01_corpusCreationAndPreprocessing

Corpus Creation and Preprocessing¶

In [2]:

```
#from gutenbergpy.gutenbergcache import GutenbergCache, GutenbergCacheSettings, GutenbergCac
#import gutenbergpy.textget
from gensim.models import Word2Vec
from joblib import dump
import pandas as pd
import numpy as np
import spacy
import re
```

In [3]:

```
# Download and cache the Project Gutenberg Metadata to query it
#GutenbergCache.create() #refresh=False, download=True, unpack=True, parse=True, cache=True,
```

The titles for the corpus are taken from several sources: While the list of books collected by Caroline Winter and Eleanor Stribling as part of their works on exploring the color space in use in texts of gothic fiction, contributed both names and texts themselves, all further texts have been taken from Project Gutenberg. Ted Underwood created a dataframe of metadata on texs that David Punters and Glennis Byrons regard as the most prominent contributors to the genre within their seminal works on the Gothic. Many of the titles on that list have been gathered and enriched with further titles form authors mentioned in their text 'The Gothic' as well as the titles that Project Gutenberg attributes to this or closely related genres.

In [4]:

```
#cache = GutenbergCache.get_cache()
```

In [5]:

```
# cursor=cache.native_query("""
#                           SELECT
#                            MAX(b.gutenbergbookid) as book_id,
#                            CASE
#                                 WHEN MAX(a.name) = 'Boz' THEN 'Dickens, Charles'
#                                 WHEN MAX(a.name) = 'Marshall, William' THEN 'Walpole, Hora
#                                 WHEN MAX(a.name) = 'Grile, Dod' THEN 'Bierce, Ambrose'
#                                 ELSE MAX(a.name)
#                             END as authors,
#                             t.name as titles
#                           FROM books b
#                           join languages l on l.id=b.languageid and l.name='en'
#                           join book_subjects bsu on b.id=bsu.bookid
#                           join bookshelves bsh on b.bookshelveid=bsh.id
#                           join subjects s on s.id=bsu.subjectid
#                           join titles t on t.bookid=b.id
#                           join book_authors ba on ba.bookid=b.id
#                           join authors a on ba.authorid=a.id
#                           where
#                                 (lower(bsh.name) like '%horror%'
#                                 or lower(s.name) like '%horror%'
#                                 or lower(bsh.name) like '%gothic%'
#                                 or lower(s.name) like '%gothic%'
#                                 or lower(bsh.name) like '%supernatural%'
#                                 or lower(s.name) like '%supernatural%'
#                                 or lower(bsh.name) like '%paranormal%'
#                                 or lower(s.name) like '%paranormal%'
#                                 or lower(bsh.name) like '%vampire%'
#                                 or lower(s.name) like '%vampire%'
#                                 or lower(bsh.name) like '%ghost%'
#                                 or lower(s.name) like '%ghost%')
#                                 and a.name not in
#                                 ('Arthur, Robert','Baker, Frank','Baldwin, Edward','Birkhe
#                                 'Blackwood, Algernon','Bloxam, Matthew Holbeche','De Quinc
#                                 'DeQuincey, Thomas',
#                                 'De Vet, Charles V.','Glad, Victoria','Hammond, Keith', 'H
#                                 'Hopkins, R. Thurston (Robert Thurston)','Kafka, F. (Franz
#                                 'Leroux, Gaston','Littlewit, Humphrey','Marks, Winston K.
#                                 'O''Donnell, Elliot', 'Oliver, George', 'Olivieri, David',
#                                 'Peterson, Don', 'Lovecraft, H. P. (Howard Phillips)','Phi
#                                 'Tenneshaw, S. M.','Weinbaum, Stanley G. (Stanley Grauman)
#                                 and b.id not in (12728, 12739, 12751, 12762) -- collected
#                                 and b.gutenbergbookid not in (2147, 31469, 50133, 50133,20
#                                 50133, 24022, 20673, 20038, 13334, 6542, 19505, 19337, 182
#                                 14168,32076,18233,20034,14317, 24350, 25037) -- ill-fittin
#                             GROUP BY t.name
#                           """)

# results = cursor.fetchall()
# df = pd.DataFrame(results, columns=[column[0] for column in cursor.description])
```

Given that the package does not fetch biographical data on the authors or original publication dates, the filtering process had to be done manually.

