Info

Name: Jacqui Unciano **Date:** Feb 22, 2024 **Assignment:** HW6

Directions

In this week's code exercise, you will compute and explore vector space distances between documents for a corpus of Jane Austen's novels.

Use the notebook from class as your guide, as well as any relevant previous notebook . For source data, use the LIB and CORPUS tables you used last week for the Austen and Melville set. These are in the /data/output directory of the course repo.

In [1]:
```
import pandas as pd
import numpy as np
from numpy.linalg import norm
from scipy.spatial.distance import pdist
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
import plotly_express as px
import seaborn as sns; sns.set()
```

In [2]:
```
import configparser
config = configparser.ConfigParser()
config.read("../../../env.ini")
data_home = config['DEFAULT']['data_home']
output_dir = config['DEFAULT']['output_dir']
data_prefix = 'austen-melville'

OHCO = ['book_id', 'chap_id', 'para_num', 'sent_num', 'token_num']
bags = dict(
    SENTS = OHCO[:4],
    PARAS = OHCO[:3],
    CHAPS = OHCO[:2],
    BOOKS = OHCO[:1]
)

LIB = pd.read_csv(f"{output_dir}/{data_prefix}-LIB.csv").set_index('book_id')
CORPUS = pd.read_csv(f'{output_dir}/{data_prefix}-CORPUS.csv').set_index(OHCO).drop
```

In [3]:
```
ab_dict = {key: group.index.tolist() for key, group in LIB.groupby('author')}
authors = []
for id in CORPUS.index.get_level_values(level=0):
    if id in ab_dict["AUSTEN, JANE"]:
        authors.append("AUSTEN, JANE")
    else:
        authors.append("MELVILLE, HERMAN")

CORPUS['author'] = authors
```

```
CORPUS = CORPUS.reset_index().set_index(['author']+OHCO)
CORPUS = CORPUS.query('author=="AUSTEN, JANE"')
```

In [4]:
```
CORPUS = CORPUS.reset_index().set_index('book_id')
CORPUS = CORPUS.drop('author', axis=1)
CORPUS = CORPUS.reset_index().set_index(OHCO)
```

In [5]:
```
CORPUS.sample(10)
```

Out[5]:

					pos_tuple	pos	token_str	te
book_id	**chap_id**	**para_num**	**sent_num**	**token_num**				
141	**14**	**9**	**0**	**10**	('was', 'VBD')	VBD	was	
1342	**26**	**26**	**2**	**31**	('again.', 'JJ')	JJ	again.	
161	**50**	**7**	**0**	**39**	('shrubberies,', 'NN')	NN	shrubberies,	shrul
105	**1**	**19**	**0**	**36**	('whose', 'WP$')	WP$	whose	
141	**25**	**36**	**1**	**114**	('to', 'TO')	TO	to	
161	**49**	**46**	**0**	**12**	('convenient', 'NN')	NN	convenient	conv
	22	**1**	**0**	**91**	('engaging', 'VBG')	VBG	engaging	en
158	**26**	**66**	**5**	**4**	('have', 'VB')	VB	have	
121	**19**	**16**	**2**	**0**	('A', 'DT')	DT	A	
158	**26**	**13**	**3**	**3**	('to', 'TO')	TO	to	

Also, you will need to generate the VOCAB table from the Austen corpus; you can import your work from your last homework if you'd like.

In [6]:
```
VOCAB = CORPUS.term_str.value_counts().to_frame('n').reset_index().set_index('term_
VOCAB['max_pos'] = CORPUS[['term_str','pos']].value_counts().unstack(fill_value=0).
VOCAB.sample(10)
```

Out[6]:

term_str	n	max_pos
accusing	1	VBG
swisserland	4	NNP
repent	15	VB
epsom	3	NNP
undutiful	1	JJ
execrable	1	JJ
pattens	1	NN
times	132	NNS
catching	27	VBG
centered	3	VBN

Add a feature to the LIB table for the publication year of the book, using the data provided below.

1. Create a label for each book using a combination of the year and the book title.
2. Scholarly side note: This is the publication year in most cases. For works published posthumously, the year refers to when scholars think the work was actually completed. Note also, there is often a lag between date of completion and data of publication. We will not concern ourselves with these nuances here, but it is always helpful to understand how your data are actually produced.

