TAMIL TEXT SUMMARIZATION

AN EXTRACTIVE APPROACH

By: MYDEEN K M- 2021103717

1.Introduction:

In today's digital era, Natural Language Processing (NLP) stands as a pivotal technology, revolutionizing how we interact with textual data across various domains. Among its diverse applications, text summarization emerges as a critical tool, enabling the distillation of extensive texts into concise and informative summaries. This project focuses on the specific challenge of Tamil text summarization, recognizing the linguistic richness and cultural significance of the Tamil language. By leveraging advanced NLP techniques, we aim to facilitate efficient comprehension and dissemination of Tamil textual content, thereby bridging the gap between cutting-edge technology and linguistic heritage.

2. Problem Statement:

Tamil text summarization aims to condense lengthy Tamil documents into shorter, coherent summaries while retaining the key information. This project focuses on developing an extractive approach for Tamil text summarization, which involves selecting and extracting the most relevant sentences from the original text. The challenge lies in accurately identifying and extracting these critical sentences from Tamil texts, ensuring that the summary remains meaningful and contextually accurate. The goal is to create an effective and reliable tool that can automatically generate summaries, thereby aiding users in quickly understanding the main points of extensive Tamil documents.

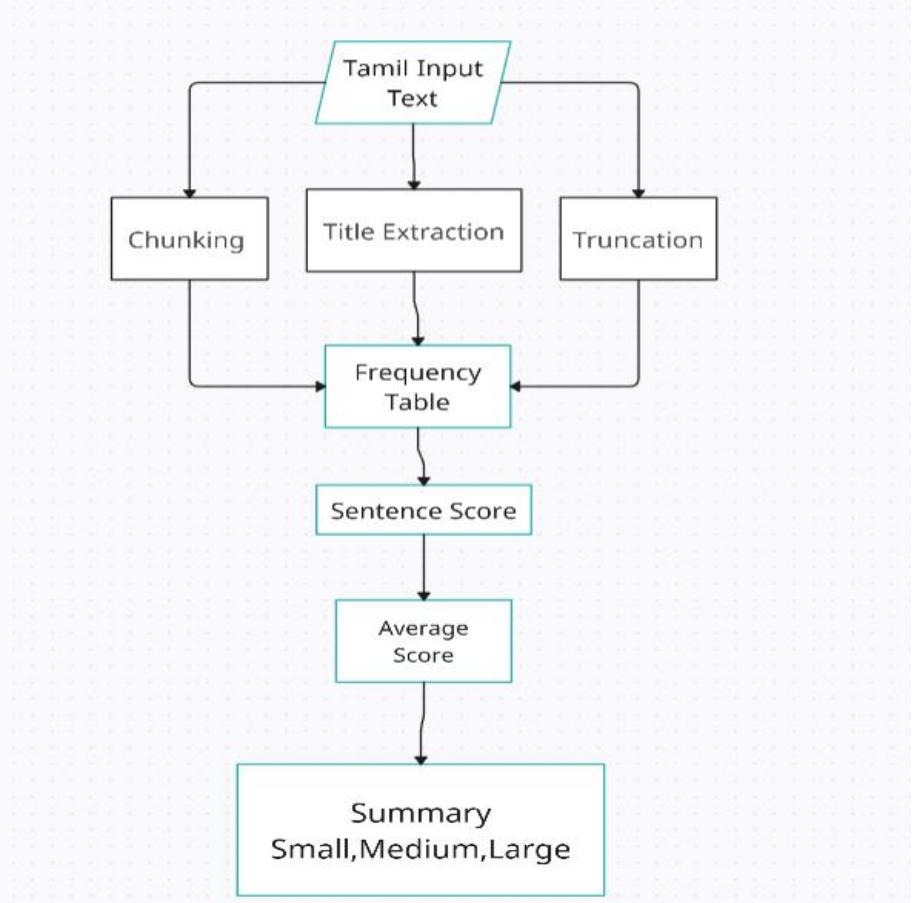
2.1 Objectives:

- Develop a robust NLP model tailored for Tamil text summarization.
- Address the linguistic nuances and challenges specific to Tamil language summarization.
- Explore and implement state-of-the-art algorithms and techniques in NLP for optimal summarization results.
- Evaluate the performance of the developed model through comprehensive metrics and comparisons with existing methods.
- Provide a tool or platform for Tamil speakers to summarize and extract key information from Tamil texts efficiently.

