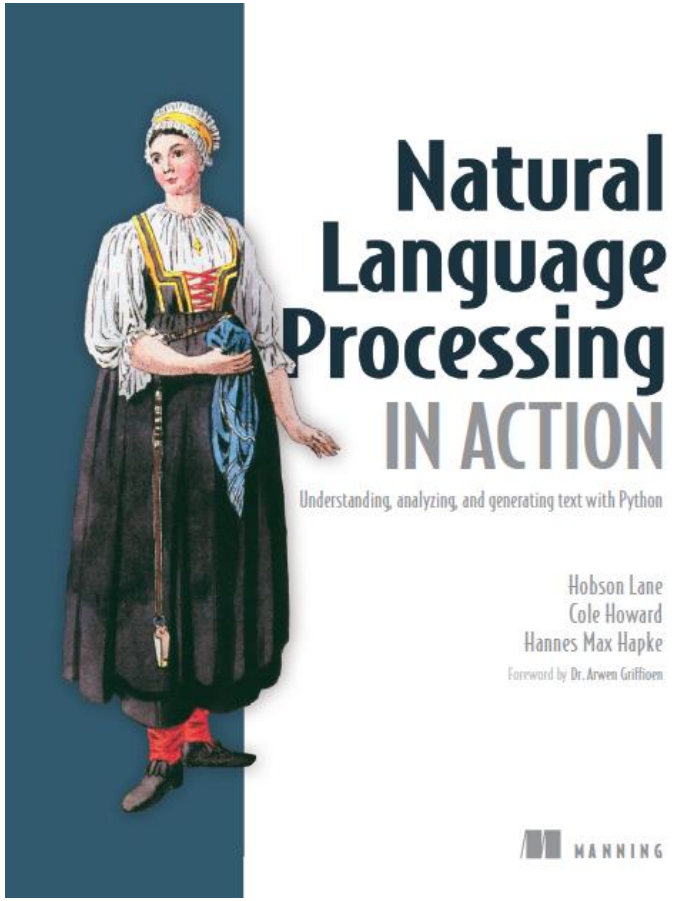
Word Tokenization

Prof. Gheith Abandah

Reference 1

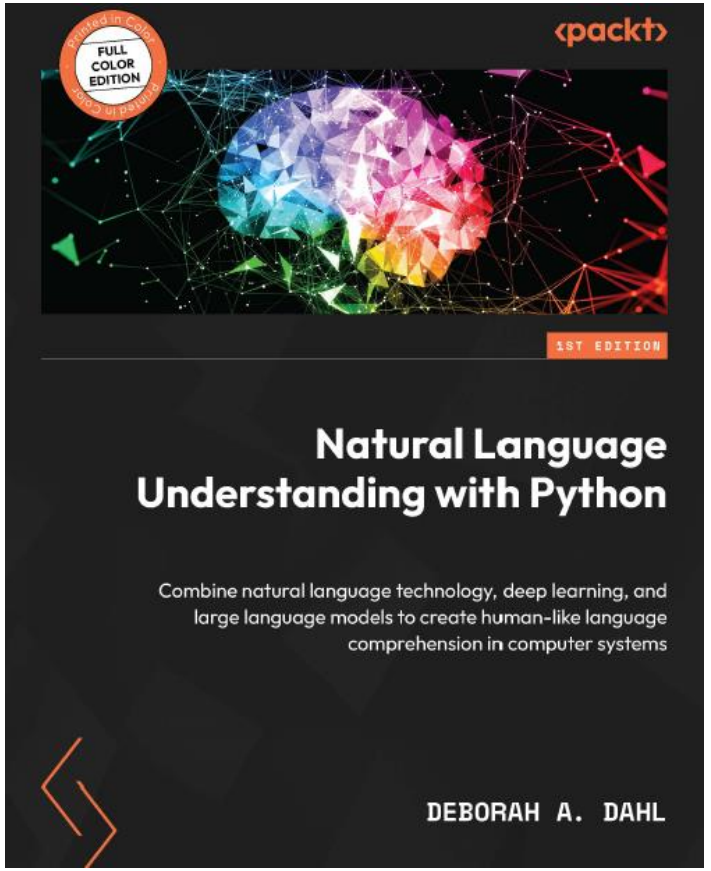
- Chapter 2: **Build your vocabulary (word tokenization)**



- H. Lane, C. Howard, and H. Hapke, **Natural Language Processing in Action:** Understanding, analyzing, and generating text with Python, Manning, 2019.

