VECTORIZED REPRESENTATIONS OF MOVIE REVIEWS

Steve Desilets

MSDS 453: Natural Language Processing

Section 56 – Summer 2023

July 9, 2023

1. Introduction and Problem Statement

Movie lovers often rely on the opinions of trusted movie critics to provide candid descriptions of newly released movies, so that they can determine whether to go see a movie. Given the important role that movie reviews can play in influencing the decisions of cinematic consumers, we are interested in exploring the effectiveness of natural language processing (NLP) pipelines in better supporting consumers by providing information about movies. The underlying data leveraged in our experiment consists of 190 rows and 9 columns, as described in Table 1 below.

Table 1: Description of the Structure of the Movie Review Dataset

Variable Name	Python Data Type	Description
DSI_Title	Object	Unique identifier for the movie review (including student initials, Doc ID, and movie title)
Submission File Name	Object	Unique identifier for the movie review (including student initials, Doc ID, and movie title)
Student Name	Object	Initials of student who added review to the corpus
Genre of Movie	Object	Movie genre (action, comedy, horror, or sci-fi)
Review Type (pos or neg)	Object	Type of review (positive or negative)
Movie Title	Object	Movie title
Text	Object	First 500 words of the original movie review's text
Descriptor	Object	One-word summary of movie review (including genre, movie name, review type, and Doc ID)
Doc_ID	Object	Unique ID for movie review document

In this paper, we explore some of the primary components of NLP pipelines, including normalization and vectorization techniques. Exploration of these techniques will empower us to identify important and prevalent terms for inclusion in our movie review vocabulary and to assess the effectiveness of various data wrangling techniques via cosine similarity heatmaps and T-SNE plots. The critical analyses in this paper will allow us to lay a fantastic foundation for subsequent additions to our NLP pipeline (in future assignments) that will include creating clustering and classification tools, knowledge graphs, and chatbots that can better empower movie lovers.

2. Research Design and Modeling Methods

This research study was conducted in two phases that consisted of qualitative and quantitative approaches to vocabulary selection and NLP data wrangling. In the first (qualitative) phase, we imported the movie review data into a Jupyter Notebook via Python, conducted initial exploratory data analysis, normalization, and tokenization, and leveraged key performance indicators to assess the utility of various terms for clustering movie reviews in our corpus. In the second (quantitative) phase, we leveraged 3 data wrangling techniques and 7 word / document vectorization techniques to conduct 21 experiments designed to determine which data wrangling techniques would be most appropriate for our NLP pipeline.

Here we can provide a more detailed description of the first qualitative phase of our analysis. After importing the corpus data and conducting exploratory data analysis, we first normalized the corpus by removing punctuation, transforming letters to lowercase, removing HTML tags, and then we tokenized the document via the NLTK Word_Tokenize method. We then identified 13 terms that we thought would be helpful for clustering reviews of our primary movie of interest, Sisters, with other comedy movies (Moore 2015). We calculated the mean number of times each of these terms appeared in the Sisters movie reviews, in the comedy movie reviews, and in the non-comedy movie reviews to gauge their comedy review clustering potential. Using the resulting table (pictured in Appendix A), we identified four terms that seemed to be important to comedy movie reviews (“party”, “comedy”, “funny”, and “joke”) and calculated the number of times that these terms appeared in each of the “Sisters” reviews (as displayed in Appendix C).

Having completed the qualitative analysis, we began the quantitative analysis by first applying three data wrangling methods to our originally imported movie review corpus. These data wrangling techniques are summarized in Table 2 below.

Table 2: Descriptions of the three Data Wrangling Methodologies

Data Wrangling Method 1	Data Wrangling Method 2	Data Wrangling Method 3
• Punctuation Removal • Transformation to Lowercase • HTML Tag Removal • Tokenization (via NLTK Word_Tokenize method) • Stemming	• Punctuation Removal • Transformation to Lowercase • HTML Tag Removal • Removal of Special Characters and Digits • Tokenization (via NLTK Word_Tokenize method) • Lemmatization • Removal of NLTK's English Stop Words	• Punctuation Removal • Transformation to Lowercase • HTML Tag Removal • Removal of Special Characters and Digits • Tokenization (via NLTK Word_Tokenize method) • Lemmatization • Removal of NLTK's English Stop Words • Removal of 21 of the top 100 terms (based on TF-IDF score) that we believed might add little value to a NLP pipeline focused on movie reviews (i.e. "film", "movie", etc.)

After implementing these three data wrangling methodologies, we conducted several experiments via TF-IDF vectorization. First we calculated the top 10 terms based on TF-IDF scores for each of our pre-processed corpora, as well as the mean TF-IDF scores for each of our previously identified top four vocabulary terms of interest ("party", "comedy", "funny", and "joke") and displayed those results in Appendices B and D. These tables enabled us to compare the TF-IDF scores for our terms of interest to highest TF-IDF scores in each pre-processed corpus, so we could assess their relative importances. We also created heatmaps to visualize the cosine similarities between each of the 190 documents after TF-IDF vectorization of the documents. We displayed these heatmaps in Appendix E so that we could understand how well each data wrangling method performs at producing similar TF-IDF vectors for reviews of each of the 19 movies (which would be helpful for future clustering exercises).

Having finished our TF-IDF vectorization experiments, we next turned our attention to Doc2Vec vectorization experiments using each of our three pre-processes corpora. We conducted nine Doc2Vec experiments by applying Doc2Vec vectorization using 100, 200, and 300 embedding dimensions to each document in our three pre-processed corpora (that had leveraged data wrangling methods 1, 2, and 3).

After obtaining the nine resulting dataframes, we created T-SNE plots of the resulting document vectors, which are displayed in Appendix I, to identify whether certain data wrangling and embedding methods produce the most accurate clusters for our 19 movies. Similarly, we created heatmaps, displayed in Appendix H, to visualize whether certain data wrangling and embedding methods were better at producing cosine similarity measures that were highly capable of distinguishing our 19 movies.

Our final batch of nine experiments examined the utility of Word2Vec vectorization of the terms in our corpus. For these experiments, we first identified the top 96 terms (as determined via TF-IDF scores) in each of our pre-processed corpora that had leveraged data wrangling methods 1, 2, and 3. We then added the stemmed / lemmatized versions of our top four terms of interest for the vocabulary ("party", "comedy", "funny", and "joke") to create a list of 100 terms for each corpora. We then vectorized each of these three lists of 100 terms via Word2Vec vectorization that leveraged 100, 200, and 300 embedding dimensions. We then created T-SNE plots to visualize whether each of our nine data wrangling and Word2Vec vectorization techniques would produce more useful clusters of similar words. Similarly, we created nine cosine similarity heatmaps to visualize whether certain data wrangling and vectorization techniques would result in term-level vectors capable of underscoring the semantic similarity of more similar terms. These Word2Vec T-SNE plots and cosine similarity heatmaps are displayed in Appendices F and G.

3. Results

The results from the qualitative and quantitative analyses for this study are displayed in Appendices A - I. Appendices A and C help us assess the frequency of terms of interest by displaying the mean frequency of our top 13 terms of interest in the "Sisters", comedy, and non-comedy movies, and the number of times that each of our resulting top four terms of interest appears in each of the "Sisters" reviews. Appendices B and D display some of the results of our TF-IDF vectorization experiments, including the TF-IDF scores for our top 4 terms and for the top 10 overall terms in each of our 3 corpora, so we can directly compare the importance of the most important terms to the importance of our terms of interest. Appendix E shows the last result of the TF-IDF vectorization experiments, which are the cosine

similarity heatmaps that can display whether any of our data wrangling techniques, combined with TF-IDF vectorization, performs well at highlighting relationships between reviews of similar movies. Appendices F and G display the nine Word2Vec-based T-SNE plots and cosine similarity heatmaps we can leverage to interpret which of our data wrangling and vectorization techniques produce the most meaningful clusters / depictions of semantically similar terms. Appendices H and I display the nine Doc2Vec-based T-SNE plots and cosine similarity heatmaps that will illuminate whether any of our data wrangling and Doc2Vec vectorization techniques produce visualizations that clearly identify which reviews correspond to similar movies. To promote transparency and reproducibility of our results, Appendix J contains the Python code leveraged to conduct our full analysis as well.

4. Analysis and Interpretation

The Qualitative Analysis empowered us to identify four terms ("party", "comedy", "funny", and "joke") that have strong potential for helping NLP pipelines cluster comedy movie reviews together. As displayed in Appendix A, of our original 13 terms of interest, only some words like "comedy", "funny", "joke", "party", and "humor" had much higher mean count frequencies in comedy movie reviews (including reviews of "Sisters") than they did in reviews of non-comedy movies. On the other hand, this table suggested that other words, like "character", "script", "smile", and "love" would not be useful at helping to cluster "Sisters" with comedy movies, rather than non-comedy movies. The analysis in Appendix C reinforced this finding by displaying that our top four terms appeared in 40% - 90% of the "Sisters" reviews.

Our quantitative analysis began by confirming that our top four terms of interest for the vocabulary not only frequently appear in the comedy movie reviews, but also are important terms in the entire corpus. For example, for Data Wrangling Methods 2 and 3, the TF-IDF score for "comedy" was 1.12, while the TF-IDF score cutoffs for the top 10 terms were 1.58 and 1.36 for each method, respectively. While the gap between the "comedy" TF-IDF score and the top 10 TF-IDF scores was larger for Data Wrangling Method 1, those results were influenced by the presence of all stop words in that corpus. These results, which are displayed in Appendices B and D, further confirm that our top four terms

of interest for the vocabulary ("party", "comedy", "funny", and "joke") are important to the corpus – especially after removing stop words.

The TF-IDF vectorization experiment results in Appendix E allowed us to determine which of our data wrangling techniques, when combined with TF-IDF vectorization of documents, most strongly identified relationships between reviews of similar movies in cosine similarity heatmaps. In heatmaps 2 and 3, the clear depiction of 19 clusters along the diagonal of the image emphasize that data wrangling methods 2 and 3 outperform data wrangling method 1 at identifying relationships between TF-IDF vectorizations of similar movie reviews. Given that TF-IDF vectorization is often used in search-related NLP pipeline scenarios, we likely would not want to remove additional custom stop words that could be searched, so if we were using TF-IDF vectorization, we'd likely want to use Data Wrangling Method 2 (not 3).

The results of the Doc2Vec experiments displayed in Appendices H and I help us to determine which of our nine data wrangling and Doc2Vec vectorization techniques would be best for highlighting relationships between reviews of similar movies. For example, the Doc2Vec cosine similarity heatmaps associated with data wrangling methods 2 and 3 strongly visually outperform the heatmaps associated with method 1 in highlighting 19 boxes that correspond to our 19 movies along the diagonal. When examining the six Doc2Vec T-SNE plots, two clear winners emerge in clustering Sisters movies together. Data Wrangling Method 2 combined with 200-dimensional embedding produces a cluster of 9 "Sisters" reviews very close together in the resulting T-SNE plot, while Data Wrangling Method 3 combined with 200-dimensional embedding produces a cluster 7 "Sisters" reviews that are also closely clustered with many of the comedy movie reviews. Based on these Doc2Vec-based T-SNE plots and cosine similarity heatmaps, I would elect to use 200-dimensional embedding combined with Data Wrangling Methods 2 or 3, depending on whether I were most interested in highlighting relationships between "Sisters" or comedy movie reviews, respectively.

The results of the Word2Vec-based T-SNE plots and cosine similarity heatmaps displayed in Appendices F and G assist us in identifying which of our nine Data Wrangling and Word2Vec

vectorization techniques best help us identify relationships between semantically similar terms. Examination of the heatmaps displays that Data Wrangling Methods 2 and 3 produce more evenly dispersed ranges of colors (which could be helpful in discriminating between 100 semantically different terms) than the Data Wrangling Method 1 heatmaps do. For this reason, the heatmaps associated with Data Wrangling Methods 2 and 3 are likely more useful to data scientists interested in the varying strength of relationships between words (which makes sense given that Data Wrangling Method 1 did not remove stop words, which could be semantically similar to one another). Examination of the Word2Vec based T-SNE plots reveals that Data Wrangling Method 2, combined with 300 dimensional embedding, produces a fantastic T-SNE plot for compactly clustering comedy terms, including "blonde," "party," "despicable," "poehler," "lost," "city," "fey", "sister," "elle," "watson," and "funny" all neatly together in the top left corner of the plot. For this reason, Data Wrangling Method 2 combined with 300-dimensional Word2Vec embedding vectorization would likely be the best technique for preparing a NLP pipeline focused on identifying semantically similar words in our corpus – especially comedy-related terms.

5. Conclusions

The experiments conducted throughout this analysis have helped us lay a strong foundation for building out more advanced NLP pipelines capable of clustering, topic modeling and chatting. The qualitative analysis confirmed that our top four recommended vocabulary terms, "party", "comedy", "funny", and "joke", appear frequently in the corpus and are important terms in reviews of comedy movies. The cosine similarity heatmaps constructed via TF-IDF vectorization of our 190 documents revealed that would likely want to leverage Data Wrangling Method 2 in combination with this embedding method for search-related NLP pipelines. The nine cosine similarity heatmaps and T-SNE plots created via Doc2Vec vectorization of our 190 movie reviews highlighted that we should leverage 200 embedding dimensions combined with Data Wrangling Method 2 or 3 for NLP pipelines focused on underscoring relationships between "Sisters" reviews or comedy reviews, respectively. Last, the Word2Vec-based cosine similarity heatmaps and T-SNE plots emphasized that Data Wrangling Method 2 combined with 300-dimensional embedding vectorization would be the best option for preparing a NLP

pipeline focused on identifying semantically similar (comedy) terms. With the results of these analyses in hand, we are ready to expand our NLP pipelines, so that they accomplish more sophisticated applications.

References

Moore, Jason. 2015. *Sisters*. United States: Universal Pictures.

Appendix A –Mean Frequency of Terms of Interest in Subsets of Corpus Documents

The table below displays the mean number of times that our top 13 terms of interest appeared in each of the "Sisters" reviews, in each of the comedy movie reviews, and in each of the non-comedy movie reviews. The Python code leveraged to generate this table may be found in section 1.6 of Appendix J.

Mean Frequency of Terms of Interest in Subsets of Corpus Documents

Mean Term Frequencies

	Sisters	All Comedy	All Non-Comedy
party	2.4	0.54	0.057143
comedy	1.4	1.08	0.214286
funny	0.8	0.50	0.135714
character	0.6	0.56	0.485714
family	0.6	0.58	0.321429
joke	0.6	0.38	0.050000
humor	0.5	0.32	0.078571
friend	0.4	0.12	0.128571
love	0.3	0.26	0.242857
script	0.3	0.16	0.150000
school	0.3	0.68	0.021429
laugh	0.1	0.14	0.064286
smile	0.0	0.06	0.028571

Appendix B –TF-IDF Scores for the Initially Identified Potential Vocabulary Terms

For each of the three data wrangling methods, the tables below display the Term Frequency – Inverse Document Frequency (TF-IDF) scores associated with each of the initially identified potential vocabulary terms. The Python code leveraged to generate these tables may be found in section 2.2 of Appendix J.

TF-IDF Scores for the Initially Identified Potential Vocabulary Terms

Data Wrangling Method 1

	Mean TF-IDF
parti	0.73
comedi	1.12
funni	0.65
joke	0.85

Data Wrangling Method 2

	Mean TF-IDF
party	0.68
comedy	1.12
funny	0.65
joke	0.83

Data Wrangling Method 3

	Mean TF-IDF
party	0.68
comedy	1.12
funny	0.65
joke	0.83

Appendix C –Number of Times That Potential Vocabulary Terms Occur In “Sisters” Documents

The table below displays the number of times that the top four terms of interest appear in each of the “Sisters” documents.

Number of Times That Potential Vocabulary Terms Occur In “Sisters” Documents

Term	“Sisters” Document Number										Percent of “Sisters” Documents In Which Term Occurs
	1	2	3	4	5	6	7	8	9	10	
Party	1	2	1	0	3	2	8	5	2	3	90%
Comedy	4	2	0	2	2	1	1	1	1	0	80%
Funny	2	0	0	1	0	0	1	4	0	0	40%
Joke	0	0	1	3	2	1	3	0	0	0	50%

Appendix D – Terms with the Top 10 Highest TF-IDF Scores

For each of the three data wrangling methods, the tables below display the top ten terms from the movie reviews corpus when ranked by the terms' Term Frequency – Inverse Document Frequency (TF-IDF) scores. The TF-IDF scores associated with each term may be found adjacent to each of the terms. The Python code leveraged to generate these tables may be found in section 2.2 of Appendix J.

Terms with the Top 10 Highest TF-IDF Scores

Data Wrangling Method 1

	Mean TF-IDF
the	27.83
and	14.54
of	13.23
to	12.47
in	8.55
is	7.71
it	7.67
that	6.31
as	4.51
with	4.45

Data Wrangling Method 2

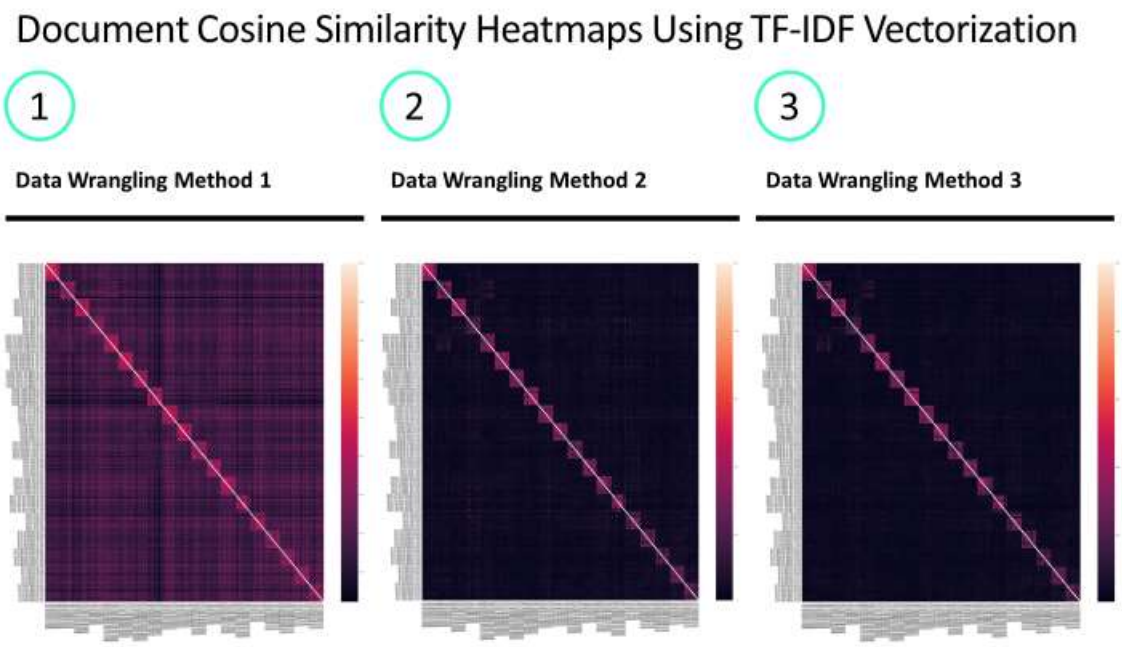
	Mean TF-IDF
film	3.56
movie	2.98
ha	2.39
wa	2.29
one	2.18
like	2.14
bond	2.01
get	1.78
time	1.68
two	1.58

Data Wrangling Method 3

	Mean TF-IDF
one	2.18
like	2.14
bond	2.01
time	1.68
two	1.58
character	1.58
dream	1.53
horror	1.44
even	1.39
first	1.36

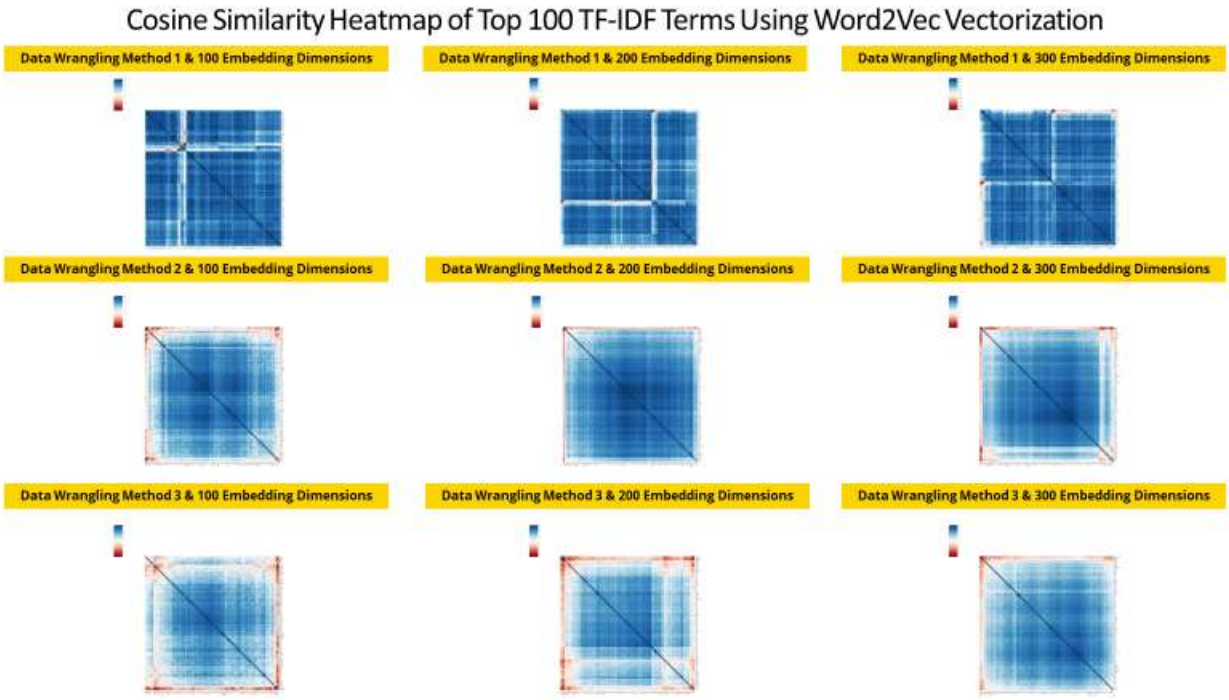
Appendix E – Document Cosine Similarity Heatmaps Using TF-IDF Vectorization

For each of the three data wrangling methods, the heatmaps below visualize the cosine similarity measures between each of the corpus documents when leveraging TF-IDF vectorization to transform each document into a vector. Larger versions of these heatmaps and the Python code leveraged to generate these heatmaps may be found in section 2.2 of Appendix J.



Appendix F – Cosine Similarity Heatmap of the Top 100 TF-IDF Terms Using Word2Vec Vectorization

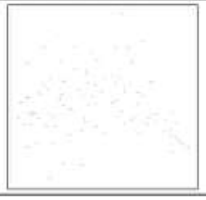
The heatmaps below display the cosine similarity measures between the top 100 terms (as measured via TF-IDF scores) in the corpus when leveraging Word2Vec vectorization to transform terms into vectors. The nine experiments correspond to the results when using 100, 200, and 300 embedding dimensions on the terms gathered via data wrangling methods 1, 2, and 3. Larger versions of these heatmaps and the Python code leveraged to generate the heatmaps may be found in section 2.4 of Appendix J.



Appendix G – T-SNE Plot of the Top 100 TF-IDF Terms Using Word2Vec Vectorization

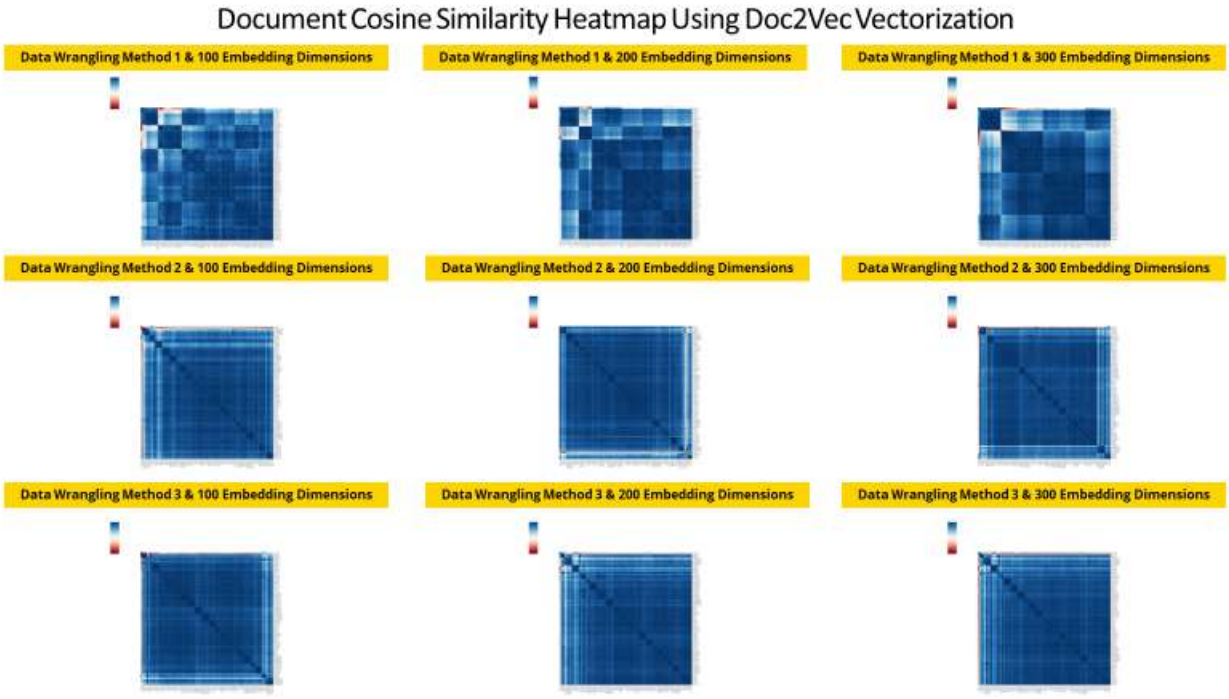
The T-SNE charts below visualize the embedding vectors of the top 100 terms (as measured via TF-IDF scores) in the corpus when leveraging Word2Vec vectorization to transform the terms into vectors. The nine experiments correspond to the results when using 100, 200, and 300 embedding dimensions on the terms gathered via data wrangling methods 1, 2, and 3. Larger versions of these T-SNE plots and the Python code leveraged to generate the T-SNE plots may be found in section 2.4 of Appendix J.

T-SNE Plot of Top 100 TF-IDF Terms Using Word2Vec Vectorization

Data Wrangling Method	Embedding Dimensions		
	100	200	300
Data Wrangling Method 1			
Data Wrangling Method 2			
Data Wrangling Method 3			

Appendix H – Document Cosine Similarity Heatmap Using Doc2Vec Vectorization

The heatmaps below display the cosine similarity measures between the documents in the corpus when leveraging Doc2Vec vectorization to transform documents into vectors. The nine experiments correspond to the results when using 100, 200, and 300 embedding dimensions on the documents gathered via data wrangling methods 1, 2, and 3. Larger versions of these heatmaps and the Python code leveraged to generate the heatmaps may be found in section 2.3 of Appendix J.



Appendix I – T-SNE Plot of Documents Using Doc2Vec Vectorization

The T-SNE charts below visualize the embedding vectors associated with each document in the corpus when leveraging Doc2Vec vectorization to transform the documents into vectors. The nine experiments correspond to the results when using 100, 200, and 300 embedding dimensions on the documents gathered via data wrangling methods 1, 2, and 3. Larger versions of these T-SNE plots and the Python code leveraged to generate the T-SNE plots may be found in section 2.3 of Appendix J.

T-SNE Plot of Documents Using Doc2Vec Vectorization

Data Wrangling Method	Embedding Dimensions		
	100	200	300
Data Wrangling Method 1			
Data Wrangling Method 2			
Data Wrangling Method 3			

Appendix J - Supporting Python Code

1) Qualitative Approach To Vocabulary Selection

1.1) Loading the Corpus

First, we will load many of the Python packages that will be needed for our analysis.

