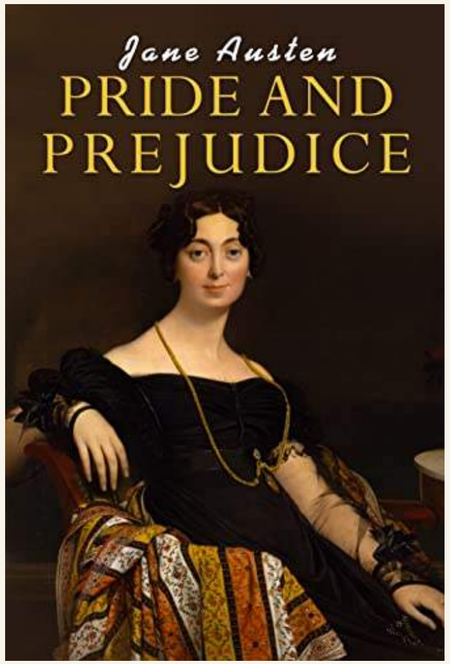
# Literary Landscapes

## Through the Lens of Text Mining

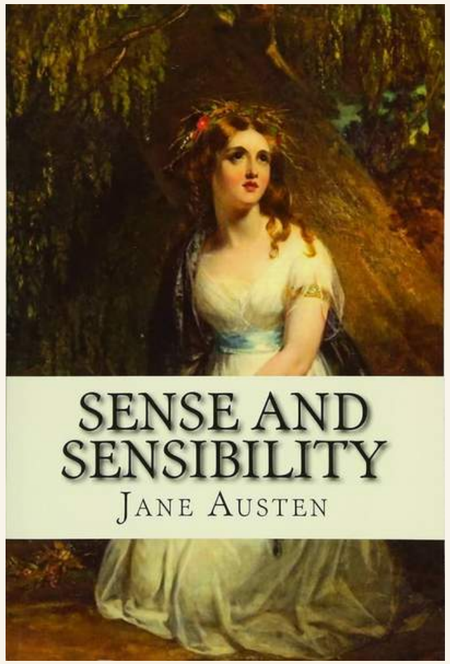
# Table of contents

# Documents




“Pride and Prejudice”

by Jane Austen




“Sense and Sensibility”

by Jane Austen

# Text Cleaning Steps

Step 1: Extracting the relevant text from the novel

Step 2: Removing any extra white spaces

Step 3: Removing non-alphabetical characters

Step 4: Lowercasing all characters

Step 5: Removing stop words

# Text Cleaning

## Step 1: Extracting the relevant text from the novel

```
The Project Gutenberg eBook of Pride and Prejudice

This ebook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this ebook or online
at www.gutenberg.org. If you are not located in the United States,
you will have to check the law of the country where you are located
before using this eBook.

Title: Pride and Prejudice


Author: Jane Austen

Release date: June 1, 1998 [eBook #1342]
                Most recently updated: April 14, 2023

Language: English
```

```
pp_start = re.search("It is a truth universally acknowledged", pp)
pp_end = re.search("had been the means of uniting them.", pp)
pp_content = pp[pp_start.start():pp_end.end()]
print ('Start of content:', pp_content[:50])
print ('End of content:',pp_content[-50:])
```

✓ 0.0s

```
Start of content: It is a truth universally acknowledged, that a sin
End of content: to Derbyshire, had been the means of uniting them.
```

# Text Cleaning

## Step 2: Removing any extra white spaces

```
pp_content[95:200]
```

✓ 0.0s

```
'be in want of a wife.\n\nHowever little known the feelings or views of such a man may be on his\nfirst enter'
```

```
pp_content = re.sub('\s+', ' ', pp_content).strip()
pp_content[95:200]
```

✓ 0.0s

```
'be in want of a wife. However little known the feelings or views of such a man may be on his first enteri'
```