Reference 2

- Chapter 4: **Selecting Libraries and Tools for Natural Language Understanding**



- Deborah Dahl, **Natural Language Understanding with Python**, Packt, 2023.

Outline

- NLP Libraries
- Analysis of Movie Reviews Dataset
- Preprocessing Text
- Bag of Words
- Summary

NLP Libraries

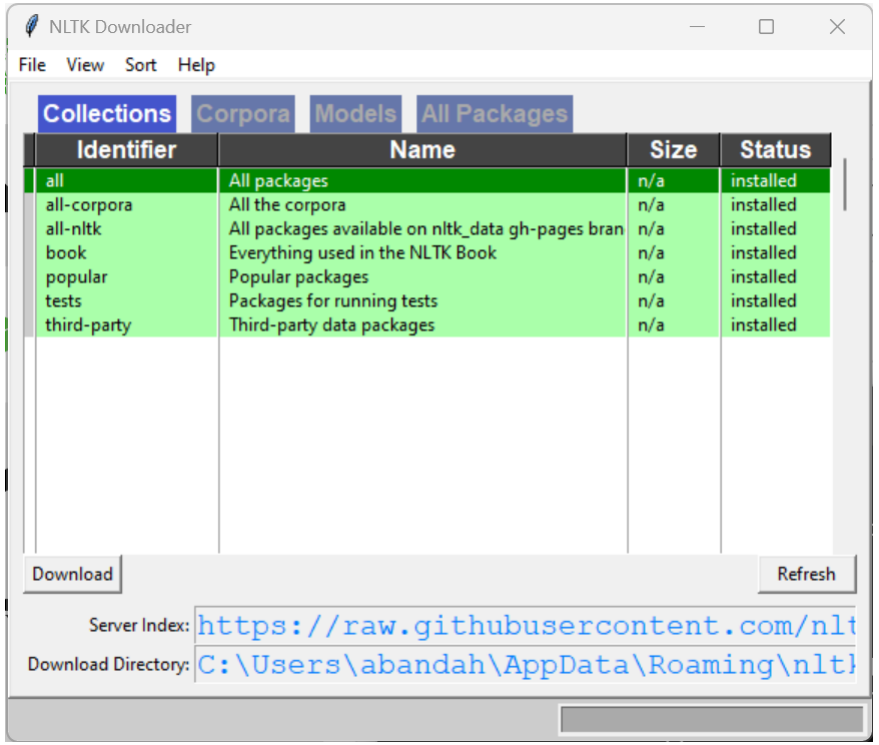
- Natural Language Toolkit (NLTK)
- spaCy
- TensorFlow/Keras
- PyTorch
- scikit-learn
- Gensim
- WorldCloud
- PyArabic, Farasa, camel-tools

NLTK

- **NLTK**: Python's leading platform for Natural Language Processing (NLP).
- **Features**: Text processing, classification, tokenization, stemming, tagging.
- **Resources**: Over 50 corpora and lexical databases, e.g., WordNet.
- **Language Support**: Extensive for English; tools available for other languages, including Arabic.
- **Applications**: Text analysis, sentiment analysis, linguistic research.
- **Audience**: Ideal for beginners and researchers.
- **Getting Started**: `pip install nltk`, followed by `nltk.download()` to fetch data.

Tokenization with NLTK

- **Tokenization**: breaking text into words




```
import nltk

nltk.download() # Run this once

from nltk import word_tokenize
text = "we'd like to book a flight from boston to London"
tokenized_text = word_tokenize(text)
print(tokenized_text)
```

```
['we', "'d", 'like', 'to', 'book', 'a', 'flight',
 'from', 'boston', 'to', 'London']
```

Word Frequency with NLTK

```
from nltk.probability import FreqDist
freq_dist = FreqDist(tokenized_text)
for word, frequency in freq_dist.items():
    print(f"{word}: {frequency}")
```

```
we: 1
'd: 1
like: 1
to: 2
book: 1
…
```

Part-of-speech (POS) Tagging with NLTK

```
pos = nltk.pos_tag(tokenized_text)
print(pos)
```

```
[('we', 'PRP'), ("'d", 'MD'), ('like', 'VB'), ('to', 'TO'),
 ('book', 'NN'), ('a', 'DT'), ('flight', 'NN'),
 ('from', 'IN'), ('boston', 'NN'), ('to', 'TO'),
 ('London', 'NNP')]
```

Tag Glossary

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

spaCy

- A powerful, open-source NLP library designed for production use.
- **Key Features**: Efficient text processing, easy-to-use API, pre-trained models for multiple languages.
- **Use Cases**: Tokenization, part-of-speech tagging, named entity recognition (NER), dependency parsing, and more.
- **Performance**: Optimized for speed and accuracy, with support for multi-threading and GPU.
- **Integration**: Compatible with deep learning frameworks like TensorFlow and PyTorch for advanced NLP tasks.
- **Getting Started**: Install with `pip install spacy` and easily download models with `spacy download [model name]`.

