SI 671 - Homework 3 - Social Network Analysis

Ella Li

In [1]:
```
import pandas as pd
import numpy as np

# %pip install networkx
import networkx as nx
```

In [2]:
```
# %pip install --upgrade scipy --user
```

In [3]:
```
import warnings
warnings.filterwarnings("ignore")
```

Part 1: Exploratory Social Network Analysis [30 Points]

(a) Load the directed network graph (G) from the file amazonNetwork.csv. [2 points]

In [4]:
```
amazon_network = pd.read_csv('amazonNetwork.csv')
amazon_network.head()
```

Out[4]:

	FromNodeId	ToNodeId
0	1	2
1	1	4
2	1	5
3	1	15
4	2	11

In [5]:
```
G = nx.from_pandas_edgelist(amazon_network, source='FromNodeId',target='ToNodeId', create_using=nx.DiGraph())
```

In [6]:
```
G
```

Out[6]:
```
<networkx.classes.digraph.DiGraph at 0x1cf17f3d670>
```

(Each nodeid is one item, the data contains both start node and end note for each relationship.)

(b) How many items are present in the network and how many co-purchases happened?

In [7]:
```
print('how many items are present in the network is:', G.number_of_nodes())
print('how many co-purchases happened is:', G.number_of_edges())
```

```
how many items are present in the network is: 2647
how many co-purchases happened is: 10841
```

(as summary part said: if a product A is always co-purchased with product B, the graph contains a directed edge from A to B. So, the number of co-purchase can be calculated as number_of_edges)

(c) Compute the average shortest distance between the nodes in graph G. Explain your results briefly. [7 points]

In [8]:
```
print('the average shortest distance between the nodes in graph G is:', nx.average_shortest_path_length(G))
```

```
the average shortest distance between the nodes in graph G is: 9.592795477759587
```

(Here, "the average shortest distance between the nodes in graph G" means the shortest distance for all the nodes in graph G, and then take average. It is around 9.59, This shows that there are some nodes that need to pass through some other nodes to reach each other, i.e. only some of the items are bought together at a time, and may be different set in each time purchase.)

(d) Compute the transitivity and the average clustering coefficient of the network graph G. Explain your findings briefly based on the definitions of clustering coefficient and transitivity. [7 points]

In [9]:
```
# transitivity
print("transitivity of G is:", nx.transitivity(G))

# the average clustering coefficient
print("average clustering coefficient of G is:", nx.average_clustering(G))
```

```
transitivity of G is: 0.4339169154480595
average clustering coefficient of G is: 0.4086089178720651
```

Def of transitivity: the fraction of all possible triangles present in G. Possible triangles are identified by the number of "triads" (two edges with a shared vertex). T = 3*(#triangles/#triads);
Def of average clustering coefficient: $C = \frac{1}{n}\sum_{v \in G} c_v$, where n is the number of nodes in G;
Both Transitvity and Ave clustering coefficient measure the tendency for edges to form traiangles, but Transitivity weights nodes with large degree higher;
So, we can find that the 2 values are similar for our G, this makes perfect sense;

(e) Apply the PageRank algorithm to network G with damping value 0.5 and find the 10 nodes with the highest PageRank. Explain your findings briefly. NetworkX document of the PageRank algorithm: https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html [7 points]

In [10]:
```
# applying PageRank
pageranks = nx.pagerank(G, alpha=0.5)
# print(pageranks.items())
pageranks = {k:v for k, v in sorted(pageranks.items(), key=lambda t: t[1], reverse=True)}
```

In [11]:
```
# top 10 with the highest PageRank
highest_10_pagerank = list(pageranks.keys())[0:10]
count = 1
for node in highest_10_pagerank:
    print("{}. {}".format(count, node))
    count += 1
```

```
1. 8
2. 481
3. 33
4. 18
5. 23
6. 30
7. 346
8. 99
9. 93
10. 21
```

as above result shows, the 10 highest PageRank nodes are the most "important" nodes/items in our network, i.e., most frequent co-purchased items.

In [12]:
```
# also, explore dead ends
set(amazon_network.FromNodeId) == set(amazon_network.ToNodeId)
```

Out[12]:
```
False
```

some dead ends exist, means that some items will only be co-purchased with fixed items

Part 2: Predicting Review-Rating using Features derived from network properties

In [13]:
```
# read train and test dataset
review_train = pd.read_csv('reviewTrain.csv')
review_test = pd.read_csv('reviewTest.csv')
```

In [14]:
```
review_train
```

Out[14]:

	id	title	group	review
0	3	World War II Allied Fighter Planes Trading Cards	Book	5.0
1	5	Prayers That Avail Much for Business: Executive	Book	0.0
2	7	Batik	Music	4.5
3	10	The Edward Said Reader	Book	4.0
4	11	Resetting the Clock : Five Anti-Aging Hormone...	Book	5.0
...	...	...	...	...
1669	2667	Batman - The Animated Series - The Legend Beg...	DVD	0.0
1670	2670	Panatone: Warm	Music	4.5
1671	2671	Masculine Marine: Homoeroticism in the U.S. M...	Book	5.0
1672	2673	Storm	Music	4.5
1673	2674	Gold	Music	3.5

1674 rows × 4 columns

In [15]:
```
review_test
```

Out[15]:

	id	title	group	review
0	90	The Eagle Has Landed	Book	NaN
1	1372	Che in Africa: Che Guevara's Congo Diary	Book	NaN
2	1382	The Darwin Awards II : Unnatural Selection	Book	NaN
3	253	Celtic Glory	Music	NaN
4	671	Sublte Aromatherapy	Book	NaN
...	...	...	...	...
995	1097	Ahma	Music	NaN
996	1393	Loney Planet Chicago City Map (City Maps Series)	Book	NaN
997	643	Swell Style : A Girl's Guide to Turning Heads...	Book	NaN
998	976	Dark Continent : Europe's Twentieth Century	Book	NaN
999	961	Choice and Consequence	Book	NaN

1000 rows × 4 columns

In [16]:
```
print('number of unique items in train data:', review_train.id.nunique())
print('number of unique items in test data:', review_test.id.nunique())
```

```
number of unique items in train data: 1674
number of unique items in test data: 1000
```

In [82]:
```
# we need proper features for prediction, we want to use the network data in part_1 to generate some features from it
(start from suggested features):
# because these features are all improtant for network data, so I will include them all as our model features
```

In [18]:
```
# feature_1 - Page Rank
```

In [19]:
```
# pageranks
pageranks_df = pd.DataFrame.from_dict(pageranks, orient='index', columns=['page_rank'])
pageranks_df
```

Out[19]:

	page_rank
8	0.003625
481	0.002434
33	0.002297
18	0.002103
23	0.002079
...	...
2245	0.000220
1397	0.000220
1815	0.000219
1	0.000197
3	0.000197

