

**Python Programming**

**Final Project Report**

**By Dong Jinghui**

**Politechnika Krakowska**

**2024/2025**

Project Title: Process text with natural language processing (NLP)

# CONTENTS

1. Abstract
2. Introduction
  - 2.1 Aim
  - 2.2 Experimental Environment
  - 2.3 Scope
  - 2.4 Methodology
3. Theoretical Part
  - 3.1 NLTK Extractive Summary
  - 3.2 Sumy Abstractive Summary
  - 3.3 Word Cloud Visualization
4. Practical Part
  - 4.1 Data Input
  - 4.2 Data Pre\_processing
  - 4.3 Analysis and Generation:
    - 4.3.1 Extractive Summary
    - 4.3.2 Abstractive Summary
    - 4.3.3 Word Cloud Visualization
    - 4.3.4 Keyword Extraction
  - 4.4 Result and Output
5. Improvement Suggestions
6. Conclusion
7. Bibliography

# 1.Abstract

This experiment demonstrates the application of natural language processing (NLP) techniques for text summarization and visualization using Python libraries. The core objectives include generating extractive summaries with NLTK based on word frequency, creating abstractive summaries with sumy using the LexRank algorithm, and visualizing high-frequency keywords via WordCloud. Through a comparative analysis of these methods, the study evaluates their performance. The results show that extractive summarization preserves original content but may lack flow, while abstractive summarization offers semantic coherence at the cost of potential detail loss. WordCloud effectively highlights key themes through visual mapping of word frequencies. The findings provide insights into tool selection for NLP tasks, emphasizing the trade-offs between accuracy, efficiency, and interpretability.

**Keywords:** Natural Language Processing; Extractive Summary; Abstractive Summary; LexRank; Word Cloud

## 2.Introduction

### 2.1 Aim

Aim: Use natural language processing (NLP) text, based on NLTK, sumy and word cloud for text summary and visualization

### 2.2 Experimental Environment

**Programming Language:** Python 3.10

**Libraries/Tools:**

- **nltk:** For natural language processing tasks (tokenization, stopword filtering, frequency statistics).
- **sumy:** For abstractive summarization using the LexRank algorithm.
- **wordcloud + matplotlib:** For keyword visualization.
- **google.colab.files:** For interactive text file uploads in Colab.
- **re:** For text cleaning via regular expressions.

**Experimental Text:** The Rise of Artificial Intelligence (English text).

### 2.3 Scope

**Data Input:** Handle text file uploads and manage encoding (UTF-8/GBK) to ensure content integrity.

**Preprocessing:** Cleanse text by removing noise (special characters, stopwords), standardizing case, and normalizing whitespace.

**Analysis and Generation:**

- **Extractive Summarization:** Extract key sentences using word frequency statistics via NLTK.
- **Abstractive Summarization:** Generate coherent summaries using the LexRank graph-based algorithm from sumy.
- **Keyword Extraction:** Identify top (high-frequency) words through frequency distribution analysis.
- **Word Cloud Visualization:** Convert high-frequency words into visual graphs using WordCloud.

**Result Output:** Present structured results (summaries, keywords, visualizations) for intuitive understanding of text content and themes.

2.4Methodology

- 1. **NLTK**:Develop extractive summarization skills using NLTK’s word frequency-based algorithm.
- 2. **Sumy**:Implement abstractive summarization with sumy’s LexRank algorithm to generate semantically summaries.
- 3. **WordCloud**:Utilize WordCloud and Matplotlib for visual representation of keyword distributions.
- 4. Compare the applicability and effectiveness of NLTK, sumy, and WordCloud across different text analysis scenarios.