Tokenization with spaCy

```
import spacy

# Need to run:
# python -m spacy download en_core_web_sm
nlp = spacy.load('en_core_web_sm')
text = "we'd like to book a flight from boston to london"
doc = nlp(text)
words = [token.text for token in doc]
print (words)
```

```
['we', "'d", 'like', 'to', 'book', 'a', 'flight', 'from',
 'boston', 'to', 'london']
```

Word Frequency with spaCy

```
from collections import Counter
word_freq = Counter(words)
print(word_freq)
```

```
Counter({'to': 2, 'we': 1, "'d": 1, 'like': 1, 'book': 1,
         'a': 1, 'flight': 1, 'from': 1, 'boston': 1,
         'london': 1})
```

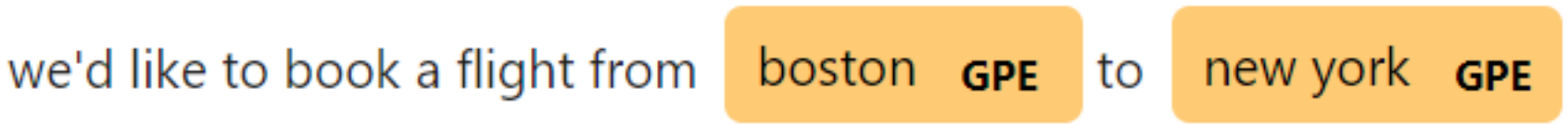
Part-of-speech (POS) Tagging with spaCy

```
for token in doc:
    print(token.text, token.pos_)
```

```
we PRON
'd AUX
like VERB
to PART
book VERB
a DET
flight NOUN
from ADP
boston PROPN
to ADP
london PROPN
```

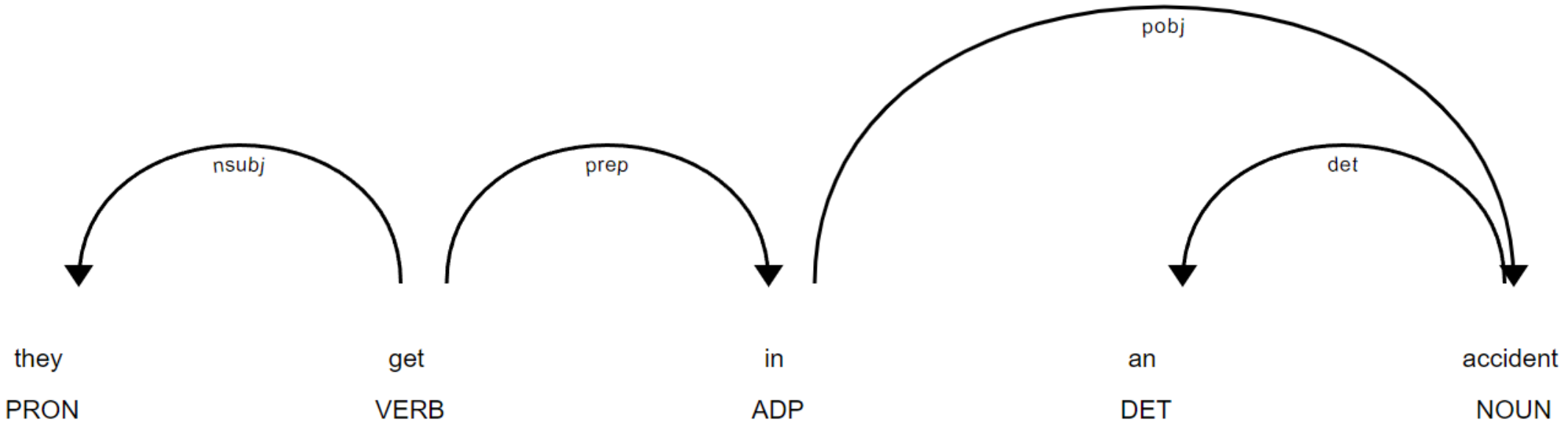
Named Entity Recognition with spaCy

```
# Run on a Jupyter Notebook

import spacy
from spacy import displacy
nlp = spacy.load("en_core_web_sm")
text = "we'd like to book a flight from boston to new york"
doc = nlp(text)
displacy.render(doc, style='ent', jupyter=True,
                options={'distance':200})
```