2647 rows × 1 columns

In [20]:
```
# feature_2 - Clustering Coefficient
```

In [21]:
```
clustering = nx.clustering(G)
clustering_df = pd.DataFrame.from_dict(clustering, orient='index', columns=['clustering'])
clustering_df
```

Out[21]:

	clustering
1	0.000000
2	0.050000
4	0.188830
5	0.142157
15	0.128743
...	...
2542	0.277778
2549	0.166667
2545	0.184211
2546	0.000000
2548	0.300000

2647 rows × 1 columns

In [22]:
```
# feature_3 - Degree centrality
```

In [23]:
```
degree_centrality = nx.degree_centrality(G)
degree_centrality_df = pd.DataFrame.from_dict(degree_centrality, orient='index', columns=['degree_centrality'])
degree_centrality_df
```

Out[23]:

	degree_centrality
1	0.001512
2	0.001890
4	0.007559
5	0.005669
15	0.007181
...	...
2542	0.001890
2549	0.001134
2545	0.002646
2546	0.000378
2548	0.001512

2647 rows × 1 columns

In [24]:
```
# feature_4 - Closeness centrality
```

In [25]:
```
closeness_centrality = nx.closeness_centrality(G)
closeness_centrality_df = pd.DataFrame.from_dict(closeness_centrality, orient='index', columns=['closeness_centrality'])
closeness_centrality_df
```

Out[25]:

	closeness_centrality
1	0.000000
2	0.000378
4	0.065922
5	0.133688
15	0.051976
...	...
2542	0.063380
2549	0.061987
2545	0.062745
2546	0.062794
2548	0.058241

2647 rows × 1 columns

In [26]:
```
# feature_5 - Betweenness centrality
```

In [27]:
```
betweenness_centrality = nx.betweenness_centrality(G)
betweenness_centrality_df = pd.DataFrame.from_dict(betweenness_centrality, orient='index', columns=
```

```
['betweenness_centrality'])
betweenness_centrality_df
```

Out[27]:

	betweenness_centrality
1	0.000000e+00
2	8.384453e-05
4	5.048816e-03
5	4.031723e-03
15	3.746212e-02
...	...
2542	2.614739e-04
2549	8.334917e-08
2545	7.222845e-04
2546	0.000000e+00
2548	7.144215e-08

2647 rows × 1 columns

In [28]:
```
# after we generated 5 features, we want to combine/merge these features with our train data
```

In [29]:
```
# check for data merge: train review data set has some extra nodes, which the amazon network data does not have
len((set(review_train.id)) - set(amazon_network.FromNodeId).union(set(amazon_network.ToNodeId)))
```

Out[29]:
```
21
```

We find train review data set has 21 extra nodes, so we will use left merge to keep info, and then, we will drop missing values (for future model training)

In [30]:
```
# here, we use left merge on node_id, for 5 features
review_train = review_train.merge(pageranks_df, left_on='id', right_index=True, how='left')
review_train = review_train.merge(clustering_df, left_on='id', right_index=True, how='left')
review_train = review_train.merge(degree_centrality_df, left_on='id', right_index=True, how='left')
review_train = review_train.merge(closeness_centrality_df, left_on='id', right_index=True, how='left')
review_train = review_train.merge(betweenness_centrality_df, left_on='id', right_index=True, how='left')
review_train
```

Out[30]:

	id	title	group	review	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
0	3	World War II Allied Fighter Planes Trading Cards	Book	5.0	0.000197	0.450000	0.001890	0.000000	0.000000
1	5	Prayers That Avail Much for Business: Executive	Book	0.0	0.000774	0.142157	0.005669	0.133688	0.004032
2	7	Batik	Music	4.5	0.001263	0.109562	0.008692	0.150353	0.018768
3	10	The Edward Said Reader	Book	4.0	0.000424	0.285714	0.003779	0.116834	0.003049
4	11	Resetting the Clock : Five Anti-Aging Hormone...	Book	5.0	0.000906	0.120344	0.010204	0.008231	0.008756
...	...	...	...	...	...	...	...	...	...
1669	2667	Batman - The Animated Series - The Legend Beg...	DVD	0.0	NaN	NaN	NaN	NaN	NaN
1670	2670	Panatone: Warm	Music	4.5	NaN	NaN	NaN	NaN	NaN
1671	2671	Masculine Marine: Homoeroticism in the U.S. M...	Book	5.0	NaN	NaN	NaN	NaN	NaN
1672	2673	Storm	Music	4.5	NaN	NaN	NaN	NaN	NaN
1673	2674	Gold	Music	3.5	NaN	NaN	NaN	NaN	NaN

1674 rows × 9 columns

In [31]:
```
# check missing values
review_train.isna().any()
```

Out[31]:
```
id                     False
title                  False
group                  False
review                 False
page_rank               True
clustering              True
degree_centrality       True
closeness_centrality    True
betweenness_centrality  True
dtype: bool
```

In [32]:
```
# drop these missing values
review_train.dropna(inplace=True)
```

In [33]:
```
review_train
```

Out[33]:

	id	title	group	review	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
0	3	World War II Allied Fighter Planes Trading Cards	Book	5.0	0.000197	0.450000	0.001890	0.000000	0.000000e+00
1	5	Prayers That Avail Much for Business: Executive	Book	0.0	0.000774	0.142157	0.005669	0.133688	4.031723e-03
2	7	Batik	Music	4.5	0.001263	0.109562	0.008692	0.150353	1.876848e-02
3	10	The Edward Said Reader	Book	4.0	0.000424	0.285714	0.003779	0.116834	3.049242e-03
4	11	Resetting the Clock : Five Anti-Aging Hormone...	Book	5.0	0.000906	0.120344	0.010204	0.008231	8.756193e-03
...	...	...	...	...	...	...	...	...	...
1648	2635	Duckling (Jumbo Animal Shaped Board Books)	Book	0.0	0.000227	0.000000	0.001512	0.057290	3.284255e-02
1649	2638	Comprehensive Curriculum of Basic Skills: Gra...	Book	4.5	0.000257	0.392857	0.002268	0.057311	1.488339e-03
1650	2641	Christian Ethics	Book	4.0	0.000236	0.888889	0.001890	0.057313	0.000000e+00
1651	2642	Social, Emotional, and Personality Developmen...	Book	5.0	0.000236	0.333333	0.001134	0.057332	2.524420e-04
1652	2644	MARC/AACR2/Authority Control Tagging: Blitz C...	Book	2.0	0.000398	0.777778	0.002646	0.057321	2.143264e-07

1653 rows × 9 columns

In [34]:
```
review_train['group'].unique()
```

Out[34]:
```
array([' Book', ' Music', ' DVD', ' Video', ' Toy'], dtype=object)
```

In [35]:
```
review_test['group'].unique()
```

Out[35]:
```
array([' Book', ' Music', ' Video', ' DVD'], dtype=object)
```

In [36]:
```
# encode categorical feature_6 "group"

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

transformer = ColumnTransformer([("hot_enc", OneHotEncoder(), ['group'])], remainder="passthrough")
review_train_enc = pd.DataFrame(transformer.fit_transform(review_train))
review_train_enc.columns = transformer.get_feature_names()
review_train_enc
```