In [6]:

```
# # This function downloads a book by its Gutenberg ID
# def download_book(book_id):
#     raw_book = gutenbergpy.textget.get_text_by_id(book_id)  # with headers
#     clean_book = gutenbergpy.textget.strip_headers(raw_book)  # without headers
#     return clean_book

# df['text'] = ''

# for idx, row in df.iterrows():
#     try:
#         df.loc[idx, 'text'] = download_book(row['book_id'])
#     except Exception:
#         continue
```

In [7]:

```
#df.to_csv('./preprocessing/corpora/Gutenberg_texts.csv', index=False)
```

The list of names obtained from the Table of contents of David Punters and Glennis Byrons book the Gothic filtered for all authors that were active before the beginning of the 20th centuary.

In [8]:

```
# names= ['William Harrison Ainsworth','William Beckford','E. F. Benson','Ambrose Bierce','
# 'Charles Brockden Brown','Robert W. Chambers','Wilkie Collins','Marie Corelli','Charlotte
# 'Isak Dinesen','Elizabeth Gaskell','William Godwin','H. Rider Haggard','Nathaniel Hawthor
# 'James Hogg','Washington Irving','G. P. R. James','Henry James','Francis Lathom','J. Sher
# 'Bulwer Lytton','George MacDonald','Arthur Machen', 'James Macpherson','Charles Robert Ma
# 'John Polidori','Ann Radcliffe','Clara Reeve','G. W. M. Reynolds','Walter Scott','Mary Wo
# 'Robert Louis Stevenson','Bram Stoker','Horace Walpole']
```

In [9]:

```
# formatted_names = []
# for name in names:
#     name_parts = name.split(" ")
#     last_name = name_parts[-1]
#     other_names = name_parts[:-1]
#     formatted_name = last_name + ', ' + " ".join(other_names)
#     formatted_names.append(formatted_name)

# names_string = ', '.join(f"'{name}'" for name in formatted_names)
```

In [10]:

```
# cursor=cache.native_query(f"""
#                           SELECT
#                             MAX(b.gutenbergbookid) as book_id,
#                             MAX(a.name) as authors,
#                             t.name as titles
#                           FROM books b
#                           join languages l on l.id=b.languageid and l.name='en'
#                           join book_subjects bsu on b.id=bsu.bookid
#                           join bookshelves bsh on b.bookshelveid=bsh.id
#                           join subjects s on s.id=bsu.subjectid
#                           join titles t on t.bookid=b.id
#                           join book_authors ba on ba.bookid=b.id
#                           join authors a on ba.authorid=a.id
#                           where
#                           (a.name in ({names_string}) or t.name in ('Wuthering Heights', '
#                           'Melmoth the Wanderer, Vol. 1','Melmoth the Wanderer, Vol. 2','M
#                           'Melmoth the Wanderer, Vol. 4','Auriol; or, The Elixir of Life',
#                           'The Wyvern mystery', 'A Stable for Nightmares; or, Weird Tales'
#                           'After Dark', 'The Haunted Hotel: A Mystery of Modern Venice', '
#                           'The Frozen Deep', 'The Dead Secret: A Novel', 'The Legacy of Ca
#                           'Phantom Fortune, a Novel', 'The Little Red Foot', 'In Search of
#                           'The Hidden Children', 'Phantastes: A Faerie Romance for Men and
#                           'Emmeline, the Orphan of the Castle', 'The House on the Borderla
#                           'The Sorrows of Satan', 'The soul of Lilith'))
#                           and a.name not in ('Mare, Walter de la', 'Melville, Herman','Art
#                           'Baldwin, Edward','Birkhead, Edith','Blackwood, Algernon','Bloxa
#                           'De Quincey, Thomas','DeQuincey, Thomas','De Vet, Charles V.','C
#                           'Hammond, Keith', 'Hodgson, William Hope','Mare, Walter de la',
#                           'Hopkins, R. Thurston (Robert Thurston)','Kafka, F. (Franz)','La
#                           'Leroux, Gaston','Littlewit, Humphrey','Marks, Winston K. (Winst
#                           'O''Donnell, Elliot', 'Oliver, George', 'Olivieri, David', 'Kafk
#                           'Peterson, Don', 'Lovecraft, H. P. (Howard Phillips)','Phillips,
#                           'Tenneshaw, S. M.','Weinbaum, Stanley G. (Stanley Grauman)')
#                           and b.id not in (12728, 12739, 12751, 12762)
#                           and b.gutenbergbookid not in (2147,8939, 8940,7023,7024, 31469,
#                           50133, 24022, 20673, 20038, 13334, 6542,4964, 4965,4966, 864,256
#                           14168,32076,18233,20034,14317, 24350, 25037, 13334, 14082, 6942,
#                           13334,9377, 771, 30486, 3606, 15697, 20656, 2834, 2833, 12031, 1
#                           and t.name not in ('The Star-Chamber: An Historical Romance, Vol
#                           'True Stories of History and Biography','The Scarlet Letter','Tw
#                           'Astoria; Or, Anecdotes of an Enterprise Beyond the Rocky Mounta
Volume 1',
#                           'The Portrait of a Lady - Volume 1', 'The Light Princess and Oth
#                           GROUP BY t.name
#                           ORDER BY a.name, t.name
#                           """)
# results = cursor.fetchall()
# df = pd.DataFrame(results, columns=[column[0] for column in cursor.description])
```