Syntactic Relationships Analysis with spaCy

```
doc = nlp('they get in an accident')
displacy.render(doc, style='dep', jupyter=True,
                options={'distance':200})
```



Keras

- A high-level **deep learning** API, written in Python, running on top of TensorFlow, designed for human beings, not machines.
- **Why Keras for NLP?**: Provides simple and flexible tools for building and training complex models, including sequence-to-sequence, attention, and more.
- **Core Features**: Supports recurrent layers like LSTM and GRU, making it perfect for handling text data and sequence analysis.
- **Ease of Use**: Simplifies tasks such as tokenization, embedding, and sequence padding with built-in support.
- **Customization and Scalability**: Allows for easy customization of models and is scalable to large datasets and complex model architectures.
- **Real-World Applications**: Widely used in sentiment analysis, language translation, text summarization, and more.
- **Getting Started**: Install with `pip install tensorflow`.

Outline

- NLP Libraries
- Analysis of Movie Reviews Dataset
- Preprocessing Text
- Bag of Words
- Summary

Analysis of Movie Reviews Dataset

```
# NLP imports
import nltk

# general numerical and visualization imports
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter

#nltk.download()
```

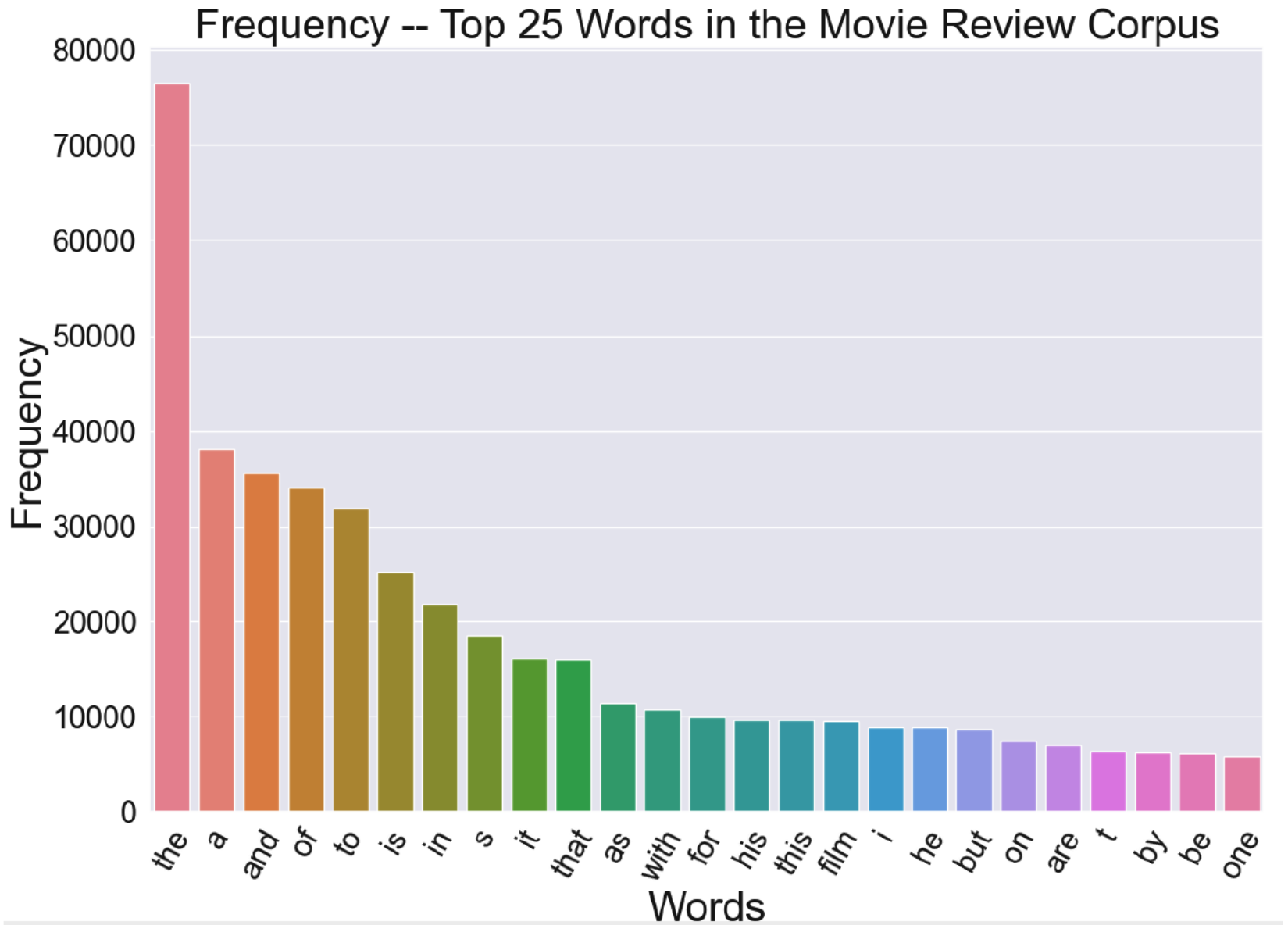
Import the training data

```
from nltk.corpus import movie_reviews
sents = movie_reviews.sents()
print(sents)
sample = sents[9]
print(sample)
```

```
[['plot', ':', 'two', 'teen', 'couples', 'go', 'to', 'a', 'church’,
  'party', ',', 'drink', 'and', 'then', 'drive', '.’],
 ['they', 'get', 'into', 'an', 'accident', '.'], ...]

 ['they', 'seem', 'to', 'have', 'taken', 'this', 'pretty', 'neat’,
  'concept', ',', 'but', 'executed', 'it', 'terribly', '.']
```