3.Theoretical Part

Indicator	NLTK Extractive Summary	Sumy Abstractive Summary	Word Cloud Visualization
Core Principle	Word frequency statistics + sentence weighting	LexRank algorithm (sentence similarity graph model)	High-frequency word mapping (font size proportional to frequency)
Code Complexity	High (requires manual implementation of tokenization, scoring, and sorting)	Low (built-in algorithm, only library functions needed)	Medium (requires parameter configuration for stopwords, colors, and dimensions)
Output Characteristics	Directly extracts original sentences (may be fragmented or out of order)	Generates coherent new sentences (integrates semantics but may simplify details)	Visually highlights keyword distribution, focusing on core themes
Suitable Scenarios	Academic literature key sentence extraction, news brief summarization	Long-text summarization (e.g., legal contracts, meeting records)	Creative report covers, social media visualization, keyword analysis
Typical Output Example	Retains original sentences like "AI marks a pivotal moment..."	Generates summary sentences like "AI systems are capable of..."	Word cloud with largest fonts for "artificial", "transformative", etc.
Advantages and Disadvantages	<b>Advantages:</b> Preserves technical details; <b>Disadvantages:</b> Poor coherence	<b>Advantages:</b> Fluent language; <b>Disadvantages:</b> May lose details	<b>Advantages:</b> Intuitive and easy to understand; <b>Disadvantages:</b> Cannot display complete semantics

## 4. Practical Part

### 4.1 Data Input

```
#Import necessary libraries
from google.colab import files
import nltk
import re
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lex_rank import LexRankSummarizer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

```
# Download NLTK resources
nltk.download('punkt_tab')
nltk.download('stopwords')
```

#### Library Import and Resource Download

**Function:** Import all necessary Python libraries and tools for the experiment, including text processing, summarization, and visualization.

**Details:**

- Import google.colab.files for uploading files in the Colab environment.
- Import nltk and its related modules for natural language processing (tokenization, stopword filtering, word frequency statistics).
- Import the summarization algorithm (LexRank) from the sumy library for abstractive summarization.
- Import wordcloud and matplotlib for word cloud visualization.
- Download NLTK's punkt tokenizer model and stopwords corpus.

```

print("Please upload a text file (.txt format):")
uploaded = files.upload()

# Get the uploaded file name
filename = next(iter(uploaded.keys()))
try:
    text = uploaded[filename].decode('utf-8') # Try UTF-8 decoding first
except UnicodeDecodeError:
    text = uploaded[filename].decode('gbk') # Try GBK decoding if needed
print(f"Successfully uploaded file: {filename}")

```

## File Upload and Encoding Handling

**Function:** Enable users to upload text files and handle different encoding formats (UTF-8/GBK) to avoid garbled text.

**Details:**

- Prompt the user to upload a .txt file and retrieve the file content through an interactive interface.
- Automatically attempt to decode the file using UTF-8; if it fails, use GBK instead.
- Output the successfully uploaded file name to confirm correct file reading.

## 4.2 Data Pre\_processing

convert to lowercase, remove special characters, and normalize whitespace

```

def clean_text(text):
    """Clean text: convert to lowercase, remove special characters, and normalize whitespace"""
    # Convert to lowercase
    text = text.lower()
    # Remove special characters and punctuation (keep spaces)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Normalize whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    return text

cleaned_text = clean_text(text)

```

## Text Cleaning

**Function:** Preprocess the raw text to remove noise and improve subsequent analysis results.

**Details:**

- Convert the text to lowercase to standardize the format (e.g., treat "AI" and "ai" as the same word).
- Use regular expressions to remove special characters, punctuation, and numbers, retaining only letters and spaces.
- Compress consecutive spaces into a single space and trim leading/trailing whitespace to ensure clean text.

## 4.3 Analysis and Generation:

### 4.3.1 Extractive Summary

```
def generate_extractive_summary(text, num_sentences=3):
    """Generate extractive summary using NLTK word frequency statistics"""
    sentences = sent_tokenize(text)

    if len(sentences) <= num_sentences:
        return text

    cleaned_text = clean_text(text)
    words = word_tokenize(cleaned_text)

    stop_words = set(stopwords.words('english'))
    filtered_words = [word for word in words if word not in stop_words]

    freq_dist = FreqDist(filtered_words)

    sentence_scores = {sent: sum(freq_dist[word] for word in word_tokenize(sent.lower()) if word in freq_dist)
                       for sent in sentences}

    sorted_sentences = sorted(sentence_scores.items(), key=lambda x: x[1], reverse=True)
    selected_sentences = [sent[0] for sent in sorted_sentences[:num_sentences]]

    ordered_summary = [sent for sent in sentences if sent in selected_sentences]
    return ' '.join(ordered_summary)
```

#### Extractive Summarization (NLTK Word Frequency Statistics)

**Function:** Extract key sentences from the original text based on word frequency statistics to generate an extractive summary.

**Details:**

- Split the text into sentences; return the original text directly if the number of sentences is insufficient.
- Clean the text, tokenize it, filter out stopwords, and then count word frequencies.
- Score each sentence (sum of frequencies of high-frequency words), sort them by score, extract key sentences, and maintain the original order to ensure logical coherence.

### 4.3.2 Abstractive Summary

```
def generate_abstractive_summary(text, num_sentences=3):
    """Generate abstractive summary using sumy's LexRank algorithm"""

    parser = PlaintextParser.from_string(text, Tokenizer("english"))
    summarizer = LexRankSummarizer()
    summary = summarizer(parser.document, num_sentences)

    return " ".join([str(sentence) for sentence in summary])
```

#### Abstractive Summarization (sumy LexRank Algorithm)



**Function:** Use the LexRank graph model algorithm to generate an abstractive summary by calculating sentence similarity to produce a coherent summary.

**Details:**

- Use sumy's parser to convert the text into an internal document object.
- Calculate sentence importance based on the LexRank algorithm, automatically select key sentences, and generate a coherent summary without directly copying sentences from the original text.

### 4.3.3 Word Cloud Visualization

```
def generate_wordcloud(text, max_words=200):  
    """Generate Word Cloud Visualization"""  
    stop_words = set(stopwords.words('english'))  
    stop_words.update(['said', 'could', 'would', 'also', 'one', 'two', 'three', 'may', 'now'])  
  
    wc = WordCloud(  
        stopwords=stop_words,  
        background_color='white',  
        max_words=max_words,  
        width=800,  
        height=400,  
        colormap='viridis').generate(text)  
  
    plt.figure(figsize=(10, 6))  
    plt.imshow(wc, interpolation='bilinear')  
    plt.axis('off')  
    plt.title('Word Cloud Visualization (High-Frequency Words)')  
    plt.tight_layout()  
    plt.show()
```

### Word Cloud Visualization

**Function:** Convert high-frequency words into a visual word cloud to intuitively display text themes.

**Details:**

- Extend custom stopwords (e.g., "said", "would") to filter more meaningless words.
- Configure word cloud parameters (maximum number of words, color scheme, size), generate the word cloud, and display it using matplotlib with axes turned off and a title added.

### 4.3.4 Keyword Extraction

```
def extract_keywords(text, top_n=10):
    """Extract keywords from text"""
    # Clean text and tokenize
    cleaned_text = clean_text(text)
    words = word_tokenize(cleaned_text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    filtered_words = [word for word in words if word not in stop_words]

    # Calculate word frequency
    freq_dist = FreqDist(filtered_words)

    # Return the top_n most frequent words
    return freq_dist.most_common(top_n)
```

### Keyword Extraction (Based on Word Frequency)

**Function:** Extract high-frequency keywords from the cleaned text to assist in quickly understanding the core content of the text.

**Details:**

- Clean the text again, tokenize it, filter out stopwords, and then count word frequencies.
- Return the top N words with the highest frequencies as the core keywords of the text.

## 4.4 Result and Output

```

print(f"\nFile: {filename}")
print(f"Text Length: {len(text)} characters")
print(f"Number of Sentences: {len(sent_tokenize(text))}")

print("\n----- Text Preview -----")
preview = text[:300] + "..." if len(text) > 300 else text
print(preview)

print("\n----- NLTK Extractive Summary -----")
summary_nltk = generate_extractive_summary(text, num_sentences=3)
print(summary_nltk)

print("\n----- Sumy Abstractive Summary -----")
summary_sumy = generate_abstractive_summary(text, num_sentences=3)
print(summary_sumy)

print("\n----- Keyword Extraction -----")
keywords = extract_keywords(text, top_n=10)
for word, freq in keywords:
    print(f"{word}: {freq} occurrences")