In [11]:

```
# # This function downloads a book by its Gutenberg ID
# def download_book(book_id):
#     raw_book = gutenbergpy.textget.get_text_by_id(book_id)  # with headers
#     clean_book = gutenbergpy.textget.strip_headers(raw_book)  # without headers
#     return clean_book

# df['text'] = ''

# for idx, row in df.iterrows():
#     try:
#         df.loc[idx, 'text'] = download_book(row['book_id'])
#     except Exception:
#         continue
```

In [12]:

```
# df_init=pd.read_csv('./preprocessing/corpora/Gutenberg_texts.csv')
# df['source'] = 'liste'
# df_init['source'] = 'bookshelf'
# union_df = pd.concat([df, df_init])
# sorted_df = union_df.sort_values('authors')
# unique_df = sorted_df.drop_duplicates(subset='titles')
```

In [13]:

```
# unique_df.to_csv('./preprocessing/corpora/Gutenberg_texts_full.csv', index=False)
```

In [14]:

```
#df_gutenberg=pd.read_csv('./preprocessing/corpora/Gutenberg_texts_full.csv')
```

Given the limitet amount that the python package is capable of retrieving, manual downloads will be necessary as well. Given the quality of the texts and the excessive paratexts, some cleaning will be necessary as well.

The gothic colors corpus contains a number of intersting texts, as well as highly relevant metadata, we shall make us of here.

It shall be restricted to relevant texts, joined with our existing dataframe and the files read in.

In [15]:

```
# df_colors=pd.read_csv('./preprocessing/corpora/gothic_texts.csv')
# df_colors = df_colors[~(df_colors['Nationality'].isin(['French', 'German', 'Italian', 'Sic
#            df_colors['Genre'].isin(['Aesthetic Theory','Criticism', 'Literary Theory', 'Me
```

In [16]:

```
# df_colors['Text'] = ""
# dir_path = "/Storage/Studium/DigitalHumanities/Semester5/Thesis/code_notebooks/color_corpu


# for index, row in df_colors.iterrows():
#     if pd.notnull(row['Filename']):
#         file_path = os.path.join(dir_path, row['Filename'])
#         with open(file_path, 'r') as file:
#             text = file.read()
#         df_colors.at[index, 'Text'] = text


# filled_rows = df_colors[df_colors['Text'] != ''].shape[0]
# print(filled_rows)
```

The features are normalization to homogenize the sources

In [17]:

```
# df_colors.columns = df_colors.columns.str.lower()
# df_colors['source'] = 'colors corpus'
# df_gutenberg = df_gutenberg.rename(columns={'authors': 'author','titles': 'title'})
# df_combined = pd.concat([df_colors, df_gutenberg])
# df_combined['text'].replace('', np.nan, inplace=True)
# df_combined['text_filled'] = df_combined['text'].notna().astype(int)
# df_combined["sort_helper"] = df_combined["source"].apply(lambda x: 0 if x == "colors corpu
# df_combined = df_combined.sort_values(by=['text_filled', 'author', 'title', 'sort_helper']
```

In [18]:

```
# #Export of the first five columns for manual correction before unification
#df_combined.iloc[:, :5].to_csv('./preprocessing/intermediary_steps/Combined_texts_non_tidy.
```

In [19]:

```
# # Import the data back into a DataFrame
# df_imported = pd.read_csv('./preprocessing/intermediary_steps/Combined_texts_non_tidy.csv'
# df_combined[df_imported.columns] = df_imported
# df_combined = df_combined.sort_values(by=['text_filled', 'author', 'title', 'sort_helper']
# df_combined = df_combined.drop_duplicates(subset=['author', 'title'], keep='first')
# df_combined.drop(['text_filled', 'sort_helper'], axis=1, inplace=True)
# df_combined = df_combined.dropna(subset=['text'])
```