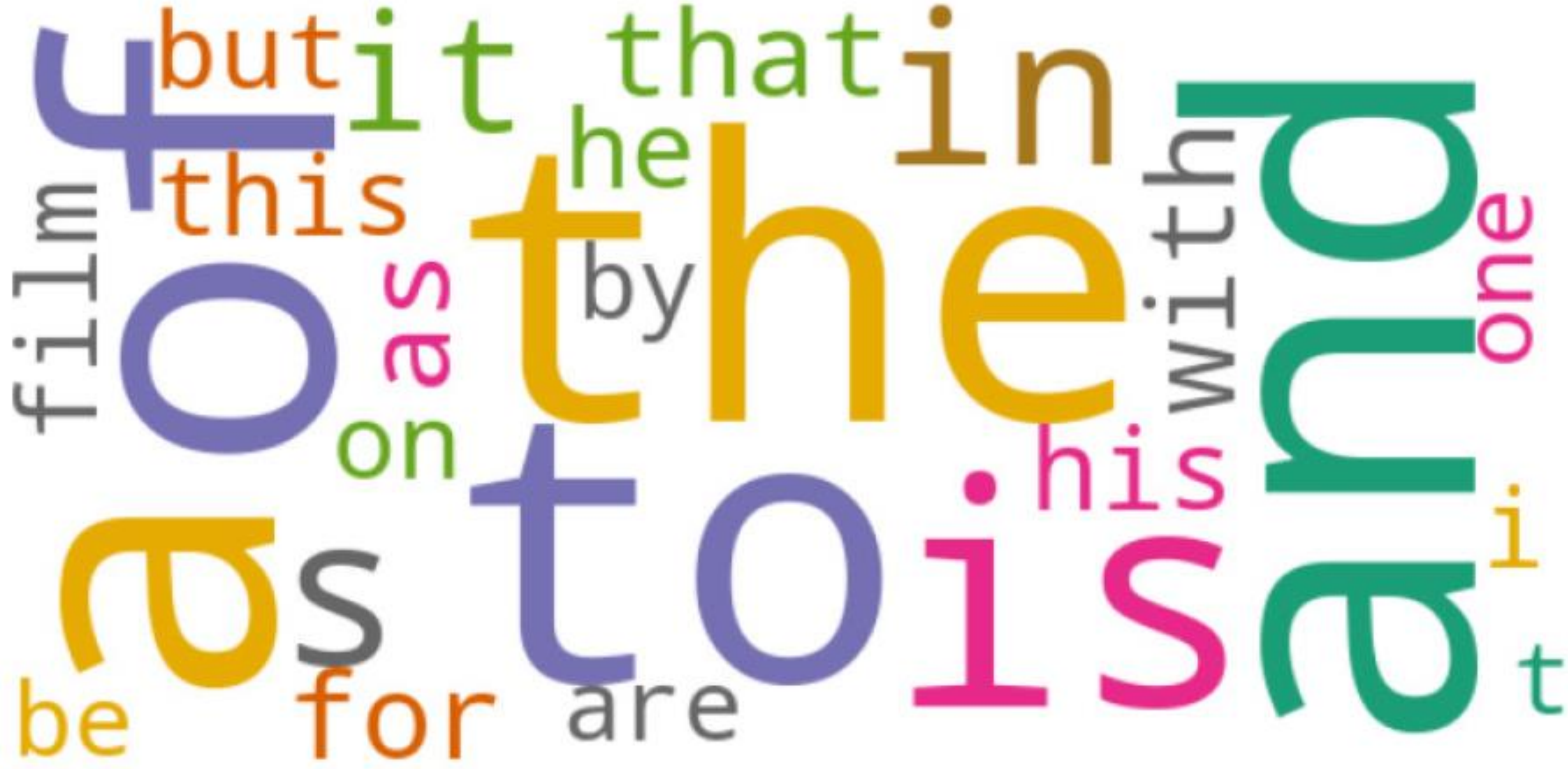
Word Frequencies

```
words = movie_reviews.words()
word_counts = nltk.FreqDist(word.lower() for word in
                            words if word.isalpha())
top_words = word_counts.most_common(25)
all_fdist = pd.Series(dict(top_words))
# Setting fig and ax into variables
fig, ax = plt.subplots(figsize=(10,10))
# Plot with Seaborn plotting tools
…
all_plot = sns.barplot(x = all_fdist.index,
                       y = all_fdist.values, ax=ax)
plt.show()
```