print("\n----- Word Cloud Visualization -----")
generate_wordcloud(cleaned_text)

```

## Results Display

**Function:** Integrate and output all analysis results, including file information, summaries, keywords, and word cloud visualization.

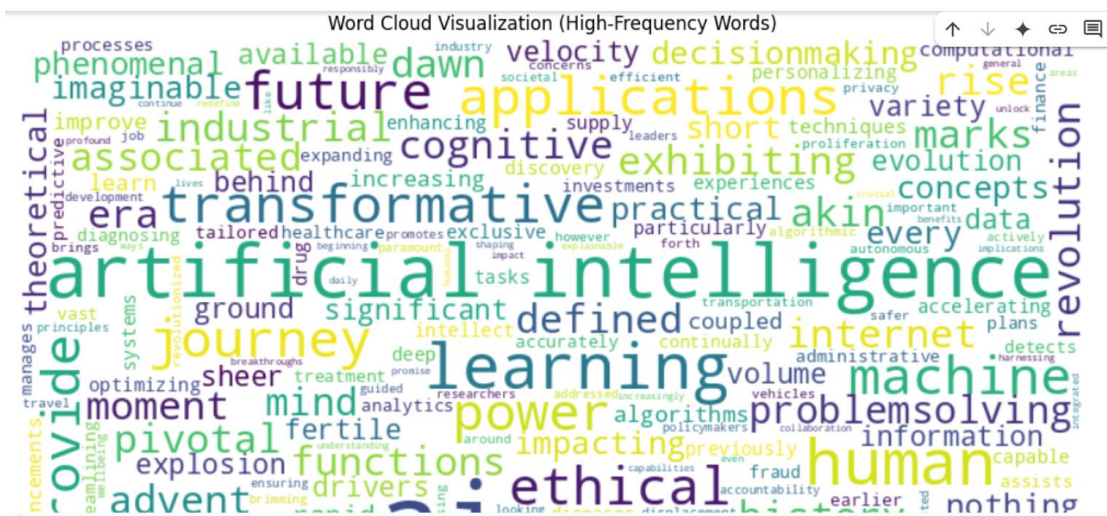
### Details:

- Output file name, text length, number of sentences, and other metadata.
- Display a text preview, both types of summaries (extractive and abstractive), and the keyword list.
- Call the word cloud generation function to visualize the distribution of high-frequency words, providing an intuitive result display.

The results are as follows:

----- Keyword Extraction -----

```
ai: 11 occurrences
artificial: 4 occurrences
intelligence: 4 occurrences
human: 3 occurrences
learning: 3 occurrences
transformative: 2 occurrences
journey: 2 occurrences
applications: 2 occurrences
sector: 2 occurrences
power: 2 occurrences
```



## Results analysis

### Keyword extraction result analysis

**High-frequency word core:** "ai" appears 11 times, which is the most frequent keyword; "artificial" and "intelligence" both appear 4 times. Combined, it shows that the core theme of this paper revolves around artificial intelligence (Artificial Intelligence).

### Word cloud result analysis

**The core theme is highlighted:** in the word cloud, the larger the font, the higher the frequency of occurrence. "artificial intelligence" has the largest and most eye-catching font in the word cloud, which once again makes it clear that the theme is artificial intelligence.

More detailed run results can be viewed in the code.

## 8. Improvement Suggestions

**(1)Abstract algorithm upgrade:** Replace with a more advanced BERT abstract model (such as BartForConditionalGeneration in transformers library) to improve the semantic accuracy of abstract summary.

**(2)Distributed processing:** For very large texts (such as millions of words), use Dask or Spark for distributed word segmentation and statistics to improve the running efficiency.

**(3)Advanced visualization:** Try dynamic word clouds (ipywordcloud library) or 3D word clouds (python-3d-wordcloud library) to enhance interactivity.

**(4)Tune parameters:** The number of summary sentences can be dynamically adjusted according to the text length. Extended stop words, customized word cloud shape and color, etc., can optimize the efficiency and quality of NLP processing.

## 9. Conclusion

1. NLTK is suitable for learning the underlying logic of NLP (such as word frequency statistics and stop word filtering), but the core algorithm needs to be implemented manually, which is suitable for education and customized scenarios.
2. Sumy's graph-based summary algorithm is more efficient and generates coherent results, which is suitable for processing long texts such as legal documents and meeting records.
3. Word cloud visualization can quickly convey the theme of the text, and combined with custom shapes (such as person head image, icon) can enhance the visual impact, suitable for report cover or data analysis conclusion display.
4. Tool collaboration: NLTK pre\_processes the text → sumy, generates the summary → wordcloud visualizes the keywords, forming a complete text analysis link.

## 10. Bibliography

Lee Vaughan\_Real\_World Python\_A Hacker's Guide to Solving Problems with Code\_No Starch Press(2020)