In [1]:
```
# Import Libraries
import pandas as pd
import os
import numpy as np
import re
import string
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
import random
from dataclasses import dataclass

from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem import PorterStemmer

import gensim
from gensim.models import Word2Vec
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.manifold import TSNE

import scipy.cluster.hierarchy

from IPython.display import display, HTML

from typing import List, Callable, Dict
```

In [2]:
```
# NLTK Downloads
# Only run this cell once. At that point, the appropriate resources will be downloaded
nltk.download('stopwords',quiet=True)
nltk.download('wordnet',quiet=True)
nltk.download('punkt',quiet=True)
nltk.download('omw-1.4',quiet=True)
```

Out[2]:
```
True
```

Gensim is a Python library for topic modelling, document indexing and similarity retrieval with large corpora. Target audience is the natural language processing (NLP) and information retrieval (IR) community

https://pypi.org/project/gensim/

In [3]:
```
import pkg_resources
```

In [4]:
```
# Print Gensim Version
print("Genism Version: ", gensim.__version__)
```

```
Genism Version:  4.1.2
```

In [5]:
```
# Suppress Warning Messages
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

Now, we may load the corpus for our natural language processing project.

In [6]:
```
# Create Document class
@dataclass
class Document:
    doc_id: str
    text: str

# Create corpus dataframe
def add_movie_descriptor(data: pd.DataFrame, corpus_df: pd.DataFrame):
    """
    Adds "Movie Description" to the supplied dataframe, in the form {Genre}_{P|N}_{Mov
    """
    review = np.where(corpus_df['Review Type (pos or neg)'] == 'Positive', 'P', 'N')
    data['Descriptor'] = corpus_df['Genre of Movie'] + '_' + corpus_df['Movie Title']

def get_corpus_df(path):
    data = pd.read_csv(path, encoding="utf-8")
    add_movie_descriptor(data, data)
    sorted_data = data.sort_values(['Descriptor'])
    indexed_data = sorted_data.set_index(['Doc_ID'])
    indexed_data['Doc_ID'] = indexed_data.index
    return indexed_data

# Define documents in this new class
corpus_df = get_corpus_df('MSDS453_ClassCorpus_Final_Sec56_v4_20230702.csv')
documents = [Document(x, y) for x, y in zip(corpus_df.Doc_ID, corpus_df.Text)]
```

1.2) Exploratory Data Analysis

Having imported the corpus of movie reviews, let's conduct exploratory data analysis on the raw dataset.

In [7]:
```
corpus_df.shape
```

Out[7]:
```
(190, 9)
```

In [8]:
```
corpus_df.head(4).T
```

Out[8]:

Doc_ID	101	102	103	1(
DSI_Title	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_Doc4_Covena
Submission File Name	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_Doc4_Covena
Student Name	SAR	SAR	SAR	SA
Genre of Movie	Action	Action	Action	Acti(
Review Type (pos or neg)	Negative	Negative	Negative	Negati
Movie Title	Covenant	Covenant	Covenant	Covena
Text	Nearly two years after the American military w...	Have Guy Ritchie and Jake Gyllenhaal switched ...	Guy Ritchie's The Covenant notably marks the f...	In a weird throwback those Chuck Nor "M
Descriptor	Action_Covenant_N_101	Action_Covenant_N_102	Action_Covenant_N_103	Action_Covenant_N_1(
Doc_ID	101	102	103	1(

In [9]:
```
print(corpus_df.info());
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 190 entries, 101 to 217
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   DSI_Title                190 non-null    object
 1   Submission File Name     190 non-null    object
 2   Student Name             190 non-null    object
 3   Genre of Movie           190 non-null    object
 4   Review Type (pos or neg) 190 non-null    object
 5   Movie Title              190 non-null    object
 6   Text                     190 non-null    object
 7   Descriptor               190 non-null    object
 8   Doc_ID                   190 non-null    int64 
dtypes: int64(1), object(8)
memory usage: 14.8+ KB
None
```

In [10]:
```
print(corpus_df['Movie Title'].unique())
```

```
['Covenant' 'Inception' 'No time to die' 'Taken' 'The Dark Knight Rises'
 'Despicable Me 3' 'Holmes and Watson' 'Legally Blonde' 'Lost City'
 'Sisters' 'Drag Me to Hell' 'Fresh' 'It Chapter Two' 'The Toxic Avenger'
 'US' 'Annihilation' 'Minority Report' 'Oblivion' 'Pitch Black']
```

In [11]:
```
# Gather the number of reviews by genre
counts_df = corpus_df[['Genre of Movie']].copy()
counts_df['Count'] = 1
counts_df.groupby(['Genre of Movie']).count().reset_index()
```

Out[11]:

	Genre of Movie	Count
0	Action	50
1	Comedy	50
2	Horror	50
3	Sci-Fi	40

In [12]:
```
corpus_df.columns
```

Out[12]:
```
Index(['DSI_Title', 'Submission File Name', 'Student Name', 'Genre of Movie',
       'Review Type (pos or neg)', 'Movie Title', 'Text', 'Descriptor',
       'Doc_ID'],
      dtype='object')
```

1.3) Normalization and Tokenization

Let's conduct normalization of our movie review corpus by removing punctuation transforming text to lowercase, removing HTML tags.

In [13]:
```
def remove_punctuation(text):
    return re.sub('[^a-zA-Z]', ' ', str(text))

def lower_case(text):
    return text.lower()

def remove_tags(text):
    return re.sub("&lt;/?.*?&gt;"," &lt;&gt; ", text)

def normalize_document(document: Document) -> Document:
    text = document.text
    text = remove_punctuation(text)
    text = lower_case(text)
    text = remove_tags(text)

    return Document(document.doc_id, text)

def normalize_documents(documents: List[Document]) -> List[Document]:
    """
    Normalizes text for all given documents.
    Removes punctuation, converts to lower case, removes tags and special characters.
    """
    return [normalize_document(x) for x in documents]

normalized_documents = normalize_documents(documents)
normalized_documents[0]
```

Out[13]:
```
Document(doc_id=101, text='nearly two years after the american military withdrew from
afghanistan  ending the longest war in u s  history  it should come as no surprise th
at filmmakers are starting to look back and ask  what just happened  that s what ar
tists do  after all  they examine the world around them  that guy ritchie is one of t
hem in this case  however  that might be a surprise  heretofore  ritchie has for the
most part been one of the fun ones  building his name on stylish action  well timed o
ne liners and heavier than a lead zeppelin british accents  substance  nah  guv  no n
eed  but in his latest  the war drama  the covenant  starring jake gyllenhaal and se
t during the war in afghanistan  he s got something to say  this is serious guy ritch
ie  this is thoughtful guy ritchie  this is earnest guy ritchie  if only profound guy
ritchie had come along for the trip  we might be onto something special  instead  we
get a capably made but simple minded rescue film  one with a big heart  even if it c
ould have used more brain  like his movie  ritchie s message is straightforward thoug
h honorable  real men live up to their commitments  by extension  his film argues  co
untries should do the same  at the heart of things  in this case  are the promises ma
de to the small army of interpreters in afghanistan who aided the u s  military durin
g its two decades there  in exchange for their service  many were promised travel vis
as for their families  and then unceremoniously left behind when the war so abruptly
ended  for storytelling purposes  the covenant  on which ritchie shares writing cr
edit with ivan atkinson and marn davies  focuses on u s  army sgt  john kinley  play
ed by gyllenhaal  he leads a squad of soldiers tasked with finding and destroying tal
iban bomb factories  it s dangerous work  as we learn early on in a scene that leaves
kinley s small squad in need of a new interpreter  enter ahmed  dar salim  who is fa
r more than your garden variety interpreter  he has skills  this ahmed  he also has a
very personal motivation for helping the americans  it s an undeniably compelling sto
ry  although it should be noted it s also a fictional one  ritchie s film pretends ve
ry  very hard to be based on actual events  it is not  although it was inspired by th
e plight of many afghan interpreters left behind in the u s  withdrawal  those storie
s are well worth sharing  the covenant  can be criticized for a number of things  bu
t there s no questioning its noble heart  like gyllenhaal s sgt  kinley  salim s ahme
d is the strong  silent type  the kind of guy who exudes confidence and capability  t
hey drink their coffee black  these guys  they grunt  they are warriors  these are th
e kind of men whose highest compliment to one another is a solemn exchange of knowing
nods  in silent recognition of each other s towering manliness  but before they get t
here  they first must learn to trust one another  which is the central purpose of the
first hour of')
```

Let's tokenize the movie review corpus to divide the documents from strings into lists of substrings.

In [14]:
```python
@dataclass
class TokenizedDocument:
    doc_id: str
    tokens: List[str]

def tokenize_document(document: Document) -> TokenizedDocument:
    tokens = nltk.word_tokenize(document.text)
    return TokenizedDocument(document.doc_id, tokens)

def tokenize_documents(documents: List[Document]) -> List[TokenizedDocument]:
    return [tokenize_document(x) for x in documents]

tokenized_documents = tokenize_documents(normalized_documents)
tokenized_documents[0]
```

Out[14]:
```
TokenizedDocument(doc_id=101, tokens=['nearly', 'two', 'years', 'after', 'the', 'amer
ican', 'military', 'withdrew', 'from', 'afghanistan', 'ending', 'the', 'longest', 'wa
r', 'in', 'u', 's', 'history', 'it', 'should', 'come', 'as', 'no', 'surprise', 'tha
t', 'filmmakers', 'are', 'starting', 'to', 'look', 'back', 'and', 'ask', 'what', 'jus
t', 'happened', 'that', 's', 'what', 'artists', 'do', 'after', 'all', 'they', 'examin
e', 'the', 'world', 'around', 'them', 'that', 'guy', 'ritchie', 'is', 'one', 'of', 't
hem', 'in', 'this', 'case', 'however', 'that', 'might', 'be', 'a', 'surprise', 'heret
ofore', 'ritchie', 'has', 'for', 'the', 'most', 'part', 'been', 'one', 'of', 'the',
'fun', 'ones', 'building', 'his', 'name', 'on', 'stylish', 'action', 'well', 'timed',
'one', 'liners', 'and', 'heavier', 'than', 'a', 'lead', 'zeppelin', 'british', 'accen
ts', 'substance', 'nah', 'guv', 'no', 'need', 'but', 'in', 'his', 'latest', 'the', 'w
ar', 'drama', 'the', 'covenant', 'starring', 'jake', 'gyllenhaal', 'and', 'set', 'dur
ing', 'the', 'war', 'in', 'afghanistan', 'he', 's', 'got', 'something', 'to', 'say',
'this', 'is', 'serious', 'guy', 'ritchie', 'this', 'is', 'thoughtful', 'guy', 'ritchi
e', 'this', 'is', 'earnest', 'guy', 'ritchie', 'if', 'only', 'profound', 'guy', 'ritc
hie', 'had', 'come', 'along', 'for', 'the', 'trip', 'we', 'might', 'be', 'onto', 'som
ething', 'special', 'instead', 'we', 'get', 'a', 'capably', 'made', 'but', 'simple',
'minded', 'rescue', 'film', 'one', 'with', 'a', 'big', 'heart', 'even', 'if', 'it',
'could', 'have', 'used', 'more', 'brain', 'like', 'his', 'movie', 'ritchie', 's', 'me
ssage', 'is', 'straightforward', 'though', 'honorable', 'real', 'men', 'live', 'up',
'to', 'their', 'commitments', 'by', 'extension', 'his', 'film', 'argues', 'countrie
s', 'should', 'do', 'the', 'same', 'at', 'the', 'heart', 'of', 'things', 'in', 'thi
s', 'case', 'are', 'the', 'promises', 'made', 'to', 'the', 'small', 'army', 'of', 'in
terpreters', 'in', 'afghanistan', 'who', 'aided', 'the', 'u', 's', 'military', 'durin
g', 'its', 'two', 'decades', 'there', 'in', 'exchange', 'for', 'their', 'service', 'm
any', 'were', 'promised', 'travel', 'visas', 'for', 'their', 'families', 'and', 'the
n', 'unceremoniously', 'left', 'behind', 'when', 'the', 'war', 'so', 'abruptly', 'end
ed', 'for', 'storytelling', 'purposes', 'the', 'covenant', 'on', 'which', 'ritchie',
'shares', 'writing', 'credit', 'with', 'ivan', 'atkinson', 'and', 'marn', 'davies',
'focuses', 'on', 'u', 's', 'army', 'sgt', 'john', 'kinley', 'played', 'by', 'gyllenha
al', 'he', 'leads', 'a', 'squad', 'of', 'soldiers', 'tasked', 'with', 'finding', 'an
d', 'destroying', 'taliban', 'bomb', 'factories', 'it', 's', 'dangerous', 'work', 'a
s', 'we', 'learn', 'early', 'on', 'in', 'a', 'scene', 'that', 'leaves', 'kinley',
's', 'small', 'squad', 'in', 'need', 'of', 'a', 'new', 'interpreter', 'enter', 'ahme
d', 'dar', 'salim', 'who', 'is', 'far', 'more', 'than', 'your', 'garden', 'variety',
'interpreter', 'he', 'has', 'skills', 'this', 'ahmed', 'he', 'also', 'has', 'a', 'ver
y', 'personal', 'motivation', 'for', 'helping', 'the', 'americans', 'it', 's', 'an',
'undeniably', 'compelling', 'story', 'although', 'it', 'should', 'be', 'noted', 'it',
's', 'also', 'a', 'fictional', 'one', 'ritchie', 's', 'film', 'pretends', 'very', 've
ry', 'hard', 'to', 'be', 'based', 'on', 'actual', 'events', 'it', 'is', 'not', 'altho
ugh', 'it', 'was', 'inspired', 'by', 'the', 'plight', 'of', 'many', 'afghan', 'interp
reters', 'left', 'behind', 'in', 'the', 'u', 's', 'withdrawal', 'those', 'stories',
'are', 'well', 'worth', 'sharing', 'the', 'covenant', 'can', 'be', 'criticized', 'fo
r', 'a', 'number', 'of', 'things', 'but', 'there', 's', 'no', 'questioning', 'its',
'noble', 'heart', 'like', 'gyllenhaal', 's', 'sgt', 'kinley', 'salim', 's', 'ahmed',
'is', 'the', 'strong', 'silent', 'type', 'the', 'kind', 'of', 'guy', 'who', 'exudes',
'confidence', 'and', 'capability', 'they', 'drink', 'their', 'coffee', 'black', 'thes
e', 'guys', 'they', 'grunt', 'they', 'are', 'warriors', 'these', 'are', 'the', 'kin
d', 'of', 'men', 'whose', 'highest', 'compliment', 'to', 'one', 'another', 'is', 'a',
'solemn', 'exchange', 'of', 'knowing', 'nods', 'in', 'silent', 'recognition', 'of',
'each', 'other', 's', 'towering', 'manliness', 'but', 'before', 'they', 'get', 'ther
e', 'they', 'first', 'must', 'learn', 'to', 'trust', 'one', 'another', 'which', 'is',
'the', 'central', 'purpose', 'of', 'the', 'first', 'hour', 'of'])
```

1.4) Definition of Useful Lookups

In [15]:
```
#index(['Doc_ID', 'DSI_Title', 'Submission File Name', 'Student Name',
#       'Genre of Movie', 'Review Type (pos or neg)', 'Movie Title', 'Text'],
#      dtype='object')
```

In [16]:
```
titles_by_doc_ids = {x: y for x, y in zip(corpus_df['Doc_ID'], corpus_df['Movie Title'
genres_by_doc_ids = {x: y for x, y in zip(corpus_df['Doc_ID'], corpus_df['Genre of Mov
descriptors_by_doc_ids = {x: y for x, y in zip(corpus_df['Doc_ID'], corpus_df['Descrip

comedy_doc_ids = [int(x) for x in corpus_df['Doc_ID'] if genres_by_doc_ids[x] == 'Come
comedy_documents = [x for x in documents if x.doc_id in comedy_doc_ids]

non_comedy_doc_ids = {int(x) for x in corpus_df['Doc_ID'] if genres_by_doc_ids[x] != '
non_comedy_documents = [x for x in documents if x.doc_id in non_comedy_doc_ids]
```

In [17]:
```
print(corpus_df['Movie Title'].unique())
```

```
['Covenant' 'Inception' 'No time to die' 'Taken' 'The Dark Knight Rises'
 'Despicable Me 3' 'Holmes and Watson' 'Legally Blonde' 'Lost City'
 'Sisters' 'Drag Me to Hell' 'Fresh' 'It Chapter Two' 'The Toxic Avenger'
 'US' 'Annihilation' 'Minority Report' 'Oblivion' 'Pitch Black']
```

In [18]:
```
sisters_doc_ids = [int(x) for x in corpus_df['Doc_ID'] if titles_by_doc_ids[x] == 'Sis
sisters_documents = [x for x in documents if x.doc_id in sisters_doc_ids]
```

1.5) Qualitative Term Determinations

Let's define some terms that we believe will be useful for grouping "Sisters" with other comedy movies prior to our more formal analysis leveraging the actual corpus.

In [19]:
```
candidate_terms = [
    'character',
    'comedy',
    'family',
    'funny',
    'friend',
    'humor',
    'joke',
    'laugh',
    'love',
    'party',
    'script',
    'school',
    'smile'
]
```

1.6) CountVectorizer

Convert a collection of text documents to a matrix of token counts. This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix. If you do not provide an a-priori dictionary and you do not use an analyzer that does some kind of feature selection then the number of features will be equal to the vocabulary size found by analyzing the data.

In [20]:
```
vectorizer = CountVectorizer(ngram_range=(1, 1))
text_for_counts = [x.text for x in normalized_documents]
matrix = vectorizer.fit_transform(text_for_counts)
```

```
words = vectorizer.get_feature_names_out()
word_counts = pd.DataFrame(matrix.toarray(), columns=words, index=corpus_df.Doc_ID)
```

In [21]:
```
def add_flags(data: pd.DataFrame, sisters_doc_ids: List[int], comedy_doc_ids: List[int
    data['is_sisters'] = data.index.isin(sisters_doc_ids)
    data['is_comedy'] = data.index.isin(comedy_doc_ids)

add_flags(word_counts, sisters_doc_ids, comedy_doc_ids)

word_counts['Doc_ID'] = word_counts.index
```

In [22]:
```
# Collect result into a dataframe
mean_frequencies = pd.DataFrame(index=candidate_terms)

sisters_mean_frequencies = word_counts[word_counts.is_sisters][[x for x in candidate_t
mean_frequencies['Sisters'] = sisters_mean_frequencies

comedy_mean_frequencies = word_counts[word_counts.is_comedy][[x for x in candidate_ter
mean_frequencies['All Comedy'] = comedy_mean_frequencies

non_comedy_mean_frequencies = word_counts[~word_counts.is_comedy][[x for x in candidat
mean_frequencies['All Non-Comedy'] = non_comedy_mean_frequencies
```

In [23]:
```
mean_frequencies.fillna(0.0).sort_values(['Sisters'], ascending=False)
```

Out[23]:

	Sisters	All Comedy	All Non-Comedy
party	2.4	0.54	0.057143
comedy	1.4	1.08	0.214286
funny	0.8	0.50	0.135714
character	0.6	0.56	0.485714
family	0.6	0.58	0.321429
joke	0.6	0.38	0.050000
humor	0.5	0.32	0.078571
friend	0.4	0.12	0.128571
love	0.3	0.26	0.242857
script	0.3	0.16	0.150000
school	0.3	0.68	0.021429
laugh	0.1	0.14	0.064286
smile	0.0	0.06	0.028571

In [24]:
```
important_prevalent_terms = [
    'party',
    'comedy',
    'funny',
    'joke'
]
stemmer = PorterStemmer()
stemmed_important_prevalent_terms = [stemmer.stem(x) for x in important_prevalent_term
```

```
lemmatizer = WordNetLemmatizer()
lemmatized_important_prevalent_terms = [lemmatizer.lemmatize(x) for x in important_pre
```

In [25]:
```
pd.options.display.float_format = '{:,.2f}'.format
mean_frequencies.fillna(0.0).loc[important_prevalent_terms].round(2).sort_values(['Sis
```

Out[25]:

	Sisters	All Comedy	All Non-Comedy
party	2.40	0.54	0.06
comedy	1.40	1.08	0.21
funny	0.80	0.50	0.14
joke	0.60	0.38	0.05

2) Quantitative Approach To Vocabulary Selection

2.1) Data Wrangling

2.1.1) Data Wrangling - Method 1

Let's complete our first methodology for data wrangling. Our first methodology will include normalization (including removing punctuation, converting capital letters into lowercase letters, removing HTML tags, and stemming), and tokenization.

In [26]:
```
# Normalization

def normalize_document_method_one(document: Document) -> Document:
    text = document.text
    text = remove_punctuation(text)
    text = lower_case(text)
    text = remove_tags(text)

    return Document(document.doc_id, text)

def normalize_documents_method_one(documents: List[Document]) -> List[Document]:
    """
    Normalizes text for all given documents.
    Removes punctuation, converts to lower case, and removes tags.
    """
    return [normalize_document_method_one(x) for x in documents]

normalized_documents_method_one = normalize_documents_method_one(documents)
normalized_documents_method_one[0]
```

Out[26]:
```
Document(doc_id=101, text='nearly two years after the american military withdrew from afghanistan  ending the longest war in u s  history  it should come as no surprise th at filmmakers are starting to look back and ask  what just happened  that s what ar tists do  after all  they examine the world around them  that guy ritchie is one of t hem in this case  however  that might be a surprise  heretofore  ritchie has for the most part been one of the fun ones  building his name on stylish action  well timed o ne liners and heavier than a lead zeppelin british accents  substance  nah  guv  no n eed  but in his latest  the war drama  the covenant  starring jake gyllenhaal and se t during the war in afghanistan  he s got something to say  this is serious guy ritch ie  this is thoughtful guy ritchie  this is earnest guy ritchie  if only profound guy ritchie had come along for the trip  we might be onto something special  instead  we get a capably made but simple minded rescue film  one with a big heart  even if it c ould have used more brain  like his movie  ritchie s message is straightforward thoug h honorable  real men live up to their commitments  by extension  his film argues  co untries should do the same  at the heart of things  in this case  are the promises ma de to the small army of interpreters in afghanistan who aided the u s  military durin g its two decades there  in exchange for their service  many were promised travel vis as for their families  and then unceremoniously left behind when the war so abruptly ended  for storytelling purposes  the covenant  on which ritchie shares writing cr edit with ivan atkinson and marn davies  focuses on u s  army sgt  john kinley  play ed by gyllenhaal  he leads a squad of soldiers tasked with finding and destroying tal iban bomb factories  it s dangerous work  as we learn early on in a scene that leaves kinley s small squad in need of a new interpreter  enter ahmed  dar salim  who is fa r more than your garden variety interpreter  he has skills  this ahmed  he also has a very personal motivation for helping the americans  it s an undeniably compelling sto ry  although it should be noted it s also a fictional one  ritchie s film pretends ve ry  very hard to be based on actual events  it is not  although it was inspired by th e plight of many afghan interpreters left behind in the u s  withdrawal  those storie s are well worth sharing  the covenant  can be criticized for a number of things  bu t there s no questioning its noble heart  like gyllenhaal s sgt  kinley  salim s ahme d is the strong  silent type  the kind of guy who exudes confidence and capability  t hey drink their coffee black  these guys  they grunt  they are warriors  these are th e kind of men whose highest compliment to one another is a solemn exchange of knowing nods  in silent recognition of each other s towering manliness  but before they get t here  they first must learn to trust one another  which is the central purpose of the first hour of')
```

In [27]:
```
# Tokenization
tokenized_documents_method_one = tokenize_documents(normalized_documents_method_one)
tokenized_documents_method_one[0]
```

Out[27]:
```
TokenizedDocument(doc_id=101, tokens=['nearly', 'two', 'years', 'after', 'the', 'amer
ican', 'military', 'withdrew', 'from', 'afghanistan', 'ending', 'the', 'longest', 'wa
r', 'in', 'u', 's', 'history', 'it', 'should', 'come', 'as', 'no', 'surprise', 'tha
t', 'filmmakers', 'are', 'starting', 'to', 'look', 'back', 'and', 'ask', 'what', 'jus
t', 'happened', 'that', 's', 'what', 'artists', 'do', 'after', 'all', 'they', 'examin
e', 'the', 'world', 'around', 'them', 'that', 'guy', 'ritchie', 'is', 'one', 'of', 't
hem', 'in', 'this', 'case', 'however', 'that', 'might', 'be', 'a', 'surprise', 'heret
ofore', 'ritchie', 'has', 'for', 'the', 'most', 'part', 'been', 'one', 'of', 'the',
'fun', 'ones', 'building', 'his', 'name', 'on', 'stylish', 'action', 'well', 'timed',
'one', 'liners', 'and', 'heavier', 'than', 'a', 'lead', 'zeppelin', 'british', 'accen
ts', 'substance', 'nah', 'guv', 'no', 'need', 'but', 'in', 'his', 'latest', 'the', 'w
ar', 'drama', 'the', 'covenant', 'starring', 'jake', 'gyllenhaal', 'and', 'set', 'dur
ing', 'the', 'war', 'in', 'afghanistan', 'he', 's', 'got', 'something', 'to', 'say',
'this', 'is', 'serious', 'guy', 'ritchie', 'this', 'is', 'thoughtful', 'guy', 'ritchi
e', 'this', 'is', 'earnest', 'guy', 'ritchie', 'if', 'only', 'profound', 'guy', 'ritc
hie', 'had', 'come', 'along', 'for', 'the', 'trip', 'we', 'might', 'be', 'onto', 'som
ething', 'special', 'instead', 'we', 'get', 'a', 'capably', 'made', 'but', 'simple',
'minded', 'rescue', 'film', 'one', 'with', 'a', 'big', 'heart', 'even', 'if', 'it',
'could', 'have', 'used', 'more', 'brain', 'like', 'his', 'movie', 'ritchie', 's', 'me
ssage', 'is', 'straightforward', 'though', 'honorable', 'real', 'men', 'live', 'up',
'to', 'their', 'commitments', 'by', 'extension', 'his', 'film', 'argues', 'countrie
s', 'should', 'do', 'the', 'same', 'at', 'the', 'heart', 'of', 'things', 'in', 'thi
s', 'case', 'are', 'the', 'promises', 'made', 'to', 'the', 'small', 'army', 'of', 'in
terpreters', 'in', 'afghanistan', 'who', 'aided', 'the', 'u', 's', 'military', 'durin
g', 'its', 'two', 'decades', 'there', 'in', 'exchange', 'for', 'their', 'service', 'm
any', 'were', 'promised', 'travel', 'visas', 'for', 'their', 'families', 'and', 'the
n', 'unceremoniously', 'left', 'behind', 'when', 'the', 'war', 'so', 'abruptly', 'end
ed', 'for', 'storytelling', 'purposes', 'the', 'covenant', 'on', 'which', 'ritchie',
'shares', 'writing', 'credit', 'with', 'ivan', 'atkinson', 'and', 'marn', 'davies',
'focuses', 'on', 'u', 's', 'army', 'sgt', 'john', 'kinley', 'played', 'by', 'gyllenha
al', 'he', 'leads', 'a', 'squad', 'of', 'soldiers', 'tasked', 'with', 'finding', 'an
d', 'destroying', 'taliban', 'bomb', 'factories', 'it', 's', 'dangerous', 'work', 'a
s', 'we', 'learn', 'early', 'on', 'in', 'a', 'scene', 'that', 'leaves', 'kinley',
's', 'small', 'squad', 'in', 'need', 'of', 'a', 'new', 'interpreter', 'enter', 'ahme
d', 'dar', 'salim', 'who', 'is', 'far', 'more', 'than', 'your', 'garden', 'variety',
'interpreter', 'he', 'has', 'skills', 'this', 'ahmed', 'he', 'also', 'has', 'a', 'ver
y', 'personal', 'motivation', 'for', 'helping', 'the', 'americans', 'it', 's', 'an',
'undeniably', 'compelling', 'story', 'although', 'it', 'should', 'be', 'noted', 'it',
's', 'also', 'a', 'fictional', 'one', 'ritchie', 's', 'film', 'pretends', 'very', 've
ry', 'hard', 'to', 'be', 'based', 'on', 'actual', 'events', 'it', 'is', 'not', 'altho
ugh', 'it', 'was', 'inspired', 'by', 'the', 'plight', 'of', 'many', 'afghan', 'interp
reters', 'left', 'behind', 'in', 'the', 'u', 's', 'withdrawal', 'those', 'stories',
'are', 'well', 'worth', 'sharing', 'the', 'covenant', 'can', 'be', 'criticized', 'fo
r', 'a', 'number', 'of', 'things', 'but', 'there', 's', 'no', 'questioning', 'its',
'noble', 'heart', 'like', 'gyllenhaal', 's', 'sgt', 'kinley', 'salim', 's', 'ahmed',
'is', 'the', 'strong', 'silent', 'type', 'the', 'kind', 'of', 'guy', 'who', 'exudes',
'confidence', 'and', 'capability', 'they', 'drink', 'their', 'coffee', 'black', 'thes
e', 'guys', 'they', 'grunt', 'they', 'are', 'warriors', 'these', 'are', 'the', 'kin
d', 'of', 'men', 'whose', 'highest', 'compliment', 'to', 'one', 'another', 'is', 'a',
'solemn', 'exchange', 'of', 'knowing', 'nods', 'in', 'silent', 'recognition', 'of',
'each', 'other', 's', 'towering', 'manliness', 'but', 'before', 'they', 'get', 'ther
e', 'they', 'first', 'must', 'learn', 'to', 'trust', 'one', 'another', 'which', 'is',
'the', 'central', 'purpose', 'of', 'the', 'first', 'hour', 'of'])
```

In [28]:
```
# Stemming
def stem(documents: List[TokenizedDocument]) -> List[TokenizedDocument]:
    result = []
    stemmer = PorterStemmer()
    for document in documents:
        output_tokens = [stemmer.stem(w) for w in document.tokens]
```