Frequency -- Top 25 Words in the Movie Review Corpus
Frequency
80000
70000
60000
50000
40000
30000
20000
10000
0
the
a
and
of
to
is
in
s
it
that
as
with
for
his
this
film
i
he
but
on
are
t
by
be
one
Words

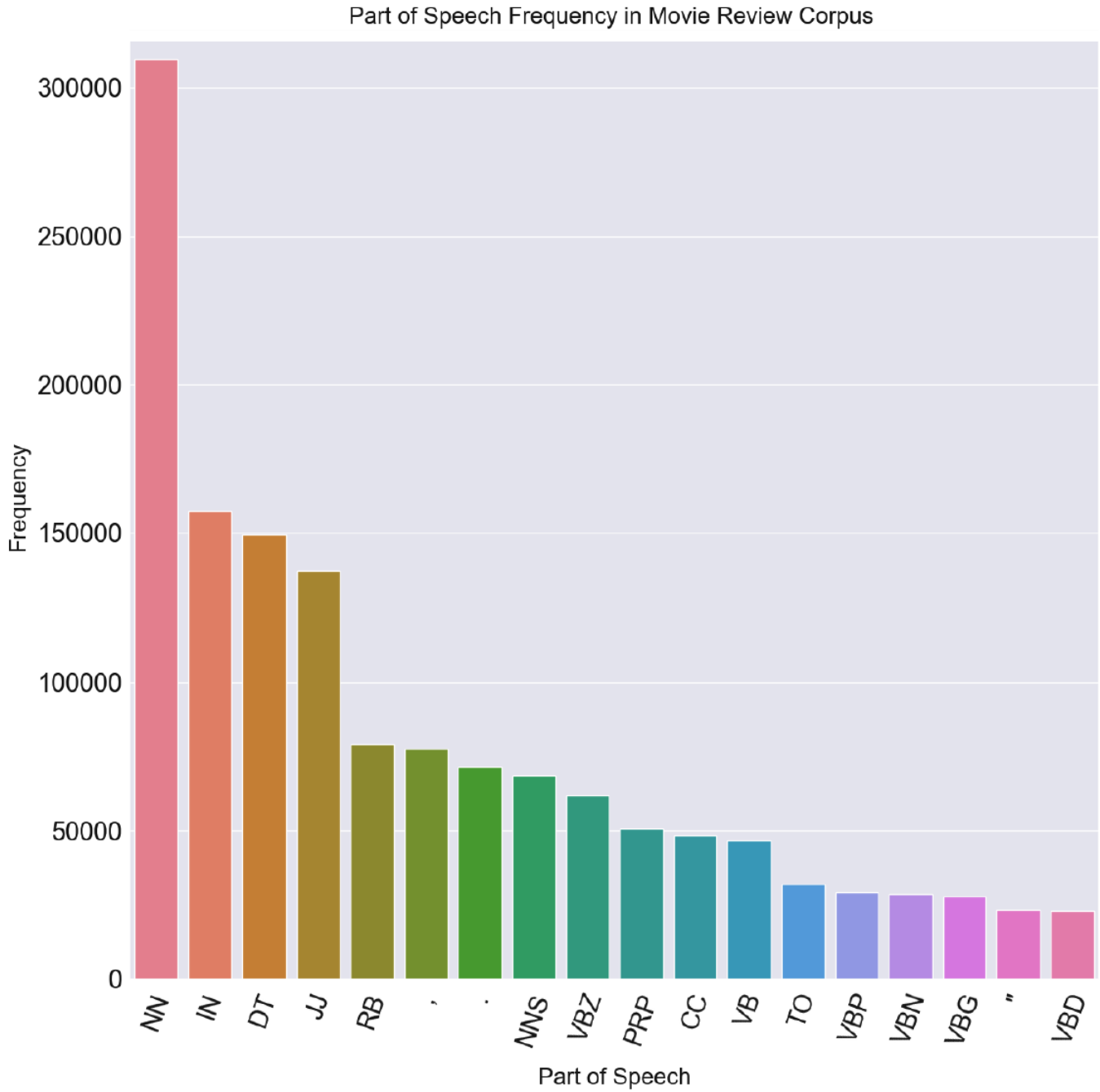
Word Cloud

```
from wordcloud import WordCloud
wordcloud =\
    WordCloud(background_color='white',
              max_words=25,
              relative_scaling=0,
              width=600, height=300,
              max_font_size=150,
              colormap='Dark2', min_font_size=10).
                generate_from_frequencies(all_fdist)

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```


of
but
it
that
in
and
the
he
this
film
with
one
by
as
to
on
his
a
s
is
i
t
be
for
are

POS Frequencies

