# Video Summarization

Team 11

01

## Introduction

Here we take a look at what we aim to achieve

02

## Assumptions

Here we can see asumption we used in out work

03

## Algorithms

We Can Take A look at our Algorithm Choices

04

## Examples

A Video Summarized by All Our Chosen Algorithms

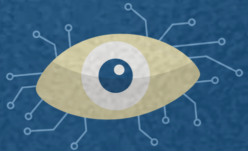
05

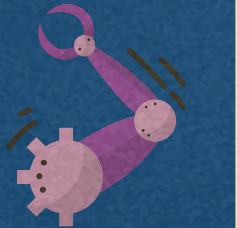
## Conclusion

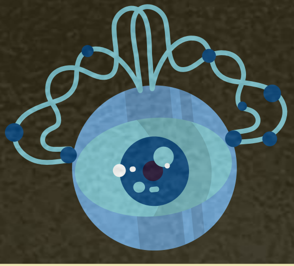
We have a look at what we achieved and concluded

# INTRODUCTION

Out Approch

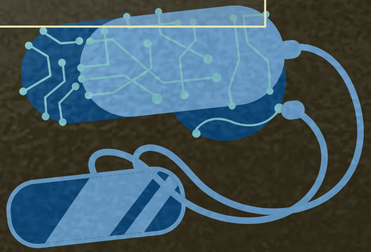






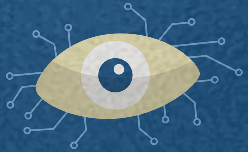
# Introduction

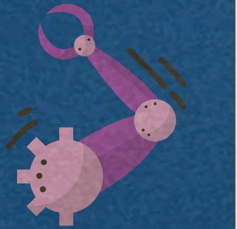
Our work focuses on the summarization of videos using extractive summarization methods by first transcribing them then applying a user chosen Natural Language Processing algorithm to get the final summary which can be chosen using our application.



# Assumptions

Prerequisites and asumptions

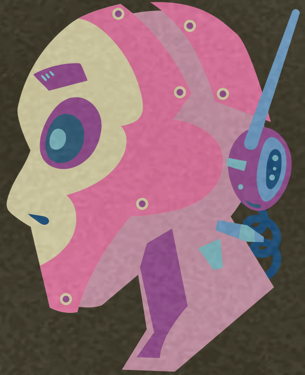


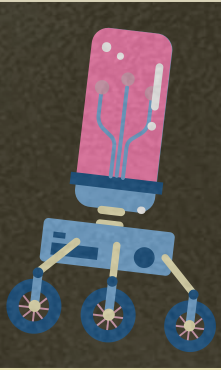


# Our Assumptions

## Summarization Method

There are two main types of summarization: Abstractive and Extractive. Here our aim is to generate a summary from already provided keywords, so extractive summarization is used.



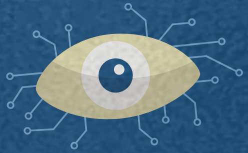


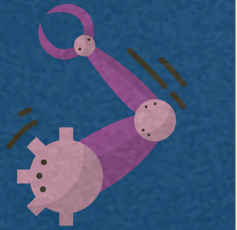
## Language Used

Videos are in the English Language. Algorithms are unfazed by accents.

# Algorithms

Our Chosen Algorithms





# Algorithms

## Pure NLTK

Algorithm using only nltk library's tokenizers

## Gensim

Algorithm using gensim library's TextRank based Algo

## Luhn

Algorithm using sumy library's Luhn Heuristic Algo

## LexRank

Algorithm using sumy library's LexRank based Algo

## KL-Sum

Algorithm using sumy library's KL-Sum based Algo

## Naïve Reduction

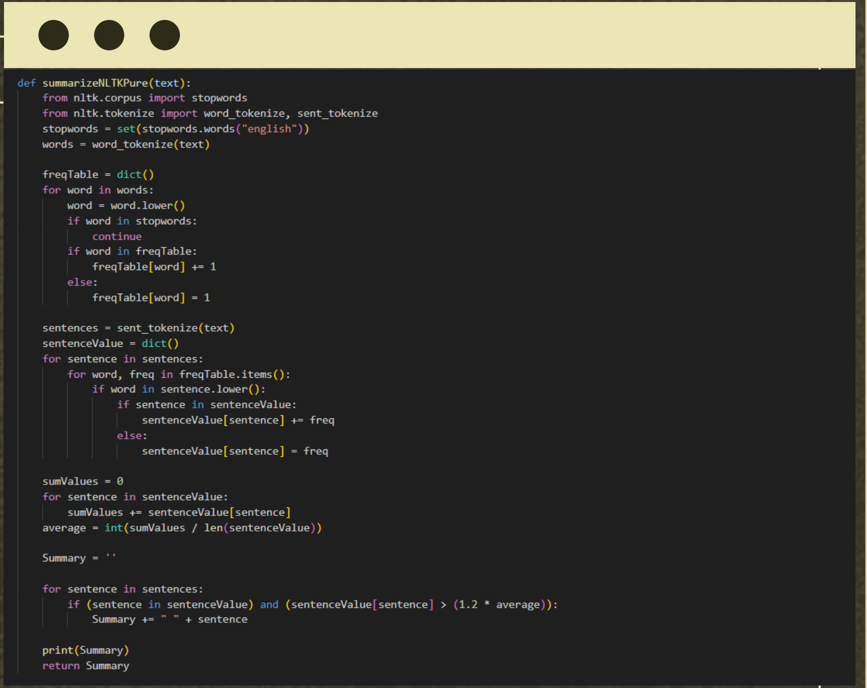
Algorithm using sumy library's Normal Reduction Algo

# Let's Dive in

Brief discreption of all our implemented Video Summarization Algorithms

# Pure NLTK Algorithm

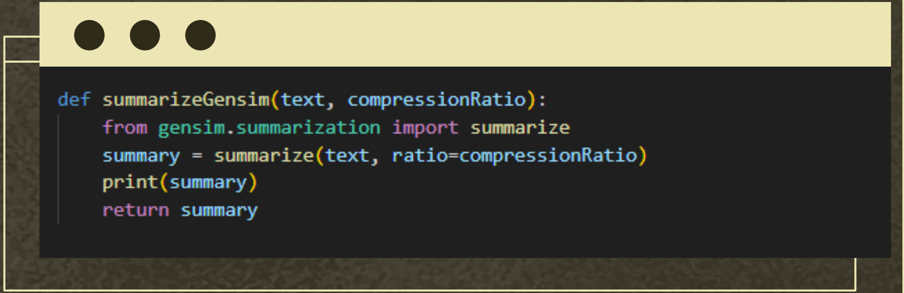
The Algorithm employs a straightforward approach by assigning importance scores to sentences based on the frequency of non-stop words.



```python
def summarizeNLTKPure(text):
    from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize, sent_tokenize
    stopwords = set(stopwords.words("english"))
    words = word_tokenize(text)

    freqTable = dict()
    for word in words:
        word = word.lower()
        if word in stopwords:
            continue
        if word in freqTable:
            freqTable[word] += 1
        else:
            freqTable[word] = 1

    sentences = sent_tokenize(text)
    sentenceValue = dict()
    for sentence in sentences:
        for word, freq in freqTable.items():
            if word in sentence.lower():
                if sentence in sentenceValue:
                    sentenceValue[sentence] += freq
                else:
                    sentenceValue[sentence] = freq

    sumValues = 0
    for sentence in sentenceValue:
        sumValues += sentenceValue[sentence]
    average = int(sumValues / len(sentenceValue))

    Summary = ''

    for sentence in sentences:
        if (sentence in sentenceValue) and (sentenceValue[sentence] > (1.2 * average)):
            Summary += " " + sentence

    print(Summary)
    return Summary
```

# Gensim Algorithm

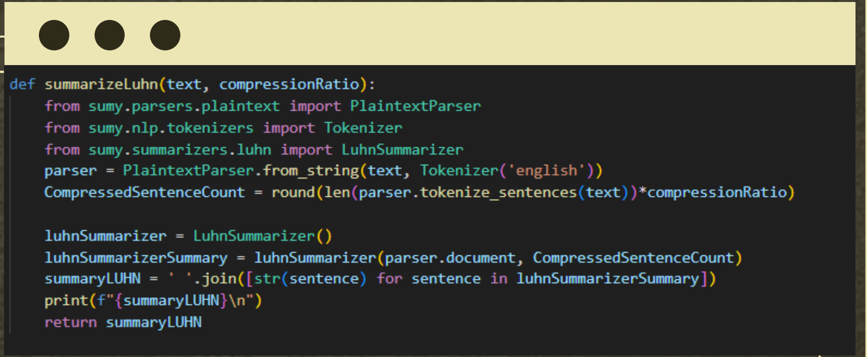
Summarization is done by representing words or sentences as points on a graph connected by relationships. Each point gets an initial score, and through a process of refining and ranking based on connections, the algorithm identifies the most important words or sentences



```python
def summarizeGensim(text, compressionRatio):
    from gensim.summarization import summarize
    summary = summarize(text, ratio=compressionRatio)
    print(summary)
    return summary
```

# Luhn Heuristic Algorithm

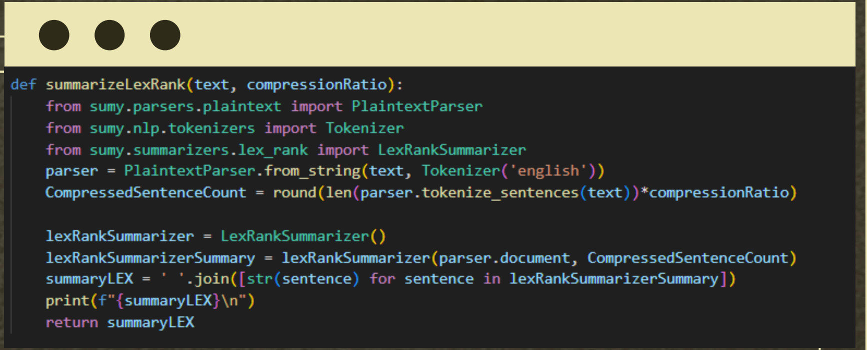
Summarization is done by giving importance to words that appear frequently. It uses a method called TF-IDF to weigh words and scores sentences based on concentration of important words. The scoring considers the number of important words in a sentence and their placement.



```
def summarizeLuhn(text, compressionRatio):
    from sumy.parsers.plaintext import PlaintextParser
    from sumy.nlp.tokenizers import Tokenizer
    from sumy.summarizers.luhn import LuhnSummarizer
    parser = PlaintextParser.from_string(text, Tokenizer('english'))
    CompressedSentenceCount = round(len(parser.tokenize_sentences(text))*compressionRatio)

    luhnSummarizer = LuhnSummarizer()
    luhnSummarizerSummary = luhnSummarizer(parser.document, CompressedSentenceCount)
    summaryLUHN = ' '.join([str(sentence) for sentence in luhnSummarizerSummary])
    print(f"{summaryLUHN}\n")
    return summaryLUHN
```

# LexRank Algorithm

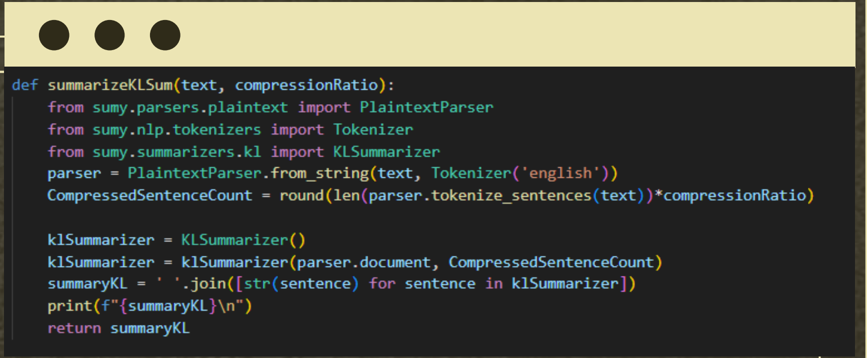
Summarization is done by measuring sentence importance based on how similar they are to others. and calculating scores using a method called power iteration.



```python
def summarizeLexRank(text, compressionRatio):
    from sumy.parsers.plaintext import PlaintextParser
    from sumy.nlp.tokenizers import Tokenizer
    from sumy.summarizers.lex_rank import LexRankSummarizer
    parser = PlaintextParser.from_string(text, Tokenizer('english'))
    CompressedSentenceCount = round(len(parser.tokenize_sentences(text))*compressionRatio)

    lexRankSummarizer = LexRankSummarizer()
    lexRankSummarizerSummary = lexRankSummarizer(parser.document, CompressedSentenceCount)
    summaryLEX = ' '.join([str(sentence) for sentence in lexRankSummarizerSummary])
    print(f"{summaryLEX}\n")
    return summaryLEX
```

# KL-Sum Algorithm

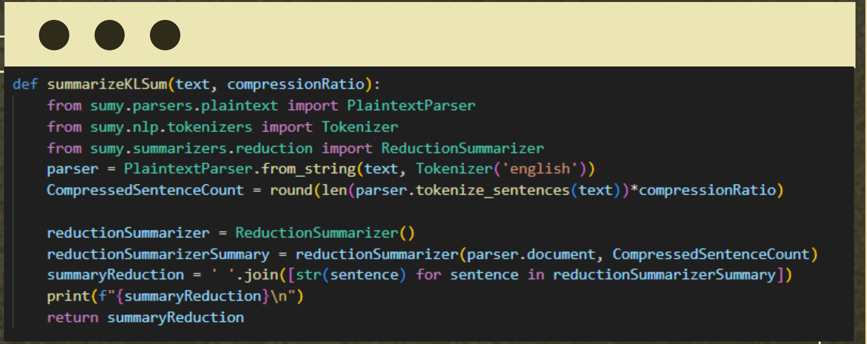
Summarization is done by trying to preserve the original document's word distribution. It uses KL Divergence to measure differences between distribution in summary and original and only chooses sentences that minimizes the difference



```python
def summarizeKLSum(text, compressionRatio):
    from sumy.parsers.plaintext import PlaintextParser
    from sumy.nlp.tokenizers import Tokenizer
    from sumy.summarizers.kl import KLSummarizer
    parser = PlaintextParser.from_string(text, Tokenizer('english'))
    CompressedSentenceCount = round(len(parser.tokenize_sentences(text))*compressionRatio)

    klSummarizer = KLSummarizer()
    klSummarizer = klSummarizer(parser.document, CompressedSentenceCount)
    summaryKL = ' '.join([str(sentence) for sentence in klSummarizer])
    print(f"{summaryKL}\n")
    return summaryKL
```

# Naïve Reduction Algorithm

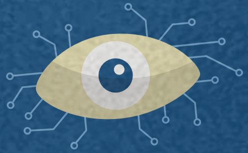
Summarization is done by rating sentences based on similarity between it and other sentence by counting common words, it then converts rating to weight and finally, selecting sentences with the highest to add to summary.

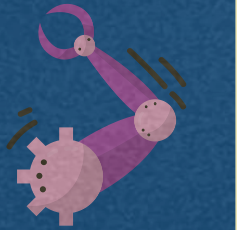


```python
def summarizeKLSum(text, compressionRatio):
    from sumy.parsers.plaintext import PlaintextParser
    from sumy.nlp.tokenizers import Tokenizer
    from sumy.summarizers.reduction import ReductionSummarizer
    parser = PlaintextParser.from_string(text, Tokenizer('english'))
    CompressedSentenceCount = round(len(parser.tokenize_sentences(text))*compressionRatio)

    reductionSummarizer = ReductionSummarizer()
    reductionSummarizerSummary = reductionSummarizer(parser.document, CompressedSentenceCount)
    summaryReduction = ' '.join([str(sentence) for sentence in reductionSummarizerSummary])
    print(f"{summaryReduction}\n")
    return summaryReduction
```

# Examples

A video that has been summarized using all of the different summarization methods

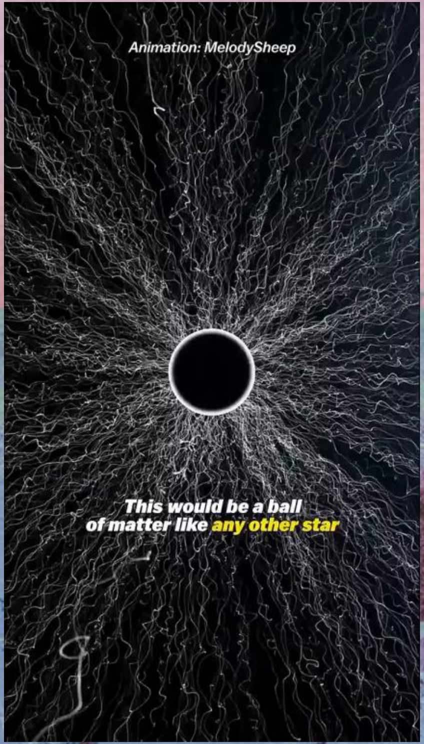




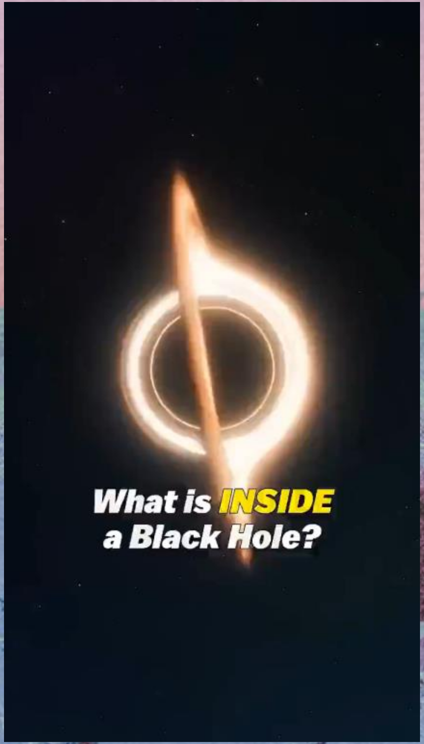
# Orignial Video

Something I get asked all the time is...

# Pure NLTK


Animation: MelodySheep
This would be a ball of matter like any other star

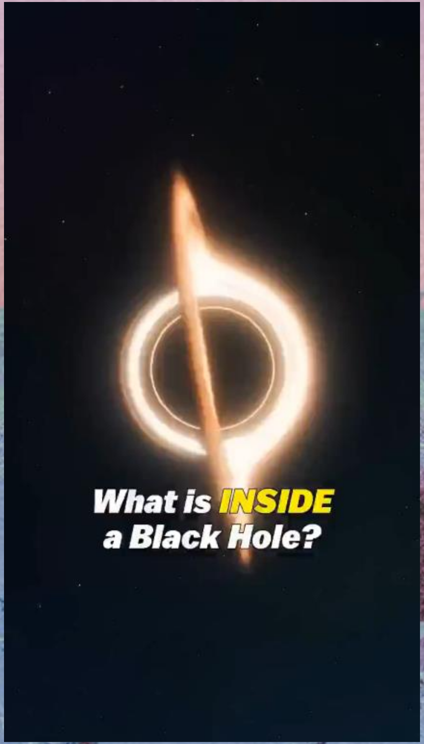
# Gensim



# Luhn Heuristic

*Something I get asked all the time is...*

# LexRank

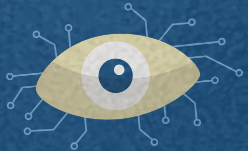


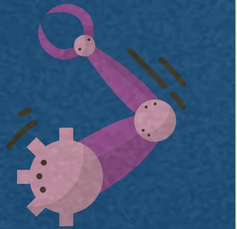
# KL-Sum

# Naïve Reduction

# Conclusion

What we achieved



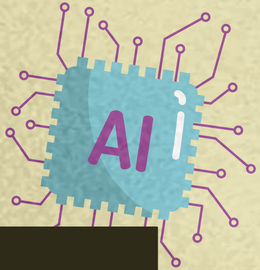


# Our Conclusion

We focused on crafting an extractive summarization app, leveraging a curated selection of algorithms that includes NLTK, Gensim TextRank, Luhn's Heuristic, LexRank, KL-Sum, and Naïve Reduction. This approach prioritizes fidelity to the source material.

Python served as the backbone, accompanied by Gensim, Sumy, json, pyqt5, nltk, whisper, pydub, and moviepy all encapsulated in a meticulously crafted GUI using pyqt5 that fosters simplicity and user engagement.

In conclusion, our work contributes to the ongoing evolution of video summarization.



# Thanks

Do you have any questions?