In [7]:

```
labels = {158: "Emma, 1815",
          946: "Lady Susan, 1794",
          1212: "Love and Friendship And Other Early Works, 1790",
          141: "Mansfield Park, 1814",
          121: "Northanger Abbey, 1803",
          105: "Persuasion, 1818",
          1342: "Pride and Prejudice, 1813",
          161: "Sense and Sensibility, 1811"}
label = []
for id in CORPUS.index.get_level_values(level=0):
    if id in labels.keys():
        label.append(labels[id])

CORPUS['label'] = label
```

In [8]:

```
CORPUS.head()
```

Out[8]:

book_id	chap_id	para_num	sent_num	token_num	pos_tuple	pos	token_str	term_str
105	1	1	0	0	('Sir', 'NNP')	NNP	Sir	sir
				1	('Walter', 'NNP')	NNP	Walter	walter
				2	('Elliot,', 'NNP')	NNP	Elliot,	elliot
				3	('of', 'IN')	IN	of	of
				4	('Kellynch', 'NNP')	NNP	Kellynch	kellynch

Bring into your notebook the functions you created previously to generate a BOW table and compute TFIDF values. Extend the TFIDF function so that it also returns the DFIDF value for each term in the VOCAB. Note that you can use the functions you created last week to compute TFIDF; if you had problems with these, you may use functions in the homework key.

In [9]:

```
def bow(DF, bag):
    BOW = DF.groupby(bags[bag]+['term_str']).term_str.count().to_frame('n')
    return BOW

def tfidf(BOW, tf_method):
    # global VOCAB
    DTCM = BOW.n.unstack(fill_value=0)

    if tf_method == 'sum':
        TF = DTCM.T / DTCM.T.sum()
    elif tf_method == 'max':
        TF = DTCM.T / DTCM.T.max()
    elif tf_method == 'log':
        TF = np.log2(1 + DTCM.T)
    elif tf_method == 'raw':
        TF = DTCM.T
    elif tf_method == 'double_norm':
        TF = DTCM.T / DTCM.T.max()
    elif tf_method == 'binary':
        TF = DTCM.T.astype('bool').astype('int')
    TF = TF.T

    DF = DTCM.astype('bool').sum()

    N = DTCM.shape[0]
    IDF = np.log2(N / DF)

    TFIDF = TF * IDF
    DFIDF = DF * IDF
```

```python
    return TFIDF, DFIDF
```

Apply these functions to the corpus of Austen's works only, and using chapters as bags and max as the TF count method.

In [10]:
```python
bow_df = bow(CORPUS, 'CHAPS')
```

bow_df.shapeDTCM = bow_df.n.unstack(fill_value=0) TF = DTCM.T / DTCM.T.max() TF = TF.T TF.shapeTF.stack().head()bow_df.head()

In [11]:
```python
tfidf, dfidf = tfidf(bow_df, 'max')
```

In [12]:
```python
display(tfidf, dfidf)
```

	term_str	0	1	10	10000	10th	11th	12	12th	1399	13th	...	youthfu
book_id	**chap_id**												
105	**1**	0.0	0.119092	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.04341
	2	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000
	3	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000
	4	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000
	5	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000
...	**...**	...	...	...	...	...	...	...	...	...	...	...	
1342	**57**	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000
	58	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000
	59	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000
	60	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000
	61	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.00000

334 rows × 14745 columns

```
term_str
0             8.383704
1            14.767409
10           20.396225
10000         8.383704
10th         14.767409
               ...
zealous      34.792451
zealously    14.767409
zephyr        8.383704
zigzags       8.383704
ł20000        8.383704
Length: 14745, dtype: float64
```

Reduce the number of features in the returned TFIDF matrix to the 1000 most significant terms, using DFIDF as your significance measure and only using terms whose maximum part-of-speech belongs to this set: NN NNS VB VBD VBG VBN VBP VBZ JJ JJR JJS RB RBR RBS. Note, these are all open categories, excluding proper nounns.

