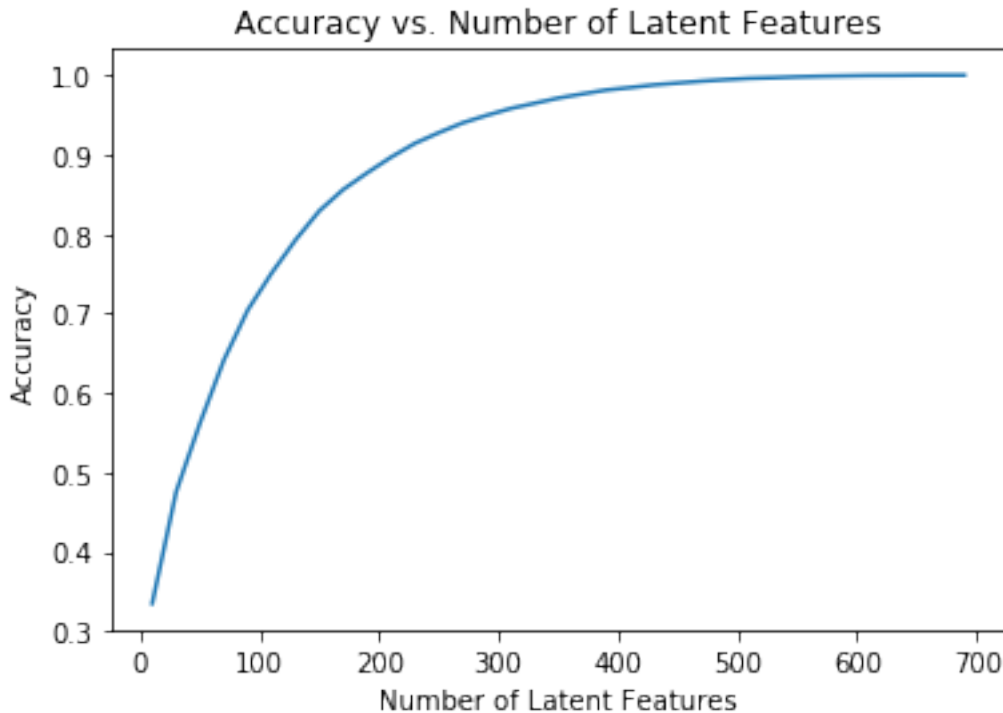
          for k in num_latent_feats:
              # restructure with k latent features
              s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]

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



4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are able to make good recommendations. Instead, we might split our dataset into a training and test set of data, as shown in the cell below.

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

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

```
In [440]: df_train = df.head(40000)
          df_test = df.tail(5993)

def create_test_and_train_user_item(df_train, df_test):
    """
    INPUT:
    df_train - training dataframe
    df_test - test dataframe

    OUTPUT:
    user_item_train - a user-item matrix of the training dataframe
```



```

        (unique users for each row and unique articles for each column)
    user_item_test - a user-item matrix of the testing dataframe
        (unique users for each row and unique articles for each column)
    test_idx - all of the test user ids
    test_arts - all of the test article ids

    '''
    # Your code here

    user_item_train = create_user_item_matrix(df_train)
    user_item_test = create_user_item_matrix(df_test)

    test_idx = user_item_test.index
    test_arts = user_item_test.columns

    return user_item_train, user_item_test, test_idx, test_arts

user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_user_item

In [441]: user_item_test.shape, user_item_train.shape, len(test_idx), len(test_arts)

Out[441]: ((682, 574), (4487, 714), 682, 574)

In [442]: # How many users can we make predictions for in the test set?
    len(np.intersect1d(user_item_test.index.values, user_item_train.index.values, assume_uniq

Out[442]: 20

In [443]: # How many users in the test set are we not able to make predictions for because of th
    len(np.setdiff1d(user_item_test.index.values, user_item_train.index.values, assume_uniq

Out[443]: 662

In [444]: # How many articles can we make predictions for in the test set?
    len(np.intersect1d(user_item_test.columns.values, user_item_train.columns.values, assu

Out[444]: 574

In [445]: # How many articles in the test set are we not able to make predictions for because of
    len(np.setdiff1d(user_item_test.columns.values, user_item_train.columns.values, assume

Out[445]: 0

In [446]: # Replace the values in the dictionary below
    a = 662
    b = 574
    c = 20
    d = 0

```

```

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

t.sol_4_test(sol_4_dict)

```

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

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

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

```

In [456]: # fit SVD on the user_item_train matrix
          u_train, s_train, vt_train = np.linalg.svd(user_item_train) # fit svd similar to above

```

```

In [457]: u_train.shape, s_train.shape, vt_train.shape

```

```

Out[457]: ((4487, 4487), (714,), (714, 714))

```

```

In [460]: #training subsets
          train_idx = user_item_train.index
          train_subset_idx = list(set(train_idx)&set(test_idx))
          train_subset_col = user_item_train.columns.intersection(test_arts)
          user_item_test = user_item_test.loc[train_subset_idx]