# Text Cleaning

## Step 3: Removing non-alphabetical characters

```
pp_content[5120:5210]
```

✓ 0.0s

```
'h,-- “I hope Mr. Bingley will like it, Lizzy.” “We are not in a way to know _what_ Mr. Bingley likes'
```

```
pp_content = re.sub(r'[^a-zA-Z\s]', '', pp_content)
pp_content[4825:4930]
```

✓ 0.0s

```
' with I hope Mr Bingley will like it Lizzy We are not in a way to know what Mr Bingley likes said her
```

*only retain alphabetical characters

# Text Cleaning

## Step 4: Lowercasing all characters

```
pp_content[380:450]
```

✓ 0.0s

```
' Mr Bennet said his lady to him one day have you heard that Netherfiel'
```

```
pp_content = pp_content.lower()
pp_content[380:450]
```

✓ 0.0s

```
' mr bennet said his lady to him one day have you heard that netherfiel'
```

# Text Cleaning

## Step 5: Removing stop words

```
pp_content[:100]
```

✓ 0.0s

```
'it is a truth universally acknowledged that a single man in possession of a good fortune must be in '
```

```
stop_words = set(stopwords.words('english'))
```

✓ 0.0s

```
pp_content_tokens = pp_content.split() # split the text into words for stopwords removal
pp_content_tokens = [word for word in pp_content_tokens if word not in stop_words] # remove stopwords
pp_content = ' '.join(pp_content_tokens) # join the words back into a single string
pp_content[:100]
```

✓ 0.0s

```
'truth universally acknowledged single man possession good fortune must want wife however little know'
```

# Text Cleaning

Compare number of unique words before and after clean

```
print (f"pandp before cleaning: {count_unique_words(pp)}, after cleaning: {count_unique_words(pp_content)}")
print (f"sands before cleaning: {count_unique_words(ss)}, after cleaning: {count_unique_words(ss_content)}")
```

✓ 0.0s

```
pandp before cleaning: 14705, after cleaning: 6603
sands before cleaning: 14247, after cleaning: 6985
```

# Tokenization

```
pp_content[:100]
```

✓ 0.0s

```
'truth universally acknowledged single man possession good fortune must want wife however little
```

```
pandp_content_tokens = word_tokenize(pp_content)
pandp_content_tokens
```

✓ 0.0s

```
['truth',
 'universally',
 'acknowledged',
 'single',
 'man',
 'possession',
 'good',
 'fortune',
 'must',
 'want',
 'wife',
 'however',
 'little',
```

Using nltk's word tokenizer to tokenize each novel

# Text Exploration

## Word Clouds

Allows us to visualize the most dominant words in each novel.

