### A Major Project Report

**on**

**NEWS ARTICLE TEXT SUMMARIZATION**

**SIDDHARTHA INSTITUTE OF TECHNOLOGY & SCIENCES**

**(UGC – AUTONOMOUS)**

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with ‘A+’ Grade.

### Narapally, Korremula Road, Ghatkesar, Medchal- Malkajgiri (Dist.)- 500088

### A blue gear with a blue and white logo Description automatically generated

(Submitted in partial fulfilment of the academic requirements of B. Tech)

In

### Department Of Computer Science and Engineering (CSD)

By

### M. Ramya Sri(20TQ1A6709)

**J. Gayathri(20TQ1A6717)**

### Ch. Roopa(20TQ1A6725)

**E. Nishanth Reddy(20TQ1A6704)**

**N. Ashish Yadav(20TQ1A6716)**

Under the Esteemed Guidance of

**Dr. A. Satyanarayana**

**SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES**

**(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NAAC(A+))**

Korremula Road, Narapally(V), Ghatkesar Mandal, Medchal-Dist:- 500088

##### CERTIFICATE

This is to certify that the project report entitled

**NEWS ARTICLE TEXT SUMMARIZATION**

being submitted

by

##### M. Ramya Sri(20TQ1A6709)

**J. Gayathri(20TQ1A6717)**

##### Ch. Roopa(20TQ1A6725)

**E. Nishanth Reddy(20TQ1A6704**)

**N. Ashish Yadav(20TQ1A6716)**

In partial fulfilment for the award of the degree of Bachelor of Technology in Computer Science and Engineering, Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma

###### Guide Head of the department

**Dr. A. Satyanarayana Mr. Satya Krishna**

Department of CSD Department of CSD.

###### Internal Examiner External Examiner

**DECLARATION**

We declare that this project report titled **NEWS ARTICLE TEXT SUMMARIZATION** submitted in partial fulfilment of the degree of **B. Tech in Computer Science and Engineering (CSD)** is a record of our original work carried out by us under the guidance and supervision of **Dr. A. Satyanarayana** and has not formed the basis for the award of any other degree or diploma, in this or any other Institute or University. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited. Guidance and supervision

##### M. Ramya Sri(20TQ1A6709)

**J. Gayathri(20TQ1A6717)**

##### Ch. Roopa(20TQ1A6725)

**E. Nishanth Reddy(20TQ1A6704**)

**N. Ashish Yadav(20TQ1A6716)**

**ACKNOWLEDGMENT**

Any endeavor in the field of development is a person’s intensive activity. A successful project is a fruitful culmination of efforts by many people, some directly involved and some others who have quietly encouraged and supported.

Salutation to be beloved and highly esteemed institute **SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES** for grooming us into Computer Science and Engineering graduate, We wish to thank **Principal Dr. M. Janardhan for** providing a great learning environment.

We wish to express profound gratitude to **Mr. Satya Krishna**, Associate Professor and **Head of Department,** Computer Science and Engineering (DS), for his continuous encouragement to ensure successful results in all my endeavors.

We would like to thank **Dr. A. Satyanarayana**, Department of Computer Science and Engineering, who patiently guided and helped us throughout our project.

We take this opportunity to thank the department’s Project Review Co- Ordinator **Dr. A. Satyanarayana** for all the review meetings, suggestions, and support throughout the project development.

### By

##### M. Ramya Sri(20TQ1A6709)

**J. Gayathri(20TQ1A6717)**

##### Ch. Roopa(20TQ1A6725)

**E. Nishanth Reddy(20TQ1A6704**)

**N. Ashish Yadav(20TQ1A6716)**

**SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES**

**(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NAAC(A+))**

Korremula Road, Narapally(V), Ghatkesar Mandal, Medchal-Dist:-500088

**Vision of the Department:** To be a Recognized Center of Computer Science Education with values and quality research.

**Mission of the Department:**

|  |  |
| --- | --- |
| **MISSION** | **STATEMENT** |
| **DM1** | Import High Quality Professional Training with An Emphasis on Basic principles Of Computer Science and Allied Engineering |
| **DM2** | Imbibe Social Awareness and Responsibility to Serve the Society. |
| **DM3** | Provide Academic Facilitates Organize Collaborated Activities To enable Overall Development of Stakeholders |

**Programme Educational Objectives (PEO)**

* **PEO1:** Graduates will be able to synthesize mathematics, science, engineering fundamentals, laboratory and work – based experiences to formulate and to solve problems proficiently in Computer science and Engineering and related domains.
* **PEO2:** Graduates will be prepared to communicate effectively and work in multidisciplinary engineering projects following the ethics in their profession.
* **PEO3:** Graduates will recognize the importance of and acquire the skill of independent learning to shine as experts in the field with a sound knowledge.

### TABLE OF CONTENTS

**DESCRIPTION PAGE NUMBER**

CERTIFICATE I

DECLARATION II

ACKNOWLEDGEMENTS III

ABSTRACT 7

[CHAPTER 1: INTRODUCTION 8-9](#_TOC_250010)

[CHAPTER 2: LITERATURE SURV](#_TOC_250008)EY 10-11

[CHAPTER 3: EXISTING SYSTEM](#_TOC_250007)  12-13

[CHAPTER 4: ARCHITECTURE 14-](#_TOC_250006)15

CHAPTER 5: SYSTEM REQUIREMENT SPECIFICATION 16-17

[CHAPTER 6: SYSTEM DESIGN](#_TOC_250004)  18-21

[CHAPTER 7: IMPLEMENTATION](#_TOC_250003) 22-32

[CHAPTER 8: SCREENSHOTS](#_TOC_250001) 33-35

[CHAPTER 9: CONCLUSION 36](#_TOC_250000)

CHAPTER 10: FUTURE SCOPE 37-38

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Name Of the Figure** | **Page No.** |
| 4.1 | Architecture | 14 |
| 6.1 | Use Case Diagram | 18 |
| 6.2 | ER Diagram | 19 |
| 6.3 | Sequence Diagram | 20 |
| 6.4 | Flow Chart Diagram | 21 |

**ABSTRACT**

News article summarization is a process designed to condense full-length articles into concise, informative summaries. This technique is essential in today's fast-paced world, where the volume of news content can be overwhelming. Summarization methods fall into two main categories: extractive and abstractive. Extractive summarization focuses on identifying and selecting key sentences from the original text, while abstractive summarization involves generating new phrases and sentences that encapsulate the core ideas.

Key steps in this process include preprocessing text data, scoring sentences for relevance, and using metrics like ROUGE scores for evaluation. The ultimate aim is to create fluent summaries that effectively capture the main points, events, and facts from news articles, enabling readers to quickly digest large amounts of information.

# 

1. **INTRODUCTION**

In today's fast-paced world, staying informed is essential, but the sheer volume of news articles can be overwhelming. Text summarization offers a solution by condensing lengthy articles into concise summaries, providing readers with key information efficiently. By leveraging natural language processing and machine learning techniques, text summarization algorithms can identify the most important points of an article, making it easier for individuals to grasp the main ideas without having to read the entire piece. This technology has numerous applications, from helping professionals stay updated in their fields to enabling quick news consumption for busy individuals. In this article, we will explore the significance of text summarization in the realm of news consumption, its underlying technologies, and its potential impact on how we consume information in the digital age.

Text summarization is revolutionizing the way we interact with information overload. With the exponential growth of online content, from news articles to research papers, users face a daunting task of sifting through vast amounts of text to extract relevant information. Text summarization algorithms streamline this process by automatically generating concise summaries, saving users time and effort. These algorithms employ advanced natural language processing techniques such as sentence extraction, abstraction, and semantic analysis to distill the essence of a text while preserving its key points.

One of the primary benefits of text summarization is its ability to cater to diverse reading preferences. Whether individuals prefer skimming through headlines or delving into in-depth analysis, summarization algorithms can adapt to their needs by generating summaries of varying lengths and complexities. This flexibility enhances user engagement and accessibility, enabling individuals with limited time or attention spans to stay informed without feeling overwhelmed.

Furthermore, text summarization has significant implications for professionals across various industries. In fields such as finance, healthcare, and law, where staying abreast of the latest developments is critical, summarization algorithms can provide timely insights and facilitate informed decision-making. By aggregating information from multiple sources and distilling it into concise summaries, these algorithms empower professionals to stay competitive in dynamic environments.

Moreover, text summarization contributes to the democratization of information by making complex topics more accessible to a wider audience. Whether it's breaking news, scientific discoveries, or policy updates, summarization algorithms enable individuals from diverse backgrounds and expertise levels to grasp the significance of complex topics without requiring specialized knowledge.

In addition to enhancing individual productivity and knowledge acquisition, text summarization has broader societal implications. By promoting efficient information consumption, these algorithms can mitigate the spread of misinformation and improve media literacy. By enabling users to quickly discern credible sources and identify key takeaways, summarization algorithms empower individuals to navigate the digital landscape with greater discernment and critical thinking skills.

# **LITERATURE SURVEY**

* A survey on automatic text summarization [ResearchGate]: This paper provides a general overview of automatic text summarization, including a section on news articles. It covers traditional approaches like extractive summarization and the use of statistical methods for sentence scoring.
* Text summarization in progress: A literature review [ResearchGate]: This survey delves into the advancements in text summarization research. It highlights various approaches used for news articles, including keyword extraction and sentence ranking techniques.
* Contrastive text summarization: a survey [SpringerLink]: This survey focuses on contrastive summarization, where the goal is to summarize the key differences between multiple related news articles. This is particularly useful for summarizing controversial topics or contrasting viewpoints.
* NewsIN: A News Summarizer and Analyzer [IJRASET]: This paper proposes a news summarization system specifically designed for online news articles. It explores techniques for sentence selection and highlights the importance of tailoring the summarization process for news content.
* Deep Learning Approaches: Some recent studies explore the application of deep learning techniques, such as recurrent neural networks (RNNs) and transformers, for text summarization. These models often achieve state-of-the-art performance by learning hierarchical representations of input text and generating concise summaries.
* Multimodal Summarization: With the increasing availability of multimedia content, there's a growing interest in multimodal summarization, which combines information from multiple modalities such as text, images, and videos to generate comprehensive summaries. Research in this area investigates fusion strategies and the integration of visual and textual features for summarization tasks.
* Evaluation Metrics: Evaluating the quality of automatic summaries is crucial for assessing the performance of summarization systems. Various evaluation metrics have been proposed, including ROUGE (Recall-Oriented Understudy for Gisting Evaluation), BLEU (Bilingual Evaluation Understudy), and METEOR (Metric for Evaluation of Translation with Explicit Ordering), each focusing on different aspects of summary quality such as content overlap and fluency.
* Domain-Specific Summarization: Text summarization techniques often need to be adapted to specific domains, such as biomedical literature, legal documents, or social media posts. Research in domain-specific summarization explores domain-specific features, terminology, and discourse patterns to improve the relevance and accuracy of summaries in specialized domains.
* Abstractive Summarization: While extractive summarization techniques select and concatenate existing sentences from the source text, abstractive summarization aims to generate novel sentences that capture the essential information of the input text in a more human-like manner. Recent advancements in natural language generation (NLG) techniques have led to significant progress in abstractive summarization methods, including the use of reinforcement learning and transformer-based architectures like GPT (Generative Pre-trained Transformer) models.

# 

**3. EXISTING SYSTEM**

**Manual Summarization**

**Traditional Approach:**

In the existing system, articles are summarized manually by editors and journalists, following a practice deeply ingrained in the history of media organizations. This manual process involves reading through the entirety of an article and condensing its key points into a shorter version.

**Drawbacks:**

Despite its historical precedence, manual summarization suffers from several drawbacks. Firstly, it is a time-consuming process, as it requires individuals to read through each article thoroughly and then craft a summary. This can be particularly problematic when dealing with large volumes of articles, leading to delays in the summarization process. Additionally, manual summarization is inherently subjective, as the quality and depth of the summary can vary depending on the individual summarizer's interpretation of the content. This subjectivity introduces inconsistency and may result in summaries that do not accurately reflect the original article's key points. Moreover, human error is a significant risk in manual summarization, as summarizers may inadvertently omit crucial information or misinterpret the article's content.

**Consistency Challenges:**

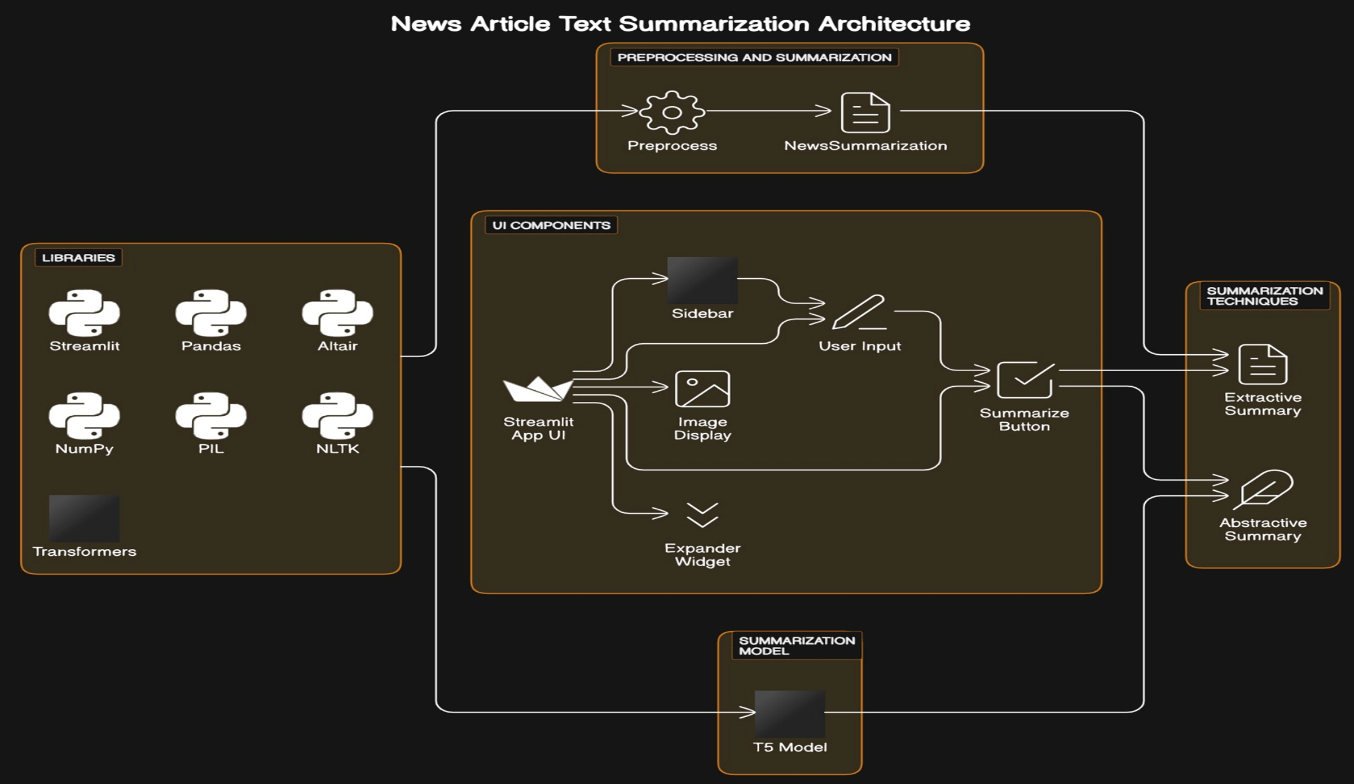
One of the primary challenges associated with manual summarization is ensuring consistency in the quality and style of the summaries produced. Because summarization is a manual task, different summarizers may have varying levels of expertise and interpretive abilities, leading to inconsistencies in the summaries they produce. These inconsistencies can manifest in the form of differences in tone, level of detail, and overall comprehensiveness across summaries of similar articles. As a result, readers may encounter summaries that lack coherence and fail to provide a clear representation of the original article's content.

**PROPOSED SYSTEM:**

News article text summarization where it does overcome all the drawbacks that mentioned in existing system includes:

1. **Automatic Summarization:** The system automates the process of summarizing news articles, reducing the need for manual effort and saving time for users. This automation enables the rapid processing of large volumes of news articles, making it suitable for applications requiring real-time summarization.
2. **Customizable Summarization Parameters:** Users can customize parameters such as the length of the extractive summary and the word limits for the abstractive summary. This flexibility allows users to tailor the summaries to their specific needs or preferences, enhancing the utility of the system.
3. **Combination of Extractive and Abstractive Techniques:** The system leverages both extractive and abstractive summarization techniques to provide a comprehensive summary of news articles. Extractive summarization preserves important sentences from the original article, while abstractive summarization generates summaries that capture the essence of the article in new language. This combination improves the coverage and quality of the summaries, catering to different user preferences and use cases.
4. **Utilization of Pre-trained Models:** The system utilizes pre-trained transformer-based models fine-tuned for summarization tasks, such as the T5 model. These models, trained on large-scale datasets, capture complex language patterns and semantic relationships, leading to more accurate and contextually relevant summaries.
5. **Interactive User Interface:** The system provides an interactive user interface through Streamlit, allowing users to input news articles and visualize the summaries easily. The interface includes sliders and widgets for parameter selection, enhancing user experience and usability.

### 4. ARCHITECTURE



**Fig. 4.1 Architecture**

The architecture presented consists of two Python scripts: highlights.py and summarize.py, which collaborate to create a web application for news article summarization.

**highlights.py:**

**Streamlit Interface Setup:** This script leverages Streamlit, a popular Python library for creating web applications, to build the user interface. It establishes the layout, user input elements, and output presentation for the summarization application.

**Model Loading:** Utilizes the Hugging Face transformers library to load a pre-trained summarization model (shivaniNK8/t5-small-finetuned-cnn-news). This model is responsible for generating abstractive summaries based on input news articles.

**User Interaction:** Provides users with text input areas to submit news articles they wish to summarize.Offers sliders for users to adjust parameters such as summary length for extractive summarization and word limits for abstractive summarization.Includes a "Summarize!" button for triggering the summarization process based on user input.

**Summarization Output:** Displays the extractive summary and abstractive summary of the input news article.Renders the summarization results on the Streamlit app interface for user consumption.

**summarize.py:**

**Text Preprocessing:** Defines a Preprocess class responsible for preparing the input text data before summarization.Offers methods for tasks like converting text to lowercase, tokenizing sentences and words, removing stopwords, and lemmatizing tokens.

**Model Evaluation:** Implements a NewsSummarization class dedicated to summarization-related functionalities.Includes methods for extractive summarization, computing Rouge scores to evaluate the quality of summaries, and evaluating summarization models against datasets.

**Word Embedding Model Training:** Provides functions for training Word2Vec and GloVe word embedding models, which are essential for understanding the semantic relationships between words in the text data.

**Support for Visualization:** Offers capabilities for generating word clouds to visually represent the most frequent words in the input text data.Enhances user understanding and engagement with the summarization process through visualization.By modularizing functionalities into distinct components and scripts, this architecture promotes code organization, reusability, and scalability. It separates concerns between user interface management, text preprocessing, summarization logic, and model evaluation, facilitating easier maintenance and future enhancements.

1. **SYSTEM REQUIREMENT SPECIFICATION**

**(5.1 SOFTWARE REQUIREMENTS)**

**Operating System:** The software should be compatible with commonly used operating systems such as Windows, macOS, or Linux.

**Python:** The code is written in Python programming language, so Python runtime environment needs to be installed on the system. Python version 3.7 or later is recommended.

**Python Libraries:** Install the required Python libraries using pip or conda package managers. The libraries used in the provided code snippets include:

* Streamlit
* pandas
* altair
* NumPy
* PIL (Python Imaging Library)
* nltk (Natural Language Toolkit)
* transformers (Hugging Face Transformers library)
* gensim
* rouge\_score
* matplotlib
* wordcloud

**Other Dependencies:** Ensure that the NLTK data (such as wordnet) is downloaded to support natural language processing tasks. Download any pre-trained models required by the transformer’s library, such as the T5 model used for abstractive summarization.

**Internet Connection:** The system should have internet connectivity to download any additional libraries or models required by the code, such as NLTK data or pre-trained models from the Hugging Face model hub.

**Development Environment:** A code editor or integrated development environment (IDE) such as Visual Studio Code, PyCharm, or Jupyter Notebook can be used for writing and running the code.

**(5.2 HARDWARE REQUIREMENT SPECIFICATION)**

**Processor (CPU):** A multi-core processor with decent processing power is recommended for handling text processing tasks efficiently.

**Memory (RAM):** At least 4GB of RAM is recommended for smooth execution, especially when working with large datasets or running complex summarization models.

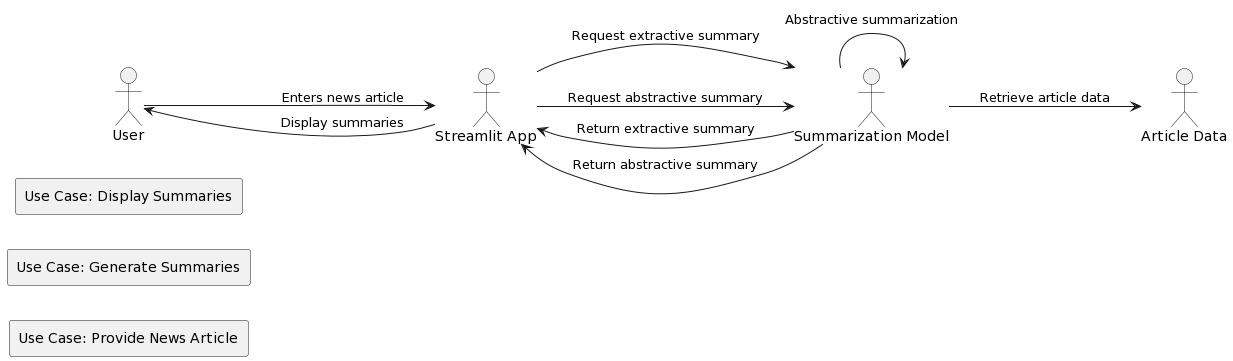
**Storage:** Sufficient disk space to store the application code, libraries, and any generated data. This requirement can vary depending on the size of the dataset and the models used.

1. **SYSTEM DESIGN**

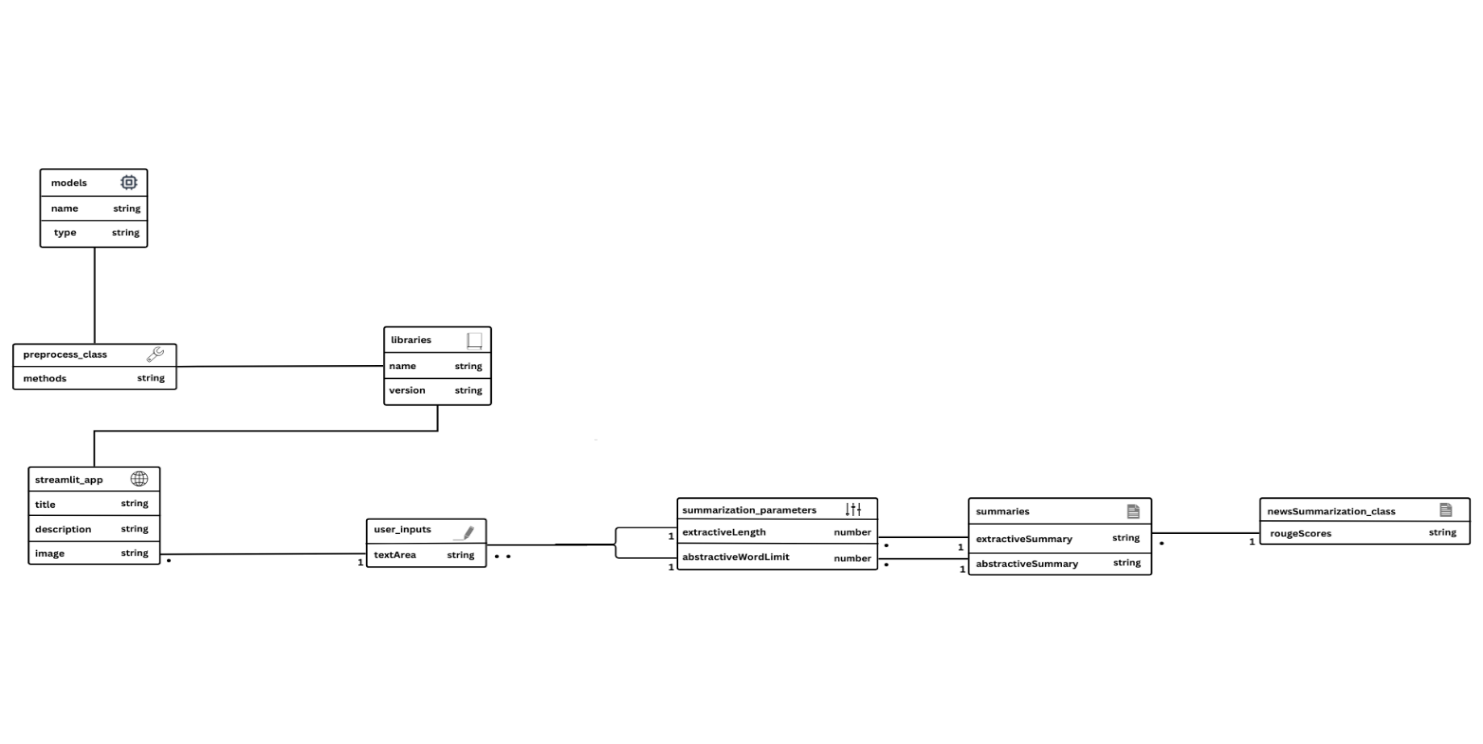
System Design refers to the process of defining the architecture, components, modules, interfaces, and data for a software system. It involves making critical decisions on how the software will meet its functional and non-functional requirements.

**1.UML (Unified Modeling Language) Diagrams**

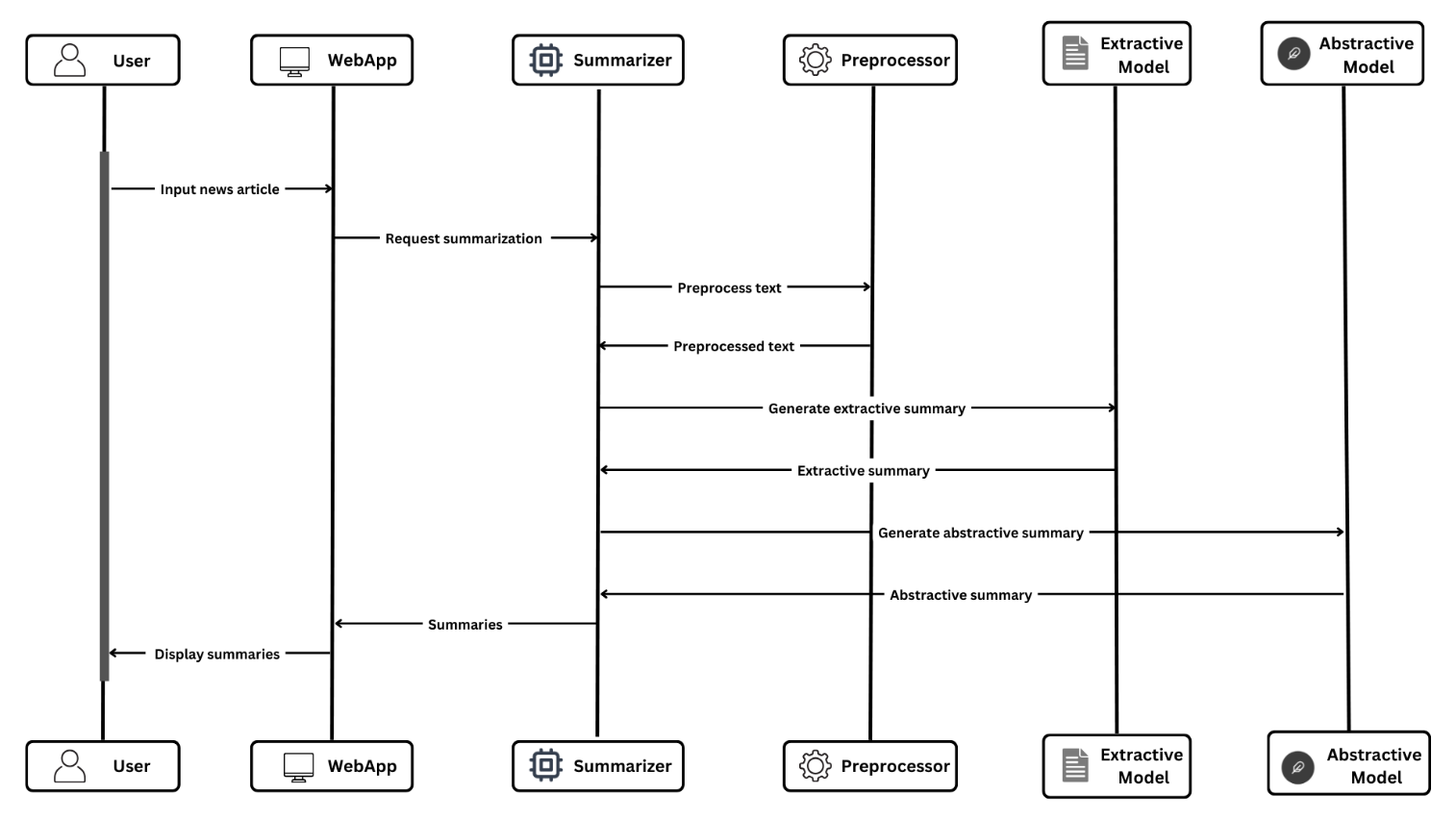
UML (Unified Modeling Language) Diagrams are visual representations used to model software systems. They include various types of diagrams to depict different aspects of the system's structure and behavior:

1. **Use Case Diagram**:
   * **Purpose**: To illustrate the interactions between users (actors) and the system.
   * **Components**: Use cases (functionalities), actors (users), and relationships between them.
   * **Usage**: Use case diagrams show how users interact with the system, identifying key functionalities like user registration, data upload, analysis, and result export.

**Fig 6.1 Use Case Diagram**

1. **ER Diagram**:
   * **Purpose**: To describe the structure of the software system, including classes, their attributes, methods, and relationships.
   * **Components**: Classes, attributes, methods, associations, and inheritance relationships.
   * **Usage**: Class diagrams provide an overview of the system's object-oriented design, representing entities like users, chat data, analysis components, and more.

**Fig 6.2** **ER diagram**

1. **Sequence Diagram**:
   * **Purpose**: To depict the interactions and message exchanges between objects (actors) in the system over time.
   * **Components**: Lifelines (objects), messages, activations, and the order of message flow.
   * **Usage**: Sequence diagrams illustrate how the system components interact during processes like user registration and data analysis, showing the chronological order of actions.

**Fig 6.3 Sequence diagram**

**5. Flowchart**

**Flowchart** is a visual representation of a process or algorithm, often using symbols and arrows to illustrate the steps, decisions, and flow of control within the process.

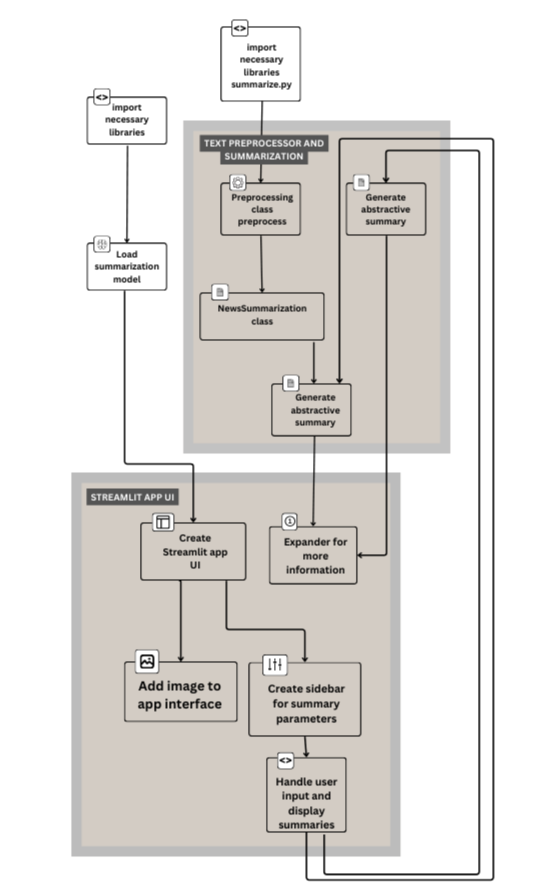
**Purpose**: Flowcharts are designed to visualize the step-by-step sequence of actions or operations within the software system. They provide a clear and easy-to-understand way of representing the logic and flow of the application's functionalities.

**Components**:

**Start/End Symbols**: These symbols represent the beginning and end points of the flowchart, typically depicted as rounded rectangles.

**Process Symbols**: Rectangles or other shapes are used to denote specific actions or operations within the system. For example, a process symbol can represent the analysis of chat data.

**Decision Symbols**: Diamonds indicate points in the flow where a decision or branching occurs. Depending on the condition or outcome, the flow may take different paths.

**Arrows/Flowlines**: Arrows connect the symbols and indicate the direction of flow from one process or decision to the next. They show the logical sequence of operations.

**Fig 6.4 Flow chart**

**7. IMPLEMENTATION**

**Language:** The implementation is primarily done using Python, a versatile and widely-used programming language known for its simplicity and readability. Python offers extensive libraries and frameworks for various tasks, making it well-suited for natural language processing (NLP) tasks like text summarization.

**Libraries and Frameworks:**

**Streamlit:** Utilized for building the web application interface. It simplifies the process of creating interactive web apps in Python.

**Hugging Face Transformers:** Used to load and employ a pre-trained model for text summarization. Hugging Face provides a vast collection of pre-trained models for NLP tasks.

**NLTK (Natural Language Toolkit):** Employed for text preprocessing tasks such as tokenization, stopwords removal, and lemmatization.

**Gensim:** Used for training Word2Vec and GloVe word embedding models, essential for understanding the semantic relationships between words in the text data.

**Matplotlib:** Utilized for generating visualizations such as word clouds to represent the most frequent words in the input text data.

**Execution:** The Streamlit web application is executed from the command line using a command like streamlit run highlights.py. This command launches a local web server, allowing users to interact with the summarization application through their web browser. When a user inputs a news article and clicks the "Summarize!" button, the summarization process is triggered. The summarization logic, implemented in summarize.py, handles tasks such as extracting important sentences from the article (extractive summarization) and generating concise summaries with the pre-trained model (abstractive summarization). The summarization results are then displayed back to the user via the Streamlit interface.

**SAMPLE CODE**

**highlights.py** import streamlit as st

import pandas as pd

import altair as alt

import numpy as np

from PIL import Image

import nltk

from collections import Counter

import heapq

import re

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from transformers import pipeline

from summarize import \*

import nltk

nltk.download('wordnet')

#Load model

hub\_model\_id = "shivaniNK8/t5-small-finetuned-cnn-news"

summarizer = pipeline("summarization", model=hub\_model\_id)

#Create header

st.write("""# HIGHLIGHTS! \n ### A News Summarizer""")

st.write("Provide a news article and get a summary within seconds! ")

# Image

image = Image.open('newspaper.jpeg')

st.image(image)

#Create and name sidebar

st.sidebar.header('Select summary parameters')

with st.sidebar.form("input\_form"):

st.write('Select summary length for extractive summary')

max\_sentences = st.slider('Summary Length', 1, 10, step=1, value=3)

st.write('Select word limits for abstractive summary')

max\_words = st.slider('Max words', 50, 500, step=10, value=200)

min\_words = st.slider('Min words', 10, 450, step=10, value=100)

submit\_button = st.form\_submit\_button("Summarize!")

article = st.text\_area(label = "Enter the article you want to summarize", height = 300, value = "Enter Article Body Here")

news\_summarizer = NewsSummarization()

if submit\_button:

st.write("## Extractive Summary")

ex\_summary = news\_summarizer.extractive\_summary(article, num\_sentences = max\_sentences)

st.write(ex\_summary)

summary = summarizer(article, max\_length = max\_words, min\_length = min\_words, do\_sample = False)

abs\_summary = summary[0]['summary\_text']

st.write("## Abstractive Summary")

st.write(abs\_summary)

with st.sidebar.expander("More About Summarization"):

st.markdown("""

In extractive summarization, we identify important sentences from the article and make a summary by selecting the most important sentences. <br>

Whereas, for abstractive summarization the model understands the context and generates a summary with the important points with new phrases and language.

Abstractive summarization is more similar to the way a human summarizes any content. A person might read the entire document,

remember a few key points and while writing the summary, will make new sentences that include these points. Abstractive summarization follows the same concept. """)

**summarize.py**  import logging

import nltk

from collections import Counter

import heapq

import re

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from transformers import pipeline

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from gensim.corpora import Dictionary # Added import

from gensim.models import word2vec

from gensim.corpora import MmCorpus

from gensim.scripts.glove2word2vec import glove2word2vec

# Added import

from rouge\_score import rouge\_scorer

class Preprocess():

def \_init\_(self):

pass

def toLower(self, x):

'''Converts string to lowercase'''

return x.lower()

def sentenceTokenize(self, x):

'''Tokenizes document into sentences'''

sent\_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")

sentences = sent\_tokenizer.tokenize(x)

return sentences

def preprocess\_sentences(self, all\_sentences):

'''Tokenizes sentences into words, removes punctuations, stopwords and

performs tokenization'''

word\_tokenizer = nltk.RegexpTokenizer(r"\w+")

sentences = []

special\_characters = re.compile("[^A-Za-z0-9 ]")

for s in all\_sentences:

# remove punctuation

s = re.sub(special\_characters, " ", s)

# Word tokenize

words = word\_tokenizer.tokenize(s)

# Remove Stopwords

words = self.removeStopwords(words)

# Perform lemmatization

words = self.wordnet\_lemmatize(words)

sentences.append(words)

return sentences

def removeStopwords(self, sentence):

'''Removes stopwords from a sentence'''

stop\_words = stopwords.words('english')

tokens = [token for token in sentence if token not in stop\_words]

return tokens

def wordnet\_lemmatize(self, sentence):

'''Lemmatizes tokens in a sentence'''

lemmatizer = WordNetLemmatizer()

tokens = [lemmatizer.lemmatize(token, pos='v') for token in sentence]

return tokens

def complete\_preprocess(self, text):

'''Performs complete preprocessing on document'''

#Convert text to lowercase

text\_lower = self.toLower(text)

#Sentence tokenize the document

sentences = self.sentenceTokenize(text\_lower)

#Preprocess all sentences

preprocessed\_sentences = self.preprocess\_sentences(sentences)

return preprocessed\_sentences

def generate\_wordcloud(self, text):

word\_cloud = WordCloud(collocations = False, background\_color = 'white').generate(text)

plt.figure(figsize=(15,8))

plt.imshow(word\_cloud, interpolation='bilinear')

plt.axis("off")

plt.show()

def calculate\_length(self, df):

df["article\_len"] = df["article"].apply(lambda x: len(x.split()))

df["highlights\_len"] = df["highlights"].apply(lambda x: len(x.split()))

return df

def most\_similar\_words(self, model, words):

'''Returns most similar words to a list of words'''

for word in words:

print("Most similar to ", word,": ", model.wv.most\_similar(word))

def word2vec\_model(self, sentences,num\_feature, min\_word\_count,

window\_size, down\_sampling, sg):

'''Creates and trains Word2Vec model'''

num\_thread = 5

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

model = word2vec.Word2Vec(sentences,

#iter = iteration,

vector\_size=num\_feature,

min\_count = min\_word\_count,

window = window\_size,

sample = down\_sampling,

workers=num\_thread,

sg = sg,

epochs = 20)

return model

def glove\_model(self, sentences, window\_size, num\_features, lr, iterations):

'''Creates and trains GloVe model'''

num\_thread = 5

corpus = MmCorpus()

# Create word co occurence matrix

corpus.fit(sentences, window=window\_size)

glove = glove2word2vec(no\_components=num\_features, learning\_rate=lr)

# Fit model

glove.fit(corpus.matrix, epochs=iterations, no\_threads=num\_thread)

glove.add\_dictionary(corpus.dictionary)

return glove

def most\_similar\_words\_glove(self, model, words):

'''Returns most similar words to a list of words for GloVe model'''

for word in words:

print("Most similar to ", word,": ", model.most\_similar(word))

def top\_10\_frequent\_words(self, model):

'''Returns top 10 frequent words'''

# sort model vocab according to top frequent words

model.sorted\_vocab

top\_words = model.wv.index\_to\_key[:10]

return top\_words

class NewsSummarization():

def \_init\_(self):

pass

def extractive\_summary(self, text, sentence\_len = 8, num\_sentences = 3):

'''Generates extractive summary of num\_sentences length using sentence scoring'''

word\_frequencies = {}

# Instantiate Custom Preprocessor class

preprocessor = Preprocess()

# preprocess and tokenize article

tokenized\_article = preprocessor.complete\_preprocess(text)

#calculate word frequencies

for sentence in tokenized\_article:

for word in sentence:

if word not in word\_frequencies.keys():

word\_frequencies[word] = 1

else:

word\_frequencies[word] += 1

#get maximum frequency for score normalisation

maximum\_frequency = max(word\_frequencies.values())

#normalize word frequency

for word in word\_frequencies.keys():

word\_frequencies[word] = (word\_frequencies[word]/maximum\_frequency)

sentence\_scores = {}

# score sentences by adding word scores

sentence\_list = nltk.sent\_tokenize(text)

for sent in sentence\_list:

for word in nltk.word\_tokenize(sent.lower()):

if word in word\_frequencies.keys():

if len(sent.split(' ')) > sentence\_len:

if sent not in sentence\_scores.keys():

sentence\_scores[sent] = word\_frequencies[word]

else:

sentence\_scores[sent] += word\_frequencies[word]

# get sentences with largest sentence scores

summary\_sentences = heapq.nlargest(num\_sentences, sentence\_scores, key=sentence\_scores.get)

# join and get extractive summary

summary = ' '.join(summary\_sentences)

return summary

def get\_rouge\_score(self, actual\_summary, generated\_summary):

scorer = rouge\_scorer.RougeScorer(['rouge1', 'rougeL'], use\_stemmer=True)

scores = scorer.score(actual\_summary, generated\_summary)

return scores

def evaluate\_extractive(self, dataset, metric):

summaries = [self.extractive\_summary(text) for text in dataset["article"]]

score = metric.compute(predictions=summaries, references=dataset["highlights"])

rouge\_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]

rouge\_dict = dict((rn, round(score[rn].mid.fmeasure \* 100, 2)) for rn in rouge\_names)

return rouge\_dict

def evaluate\_abstractive(self, dataset, metric, summarizer):

summaries = [summarizer(text, max\_length = 120, min\_length = 80, do\_sample = False)[0]['summary\_text'] for text in dataset["article"]]

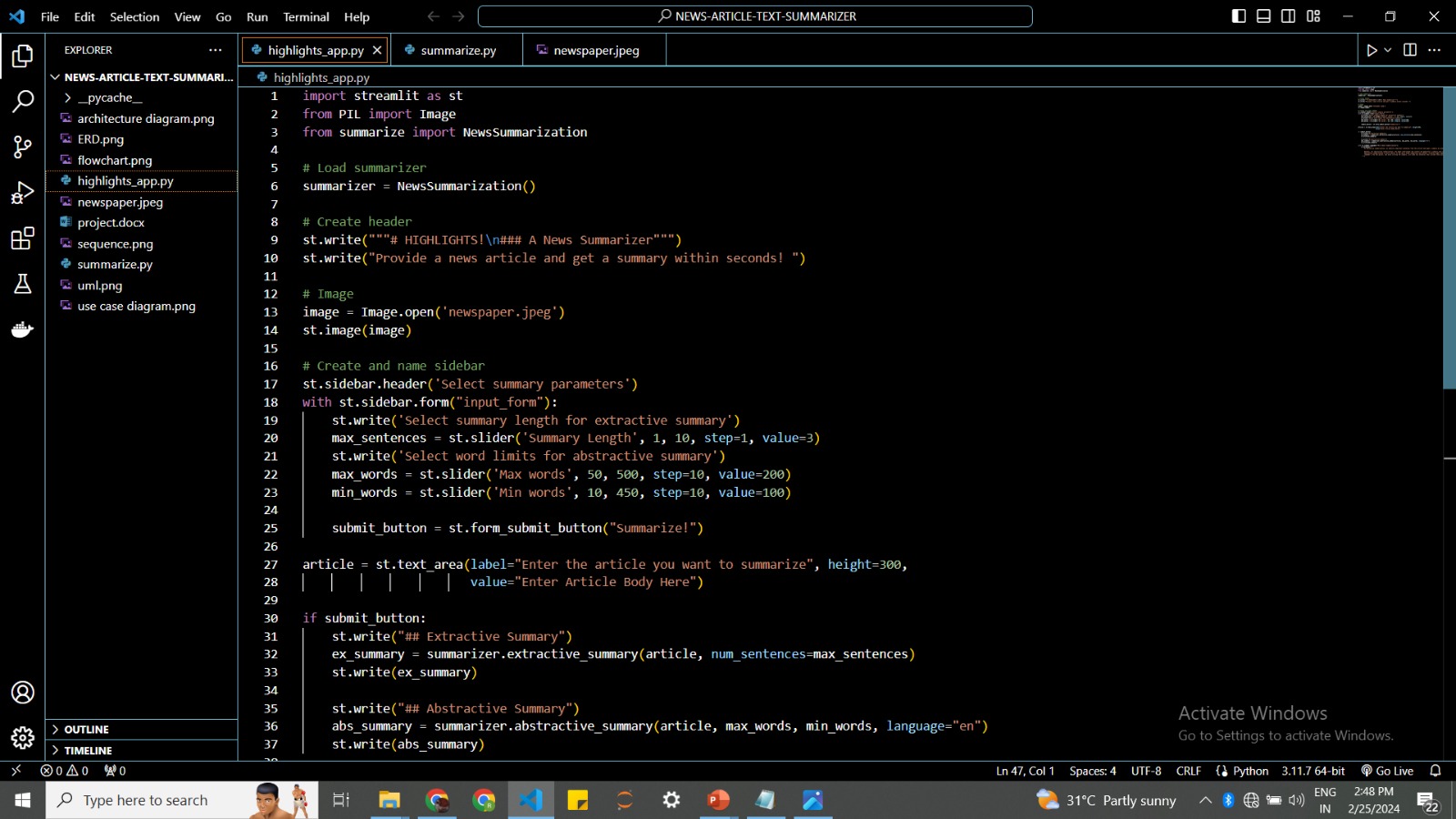
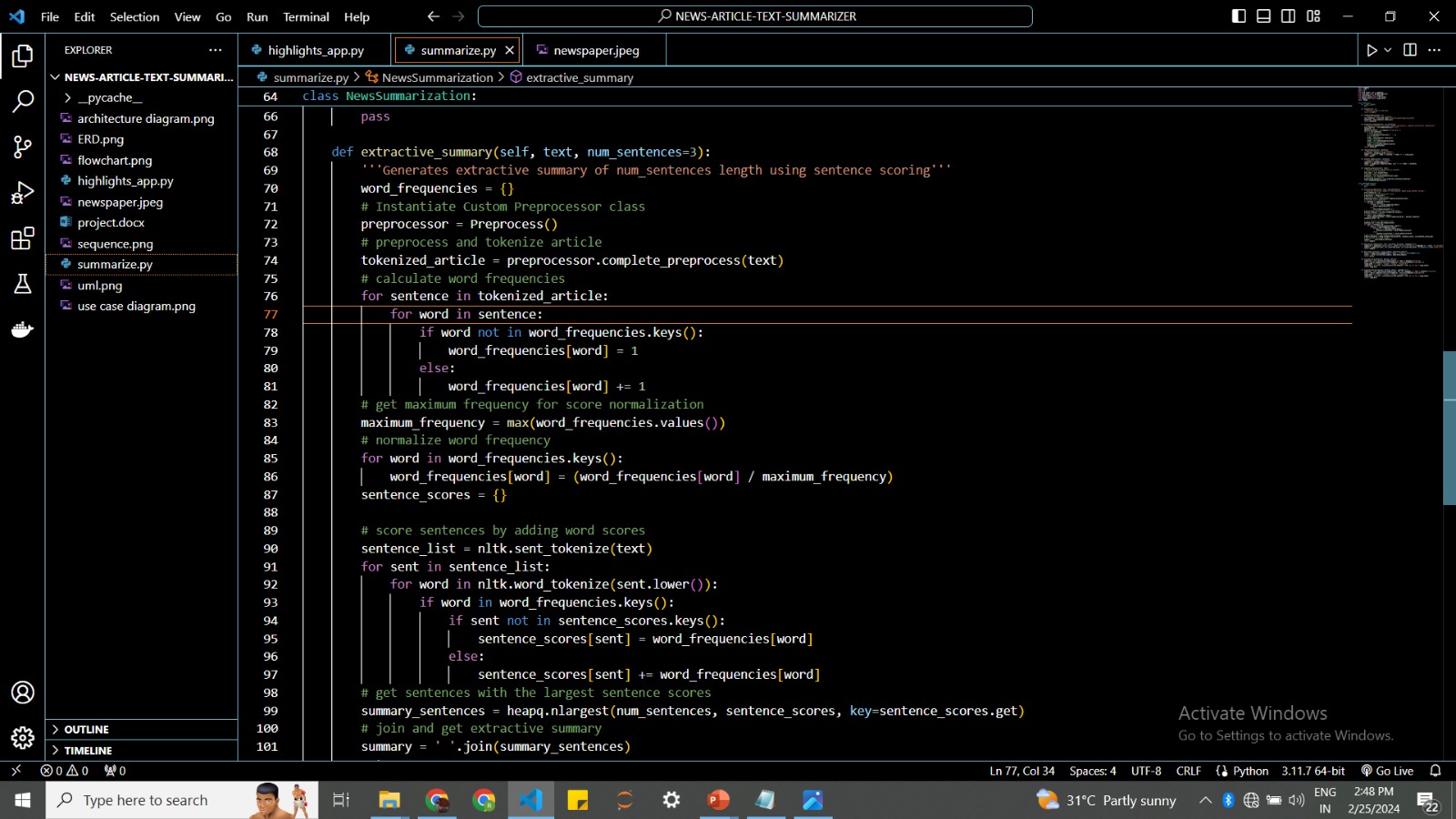
score = metric.compute(predictions=summaries, references=dataset["highlights"])

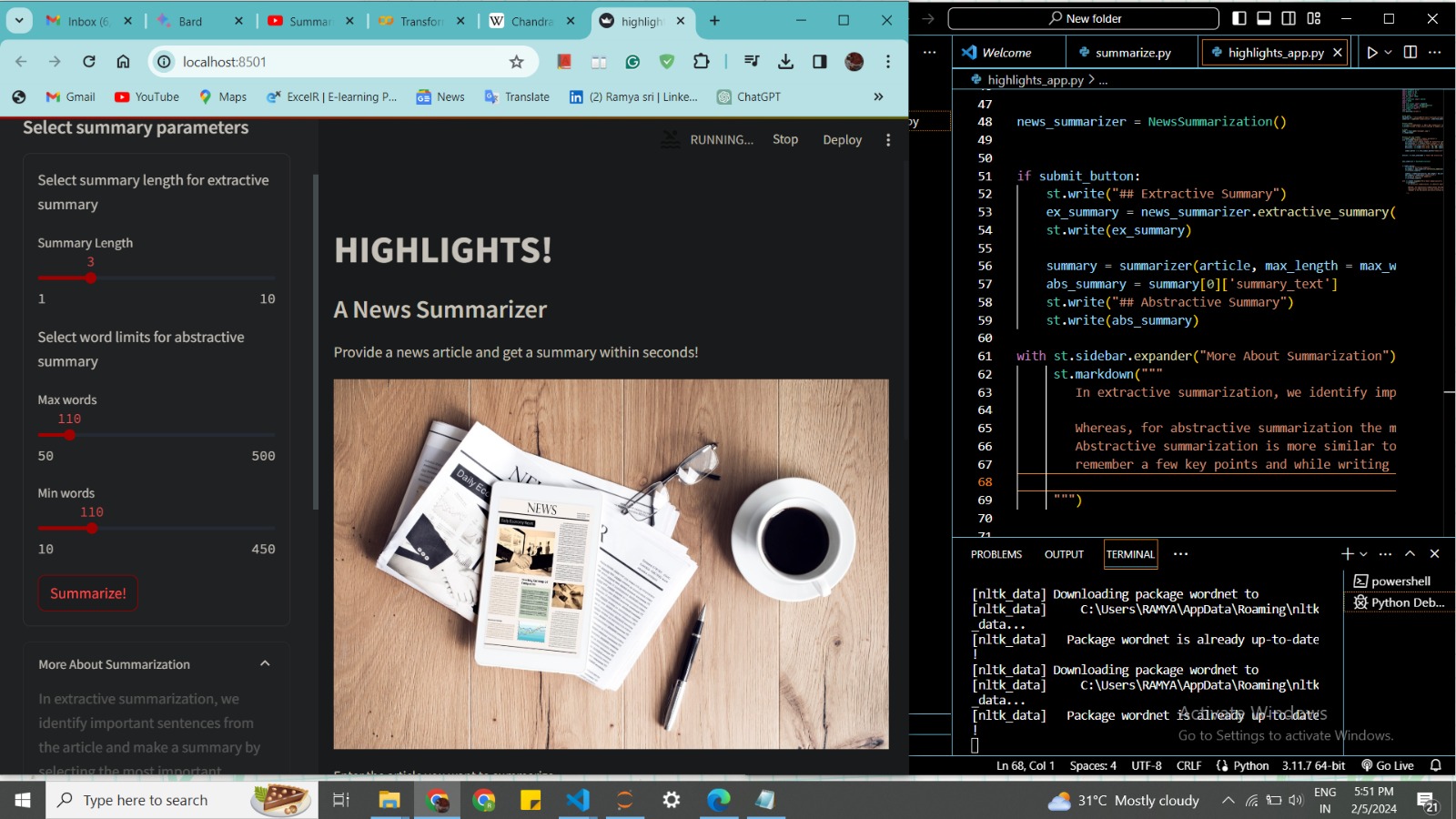
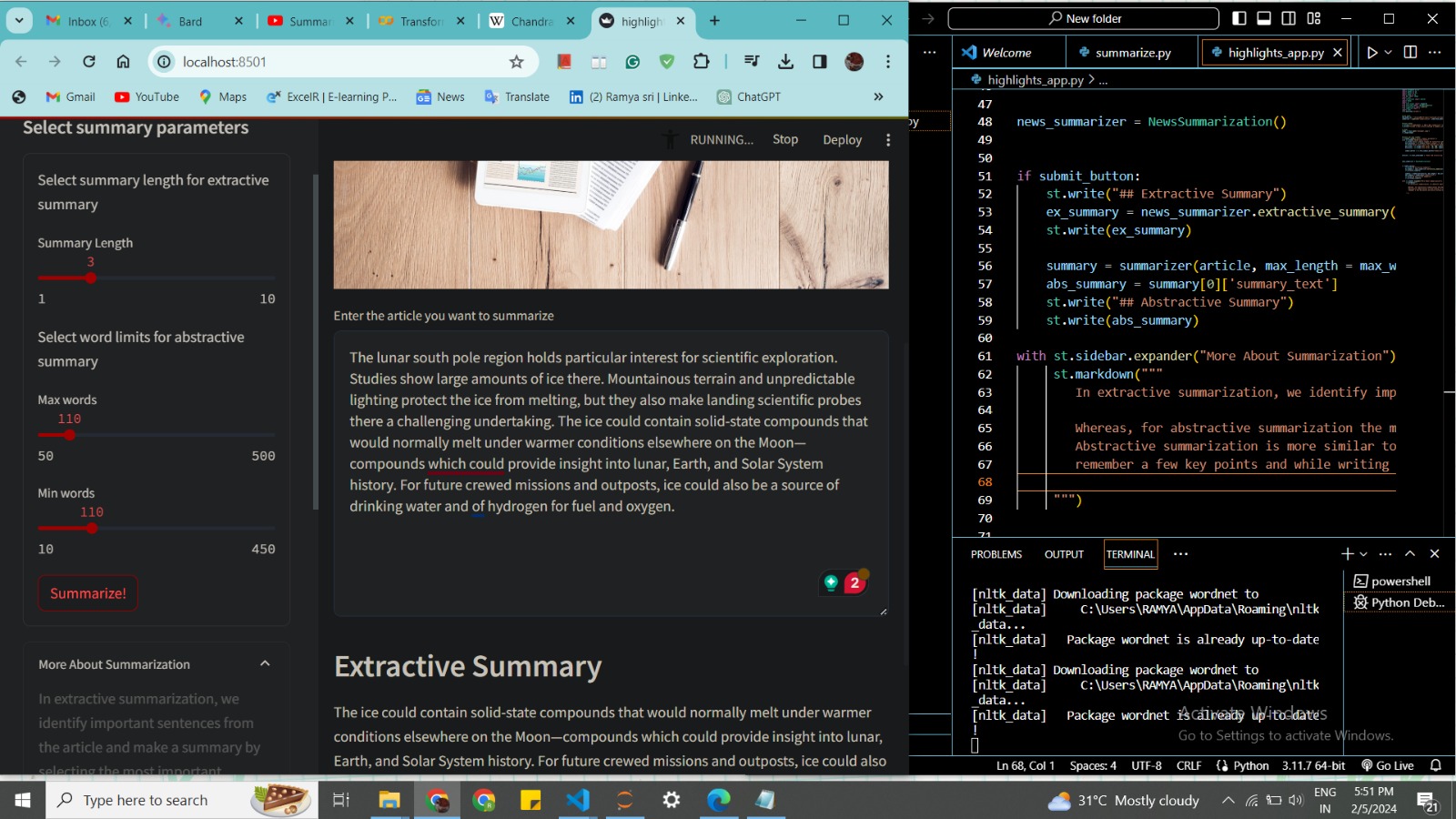
rouge\_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]

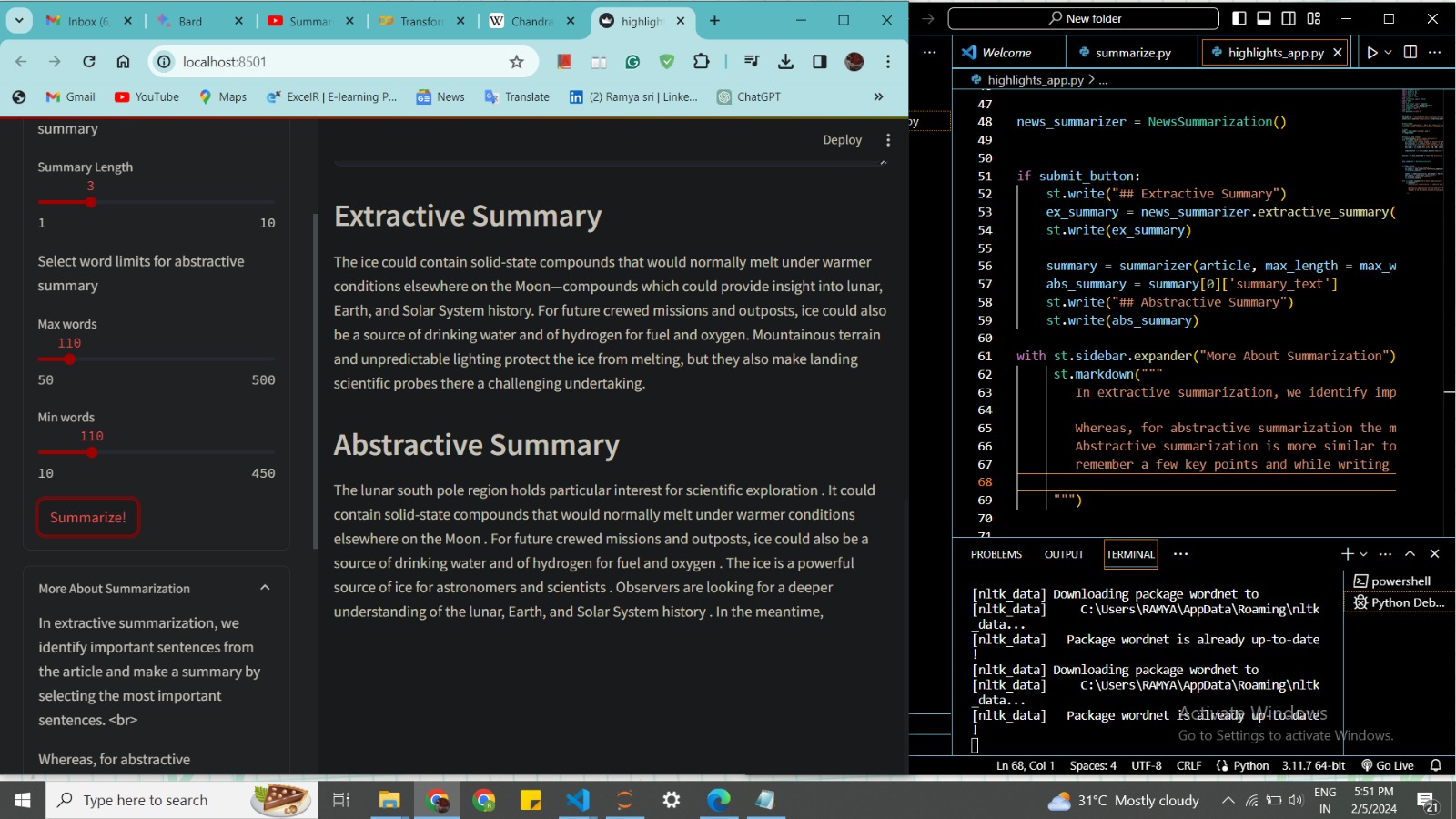
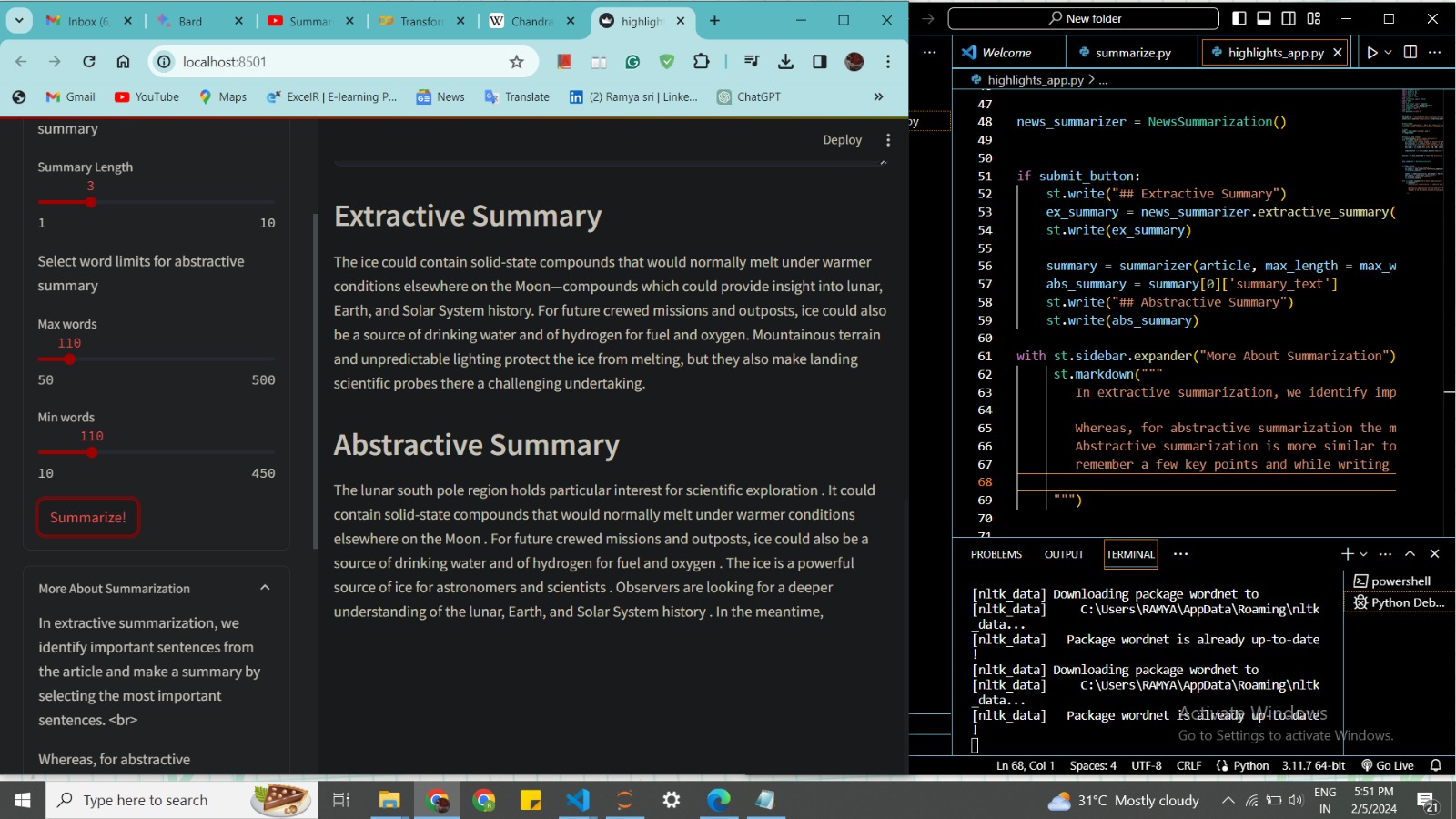
rouge\_dict = dict((rn, round(score[rn].mid.fmeasure \* 100, 2)) for rn in rouge\_names)

return rouge\_dict

1. **SCREENSHOTS**







**9. CONCLUSION**

In conclusion, the project successfully developed a news summarization system capable of generating both extractive and abstractive summaries, providing users with valuable insights into news articles in an efficient and user-friendly manner. While the system demonstrates significant achievements, ongoing refinement and optimization efforts are essential to further enhance its performance and usability. Overall, the project lays a solid foundation for future advancements in text summarization technology and its applications in various domains.

The project's development of a news summarization system that can generate both extractive and abstractive summaries reflects its versatility and adaptability. This dual capability ensures that users have access to summaries that suit their preferences and needs, whether they prefer concise, point-based extracts or more nuanced, paraphrased content. By providing users with concise yet comprehensive summaries of news articles, the project contributes to enhancing the user experience of accessing and digesting large volumes of information. Users can quickly grasp the key points and essential insights of news articles without having to read through lengthy texts, thereby saving time and effort.

**10. FUTURE SCOPE**

The project on news summarization has a promising future scope, with several avenues for further development and expansion:

1. **Domain-specific Summarization:** Customizing the summarization models to specific domains, such as finance, healthcare, or technology, would improve the relevance and accuracy of the generated summaries for users with specialized interests. Fine-tuning the models on domain-specific datasets can enhance their understanding of domain-specific terminology and context.
2. **Real-time Summarization:** Implementing real-time summarization capabilities to process and summarize news articles as they are published would provide users with up-to-date and timely summaries of the latest news events. This could involve integrating the system with news APIs or RSS feeds to automatically fetch and summarize news articles as they become available.
3. **User Feedback Integration:** Incorporating mechanisms for user feedback and evaluation would enable continuous improvement of the summarization models based on user preferences and quality assessments. Collecting feedback on the generated summaries and adjusting the models accordingly can enhance their performance and relevance over time.
4. **Visual Summarization:** Exploring techniques for visual summarization, such as generating summary infographics or key phrase extraction, could provide users with alternative formats for consuming summarized content. Visual summarization techniques can enhance comprehension and engagement, particularly for users who prefer visual information presentation.

**REFERENCES**

[1] M. Haque, et al.,” Literature Review of Automatic Multiple Documents Text Summarization”, International Journal of Innovation and Applied Studies, vol. 3, pp. 121-129, 2013.

[2] N.Moratanch ,S.Chitrakala “A Survey on Extractive Text Summarization” IEEE International Conference on Computer, Communication, and Signal Processing (ICCCSP-2017)

[3] Gune ¨ ¸s Erkan, Dragomir R. Radev, “LexRank: Graph-based Lexical Centrality as Salience in Text Summarization”. Department of EECS University of Michigan, Ann Arbor, MI 48109 USA Journal of Artificial Intelligence Research 22 (2004) 457-479.

[4] Prakhar Sethi, Sameer Sonawane, Saumitra Khanwalker, R.B. Keskar “Automatic Text Summarization of News Articles” 2017 International Conference on Big Data, IoT and Data Science.

[5] R. Mihalcea, “Graph-base ranking algorithms for sentence extraction, applied to text summarization”, in Proceedings of ACL 2004 on Interactive poster and demonstration sessions, 2004.