```

```

In [466]: u_test = u_train[user_item_train.index.isin(test_idx), :]
          vt_test = vt_train[:, user_item_train.columns.isin(test_arts)]
          u_test.shape, vt_test.shape

```

```

Out[466]: ((20, 4487), (714, 574))

```

```

In [462]: # Use these cells to see how well you can use the training
          # decomposition to predict on test data

```

```

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

```

```

          for k in num_latent_feats:

```

```

              # Restructure with k features

```

```

s_train_new, u_train_new, vt_train_new = np.diag(s_train[:k]), u_train[:, :k], vt_
s_test_new, u_test_new, vt_test_new = s_train_new, u_test[:, :k], vt_test[:k, :]

# take dot product
user_item_test_est = np.around(np.dot(np.dot(u_test_new, s_test_new), vt_test_new))
user_item_train_est = np.around(np.dot(np.dot(u_train_new, s_train_new), vt_train_

# compute error for each prediction to actual value
diffs_test = np.subtract(user_item_test, user_item_test_est)
diffs_train = np.subtract(user_item_train, user_item_train_est)

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

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

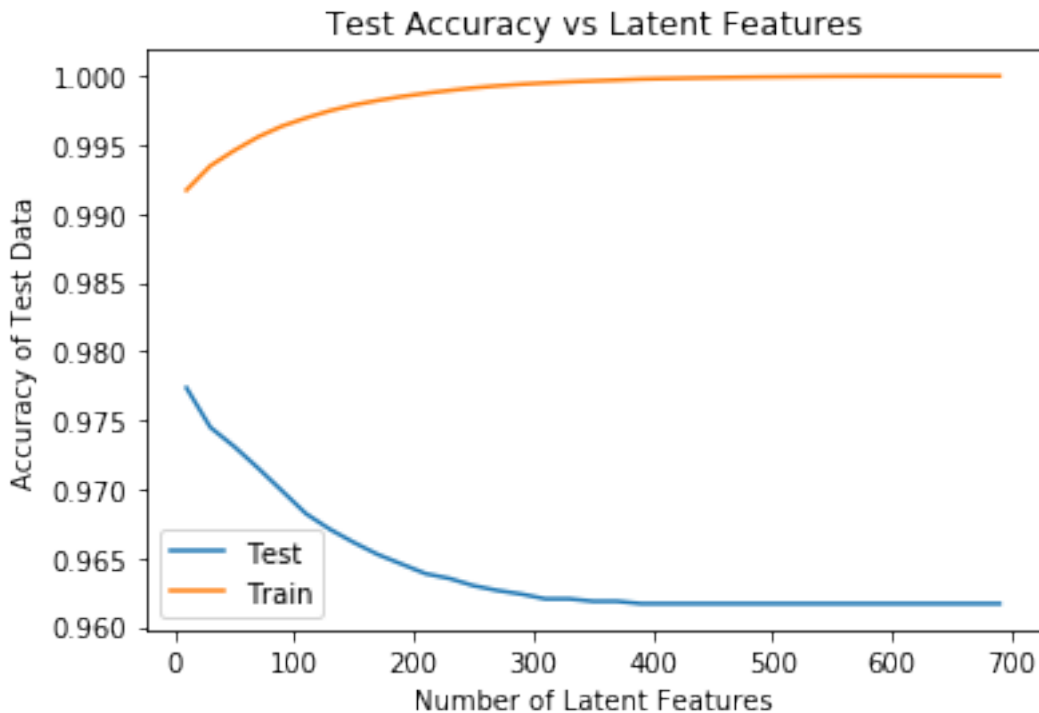
```

In [463]: # Plot test accuracy and latent features

```

plt.plot(num_latent_feats, 1 - np.array(sum_errs_test) / (user_item_test.shape[0]* use
plt.plot(num_latent_feats, 1 - np.array(sum_errs_train) / (user_item_train.shape[0]* u
plt.title('Test Accuracy vs Latent Features');
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy of Test Data');
plt.legend();

```



In []:

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

The training data suggests that as latent features increase the accuracy increases. However the test data suggests that accuracy decreased as latent features increased. There is direct conflict between the two. This difference could be due to the small overlap of users between training and testing datasets. Only 20 users in the test dataset are shared with the training dataset and were used to give the result.

We could increase the sizes of the test datasets, use online recommendation evaluations such as A/B testing, or time-based testing as opposed to matrix factorization used above which is an offline method.

All types of recommendations should be used not just user-user based collaborative and rank-based recommendations. Content-based like that described in Part III question (6) for new users and knowledge-based recommendations for existing could be used.

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

1.2 Conclusion

Congratulations! You have reached the end of the Recommendations with IBM project!

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

1.3 Directions to Submit

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

Alternatively, you can download this report as .html via the **File > Download as** sub-menu, 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 [467]: from subprocess import call
          call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

```
Out[467]: 0
```

```
In [ ]:
```