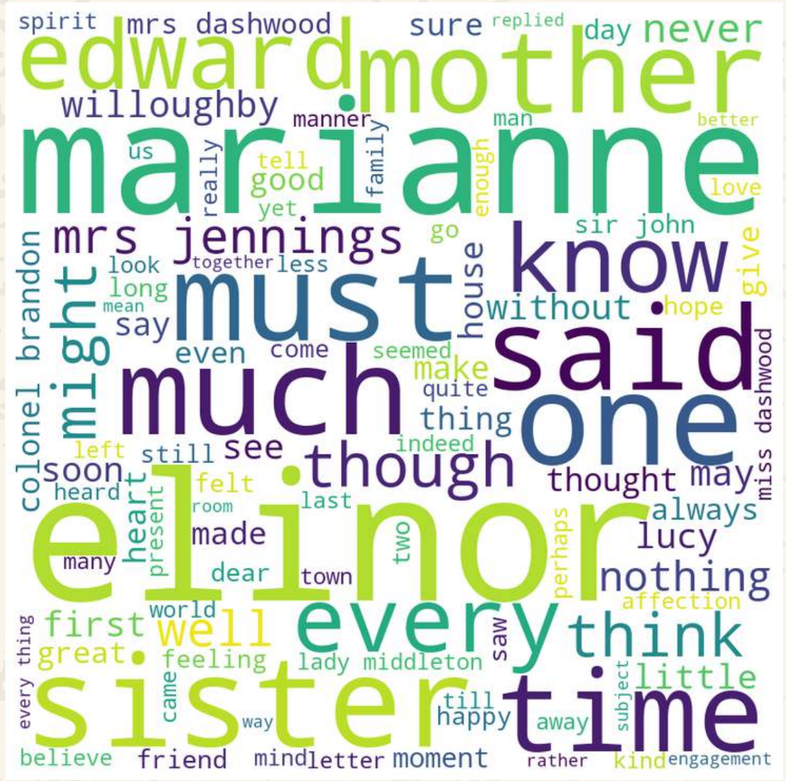
## Dispersion Plot

Shows us the distribution of a word throughout the document.

# Text Exploration

## Word Clouds

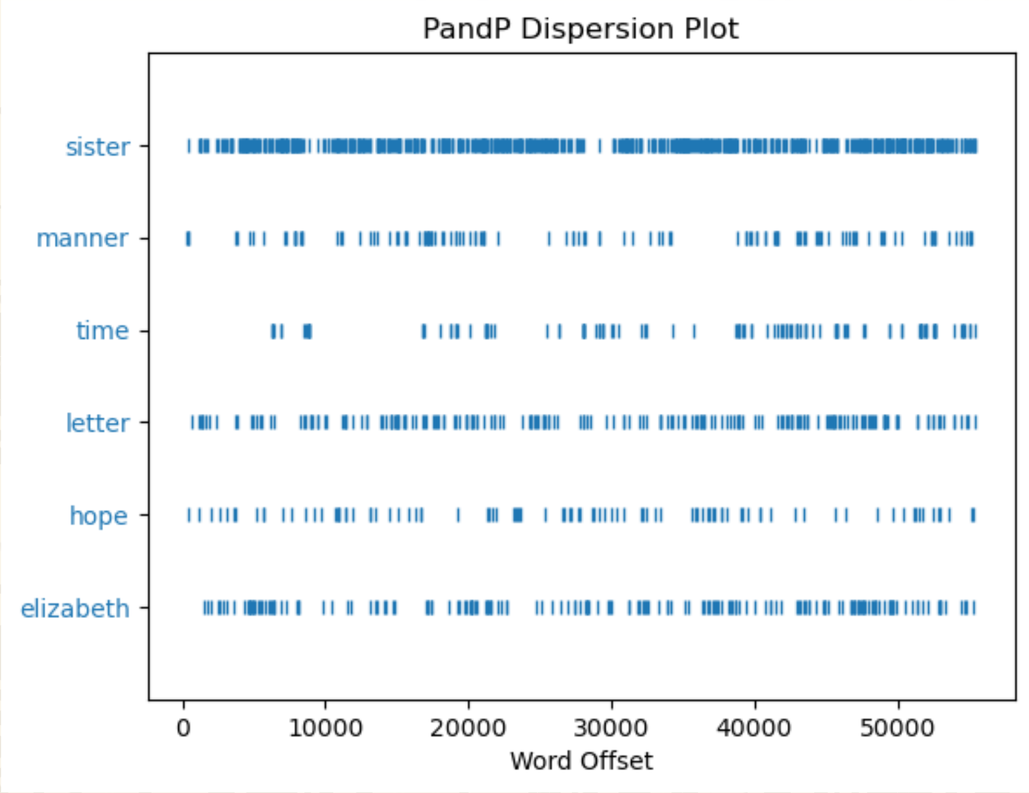

"Pride and Prejudice"