```
        result.append(TokenizedDocument(document.doc_id, output_tokens))

    return result

tokenized_documents_method_one = stem(tokenized_documents_method_one)
```

2.1.2) Data Wrangling - Method 2

Let's complete our second methodology for data wrangling. Like the first methodology, this methodology will also include normalization (including removing punctuation, converting capital letters into lowercase letters, and removing HTML tags) and tokenization. However, this methodology will differ because we will remove common stop words and will leverage lemmatization instead of stemming.

In [29]:
```
# Normalization

def remove_special_chars_and_digits(text):
    return re.sub("(\\d|\\W)+"," ", text)

def normalize_document_method_two(document: Document) -> Document:
    text = document.text
    text = remove_punctuation(text)
    text = lower_case(text)
    text = remove_tags(text)
    text = remove_special_chars_and_digits(text)

    return Document(document.doc_id, text)

def normalize_documents_method_two(documents: List[Document]) -> List[Document]:
    """
    Normalizes text for all given documents.
    Removes punctuation, converts to lower case, removes tags and special characters.
    """
    return [normalize_document_method_two(x) for x in documents]

normalized_documents_method_two = normalize_documents_method_two(documents)
normalized_documents_method_two[0]
```

Out[29]:
```
Document(doc_id=101, text='nearly two years after the american military withdrew from
afghanistan ending the longest war in u s history it should come as no surprise that
filmmakers are starting to look back and ask what just happened that s what artists d
o after all they examine the world around them that guy ritchie is one of them in thi
s case however that might be a surprise heretofore ritchie has for the most part been
one of the fun ones building his name on stylish action well timed one liners and hea
vier than a lead zeppelin british accents substance nah guv no need but in his latest
the war drama the covenant starring jake gyllenhaal and set during the war in afghani
stan he s got something to say this is serious guy ritchie this is thoughtful guy rit
chie this is earnest guy ritchie if only profound guy ritchie had come along for the
trip we might be onto something special instead we get a capably made but simple mind
ed rescue film one with a big heart even if it could have used more brain like his mo
vie ritchie s message is straightforward though honorable real men live up to their c
ommitments by extension his film argues countries should do the same at the heart of
things in this case are the promises made to the small army of interpreters in afghan
istan who aided the u s military during its two decades there in exchange for their s
ervice many were promised travel visas for their families and then unceremoniously le
ft behind when the war so abruptly ended for storytelling purposes the covenant on wh
ich ritchie shares writing credit with ivan atkinson and marn davies focuses on u s a
rmy sgt john kinley played by gyllenhaal he leads a squad of soldiers tasked with fin
ding and destroying taliban bomb factories it s dangerous work as we learn early on i
n a scene that leaves kinley s small squad in need of a new interpreter enter ahmed d
ar salim who is far more than your garden variety interpreter he has skills this ahme
d he also has a very personal motivation for helping the americans it s an undeniably
compelling story although it should be noted it s also a fictional one ritchie s film
pretends very very hard to be based on actual events it is not although it was inspir
ed by the plight of many afghan interpreters left behind in the u s withdrawal those
stories are well worth sharing the covenant can be criticized for a number of things
but there s no questioning its noble heart like gyllenhaal s sgt kinley salim s ahmed
is the strong silent type the kind of guy who exudes confidence and capability they d
rink their coffee black these guys they grunt they are warriors these are the kind of
men whose highest compliment to one another is a solemn exchange of knowing nods in s
ilent recognition of each other s towering manliness but before they get there they f
irst must learn to trust one another which is the central purpose of the first hour o
f')
```

In [30]:
```
# Tokenization

tokenized_documents_method_two = tokenize_documents(normalized_documents_method_two)
tokenized_documents_method_two[0]
```

Out[30]:
```
TokenizedDocument(doc_id=101, tokens=['nearly', 'two', 'years', 'after', 'the', 'amer
ican', 'military', 'withdrew', 'from', 'afghanistan', 'ending', 'the', 'longest', 'wa
r', 'in', 'u', 's', 'history', 'it', 'should', 'come', 'as', 'no', 'surprise', 'tha
t', 'filmmakers', 'are', 'starting', 'to', 'look', 'back', 'and', 'ask', 'what', 'jus
t', 'happened', 'that', 's', 'what', 'artists', 'do', 'after', 'all', 'they', 'examin
e', 'the', 'world', 'around', 'them', 'that', 'guy', 'ritchie', 'is', 'one', 'of', 't
hem', 'in', 'this', 'case', 'however', 'that', 'might', 'be', 'a', 'surprise', 'heret
ofore', 'ritchie', 'has', 'for', 'the', 'most', 'part', 'been', 'one', 'of', 'the',
'fun', 'ones', 'building', 'his', 'name', 'on', 'stylish', 'action', 'well', 'timed',
'one', 'liners', 'and', 'heavier', 'than', 'a', 'lead', 'zeppelin', 'british', 'accen
ts', 'substance', 'nah', 'guv', 'no', 'need', 'but', 'in', 'his', 'latest', 'the', 'w
ar', 'drama', 'the', 'covenant', 'starring', 'jake', 'gyllenhaal', 'and', 'set', 'dur
ing', 'the', 'war', 'in', 'afghanistan', 'he', 's', 'got', 'something', 'to', 'say',
'this', 'is', 'serious', 'guy', 'ritchie', 'this', 'is', 'thoughtful', 'guy', 'ritchi
e', 'this', 'is', 'earnest', 'guy', 'ritchie', 'if', 'only', 'profound', 'guy', 'ritc
hie', 'had', 'come', 'along', 'for', 'the', 'trip', 'we', 'might', 'be', 'onto', 'som
ething', 'special', 'instead', 'we', 'get', 'a', 'capably', 'made', 'but', 'simple',
'minded', 'rescue', 'film', 'one', 'with', 'a', 'big', 'heart', 'even', 'if', 'it',
'could', 'have', 'used', 'more', 'brain', 'like', 'his', 'movie', 'ritchie', 's', 'me
ssage', 'is', 'straightforward', 'though', 'honorable', 'real', 'men', 'live', 'up',
'to', 'their', 'commitments', 'by', 'extension', 'his', 'film', 'argues', 'countrie
s', 'should', 'do', 'the', 'same', 'at', 'the', 'heart', 'of', 'things', 'in', 'thi
s', 'case', 'are', 'the', 'promises', 'made', 'to', 'the', 'small', 'army', 'of', 'in
terpreters', 'in', 'afghanistan', 'who', 'aided', 'the', 'u', 's', 'military', 'durin
g', 'its', 'two', 'decades', 'there', 'in', 'exchange', 'for', 'their', 'service', 'm
any', 'were', 'promised', 'travel', 'visas', 'for', 'their', 'families', 'and', 'the
n', 'unceremoniously', 'left', 'behind', 'when', 'the', 'war', 'so', 'abruptly', 'end
ed', 'for', 'storytelling', 'purposes', 'the', 'covenant', 'on', 'which', 'ritchie',
'shares', 'writing', 'credit', 'with', 'ivan', 'atkinson', 'and', 'marn', 'davies',
'focuses', 'on', 'u', 's', 'army', 'sgt', 'john', 'kinley', 'played', 'by', 'gyllenha
al', 'he', 'leads', 'a', 'squad', 'of', 'soldiers', 'tasked', 'with', 'finding', 'an
d', 'destroying', 'taliban', 'bomb', 'factories', 'it', 's', 'dangerous', 'work', 'a
s', 'we', 'learn', 'early', 'on', 'in', 'a', 'scene', 'that', 'leaves', 'kinley',
's', 'small', 'squad', 'in', 'need', 'of', 'a', 'new', 'interpreter', 'enter', 'ahme
d', 'dar', 'salim', 'who', 'is', 'far', 'more', 'than', 'your', 'garden', 'variety',
'interpreter', 'he', 'has', 'skills', 'this', 'ahmed', 'he', 'also', 'has', 'a', 'ver
y', 'personal', 'motivation', 'for', 'helping', 'the', 'americans', 'it', 's', 'an',
'undeniably', 'compelling', 'story', 'although', 'it', 'should', 'be', 'noted', 'it',
's', 'also', 'a', 'fictional', 'one', 'ritchie', 's', 'film', 'pretends', 'very', 've
ry', 'hard', 'to', 'be', 'based', 'on', 'actual', 'events', 'it', 'is', 'not', 'altho
ugh', 'it', 'was', 'inspired', 'by', 'the', 'plight', 'of', 'many', 'afghan', 'interp
reters', 'left', 'behind', 'in', 'the', 'u', 's', 'withdrawal', 'those', 'stories',
'are', 'well', 'worth', 'sharing', 'the', 'covenant', 'can', 'be', 'criticized', 'fo
r', 'a', 'number', 'of', 'things', 'but', 'there', 's', 'no', 'questioning', 'its',
'noble', 'heart', 'like', 'gyllenhaal', 's', 'sgt', 'kinley', 'salim', 's', 'ahmed',
'is', 'the', 'strong', 'silent', 'type', 'the', 'kind', 'of', 'guy', 'who', 'exudes',
'confidence', 'and', 'capability', 'they', 'drink', 'their', 'coffee', 'black', 'thes
e', 'guys', 'they', 'grunt', 'they', 'are', 'warriors', 'these', 'are', 'the', 'kin
d', 'of', 'men', 'whose', 'highest', 'compliment', 'to', 'one', 'another', 'is', 'a',
'solemn', 'exchange', 'of', 'knowing', 'nods', 'in', 'silent', 'recognition', 'of',
'each', 'other', 's', 'towering', 'manliness', 'but', 'before', 'they', 'get', 'ther
e', 'they', 'first', 'must', 'learn', 'to', 'trust', 'one', 'another', 'which', 'is',
'the', 'central', 'purpose', 'of', 'the', 'first', 'hour', 'of'])
```

In [31]:
```
# Lemmatization

def lemmatize(documents: List[TokenizedDocument]) -> List[TokenizedDocument]:
    result = []
    lemmatizer = WordNetLemmatizer()
    for document in documents:
```

```
        output_tokens = [lemmatizer.lemmatize(w) for w in document.tokens]
        result.append(TokenizedDocument(document.doc_id, output_tokens))

    return result

tokenized_documents_method_two = lemmatize(tokenized_documents_method_two)
```

In [32]:
```
# Stop Words Elimination

def remove_stop_words(documents: List[TokenizedDocument]) -> List[TokenizedDocument]:
    result = []

    stop_words = set(nltk.corpus.stopwords.words('english'))
    for document in documents:
        filtered_tokens = [w for w in document.tokens if not w in stop_words]
        result.append(TokenizedDocument(document.doc_id, filtered_tokens))

    return result

tokenized_documents_method_two = remove_stop_words(tokenized_documents_method_two)
```

2.1.3) Data Wrangling - Method 3

Let's complete our third methodology for data wrangling. This third methodology will be very similar to the second methodology for normalization and tokenization, except in addition, this methodology will also involve removing special characters and digits and removing custom stop words that may not be helpful for our particular NLP pipeline that will be focused on clustering movie reviews by movie genre.

In [33]:
```
# Normalization

def normalize_document_method_three(document: Document) -> Document:
    text = document.text
    text = remove_punctuation(text)
    text = lower_case(text)
    text = remove_tags(text)
    text = remove_special_chars_and_digits(text)

    return Document(document.doc_id, text)

def normalize_documents_method_three(documents: List[Document]) -> List[Document]:
    """
    Normalizes text for all given documents.
    Removes punctuation, converts to lower case, removes tags and special characters.
    """
    return [normalize_document_method_three(x) for x in documents]

normalized_documents_method_three = normalize_documents_method_three(documents)
normalized_documents_method_three[0]
```

Out[33]:
```
Document(doc_id=101, text='nearly two years after the american military withdrew from
afghanistan ending the longest war in u s history it should come as no surprise that
filmmakers are starting to look back and ask what just happened that s what artists d
o after all they examine the world around them that guy ritchie is one of them in thi
s case however that might be a surprise heretofore ritchie has for the most part been
one of the fun ones building his name on stylish action well timed one liners and hea
vier than a lead zeppelin british accents substance nah guv no need but in his latest
the war drama the covenant starring jake gyllenhaal and set during the war in afghani
stan he s got something to say this is serious guy ritchie this is thoughtful guy rit
chie this is earnest guy ritchie if only profound guy ritchie had come along for the
trip we might be onto something special instead we get a capably made but simple mind
ed rescue film one with a big heart even if it could have used more brain like his mo
vie ritchie s message is straightforward though honorable real men live up to their c
ommitments by extension his film argues countries should do the same at the heart of
things in this case are the promises made to the small army of interpreters in afghan
istan who aided the u s military during its two decades there in exchange for their s
ervice many were promised travel visas for their families and then unceremoniously le
ft behind when the war so abruptly ended for storytelling purposes the covenant on wh
ich ritchie shares writing credit with ivan atkinson and marn davies focuses on u s a
rmy sgt john kinley played by gyllenhaal he leads a squad of soldiers tasked with fin
ding and destroying taliban bomb factories it s dangerous work as we learn early on i
n a scene that leaves kinley s small squad in need of a new interpreter enter ahmed d
ar salim who is far more than your garden variety interpreter he has skills this ahme
d he also has a very personal motivation for helping the americans it s an undeniably
compelling story although it should be noted it s also a fictional one ritchie s film
pretends very very hard to be based on actual events it is not although it was inspir
ed by the plight of many afghan interpreters left behind in the u s withdrawal those
stories are well worth sharing the covenant can be criticized for a number of things
but there s no questioning its noble heart like gyllenhaal s sgt kinley salim s ahmed
is the strong silent type the kind of guy who exudes confidence and capability they d
rink their coffee black these guys they grunt they are warriors these are the kind of
men whose highest compliment to one another is a solemn exchange of knowing nods in s
ilent recognition of each other s towering manliness but before they get there they f
irst must learn to trust one another which is the central purpose of the first hour o
f')
```

In [34]:
```
# Tokenization

tokenized_documents_method_three = tokenize_documents(normalized_documents_method_thre
tokenized_documents_method_three[0]
```

Out[34]:
```
TokenizedDocument(doc_id=101, tokens=['nearly', 'two', 'years', 'after', 'the', 'amer
ican', 'military', 'withdrew', 'from', 'afghanistan', 'ending', 'the', 'longest', 'wa
r', 'in', 'u', 's', 'history', 'it', 'should', 'come', 'as', 'no', 'surprise', 'tha
t', 'filmmakers', 'are', 'starting', 'to', 'look', 'back', 'and', 'ask', 'what', 'jus
t', 'happened', 'that', 's', 'what', 'artists', 'do', 'after', 'all', 'they', 'examin
e', 'the', 'world', 'around', 'them', 'that', 'guy', 'ritchie', 'is', 'one', 'of', 't
hem', 'in', 'this', 'case', 'however', 'that', 'might', 'be', 'a', 'surprise', 'heret
ofore', 'ritchie', 'has', 'for', 'the', 'most', 'part', 'been', 'one', 'of', 'the',
'fun', 'ones', 'building', 'his', 'name', 'on', 'stylish', 'action', 'well', 'timed',
'one', 'liners', 'and', 'heavier', 'than', 'a', 'lead', 'zeppelin', 'british', 'accen
ts', 'substance', 'nah', 'guv', 'no', 'need', 'but', 'in', 'his', 'latest', 'the', 'w
ar', 'drama', 'the', 'covenant', 'starring', 'jake', 'gyllenhaal', 'and', 'set', 'dur
ing', 'the', 'war', 'in', 'afghanistan', 'he', 's', 'got', 'something', 'to', 'say',
'this', 'is', 'serious', 'guy', 'ritchie', 'this', 'is', 'thoughtful', 'guy', 'ritchi
e', 'this', 'is', 'earnest', 'guy', 'ritchie', 'if', 'only', 'profound', 'guy', 'ritc
hie', 'had', 'come', 'along', 'for', 'the', 'trip', 'we', 'might', 'be', 'onto', 'som
ething', 'special', 'instead', 'we', 'get', 'a', 'capably', 'made', 'but', 'simple',
'minded', 'rescue', 'film', 'one', 'with', 'a', 'big', 'heart', 'even', 'if', 'it',
'could', 'have', 'used', 'more', 'brain', 'like', 'his', 'movie', 'ritchie', 's', 'me
ssage', 'is', 'straightforward', 'though', 'honorable', 'real', 'men', 'live', 'up',
'to', 'their', 'commitments', 'by', 'extension', 'his', 'film', 'argues', 'countrie
s', 'should', 'do', 'the', 'same', 'at', 'the', 'heart', 'of', 'things', 'in', 'thi
s', 'case', 'are', 'the', 'promises', 'made', 'to', 'the', 'small', 'army', 'of', 'in
terpreters', 'in', 'afghanistan', 'who', 'aided', 'the', 'u', 's', 'military', 'durin
g', 'its', 'two', 'decades', 'there', 'in', 'exchange', 'for', 'their', 'service', 'm
any', 'were', 'promised', 'travel', 'visas', 'for', 'their', 'families', 'and', 'the
n', 'unceremoniously', 'left', 'behind', 'when', 'the', 'war', 'so', 'abruptly', 'end
ed', 'for', 'storytelling', 'purposes', 'the', 'covenant', 'on', 'which', 'ritchie',
'shares', 'writing', 'credit', 'with', 'ivan', 'atkinson', 'and', 'marn', 'davies',
'focuses', 'on', 'u', 's', 'army', 'sgt', 'john', 'kinley', 'played', 'by', 'gyllenha
al', 'he', 'leads', 'a', 'squad', 'of', 'soldiers', 'tasked', 'with', 'finding', 'an
d', 'destroying', 'taliban', 'bomb', 'factories', 'it', 's', 'dangerous', 'work', 'a
s', 'we', 'learn', 'early', 'on', 'in', 'a', 'scene', 'that', 'leaves', 'kinley',
's', 'small', 'squad', 'in', 'need', 'of', 'a', 'new', 'interpreter', 'enter', 'ahme
d', 'dar', 'salim', 'who', 'is', 'far', 'more', 'than', 'your', 'garden', 'variety',
'interpreter', 'he', 'has', 'skills', 'this', 'ahmed', 'he', 'also', 'has', 'a', 'ver
y', 'personal', 'motivation', 'for', 'helping', 'the', 'americans', 'it', 's', 'an',
'undeniably', 'compelling', 'story', 'although', 'it', 'should', 'be', 'noted', 'it',
's', 'also', 'a', 'fictional', 'one', 'ritchie', 's', 'film', 'pretends', 'very', 've
ry', 'hard', 'to', 'be', 'based', 'on', 'actual', 'events', 'it', 'is', 'not', 'altho
ugh', 'it', 'was', 'inspired', 'by', 'the', 'plight', 'of', 'many', 'afghan', 'interp
reters', 'left', 'behind', 'in', 'the', 'u', 's', 'withdrawal', 'those', 'stories',
'are', 'well', 'worth', 'sharing', 'the', 'covenant', 'can', 'be', 'criticized', 'fo
r', 'a', 'number', 'of', 'things', 'but', 'there', 's', 'no', 'questioning', 'its',
'noble', 'heart', 'like', 'gyllenhaal', 's', 'sgt', 'kinley', 'salim', 's', 'ahmed',
'is', 'the', 'strong', 'silent', 'type', 'the', 'kind', 'of', 'guy', 'who', 'exudes',
'confidence', 'and', 'capability', 'they', 'drink', 'their', 'coffee', 'black', 'thes
e', 'guys', 'they', 'grunt', 'they', 'are', 'warriors', 'these', 'are', 'the', 'kin
d', 'of', 'men', 'whose', 'highest', 'compliment', 'to', 'one', 'another', 'is', 'a',
'solemn', 'exchange', 'of', 'knowing', 'nods', 'in', 'silent', 'recognition', 'of',
'each', 'other', 's', 'towering', 'manliness', 'but', 'before', 'they', 'get', 'ther
e', 'they', 'first', 'must', 'learn', 'to', 'trust', 'one', 'another', 'which', 'is',
'the', 'central', 'purpose', 'of', 'the', 'first', 'hour', 'of'])
```

In [35]:
```
# Lemmatization

tokenized_documents_method_three = lemmatize(tokenized_documents_method_three)
```

In [36]:
```
# Stop Words Elimination

tokenized_documents_method_three = remove_stop_words(tokenized_documents_method_three)
```

In [37]:
```
# Removal of Custom Stop Words

def remove_custom_stop_words(documents: List[TokenizedDocument]) -> List[TokenizedDocu
    result = []

    custom_stop_words = set(['film', 'movie', 'ha', 'wa', 'get', 'make', 'also', 'much
    for document in documents:
        filtered_tokens = [w for w in document.tokens if not w in custom_stop_words]
        result.append(TokenizedDocument(document.doc_id, filtered_tokens))

    return result

tokenized_documents_method_three = remove_custom_stop_words(tokenized_documents_method
tokenized_documents_method_three[0]
```

Out[37]:
```
TokenizedDocument(doc_id=101, tokens=['nearly', 'two', 'year', 'american', 'militar
y', 'withdrew', 'afghanistan', 'ending', 'longest', 'war', 'u', 'history', 'surpris
e', 'filmmaker', 'starting', 'look', 'back', 'ask', 'happened', 'artist', 'examine',
'world', 'around', 'guy', 'ritchie', 'one', 'case', 'however', 'might', 'surprise',
'heretofore', 'ritchie', 'part', 'one', 'fun', 'one', 'building', 'name', 'stylish',
'action', 'well', 'timed', 'one', 'liner', 'heavier', 'lead', 'zeppelin', 'british',
'accent', 'substance', 'nah', 'guv', 'need', 'latest', 'war', 'drama', 'covenant', 's
tarring', 'jake', 'gyllenhaal', 'set', 'war', 'afghanistan', 'got', 'something', 'ser
ious', 'guy', 'ritchie', 'thoughtful', 'guy', 'ritchie', 'earnest', 'guy', 'ritchie',
'profound', 'guy', 'ritchie', 'along', 'trip', 'might', 'onto', 'something', 'specia
l', 'instead', 'capably', 'simple', 'minded', 'rescue', 'one', 'big', 'heart', 'eve
n', 'could', 'used', 'brain', 'like', 'ritchie', 'message', 'straightforward', 'thoug
h', 'honorable', 'real', 'men', 'live', 'commitment', 'extension', 'argues', 'countr
y', 'heart', 'case', 'promise', 'small', 'army', 'interpreter', 'afghanistan', 'aide
d', 'u', 'military', 'two', 'decade', 'exchange', 'service', 'promised', 'travel', 'v
isa', 'family', 'unceremoniously', 'left', 'behind', 'war', 'abruptly', 'ended', 'sto
rytelling', 'purpose', 'covenant', 'ritchie', 'share', 'writing', 'credit', 'ivan',
'atkinson', 'marn', 'davy', 'focus', 'u', 'army', 'sgt', 'john', 'kinley', 'played',
'gyllenhaal', 'lead', 'squad', 'soldier', 'tasked', 'finding', 'destroying', 'taliba
n', 'bomb', 'factory', 'dangerous', 'work', 'learn', 'early', 'leaf', 'kinley', 'smal
l', 'squad', 'need', 'new', 'interpreter', 'enter', 'ahmed', 'dar', 'salim', 'far',
'garden', 'variety', 'interpreter', 'skill', 'ahmed', 'personal', 'motivation', 'help
ing', 'american', 'undeniably', 'compelling', 'story', 'although', 'noted', 'fictiona
l', 'one', 'ritchie', 'pretend', 'hard', 'based', 'actual', 'event', 'although', 'ins
pired', 'plight', 'afghan', 'interpreter', 'left', 'behind', 'u', 'withdrawal', 'stor
y', 'well', 'worth', 'sharing', 'covenant', 'criticized', 'number', 'questioning', 'n
oble', 'heart', 'like', 'gyllenhaal', 'sgt', 'kinley', 'salim', 'ahmed', 'strong', 's
ilent', 'type', 'kind', 'guy', 'exudes', 'confidence', 'capability', 'drink', 'coffe
e', 'black', 'guy', 'grunt', 'warrior', 'kind', 'men', 'whose', 'highest', 'complimen
t', 'one', 'another', 'solemn', 'exchange', 'knowing', 'nod', 'silent', 'recognitio
n', 'towering', 'manliness', 'first', 'must', 'learn', 'trust', 'one', 'another', 'ce
ntral', 'purpose', 'first', 'hour'])
```

2.2) Document Vectorization Using TF-IDF

2.2.1) Definition of Functions for TF-IDF Experiments

For each data wrangling methodologies, let's run some experiments. To set up these experiments, we can obtain the Term Frequency - Inverse Document Frequency (TF-IDF) values associated with each token in each document. Subsequently, we can find the mean TF-IDF score associated with each of the terms that we had previously identified for potential inclusion in the corpus vocabulary. We can also find the terms that have the highest TF-IDF scores in our vocabulary, as well as their associated TF-IDF scores. Last, we can create a heatmap that visualizes the cosine similarity measures between each of the corpus documents using this TF-IDF vectorization method.

In this subsection, we first define the functions that will be used to conduct the experiments described above.

In [38]:
```
def run_tfidf(documents: List[Document],
#               clean_func: Callable[[List[Document]], List[TokenizedDocument]],
              important_prevalent_terms: List[str],
              modified_important_prevalent_terms: List[str],
              experiment_name: str,
              output_tfidf_vectors: bool=False,
              output_vocabulary: bool=True):
#     cleaned_documents = clean_func(documents)
    cleaned_document_text = [' '.join(x.tokens) for x in documents]

    vectorizer = TfidfVectorizer(use_idf=True,
                                 ngram_range=(1, 1),
                                 norm=None)

    transformed_documents = vectorizer.fit_transform(cleaned_document_text)
    transformed_documents_as_array = transformed_documents.toarray()

    output_dir = f'output/{experiment_name}_Results'
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    if output_tfidf_vectors:
        for counter, doc in enumerate(transformed_documents_as_array):
            tf_idf_tuples = list(zip(vectorizer.get_feature_names_out(), doc))
            one_doc_as_df = pd.DataFrame.from_records(tf_idf_tuples, columns=['term',
                                      .sort_values(by='score', ascending=False)\
                                      .reset_index(drop=True)

            one_doc_as_df.to_csv(f'{output_dir}/{corpus_df["Submission File Name"][cou

    if output_vocabulary:
        with open(f'{output_dir}/vocabulary.txt', 'w') as vocab:
            words = sorted(vectorizer.get_feature_names_out())
            print('\n'.join(words), file=vocab)

    # Create document-term dataframe
    doc_term_matrix = transformed_documents.todense()
    doc_term_df = pd.DataFrame(doc_term_matrix,
                               columns=vectorizer.get_feature_names_out(),
                               index=corpus_df.Doc_ID)
    add_flags(doc_term_df, sisters_doc_ids, comedy_doc_ids)

    # Print the top 10 mean TF-IDF values
```

```
#    with pd.option_context("display.max_rows",100):
#        top10_tfidf = pd.DataFrame(doc_term_df.mean().sort_values(ascending=False).he
#        top10_tfidf.rename(columns={0: 'Mean TF-IDF'}, inplace=True)
#        display(top10_tfidf)

    top10_tfidf = pd.DataFrame(doc_term_df.mean().sort_values(ascending=False).head(10
    top10_tfidf.rename(columns={0: 'Mean TF-IDF'}, inplace=True)
    display(top10_tfidf)


    # Define the top 96 terms in terms of TF-IDF for each data wrangling method for us
    # the Word2Vec experiments

    top96_tfidf = pd.DataFrame(doc_term_df.mean().sort_values(ascending=False).head(96
    top96_tfidf.rename(columns={0: 'Mean TF-IDF'}, inplace=True)

    if documents == tokenized_documents_method_one:
        global top_96_tfidf_terms_method_one_list
        top_96_tfidf_terms_method_one_list = top96_tfidf.index.tolist()


    elif documents == tokenized_documents_method_two:
        global top_96_tfidf_terms_method_two_list
        top_96_tfidf_terms_method_two_list = top96_tfidf.index.tolist()


    else:
        global top_96_tfidf_terms_method_three_list
        top_96_tfidf_terms_method_three_list = top96_tfidf.index.tolist()


    # Collect result into a dataframe
    tfidf_results = pd.DataFrame(index=important_prevalent_terms)

    impt_tfidf_results = doc_term_df[[x for x in modified_important_prevalent_terms if
    tfidf_results['Important Movies'] = impt_tfidf_results
    impt_tfidf_results_df = impt_tfidf_results.to_frame()
    impt_tfidf_results_df.rename(columns={0: 'Mean TF-IDF'}, inplace=True)
    display(impt_tfidf_results_df)




    # Print the vocabulary size and heatmap
    plt.hist(doc_term_df.mean(), 100, range=(0, 8))

    print(f'Vocabulary size: {doc_term_df.shape[1]}')

    descriptors = corpus_df['Descriptor']

#    similarities = cosine_similarity(doc_term_df.loc[comedy_doc_ids], doc_term_df.loc
    similarities = cosine_similarity(doc_term_df, doc_term_df)
    fig, ax = plt.subplots(figsize=(30, 30))
    labels = [descriptors_by_doc_ids[x.doc_id] for x in documents]
    sns.heatmap(ax=ax, data=similarities, xticklabels=labels, yticklabels=labels)
    #plt.savefig(f'figures/{experiment_name}_heatmap_documents.png')
    plt.show()
```

2.2.2) TF-IDF Experiment Output - Data Wrangling Method 1

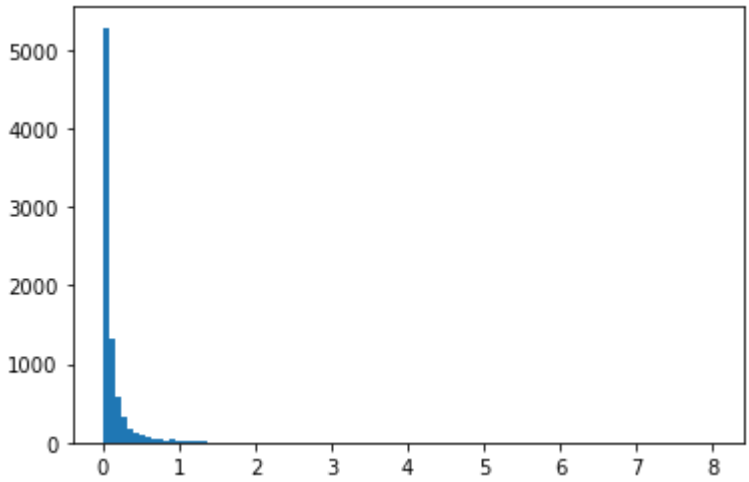
Let's run our TF-IDF experiments using the documents produced from processing the corpus data using data wrangling method one.

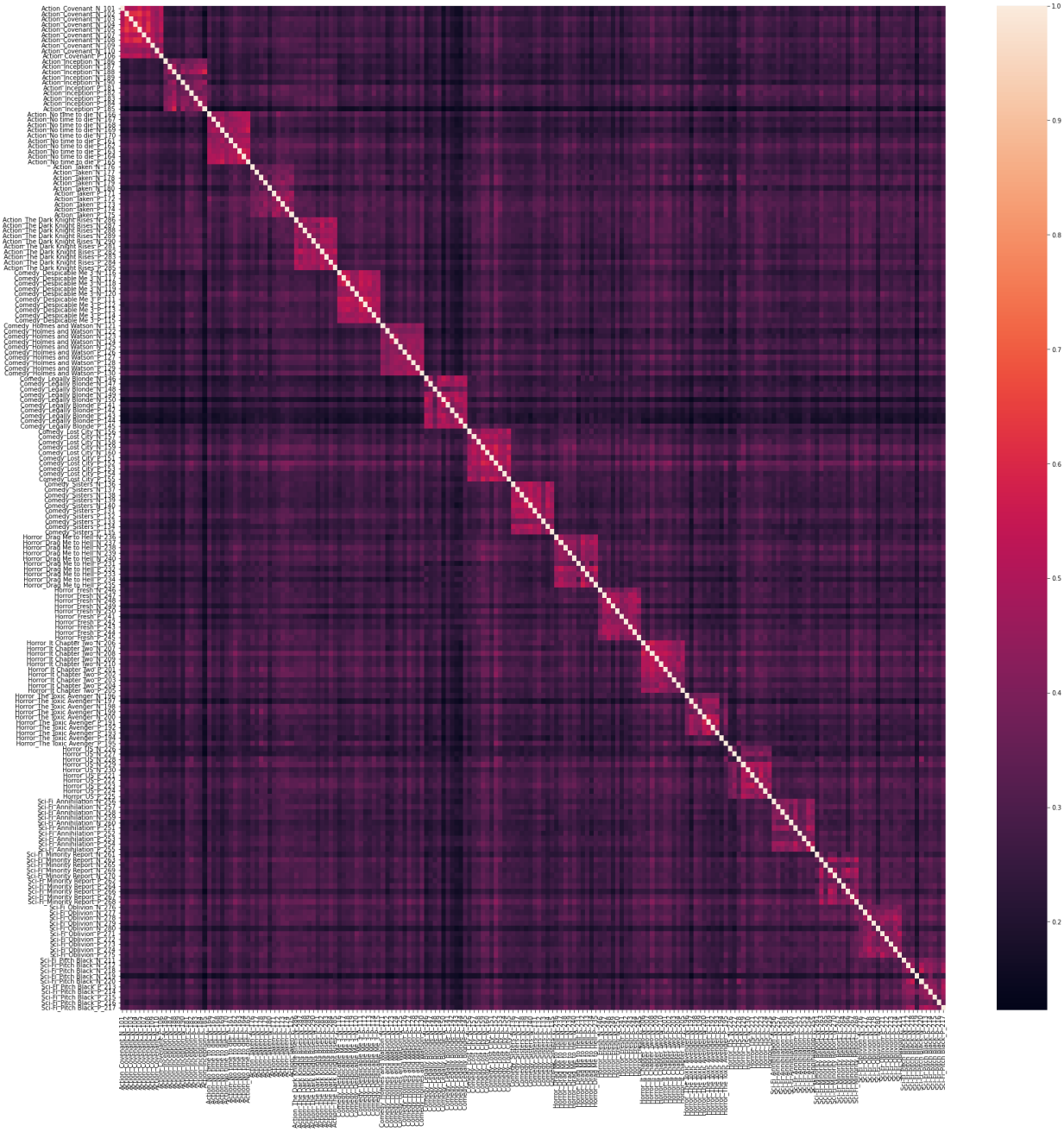
In [39]:
```
run_tfidf(tokenized_documents_method_one, important_prevalent_terms, stemmed_important
          'TF_IDF_Experiment_Method_1')
```

	Mean TF-IDF
the	27.83
and	14.54
of	13.23
to	12.47
in	8.55
is	7.71
it	7.67
that	6.31
as	4.51
with	4.45

	Mean TF-IDF
parti	0.73
comedi	1.12
funni	0.65
joke	0.85