Out[36]:

	hot_enc__x0_ Book	hot_enc__x0_ DVD	hot_enc__x0_ Music	hot_enc__x0_ Toy	hot_enc__x0_ Video	id	title	review	page_rank	clustering	degree_centrality	closeness
0	1.0	0.0	0.0	0.0	0.0	3	World War II Allied Fighter Planes Trading Cards	5.0	0.000197	0.45	0.00189	
1	1.0	0.0	0.0	0.0	0.0	5	Prayers That Avail Much for Business: Executive	0.0	0.000774	0.142157	0.005669	
2	0.0	0.0	1.0	0.0	0.0	7	Batik	4.5	0.001263	0.109562	0.008692	
3	1.0	0.0	0.0	0.0	0.0	10	The Edward Said Reader	4.0	0.000424	0.285714	0.003779	
4	1.0	0.0	0.0	0.0	0.0	11	Resetting the Clock : Five Anti-Aging Hormone...	5.0	0.000906	0.120344	0.010204	
...	...	...	...	...	...	...	...	...	...	...	...	
1648	1.0	0.0	0.0	0.0	0.0	2635	Duckling (Jumbo Animal Shaped Board Books)	0.0	0.000227	0.0	0.001512	
1649	1.0	0.0	0.0	0.0	0.0	2638	Comprehensive Curriculum of Basic Skills: Gra...	4.5	0.000257	0.392857	0.002268	
1650	1.0	0.0	0.0	0.0	0.0	2641	Christian Ethics	4.0	0.000236	0.888889	0.00189	
1651	1.0	0.0	0.0	0.0	0.0	2642	Social, Emotional, and Personality Developmen...	5.0	0.000236	0.333333	0.001134	
1652	1.0	0.0	0.0	0.0	0.0	2644	MARC/AACR2/Authority Control Tagging: Blitz C...	2.0	0.000398	0.777778	0.002646	

1653 rows × 13 columns

We will also drop "hot*enc__x0* Toy" because test data does not contain this "group", so we will not use this as one of our features

In [37]:
```
# assign X, y
```

```
train_X = review_train_enc.drop(['review','title','id', 'hot_enc__x0_ Toy'], axis=1)
train_y = review_train_enc['review']
```

In [38]:
```
train_X
```

Out[38]:

	hot_enc__x0_ Book	hot_enc__x0_ DVD	hot_enc__x0_ Music	hot_enc__x0_ Video	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
0	1.0	0.0	0.0	0.0	0.000197	0.45	0.00189	0.0	0.0
1	1.0	0.0	0.0	0.0	0.000774	0.142157	0.005669	0.133688	0.004032
2	0.0	0.0	1.0	0.0	0.001263	0.109562	0.008692	0.150353	0.018768
3	1.0	0.0	0.0	0.0	0.000424	0.285714	0.003779	0.116834	0.003049
4	1.0	0.0	0.0	0.0	0.000906	0.120344	0.010204	0.008231	0.008756
...	...	...	...	...	...	...	...	...	...
1648	1.0	0.0	0.0	0.0	0.000227	0.0	0.001512	0.05729	0.032843
1649	1.0	0.0	0.0	0.0	0.000257	0.392857	0.002268	0.057311	0.001488
1650	1.0	0.0	0.0	0.0	0.000236	0.888889	0.00189	0.057313	0.0
1651	1.0	0.0	0.0	0.0	0.000236	0.333333	0.001134	0.057332	0.000252
1652	1.0	0.0	0.0	0.0	0.000398	0.777778	0.002646	0.057321	0.0

1653 rows × 9 columns

In [39]:
```
train_y
```

Out[39]:
```
0       5.0
1       0.0
2       4.5
3       4.0
4       5.0
       ...
1648    0.0
1649    4.5
1650    4.0
1651    5.0
1652    2.0
Name: review, Length: 1653, dtype: object
```

now, we have features, so start to train model and select models + tuning, start from suggested models:

- Logistic Regression
- Support Vector Machine (SVM)
- Multi-layer perceptron

In [40]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
```

In [41]:
```
# split the whole train dataset to train&valid set for MAE calculation
```

In [42]:
```
X_train, X_valid, y_train, y_valid = train_test_split(train_X, train_y, test_size=0.2, random_state=0)
```

In [43]:
```
# # LogisticReg
# logreg = LogisticRegression(random_state=0)
# logreg.fit(X_train, y_train)
# y_valid_pred = logreg.predict(X_valid)

# mean_absolute_error(y_valid, y_valid_pred)
```

Logistic Regression MAE result is not great, drop

In [44]:
```
X_train
```

Out[44]:

	hot_enc__x0_ Book	hot_enc__x0_ DVD	hot_enc__x0_ Music	hot_enc__x0_ Video	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
1422	1.0	0.0	0.0	0.0	0.000391	0.585366	0.003779	0.061473	0.000217
319	1.0	0.0	0.0	0.0	0.000399	0.264706	0.003401	0.087177	0.00236
711	0.0	0.0	1.0	0.0	0.000537	0.445946	0.004913	0.052357	0.00081
867	1.0	0.0	0.0	0.0	0.000323	0.538462	0.003023	0.066398	0.000163
1607	1.0	0.0	0.0	0.0	0.00023	0.5	0.002268	0.078386	0.0
...	...	...	...	...	...	...	...	...	...
763	1.0	0.0	0.0	0.0	0.000281	0.75	0.002268	0.07946	0.0
835	1.0	0.0	0.0	0.0	0.000227	0.111111	0.00189	0.061179	0.000687
1216	0.0	0.0	1.0	0.0	0.000388	1.0	0.003023	0.049195	0.0
559	0.0	0.0	0.0	1.0	0.000348	0.147059	0.003401	0.066639	0.022544
684	1.0	0.0	0.0	0.0	0.000227	0.3	0.002268	0.053234	0.000043

1322 rows × 9 columns

In [45]:

```
# scale for mlp&svr regressor

from sklearn.preprocessing import StandardScaler

std = StandardScaler()