In [20]:

```
#df_combined.drop(['publisher', 'pseudonym', 'publishing house', 'city of publication', 'loc
#df_combined.insert(8, 'gender', np.nan)
#df_combined.insert(9, 'birthdate', np.nan)
```

Given that the features gende, publication date, birthyear and nationality provided by the colors corpus seem very promisingn and useful, but are not prvided by any of the other sources, The gap will need to be filled with manual research

In [21]:

```
#df_combined.to_csv('./preprocessing/intermediary_steps/Combined_texts_work_in_progress.csv'
```

In [22]:

```
##Now that it is clear which rows will remain we will do some manual enrichment to fill up t
#df_combined.iloc[:, :10].to_csv('./preprocessing/intermediary_steps/combined_texts_to_be_en
```

Now we import and finalize the texts.

In [24]:

```
#df_combined=pd.read_csv('./preprocessing/intermediary_steps/Combined_texts_work_in_progress
#df_enriched = pd.read_csv('./preprocessing/intermediary_steps/Combined_texts_non_tidy.csv')
#df_combined[df_enriched.columns] = df_enriched
```

We import the texts poreviously used by Ted Underwood in one of his papers and reduce it to two categories, the stanford gothic collection and the byron and punther gothic selection: Ted Underwood, “The Life Cycles of Genres,” Cultural Analytics May 23, 2016. DOI: 10.22148/16.005

p.34f.) The metadata I use for the “Stanford Gothic” were developed at the Stanford Literary Lab; many hands may have been involved, including certainly those of Ryan Heuser and Matthew L. Jockers.

In [26]:

```
# stan_df = pd.DataFrame()

# for index, row in df_genre.iterrows():
#     # Check if 'stangothic' is in 'genretags' column for the row
#     tags = [tag.strip() for tag in row['genretags'].split('|')]
#     if 'stangothic' in tags:
#         # Append the row to stan_df
#         stan_df = pd.concat([stan_df, pd.DataFrame(row).T])

# # Reset index for the new DataFrame
# stan_df = stan_df.reset_index(drop=True)
```

In [27]:

```
# pb_df = pd.DataFrame()

# for index, row in df_genre.iterrows():
#     # Check if 'pbgothic' is in 'genretags' column for the row
#     tags = [tag.strip() for tag in row['genretags'].split('|')]
#     if 'pbgothic' in tags:
```

```
#         # Append the row to pb_df
#         pb_df = pd.concat([pb_df, pd.DataFrame(row).T])

# # Reset index for the new DataFrame
# pb_df = pb_df.reset_index(drop=True)
```

In [28]:

```
#stan_df.drop(['enumcron', 'date', 'imprint', 'locnum','oclc', 'recordid'], axis=1, inplace=
#pb_df.drop(['enumcron', 'date', 'imprint', 'locnum','oclc', 'recordid'], axis=1, inplace=Tr
#pb_df['source'] = 'life cycle: pb'
#stan_df['source'] = 'life cycle: stan'
#stan_df.rename(columns={'firstpub': 'date'}, inplace=True)
#pb_df.rename(columns={'firstpub': 'date'}, inplace=True)
#stan_df['nationality'].replace({'uk': 'English', 'ir': 'Irish', 'us': 'American'}, inplace=
#pb_df['nationality'].replace({'uk': 'English', 'ir': 'Irish', 'us': 'American'}, inplace=Tr
#pb_df = pb_df[pb_df['date'] <= 1910]
```

In [29]:

```
# df_colors['Text'] = ""
# dir_path = "/Storage/Studium/DigitalHumanities/Semester5/Thesis/code_notebooks/preprocessi


# for index, row in df_colors.iterrows():
#     if pd.notnull(row['Filename']):
#         file_path = os.path.join(dir_path, row['Filename'])
#         with open(file_path, 'r') as file:
#             text = file.read()
#         df_colors.at[index, 'Text'] = text


# filled_rows = df_colors[df_colors['Text'] != ''].shape[0]
# print(filled_rows)
```

In [30]:

```
# pb_df.to_csv('./preprocessing/intermediary_steps/Underwood_punter_selection.csv', index=Fa
# stan_df.to_csv('./preprocessing/intermediary_steps/Underwood_stanford_selection.csv', inde
```

In [31]:

```
#df_text=pd.read_csv('./preprocessing/intermediary_steps/df_books_v2.csv')
```

Correcting Mojibake in Project Gutenberg package texts. First we replace the \n sequences with actual newlines. The decode_match function is used to convert each matched sequence to its actual UTF-8 character. Then a regular expression is used to find all byte-like sequences (e.g., \xe2) and allow for recognition and removal of these characters.