```
Vocabulary size: 8277
```





2.2.3) TF-IDF Experiment Output - Data Wrangling Method 2

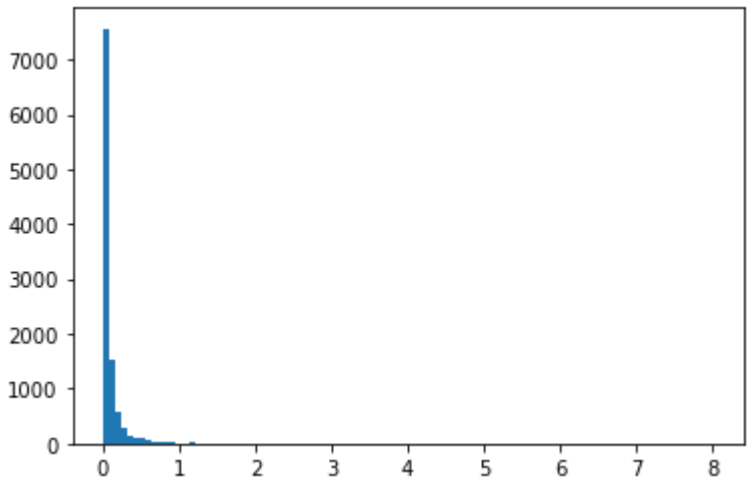
Let's run our TF-IDF experiments using the documents produced from processing the corpus data using data wrangling method two.

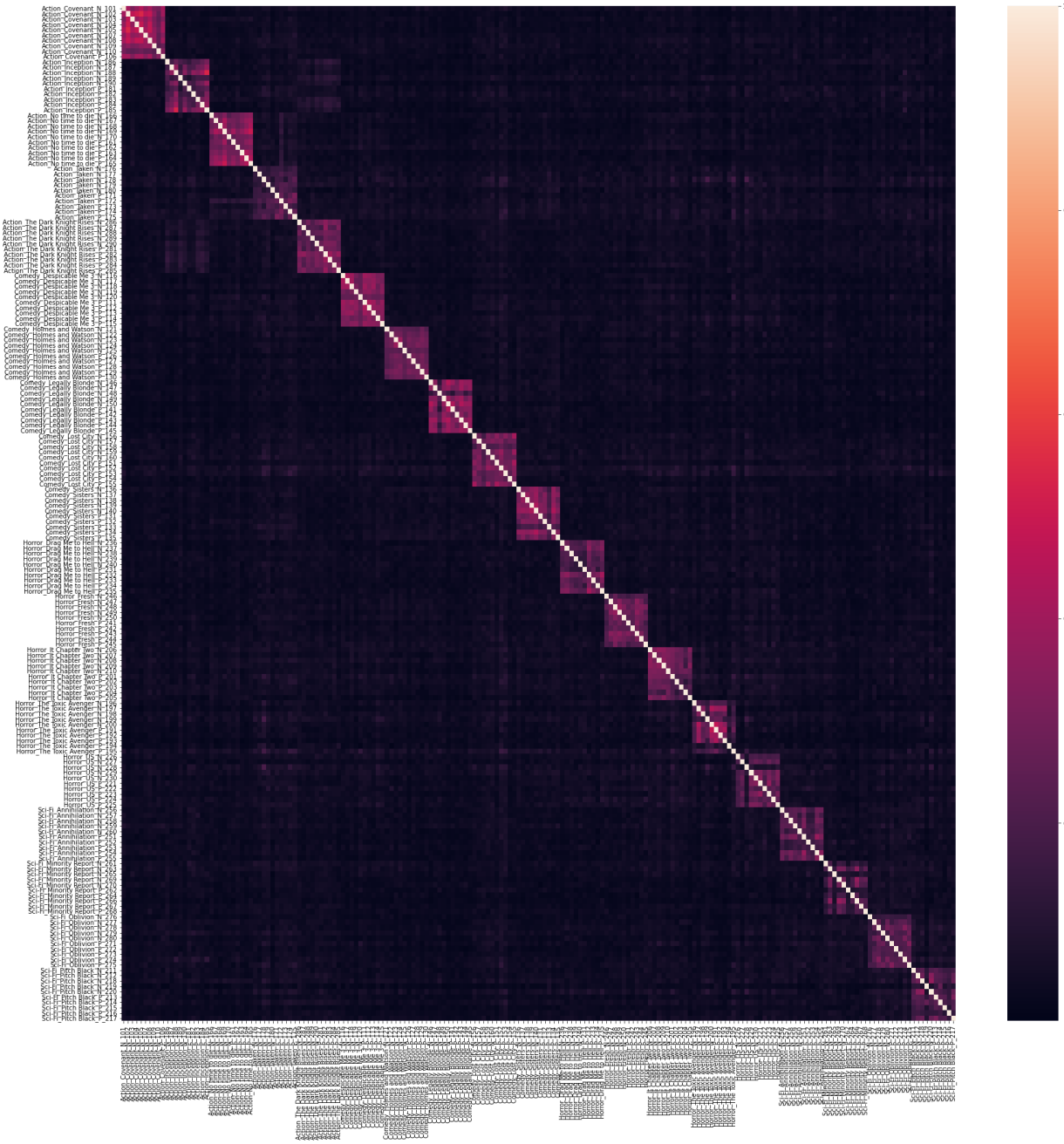
In [40]:
```
#run_tfidf(documents, clean_method, important_prevalent_terms, 'TFIDF_exp')

run_tfidf(tokenized_documents_method_two, important_prevalent_terms, lemmatized_import
          'TF_IDF_Experiment_Method_2')
```

	Mean TF-IDF
film	3.56
movie	2.98
ha	2.39
wa	2.29
one	2.18
like	2.14
bond	2.01
get	1.78
time	1.68
two	1.58

	Mean TF-IDF
party	0.68
comedy	1.12
funny	0.65
joke	0.83

```
Vocabulary size: 10558
```





2.2.4) TF-IDF Experiment Output - Data Wrangling Method 3

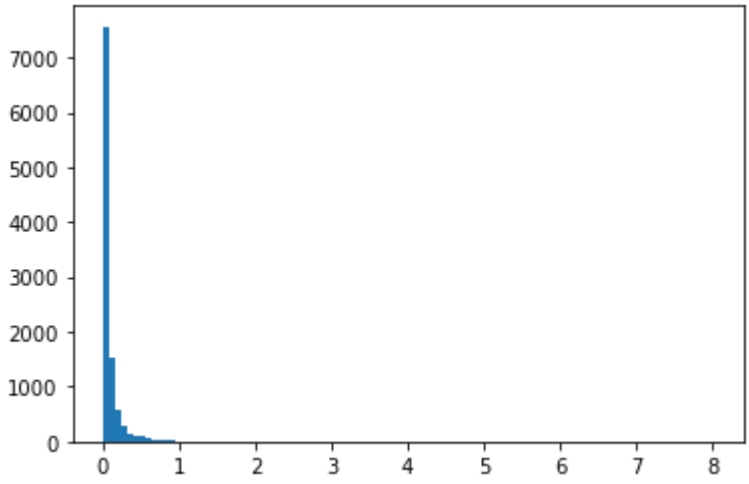
Let's run our TF-IDF experiments using the documents produced from processing the corpus data using data wrangling method three.

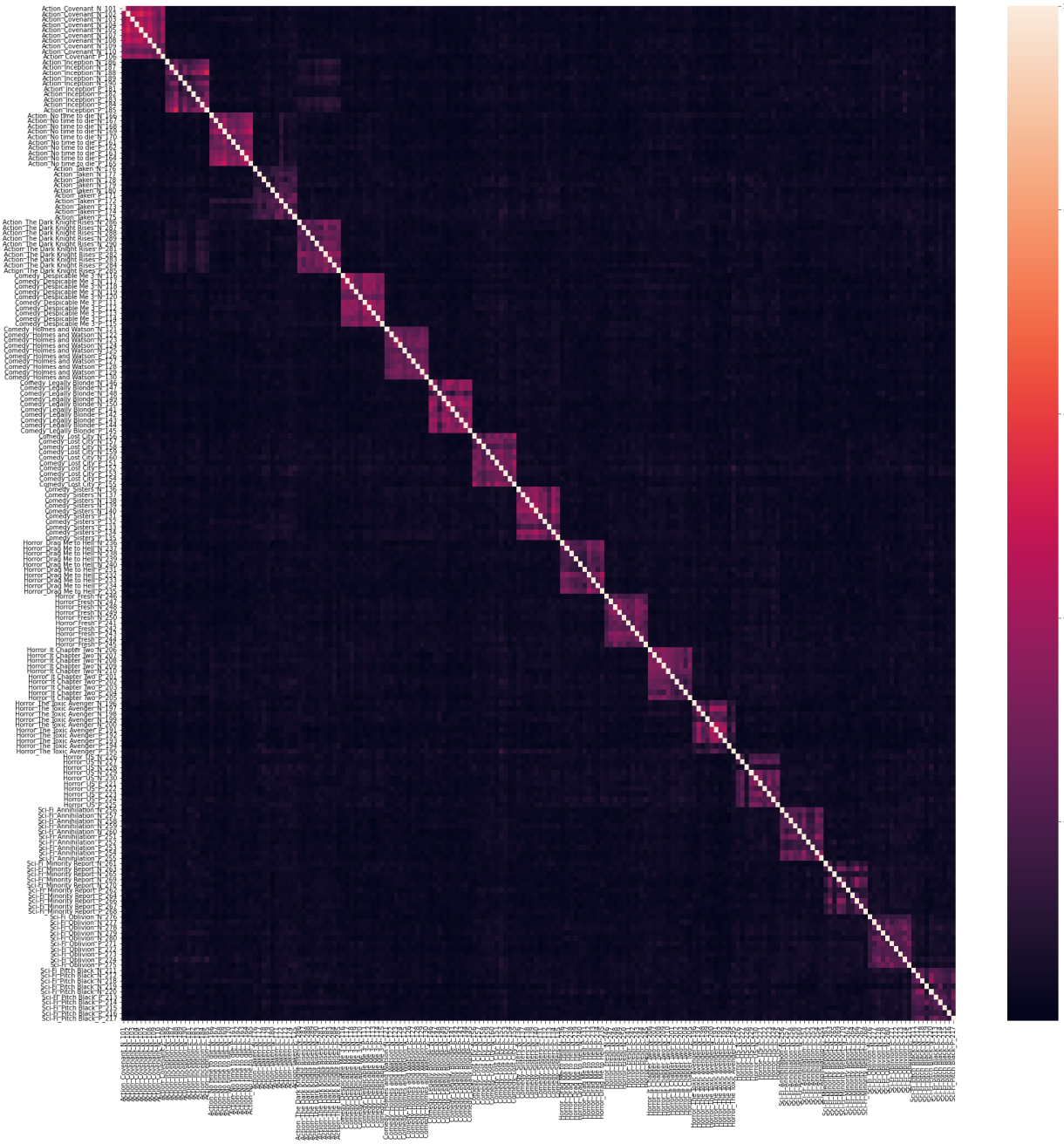
In [41]:
```
run_tfidf(tokenized_documents_method_three, important_prevalent_terms, lemmatized_impo
          'TF_IDF_Experiment_Method_3')
```

	Mean TF-IDF
one	2.18
like	2.14
bond	2.01
time	1.68
two	1.58
character	1.58
dream	1.53
horror	1.44
even	1.39
first	1.36

	Mean TF-IDF
party	0.68
comedy	1.12
funny	0.65
joke	0.83

```
Vocabulary size: 10537
```





2.3) Document Vectorization Using Doc2Vec

2.3.1) Definition of Utility Functions for Doc2Vec Experiments

Let's define functions that can be leveraged to conduct experiments that transform the normalized and tokenized movie reviews into document embedding vectors via Doc2Vec. Using these functions, we can plot heat maps of the cosine similarities of the document embedding vectors, as well as T-SNE plots of the document embedding vectors. These resources can help us identify which NLP data wrangling methods seem to be most effective at producing Doc2Vec embedding vectors that are best for grouping reviews of similar movies together.

In [42]:
```
def plot_similarity_matrix(data: pd.DataFrame, experiment_name: str, figsize=(25, 25)):
    similarities = cosine_similarity(data, data)
    fig, ax = plt.subplots(figsize=figsize)
```

```
    sns.heatmap(ax=ax, data=similarities, xticklabels=data.index, yticklabels=data.inc
    #plt.savefig(f'figures/{experiment_name}_heatmap.png')
    plt.close()

def plot_similarity_clustermap(data: pd.DataFrame, experiment_name: str, figsize=(25,
    similarities = cosine_similarity(data, data)
    cm = sns.clustermap(similarities, metric='cosine', xticklabels=data.index, yticklа
    cm.ax_row_dendrogram.set_visible(False)
    cm.ax_col_dendrogram.set_visible(False)
    plt.legend(loc='upper left')
    #plt.savefig(f'figures/{experiment_name}_clustermap.png')
    plt.show()
    plt.close()

def plot_tsne(data: pd.DataFrame, perplexity: int, experiment_name: str, figsize=(40,
    """
    Creates a TSNE plot of the supplied dataframe
    """
    tsne_model = TSNE(perplexity=perplexity, n_components=2, learning_rate='auto', ini
    new_values = tsne_model.fit_transform(data)

    x = []
    y = []
    for value in new_values:
        x.append(value[0])
        y.append(value[1])

    plt.figure(figsize=figsize)
    labels = list(data.index)
    for i in range(len(x)):
        new_value = new_values[i]
        x = new_value[0]
        y = new_value[1]

        plt.scatter(x, y)
        plt.annotate(labels[i],
                     xy=(x, y),
                     xytext=(5, 2),
                     textcoords='offset points',
                     ha='right',
                     va='bottom')
    #plt.savefig(f'figures/{experiment_name}_tsne.png')
    plt.show()
    plt.close()

def run_doc2vec(documents: List[TokenizedDocument], embedding_size: int, descriptors_b
    tagged_documents = [TaggedDocument(document.tokens, [i]) for i, document in enumer
    doc2vec_model = Doc2Vec(tagged_documents, vector_size=embedding_size, window=3, mi

    doc2vec_df = pd.DataFrame()
    for document in documents:
        vector = pd.DataFrame(doc2vec_model.infer_vector(document.tokens)).transpose()
        doc2vec_df = pd.concat([doc2vec_df, vector], axis=0)

    doc2vec_df['Descriptor'] = [descriptors_by_doc_ids[x.doc_id] for x in documents]
    doc2vec_df.set_index(['Descriptor'], inplace=True)
    return doc2vec_df

def run_doc2vec_experiment(documents: List[Document],
#                            clean_func: Callable[[List[Document]], List[TokenizedDocum
```

```
                            embedding_size: int,
                            experiment_name: str):
#     cleaned_documents = clean_func(documents)
#     doc2vec_df = run_doc2vec(cleaned_documents, embedding_size, descriptors_by_doc_id
    doc2vec_df = run_doc2vec(documents, embedding_size, descriptors_by_doc_ids)

    plot_similarity_matrix(doc2vec_df, experiment_name)
    plot_similarity_clustermap(doc2vec_df, experiment_name, figsize=(50, 50))
    plot_tsne(doc2vec_df, 30, experiment_name)
```

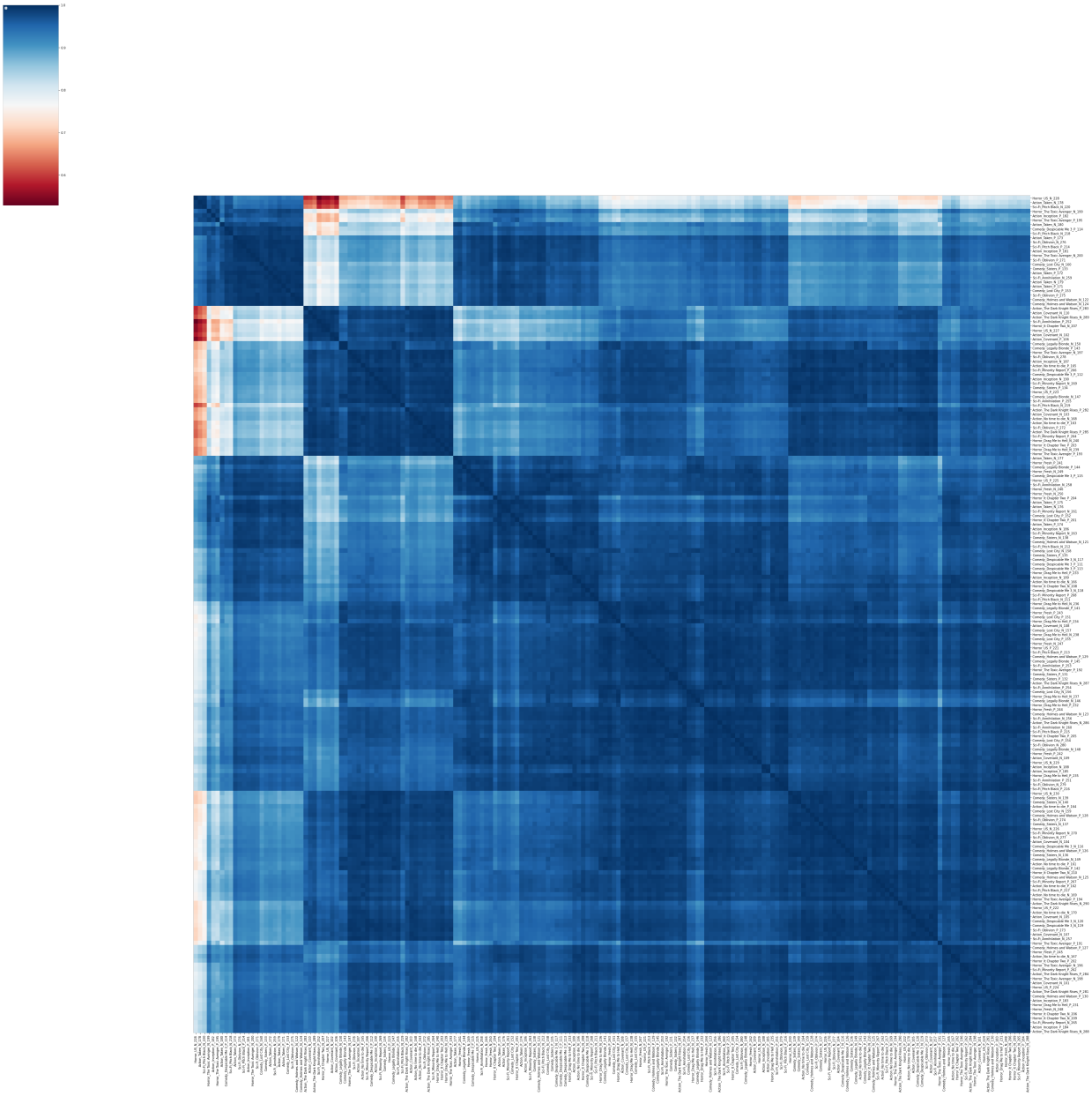
2.3.2) Doc2Vec Experiment Outputs - T-SNE Plots and Heatmaps

Let's run our Doc2Vec experimentation function for each of the nine test cases of interest. These nine experiments will leverage each of our three data wrangling methodologies. For each of the data wrangling methods, we will run three experiments that use 100, 200, and 300 Doc2Vec embedding dimensions, respectively, to represent the corpus documents.

Doc2Vec Experiment 1 - Data Wrangling Method 1 - 100 Embedding Dimensions

In [43]:
```
run_doc2vec_experiment(tokenized_documents_method_one, 100, 'Doc2Vec_Experiment_1')
```

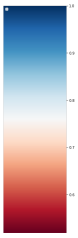
```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

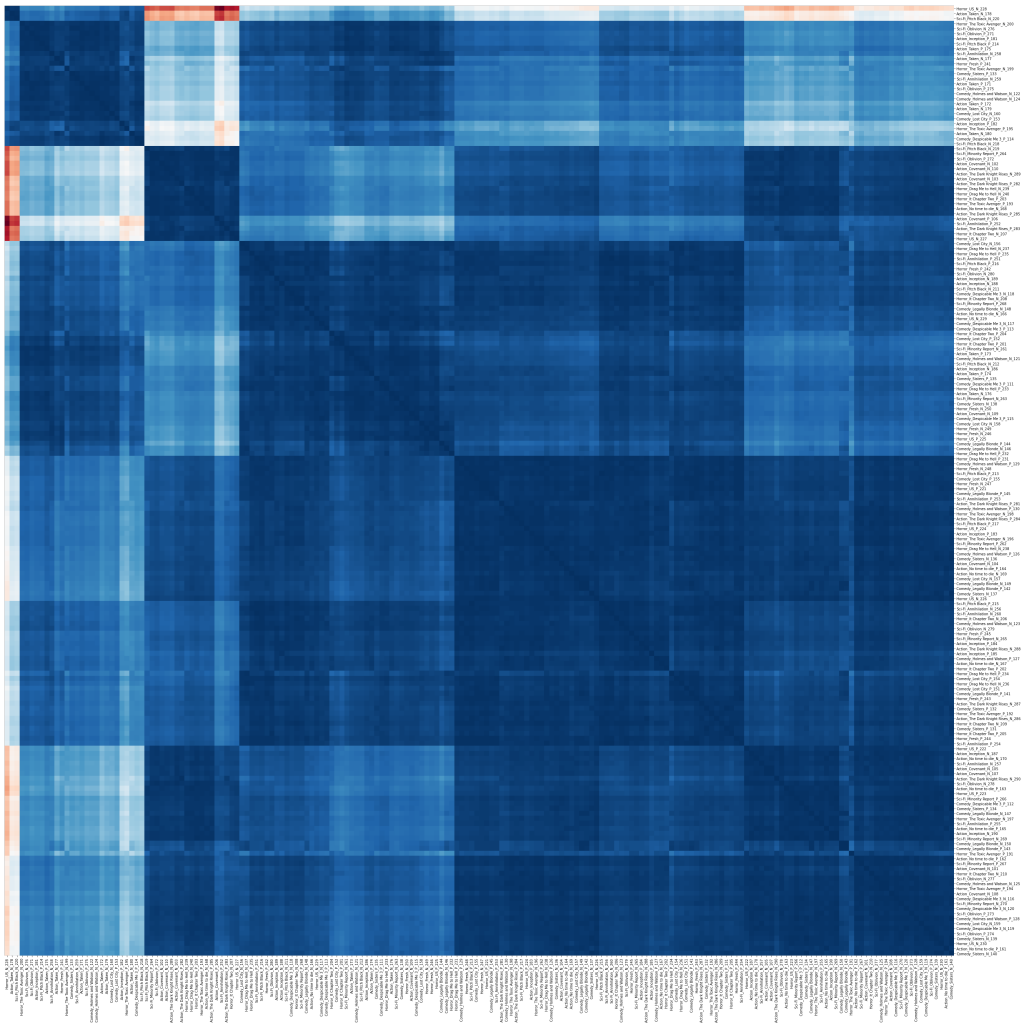


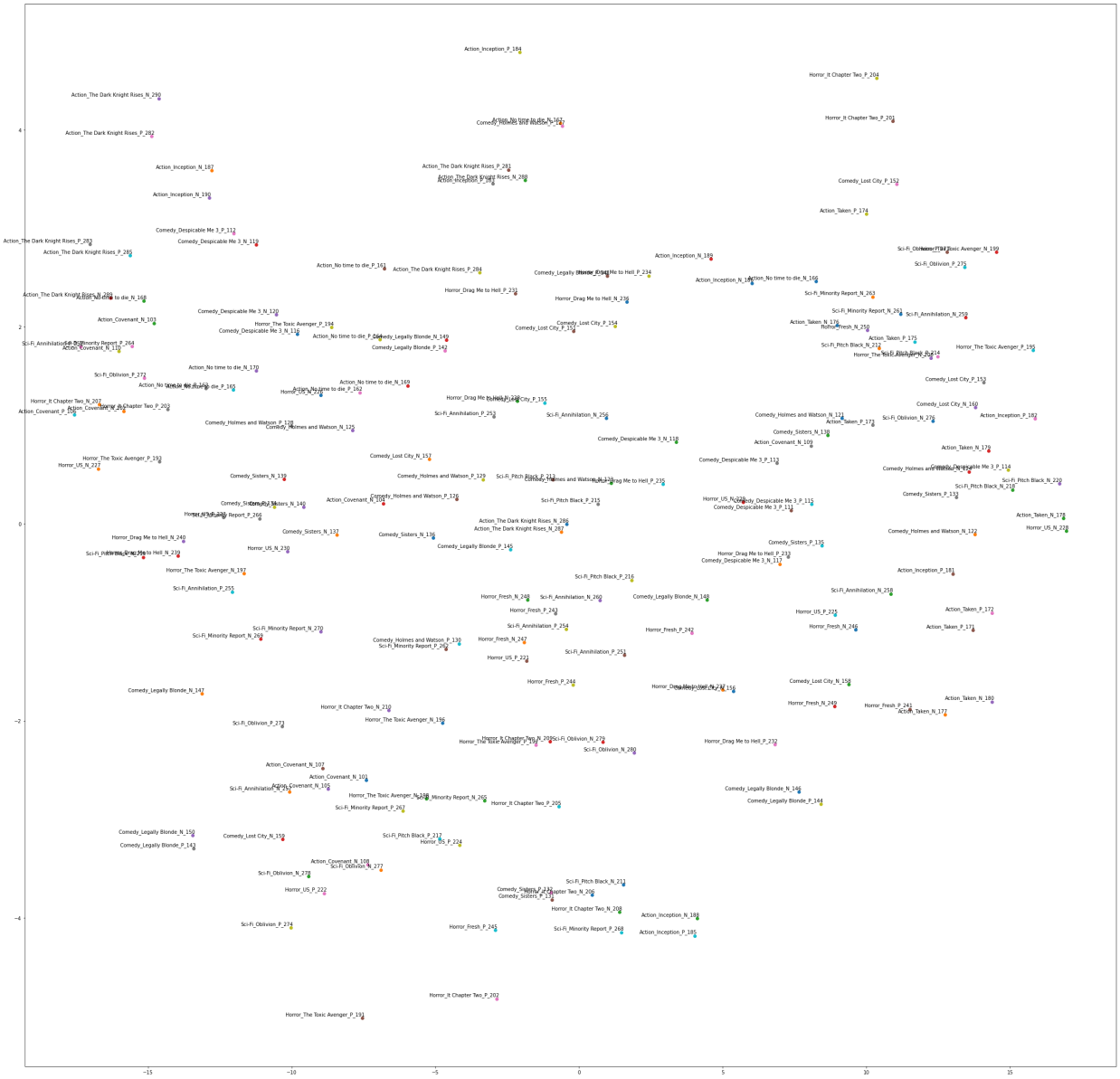
Doc2Vec Experiment 2 - Data Wrangling Method 1 - 200 Embedding Dimensions

In [44]:
```
run_doc2vec_experiment(tokenized_documents_method_one, 200, 'Doc2Vec_Experiment_2')
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```



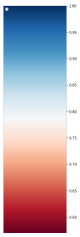


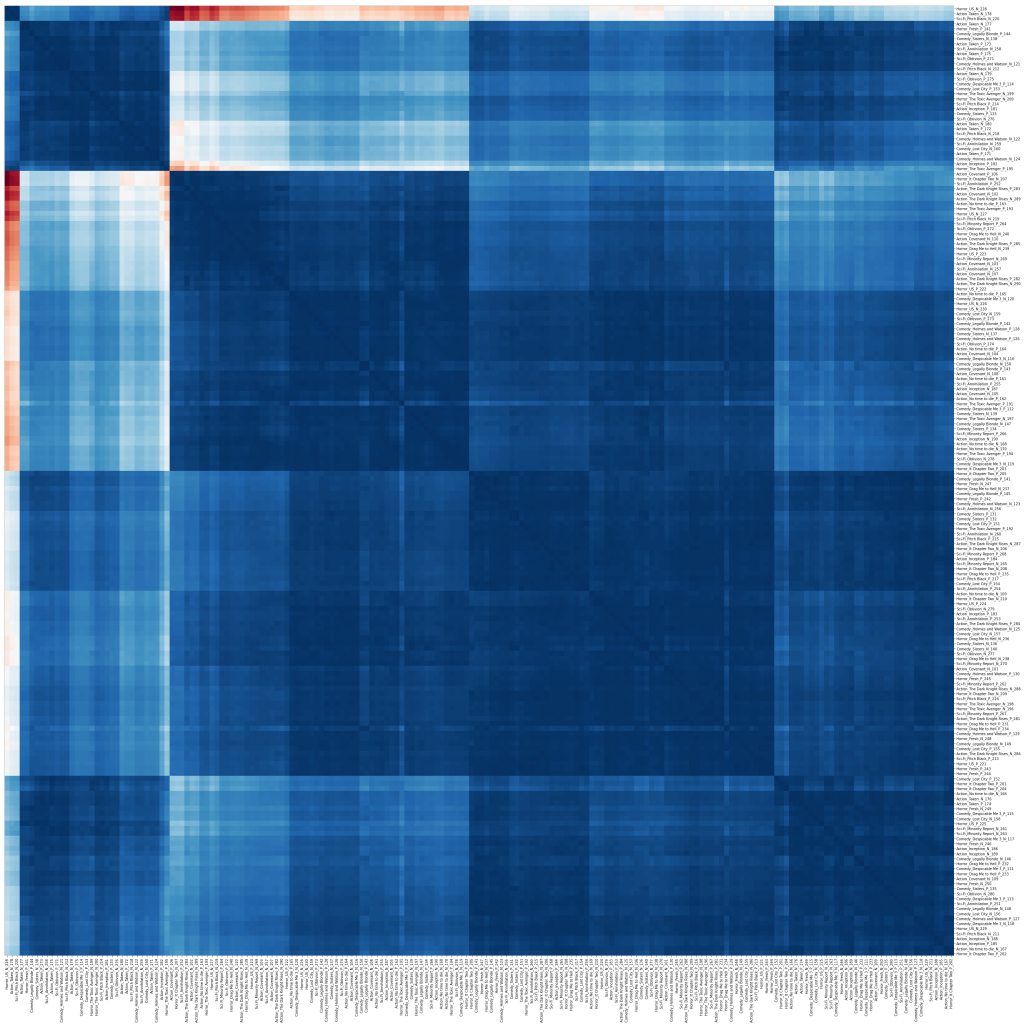


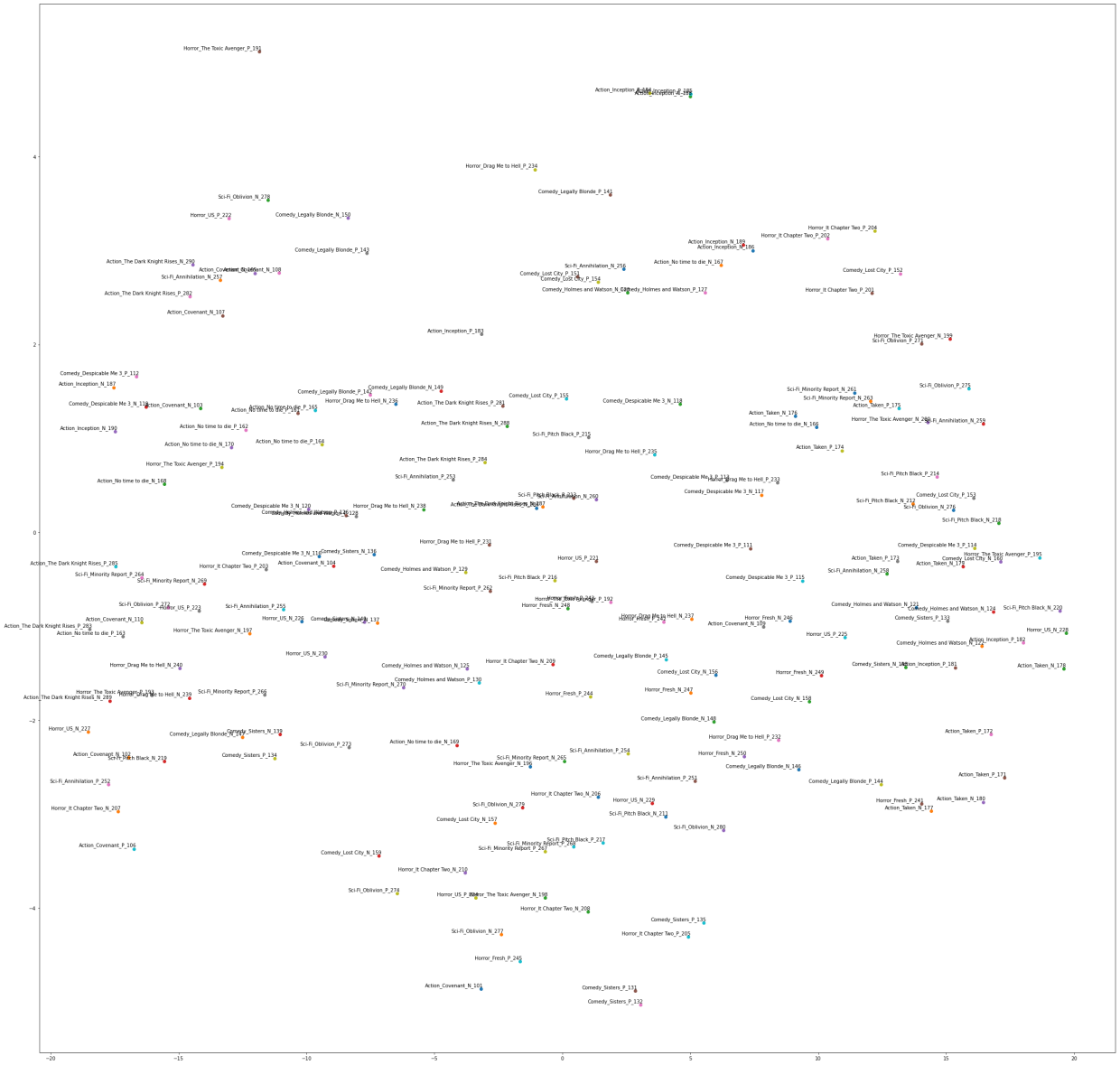
Doc2Vec Experiment 3 - Data Wrangling Method 1 - 300 Embedding Dimensions

In [45]:
```
run_doc2vec_experiment(tokenized_documents_method_one, 300, 'Doc2Vec_Experiment_3')
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```