3 .Proposed System:

The proposed system adopts an extractive approach for Tamil text summarization, enabling the generation of small, medium, and large summaries from input text. Through advanced Natural Language Processing (NLP) techniques, the system preprocesses the Tamil text, identifies key sentences or passages, and selects the most relevant ones to construct summaries. Users can choose their preferred summary length, with small summaries providing concise overviews, medium summaries offering additional detail, and large summaries encompassing comprehensive insights. The system's intuitive interface facilitates seamless interaction, empowering users to efficiently access summarized Tamil content tailored to their specific information needs and preferences.

4 .Block Diagram:



5. Methodology:

Our approach to Tamil text summarization follows an extractive methodology, leveraging various techniques tailored for the language's unique characteristics. We utilize the Dinamalar news dataset as our primary source of input text, containing a diverse range of articles in Tamil. Additionally, we incorporate a list of Tamil stopwords from an external text file to aid in the removal of irrelevant terms during preprocessing.

5.1 Dataset:

The Dinamalar news dataset serves as the foundation for our training and evaluation processes. This dataset encompasses a wide array of news articles written in Tamil, offering a rich source of textual data for our summarization task. By utilizing real-world news articles, we ensure that our system can effectively summarize relevant and timely content, reflecting the nuances of contemporary Tamil discourse.

```
import glob
txt_files = glob.glob("C:\\Users\\MYDEEN K M\\Desktop\\mydeen\\NLP_Project\\dataset\\Dina\\*.txt")
```

```
ta_txt_files = txt_files
```

```
print("Number of tamil text docs : ", len(ta_txt_files))
```

```
Number of tamil text docs :  80136
```

```
stop_words = list()
with open("C:\\Users\\MYDEEN K M\\Desktop\\mydeen\\NLP_Project\\TamilStopWords (1).txt", 'r', encoding='utf-8') as f:
    stop_words += f.readlines()

for i in range(len(stop_words)):
    stop_words[i] = stop_words[i].rstrip("\n")
```

```
print(stop_words)
```

```
['ஒரு', 'என்று', 'மற்றும்', 'இந்த', 'இது', 'என்ற', 'கொண்டு', 'என்பது', 'பல', 'ஆகும்', 'அல்லது', 'அவர்', 'நான்', 'உள்ள', 'அந்த', 'இவ
ர்', 'என', 'முதல்', 'என்ன', 'இருந்து', 'சில', 'என்', 'போன்ற', 'வேண்டும்', 'வந்து', 'இதன்', 'அது', 'அவன்', 'தான்', 'பலரும்', 'என்னும்',
'மேலும்', 'பின்னர்', 'கொண்ட', 'இருக்கும்', 'தனது', 'உள்ளது', 'போது', 'என்றும்', 'அதன்', 'தன்', 'பிறகு', 'அவர்கள்', 'வரை', 'அவள்',
'நீ', 'ஆகிய', 'இருந்தது', 'உள்ளன', 'வந்த', 'இருந்த', 'மிகவும்', 'இங்கு', 'மீது', 'ஓர்', 'இவை', 'இந்தக்', 'பற்றி', 'வரும்', 'வேறு', 'இரு',
'இதில்', 'போல்', 'இப்போது', 'அவரது', 'மட்டும்', 'இந்தப்', 'எனும்', 'மேல்', 'பின்', 'சேர்ந்த', 'ஆகியோர்', 'எனக்கு', 'இன்னும்', 'அந்தப்',
'அன்று', 'ஒரே', 'மிக', 'அங்கு', 'பல்வேறு', 'விட்டு', 'பெரும்', 'அதை', 'பற்றிய', 'உன்', 'அதிக', 'அந்தக்', 'பேர்', 'இதனால்', 'அவை',
'அதே', 'ஏன்', 'முறை', 'யார்', 'என்பதை', 'எல்லாம்', 'மட்டுமே', 'இங்கே', 'அங்கே', 'இடம்', 'இடத்தில்', 'அதில்', 'நாம்', 'அதற்கு', 'என
வே', 'பிற', 'சிறு', 'மற்ற', 'விட', 'எந்த', 'எனவும்', 'எனப்படும்', 'எனினும்', 'அடுத்த', 'இதனை', 'இதை', 'கொள்ள', 'இந்தத்', 'இதற்கு',
'அதனால்', 'தவிர', 'போல', 'வரையில்', 'சற்று', 'எனக்']
```

```
display(ta_text_df.head())
```

	Text
	மின் தடையை கண்டித்து கிராம மக்கள் மறியல் : போலீஸ் தடியடியால் பதட்டம்\tபுதுக்கோட்டை : மின்தடையை கண்டித்து மறியல் போராட்டத்தில் ஈடுபட முயன்ற கிராம மக்கள் மீது, போலீசார் திடீரென தடியடி நடத்தினர். இதில் ஊராட்சி தலைவர் உட்பட 20 பேர் காயமடைந்தனர். ஆவேசமடைந்த கிராம மக்கள், போலீசார் மீது கல்வீசி எதிர் தாக்குதல் நடத்தியதில் போலீசார் ஐந்து பேர் காயமடைந்தனர்.புதுக்கோட்டை மாவட்டம், பொன்னமராவதி மற்றும் சுற்றுவட்டார கிராமப் பகுதிகளில் கடந்த ஒரு மாதமாக அடிக்கடி மின்தடை ஏற்படுகிறது. முன்னறிவிப்பின்றி மின்தடை செய்வதால் விவசாயிகள், வியாபாரிகள், மாணவர்கள் என அனைத்து தரப்பு மக்களும் மிகுந்த பாதிப்புக்குள்ளாகி வருகின்றனர். மின்தடை மற்றும் மின்னழுத்த குறைபாடுகளால் ஊராட்சிகளில் குடிநீர் வினியோகம் ஸ்தம்பித்துள்ளது.விவசாயப் பணிகளுக்கு நாள்தோறும் மூன்று மணி நேரம் கூட மின்சாரம் வழங்கப்படுவதில்லை. இதனால், அப்பகுதிகளில் சாகுபடி செய்யப்பட்டுள்ள பயிர்கள் கருகும் நிலையில் உள்ளன. பொறுமை இழந்த விவசாயிகள், பொன்னமராவதி மின்வாரிய கிளை அலுவலக உதவி செயற்பொறியாளரை நேரில் தொடர்புகொண்டு விவசாயப் பணிகளுக்கு தடையின்றி மின்சாரம் வழங்கவேண்டும். முடியாத பட்சத்தில் முன்னறிவிப்புடன் மின்தடை செய்யவேண்டும் என கோரிக்கை வைத்தனர்.இதை ஏற்ற மின்வாரிய அதிகாரிகள், முன்னறிவிப்புடன் மட்டுமே மின்தடை செய்வதாக உறுதியளித்தனர். இருந்தும் அப்பகுதிகளில் முன்னறிவிப்பின்றி அடிக்கடி மின் வினியோகம் தடை செய்யப்பட்டு வருகிறது. இதர நேரங்களில் மின்னழுத்த குறைபாடுகள் காரணமாக மின் மோட்டார்களை இயக்க முடியாத சூழ்நிலை விவசாயிகளுக்கு ஏற்பட்டுள்ளது.மின் வாரியத்தின் இத்தகைய அலட்சியப் போக்கை கண்டித்தும், தடையற்ற மின்சாரம் வழங்கக் கோரியும் பொன்னமராவதி மற்றும் ஆலவயல், நகரப்பட்டி, கண்டியாநத்தம், அம்மாப்பட்டி உள்ளிட்ட ஏழு கிராமங்களைச் சேர்ந்த விவசாயிகள் மற்றும் பொதுமக்கள் 1,000க்கும் மேற்பட்டோர் பொன்னமராவதி தாலுகா அலுவலகம் முன் சாலை மறியல் போராட்டத்தில் ஈடுபட முடிவு செய்தனர்.இதற்காக நேற்று காலை 11 மணிக்கு கொப்பனாப்பட்டி சந்திப்பில் குவிந்தனர். இதனால், அப்பகுதியில்

5.2 Loading Stop Words:

```
stop_words = list()
with open("C:\\Users\\MYDEEN K M\\Desktop\\mydeen\\NLP_Project\\TamilStopWords (1).txt", 'r', encoding='utf-8') as f:
    stop_words += f.readlines()

for i in range(len(stop_words)):
    stop_words[i] = stop_words[i].rstrip("\n")
```

```
print(stop_words)
```

```
['ஒரு', 'என்று', 'மற்றும்', 'இந்த', 'இது', 'என்ற', 'கொண்டு', 'என்பது', 'பல', 'ஆகும்', 'அல்லது', 'அவர்', 'நான்', 'உள்ள', 'அந்த', 'இவ
ர்', 'என', 'முதல்', 'என்ன', 'இருந்து', 'சில', 'என்', 'போன்ற', 'வேண்டும்', 'வந்து', 'இதன்', 'அது', 'அவன்', 'தான்', 'பலரும்', 'என்னும்',
'மேலும்', 'பின்னர்', 'கொண்ட', 'இருக்கும்', 'தனது', 'உள்ளது', 'போது', 'என்றும்', 'அதன்', 'தன்', 'பிறகு', 'அவர்கள்', 'வரை', 'அவள்',
'நீ', 'ஆகிய', 'இருந்தது', 'உள்ளன', 'வந்த', 'இருந்த', 'மிகவும்', 'இங்கு', 'மீது', 'ஓர்', 'இவை', 'இந்தக்', 'பற்றி', 'வரும்', 'வேறு', 'இரு',
'இதில்', 'போல்', 'இப்போது', 'அவரது', 'மட்டும்', 'இந்தப்', 'எனும்', 'மேல்', 'பின்', 'சேர்ந்த', 'ஆகியோர்', 'எனக்கு', 'இன்னும்', 'அந்தப்',
'அன்று', 'ஒரே', 'மிக', 'அங்கு', 'பலவேறு', 'விட்டு', 'பெரும்', 'அதை', 'பற்றிய', 'உன்', 'அதிக', 'அந்தக்', 'பேர்', 'இதனால்', 'அவை',
'அதே', 'ஏன்', 'முறை', 'யார்', 'என்பதை', 'எல்லாம்', 'மட்டுமே', 'இங்கே', 'அங்கே', 'இடம்', 'இடத்தில்', 'அதில்', 'நாம்', 'அதற்கு', 'என
வே', 'பிற', 'சிறு', 'மற்ற', 'விட', 'எந்த', 'எனவும்', 'எனப்படும்', 'எனினும்', 'அடுத்த', 'இதனை', 'இதை', 'கொள்ள', 'இந்தத்', 'இதற்கு',
'அதனால்', 'தவிர', 'போல', 'வரையில்', 'சற்று', 'எனக்']
```

5.3 Import NLTK tool:

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
```

5.4 Algorithms and Techniques:

Chunking: We employ chunking to segment the input text into meaningful units, such as sentences or paragraphs, facilitating subsequent analysis and summarization.

```
txt = ""
word_count = 0
for line in sample_text.split('\n'):  # Iterate over each line of the text
    if word_count >= 500:
        word_count = 0
        if txt != "":
            ta_text.append(txt)
            txt = ""
        continue

    if not line.strip():  # Skip empty lines
        continue

    d = line.strip()
    word_count += len(d.split())
    txt += d + " "  # Assuming there are no <doc> tags to strip

# Append any remaining text
if txt:
    ta_text.append(txt)
```

Title Extraction: Recognizing the significance of article titles in summarization, we extract titles from the input text to serve as potential summary candidates or guiding elements.

```
# Now, ta_text contains the segments of the text with approximately 448 words each.
def extract_title(text):
    # Find the index of the first ":"
    title_end_index = text.find(":")

    # Extract the title from the text
    title = text[:title_end_index].strip()

    return title

# Extract the title
title = extract_title(sample_text)

# Display the title
print("Title:", title)
```

```
Title: TN +2 Result 2024 Live
```

Truncation: In cases where the input text exceeds the desired summary length, we truncate the content to ensure that the generated summary remains concise and coherent.

```
def truncate(x, n_tokens = 448):
    tokens = x.split()
    if len(tokens) > n_tokens:
        return " ".join(tokens[:n_tokens])
    return x
```

Frequency Table: To identify key terms and phrases within the text, we construct a frequency table that captures the occurrence of words and phrases across the dataset. This allows us to prioritize essential content during the summarization process.

```python
def frequency_table(text) -> dict:
    words = word_tokenize(text)
    ps = PorterStemmer()

    freq_table = dict()
    for word in words:
        word = ps.stem(word)
        if word in stop_words:
            continue
        elif word in freq_table:
            freq_table[word] += 1
        else:
            freq_table[word] = 1
    return freq_table
```

Sentence Score and Average Score: Each sentence's relevance and importance are determined based on various factors, including frequency of key terms, position within the text, and proximity to important phrases. These scores are aggregated to compute an average score for each sentence, guiding the selection of sentences for inclusion in the summary.

```python
def score_sentences(tokenized_sentences, freq_table) -> dict:
    sentence_scores = dict()
    for sent in tokenized_sentences:
        word_count_in_sentence = (len(word_tokenize(sent)))
        word_count_in_sentence_wo_sw = 0    # Without stopwords
        for word in freq_table:
            if word in sent.lower():
                word_count_in_sentence_wo_sw += 1
                if sent in sentence_scores:
                    sentence_scores[sent] += freq_table[word]
                else:
                    sentence_scores[sent] = freq_table[word]
        if sent in sentence_scores:
            sentence_scores[sent] = sentence_scores[sent] / word_count_in_sentence_wo_sw
    return sentence_scores
```

```python
def avg(sentence_scores) -> float:
    average = 0
    for key in sentence_scores:
        average += sentence_scores[key]
    # len(sentence_scores) becomes zero when stopwords
    # are removed and no word left in the doc
    average = average / (len(sentence_scores) + 1e-100) # Avoid zero div error
    return average
```

6. Summary Generator:

6.1 Summarization (summarize):

This function generates a summary by selecting sentences whose scores are above a certain threshold. It takes tokenized sentences, sentence scores, and a threshold as input. It iterates through each sentence, checks if its score is above the threshold, and if so, adds it to the summary.

```
def summarize(tokenized_sentences, sentence_scores, threshold) -> str:
    sentence_count = 0
    summary = ""
    for sent in tokenized_sentences:
        if (sent in sentence_scores
            and sentence_scores[sent] >= threshold and sentence_scores[sent] !=title ):
            summary += " " + sent
            sentence_count += 1
    return summary
```

6.2 Summarize Series :

This function summarizes a series of documents. It iterates through each document in the series, generates a frequency table for the document, tokenizes its sentences, calculates sentence scores, computes the average score, and generates a summary using the summarize function. It then collects these summaries into a list and returns a Data Frame containing the original documents and their summaries.

Overall, these functions work together to summarize a series of documents by scoring and selecting sentences based on their frequency of non-stop words.

```python
import pandas as pd
from nltk.tokenize import sent_tokenize, word_tokenize
from rouge import Rouge

def summarize_series(text_series) -> pd.DataFrame:
    print(f"Total number of docs: {text_series.shape[0]}")
    summarized_text_small = []
    summarized_text_medium = []
    summarized_text_large = []

    rouge = Rouge()

    for i in range(text_series.shape[0]):
        freq_table = frequency_table(text_series[i])
        tokenized_sentences = sent_tokenize(text_series[i])
        sentence_scores = score_sentences(tokenized_sentences, freq_table)

        avg_score = avg(sentence_scores)
        small_threshold = avg_score + 0.3 * avg_score
        medium_threshold = avg_score
        large_threshold = avg_score - 0.3 * avg_score

        summary_small = summarize(tokenized_sentences, sentence_scores, small_threshold)
        summary_medium = summarize(tokenized_sentences, sentence_scores, medium_threshold)
        summary_large = summarize(tokenized_sentences, sentence_scores, large_threshold)

        summarized_text_small.append(summary_small)
        summarized_text_medium.append(summary_medium)
        summarized_text_large.append(summary_large)

        # Calculate ROUGE scores
        scores_small = rouge.get_scores(summary_small, text_series[i])
        scores_medium = rouge.get_scores(summary_medium, text_series[i])
        scores_large = rouge.get_scores(summary_large, text_series[i])

        print(f"Document {i}:")
        print("ROUGE Scores for Small Summary:")
        print(scores_small)
        print("ROUGE Scores for Medium Summary:")
        print(scores_medium)
        print("ROUGE Scores for Large Summary:")
        print(scores_large)

        # Print number of words in each summary
        small_word_count = sum(len(word_tokenize(sentence)) for sentence in summary_small.split())
        medium_word_count = sum(len(word_tokenize(sentence)) for sentence in summary_medium.split())
        large_word_count = sum(len(word_tokenize(sentence)) for sentence in summary_large.split())

        print(f"Small Summary Word Count: {small_word_count}")
        print(f"Medium Summary Word Count: {medium_word_count}")
        print(f"Large Summary Word Count: {large_word_count}")

        print(f"Number of docs summarized: {i}", end='\r')

    print("<" + "-" * 40 + " Completed summarizing " + "-" * 40 + ">")

    # Create DataFrame with all three summaries and ROUGE scores
    summary_df = pd.DataFrame({
        "Original": text_series,
        "Summary_Small": summarized_text_small,
        "Summary_Medium": summarized_text_medium,
        "Summary_Large": summarized_text_large
    })

    return summary_df
```

7. Evaluation Metrics:

7.1 ROUGE Score for Text Summarization:

The ROUGE score is a metric used to evaluate the quality of text summaries by comparing them to reference (human-created) summaries. It measures how much the machine-generated summary overlaps with the reference summary in terms of words and phrases.

Key Types of ROUGE:

ROUGE-N:

ROUGE-1: Measures the overlap of single words (unigrams).

ROUGE-2: Measures the overlap of two-word sequences (bigrams).

ROUGE-L: Focuses on the longest common subsequence (LCS) between the generated and reference summaries. This captures the longest matching sequence of words in the same order.

7.2 Key Metrics:

Precision: How much of the generated summary matches the reference summary.

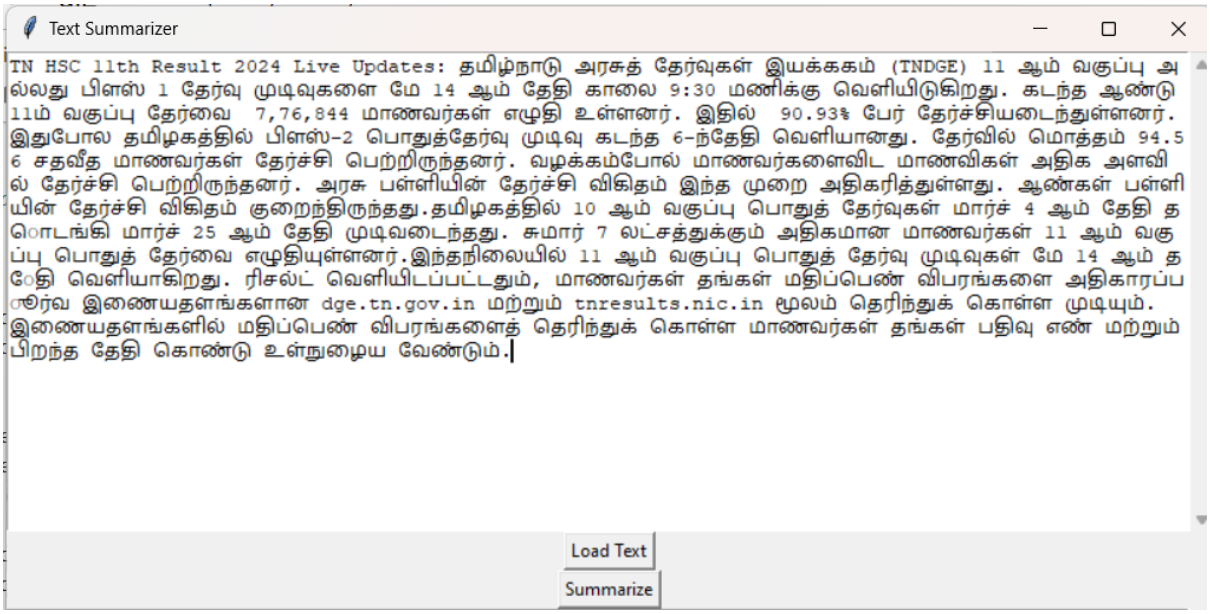
Recall: How much of the reference summary is captured in the generated summary.

F1 Score: The harmonic mean of precision and recall, providing a balanced measure.

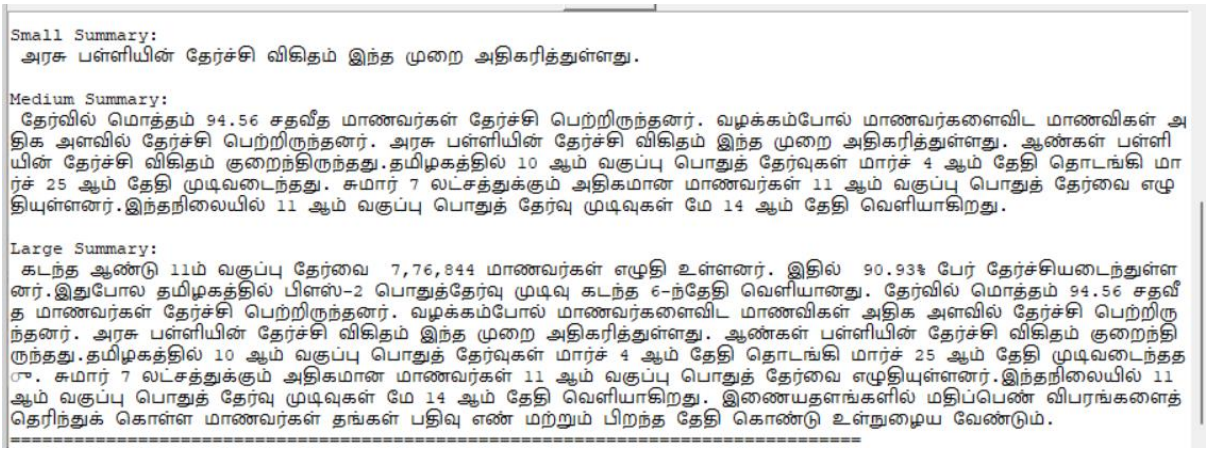
```
# Calculate ROUGE scores
scores_small = rouge.get_scores(summary_small, text_series[i])
scores_medium = rouge.get_scores(summary_medium, text_series[i])
scores_large = rouge.get_scores(summary_large, text_series[i])
```

8. Results:

Example Input text: (About 11th results)



8.1 Output:

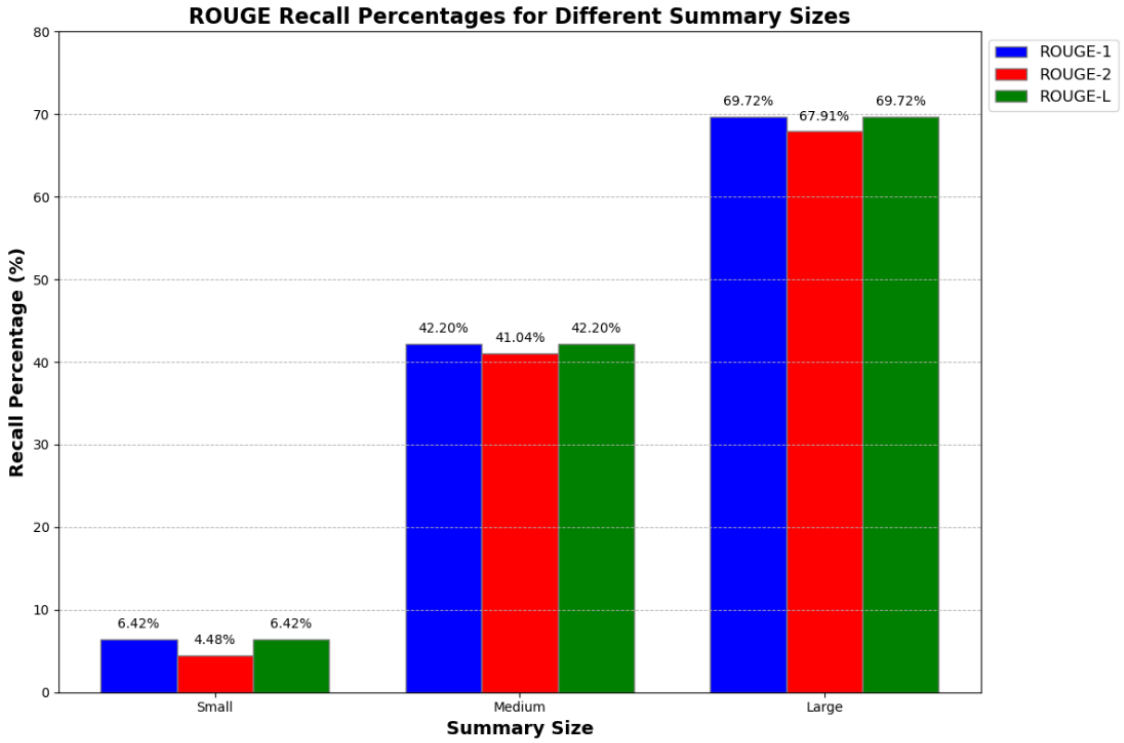


8.2 Rouge Score Evaluation:

```
ta_text_summarized_df = summarize_series(ta_text_df["Text"])
```

```
Total number of docs: 1
Document 0:
ROUGE Scores for Small Summary:
[{'rouge-1': {'r': 0.06422018348623854, 'p': 1.0, 'f': 0.1206896540383472}, 'rouge-2': {'r': 0.04477611940298507, 'p': 1.0, 'f': 0.0857142848938775},
'rouge-l': {'r': 0.06422018348623854, 'p': 1.0, 'f': 0.1206896540383472}}]
ROUGE Scores for Medium Summary:
[{'rouge-1': {'r': 0.42201834862385323, 'p': 1.0, 'f': 0.5935483829227888}, 'rouge-2': {'r': 0.41044776119402987, 'p': 1.0, 'f': 0.5820105778841579}, 'ro
uge-l': {'r': 0.42201834862385323, 'p': 1.0, 'f': 0.5935483829227888}}]
ROUGE Scores for Large Summary:
[{'rouge-1': {'r': 0.6972477064220184, 'p': 1.0, 'f': 0.821621616780716}, 'rouge-2': {'r': 0.6791044776119403, 'p': 0.9891304347826086, 'f': 0.8053097296
859583}, 'rouge-l': {'r': 0.6972477064220184, 'p': 1.0, 'f': 0.821621616780716}}]
Small Summary Word Count: 8
Medium Summary Word Count: 68
Large Summary Word Count: 107
<====================================== Completed summarizing ======================================>
```

	Small Summary	Medium Summary	Large Summary
Rouge-1	6.42%	42.20%	69.72%
Rouge-2	4.48%	41.04%	67.91%
Rouge-L	6.42%	42.20%	69.72%



9. Conclusion:

Our methodical approach to Tamil text summarization, employing techniques such as frequency tables with stopword removal, sentence scoring, and threshold variation to generate small, medium, and large summaries, demonstrates a versatile and effective strategy. By fine-tuning the summarization process, you've shown an understanding of the nuanced balance between brevity and comprehensiveness, while also considering the linguistic complexities of the Tamil language. This systematic approach lays a strong foundation for further exploration and refinement in the field of Tamil text summarization.