In [32]:

```
def clean_text(raw_text):
    # Convert \n sequences to actual newlines
    text = raw_text.replace('\\n', '\n')

    # Convert byte-like sequences to their actual characters
    def decode_match(match):
        return bytes.fromhex(match.group(1)).decode('utf-8', errors='replace')

    text = re.sub(r'\\x([a-fA-F0-9]{2})', decode_match, text)
    if text.startswith("b'"):
        text = text[2:]

    # Remove any leading newline characters and still unrecognized bytestring
    text = text.lstrip('\n')
    text = re.sub(r' +', ' ', text)

    return text
```

In [33]:

```
# # Apply the cleaning function to the  gutenberg books
# df_text.loc[df_text['book_id'].notna(), 'text'] = df_text.loc[df_text['book_id'].notna(),
```

In [34]:

```
# df_text['source'].replace({'liste': 'pb-manual', 'colors corpus': 'colors', 'life cycle: p
# df_text.drop(['filename', 'subjects','book_id','Unnamed: 0', 'docid', 'genretags'], axis=1
```

Now we intruduce a telling identifier to use for further text identification. The first up to ten letters of both authors last name and title of the text. In the scant cases of overlapping ids we shall ad a distinguishing number at the end

In [35]:

```
def remove_punctuation(s):
    return re.sub(r'[^\w\s]', '', s)

# Function to generate the unique value for the index
def generate_unique_value(row):
    # Extract from author
    author_name = remove_punctuation(row['author'].split(',')[0]).replace(' ', '')
    author_part = author_name[:10]

    # Extract from title
    title_part = remove_punctuation(row['title']).replace(' ', '')
    for article in ['A', 'An', 'The']:
            title_part = re.sub(r'\b' + article + r'\b', '', title_part, flags=re.IGNORECASE
    title_part = title_part[:10]
    return author_part + '_' + title_part
```

In [36]:

```
# # Apply the function to each row and store the result in a temporary variable
# reference_values = df_text.apply(generate_unique_value, axis=1)

# # Insert the reference column as the first column in the DataFrame
# df_text.insert(1, 'reference', reference_values)
```

In [37]:

```
# def check_for_duplicates(df):
#     # Check for duplicates in the reference column
#     duplicates = df[df['reference'].duplicated(keep=False)]

#     # Print the duplicates
#     if not duplicates.empty:
#         print("Duplicate values in the reference column:")
#         print(duplicates[['reference']])
#     else:
#         print("No duplicate values found in the reference column.")
```

In [38]:

```
#check_for_duplicates(df_text)
```

lets adjust the relevant entries, there were still some duplicate books in there
and two books sadly are identical in their reference till the 10th character

In [39]:

```
# values_to_remove = [228, 173, 174, 14]
# df_text = df_text[~df_text['index'].isin(values_to_remove)]
```

In [40]:

```
# # Identify the second occurrence of duplicates
# second_occurrences = df_text[df_text.duplicated(subset='reference', keep='first')]

# # Add '2' to the end of the reference value for these rows
# df_text.loc[second_occurrences.index, 'reference'] = second_occurrences['reference'] + '2
```

In [41]:

```
#check_for_duplicates(df_text)
```

none

In [42]:

```
#df_text.to_csv('./preprocessing/intermediary_steps/df_books_completed.csv', index=False)
```

In [43]:

```
#df_reg=pd.read_csv('./preprocessing/intermediary_steps/df_books_completed.csv')
```

Now we will go on to finalize the preprocessing for the following modelling in the subsequent notebook.

In [98]:

```
# max_length = df_reg['text'].apply(len).max()
# print(f"The maximum text length in the 'text' column is: {max_length}")
```

The literature suggests that the most important and salient types of words for topic modeling are nouns, verbs, adjectives and adverbs. Those are extracted with the use of a spacy language model. Followed by regex based cleaning

In [44]:

```
# # Load English tokenizer, tagger, parser, NER and word vectors
# nlp = spacy.load("en_core_web_lg")
# nlp.max_length = 1640523
# # Define the function to preprocess text
# def preprocess_text(doc):
#     # Parse the sentence using the loaded 'en' model object `nlp`
#     doc = nlp(doc)
#     # Lower case the text, remove stop words, punctuation and words not chosen for the moc
#     result = []
#     for token in doc:
#         if token.pos_ in ('NOUN', 'VERB', 'ADJ', 'ADV') and not token.is_stop and not toke
#             result.append(token.text.lower())
#     return ' '.join(result)

# # Now apply this function to the 'text' column in the dataframe
# df_reg['preprocessed_text'] = df_reg['text'].apply(preprocess_text)
```

In [47]:

```
# def remove_unwanted_elements(text):
#     # Define the pattern to match unwanted symbols
#     symbols_pattern = re.compile(r"[+\-|\\\"\"\[\]â~ - \‘‘’\(\)\•€\•\?\,\’\;\-
"\*\{\}!?\./':\_\<\>;=,?\d+]")

#     # Define the pattern to match unwanted words
#     words_pattern = re.compile(r"\b(illustration|use|cost|restriction|restrictions|proofre
#                                r"chapter|ebook|ebooks|chapters|contents|author|published|i
#                                r"httpswwwpgdpnet|  |version|file|original|volume|copyright

#     # Read the additional unwanted words from a file
#     with open('./preprocessing/unwanted_terms.txt', 'r') as file:
#         additional_words = [line.strip() for line in file if line.strip()]
#     # Create a regex pattern for the additional unwanted words
#     additional_words_pattern = re.compile(r'\b(' + '|'.join(additional_words) + r')\b', re

#     # Remove unwanted symbols
#     text = symbols_pattern.sub(" ", text)

#     # Remove unwanted words
#     text = words_pattern.sub("", text)

#     # Regex for additional unwanted words - mostly in old greek and latin found in later s
#     #Those are too long to be displayed here, so theyx aqre read-in from a file

#     text = additional_words_pattern.sub("", text)

#     # Remove extra spaces
#     text = re.sub(' +', ' ', text).strip()

#     return text

# # Example of applying the function to a dataframe column
# df_reg['preprocessed_text'] = df_reg['preprocessed_text'].apply(remove_unwanted_elements)
```

In [3]:

```
#df_reg.to_csv('./preprocessing/results/df_books_prep.csv', index=False)
df_prep=pd.read_csv('./preprocessing/results/df_books_prep.csv')
```

Now for a final clean up to remove further texts from the selection. After the modeling had been completed, due to unsatisfactory results and a skew in the data, many of the entries from the pb-under, as well as some of the color and many of the color works had been manually re-evaluated and taken out of the corpus.

In [4]:

```
to_remove=[
 'Armadale',
 'The Sketch-Book of Geoffrey Crayon',
 'The Alhambra',
 'Godolphin',
 "King Solomon's Mines",
 'Phantastes',
 'Hours Of Solitude', 'Ormond; Or, The Secret Witness',
 'A Christmas Carol in Prose; Being a Ghost Story of Christmas',
 'Bleak House',
 'Great Expectations',
 'Oliver Twist',
 'She: A History Of Adventure',
 'The People of the Mist',
 'At the Back of the North Wind',
 'St. George and St. Michael, Volume 1',
 'The Light Princess',
 'The Princess and Curdie',
 'A Legend of Montrose',
 'Guy Mannering; or, The Astrologer',
 'Ivanhoe: A Romance',
 'Kenilworth',
 "Letters on Demonology and Witchcraft",
 'Old Mortality, Complete',
 'Peveril of the Peak',
 'Quentin Durward',
 'Redgauntlet: A Tale of the Eighteenth Century',
 'Rob Roy - Complete',
 "St. Ronan's Well",
 'The Antiquary - Complete',
 'The Abbot',
 'The Betrothed',
 'The Bride of Lammermoor',
 "The Fair Maid of Perth; Or, St. Valentine's Day",
 'The Fortunes of Nigel',
 'The Heart of Mid-Lothian, Complete',
 'The Monastery',
 "The Surgeon's Daughter",
 'The Talisman',
 "Waverley; Or, 'Tis Sixty Years Since"]
```

```
#df_prep = df_prep[~df_prep['title'].isin(to_remove)]
```

In [5]:

```
# Split the preprocessed text into lists of words
df_prep['tokenized_text'] = df_prep['preprocessed_text'].apply(lambda text: text.split())
#df_prep.to_csv('./preprocessing/results/df_books_prep.csv', index=False)
```

Word2vec embeddings, a representation of the language space of the corpus is generated in order to use this as an input for our evaluation metrics in the modeling, as well as input for the ETM

According to research, the quality for vector representations improves as you increase the vector size until you reach 300 dimensions. After 300 dimensions, the quality of vectors starts to decrease.¶

In [7]:

```
# # # Train Word2Vec Model
# # # Train the model on the tokenized sentences
# word2vec_model = Word2Vec(sentences=df_prep['tokenized_text'].tolist(), vector_size=300, w

# # # # If you want to save the model to disk
# word2vec_model.save("./word2vec/word2vec_model")
```

In [6]:

```
# word2vec_model = Word2Vec.load("./word2vec/word2vec_model")
# word_vectors = word2vec_model.wv
# print(word_vectors['human']) #looking good
```

```
[-2.87400156e-01  3.63597661e-01  9.53967869e-02  1.36146903e-01
  6.52683750e-02 -3.91167045e-01  2.38470286e-01  7.39968777e-01
  5.86055398e-01 -9.86738428e-02 -3.09907436e-01 -9.16796252e-02
  4.71953511e-01  2.92291611e-01 -4.55206633e-01 -2.37952515e-01
  6.80520851e-03 -1.94612384e-01  7.38793671e-01  2.93383271e-01
 -2.06774905e-01 -2.27684334e-01 -3.39814901e-01  5.76493703e-02
  7.49362350e-01 -2.32862487e-01 -5.62201552e-02 -2.53030598e-01
  1.99317724e-01 -2.14907274e-01 -6.57019794e-01  2.35250592e-02
 -7.10008144e-01 -3.74690443e-01  5.33717200e-02  1.44490167e-01
  3.80775511e-01 -1.16112180e-01 -1.98471501e-01 -8.15009996e-02
 -4.85139340e-01  2.81295776e-02  2.77968466e-01 -5.70377827e-01
 -3.29397559e-01  3.54064226e-01  6.64371848e-02  5.64452648e-01
 -2.53516406e-01  5.39948523e-01 -6.51434958e-02 -4.51897323e-01
 -3.05233300e-01 -4.59913433e-01 -6.84654713e-02  2.69153388e-03
  3.27103555e-01  1.51303485e-01 -1.21298328e-01 -3.36090714e-01
 -4.83732879e-01  2.24577770e-01 -3.26014370e-01 -4.50170115e-02
  3.96486282e-01 -3.65934968e-01  2.29305699e-01  9.22877565e-02
 -5.48429824e-02 -7.19027400e-01  1.22388795e-01  4.99580018e-02
  2.03850076e-01 -1.74665649e-03 -6.15881145e-01  2.02150017e-01
 -4.48896945e-01  1.99548885e-01 -3.34170580e-01  5.95486403e-01
  9.28457975e-02 -6.76300466e-01  8.34489167e-01  3.22255827e-02
 -1.32272050e-01  1.27083987e-01 -4.34984833e-01  5.77331670e-02
  2.08956033e-01  1.61211550e-01  5.22046268e-01 -1.30651116e-01
  4.60883021e-01  8.16494077e-02  4.93989646e-01  1.65723816e-01
  3.44992787e-01 -1.00315899e-01 -6.17311120e-01  5.98827720e-01
```

```
-1.08771428e-01 -9.65223670e-01  8.50656927e-01 -2.95058191e-01
 5.46456039e-01 -1.33648977e-01 -4.88326652e-03  4.02789474e-01
-1.07322484e-02  2.97062576e-01 -5.55291951e-01 -4.28529114e-01
 2.59927183e-01  8.07762086e-01  5.55154026e-01  3.74313653e-01
-2.96862841e-01  5.10337174e-01  5.60580432e-01 -1.19672254e-01
 6.00529253e-01  1.77555174e-01  2.68931031e-01 -1.67640746e-02
 2.42678095e-02  7.16721043e-02  5.81104279e-01 -1.84345663e-01
-3.54693770e-01 -2.58687083e-02 -5.08208871e-01  4.84709114e-01
-2.03549504e-01 -2.72275746e-01  3.27258587e-01  6.01545691e-01
 1.55964181e-01 -5.24784803e-01 -1.74654052e-01 -6.43637419e-01
-1.51754953e-02 -3.41613293e-01  1.98907137e-01  2.45910004e-01
 5.60909927e-01 -1.24447778e-01 -6.73812330e-01 -3.15159746e-02
 4.95574564e-01 -9.14602131e-02  6.15336299e-01 -8.38087022e-01
-3.83393288e-01 -1.05536424e-01  1.70029774e-01 -3.21162909e-01
-2.65948445e-01  1.47736803e-01  6.62794054e-01  3.03202152e-01
-6.15772754e-02  1.67539343e-01 -6.32475376e-01  5.52159548e-01
-8.37674141e-02  3.59309882e-01 -6.18261211e-02 -2.22689718e-01
 5.14163196e-01  5.83682418e-01  2.89020449e-01  2.58361429e-01
 4.62115437e-01 -2.47972354e-01 -8.43182981e-01  2.70439595e-01
-1.21998368e-02 -6.97058678e-01 -6.10297620e-01 -3.57511878e-01
-3.01906705e-01  3.28770697e-01 -2.20689133e-01 -3.01161349e-01
 1.37058854e-01  7.69706309e-01  5.34628749e-01 -2.11039320e-01
 2.81650156e-01 -5.78221500e-01 -5.67640424e-01 -9.39728767e-02
-5.35979152e-01  2.30143711e-01  3.67872953e-01 -9.00316715e-01
 4.83984828e-01 -4.91745859e-01 -6.17337488e-02  1.98387697e-01
-3.44219744e-01 -1.27484649e-03  2.01181561e-01 -1.04943141e-01
-7.16504097e-01 -3.37424576e-02 -5.17251194e-01 -9.53595042e-02
 2.59930789e-01 -3.37784737e-01  3.35788995e-01 -4.48472798e-01
-1.06492531e+00  3.28426287e-02  2.21173882e-01 -2.14307249e-01
-6.49056196e-01 -2.72456050e-01 -4.05343175e-01 -5.58132589e-01
 4.93515611e-01  9.81374308e-02 -3.02693635e-01 -1.81432486e-01
-4.79328901e-01 -5.39891422e-01  1.63197517e-02 -2.96862662e-01
-7.27285445e-01 -3.15663256e-02  3.01897526e-01 -8.14774215e-01
-2.02014089e-01  2.65434980e-01  1.17220327e-01 -3.78246903e-01
-2.75616109e-01  4.83017594e-01  6.85966462e-02 -3.08753759e-01
 1.83665246e-01 -4.46762861e-04 -7.04556108e-01 -3.04780714e-02
-1.38077393e-01 -5.35294831e-01 -9.94290411e-02  5.92000663e-01
 8.12375769e-02  8.26339349e-02  2.72214979e-01  9.14443806e-02
 2.29964554e-01  2.78085917e-01 -3.00542861e-01  3.35615575e-01
 4.98034954e-01  6.33791924e-01 -6.53109372e-01 -2.64770538e-01
-2.91193742e-02  3.26866060e-01 -1.82360262e-01 -8.85515332e-01
-2.40380406e-01  2.10291401e-01 -3.32346139e-03  7.46673718e-02
 4.75635648e-01  5.51923886e-02 -1.60784602e-01  8.54162499e-02
-1.74284235e-01  2.31493175e-01  4.76191312e-01  5.81296673e-03
 5.79107031e-02 -3.69462162e-01  1.45074159e-01 -2.45889034e-02
 2.39252560e-02 -5.75524643e-02 -2.74673343e-01  7.13665336e-02
-3.11227500e-01 -2.60104556e-02 -5.77045977e-01  2.02265367e-01
 1.16648383e-01  5.75569332e-01  6.27159849e-02  1.29331917e-01
-1.43592715e-01 -3.08875497e-02 -4.58984897e-02  6.71775401e-01
-5.16596716e-03 -4.69771594e-01  4.32941988e-02 -1.11748360e-01]
```

In [2]:

```
#word2vec_model=Word2Vec.load("./word2vec/word2vec_model")
```

In [5]:

```
def create_length_dataframe(df):
    # Create a new DataFrame with the required columns
    length_df = pd.DataFrame({
        'reference': df['reference'],
        'len(text)': df['text'].apply(lambda x: len(str(x).split())),
        'len(preprocessed_text)': df['preprocessed_text'].apply(lambda x: len(str(x).split()
    })

    return length_df

df_length=create_length_dataframe(df_prep)
```

The texts are finally seperated into 5000 word chunks in order to improve the quality of the the topic modeling, a common practice recommended in the literature.

In [5]:

```
def chunk_text(text, chunk_size):
    words = text.split()  # Split text into words
    for i in range(0, len(words), chunk_size):
        yield ' '.join(words[i:i + chunk_size])

chunks_data = []

# Iterate over the DataFrame and chunk the text
for index, row in df_prep.iterrows():
    text_chunks = list(chunk_text(row['preprocessed_text'], 5000))
    for i, chunk in enumerate(text_chunks, 1):
        chunks_data.append({
            'preprocessed_text': chunk,
            'reference': f"{row['reference']}_{i}"
        })

# Create the new DataFrame from the list of data
df_chunk = pd.DataFrame(chunks_data)
```

In [6]:

```
df_chunk.to_csv('./preprocessing/results/df_books_chunk.csv', index=False)
```