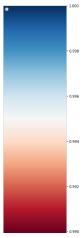


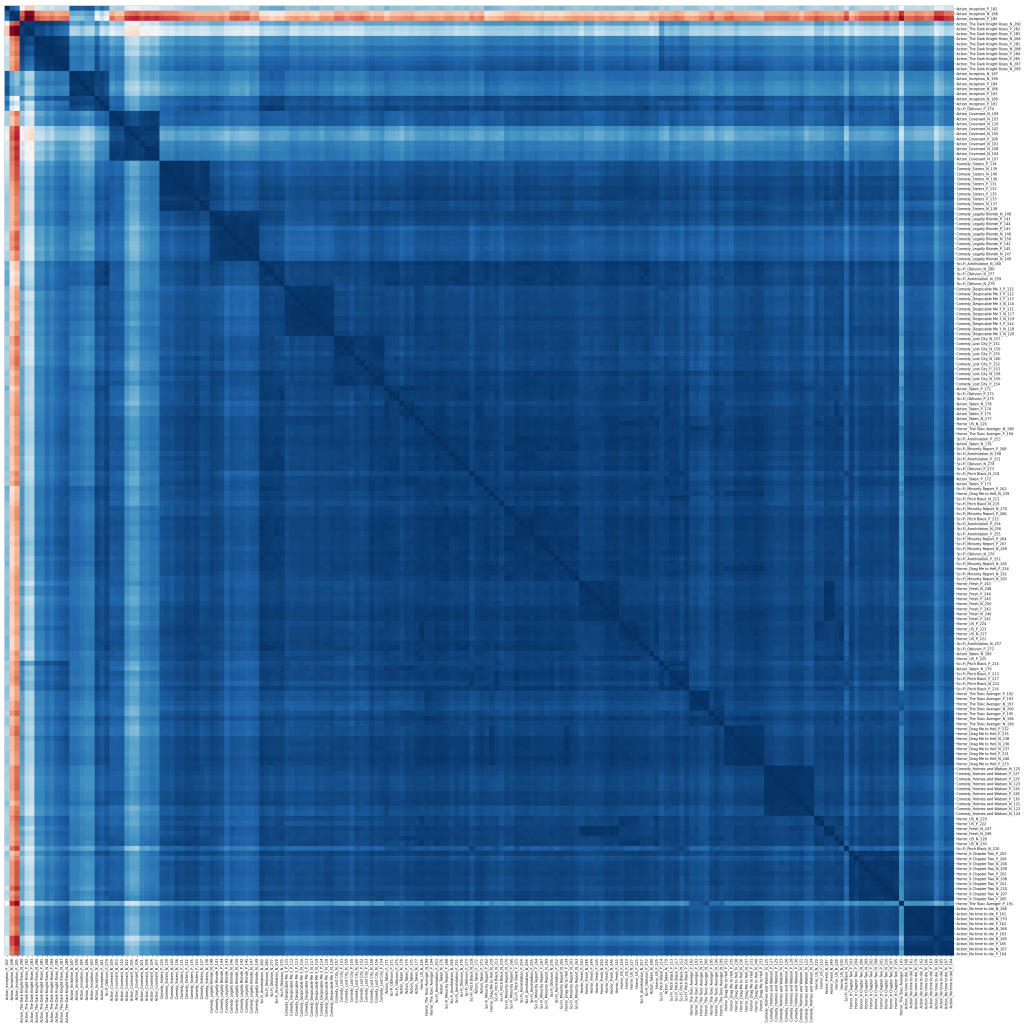


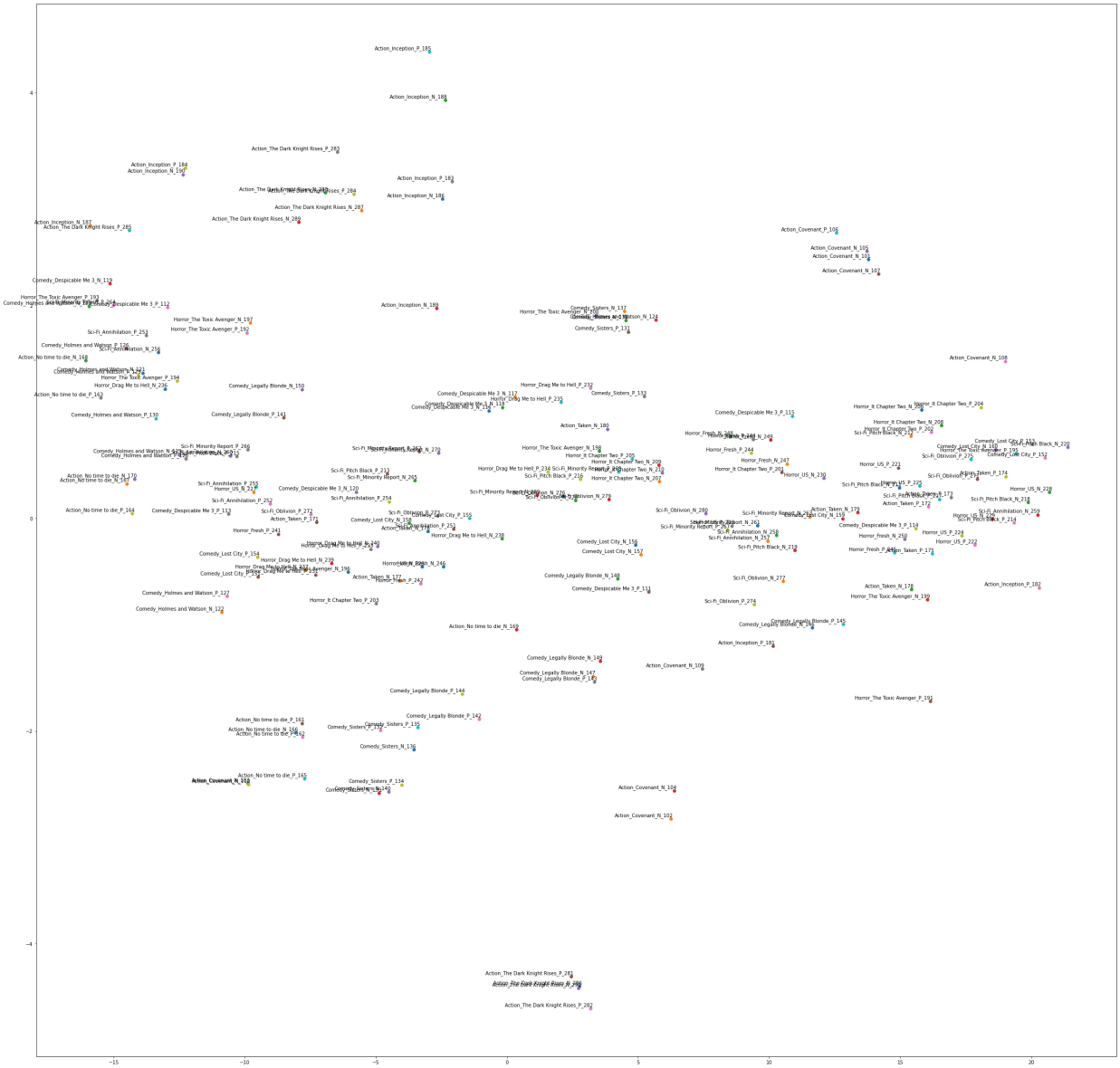
Doc2Vec Experiment 4 - Data Wrangling Method 2 - 100 Embedding Dimensions

In [46]:
```
run_doc2vec_experiment(tokenized_documents_method_two, 100, 'Doc2Vec_Experiment_4')
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```



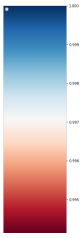


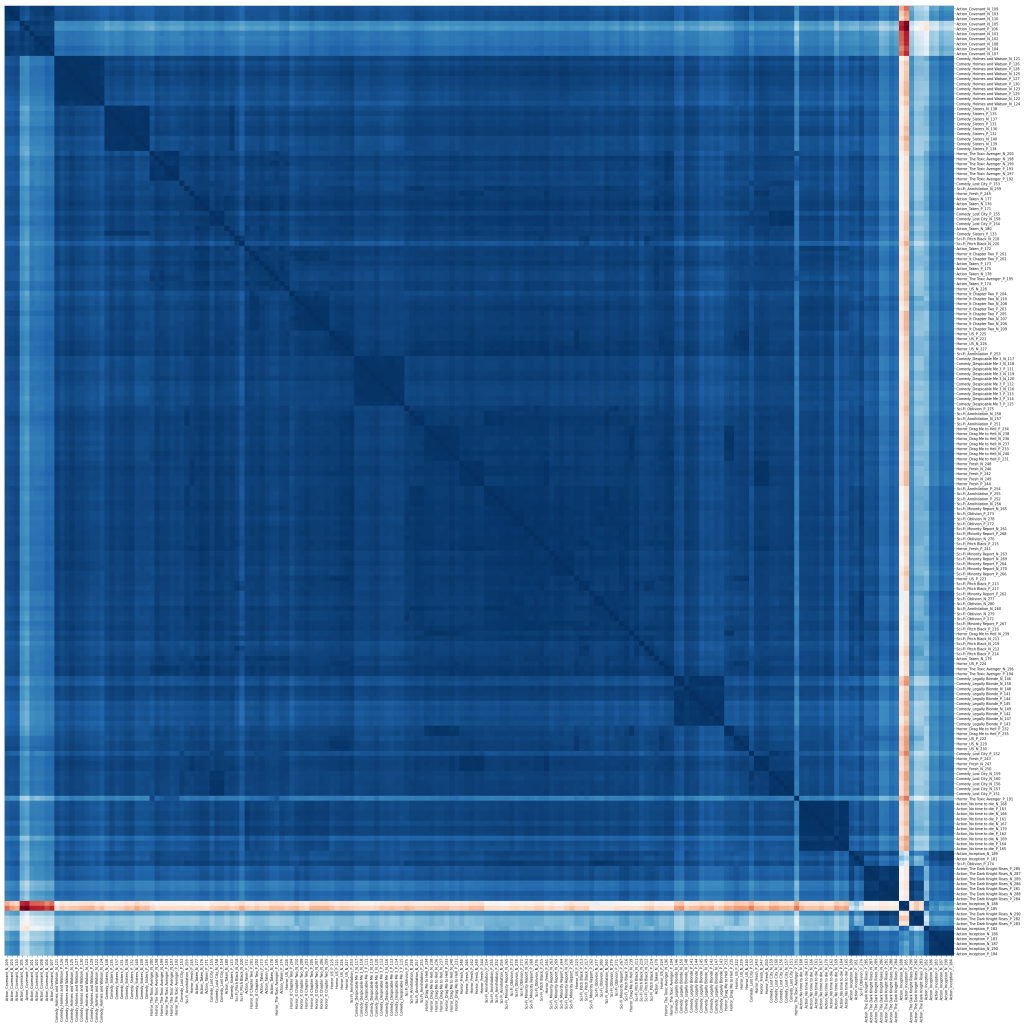


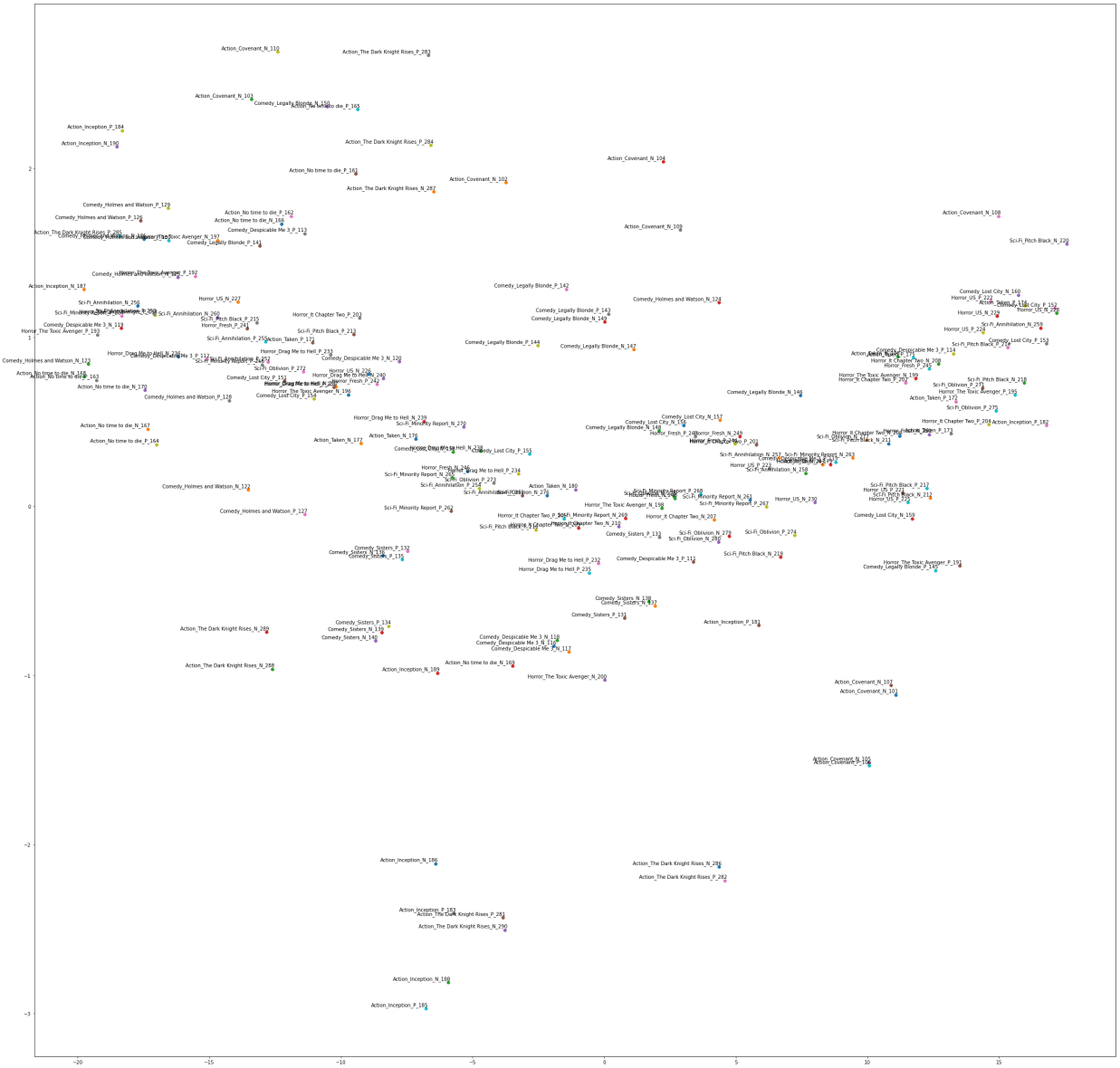
Doc2Vec Experiment 5 - Data Wrangling Method 2 - 200 Embedding Dimensions

In [47]:
```
run_doc2vec_experiment(tokenized_documents_method_two, 200, 'Doc2Vec_Experiment_5')
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```



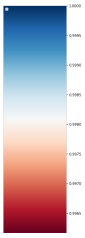


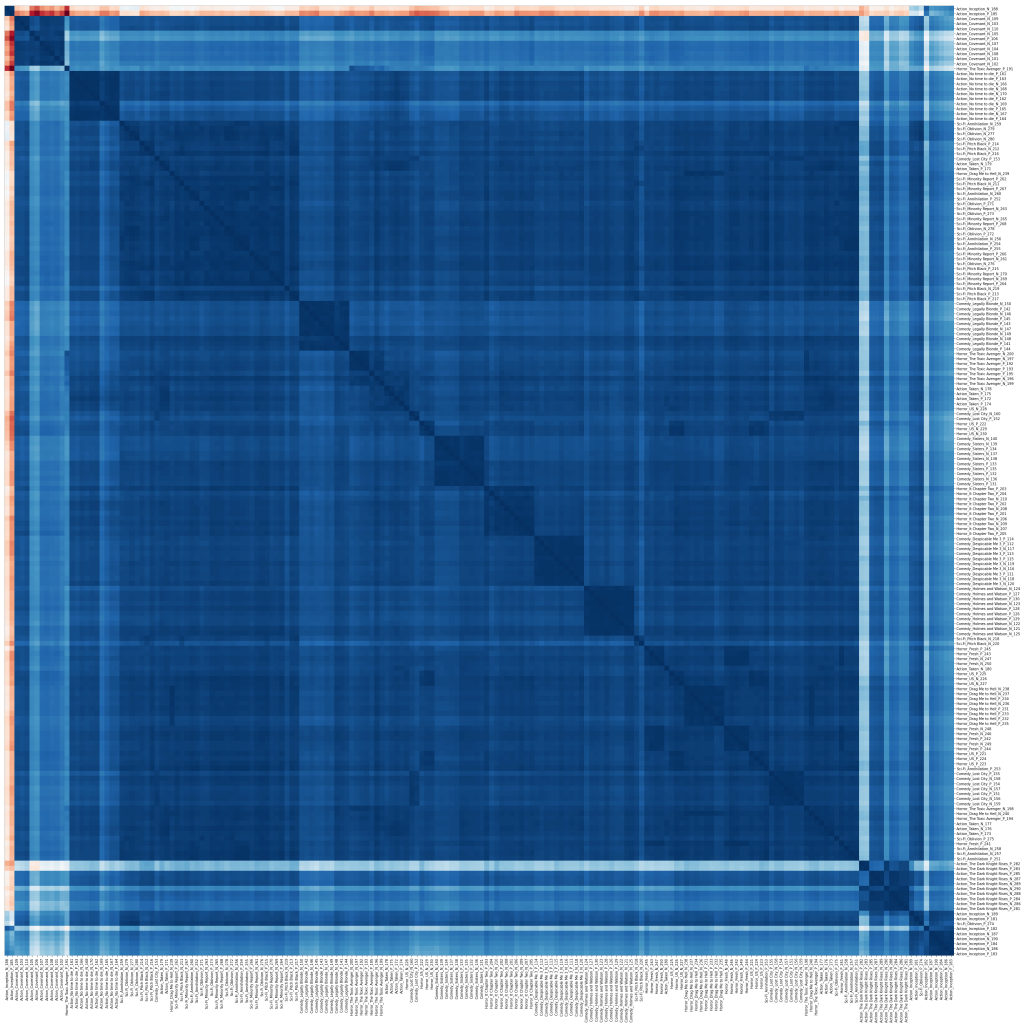


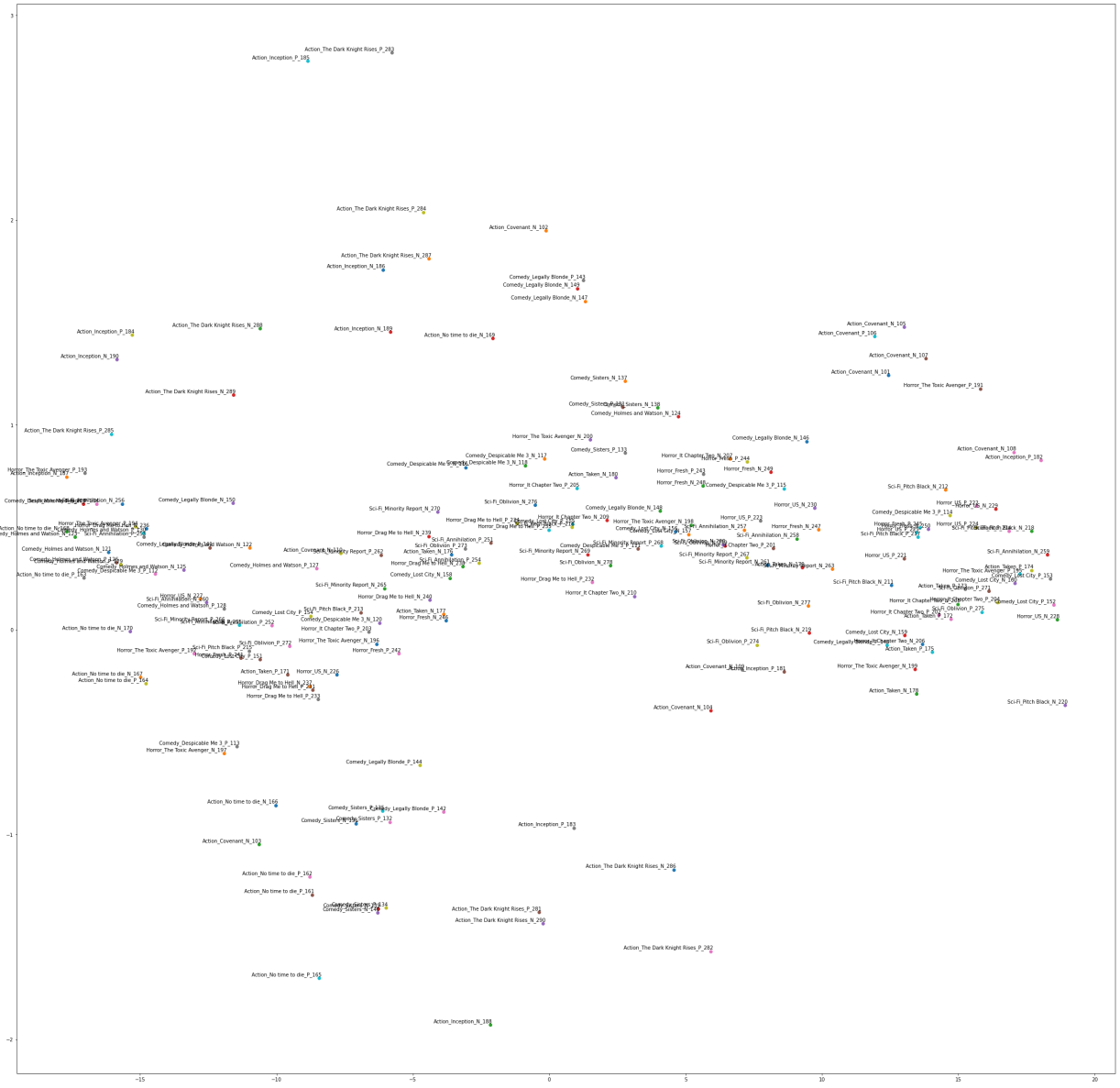
Doc2Vec Experiment 6 - Data Wrangling Method 2 - 300 Embedding Dimensions

In [48]:
```
run_doc2vec_experiment(tokenized_documents_method_two, 300, 'Doc2Vec_Experiment_6')
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```



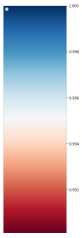


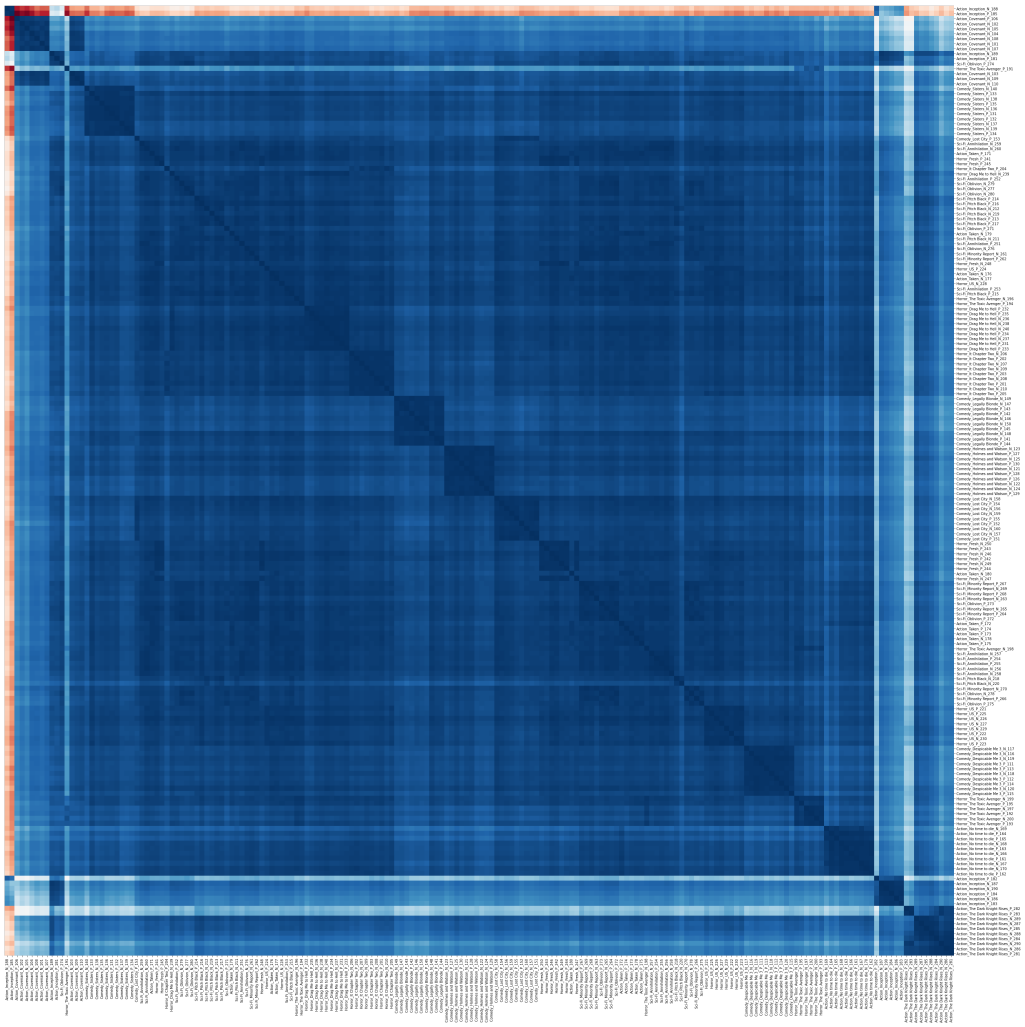


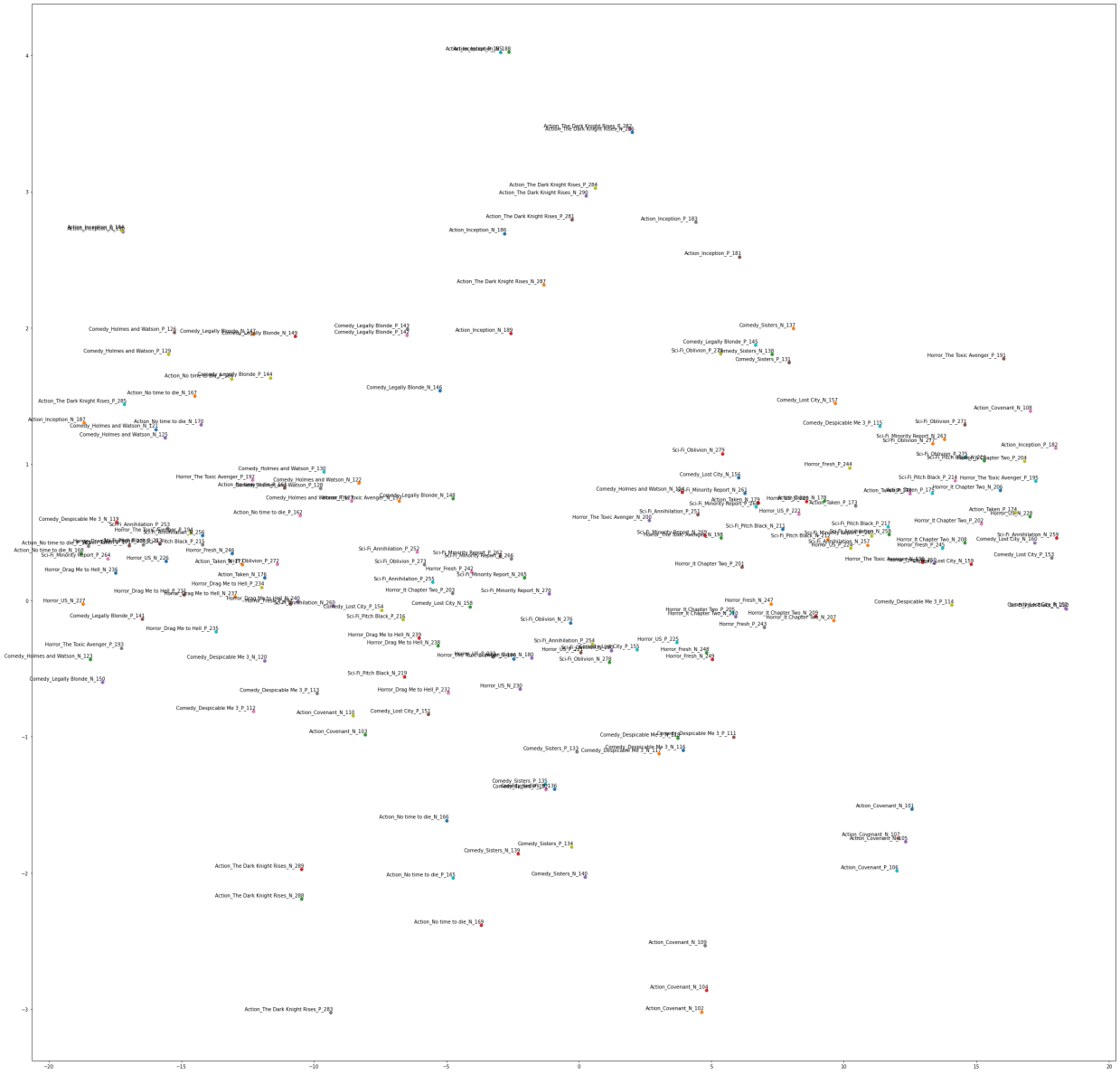
Doc2Vec Experiment 7 - Data Wrangling Method 3 - 100 Embedding Dimensions

In [49]:
```
run_doc2vec_experiment(tokenized_documents_method_three, 100, 'Doc2Vec_Experiment_7')
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```