X_train_scaled = pd.DataFrame(std.fit_transform(X_train), index = X_train.index, columns = X_train.columns)
X_valid_scaled = pd.DataFrame(std.transform(X_valid), index = X_valid.index, columns = X_valid.columns)
X_train_scaled
```

Out[45]:

	hot_enc__x0_ Book	hot_enc__x0_ DVD	hot_enc__x0_ Music	hot_enc__x0_ Video	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
1422	0.610568	-0.202348	-0.478147	-0.218045	0.043397	0.699126	0.361947	-0.511147	-0.345633
319	0.610568	-0.202348	-0.478147	-0.218045	0.083554	-0.495789	0.141646	0.641069	-0.131002
711	-1.637819	-0.202348	2.091407	-0.218045	0.796668	0.179589	1.022850	-0.919802	-0.286246
867	0.610568	-0.202348	-0.478147	-0.218045	-0.312224	0.524341	-0.078655	-0.290385	-0.351051
1607	0.610568	-0.202348	-0.478147	-0.218045	-0.794860	0.381017	-0.519257	0.247011	-0.367295
...	...	...	...	...	...	...	...	...	...
763	0.610568	-0.202348	-0.478147	-0.218045	-0.527159	1.312623	-0.519257	0.295157	-0.367341
835	0.610568	-0.202348	-0.478147	-0.218045	-0.806735	-1.068148	-0.739558	-0.524320	-0.298560
1216	-1.637819	-0.202348	2.091407	-0.218045	0.028226	2.244229	-0.078655	-1.061503	-0.367341
559	-1.637819	-0.202348	-0.478147	4.586211	-0.182133	-0.934191	0.141646	-0.279559	1.890634
684	0.610568	-0.202348	-0.478147	-0.218045	-0.809591	-0.364268	-0.519257	-0.880475	-0.363058

1322 rows × 9 columns

In [46]:

```
# MLP
mlp_reg = MLPRegressor(random_state=0)
mlp_reg.fit(X_train_scaled, y_train)
y_valid_pred = mlp_reg.predict(X_valid_scaled)

mean_absolute_error(y_valid, y_valid_pred)
```

Out[46]:

```
1.6356455532036933
```

In [47]:

```
# SVR

svr_reg = SVR(kernel='linear')
svr_reg.fit(X_train_scaled, y_train)
y_valid_pred = svr_reg.predict(X_valid_scaled)

mean_absolute_error(y_valid, y_valid_pred)
```

Out[47]:

```
1.3590956627459083
```

SVR has better MAE result, so I will choose this model and do further tuning: (use GridSearchCV to perform 5-fold cv search to tune best parameters)

In [48]:

```
# SVR tuning
```

```
param_grid = {'kernel':('linear', 'poly', 'rbf', 'sigmoid'), 'C':[1,5,10], 'degree': [3,8],'gamma' : ('auto','scale')}


grid = GridSearchCV(svr_reg, param_grid, n_jobs= -1, cv=5, scoring='neg_mean_absolute_error')
grid.fit(X_train_scaled, y_train)

print(grid.best_params_)
# default refit=True, so we can get best prediction with best parameters automaticality
y_valid_pred_best = grid.predict(X_valid_scaled)

mean_absolute_error(y_valid, y_valid_pred_best)
```

Out[48]:
```
{'C': 1, 'degree': 3, 'gamma': 'auto', 'kernel': 'linear'}
1.3590956627459083
```

So, this is our final model and parameters,

SVR model
{'C': 1, 'degree': 3, 'gamma': 'auto', 'kernel': 'linear'}

In [49]:
```
# # mlp tuning
# param_grid = {
#     'hidden_layer_sizes': [(150,100,50), (120,80,40), (100,50,30), (50,50,50), (50,100,50)],
#     'max_iter': [50, 100, 500, 700, 1000],
#     'activation': ['logistic','tanh','relu'],
#     'solver': ['sgd', 'adam'],
#     'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 1, 10],
#     'learning_rate': ['constant','adaptive'],
# }

# grid = GridSearchCV(mlp_reg, param_grid, n_jobs= -1, cv=5, scoring='neg_mean_absolute_error')
# grid.fit(X_train_scaled, y_train)

# print(grid.best_params_)
# # default refit=True, so we can get best prediction with best parameters automaticality
# y_valid_pred_best = grid.predict(X_valid_scaled)

# mean_absolute_error(y_valid, y_valid_pred_best)
```

In [ ]:
```

```

In [50]:
```
# read test.csv to get prediction results and fill in csv.file
```

In [65]:
```
review_test = pd.read_csv('reviewTest.csv')
review_test
```

Out[65]:

	id	title	group	review
0	90	The Eagle Has Landed	Book	NaN
1	1372	Che in Africa: Che Guevara's Congo Diary	Book	NaN
2	1382	The Darwin Awards II : Unnatural Selection	Book	NaN
3	253	Celtic Glory	Music	NaN
4	671	Sublte Aromatherapy	Book	NaN
...	...	...	...	...
995	1097	Ahma	Music	NaN
996	1393	Loney Planet Chicago City Map (City Maps Series)	Book	NaN
997	643	Swell Style : A Girl's Guide to Turning Heads...	Book	NaN
998	976	Dark Continent : Europe's Twentieth Century	Book	NaN
999	961	Choice and Consequence	Book	NaN

1000 rows × 4 columns

In [66]:
```
# preprocessiong data (same as the process for train_data.csv)
```

```
len((set(review_test.id)) - set(amazon_network.FromNodeId).union(set(amazon_network.ToNodeId)))
```

Out[66]:
```
6
```

We find test review data set has 6 extra nodes, so we will use left merge to keep info, and then, we will drop missing values

In [67]:
```
review_test = review_test.merge(pageranks_df, left_on='id', right_index=True, how='left')
review_test = review_test.merge(clustering_df, left_on='id', right_index=True, how='left')
review_test = review_test.merge(degree_centrality_df, left_on='id', right_index=True, how='left')
review_test = review_test.merge(closeness_centrality_df, left_on='id', right_index=True, how='left')
review_test = review_test.merge(betweenness_centrality_df, left_on='id', right_index=True, how='left')
review_test
```

Out[67]:

	id	title	group	review	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
0	90	The Eagle Has Landed	Book	NaN	0.000347	0.250000	0.003779	0.116428	3.048563e-02
1	1372	Che in Africa: Che Guevara's Congo Diary	Book	NaN	0.000300	0.288462	0.003023	0.080232	9.535202e-03
2	1382	The Darwin Awards II : Unnatural Selection	Book	NaN	0.000338	0.750000	0.003401	0.063412	3.095826e-07
3	253	Celtic Glory	Music	NaN	0.000268	0.750000	0.002268	0.072458	1.031101e-04
4	671	Sublte Aromatherapy	Book	NaN	0.000358	0.562500	0.003401	0.093620	7.927132e-04
...	...	...	...	...	...	...	...	...	...
995	1097	Ahma	Music	NaN	0.000454	0.414634	0.003779	0.064222	1.646917e-03
996	1393	Loney Planet Chicago City Map (City Maps Series)	Book	NaN	0.000327	0.315789	0.002646	0.078876	6.684665e-06
997	643	Swell Style : A Girl's Guide to Turning Heads...	Book	NaN	0.000398	0.550000	0.003779	0.093544	1.315945e-03
998	976	Dark Continent : Europe's Twentieth Century	Book	NaN	0.001183	0.101604	0.010582	0.085334	9.521942e-03
999	961	Choice and Consequence	Book	NaN	0.000494	0.328125	0.004535	0.087325	1.983496e-03

1000 rows × 9 columns

In [68]:
```
review_test['group'].unique()
```

Out[68]:
```
array([' Book', ' Music', ' Video', ' DVD'], dtype=object)
```

In [69]:
```
# encode categorical feature_6 "group"

transformer = ColumnTransformer([("hot_enc", OneHotEncoder(), ['group'])], remainder="passthrough")
review_test_enc = pd.DataFrame(transformer.fit_transform(review_test))
review_test_enc.columns = transformer.get_feature_names()
review_test_enc
```

Out[69]:

	hot_enc__x0_ Book	hot_enc__x0_ DVD	hot_enc__x0_ Music	hot_enc__x0_ Video	id	title	review	page_rank	clustering	degree_centrality	closeness_centrality	betweenness
0	1.0	0.0	0.0	0.0	90	The Eagle Has Landed	NaN	0.000347	0.25	0.003779	0.116428	
1	1.0	0.0	0.0	0.0	1372	Che in Africa: Che Guevara's Congo Diary	NaN	0.0003	0.288462	0.003023	0.080232	
2	1.0	0.0	0.0	0.0	1382	The Darwin Awards II : Unnatural Selection	NaN	0.000338	0.75	0.003401	0.063412	
3	0.0	0.0	1.0	0.0	253	Celtic Glory	NaN	0.000268	0.75	0.002268	0.072458	
4	1.0	0.0	0.0	0.0	671	Sublte Aromatherapy	NaN	0.000358	0.5625	0.003401	0.09362	
...	...	...	...	...	...	...	...	...	...	...	...	...
995	0.0	0.0	1.0	0.0	1097	Ahma	NaN	0.000454	0.414634	0.003779	0.064222	
996	1.0	0.0	0.0	0.0	1393	Loney Planet Chicago City Map (City Maps Series)	NaN	0.000327	0.315789	0.002646	0.078876	
997	1.0	0.0	0.0	0.0	643	Swell Style : A Girl's Guide to Turning Heads...	NaN	0.000398	0.55	0.003779	0.093544	
998	1.0	0.0	0.0	0.0	976	Dark Continent : Europe's Twentieth Century	NaN	0.001183	0.101604	0.010582	0.085334	
999	1.0	0.0	0.0	0.0	961	Choice and Consequence	NaN	0.000494	0.328125	0.004535	0.087325	

1000 rows × 12 columns

In [70]:
```
# assign X
test_X = review_test_enc.drop(['review','title','id'], axis=1)
```

In [71]:
```
test_X
```

Out[71]:

	hot_enc__x0_ Book	hot_enc__x0_ DVD	hot_enc__x0_ Music	hot_enc__x0_ Video	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
0	1.0	0.0	0.0	0.0	0.000347	0.25	0.003779	0.116428	0.030486
1	1.0	0.0	0.0	0.0	0.0003	0.288462	0.003023	0.080232	0.009535
2	1.0	0.0	0.0	0.0	0.000338	0.75	0.003401	0.063412	0.0
3	0.0	0.0	1.0	0.0	0.000268	0.75	0.002268	0.072458	0.000103
4	1.0	0.0	0.0	0.0	0.000358	0.5625	0.003401	0.09362	0.000793
...	...	...	...	...	...	...	...	...	...
995	0.0	0.0	1.0	0.0	0.000454	0.414634	0.003779	0.064222	0.001647
996	1.0	0.0	0.0	0.0	0.000327	0.315789	0.002646	0.078876	0.000007
997	1.0	0.0	0.0	0.0	0.000398	0.55	0.003779	0.093544	0.001316
998	1.0	0.0	0.0	0.0	0.001183	0.101604	0.010582	0.085334	0.009522
999	1.0	0.0	0.0	0.0	0.000494	0.328125	0.004535	0.087325	0.001983

1000 rows × 9 columns

In [72]:
```
# check missing values
test_X.isna().any()
```

Out[72]:
```
hot_enc__x0_ Book         False
hot_enc__x0_ DVD          False
hot_enc__x0_ Music        False
hot_enc__x0_ Video        False
page_rank                  True
clustering                 True
degree_centrality          True
closeness_centrality       True
betweenness_centrality     True
dtype: bool
```

In [73]:
```
# fill these missing values with 0
test_X.fillna(0, inplace=True)
```

In [73]:
```
test_X
```

Out[73]:

	hot_enc__x0_ Book	hot_enc__x0_ DVD	hot_enc__x0_ Music	hot_enc__x0_ Video	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
0	1.0	0.0	0.0	0.0	0.000347	0.250000	0.003779	0.116428	3.048563e-02
1	1.0	0.0	0.0	0.0	0.000300	0.288462	0.003023	0.080232	9.535202e-03
2	1.0	0.0	0.0	0.0	0.000338	0.750000	0.003401	0.063412	3.095826e-07
3	0.0	0.0	1.0	0.0	0.000268	0.750000	0.002268	0.072458	1.031101e-04
4	1.0	0.0	0.0	0.0	0.000358	0.562500	0.003401	0.093620	7.927132e-04
...	...	...	...	...	...	...	...	...	...
995	0.0	0.0	1.0	0.0	0.000454	0.414634	0.003779	0.064222	1.646917e-03
996	1.0	0.0	0.0	0.0	0.000327	0.315789	0.002646	0.078876	6.684665e-06
997	1.0	0.0	0.0	0.0	0.000398	0.550000	0.003779	0.093544	1.315945e-03
998	1.0	0.0	0.0	0.0	0.001183	0.101604	0.010582	0.085334	9.521942e-03
999	1.0	0.0	0.0	0.0	0.000494	0.328125	0.004535	0.087325	1.983496e-03

1000 rows × 9 columns

In [74]:
```
test_X.isna().any()
```

Out[74]:
```
hot_enc__x0_ Book          False
hot_enc__x0_ DVD           False
hot_enc__x0_ Music         False
hot_enc__x0_ Video         False
page_rank                  False
clustering                 False
degree_centrality          False
closeness_centrality       False
betweenness_centrality     False
dtype: bool
```

In [75]:
```
# scale
test_X_scaled = pd.DataFrame(std.transform(test_X), index = test_X.index, columns = test_X.columns)
test_X_scaled
```

Out[75]:

	hot_enc__x0_ Book	hot_enc__x0_ DVD	hot_enc__x0_ Music	hot_enc__x0_ Video	page_rank	clustering	degree_centrality	closeness_centrality	betweenness_centrality
0	0.610568	-0.202348	-0.478147	-0.218045	-0.184852	-0.550589	0.361947	1.952311	2.686079
1	0.610568	-0.202348	-0.478147	-0.218045	-0.432318	-0.407265	-0.078655	0.329782	0.587698
2	0.610568	-0.202348	-0.478147	-0.218045	-0.232516	1.312623	0.141646	-0.424202	-0.367310
3	-1.637819	-0.202348	2.091407	-0.218045	-0.597469	1.312623	-0.519257	-0.018710	-0.357013
4	0.610568	-0.202348	-0.478147	-0.218045	-0.130080	0.613918	0.141646	0.929886	-0.287943
...	...	...	...	...	...	...	...	...	...
995	-1.637819	-0.202348	2.091407	-0.218045	0.367966	0.062908	0.361947	-0.387911	-0.202387
996	0.610568	-0.202348	-0.478147	-0.218045	-0.292249	-0.305430	-0.298956	0.268986	-0.366671
997	0.610568	-0.202348	-0.478147	-0.218045	0.076686	0.567338	0.361947	0.926502	-0.235537
998	0.610568	-0.202348	-0.478147	-0.218045	4.151470	-1.103574	4.327363	0.558490	0.586370
999	0.610568	-0.202348	-0.478147	-0.218045	0.577807	-0.259462	0.802549	0.647736	-0.168675

1000 rows × 9 columns

In [76]:
```
# use our selected model to predict


test_pred = grid.predict(test_X_scaled)
test_pred
```

Out[76]:

```
array([4.09539866, 4.04004939, 4.01098166, 4.48985048, 3.9691002 ,
       3.93080315, 4.44981837, 3.98966302, 4.06263519, 4.49641858,
       3.95055563, 4.09517233, 4.00432567, 4.06223454, 4.03141852,
       4.31292168, 4.36385878, 3.94306342, 3.9305065 , 3.96181945,
       3.87637604, 4.01202861, 4.16152117, 4.05231527, 4.47744393,
       4.07433529, 3.92352571, 4.35721555, 4.50373635, 4.45337999,
       3.97118216, 4.14773023, 4.49347479, 4.10167857, 4.63632845,
       4.01096847, 3.94745217, 4.04705817, 3.98778767, 4.07398871,
       3.8964986 , 3.95160118, 4.1027866 , 4.01042104, 3.92407618,
       4.04093286, 3.95686282, 3.95018377, 3.99878355, 3.92742955,
       4.0180586 , 3.98909366, 4.05913223, 4.01198378, 3.95114198,
       4.40862212, 3.97280877, 4.07080667, 3.96204228, 4.44348106,
       4.01830272, 3.94166694, 4.20921194, 3.62866302, 3.98829441,
       4.13986078, 4.0709272 , 3.97998229, 4.02713512, 4.47227619,
       4.46211276, 3.62001922, 4.47697519, 3.95002477, 3.95424729,
       4.46570715, 3.97448332, 3.89942453, 3.95036474, 3.75905084,
       3.99148629, 4.04087561, 4.46249353, 4.05413509, 4.01776109,
       4.38501311, 4.48025027, 4.0022613 , 4.12937223, 4.0204008 ,
       3.91918047, 4.1444372 , 3.95856581, 3.97607869, 3.94705606,
       4.03646033, 3.90462247, 3.99220103, 3.84510231, 4.39480493,
       3.92191284, 4.39975058, 4.0298016 , 4.3775645 , 3.96441528,
       4.06253519, 4.15000965, 4.0895576 , 3.96834029, 3.94201454,
       4.40835466, 4.04549445, 4.13657605, 4.010124  , 4.42734674,
       4.49996921, 4.11137314, 3.95164253, 4.01467187, 4.16071441,
       3.98481905, 4.05021829, 4.02458819, 3.97847288, 4.08624951,
       4.14342451, 3.91408536, 4.01001076, 3.99925718, 4.02303247,
       4.42613229, 4.02648039, 3.99439043, 4.00448407, 3.94759467,
       3.96315776, 4.07644818, 4.44045471, 3.9501285 , 4.02463078,
       4.34305958, 3.95159038, 4.38576696, 4.40548098, 4.45573211,
       4.63632845, 3.89047713, 4.52274546, 3.95067385, 3.97824487,
       4.06537881, 4.08265483, 4.06800053, 4.45046222, 4.03679627,
       4.01109718, 4.49874206, 4.41187387, 4.03238702, 4.05108382,
       4.06993076, 4.4353766 , 4.00899153, 4.05067731, 4.04264265,
       4.06953135, 3.95165687, 4.05221185, 4.31686593, 4.58078375,
       4.05610399, 4.06530638, 4.04292093, 4.16379424, 4.06921203,
       4.06294137, 4.0722052 , 3.93587315, 3.99778122, 4.04260233,
       4.38667032, 4.09942746, 3.95512489, 4.01838961, 3.84739936,
       3.97019569, 4.0505413 , 4.04166582, 4.48480212, 4.41615703,
       4.06459788, 4.06275811, 4.3608579 , 3.97056233, 4.01288903,
       4.54064957, 4.04110254, 4.06548775, 4.00599957, 4.01496562,
       4.1067552 , 4.48031413, 4.03547862, 4.42578973, 4.12097656,
       3.85518866, 4.04903353, 3.96756137, 4.33683442, 4.47204985,
       4.09667087, 4.4554614 , 4.49500941, 4.05222902, 3.99126023,
       4.04248467, 3.8274885 , 4.08830012, 4.08611071, 3.99218127,
       4.20890374, 4.07538319, 3.93184135, 3.95849762, 4.07641147,
       4.30013028, 4.00233426, 4.03498639, 3.99424917, 3.94690924,
       4.41051076, 3.86236965, 4.007763  , 3.89435384, 4.08880689,
       4.03514984, 4.0592069 , 4.3413366 , 4.0575283 , 3.93262742,
       4.20971049, 3.61248174, 4.02427413, 4.02365717, 4.56829494,
       4.04026556, 4.08062679, 4.10530625, 4.08600543, 3.99809897,
       3.96389744, 3.94536891, 4.06633024, 4.24119419, 3.88562922,
       4.07674559, 4.00269436, 4.5584441 , 3.9694907 , 3.96033246,
       3.81723303, 4.04823603, 4.07405844, 3.77163821, 4.40745987,
       3.99146953, 4.09118862, 3.97534363, 3.93848282, 4.33079735,
       3.95749212, 3.99229339, 3.98963806, 4.13013947, 4.0231701 ,
       3.99179695, 4.02664579, 3.98777686, 4.11647245, 3.75339004,
       4.02679407, 3.9846249 , 4.04099048, 3.65840003, 4.40729933,
       4.065922  , 3.95729136, 3.88935177, 3.86772272, 3.81910338,
       4.02727905, 3.61575059, 3.90839082, 4.08861525, 4.07012164,
       4.10177633, 4.42990997, 4.02878606, 3.7978184 , 4.47771049,
       4.39340859, 3.93685145, 3.65832492, 4.03135702, 4.11019946,
       3.93773266, 4.16765491, 4.02281013, 4.4295492 , 3.94347054,
       4.04686436, 4.51365444, 3.99315777, 3.90000522, 3.99994389,
       3.84441994, 3.99131998, 3.91693984, 4.45606488, 4.03904914,
       3.89322661, 4.5041703 , 4.43201937, 4.00609765, 4.04013378,
       4.49874206, 4.16022193, 4.07538319, 4.02641083, 3.97555887,
       3.90438834, 4.00586337, 3.89737231, 3.93972652, 4.096009  ,
       4.09028617, 4.51785755, 3.67700693, 4.05481277, 4.06994045,
       3.99439043, 4.08724956, 4.63632845, 4.01926945, 4.13500122,
       4.06356966, 4.44654143, 4.00570621, 4.0159077 , 4.05388859,
       3.70651744, 4.50943257, 4.25191759, 4.00880145, 4.01996633,
       4.24664449, 4.08705549, 3.98480555, 4.00071007, 3.99537194,
       4.04357848, 4.02590179, 3.97953521, 3.98639616, 4.42790224,
       4.04452364, 3.98009663, 4.38545447, 3.91634764, 4.08013732,
       4.45993392, 4.44058136, 4.40211659, 3.91893161, 4.39583368,
       3.99642377, 4.00931412, 3.97156275, 4.25498372, 4.0272176 ,
       4.06610012, 4.41245722, 4.07035802, 4.13630315, 3.8628597 ,
       4.06890291, 4.39298691, 3.91540108, 3.73950948, 3.96852444,
       4.06001577, 4.37715993, 4.00399036, 4.05329434, 4.03393353,
       4.01410461, 4.02523628, 4.04176405, 3.96081759, 3.84169582,
       4.11793804, 3.96820558, 4.47754571, 3.99388648, 4.04703794,
       4.5333408 , 3.90782873, 4.39997596, 3.86244538, 4.12471378,
       4.48825443, 3.89851031, 3.98154919, 4.41644176, 3.9594109 ,
       4.10520727, 4.03618748, 4.46989798, 4.36060576, 3.97849779,
       4.19826769, 3.941537  , 4.00683942, 4.00410507, 3.96725334,
```

```
3.91579353, 4.09855584, 4.00620569, 4.47242469, 3.99893409,
4.56279182, 4.44919711, 3.98923683, 4.02805387, 3.97764496,
3.9967591 , 4.01441589, 4.03398771, 4.4864357 , 4.01212279,
4.4993724 , 3.97896445, 3.94980895, 3.98495442, 4.07552556,
4.03987188, 3.96923218, 4.01764182, 3.92740445, 3.97555071,
4.40423422, 3.9978407 , 3.97106581, 4.28809495, 3.82395571,
4.0649278 , 3.69001163, 4.05554386, 4.0173955 , 4.02946479,
4.53207479, 4.28083531, 4.21028699, 4.5261392 , 3.99199875,
4.09443681, 3.92393335, 4.06919553, 4.08649091, 3.98868959,
4.08187123, 4.01124706, 4.09568222, 4.25774164, 3.42622967,
4.52775683, 4.04024803, 3.87612691, 4.34232149, 3.99097222,
4.02123877, 4.01369015, 3.9704928 , 3.95093795, 3.80218053,
3.85398857, 4.00723059, 4.08708605, 4.01692998, 3.96003334,
4.10729484, 4.52582593, 4.01760358, 4.08903959, 4.03813787,
3.93274251, 4.06582575, 4.45207905, 4.15562665, 4.02290608,
4.03289626, 3.99294156, 3.82937424, 4.11327706, 3.9489988 ,
4.02666711, 4.04130325, 4.02464279, 4.33245752, 4.05657421,
3.92319768, 3.86367751, 4.04649515, 4.37215197, 3.95952912,
4.47728029, 3.9684319 , 3.94809934, 4.02864346, 3.92539118,
3.96832973, 4.48066273, 4.38873095, 4.10068954, 3.89635841,
4.04516884, 4.02137896, 4.14279337, 4.40834697, 4.35469474,
4.02814317, 4.38607848, 4.09436556, 4.0414359 , 3.95824453,
4.01635392, 4.04370408, 4.04364834, 4.31357307, 4.47048368,
3.92034595, 4.03909747, 4.01061162, 4.11407869, 4.4566573 ,
4.42971321, 4.06877832, 4.12813992, 4.03516857, 4.02889996,
3.89348729, 4.07515117, 4.55117063, 4.06413046, 4.02473283,
4.00400322, 3.99678267, 4.0551604 , 4.14094163, 3.97008551,
3.97472618, 4.03304418, 4.48611071, 4.00975633, 4.17138262,
4.15207909, 4.04068281, 3.98589565, 3.95967698, 4.01292363,
4.16618652, 3.91639453, 3.99709082, 4.09813384, 3.99334074,
4.03736599, 4.06970791, 4.02492179, 4.05882026, 4.53549776,
4.07116217, 3.90167756, 4.04049102, 4.41934855, 4.07743113,
4.08741131, 4.00435839, 4.14152952, 4.10325158, 3.79191737,
4.23225927, 4.0621834 , 4.06528618, 4.48040665, 3.93087121,
3.91724171, 4.5297662 , 3.93597711, 4.1055277 , 3.92910135,
3.98130275, 4.52880938, 4.10528329, 4.04296953, 3.82721642,
4.00857966, 4.14015733, 3.99643929, 4.02244443, 3.99659849,
4.0359566 , 4.11684757, 4.05369131, 4.38866976, 4.39336566,
4.14751446, 4.09632307, 4.06376925, 3.97166643, 4.05499788,
4.42862939, 4.66329505, 3.94772311, 3.94985575, 4.10876808,
4.0871472 , 3.83470153, 3.75999088, 4.47873796, 4.46481541,
3.94461639, 4.32063512, 4.33901344, 3.98612182, 4.00840563,
4.20318848, 4.14464042, 3.95826323, 4.07306878, 4.06489096,
4.12332293, 4.04884244, 4.03612617, 3.87495778, 4.04931923,
3.92379915, 4.04307373, 4.01105964, 4.29774214, 4.14562992,
4.0759891 , 3.95188774, 4.03904914, 4.09209729, 3.93393466,
3.95972113, 4.03645901, 3.97469188, 3.98501718, 4.46385353,
3.91929773, 4.38542615, 4.00401814, 4.11468211, 4.09949453,
3.96063233, 3.9620318 , 3.9505587 , 3.98940654, 4.0181591 ,
4.06103348, 4.30388022, 4.07354229, 4.43400646, 4.48611865,
4.08771631, 4.16790128, 3.8983826 , 4.01676675, 4.00231369,
3.99653086, 4.09380774, 4.07087576, 4.0031202 , 3.98256142,
4.03135702, 4.0780013 , 4.38667681, 4.04721128, 3.89807646,
3.93795845, 4.3714755 , 3.97089718, 4.00513598, 4.07035802,
4.41416147, 4.04466281, 3.91667733, 4.29274585, 3.99519455,
4.45551982, 4.46113343, 3.31093757, 4.37895344, 4.44507997,
3.99818543, 4.08030519, 4.54237655, 4.42008718, 4.43232491,
3.99441045, 4.45351657, 4.05766576, 4.41240138, 4.11982741,
3.99181358, 3.98492354, 3.99904789, 4.03449169, 4.06354968,
4.08320502, 3.90066159, 4.03534805, 4.00216126, 3.98779634,
4.43373313, 4.35319615, 4.04485636, 4.05357289, 4.01317549,
4.00935902, 4.08757692, 4.50936747, 4.07964198, 3.95788906,
3.9049839 , 3.93397563, 4.04227348, 4.09484796, 4.08135952,
3.91497869, 3.87181538, 4.00216126, 4.05704908, 4.2535617 ,
3.98446831, 4.40903309, 4.04306303, 4.0641413 , 4.32729768,
4.12562427, 3.99450774, 4.02795766, 4.07053629, 4.07538319,
3.99766945, 3.94757354, 3.87234149, 3.97939116, 3.99953571,
4.03886845, 4.09383438, 4.05574674, 3.99584284, 4.33063517,
4.03585681, 4.08116267, 4.44707892, 4.09108192, 3.91161175,
3.95138253, 4.17170212, 4.07501711, 4.35074393, 4.60324314,
3.98587323, 4.02334256, 3.99844412, 4.03830494, 3.85902579,
4.04934548, 4.03094111, 4.0335883 , 4.43431415, 4.07922391,
4.02885222, 4.09763327, 3.7148296 , 4.0369128 , 4.07437924,
3.97051215, 4.03761615, 4.00102396, 4.44564327, 4.03034227,
4.05800733, 3.99739477, 3.9844259 , 4.07540183, 3.98157188,
3.95897823, 4.45663582, 4.45162189, 4.4778699 , 3.9322992 ,
4.52349963, 4.10241466, 4.00542438, 3.92263922, 4.00465451,
3.96374899, 3.97432415, 3.95037708, 3.99380497, 4.0646782 ,
4.05577773, 4.03560573, 3.88880056, 4.53363784, 3.93490273,
3.85014007, 3.97166643, 4.00995965, 3.99960252, 3.9619269 ,
4.30088213, 3.97066885, 4.06957083, 4.49410562, 4.04369476,
4.48743707, 4.15880203, 4.06351174, 4.04773762, 4.4307143 ,
3.95189686, 4.20314822, 3.96144002, 4.10583553, 3.96288955,
4.06311748, 4.07178036, 4.04421644, 4.07828321, 3.95440341,
4.06998044, 4.44103992, 3.86889946, 4.46072083, 4.01459343,
```

```
       4.43621676, 4.00187518, 4.00776802, 4.07665666, 3.93865969,
       4.51613468, 4.13199972, 3.9763936 , 4.43051425, 4.0458254 ,
       3.99633813, 4.03317097, 4.49743247, 4.16124619, 3.84446226,
       4.09063206, 4.03468055, 4.00575803, 4.03062805, 3.98847701,
       4.01094216, 4.05603501, 4.05860399, 4.20890374, 3.84508207,
       4.11704277, 3.97466419, 4.04571296, 4.13917198, 3.62593326,
       4.30058031, 4.0879115 , 4.14940239, 4.08739046, 3.88753479,
       3.94253665, 4.00729873, 4.42681627, 4.02875479, 4.37447076,
       4.03245908, 4.07412894, 4.43480658, 3.95328256, 3.97732858,
       4.01367859, 3.95011446, 3.86584903, 4.04473521, 4.1093259 ,
       4.01800304, 4.00419161, 4.03582084, 4.55250467, 4.47256597,
       4.05982273, 4.06122233, 4.00273271, 4.02608404, 4.08994534,
       3.92393197, 4.10137919, 4.375962  , 3.07902993, 4.07286475,
       4.02113331, 4.14476545, 4.06334561, 4.06493435, 3.96794786,
       4.02732671, 3.99911392, 4.07067665, 4.00522023, 4.06365864,
       4.51942258, 4.19681076, 3.98324099, 4.01376653, 4.04913514,
       4.45510541, 4.48550401, 4.01117965, 3.8204096 , 4.40217663,
       4.40819638, 4.03808145, 4.42396266, 4.05990625, 4.03583472,
       4.13624007, 4.00441948, 3.98505611, 3.88740776, 4.28903442,
       4.01890428, 3.92836291, 4.02049767, 4.06941682, 4.52666593,
       4.04304166, 4.19901702, 4.10173969, 4.16345972, 4.53831299,
       4.05050658, 4.1258934 , 4.13136118, 4.05348136, 4.01134811,
       4.07939383, 3.98469541, 4.00524383, 4.08536401, 4.07822253,
       4.06625452, 3.93907473, 4.1055277 , 4.04751598, 3.90166666,
       3.97972849, 4.07787022, 3.90997879, 4.08834587, 4.07055137,
       4.48212936, 4.00399784, 4.52173092, 3.92818856, 4.06417553,
       4.01900402, 4.46578855, 4.5201489 , 4.46328116, 3.8755282 ,
       3.82398148, 4.05236629, 4.24487571, 4.0054062 , 3.99939156,
       4.44241041, 3.94183564, 3.92802187, 4.05512846, 4.16434048,
       4.38431657, 3.99433266, 3.94876309, 3.55097018, 3.8922259 ])
```

These are the reviews prediction for Test.csv

In [78]:
```
# insert them into reviewTest.csv
review_test_pre = pd.read_csv('reviewTest.csv')
review_test_pre['review'] = test_pred
review_test_pre
```

Out[78]:

	id	title	group	review
0	90	The Eagle Has Landed	Book	4.095399
1	1372	Che in Africa: Che Guevara's Congo Diary	Book	4.040049
2	1382	The Darwin Awards II : Unnatural Selection	Book	4.010982
3	253	Celtic Glory	Music	4.489850
4	671	Sublte Aromatherapy	Book	3.969100
...	...	...	...	...
995	1097	Ahma	Music	4.384317
996	1393	Loney Planet Chicago City Map (City Maps Series)	Book	3.994333
997	643	Swell Style : A Girl's Guide to Turning Heads...	Book	3.948763
998	976	Dark Continent : Europe's Twentieth Century	Book	3.550970
999	961	Choice and Consequence	Book	3.892226

1000 rows × 4 columns

as prof said in slack, we can fill in the "review" column with continious value, so we will output this df to csv. to submit

In [80]:
```
review_test_pre = review_test_pre.set_index('id')
review_test_pre
```

Out[80]:

id	title	group	review
90	The Eagle Has Landed	Book	4.095399
1372	Che in Africa: Che Guevara's Congo Diary	Book	4.040049
1382	The Darwin Awards II : Unnatural Selection	Book	4.010982
253	Celtic Glory	Music	4.489850
671	Sublte Aromatherapy	Book	3.969100
...	...	...	...
1097	Ahma	Music	4.384317
1393	Loney Planet Chicago City Map (City Maps Series)	Book	3.994333
643	Swell Style : A Girl's Guide to Turning Heads...	Book	3.948763
976	Dark Continent : Europe's Twentieth Century	Book	3.550970
961	Choice and Consequence	Book	3.892226

1000 rows × 3 columns

In [81]:
```
review_test_pre.to_csv('671_hw3_q2_ella_submission_2.csv')
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: