

Chapter 1: Introduction

With the increasing of online information and recourse texts, text summarization has become an essential and more favourite domain to preserve and show the main purpose of textual information. It is very difficult for human beings to summarize manually large documents of text. Text summarization is the process of automatically creating and condensing form of a given document and preserving its information content source into a shorter version with overall meaning. Nowadays text summarization is one of the most favourite research areas in natural language processing and could attracted more attention of NLP researchers. There are also much more close relationships between text mining and text summarization. According to difference requirements summary with respect to input text, established summarization systems should be created and classified based on the type of input text. In this study, at first, the topic of text mining and its relationship with text summarization are considered. Then a review has been done on some of the summarization approaches and their important parameters for extracting predominant sentences, identified the main stages of the summarizing process, and the most significant extraction criteria are presented. Finally, the most fundamental proposed evaluation methods are considered. The subfield of summarization has been investigated by the NLP community for nearly the last half century. Radev et al. (2002) define a summary as "a text that is produced from one or more texts, that conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually significantly less than that".

Extraction is the procedure of identifying important sections of the text and producing them verbatim and compression aims to throw out unimportant sections of the text. Text data contains a lot of information but not all of it will be important to you. We might be looking for names of entities, others would want to extract specific relationships between those entities. Our intentions differ according to our requirements. Imagine having to go through all the legal documents to find legal precedence to validate your current case. Or having to go through all the research papers to find relevant information to cure a disease. There are many more examples like resume harvesting, media analysis, email scanning, etc. But just imagine having to manually go through all of the textual data and extracting the most relevant information. Clearly, it is an uphill battle and you might even end up skipping some important information

Problem Statement

To create a text summarizer which summarises the text or the content of the paragraph in minimum words without changing its meaning. This system is made using NLP based model which is branch of machine learning. This text summarizer also summarizes text from the weblinks and also summarises text from a document.

1.1 Literature Survey

Chin-Yew Lin [16] In this paper author introduced Recall Oriented Understudy for Gisting Evaluation ROUGE. That is an automatic evaluation package for text summarization. The paper also introduced four different measures of ROUGE: - ROUGE-N, ROUGE-L, ROUGE-W and ROUGE-S. It measures the quality of summary by comparing the generated summary with other ideal summaries that are created by humans. These methods are efficient for automatic evaluation of single document summary as well as multi-document summaries.

Akshil Kumar et al. [17] In this paper author has analyzed and compared the performance of three different algorithms. Firstly, the different text summarization techniques explained. Extraction based techniques are used to extract important keywords to be included in the summary. The ROUGE 1 is used to evaluate the effectiveness of the extracted keywords. The results of the algorithms compared with the handwritten summaries and evaluate the performance. In the end, the TextRank Algorithm gives a better result than other two algorithms.

Pankaj Gupta et al. [18] In this paper author has reviewed different techniques of Sentiment analysis and different techniques of text summarization. Sentiment analysis is a machine learning approach in which machine learns and analyze the sentiments, emotions present in the text. The machine learning methods like Naive Bayes Classifier and Support In Text summarization, uses the natural language processing (NLP) and linguistic features of sentences are used for checking the importance of the words and sentences that can be included in the final summary.

Harsha Dave et al. [19] In this paper author has proposed a system to generate the extractive summary using WordNet ontology. The multiple documents had been used like text, pdf, word files etc. The experiment result is compared with the existing online extractive tools as well as with human generated summaries and shows the proposed system gives good results.

. Yihong Gong et al. [20] In this research paper the author proposes two methods that create the generic text summaries by ranking and extracting sentences from the main text documents. The first method uses information retrieval (IR) methods that rank the sentence relevance and provides the relevance scores to sentences and the second method uses the latent semantic analysis (LSA) technique that based on latent semantic indexing (LSI) in order to identify the semantic importance of the sentences, for summary creations.

1.2 Objectives

The main objectives of the text extraction are as follows :23

- Summaries reduce reading time.
- When researching documents, summaries make the selection process easier.
- Automatic summarization improves the effectiveness of indexing.
- Automatic summarization algorithms are less biased than human summarizers.

1.3 Scope and Limitations

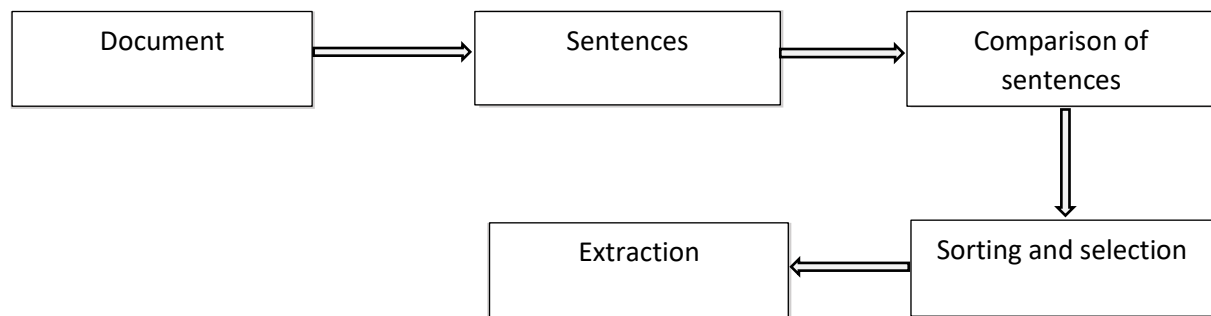
The future scope of our project is to reducing length of data. The proposed system along with text to speech converter will be helpful to blind people. The web application can be convert into mobile application. Limitation of our project is we are unable to extract text from unstructured big data and we face various language and domain limitation.

Chapter 2: Methodology

2.1 Methodology

As project is web application, the proposed system contain only software phases.

The software phase used different algorithm and function for extract text from large data. Google collab is used to run the project. As a result of all these action together, the required text is extract from large data.



Document is upload along with code. Document is break into paragraph then sentences. Sentences are compare to identify importance of them. After that sorting the sentences and select according to importance. Get extraction as output.

Chapter 3 Software and components used

3.1 Software Tools:

3.1.1 Vs code

Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including Java, JavaScript, Go, Node.js, Python, C++, C, Rust and Fortran. It is based on the electron framework, which is used to develop Node.js web application that run on the blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services). Visual Studio Code was first announced on April 29, 2015, by Microsoft at the 2015 Build conference. A preview build was released shortly thereafter. On November 18, 2015, the source of Visual Studio Code was released under the MIT license, and made available on GitHub. Extension support was also announced. On April 14, 2016, Visual Studio Code graduated from the public preview stage and was released to the web. Microsoft has released most of Visual Studio Code's source code on GitHub under the permissive MIT license while the releases by Microsoft are proprietary freeware.

3.2 Components

3.1.1 NLTK

NLTK is a popular Python framework for dealing with data of human language. It includes a set of text processing libraries for classification and semantic reasoning, as well as wrappers for industrial-strength NLP libraries and an active discussion forum. NLTK is ideal for linguists, engineers, students, and industry users. NLTK is very useful in a python programming language. Natural Language Processing (NLP) is a process of manipulating or understanding the text or speech by any software or machine. An analogy is that humans interact and understand each other's views and respond with the appropriate answer. In NLP, this interaction, understanding, and response are made by a computer instead of a human. NLTK (Natural Language Toolkit) Library is a suite that contains libraries and programs for statistical language processing. It is one of the most powerful NLP libraries, which contains packages to make machines understand human language and reply to it with an appropriate response.

3.2.2 Bag of word

Bag of words is a Natural Language Processing technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents. A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a “bag” of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document. One of the biggest problems with text is that it is messy and unstructured, and machine learning algorithms prefer structured, well defined fixed-length inputs and by using the Bag-of-Words technique we can convert variable-length texts into a fixed-length **vector**. Also, at a much granular level, the machine learning models work with numerical data rather than textual data. So to be more specific, by using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers.

3.2.3 Tokenization

Tokenization is the first step in any NLP pipeline. It has an important effect on the rest of your pipeline. A tokenizer breaks unstructured data and natural language text into chunks of information that can be considered as discrete elements. The token occurrences in a document can be used directly as a vector representing that document. This immediately turns an unstructured string (text document) into a numerical data structure suitable for machine learning. They can also be used directly by a computer to trigger useful actions and responses. Or they might be used in a machine learning pipeline as features that trigger more complex decisions or behavior. Tokenization can separate sentences, words, characters, or subwords. When we split the text into sentences, we call it sentence tokenization. For words, we call it word tokenization.

3.2.4 Lemmatizer

In NLTK, lemmatization is nothing but the process of algorithmic determining a word's lemma based on its meaning and context. In most cases, lemmatization refers to the morphological study of words to remove inflectional endings. It aids in the retrieval of the lemma, or basic or dictionary form, of a word. The NLTK Lemmatization technique uses the built-in morph function in WordNet. The suffix word is how the stemming method works. In a larger sense, it slashes the word's beginning or finish.

3.2.5 Term Frequency Inverse Document Frequency

TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus. Term Frequency: In document d , the frequency represents the number of instances of a given word t . Therefore, we can see that it becomes more relevant when a word appears in the text, which is rational. Since the ordering of terms is not significant, we can use a vector to describe the text in the bag of term models. For each specific term in the paper, there is an entry with the value being the term frequency. Document Frequency: This tests the meaning of the text, which is very similar to TF, in the whole corpus collection. The only difference is that in document d , TF is the frequency counter for a term t , while df is the number of occurrences in the document set N of the term t . In other words, the number of papers in which the word is present is DF. Inverse Document Frequency: Mainly, it tests how relevant the word is. The key aim of the search is to locate the appropriate records that fit the demand. Since tf considers all terms equally significant, it is therefore not only possible to use the term frequencies to measure the weight of the term in the paper.

3.2.6 Cosine similarity

Cosine similarity is a metric used to determine how similar two entities are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space.

3.2.7 TextRank

TextRank is a text summarization technique which is used in Natural Language Processing to generate Document Summaries. PageRank is an algorithm used to calculate rank of web pages, and is used by search engines such as Google. TextRank is based on the PageRank Algorithm.

3.2.8 NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

Audio converter

Speech Recognition is an important feature in several applications used such as home automation, artificial intelligence, etc.