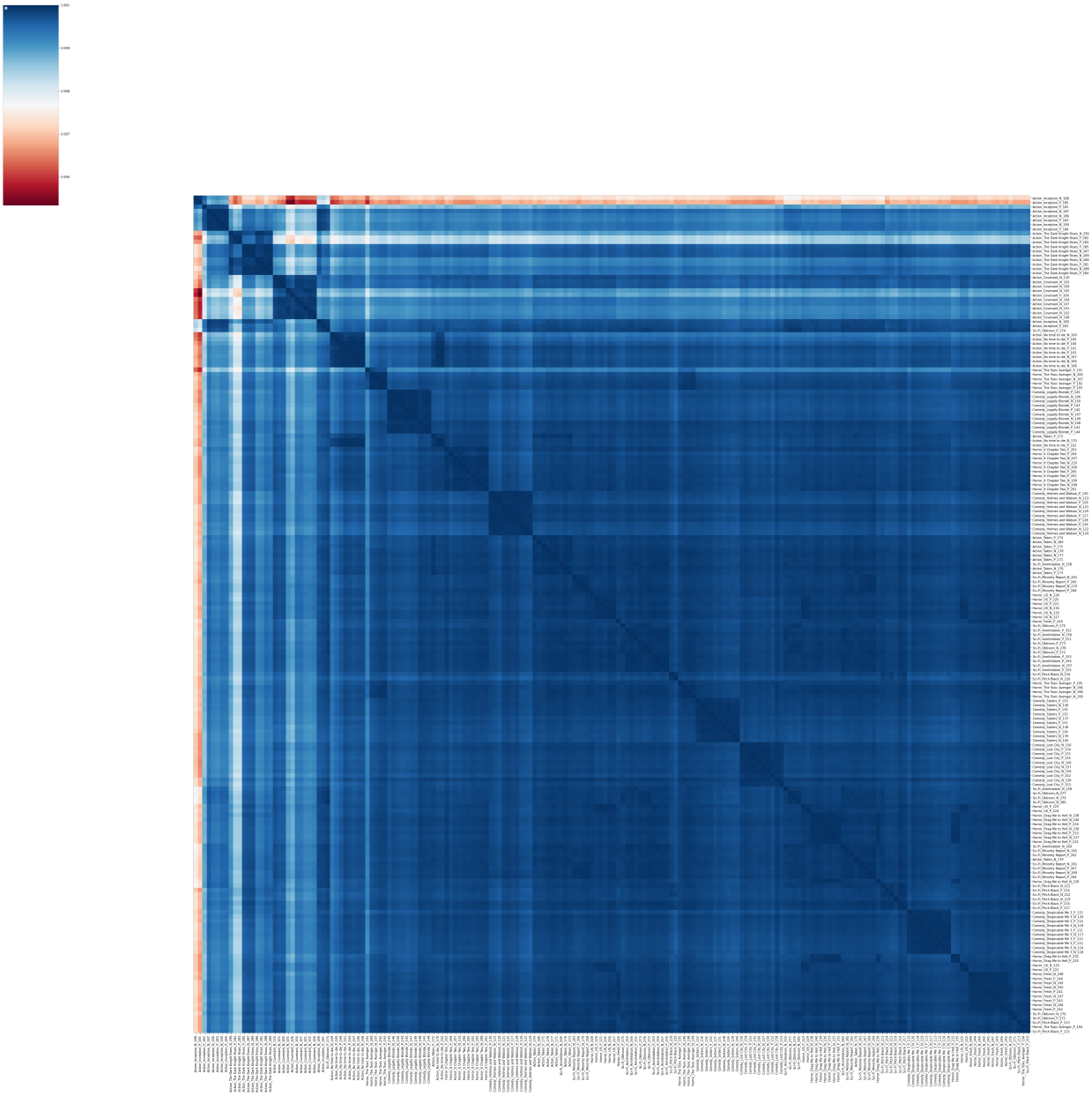


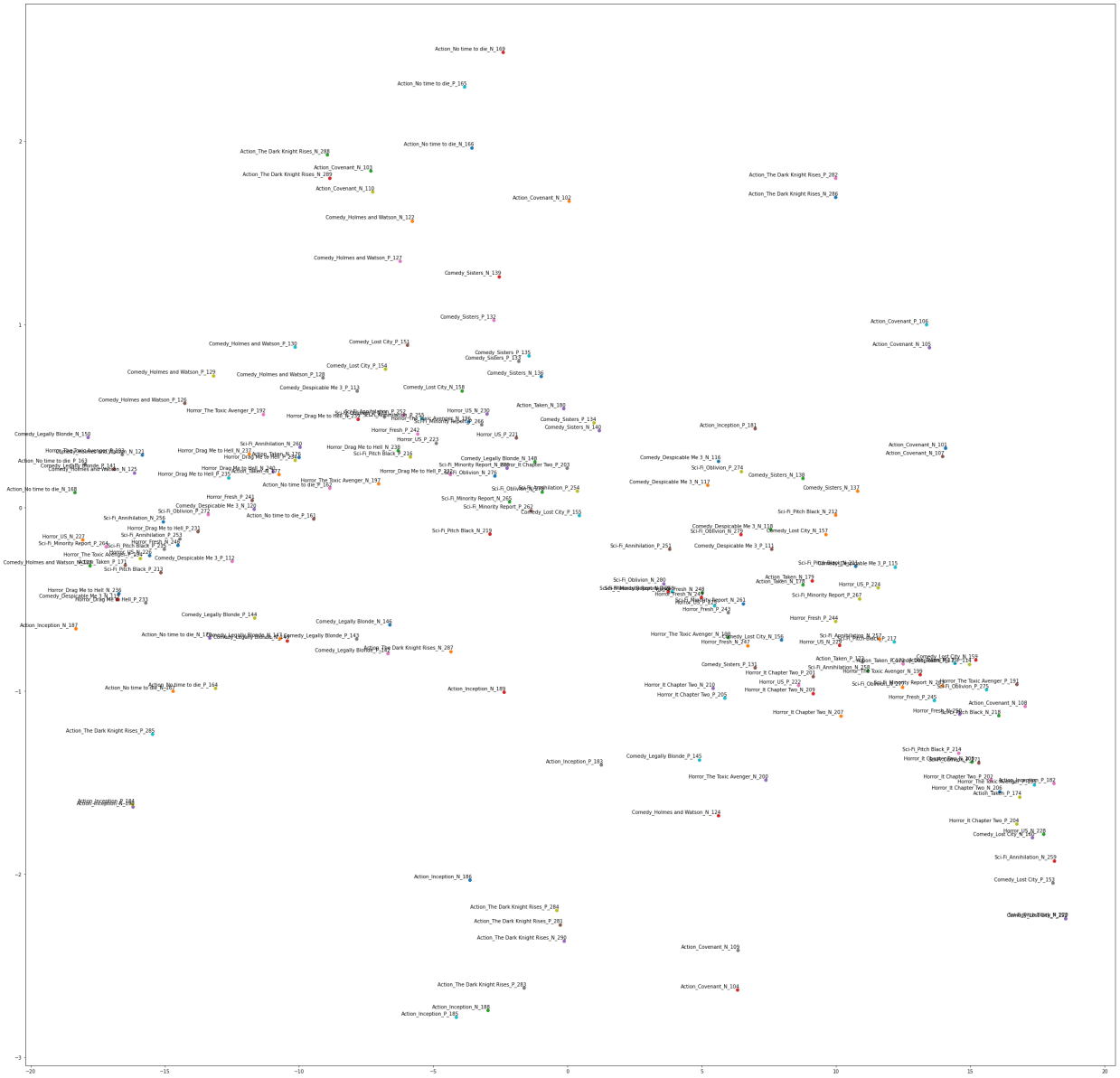


Doc2Vec Experiment 8 - Data Wrangling Method 3 - 200 Embedding Dimensions

In [50]:
```
run_doc2vec_experiment(tokenized_documents_method_three, 200, 'Doc2Vec_Experiment_8')
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

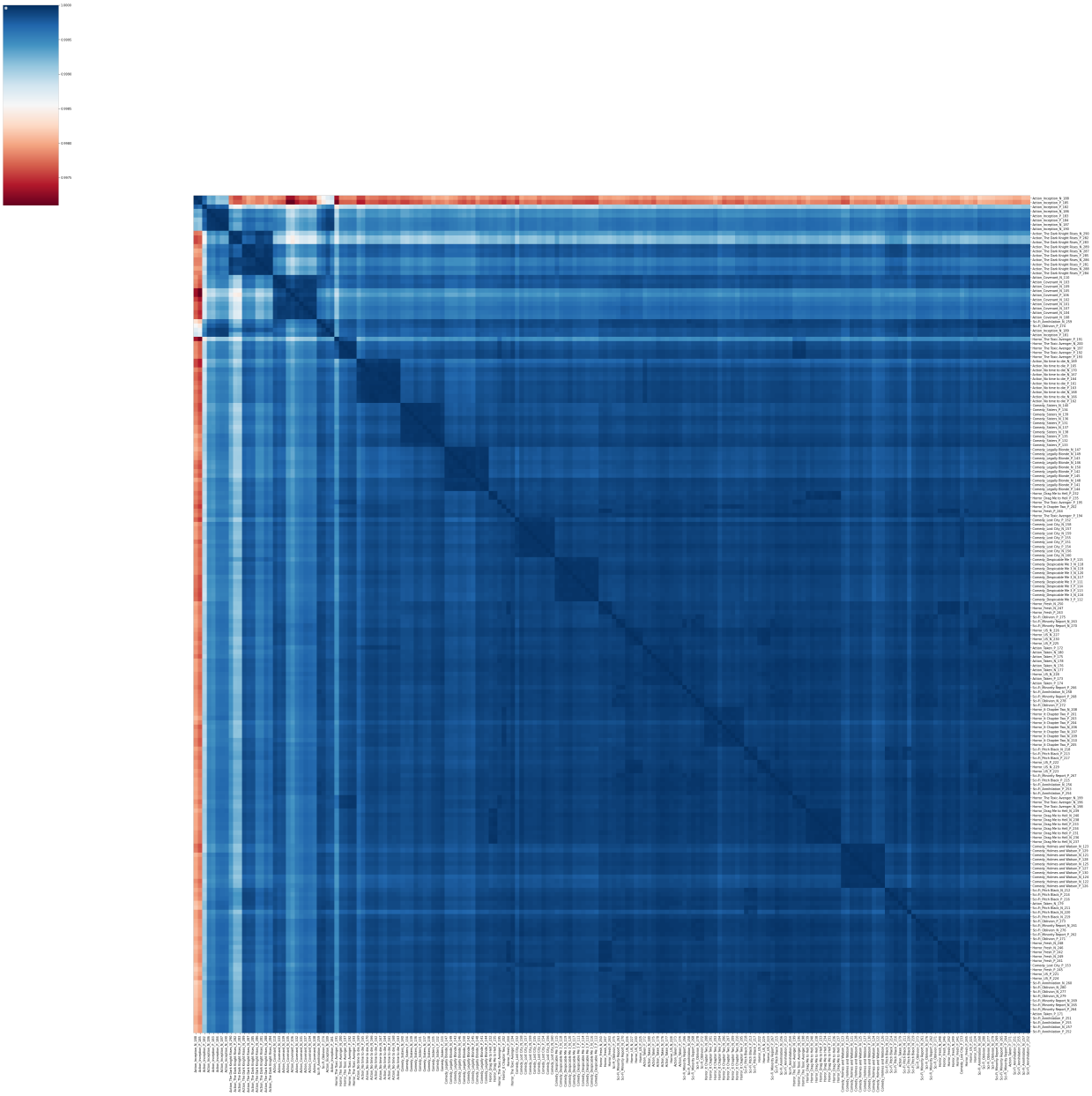


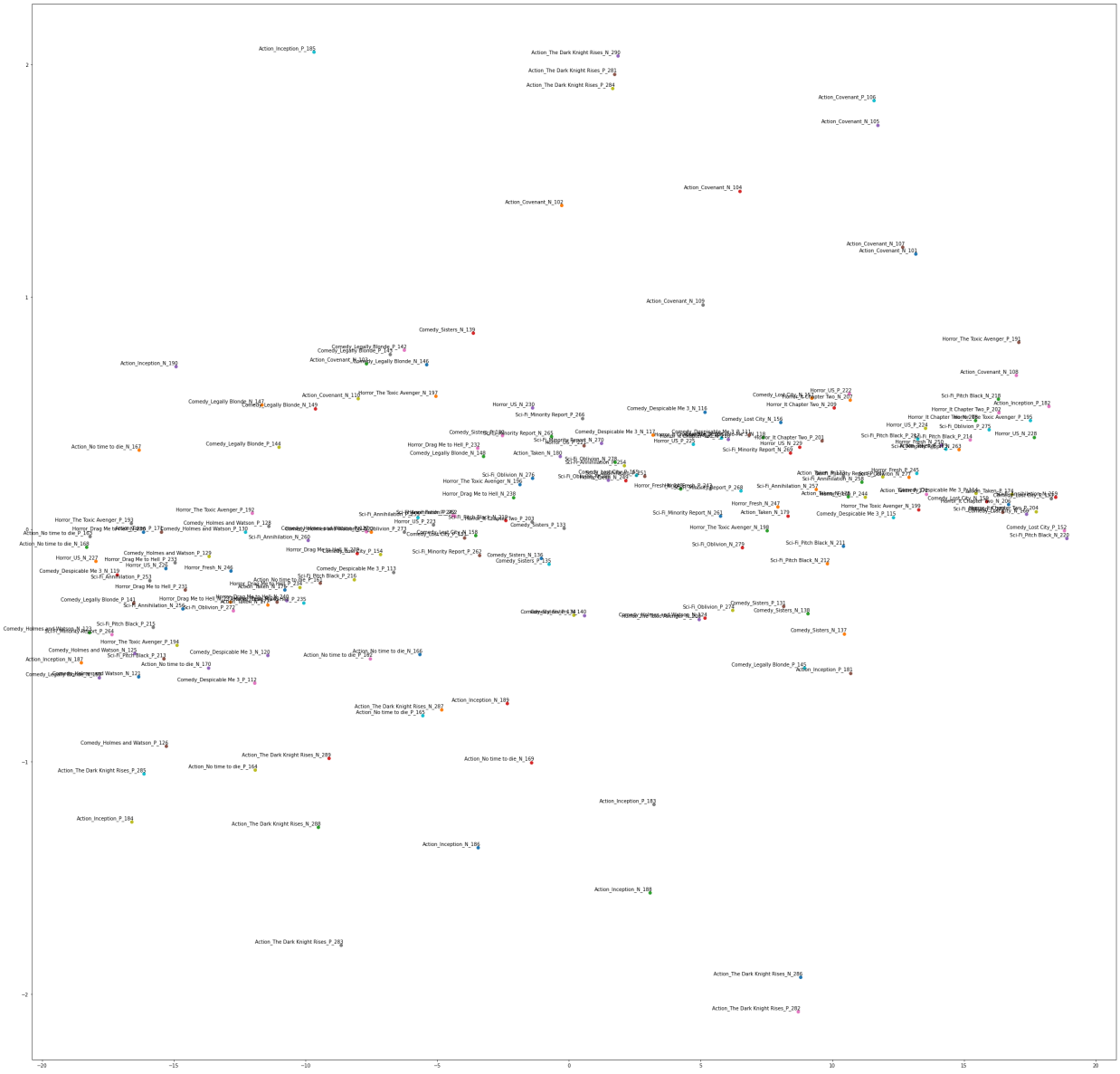


Doc2Vec Experiment 9 - Data Wrangling Method 3 - 300 Embedding Dimensions

In [51]:
```
run_doc2vec_experiment(tokenized_documents_method_three, 300, 'Doc2Vec_Experiment_9')
```

```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```





2.4) Document Vectorization Using Word2Vec

2.4.1) Definition of Utility Functions For Word2Vec Experiments

We can now examine how closely related certain words from our corpus are related to one another semantically. For this analysis, we can begin by selecting 100 terms from our corpus and representing them via their respective Word2Vec embedding vectors. We can then create heatmaps that visualize the pairwise cosine similarity measures between these embedding vectors and T-SNE plots that depict how the embedding vectors appear to cluster together.

We can conduct these experiments 9 times: three times for each data wrangling method leveraging 100, 200, and 300 Word2Vec embedding dimensions in each experiment, respectively.

In [52]:
```
def get_word2vec_vectors(documents: List[TokenizedDocument], embedding_size: int) -> p
    tokens = [x.tokens for x in documents]
```

```
    # https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4

    # gensim 3.8
    #word2vec_model = Word2Vec(tokens, size=embedding_size, window=3, min_count=1, wor

    # gensim 4.3.1
    word2vec_model = Word2Vec(sentences=tokens, vector_size=embedding_size, window=3,

    # gensim 3.8
    #vectors = {}
    #for i in word2vec_model.wv.vocab:
    #    temp_vec = word2vec_model.wv[i]
    #    vectors[i] = temp_vec

    # gensim 4.3.1
    vectors = {}
    for i in word2vec_model.wv.index_to_key:
        temp_vec = word2vec_model.wv[i]
        vectors[i] = temp_vec

    # Changes made:
    #   1. The size parameter in Word2Vec has been renamed to vector_size.
    #   2. Instead of iterating through word2vec_model.wv.vocab, you should now use
    #      word2vec_model.wv.index_to_key to get the list of words in the vocabulary.


    result = pd.DataFrame(vectors).transpose()
    result = result.sort_index()
    return result


def run_word2vec_experiment(documents: List[Document],
#                            clean_func: Callable[[List[Document]], List[TokenizedDocu
                            embedding_size: int,
                            chosen_tokens: List[str],
                            experiment_name: str):
#    cleaned_documents = clean_func(documents)
    cleaned_documents = documents

    word2vec_df = get_word2vec_vectors(cleaned_documents, embedding_size)
    filtered_word2vec_df = word2vec_df.loc[chosen_tokens].copy()

    plot_tsne(filtered_word2vec_df, 30, experiment_name)
    plot_similarity_matrix(filtered_word2vec_df, experiment_name)
    plot_similarity_clustermap(filtered_word2vec_df, experiment_name)



#extra_terms_method_one = [
#    'bond',
#    'craig',
#    'star',
#    'casino',
#    'action'
#]


#def get_all_tokens(documents: List[TokenizedDocument]) -> List[str]:
#    tokens = {y for x in documents for y in x.tokens}
#    return sorted(list(tokens))
```

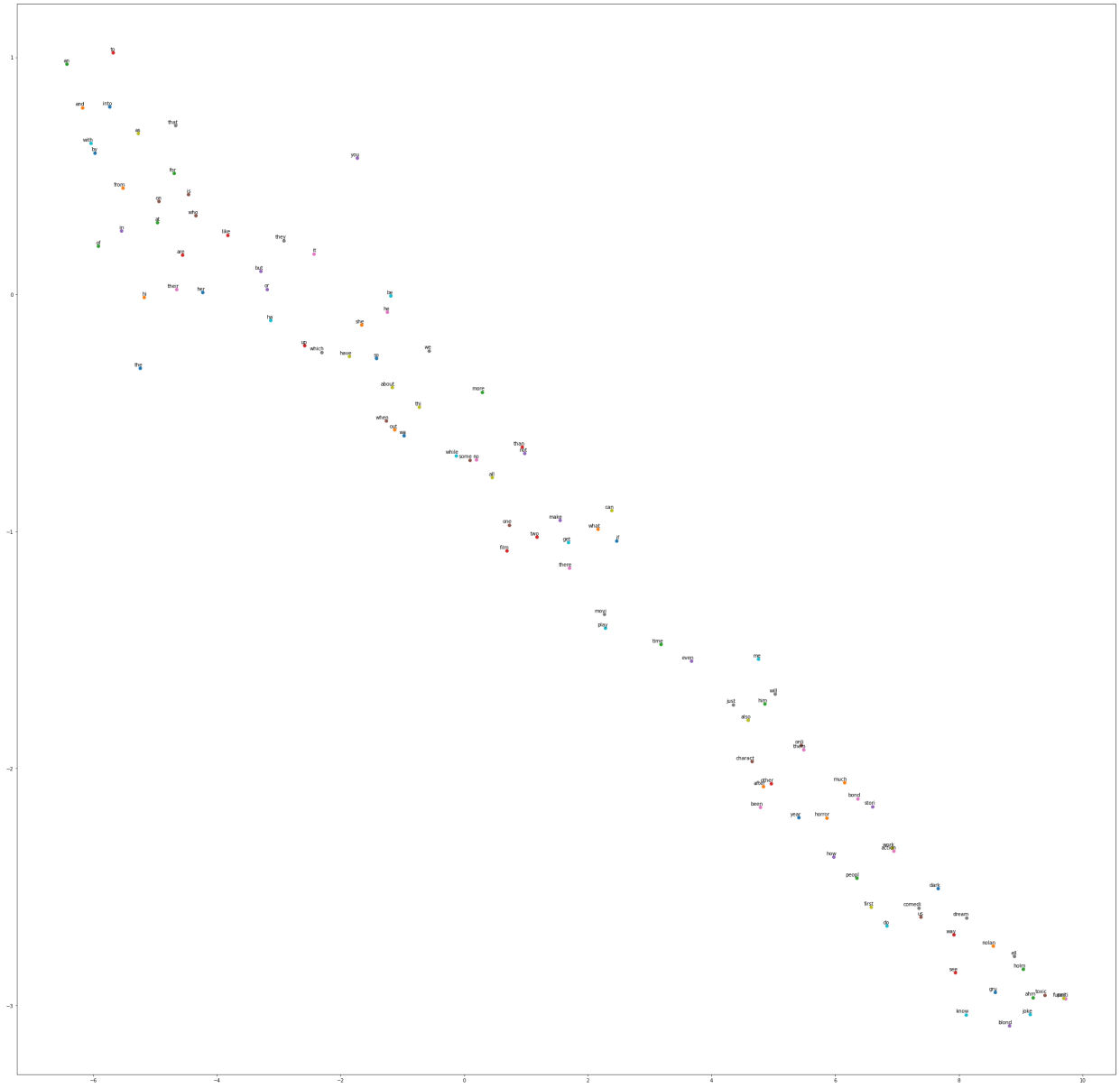
```
#all_tokens = get_all_tokens(tokenized_documents)



# Get our terms to examine in experiements 4-12
#all_tokens_method_one = get_all_tokens(tokenized_documents_method_one)
#all_tokens_method_two = get_all_tokens(tokenized_documents_method_two)
#all_tokens_method_three = get_all_tokens(tokenized_documents_method_three)


#chosen_tokens_method_one = random.choices(all_tokens_method_one, k=100 - len(stemmed_
#chosen_tokens_method_two = random.choices(all_tokens_method_two, k=100 - len(lemmatiz
#chosen_tokens_method_three = random.choices(all_tokens_method_three, k=100 - len(lemm


#lemmatizer = WordNetLemmatizer()
#lemmatized_chosen_tokens = [lemmatizer.lemmatize(x) for x in chosen_tokens]
#stemmed_chosen_tokens = [stemmer.stem(x) for x in lemmatized_chosen_tokens]




chosen_tokens_method_one = top_96_tfidf_terms_method_one_list + stemmed_important_prev
chosen_tokens_method_two = top_96_tfidf_terms_method_two_list + lemmatized_important_p
chosen_tokens_method_three = top_96_tfidf_terms_method_three_list + lemmatized_importa
```

2.4.2) Word2Vec Experiment Outputs - T-SNE Plots and Heatmaps

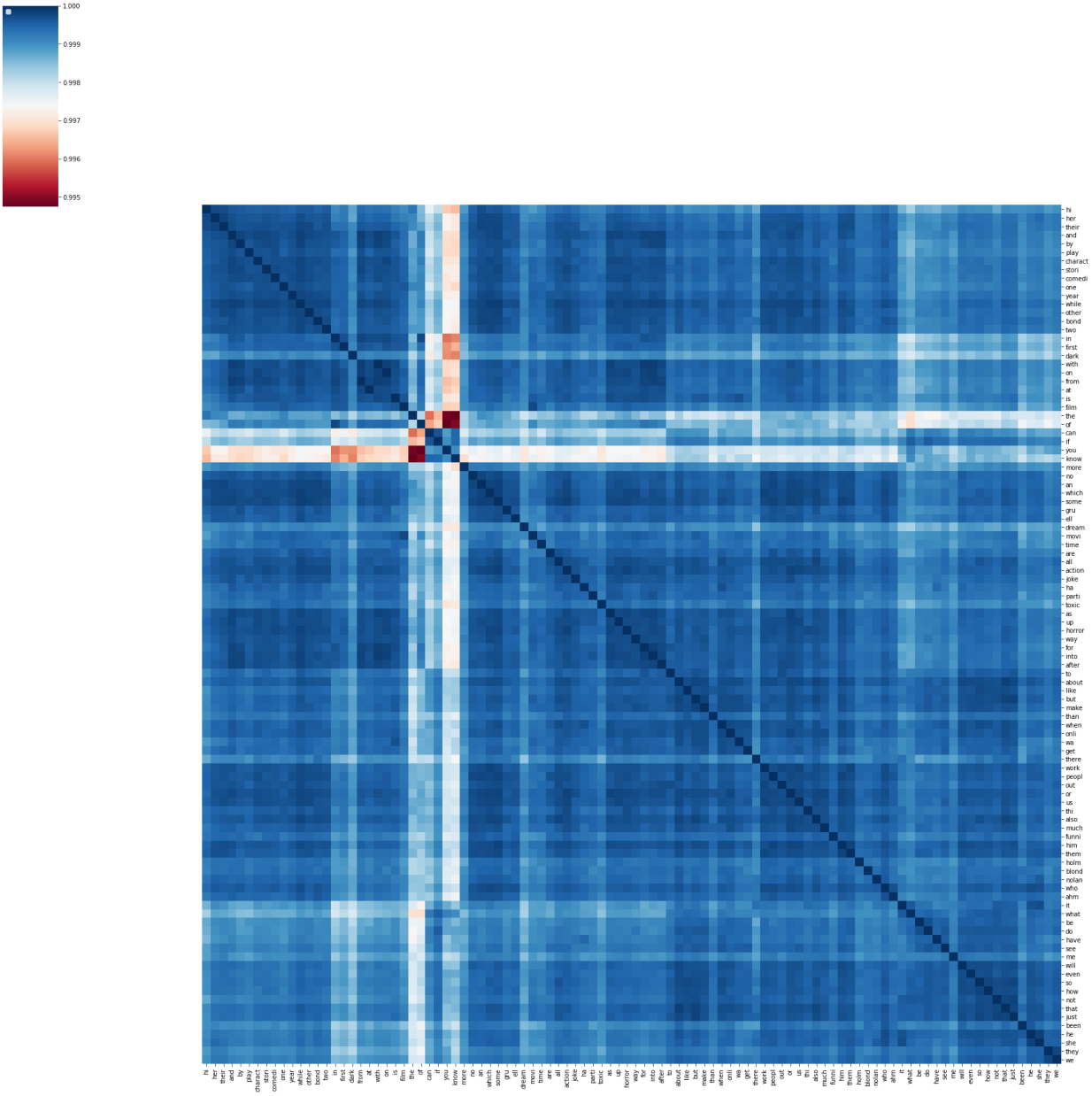
Having defined the functions needed for our analyses, let's run the experiment for each of our nine experimental scenarios of interest.