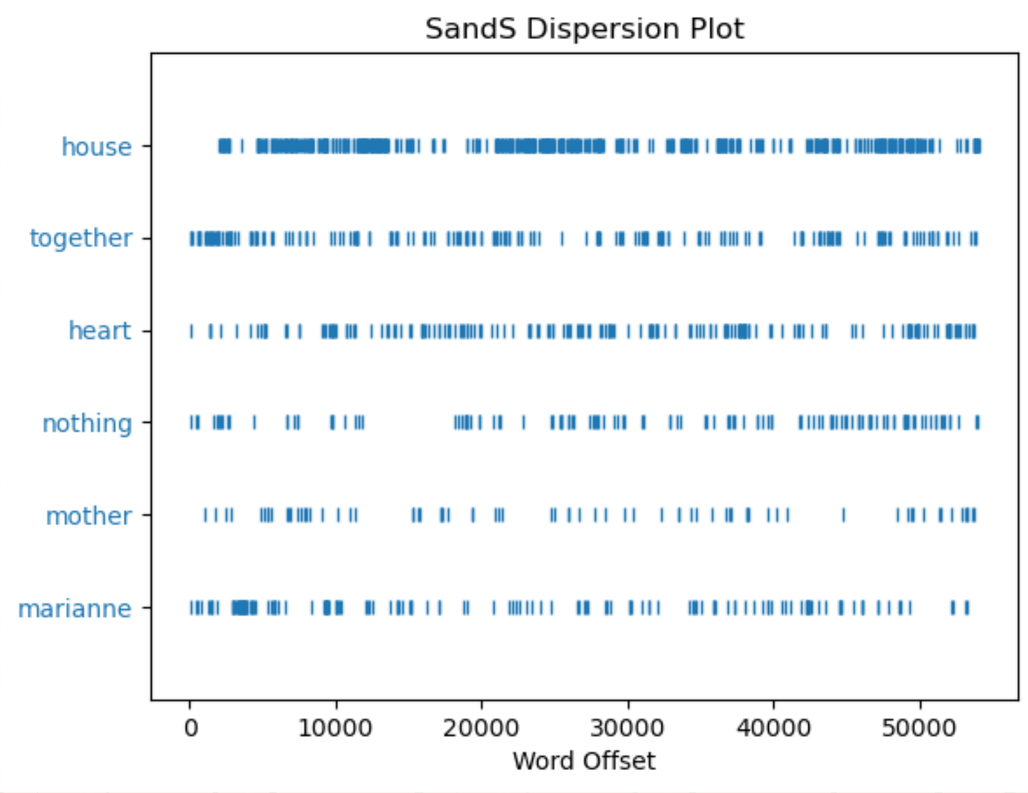

"Sense and Sensibility"

# Text Exploration

## Dispersion Plot




“Pride and Prejudice”




“Sense and Sensibility”

# Feature Engineering

## POS and Lemmatization

To standardize each word to its root form.

## N-grams (Bi-grams)

To retain the sequential order by concatenating two consecutive words.

# Feature Engineering

## POS and Lemmatization

```
pp_content_tokens[15:20]
```

✓ 0.0s

```
['views', 'man', 'may', 'first', 'entering']
```

```
pos_map = {
    'J': wordnet.ADJ,
    'V': wordnet.VERB,
    'N': wordnet.NOUN,
    'R': wordnet.ADV
}
tagged_tokens = pos_tag(pp_content_tokens)

lemmatizer = WordNetLemmatizer()
lemmatized_pp_content_tokens = [
    lemmatizer.lemmatize(word, pos=pos_map.get(tag[0], wordnet.NOUN)) for word, tag in tagged_tokens
]
lemmatized_pp_content_tokens[15:20]
```

✓ 2.2s

```
['view', 'man', 'may', 'first', 'enter']
```

# Feature Engineering

## N-grams (Bi-grams)

```
pp_content_tokens[:7]
```

✓ 0.0s

```
['truth', 'universally', 'acknowledged', 'single', 'man', 'possession', 'good']
```

```
n = 2 # bi-grams
pp_ngrams = []

for i in range(len(pp_content_tokens) - n + 1):
    # select n words starting from the current index
    ngram = pp_content_tokens[i:i+n]

    # join the words to create a single n-gram string
    ngram_string = '-'.join(ngram)

    pp_ngrams.append(ngram_string)

pp_ngrams[:3]
```

✓ 0.0s

```
['truth-universally', 'universally-acknowledged', 'acknowledged-single']
```

# Vectorization

We have 02 versions of each document.

We vectorize each version of each book with TF-IDF from sklearn.

## TD-IDF for Lemmatization (A)

```
tfidf = TfidfVectorizer()
tfidfmatrix_feature_a = tfidf.fit_transform([
    ' '.join(lemmatized_pp_content_tokens),
    ' '.join(lemmatized_ss_content_tokens)
])
tfidfmatrix_feature_a.shape
```

✓ 0.0s

```
(2, 7698)
```

## TD-IDF for N-grams (B)

```
tfidf = TfidfVectorizer()
tfidfmatrix_feature_b = tfidf.fit_transform([
    ' '.join(pp_ngrams),
    ' '.join(ss_ngrams)
])
tfidfmatrix_feature_b.shape
```

✓ 0.0s

```
(2, 9390)
```

# Comparison

The cosine similarity between two features A (lemmatization) is 0.70.

```
consine_sim_feature_a = cosine_similarity(tfidfmatrix_feature_a[0], tfidfmatrix_feature_a[1])
consine_sim_feature_a
```

✓ 0.0s

```
array([[0.69651372]])
```

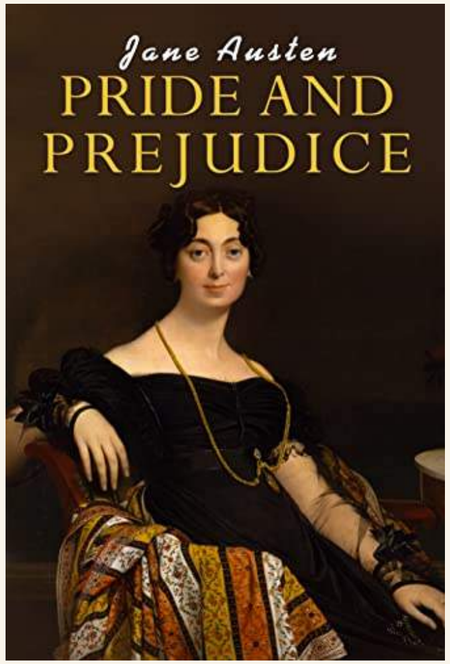
The cosine similarity between two features B (Bi-grams) is 0.63.

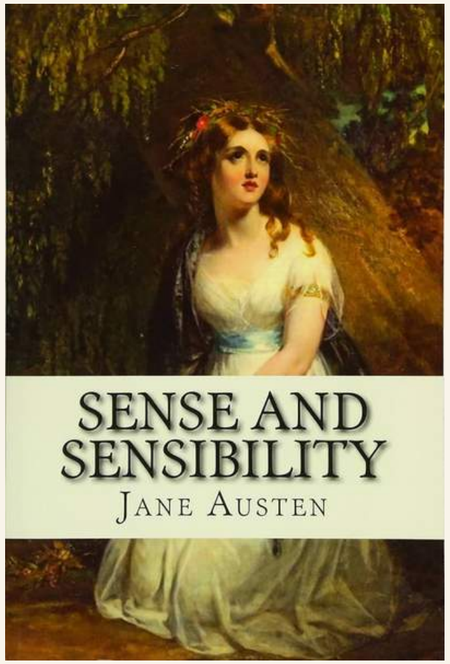
```
consine_sim_feature_b = cosine_similarity(tfidfmatrix_feature_b[0], tfidfmatrix_feature_b[1])
consine_sim_feature_b
```

✓ 0.0s

```
array([[0.62867521]])
```

# Insights




- High similarity between 02 novels (0.70 & 0.63).
- Shared vocabulary and themes: reflective of Austen's focus on morality, courtship, and the social standing of women.
- Shows consistency in Austen's writing styles across works.

# Insights

## Lemmatization

- Focus on the meaning of words.
- More effective for understanding thematic similarities (shared themes, vocabulary, and overarching narrative elements).

Main themes and ideas

## N-grams

- Take into account the context in which words appear.
- Provide richer insights about the narrative style, sentence construction, and the specific use of language.

Writing style

# Thank you!