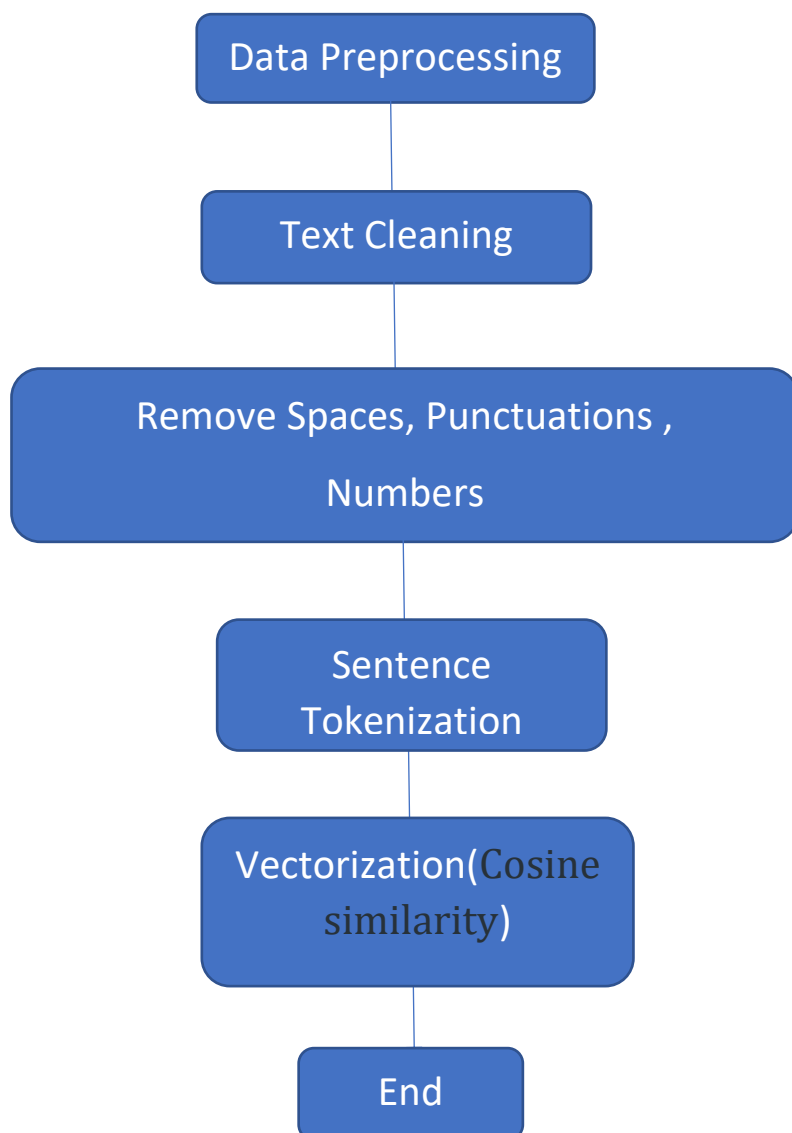
This article aims to provide an introduction on how to make use of the Speech Recognition and pyttsx3 library of Python.

Installation required:

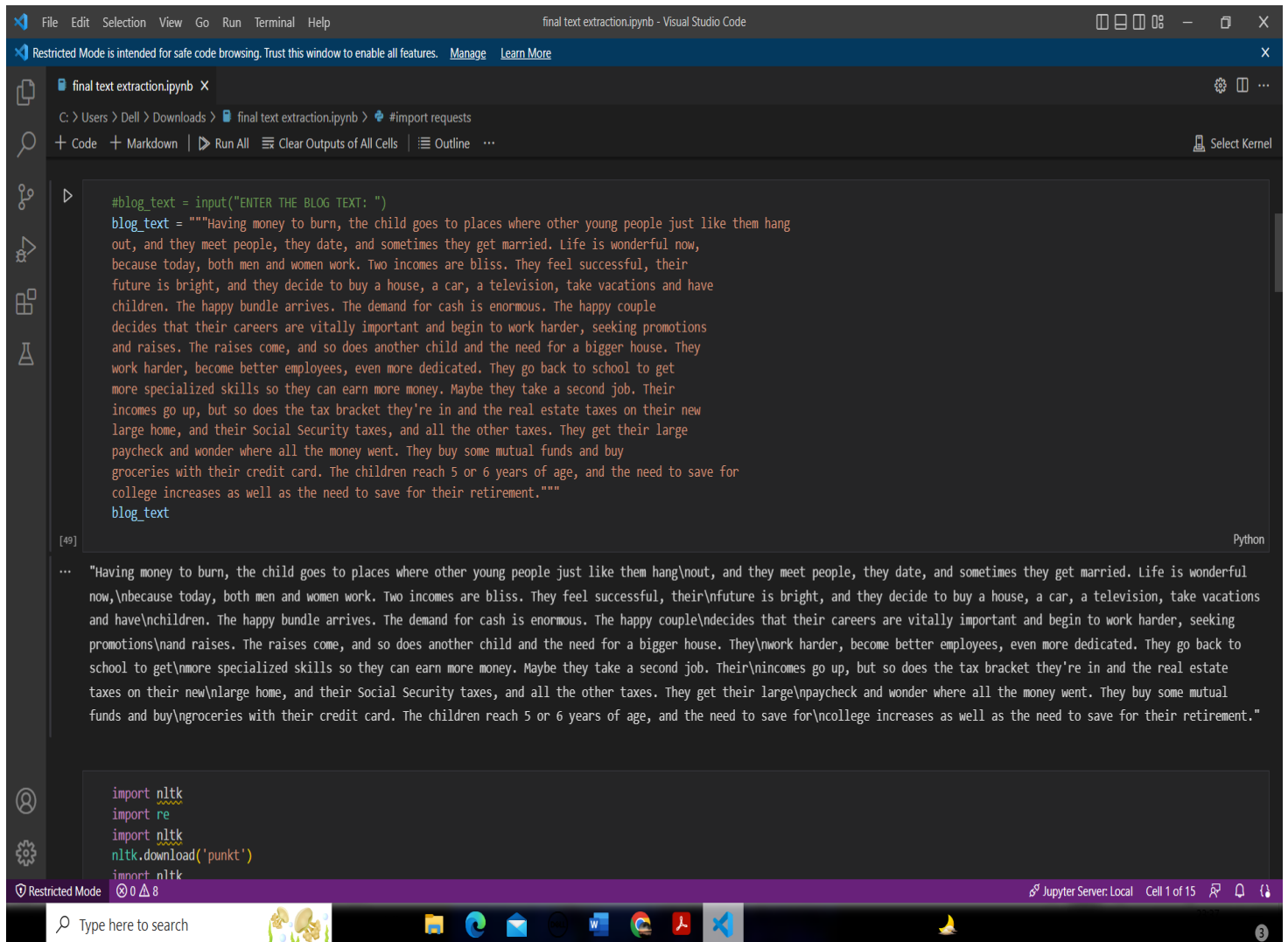
- **Python Speech Recognition module:**
`pip install speech recognition`
- **PyAudio:** Use the following command for linux users
`sudo apt-get install python3-pyaudio`
- Windows users can install pyaudio by executing the following command in a terminal
`pip install pyaudio`
- **Python pyttsx3 module:**
`pip install pyttsx3`

Chapter 4 Mini Project Working

4.1 Model Working:



4.2 Program



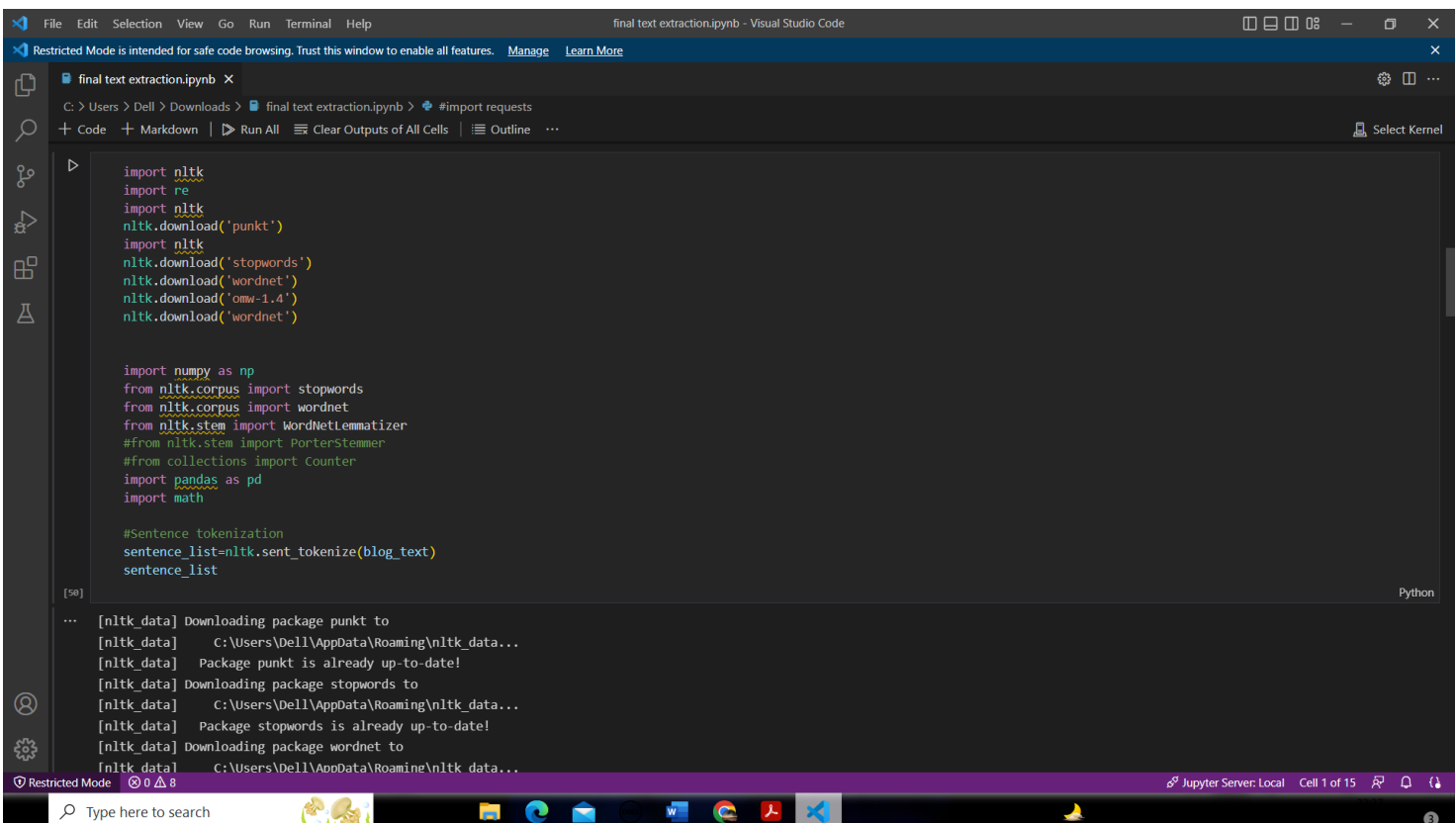
The screenshot shows a Jupyter Notebook titled "final text extraction.ipynb" in Visual Studio Code. The notebook is in "Restricted Mode". The code in the first cell defines a variable `blog_text` with a long paragraph of text. The output of this cell is the same text, displayed as a single line. The second cell contains code to import `nlTK` and `re` modules, and to download the `punkt` model. The output of this cell is the text: "Jupyter Server: Local Cell 1 of 15".

```
#blog_text = input("ENTER THE BLOG TEXT: ")
blog_text = """"Having money to burn, the child goes to places where other young people just like them hang
out, and they meet people, they date, and sometimes they get married. Life is wonderful now,
because today, both men and women work. Two incomes are bliss. They feel successful, their
future is bright, and they decide to buy a house, a car, a television, take vacations and have
children. The happy bundle arrives. The demand for cash is enormous. The happy couple
decides that their careers are vitally important and begin to work harder, seeking promotions
and raises. The raises come, and so does another child and the need for a bigger house. They
work harder, become better employees, even more dedicated. They go back to school to get
more specialized skills so they can earn more money. Maybe they take a second job. Their
incomes go up, but so does the tax bracket they're in and the real estate taxes on their new
large home, and their Social Security taxes, and all the other taxes. They get their large
paycheck and wonder where all the money went. They buy some mutual funds and buy
groceries with their credit card. The children reach 5 or 6 years of age, and the need to save for
college increases as well as the need to save for their retirement."""
blog_text

[49] Python
```

```
import nltk
import re
import nltk
nltk.download('punkt')
import nltk
```

Jupyter Server: Local Cell 1 of 15



The screenshot shows the same Jupyter Notebook in Visual Studio Code. The code in the third cell imports `numpy` and `nlTK` modules, and downloads the `stopwords`, `wordnet`, and `omw-1.4` models. The output of this cell is the text: "Jupyter Server: Local Cell 1 of 15".

```
import nltk
import re
import nltk
nltk.download('punkt')
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('wordnet')

import numpy as np
from nltk.corpus import stopwords
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
#from nltk.stem import PorterStemmer
#from collections import Counter
import pandas as pd
import math

#Sentence tokenization
sentence_list=nltk.sent_tokenize(blog_text)
sentence_list

[50] Python
```

```
[nlTK_data] Downloading package punkt to
[nlTK_data] C:\Users\Dell\AppData\Roaming\nltk_data...
[nlTK_data] Package punkt is already up-to-date!
[nlTK_data] Downloading package stopwords to
[nlTK_data] C:\Users\Dell\AppData\Roaming\nltk_data...
[nlTK_data] Package stopwords is already up-to-date!
[nlTK_data] Downloading package wordnet to
[nlTK_data] C:\Users\Dell\AppData\Roaming\nltk_data...
```

Jupyter Server: Local Cell 1 of 15

File Edit Selection View Go Run Terminal Helpfinal text extraction.ipynb - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

final text extraction.ipynb X

C:\Users\> Dell > Downloads > final text extraction.ipynb > #import requests

+ Code + Markdown Run All Clear Outputs of All Cells Outline ...

Select Kernel

```
change_sentence_list = sentence_list.copy()
#creattig deep copy of sentence tokenizing list

for i in range(0,len(change_sentence_list)):
    change_sentence_list[i] = re.sub(r'^\w\s','',change_sentence_list[i])
# removing all punctuations

# stopwords list, lemmatizer and stemmer
stopwords_list=stopwords.words('english')
lemmatizer = WordNetLemmatizer()
#stemmer = PorterStemmer()

#splitting sentences into words and storing them in place of sentences as list of words
for i in range(0,len(change_sentence_list)):
    change_sentence_list[i] = change_sentence_list[i].split()

# a list to store unique important words
unique_words=[]

# lemmitizing , stemming and removing stopwords and digits from words lists
for i in range(0,len(change_sentence_list)):
    b=[]
    for j in range(0,len(change_sentence_list[i])):
        change_sentence_list[i][j]=change_sentence_list[i][j].lower()

        #change_sentence_list[i][j]=stemmer.stem(change_sentence_list[i][j])

        change_sentence_list[i][j]=lemmatizer.lemmatize(change_sentence_list[i][j])

        if change_sentence_list[i][j] not in stopwords_list and change_sentence_list[i][j].isnumeric()!=True:
            b.append(change_sentence_list[i][j])
            unique_words.append(change_sentence_list[i][j])

    change_sentence_list[i]=b
```

Restricted Mode 0 8 Jupyter Server: Local Cell 1 of 15

File Edit Selection View Go Run Terminal Helpfinal text extraction.ipynb - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

final text extraction.ipynb X

C:\Users\> Dell > Downloads > final text extraction.ipynb > #import requests

+ Code + Markdown Run All Clear Outputs of All Cells Outline ...

Select Kernel

```
change_sentence_list[i]=b

#removing the words that are repeated by changin it into set
unique_words = list(set(unique_words))
unique_words.sort()
```

[51] Python

```
change_sentence_list = sentence_list.copy()
#creattig deep copy of sentence tokenizing list

for i in range(0,len(change_sentence_list)):
    change_sentence_list[i] = re.sub(r'^\w\s','',change_sentence_list[i])
# removing all punctuations

# stopwords list, lemmatizer and stemmer
stopwords_list=stopwords.words('english')
lemmatizer = WordNetLemmatizer()
#stemmer = PorterStemmer()

#splitting sentences into words and storing them in place of sentences as list of words
for i in range(0,len(change_sentence_list)):
    change_sentence_list[i] = change_sentence_list[i].split()

# a list to store unique important words
unique_words=[]

# lemmitizing , stemming and removing stopwords and digits from words lists
```

Restricted Mode 0 8 Jupyter Server: Local Cell 1 of 15

```
File Edit Selection View Go Run Terminal Help
final text extraction.ipynb - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

final text extraction.ipynb X
C:\Users> Dell > Downloads > final text extraction.ipynb > #import requests

+ Code + Markdown | Run All | Clear Outputs of All Cells | Outline ...
Select Kernel

#splitting sentences into words and storing them in place of sentences as list of words
for i in range(0,len(change_sentence_list)):
    change_sentence_list[i] = change_sentence_list[i].split()

# a list to store unique important words
unique_words=[]

# lemmitizing , stemming and removing stopwords and digits from words lists
for i in range(0,len(change_sentence_list)):
    b=[]
    for j in range(0,len(change_sentence_list[i])):
        change_sentence_list[i][j]=change_sentence_list[i][j].lower()

        #change_sentence_list[i][j]=stemmer.stem(change_sentence_list[i][j])

        change_sentence_list[i][j]=lemmatizer.lemmatize(change_sentence_list[i][j])

        if change_sentence_list[i][j] not in stopwords_list and change_sentence_list[i][j].isnumeric()!=True:
            b.append(change_sentence_list[i][j])
            unique_words.append(change_sentence_list[i][j])

    change_sentence_list[i]=b

#removing the words that are repeated by changin it into set
unique_words = list(set(unique_words))
unique_words.sort()

[52] Python

# BAG OF WORDS
bag_of_words={}
```

```
File Edit Selection View Go Run Terminal Help
final text extraction.ipynb - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

final text extraction.ipynb X
C:\Users> Dell > Downloads > final text extraction.ipynb > #import requests

+ Code + Markdown | Run All | Clear Outputs of All Cells | Outline ...
Select Kernel

# BAG OF WORDS
bag_of_words={}
for j in range(0,len(change_sentence_list)):
    b=[]
    for i in unique_words:
        if i in change_sentence_list[j]:
            b.append(change_sentence_list[j].count(i))
        else:
            b.append(0)
    bag_of_words[j] = b

bow_df=pd.DataFrame(bag_of_words,index=unique_words)
bow_df

[53] Python

...

# Tf-IDF
document_freq=[]
for i in range(0,len(unique_words)):
    b=0
    for j in change_sentence_list:
        if unique_words[i] in j:
            b=b+1
    document_freq.append(math.log(len(change_sentence_list)/b)+1)

tf_idf={}
for i in range(0,len(change_sentence_list)):
    tf_idf[i] = bow_df[i]*document_freq

tfidf_df = pd.DataFrame(tf_idf)
tfidf_df

[54] Python
```

File Edit Selection View Go Run Terminal Help

final text extraction.ipynb - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

final text extraction.ipynb X

C:\Users> Dell > Downloads > final text extraction.ipynb > #import requests

+ Code + Markdown | ▶ Run All | Clear Outputs of All Cells | Outline ...

Select Kernel

COSINE SIMILARITY MATRIX

sim_matrix={}for i in range(0,len(change_sentence_list)):b=[]for j in range(0,len(change_sentence_list)):a = np.dot(tfidf_df[i],tfidf_df[j])a = a/(np.linalg.norm(tfidf_df[i])*np.linalg.norm(tfidf_df[j]))b.append(a)sim_matrix[i]=bsim_matrix=pd.DataFrame(sim_matrix)sim_matrix

[55] Python

...

sent_ranks={}for i in range(0,len(change_sentence_list)):a=0for j in range(0,len(unique_words)):a=a+bow_df[i][j]sent_ranks[i] = aprint(sent_ranks)

[56] Python

... {0: 15, 1: 6, 2: 3, 3: 12, 4: 3, 5: 3, 6: 12, 7: 8, 8: 7, 9: 8, 10: 4, 11: 16, 12: 6, 13: 7, 14: 12}

Restricted Mode 0 8

Jupyter Server: Local Cell 1 of 15

Type here to search

File Edit Selection View Go Run Terminal Help

final text extraction.ipynb - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

final text extraction.ipynb X

C:\Users> Dell > Downloads > final text extraction.ipynb > #import requests

+ Code + Markdown | ▶ Run All | Clear Outputs of All Cells | Outline ...

Select Kernel

[55] Python

...

sent_ranks={}for i in range(0,len(change_sentence_list)):a=0for j in range(0,len(unique_words)):a=a+bow_df[i][j]sent_ranks[i] = aprint(sent_ranks)

[56] Python

... {0: 15, 1: 6, 2: 3, 3: 12, 4: 3, 5: 3, 6: 12, 7: 8, 8: 7, 9: 8, 10: 4, 11: 16, 12: 6, 13: 7, 14: 12}

▶

sent_ranks= sorted(sent_ranks.items(), key=lambda x: x[1],reverse=True)print(sent_ranks)imp_sent=[]for i in range(0,5):imp_sent.append(sent_ranks[i][0])imp_sent.sort()for i in range(0,5):print("\n",sentence_list[imp_sent[i]])

[57] Python

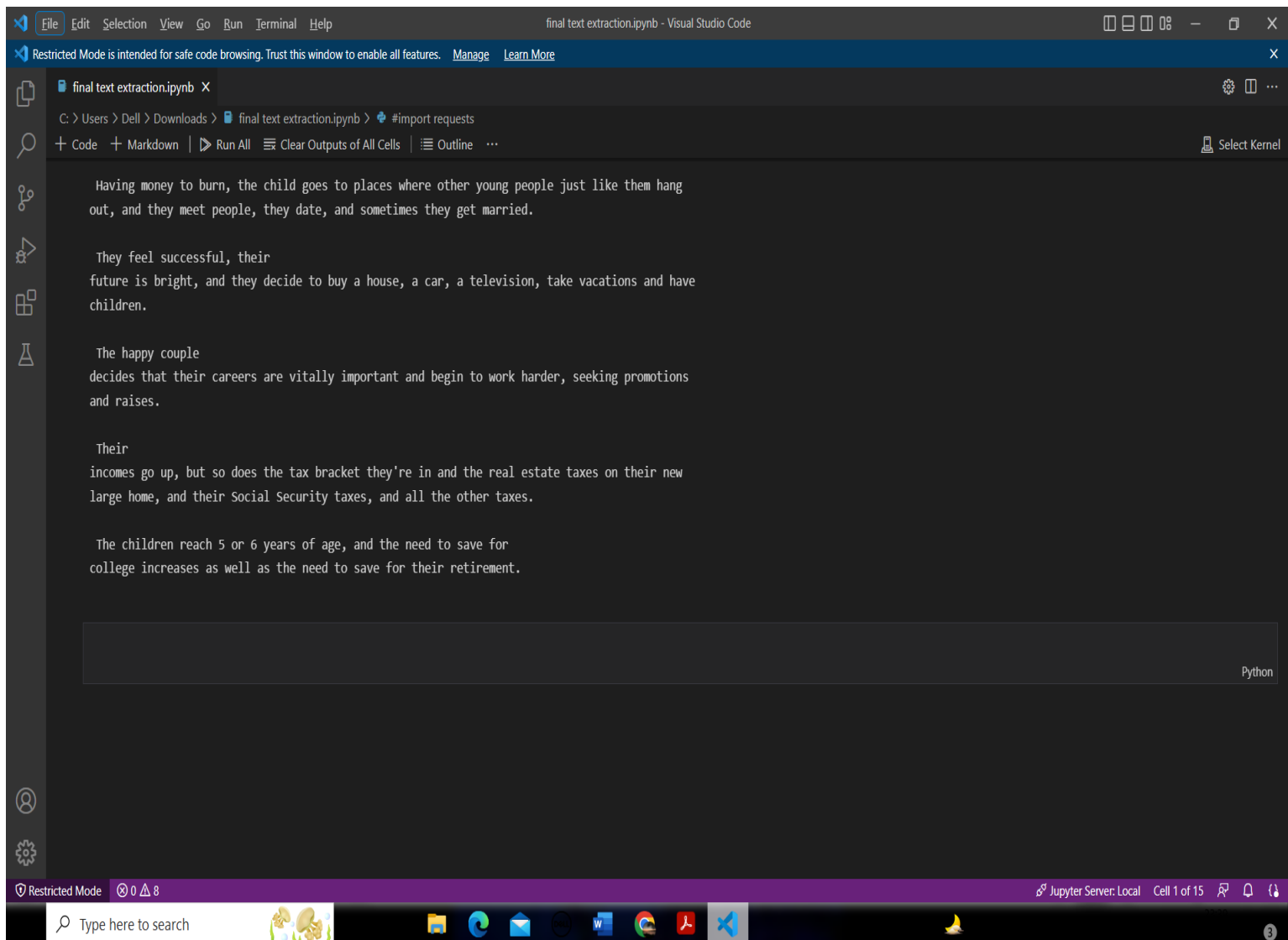
... [(11, 16), (0, 15), (3, 12), (6, 12), (14, 12), (7, 8), (9, 8), (8, 7), (13, 7), (1, 6), (12, 6), (10, 4), (2, 3), (4, 3), (5, 3)]

Restricted Mode 0 8

Jupyter Server: Local Cell 1 of 15

Type here to search

4.3 Result



The screenshot shows a Jupyter Notebook titled "final text extraction.ipynb" in Visual Studio Code. The notebook is in "Restricted Mode" and contains a story about a child and their family. The story is written in a simple, conversational style. The text is as follows:

```
Having money to burn, the child goes to places where other young people just like them hang out, and they meet people, they date, and sometimes they get married.
```

```
They feel successful, their future is bright, and they decide to buy a house, a car, a television, take vacations and have children.
```

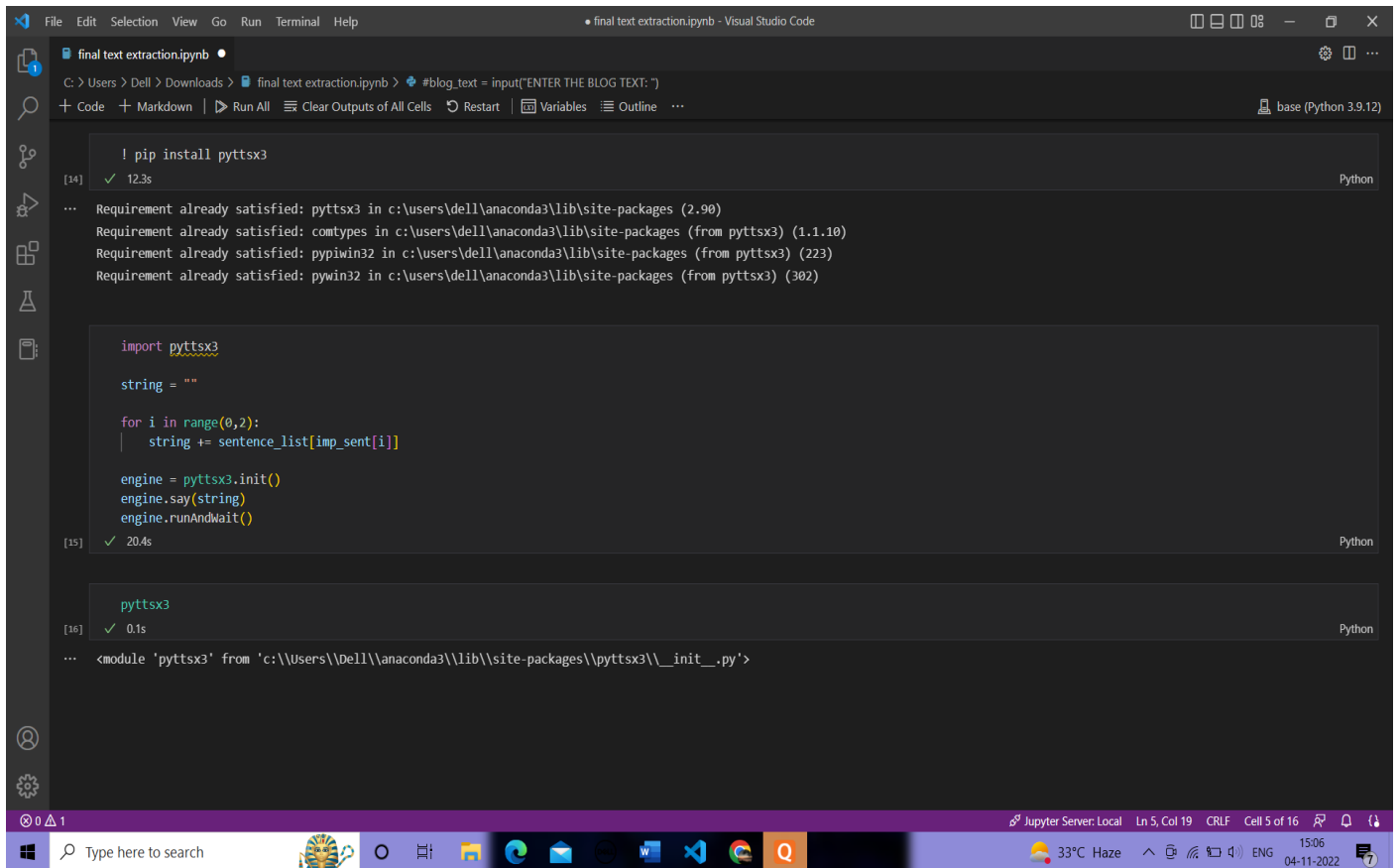
```
The happy couple decides that their careers are vitally important and begin to work harder, seeking promotions and raises.
```

```
Their incomes go up, but so does the tax bracket they're in and the real estate taxes on their new large home, and their Social Security taxes, and all the other taxes.
```

```
The children reach 5 or 6 years of age, and the need to save for college increases as well as the need to save for their retirement.
```

The notebook is running on a Python kernel. The status bar at the bottom indicates "Restricted Mode" and "Jupyter Server: Local". The taskbar at the bottom shows various application icons, including a search bar, a folder icon, a blue circle icon, a blue envelope icon, a blue square icon, a red square icon, and a blue square icon.

Output of Audio Converter



The screenshot shows a Jupyter Notebook titled 'final text extraction.ipynb' running in Visual Studio Code. The notebook is using the 'base (Python 3.9.12)' kernel. The first cell (index 14) contains the command `! pip install pytsx3`, which executed successfully in 12.3 seconds. The output shows that the requirements for `pytsx3` are already satisfied: `comtypes` (1.1.10), `pypiwin32` (223), and `pywin32` (302). The second cell (index 15) contains Python code that imports `pytsx3`, initializes an engine, and processes a list of sentences. It executed successfully in 20.4 seconds. The third cell (index 16) contains the command `pytsx3`, which executed successfully in 0.1 seconds, returning the module path: `<module 'pytsx3' from 'c:\\Users\\Dell\\anaconda3\\lib\\site-packages\\pytsx3__init__.py'>`.

```
! pip install pytsx3
```

```
[14] ✓ 12.3s Python
```

```
... Requirement already satisfied: pytsx3 in c:\users\dell\anaconda3\lib\site-packages (2.90)
Requirement already satisfied: comtypes in c:\users\dell\anaconda3\lib\site-packages (from pytsx3) (1.1.10)
Requirement already satisfied: pypiwin32 in c:\users\dell\anaconda3\lib\site-packages (from pytsx3) (223)
Requirement already satisfied: pywin32 in c:\users\dell\anaconda3\lib\site-packages (from pytsx3) (302)
```

```
import pytsx3

string = ""

for i in range(0,2):
    string += sentence_list[imp_sent[i]]

engine = pytsx3.init()
engine.say(string)
engine.runAndWait()
```

```
[15] ✓ 20.4s Python
```

```
pytsx3
```

```
[16] ✓ 0.1s Python
```

```
... <module 'pytsx3' from 'c:\\Users\\Dell\\anaconda3\\lib\\site-packages\\pytsx3\\__init__.py'>
```

Jupyter Server: Local Ln 5, Col 19 CRLF Cell 5 of 16 15:06 04-11-2022

Chapter 5 Summary and Conclusion

Summary

This survey emphasizes extractive approaches to summarization using statistical methods. A distinction has been made between single document and multi document summarization. Since a lot of interesting work is being done far from the mainstream research in this field, we have chosen to include a brief discussion on some methods that we found relevant to future research, even if they focus only on small details related to a general summarization process and not on building an entire summarization system.

CONCLUSION

Text summarization is an interesting machine learning field that is increasingly gaining attraction. As research in this area continues, we can expect to see breakthroughs that will assist in fluently and accurately shortening long text documents. Hereby, We can say we have successfully completed text summarization using NLP as per problem statement with efficiency. By this project we have solved the problem by the summaries of the text to gain information. We have tried our best to make these summaries as important as possible in the aspect of text intention .We can also integrate features like the voice text acceptance for the text summarization. Example, someone reads out loud the text paragraph from the newspaper or passage from novel which is difficult to understand and needs to be summarized. We have certain limitation while dealing with punctuation marks and spaces so in future we will try to make it as proper as possible.

References

Journals / Conference Papers:

[1] Sinha, Aakash, Abhishek Yadav, and Akshay Gahlot. "Extractive text summarization using neural networks." arXiv preprint arXiv:1802.10137 (2018).

[2] Peter J. Liu, and Christopher D. Manning. "Get to the point: Summarization with pointer-generator networks." arXiv preprint arXiv:1704.04368 (2017).

[3] Liu, Linqing, et al. "Generative adversarial network for abstractive text summarization." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.

[4] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." Journal of machine Learning research 3.Jan (2003): 993-1022.

Weblinks:

[5] Comprehensive Guide to Text Summarization using Deep Learning in Python. -
<https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guidetext-summarization-using-deep-learning-python/>

[6] Text Summarization using Machine Learning. =
<https://data-flair.training/blogs/machine-learning-text-summarization/>

[7] Approaches to Text Summarization: An Overview. =
<https://www.kdnuggets.com/2019/01/approaches-text-summarizationoverview.html>

Virender dehu, Pradeep kumar Tiwari, Gaurav Aggarwal, Bhavya joshi and pawan kartik "Text summarization technique and application" (Manipal University Jaipur, india).

https://www.researchgate.net/publication/350081502_Text_Summarization_Techniques_and_Applications

<https://in.search.yahoo.com/search?fr=mcafee&type=E211IN826G0&p=greesforgreek>