Word2Vec Experiment 1 - Data Wrangling Method 1 - 100 Embedding Dimensions

In [53]:
```
run_word2vec_experiment(tokenized_documents_method_one, 100, chosen_tokens_method_one,
```

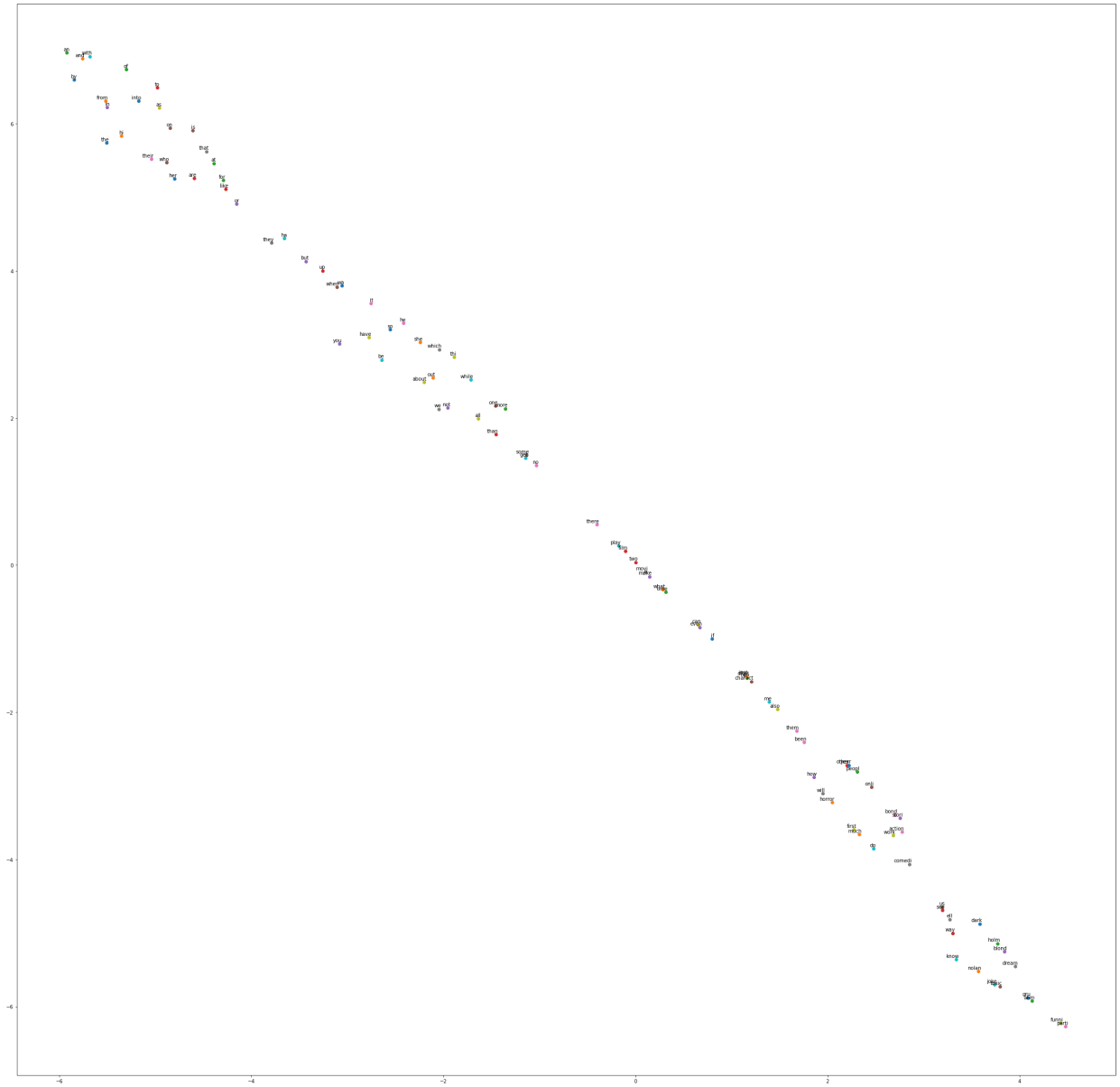


```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

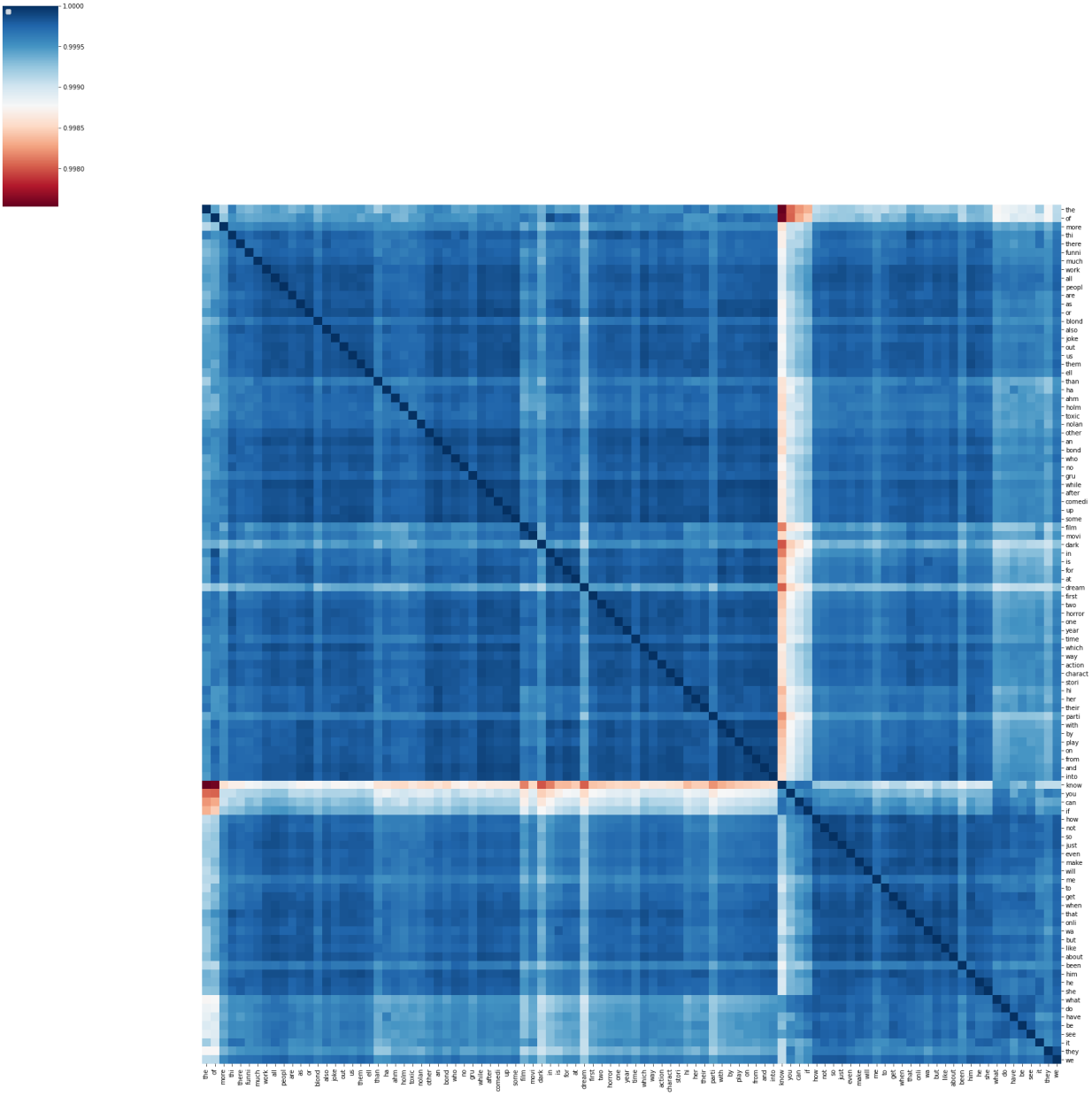


Word2Vec Experiment 2 - Data Wrangling Method 1 - 200 Embedding Dimensions

In [54]:
```
run_word2vec_experiment(tokenized_documents_method_one, 200, chosen_tokens_method_one,
```

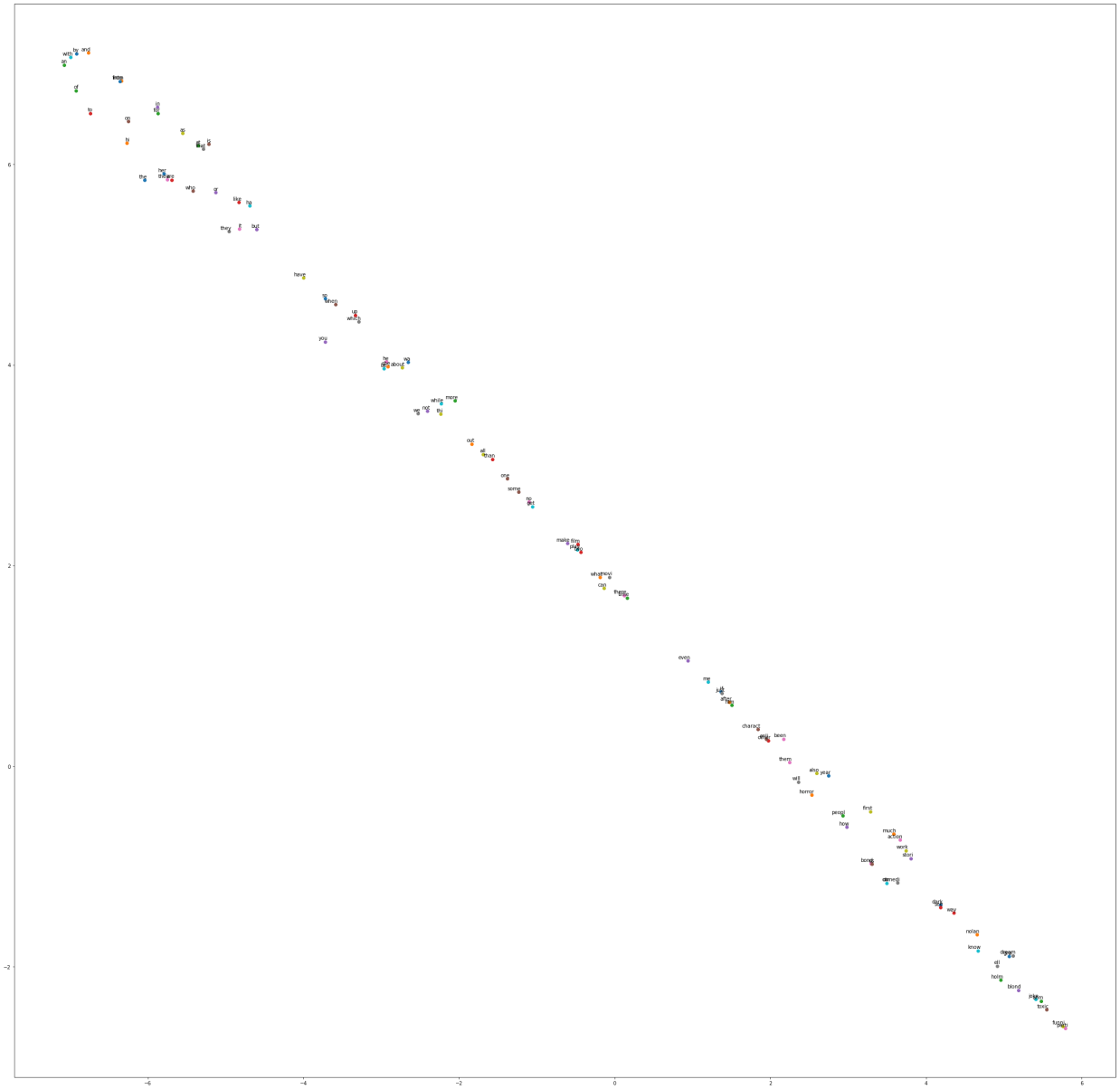


```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

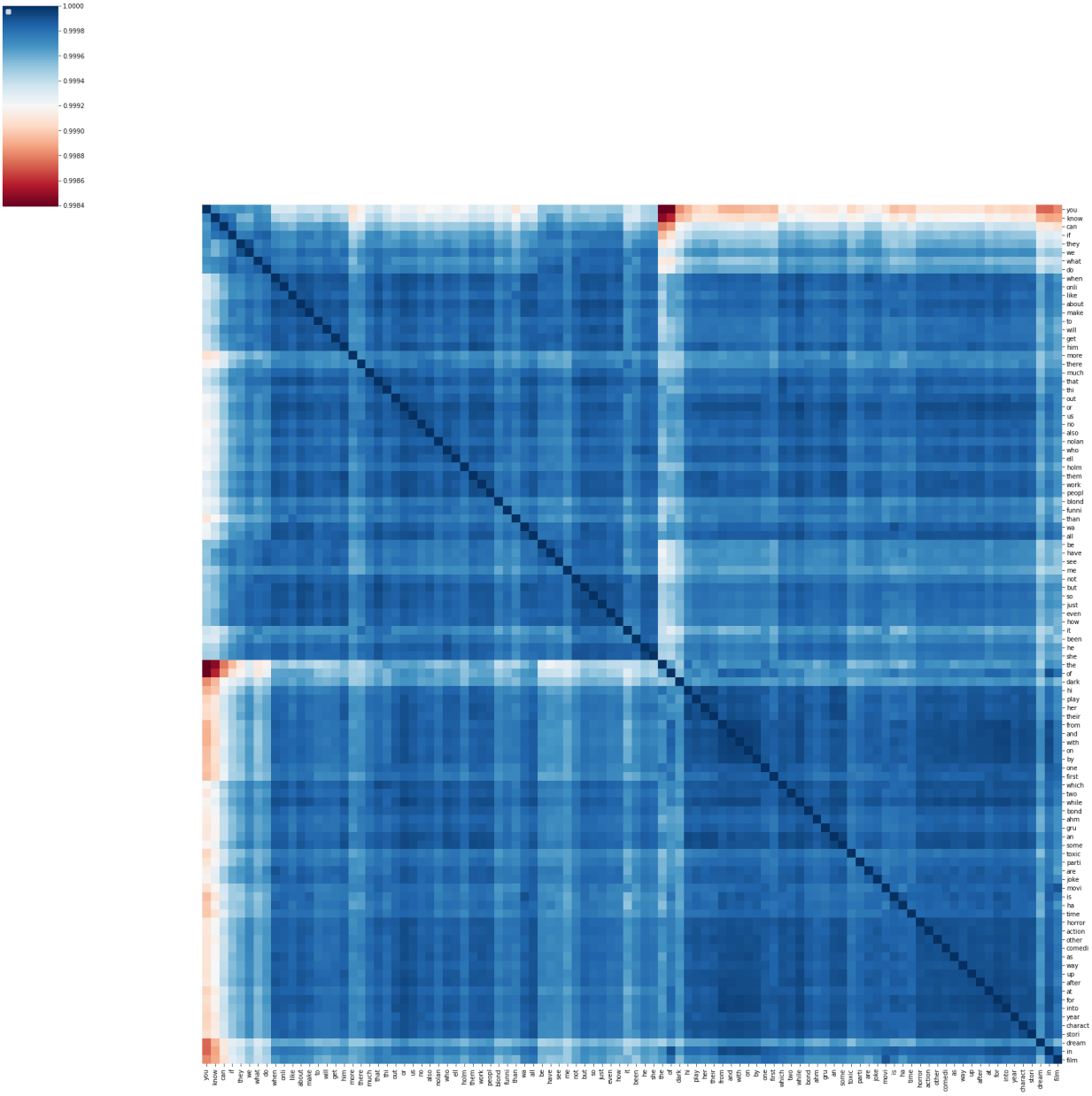


Word2Vec Experiment 3 - Data Wrangling Method 1 - 300 Embedding Dimensions

In [55]:
```
run_word2vec_experiment(tokenized_documents_method_one, 300, chosen_tokens_method_one,
```

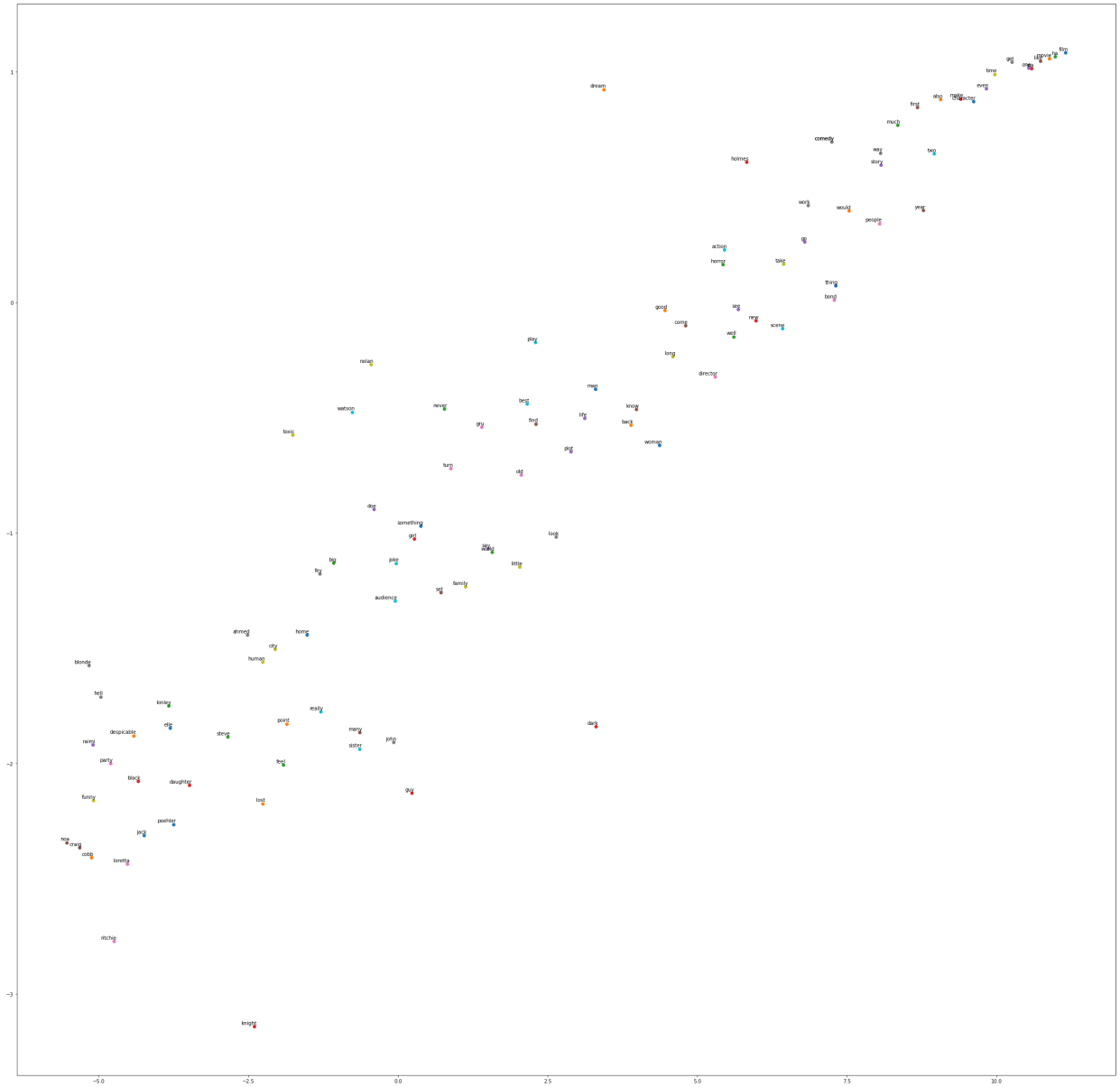


```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

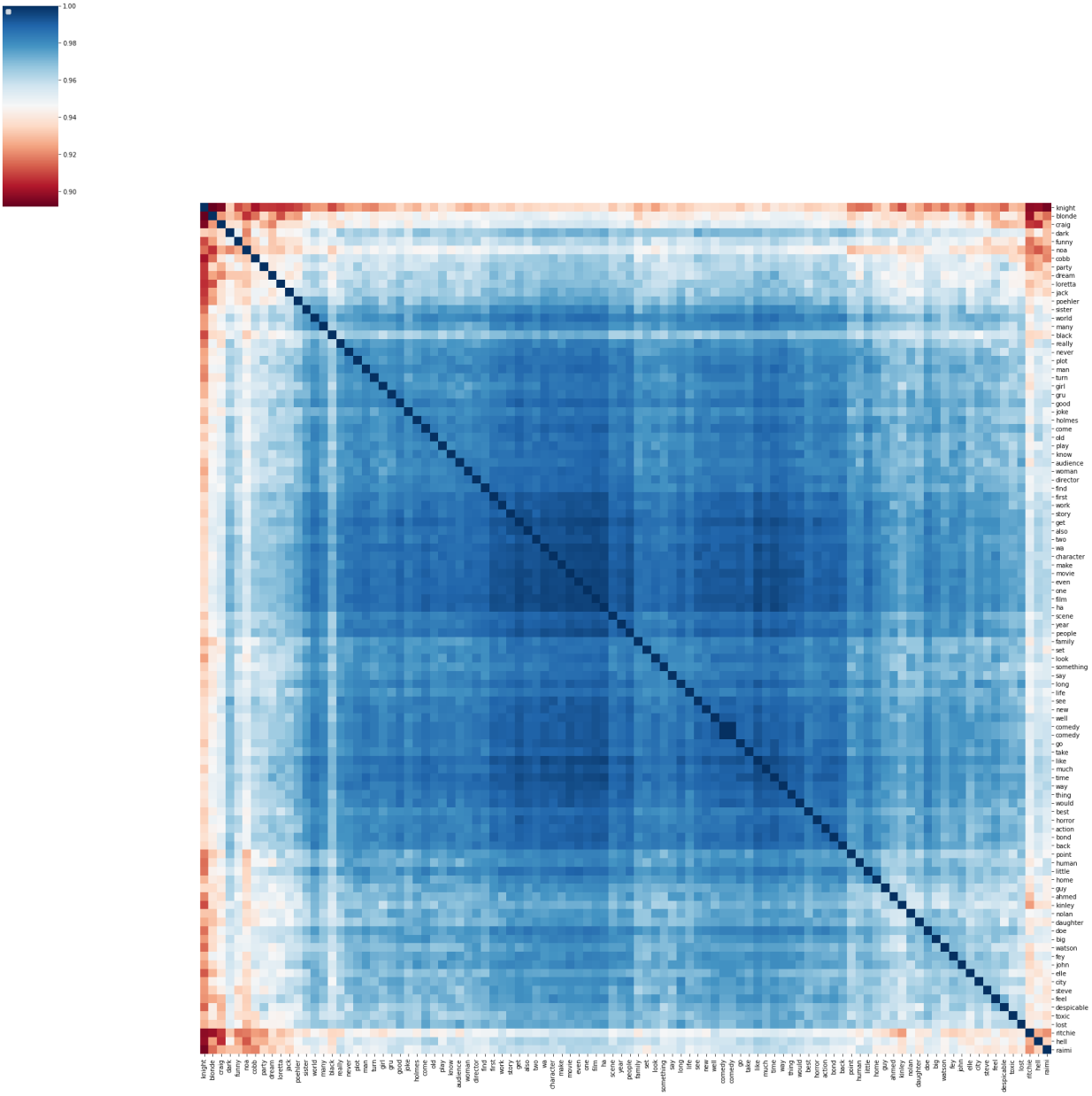


Word2Vec Experiment 4 - Data Wrangling Method 2 - 100 Embedding Dimensions

In [56]:
```
run_word2vec_experiment(tokenized_documents_method_two, 100, chosen_tokens_method_two,
```

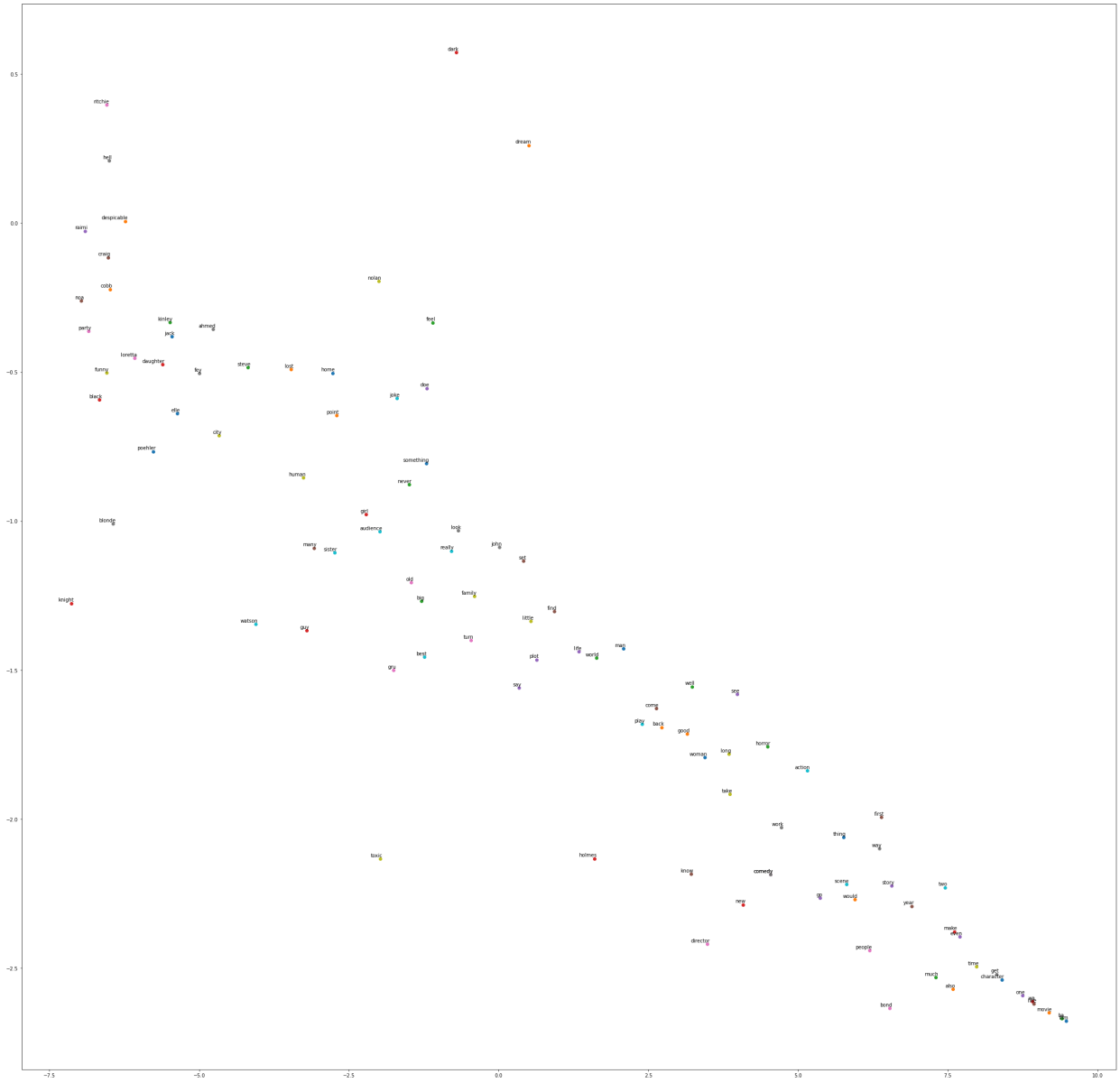



```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

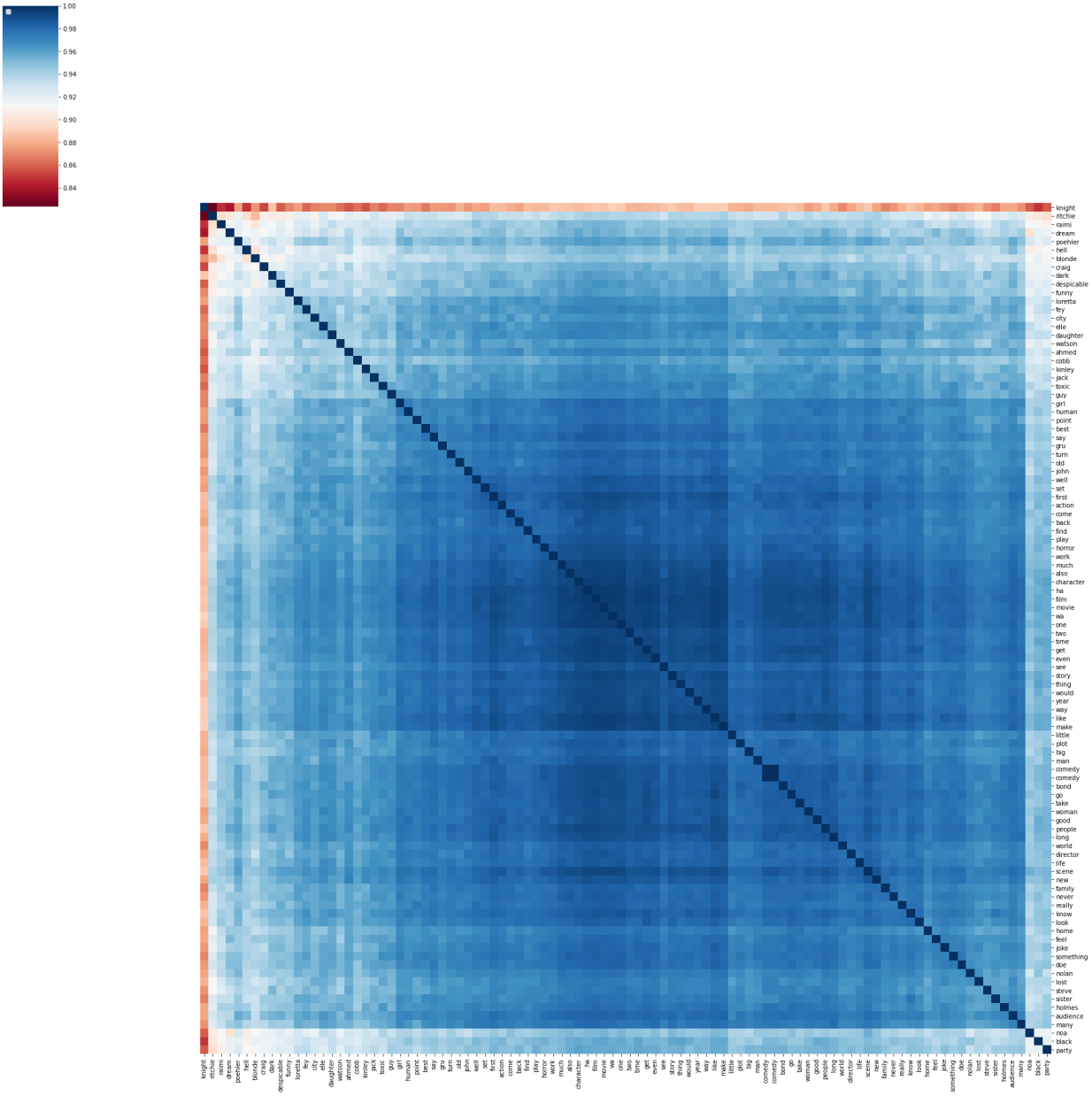


Word2Vec Experiment 5 - Data Wrangling Method 2 - 200 Embedding Dimensions

In [57]:
```
run_word2vec_experiment(tokenized_documents_method_two, 200, chosen_tokens_method_two,
```