```
tagged_sents = nltk.pos_tag_sents(sents)
total_counts = {}
for sentence in tagged_sents:
    counts = Counter(tag for word, tag in sentence)
    total_counts = Counter(total_counts) + Counter(counts)

sorted_tag_list = sorted(total_counts.items(), key = lambda x:
                         x[1],reverse = True)
all_tags = pd.DataFrame(sorted_tag_list)
most_common_tags = all_tags.head(18)
# Setting figure and ax into variables
fig, ax = plt.subplots(figsize=(15,15))
all_plot = sns.barplot(x = most_common_tags[0],
                       y = most_common_tags[1], ax = ax)
…
plt.show()
```


Part of Speech Frequency in Movie Review Corpus
Frequency
300000
250000
200000
150000
100000
50000
0
NN
IN
DT
JJ
RB
,
.
NNS
VBZ
PRP
CC
VB
TO
VBP
VBN
VBG
"
VBD
Part of Speech

Outline

- NLP Libraries
- Analysis of Movie Reviews Dataset
- Preprocessing Text
- Bag of Words
- Summary

Preprocessing Text

- Removing emojis
- Removing smart quotes
- Lower casing
- Lemmatization
- Stopword removal
- Removing punctuation

Removing emojis

```
$ pip install demoji
#replacing emojis with their intepretation
import demoji
text = "Happy birthday!🎂"

emojis_replaced = demoji.replace_with_desc(text)
print(emojis_replaced)

no_emojis = demoji.replace(text,"")
print(no_emojis)

Happy birthday!:birthday cake:
Happy birthday!
```

Removing Smart Quotes

```
text = "here is a string with “smart” quotes"
text = text.replace("“", "\"").replace("”","\"")
print(text)
```

```
here is a string with "smart" quotes
```

Lower Casing

```
import nltk
mixed_text = "WALK! Going for a walk is great."
mixed_words = nltk.word_tokenize(mixed_text)
print(mixed_words)

lower_words = []
for mixed_word in mixed_words:
    lower_words.append(mixed_word.lower())
print(lower_words)
```

```
['WALK', '!', 'Going', 'for', 'a', 'walk', 'is', 'great', '.']
['walk', '!', 'going', 'for', 'a', 'walk', 'is', 'great', '.']
```

Lemmatization (Word Root)

```
# lemmatizing with WordNet
import nltk
nltk.download("wordnet")
from nltk.stem.wordnet import WordNetLemmatizer
from nltk import word_tokenize, pos_tag
from nltk.corpus import wordnet
from collections import defaultdict

# align names for POS between WordNet and POS tagger.
tag_map = defaultdict(lambda: wordnet.NOUN)
tag_map["J"] = wordnet.ADJ
tag_map["V"] = wordnet.VERB
tag_map["R"] = wordnet.ADJ

lemmatizer = WordNetLemmatizer()
text_to_lemmatize = "going for a walk is the best exercise. i've
walked every evening this week"
```

Lemmatization (Word Root)

```
tokens = nltk.word_tokenize(text_to_lemmatize)
lemmatized_result = ""
for token, tag in pos_tag(tokens):
    lemma = lemmatizer.lemmatize(token, tag_map[tag[0]])
    lemmatized_result = lemmatized_result + " " + lemma

print(lemmatized_result)
```

```
go for a walk be the best exercise . i 've walk every
evening this week
```

Stopword Removal

- **Stopwords**: Common words that are not helpful in distinguishing documents and so they are often removed.

```
import spacy
nlp = spacy.load("en_core_web_sm")

stop_words = nlp.Defaults.stop_words
print(len(stop_words))

text = "This is a sample sentence demonstrating the removal of
stopwords using spaCy."

doc = nlp(text)

filtered_tokens = [token.text for token in doc if not token.is_stop]
filtered_text = " ".join(filtered_tokens)
print(filtered_text)
```

```
326
sample sentence demonstrating removal stopwords spaCy .
```

Removing Punctuation

```
import spacy
nlp = spacy.load("en_core_web_sm")

text = "Hello, world! This is an example sentence; let's see
how it works."
doc = nlp(text)

filtered_tokens = [token.text for token in doc if
                   not token.is_punct]
filtered_text = ' '.join(filtered_tokens)
print(filtered_text)
```

```
Hello world This is an example sentence let 's see how it
works
```

Outline

- NLP Libraries
- Analysis of Movie Reviews Dataset
- Preprocessing Text
- Bag of Words
- Summary

Bag-of-Words

- **Definition**: A simple yet powerful feature extraction technique used in NLP for text analysis.
- **Functionality**: Transforms text into fixed-length vectors by counting how many times each word appears.
- **Application**: Essential for tasks like document classification, sentiment analysis, and topic modeling.
- **Advantages**: Easy to understand and implement, making it perfect for beginners in NLP.
- **Limitations**: Ignores the order of words, resulting in potential loss of contextual meaning.
- **Variations**: Includes binary Bag-of-Words (presence/absence of words) and TF-IDF (Term Frequency-Inverse Document Frequency) for weighting word importance.

Bag-of-Words (bag for each sentence)

```
import pandas as pd

sentences = "The school is large.\n" \
            "His school is my school too.\n" \
            "The student goes to school."
corpus = dict()
for i, sent in enumerate(sentences.split('\n')):
     corpus['sent{}'.format(i)] = dict((tok, 1)
                                       for tok in sent.split())

df = pd.DataFrame.from_records(corpus).fillna(0).astype(int).T
print(df)
```

```
       The  school  is  large.  His  my  too.  student  goes  to  school.
sent0    1       1   1       1    0   0     0        0     0   0        0
sent1    0       1   1       0    1   1     1        0     0   0        0
sent2    1       0   0       0    0   0     0        1     1   1        1
```

Dot Product Similarity Measure

```
print(df)
df = df.T
print(df.sent0.dot(df.sent1))
print(df.sent0.dot(df.sent2))
print(df.sent1.dot(df.sent2))
```

```
       The  school  is  large.  His  my  too.  student  goes  to  school.
sent0    1       1   1       1    0   0     0        0     0   0        0
sent1    0       1   1       0    1   1     1        0     0   0        0
sent2    1       0   0       0    0   0     0        1     1   1        1
2
1
0
```

Example: Similarity using Bag-of-Words

```
import spacy
from collections import Counter
import numpy as np
nlp = spacy.load("en_core_web_sm")

sentence1 = "The quick brown fox jumps over the lazy dog."
sentence2 = "A quick brown dog outpaces a lazy fox."

def preprocess(text):
    doc = nlp(text.lower())
    clean_tokens = [token.text for token in doc if
                    not token.is_punct and not token.is_stop]
    return clean_tokens

tokens1 = preprocess(sentence1)
tokens2 = preprocess(sentence2)
print(tokens1, '\n', tokens2)
```

```
['quick', 'brown', 'fox', 'jumps', 'lazy', 'dog']
['quick', 'brown', 'dog', 'outpaces', 'lazy', 'fox']
```

Example: Similarity using Bag-of-Words

```
bow1 = Counter(tokens1)
bow2 = Counter(tokens2)

# Ensure vectors are in the same dimension
all_tokens = set(bow1.keys()).union(set(bow2.keys()))
vector1 = np.array([bow1.get(token, 0) for token in all_tokens])
vector2 = np.array([bow2.get(token, 0) for token in all_tokens])

# Compute the dot product for similarity
dot_product = np.dot(vector1, vector2)
norm_product = np.linalg.norm(vector1) * np.linalg.norm(vector2)
similarity = dot_product / norm_product if norm_product else 0

print(f"Similarity: {similarity}")
```

```
Similarity: 0.8333333333333335
```

Summary

- NLP Libraries
- Analysis of Movie Reviews Dataset
- Preprocessing Text
- Bag of Words
- Summary