In [13]:
```
pos_tags = ["NN", "NNS", "VB", "VBD", "VBG", "VBN", "VBP", "VBZ", "JJ", "JJR", "JJS
```

In [14]:
```
df = pd.concat([VOCAB, dfidf.to_frame('dfidf')], axis=1)
df.head()
```

Out[14]:

	n	max_pos	dfidf
term_str			
the	28274	DT	0.000000
to	26029	TO	0.000000
and	24060	CC	1.440533
of	22927	IN	0.000000
a	14301	DT	2.876734

In [15]:
```
dfidf_1000 = df[df.max_pos.isin(pos_tags)].nlargest(1000, 'dfidf')
dfidf_1000.head()
```

Out[15]:

	n	max_pos	dfidf
term_str			
stay	201	VB	177.266344
thinking	200	VBG	177.266344
forward	182	RB	177.266344
respect	174	NN	177.266344
greatest	161	JJS	177.266344

In [16]:
```
top_1000 = dfidf.index.get_level_values(0).to_list()
not_1000 = [col for col in tfidf.columns if col not in top_1000]
```

In [17]:
```
red_tfidf = tfidf.drop(columns = not_1000)
red_tfidf.head()
```

Out[17]:

	term_str	0	1	10	10000	10th	11th	12	12th	1399	13th	...	youtl
book_id	chap_id												
105	1	0.0	0.119092	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.043
	2	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000
	3	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000
	4	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000
	5	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000

5 rows × 14745 columns

"Collapse" TFIDF matrix so that it contains mean TFIDF of each term by book. This will result in a matrix with book IDs as rows, and significant terms as columns.

In [18]:

```
tfidf_mean = red_tfidf.groupby(level=0).mean()
tfidf_mean
```

Out[18]:

term_str	0	1	10	10000	10th	11th	12	12th	
book_id									
105	0.000000	0.004962	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
121	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
141	0.000000	0.000000	0.003480	0.000000	0.000000	0.000000	0.000000	0.000000	0.
158	0.000000	0.000000	0.000000	0.004234	0.000000	0.000000	0.000000	0.000000	0.
161	0.000000	0.001522	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
946	0.000000	0.000000	0.004482	0.000000	0.000000	0.000000	0.000000	0.000000	0.
1212	0.001968	0.000000	0.000000	0.000000	0.009915	0.000984	0.006986	0.000984	0.
1342	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.

8 rows × 14745 columns

Use the reduced and collapsed TFIDF matrix to compute distance missures between all pairs of books, as we computed in Lab (using pdist()). See the table below for the measures to take.

1. As in the notebook from class, use the appropriate normed vector space for each metric.
2. You will need to create a table of book pairs (e.g. PAIRS).
3. You do not need to compute k-means clusters.

In [19]:
```
L0 = tfidf_mean.astype('bool').astype('int') # Binary (Pseudo L)
L1 = tfidf_mean.apply(lambda x: x / x.sum(), 1) # Probabilistic
L2 = tfidf_mean.apply(lambda x: x / norm(x), 1) # Pythagorean, AKA Euclidean
```

In [20]:
```
CORPUS.index.get_level_values(0).unique().tolist()
```

Out[20]:
```
[105, 121, 141, 158, 161, 946, 1212, 1342]
```

In [21]:
```
PAIRS = pd.DataFrame(index=pd.MultiIndex.from_product([CORPUS.index.get_level_value
                                                       CORPUS.index.get_level_values(0).uni
# Keep only unique pairs of different books
PAIRS = PAIRS[PAIRS.level_0 < PAIRS.level_1].set_index(['level_0','level_1'])
# Name index cols
PAIRS.index.names = ['book_a', 'book_b']
PAIRS.head()
```

Out[21]:

book_a	book_b
105	121
	141
	158
	161
	946

In [22]:
```
PAIRS['cityblock'] = pdist(tfidf_mean, 'cityblock')
PAIRS['euclidean'] = pdist(tfidf_mean, 'euclidean')
PAIRS['cosine'] = pdist(tfidf_mean, 'cosine')
PAIRS['jaccard'] = pdist(L0, 'jaccard')
PAIRS['dice'] = pdist(L0, 'dice')
PAIRS['js'] = pdist(L1, 'jensenshannon')
```

In [23]:
```
PAIRS.loc[105]
```

Out[23]:

book_b	cityblock	euclidean	cosine	jaccard	dice	js
121	29.327485	0.997043	0.791620	0.569377	0.397993	0.539728
141	25.542050	0.965815	0.790309	0.546583	0.376067	0.503049
158	26.615229	1.030791	0.813057	0.550828	0.380099	0.510206
161	27.405552	1.056178	0.829321	0.542638	0.372342	0.520650
946	32.940496	1.169664	0.881663	0.669782	0.503513	0.617020
1212	37.028466	1.057379	0.839983	0.662244	0.495041	0.627852
1342	27.138958	0.978181	0.778770	0.544395	0.373999	0.517983

Create hierarchical agglomerative cluster diagrams for the distance measures, using the appropriate linkage type for each distance measure. Again, see the table below for the appropriate linkage type. 1. Use the labels you created in the LIB in your dendograms to help interpret your results.

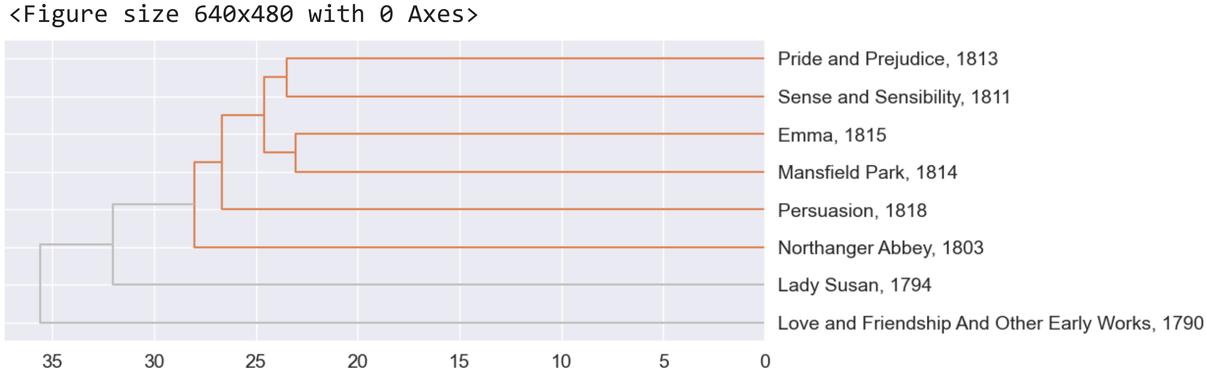
In [24]:
```
CORPUS.label.unique()
```

Out[24]:
```
array(['Persuasion, 1818', 'Northanger Abbey, 1803',
       'Mansfield Park, 1814', 'Emma, 1815',
       'Sense and Sensibility, 1811', 'Lady Susan, 1794',
       'Love and Friendship And Other Early Works, 1790',
       'Pride and Prejudice, 1813'], dtype=object)
```

In [25]:
```
def hac(sims, linkage_method='complete', color_thresh=.3, figsize=(10, 4)):

    # Generate the clustering
    tree = sch.linkage(sims, method=linkage_method)

    # Get labels for the leaves
    labels = CORPUS.label.unique()

    # Create a figure
    plt.figure()
    fig, axes = plt.subplots(figsize=figsize)

    # Create a dendrogram with the tree
    dendrogram = sch.dendrogram(tree,
                                labels=labels,
                                orientation="left",
                                count_sort=True,
                                distance_sort=True,
                                above_threshold_color='.75',
                                color_threshold=color_thresh
                               )

    # Change the appearance of ticks, tick labels, and gridlines
    plt.tick_params(axis='both', which='major', labelsize=14)
```

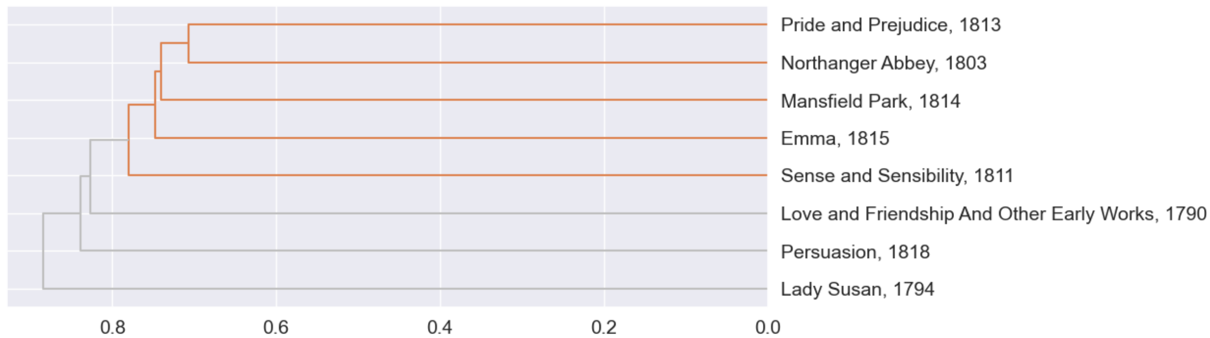
In [26]:
```
hac(PAIRS.cityblock, linkage_method='weighted', color_thresh=30)
```

```
<Figure size 640x480 with 0 Axes>
```



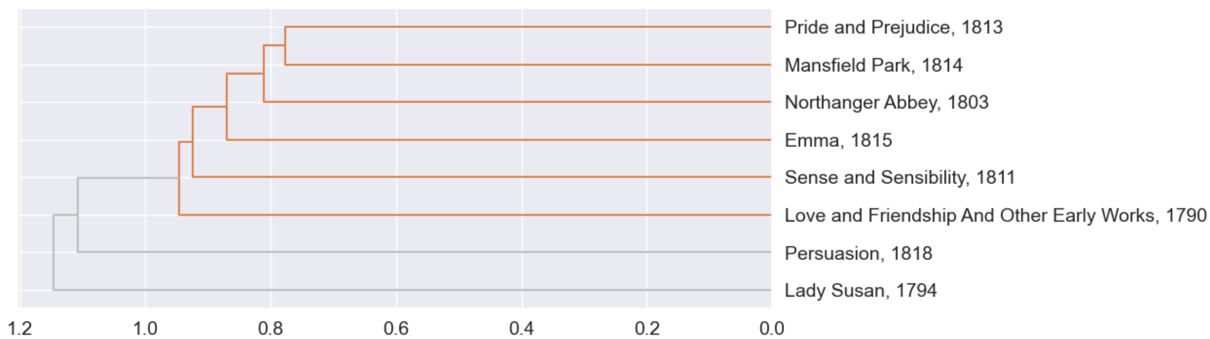
In [27]:
```
hac(PAIRS.cosine, linkage_method='ward', color_thresh=0.8)
```

<Figure size 640x480 with 0 Axes>



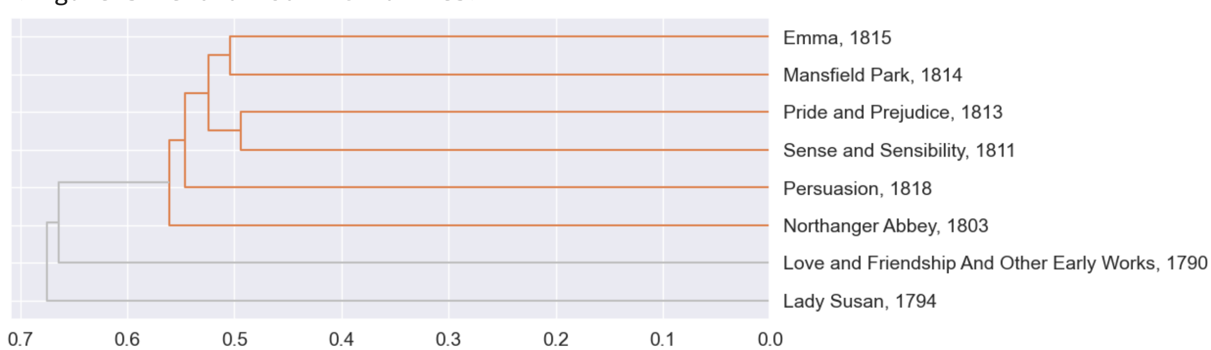
In [28]:
```
hac(PAIRS.euclidean, linkage_method='ward', color_thresh=1.0)
```

<Figure size 640x480 with 0 Axes>



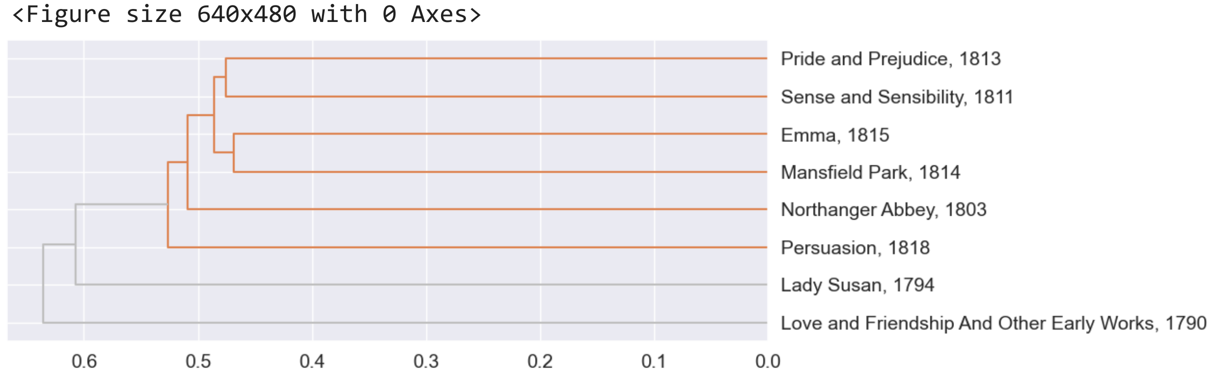
In [29]:
```
hac(PAIRS.jaccard, linkage_method='weighted', color_thresh=0.6)
```

<Figure size 640x480 with 0 Axes>



In [30]:
```
hac(PAIRS.js, linkage_method='weighted', color_thresh=0.6)
```

<Figure size 640x480 with 0 Axes>



Question 1

What are the top 10 nouns by DFIDF, sorted in descending order? Include plural nouns, but don't include proper nouns.

Answer 1

In [31]:
```
dfidf_1000.query('max_pos == "NN" or max_pos == "NNS"').head(10)
```

Out[31]:

term_str	n	max_pos	dfidf
respect	174	NN	177.266344
marriage	246	NN	177.261968
fortune	222	NN	177.261968
ladies	240	NNS	177.258990
question	171	NN	177.258990
behaviour	200	NN	177.240001
farther	181	NN	177.240001
advantage	166	NN	177.217644
girl	254	NN	177.209470
voice	228	NN	177.209470

Question 2

Grouping your TFIDF results by book, and taking the mean TFIDF of all terms per book, what is Austen's most "significant" book? This value is computed from the TFIDF matrix your function returned.

In [32]:
```
tfidf_mean.mean(axis=1).to_frame('mean').nlargest(1,"mean")
```

Out[32]:

book_id	mean
121	0.001851

Answer 2

Northanger Abbey

Question 3

Using the dendograms you generated, which distance measure most clearly distinguishes Austen's two youthful works from her later works? That is, which measure show the greatest separation between the first two work and the rest? Note that the two youthful works were published before 1800.

Answer 3

I believe it is jaccard measure that most clearly distinguishes Austen's youthful works from her other works. Jensen-Shannon and Cityblock also do it clearly though.

Question 4

Do any of the distance measures produce dendrograms with works sorted in the exact order of their publication years?

Answer 4

No.

Question 5

Some literary critics believe that Northanger Abbey is, among Austen's mature works, the one that most resembles her juvenalia, i.e. her two works written as a young adult. Which distance measure dendrograms appear to corroborate this thesis? In other words, do any of them show that Northanger Abbey is closer to her juvenalia than the her other adult works?

Answer 5

Cityblock and Jaccard both have Northanger Abbey closer to her two younger works. They still have Northanger closer in distance to her later works compared to her younger works though.