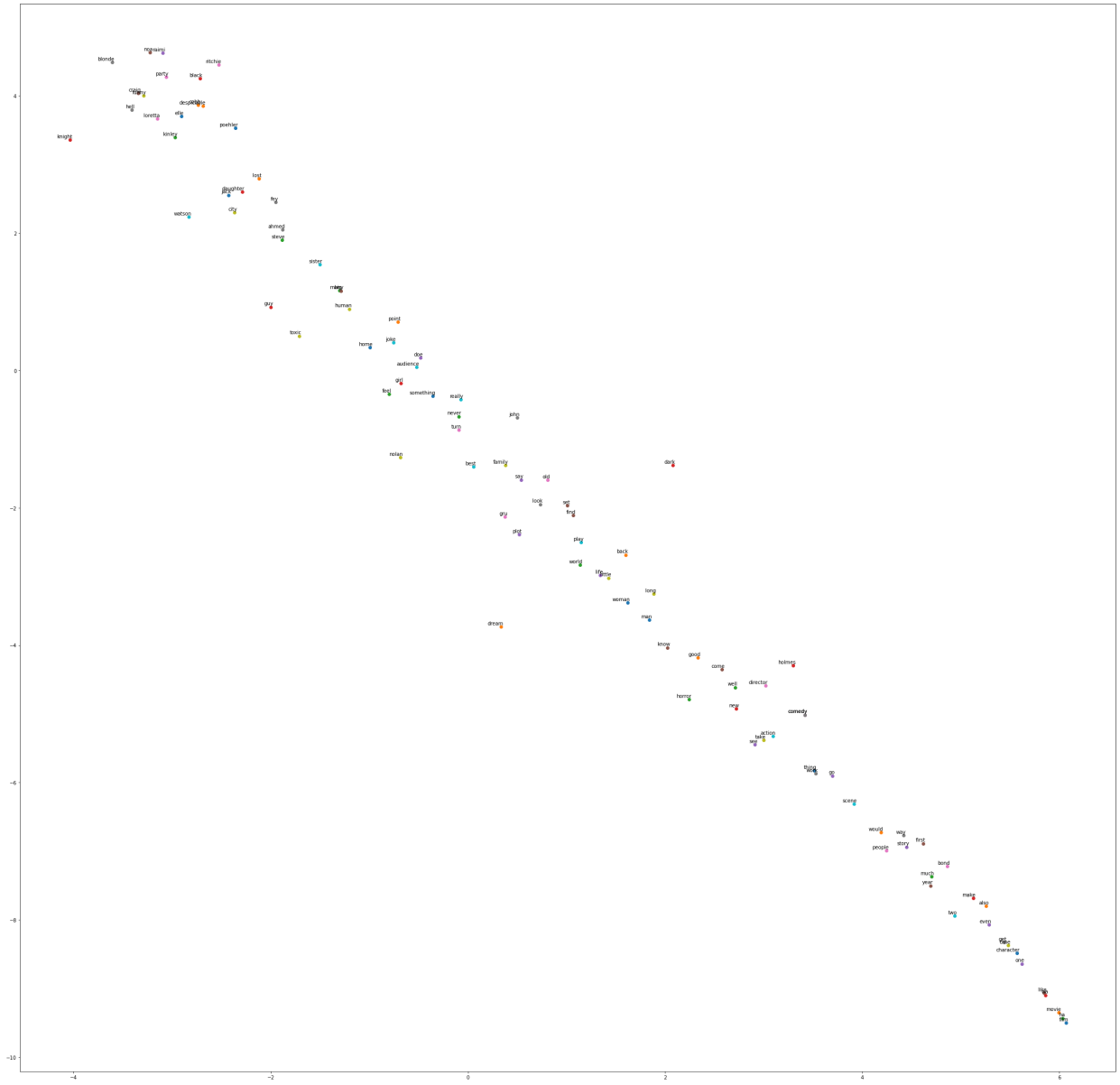



```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

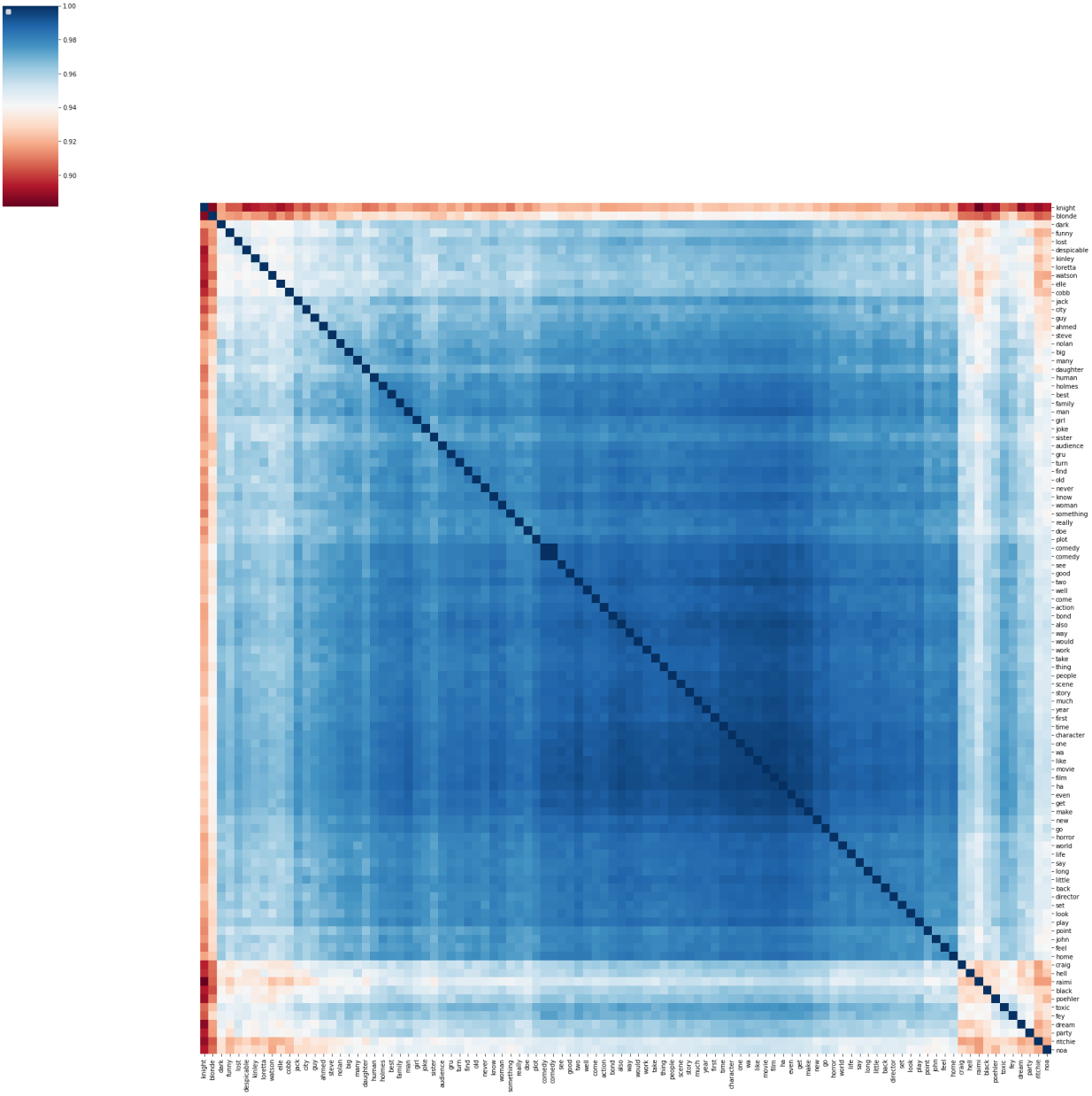


Word2Vec Experiment 6 - Data Wrangling Method 2 - 300 Embedding Dimensions

In [58]:
```
run_word2vec_experiment(tokenized_documents_method_two, 300, chosen_tokens_method_two,
```

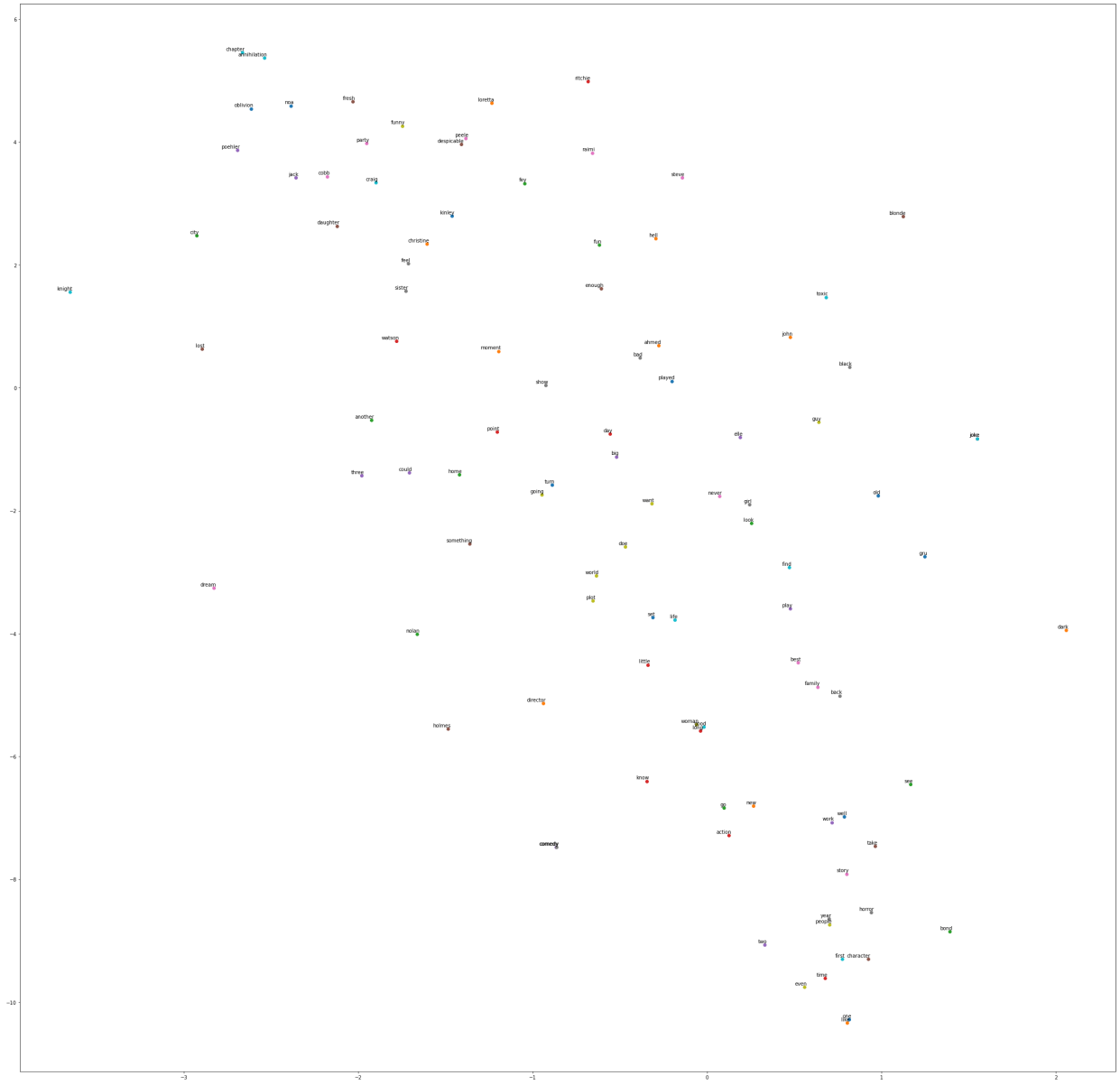



```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

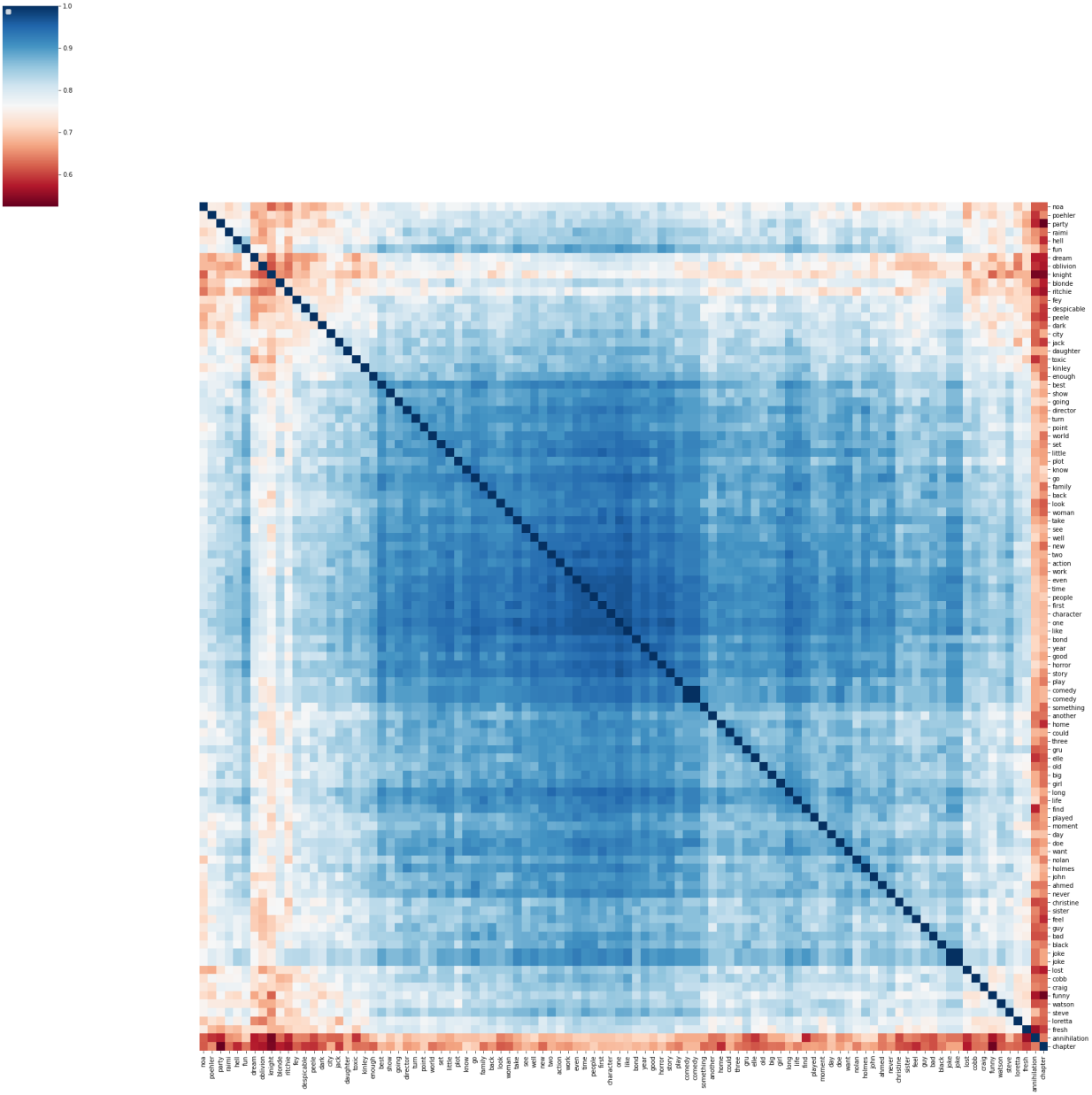


Word2Vec Experiment 7 - Data Wrangling Method 3 - 100 Embedding Dimensions

In [59]:
```
run_word2vec_experiment(tokenized_documents_method_three, 100, chosen_tokens_method_th
```

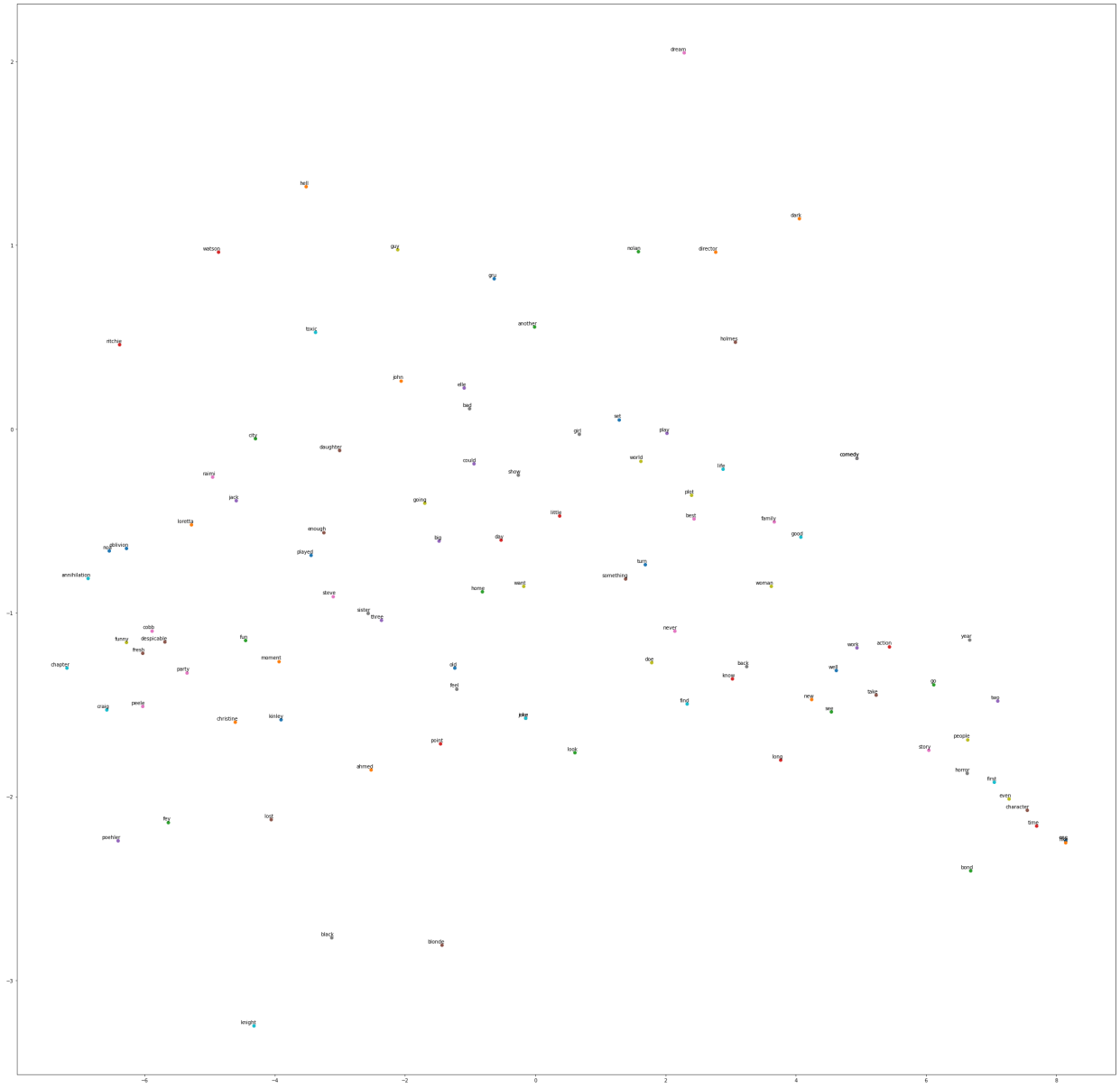


```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

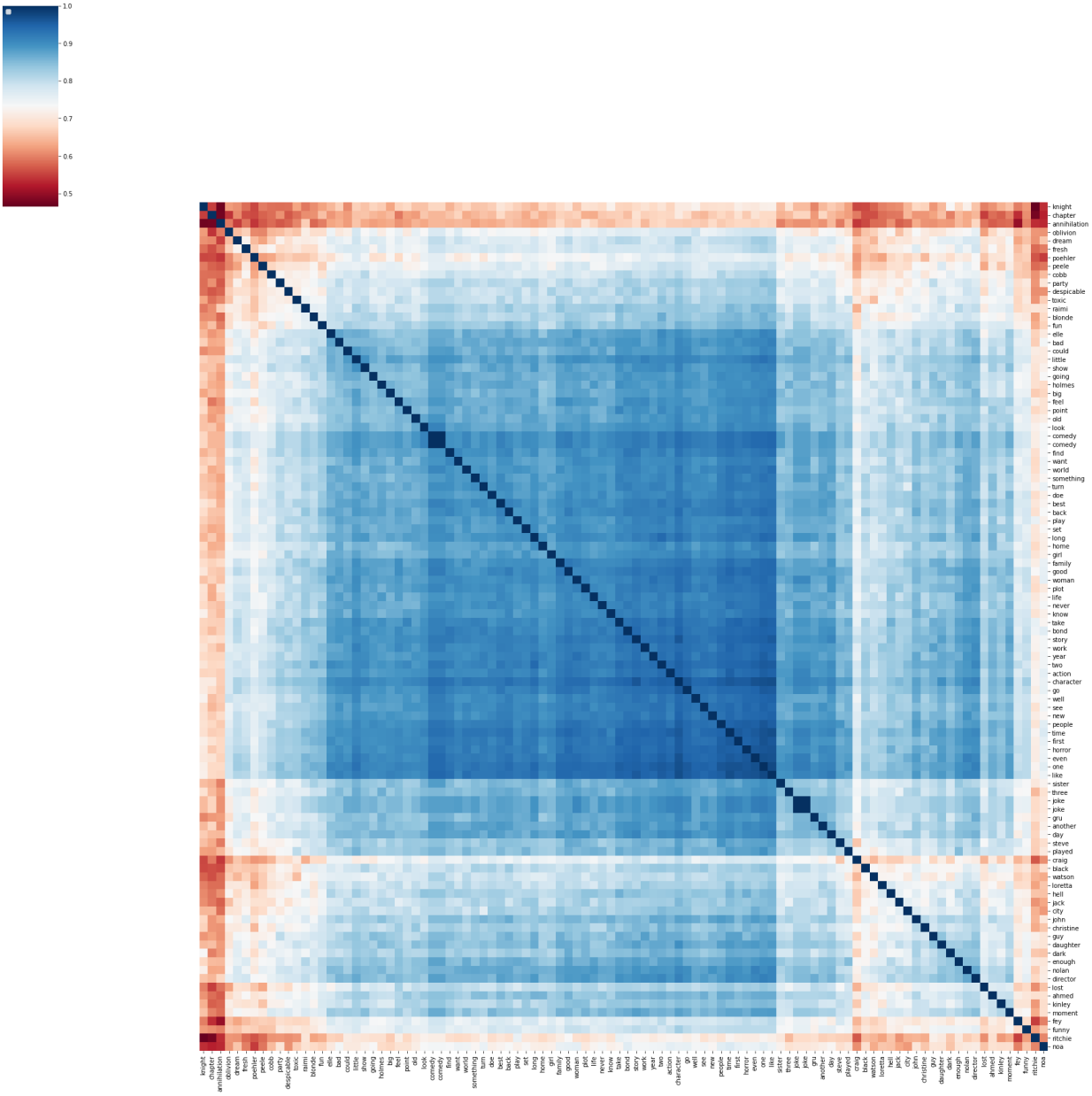


Word2Vec Experiment 8 - Data Wrangling Method 3 - 200 Embedding Dimensions

In [60]:
```
run_word2vec_experiment(tokenized_documents_method_three, 200, chosen_tokens_method_th
```

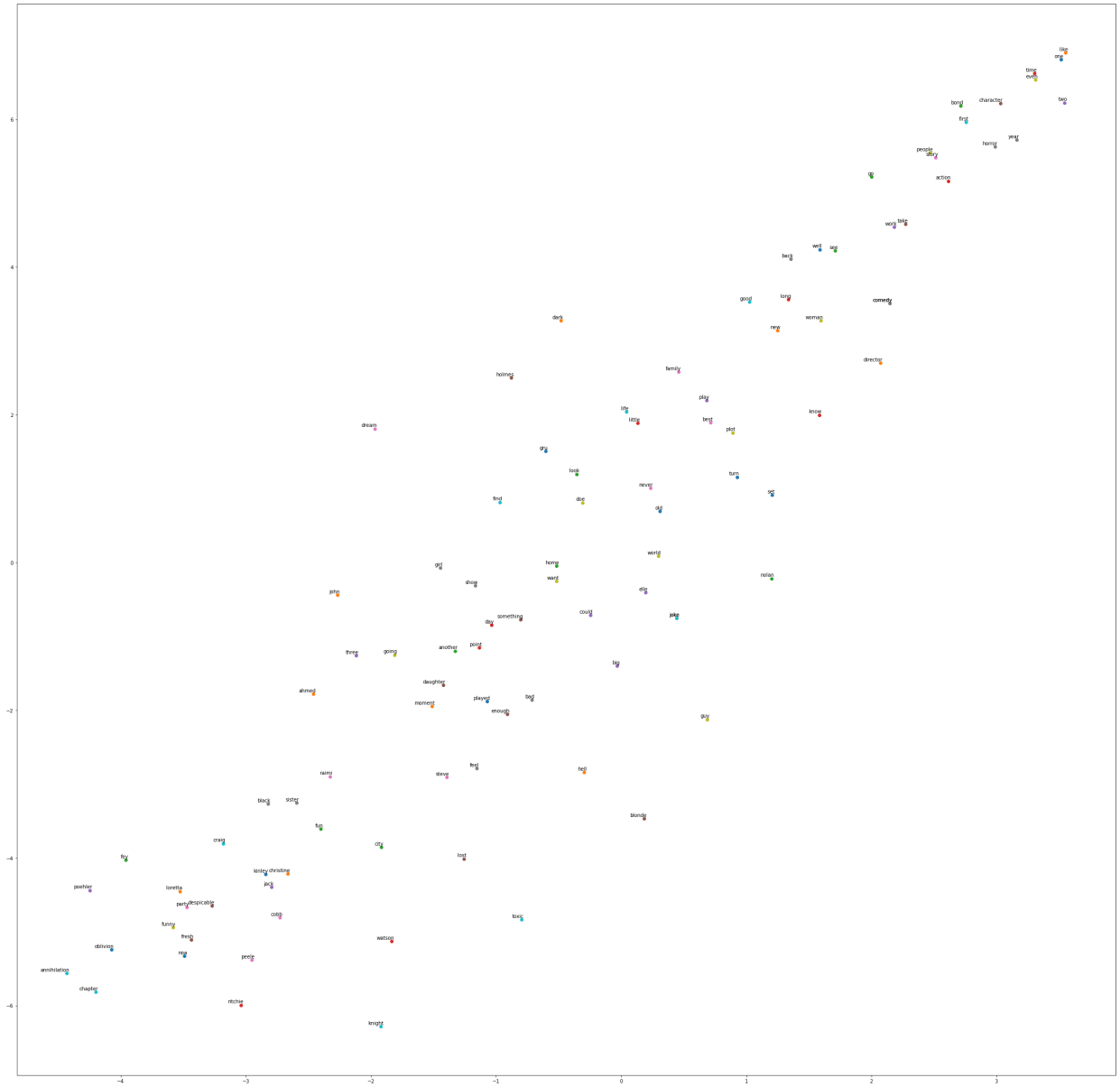



```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```

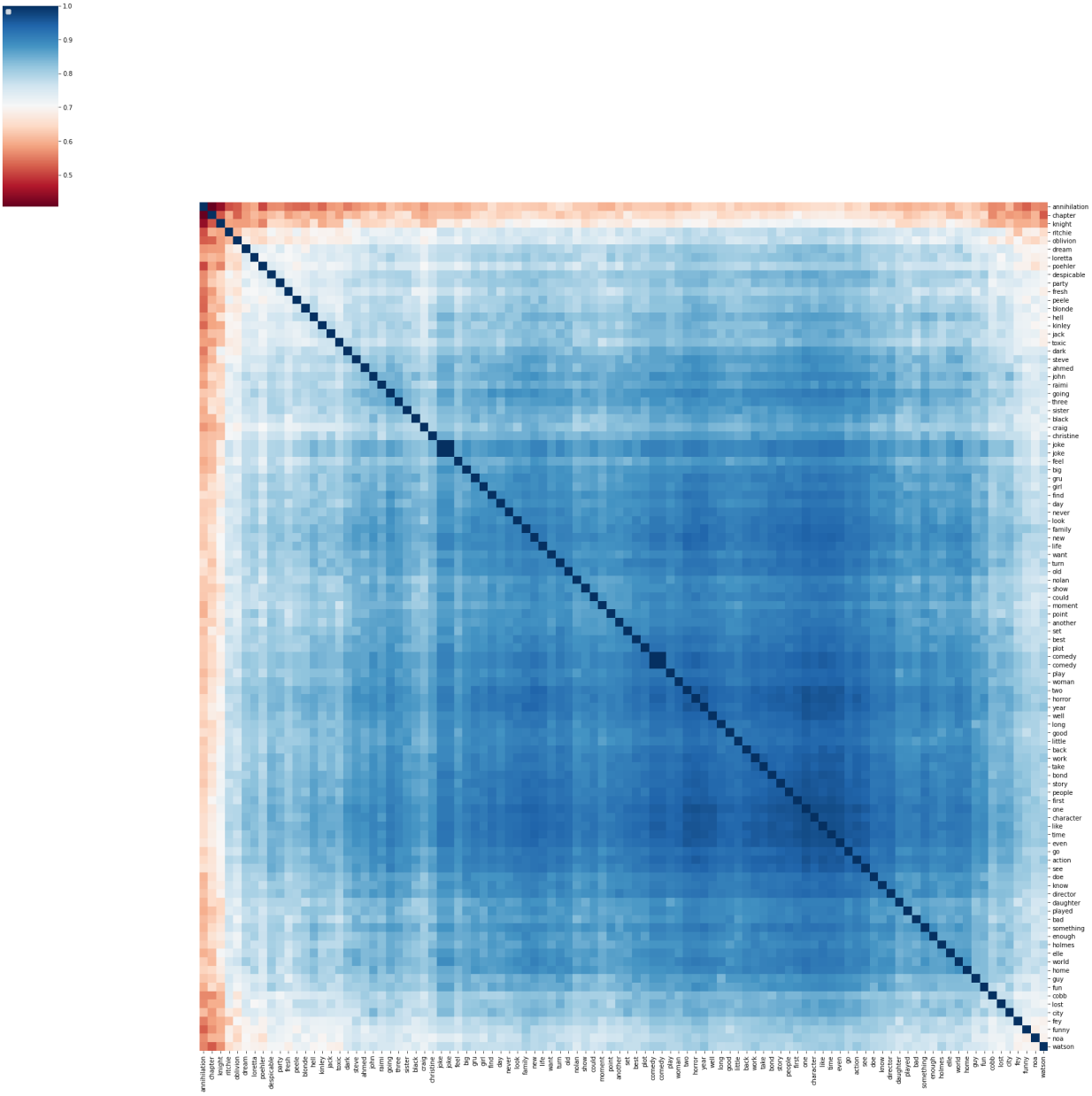


Word2Vec Experiment 9 - Data Wrangling Method 3 - 300 Embedding Dimensions

In [61]:
```
run_word2vec_experiment(tokenized_documents_method_three, 300, chosen_tokens_method_th
```




```
No artists with labels found to put in legend.  Note that artists whose label start w
ith an underscore are ignored when legend() is called with no argument.
```


1.0
0.9
0.8
0.7
0.6
0.5
annihilation
chapter
knight
ritchie
oblivion
dream
loretta
poehler
despicable
party
fresh
peele
blonde
hell
kinley
jack
toxic
dark
steve
ahmed
john
raimi
going
three
sister
black
craig
christine
joke
joke
feel
big
gru
girl
find
day
never
look
family
new
life
want
turn
old
nolan
show
could
moment
point
another
set
best
plot
comedy
comedy
play
woman
two
horror
year
well
long
good
little
back
work
take
bond
story
people
first
one
character
like
time
even
go
action
see
doe
know
director
daughter
played
bad
something
enough
holmes
elle
world
home
guy
fun
cobb
lost
city
fry
funny
noa
watson

In [ ]: