AKSHAR

A Project Report

Submitted by:

Konagandla Bhaskar (A20405217088)

Kartik Sharma (A20405217121)

P Venkata Hemanth (A20405217065)

in partial fulfillment

for the award of the Degree of

Bachelor of Technology

in

Department of Computer Science and Engineering

Submitted to :

Dr Rekha Chaturvedi

Department of Computer Science and Engineering

Amity School of Engineering & Technology

Amity University Rajasthan

May , 2021

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"AKSHAR"** submitted for the B. Tech. CSE degree is our original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Bhaskar Konagandla
Computer science and Engineering
Enrollment No : **A20405217088**

Kartik Sharma
Computer science and Engineering
Enrollment No : **A20405217121**

P Venkata Hemanth
Computer science and Engineering
Enrollment No : **A20405217065**

Place : Chirala , Guntur , Delhi
Date :

CERTIFICATE

This is to certify that the project titled **"AKSHAR"** is the bonafide work Carried out by **Bhaskar Konagandla , Kartik Sharma , P Venkata Hemanth** are the students of B Tech(CSE) of **DEPARTMENT OF CSE , AMITY SCHOOL OF ENGINERRING & TECHNOLOGY , AMITY UNVERSITY RAJASTHAN.** during the academic year 2020-2021 , in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology CSE and that the project has not formed the basis for the award previously of any other degree diploma fellowship Or any other similar title.

Signature of the Guide

Place : Chirala , Guntur , Delhi

Date :

ABSTRACT

As the massive amounts of both structured and unstructured data produced daily continues to grow , there are lot of actions that we do on data takes much time and consume more human resources. These tasks can also be performed by an trained program that can do the same work with in less time. However, as artificial intelligence technologies allow computers to track, extract, and analyse the vast array of information. One of these technologies makes use of a set of algorithms known as Natural Language Processing and deep learning, which allows computers to analyse the content of natural human language and derive new information from it.

This application is an one stop solution for most of the text related tasks that we encounter daily on our large corpus of data which include large articles , text documents etc. It is a web application that manages user given data and aids users to grasp much information and get valuable insights of their data. The application provides textual summary of his input in both extractive and abstractive ways. This summaries can also preserve sentiments and semantics of the context which can be further utilized for product reviews , users opinion thus one can generate reliable ratings of the products.Also, the application generates visualizations of your textual data (through word cloud representations , statistics of different kinds of words) , keywords extraction etc. for gaining better knowledge. Automatic Text Summarization will present one or more text documents while maintaining the main information content using an automatic machine with no more than half the original text or less than the original text. The extractive approach is still in demand in the past three years because the extractive is easier than abstractive. Users can also query their own text for answers or just give an question to the application which can try to fetch relevant answers.

ACKNOWLEDGEMENT

This acknowledgement is to show sincere gratitude towards the project guide **Dr Rekha Chaturvedi** , for their guidance and efforts for the completion of this project. This project was planned a year before and since then we are getting support from all our department. The motive of the project is to help reduce the mundane task of handling large amount of textual data. We sincerely thank Dr. Sunil Pathak our HOD for his vision and support for the entire project duration.

Bhaskar Konagandla
Enrollment No : **A20405217088**

Kartik Sharma
Enrollment No : **A20405217121**

P Venkata Hemanth
Enrollment No : **A20405217065**

TABLE OF CONTENTS

LIST OF IMAGES

LIST OF TABLES

1. PROJECT CONCEPT MAP

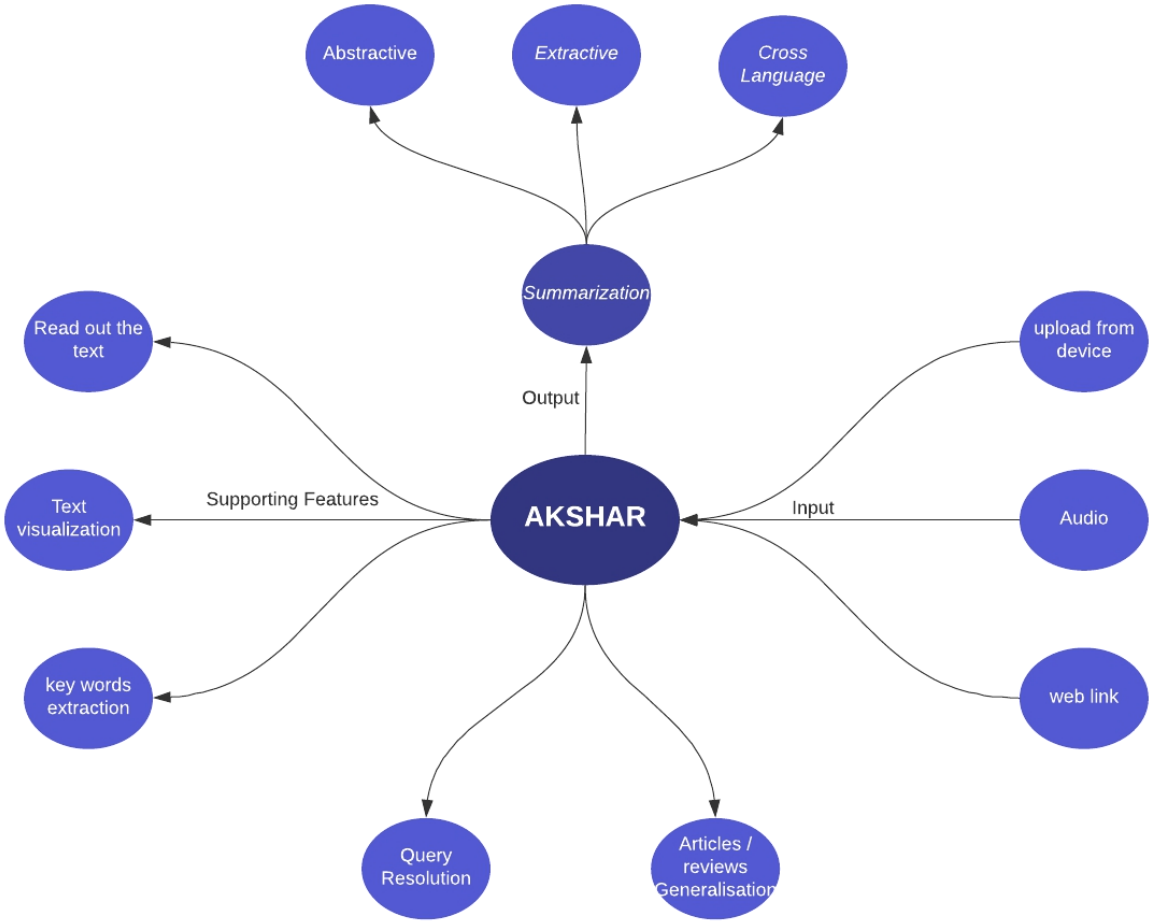

Fig 1.1. Project Overview

2. INTRODUCTION

2.1. Motivation Of The Project

The idea of the project came from the hectic work of understanding large text corpus in least possible time with out having much efforts for trivial tasks and getting as much information as possible. Through this platform we want to give users the essence of their inputs to make their work easier.

The application generates summaries of the text which is a subtext that does not exceed more than half of the length of the input text often less than half of it.It captures the main idea of the text and removes redundancies if any. It helps readers quickly identify information of their interest.There are many potential applications for this some of them could be :

- ✓ Abstracts (of documents)
- ✓ Head lines (for news articles)
- ✓ Table of contents (of a large document)
- ✓ Outlines (notes for students)
- ✓ Minutes (of a meeting)
- ✓ Previews (of movies)

And so on …

We even want to give the text a visual feeling for better interactivity with the text to the user.we want to automate many things that we do on text daily. We want to leverage the power of natural language processing and deep learning and make lives easier.

2.2. Problem Definition

This project is based on shrinking the text to the extent that is useful for the user. The aim is to generate summaries by keeping sentence relativity , meaning and mainly important and required information form the original text , also to request answers to the questions generated by users , to make some pictures out of text that will help more for us to get into it.

2.3. Objectives

Through this project we want to achieve the following objectives :

- ✓ The model must summarize the given corpus of text (of any length).
- ✓ Able to summarize the review / comments by preserving the sentiments.
- ✓ Help visualizing the text for the user.
- ✓ Keywords extraction.
- ✓ Give short information upon user's search query.
- ✓ In document Query resolution.

2.4. Goals

- ✓ We want to achieve human level performance for the stated objectives.
- ✓ Better user experience with the platform.
- ✓ Reduce human efforts for trivial tasks.
- ✓ Ease of use.
- ✓ Easy extraction of information.

3. LITERATURE SURVEY

3.1. Background

3.1.1. Automatic text Summarization

Automatic text Summarization (ATS) is nowadays a popular research area among researchers. Automatic text Summarization is the approach of generating the subset of the main text. This subset of the main text represents the complete text and the main idea of the text. Automatic Text Summarization is also known as Test Summarization. ATS is the important field of Natural Language Processing (NLP) and Data Mining (DM). This includes the abstractive and extractive summaries of the text.

With the advancement of technology, the internet is accessible through various devices like smart phones, smart watches and within the reach of common people. That leads to the accessibility of lot of information through world wide web (WWW). More information on the internet sometimes it becomes so difficult to select only required information from large texts. Due to the information, manual summarization of information is very challenging and also time-consuming task. Thus, we need an automatic text summarization system.Summary is a text that is produced from one or more texts, that conveys the important information text(s), and that is no longer than half of the original text(s) and usually significantly less than that. Summaries make the task of understanding the meaning of text easier. Text summarization helps user to manage vast amount of information by condensing document and include more relevant facts into them.

Fig 3.1. Setps in summary generation

The basic concept of the Automatic text summarization process based on literature review can be divided into 3 types of text summarization, namely numbers of documents, namely single document and multi document, techniques, namely extractive and abstractive, classification based, namely supervised and unsupervised.

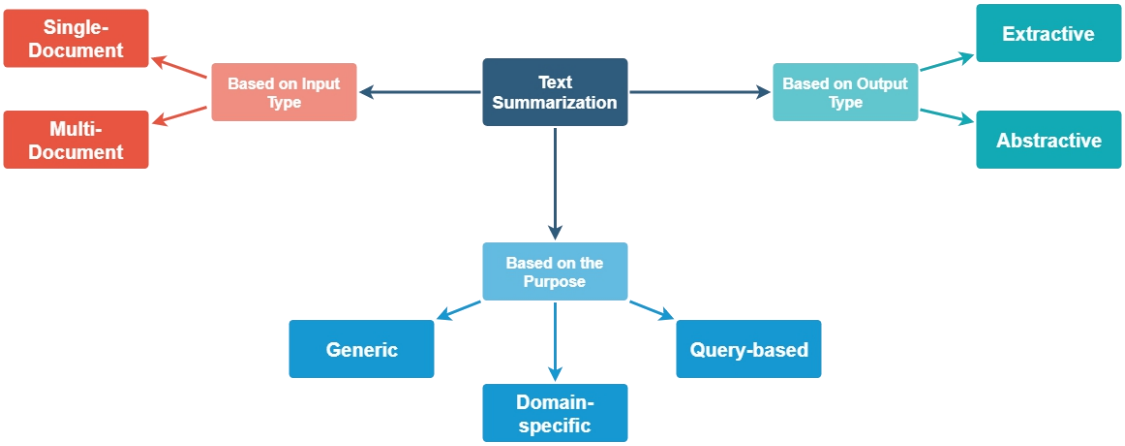

Fig 3.2. Text summarization classification

Single Document

A single document summary system will produce a summary based on one document source. A single document can consist of several sub documents with several paragraphs. The content described in each sub-document emphasizes all the different aspects around the same topic.

3.1.2. Extractive Summarization

Extractive summarization is extraction based summarization whose summary consists entirely of extracted content. Initially the research centered on techniques for managing documents with several approaches such as based on sentence position or word frequency in text. The experiment was then carried out using the Information Extraction (IE) Extraction technique for automatic summarizing on the grounds of increased accuracy and more specific results.

3.1.3. Abstractive Summarization

Abstractive summarization is a summarizing system by producing new phrases or using words that are not in the original text. For perfect abstractive summaries, the

model must really understand the document and then try to express that understanding briefly using new words and phrases or arrange them in different forms. The extract field is more well-researched, in contrast to abstracts which have more challenging problems and require extensive natural language processing.

3.1.4. Keywords Extraction

Keyword extraction (also known as keyword detection or keyword analysis) is a text analysis technique that automatically extracts the most used and most important words and expressions from a text. It helps summarize the content of texts and recognize the main topics discussed.

Text analysis uses machine learning artificial intelligence (AI) with natural language processing (NLP) to break down human language so that it can be understood and analyzed by machines. Keyword analysis can find keywords from all manner of text: regular documents and business reports, social media comments, online forums and reviews, news reports, and more.

To analyze thousands of online reviews about your product. Keyword extraction helps you sift through the whole set of data and obtain the words that best describe each review in just seconds. That way, you can easily and automatically see what your customers are mentioning most often, saving your teams hours upon hours of manual processing.

3.1.5. Word cloud representations

Word Clouds (also known as wordle, word collage or tag cloud) are visual representations of words that give greater prominence to words that appear more frequently. For Mentimeter Word Clouds, the words that are added most frequently by audience members using their smartphones. This type of visualization can help presenters to quickly collect data from their audience, highlight the most common answers and present the data in a way that everyone can understand.

Choosing the word can be based on

- ✓ Frequency
- ✓ Significance
- ✓ Categorization

3.2. Existing System

3.2.1. Sentence Scoring based on Word Frequency

The first approach will explore the simplest of the three. Here we assign weights to each word based on the frequency of the word in the passage. For example, if “Soccer” occurs 4 times within the passage, it will have a weight of 4.

Using the weights assigned to each word, we will create a score for each sentence. In the end, we will be taking the score of the top `N` sentences for the summary. As you’d imagine, just by leveraging the raw score of each sentence, the length of certain sentences will skew the results. This is why we will normalize the scores by dividing by the length of each sentence.

Now, to create the summary, we will take any sentence that has a score that exceeds a threshold. In this case, the threshold will be the average score of all of the sentences.

3.2.2. Text Rank approach

This is essentially a derivative of the famous PageRank created by the Google Cofounders. In PageRank, they generated a matrix that calculates the probability that a user will move from one page to another. In the case of TextRank, we generate a cosine similarity matrix where we have the similarity of each sentence to each other.

A graph is then generated from this cosine similarity matrix. We will then apply the PageRank ranking algorithm to the graph to calculate scores for each sentence.

3.3. Proposed Methods

3.3.1. Extractive text Summarization

Extractive text summarization done by picking up the most important sentences from the original text in the way that forms the final summary. Extractive techniques generally generate summaries through 3 phases or it essentially based on them.
These phases are preprocessing step, processing step and generation
step:

1) Preprocessing step: the representation space dimensionality of the original text is reduced to involve a new structure representation.
It usually includes:

a. **Stop-word elimination:** Common words without semantics that do not collect information relevant to the task (for example, "the", "a", "an", "in") are eliminated.

2) **Processing step:** It uses an algorithm with the help of features generated in the preprocessing step to convert the text structure to the summary structure. In which, the sentences are scored.

3) **Generation step:** sentences are ranked. Then, it pick up the most important sentences from the ranked structure to generate the final required summary.

The last two stages - processing and generation steps - can be also described approximately as three main components: sentence scoring, selection and paraphrasing (reformulation). At sentence scoring, for each sentence a score is assigned which points to its significance. After that, the most important sentences is extracted.. At sentence selection, the summarization system has to specify the best collection of significant sentences that form the final summary with taking into consideration the most prominent factors: redundancy and cohesion. The traditional method for sentence selection is to pick up the top ranked sentences directly but, the redundancy elimination is the key issue.

Features	Description
Sentence Position	It implies that in a specific position, the important sentences will be presented such as first or last positions.
Title Similarity	The sentence is considered to be important if it has similarity with the document title. This similarity can be calculated by cosine similarity measure.
Similarity to Keywords	Compute the similarity between each sentence and set of keywords based on the cosine similarity measure.
Sentence Length	Sentences with specific length are considered to be important.
Term Frequency	This means terms that have occurred over and over and that increase the score of their sentences. It reflects how important the word is for the document.
Cue Method	Words that have positive or negative effect on sentence weight.
Proper Noun	Sentences which have proper nouns are considered to be important.
Sentence to Sentence Similarity	The similarity between each sentence and all other sentences calculated, added up and then normalized [37].
Sentence to Centroid Similarity	Centroid sentence is calculated first. Then similarity between each sentence and the centroid sentence calculated [37].
Numerical Data	The Appearance of such data in a sentence can reflect important statistics and can increase its chance to be selected for the summary.
Presence of Special Characters or Words	Some of them give the sentences lower probability to be selected such as: presence of brackets. And others give the sentences higher probability such as: presence of commas, inverted commas, acronym words and upper case words.

Fig 3.3. Features

3.3.2. Keyword extraction

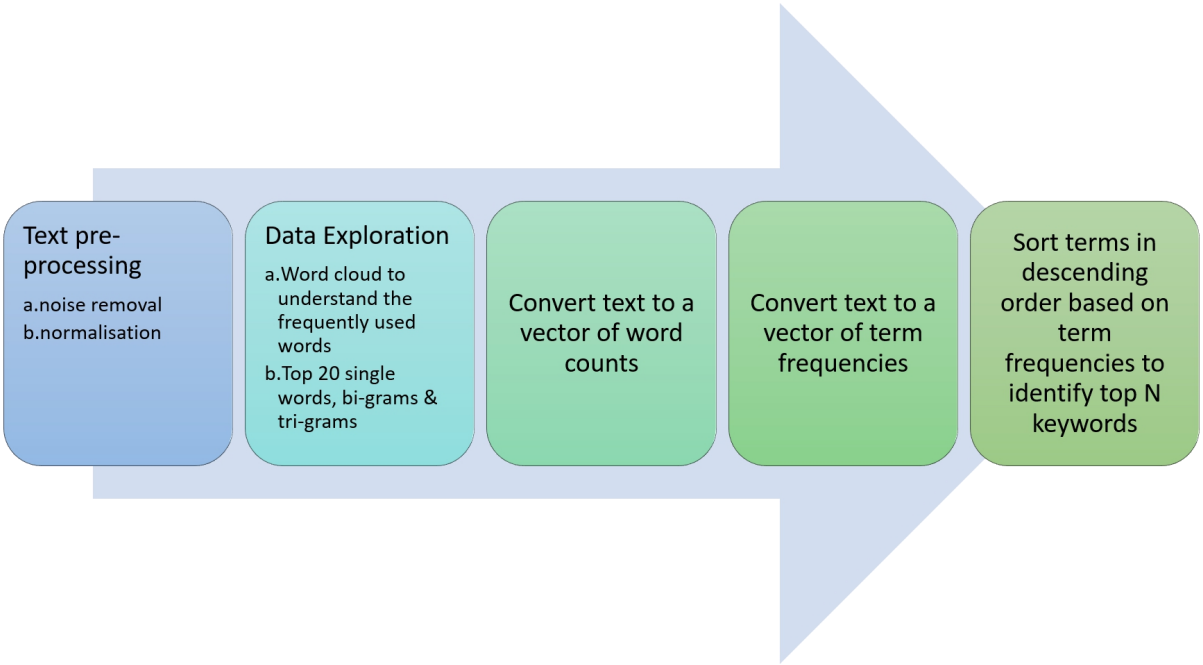

Fig 3.4. Keword Extraction

Text Pre-processing

Text pre-processing can be divided into two broad categories — noise removal & normalization. Data components that are redundant to the core text analytics can be considered as noise.

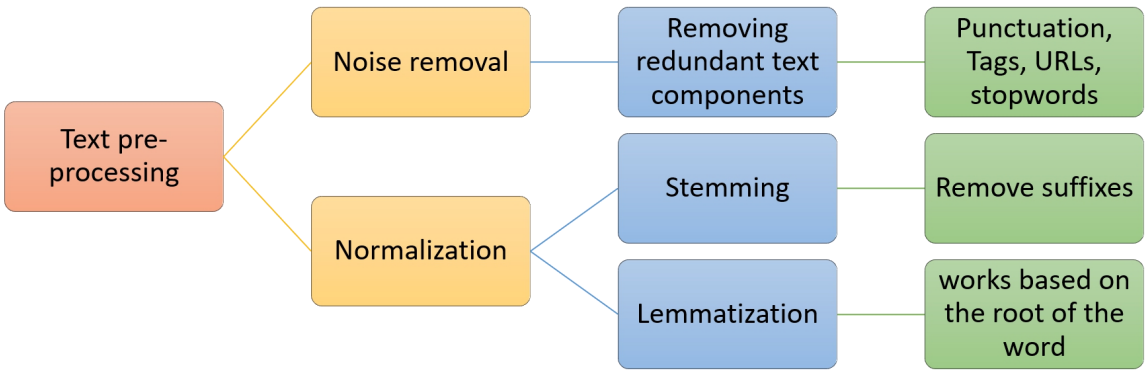

Fig 3.5. pre-processing

Handling multiple occurrences / representations of the same word is called normalization. There are two types of normalization — stemming and lemmatization. Let us consider an example of various versions of the word learn — learn, learned, learning, learner. Normalisation will convert all these words to a single normalised version — "learn".
Stemming normalizes text by removing suffixes.
Lemmatisation is a more advanced technique which works based on the root of the word.

Removing stopwords:
Stop words include the large number of prepositions, pronouns, conjunctions etc in sentences. These words need to be removed before we analyse the text, so that the frequently used words are mainly the words relevant to the context and not common words used in the text.

Text preparation
Text in the corpus needs to be converted to a format that can be interpreted by the machine learning algorithms. There are 2 parts of this conversion — Tokenisation and Vectorisation.

Tokenisation is the process of converting the continuous text into a list of words. The list of words is then converted to a matrix of integers by the process of vectorisation. Vectorisation is also called feature extraction.
For text preparation we use the bag of words model which ignores the sequence of the words and only considers word frequencies.

As the first step of conversion, we will use the CountVectoriser to tokenise the text and build a vocabulary of known words. We first create a variable "cv" of the CountVectoriser class, and then evoke the fit_transform function to learn and build the vocabulary.

The next step of refining the word counts is using the TF-IDF vectoriser. The deficiency of a mere word count obtained from the countVectoriser is that, large counts of certain common words may dilute the impact of more context specific

words in the corpus. This is overcome by the TF-IDF vectoriser which penalizes words that appear several times across the document. TF-IDF are word frequency scores that highlight words that are more important to the context rather than those that appear frequently across documents.

TF-IDF consists of 2 components:

TF — term frequency

IDF — Inverse document frequency

- $$TF = \frac{\text{Frequency of a term in a document}}{\text{total number of terms in the document}}$$

- $$IDF = \frac{\log(\text{Total documents})}{\text{\# of documents with the term}}$$

Based on the TF-IDF scores, we can extract the words with the highest scores to get the keywords for a document.

3.3.3. Query resolution

When ever user enters any query , with the help of python library that helps in accessing the links from the internet (these are the links that you get when you search the same query on google search engine.) up on users request multiple webpages are scraped and text is collected form those websites. Using our extractive or abstractive approaches we can get an answer for the users query.

4. PROJECT METHODOLOGY

A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.

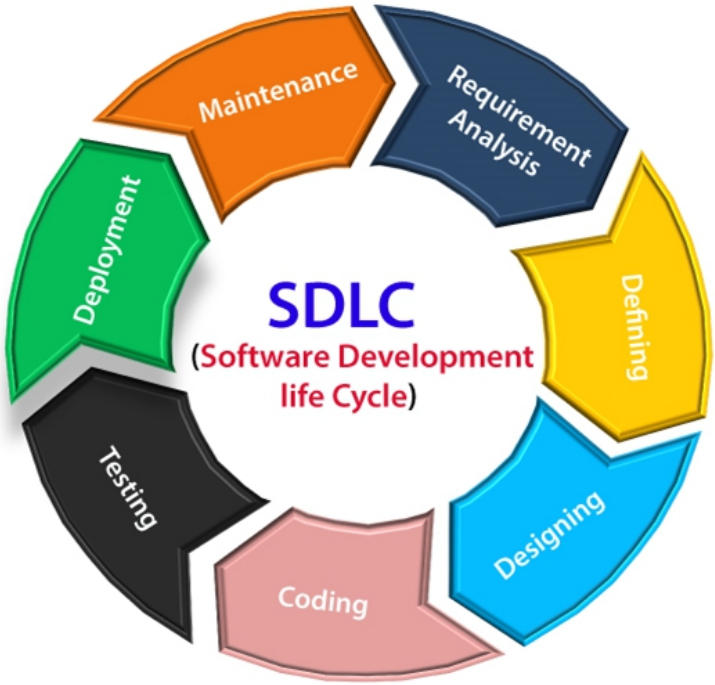

Fig 4.1. SDLC

There are many types of SDLC models available , but we used the iterative model for our project.

4.1. Iterative Model

This model leads the software development process in iterations. It projects the process of development in cyclic manner repeating every step after every cycle of SDLC process

The software is first developed on very small scale and all the steps are followed which are taken into consideration. Then, on every next iteration, more features and modules are designed, coded, tested and added to the software. Every cycle produces a software, which is complete in itself and has more features and capabilities than that of the previous one.

After each iteration, we can do work on risk management and prepare for the next iteration. Because a cycle includes small portion of whole software process, it is easier to manage the development process but it consumes more resources.

Why we used the Iterative Model?

- ✓ Requirements are defined clearly and easy to understand.
- ✓ The software application is large.
- ✓ There are chances of requirement of changes in future.
- ✓ Advantages of Iterative Model:
- ✓ Testing and debugging during smaller iteration is easy.
- ✓ A Parallel development can plan.
- ✓ It is easily acceptable to ever-changing needs of the project.
- ✓ Risks are identified and resolved during iteration.
- ✓ Limited time spent on documentation and extra time on designing.

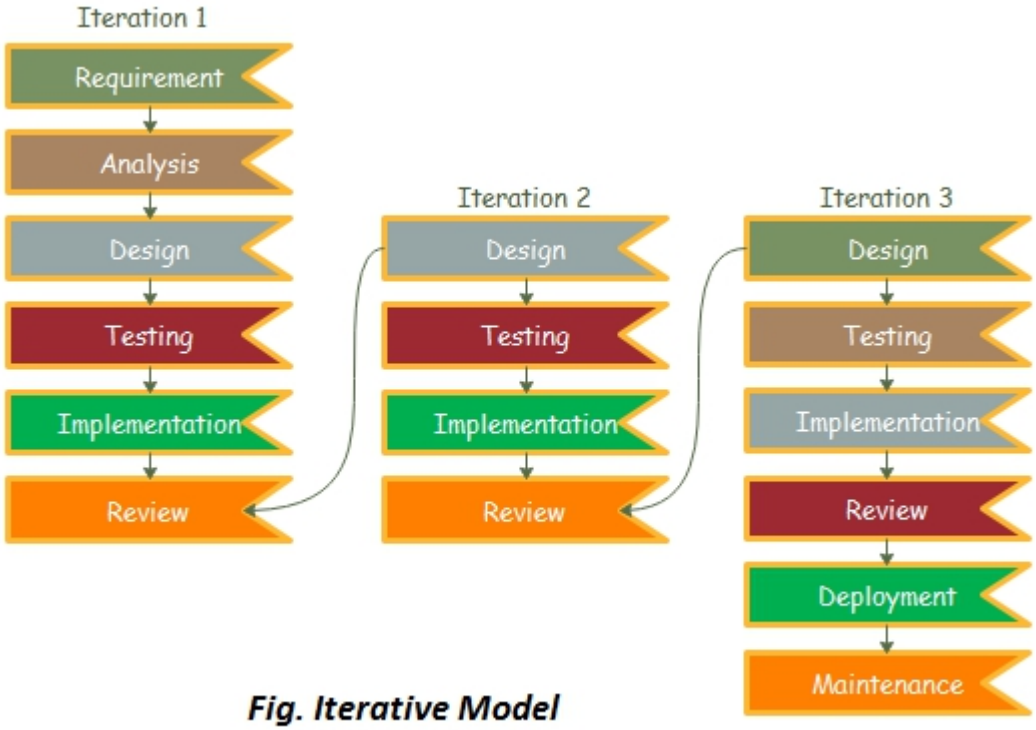

Fig. Iterative Model

5. REQUIREMENT ENGINEERING

Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

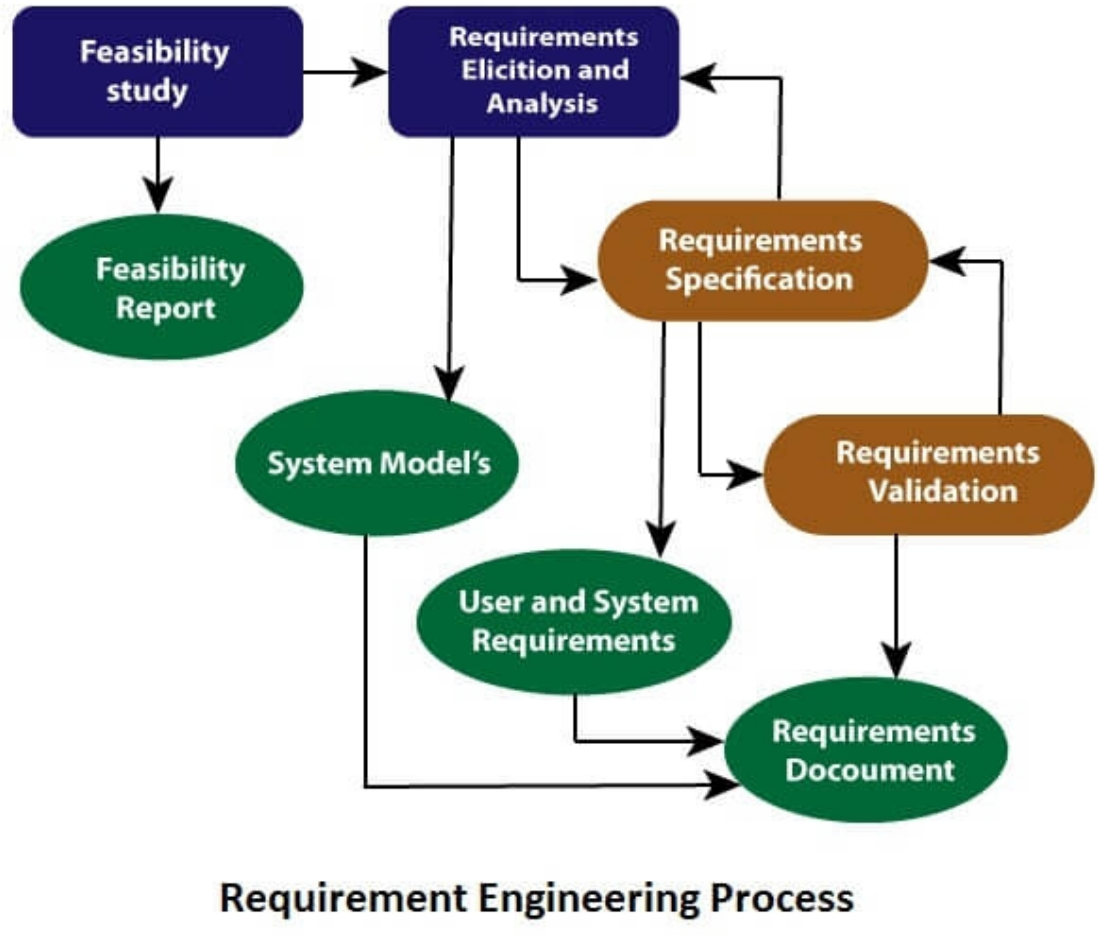

Fig 5.1. Requirement Engineering Process

5.1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.
Technologically feasible: The language used is python and python idle is used as the platform which is very much popular and usable in most of the devices. The idea is practically possible but uniquely presented through this project.

Economically feasible: The project is totally a software project and needs only a platform for presentation. There is no hardware involved in making the project as part of the project. The project can run on a normal computer.

Legally feasible: The data used for visualization will be taken with the permission of the candidates and doesn't harm any organization. The data used for summarization and visualization will be used for study and only project purpose.

5.2. Requirement Elicitation and Analysis

This is also known as the gathering of requirements.
Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.
Requirements are:

- The model must summarize the given corpus of text (of any length).
- Able to summarize the review / comments by preserving the sentiments.
- Help visualizing the text for the user.
- Keywords extraction.
- Give short information upon user's search query.

5.3. System Analysis

At this step the developers decide a road map of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

5.3.1. Functional requirements

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provide a permanent copy of the results for later consultation.These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

They are :-

- The application must accept most types of text related sources I.e pdf , word document , able to insert text etc.
- It must able to scrape data neatly for the specified web links.
- Able to give summaries extracted from text.
- Able to write summaries in its own way also.
- Must generate word clouds and histograms for the given text depending up on the specified requirements.
- Extract keywords and phrases in the given text.
- Highlight the extracted keywords and phrases.
- Answer the queries asked by the user form his source .
- If user did not give any source and still want to query , application must scrape and show the results from the internet.

5.3.2. Non - Functional requirements

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.
A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

- Performance – for example Response Time, Throughput, Utilization
- Scalability
- Capacity
- Availability
- Reliability
- Recoverability
- Maintainability
- Serviceability
- Security
- Data Integrity
- Usability
- Interoperability

5.3.3. HARDWARE REQUIREMENTS

The hardware requirement specifies each interface of the software elements and the hardware elements of the system. These hardware requirements include configuration characteristics.

System	:	Any processor after Pentium
Hard Disk	:	40GB.
Monitor	:	14'ColourMonitor.
Mouse	:	Optical Mouse.
Ram	:	512 Mb.

5.3.4. SOFTWARE REQUIREMENTS:

The software requirements specify the use of all required software products like editors , ide's , compilers etc. Each interface specifies the purpose of the interfacing software as related to this software product.

- ✓ HTML
- ✓ CSS
- ✓ JAVA SCRIPT.
- ✓ PYTHON .
- ✓ Flask

Libraries Required

- ✓ Flask
- ✓ Nltk
- ✓ Pandas
- ✓ Matplotlib
- ✓ Mpld3
- ✓ Re
- ✓ Urllib
- ✓ Requests
- ✓ Pdfminer
- ✓ Docx
- ✓ Google Trans

6. DESIGN

6.1. System Architecture

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

A system architecture can consist of system components and the sub-systems developed, that will work together to implement the overall system.

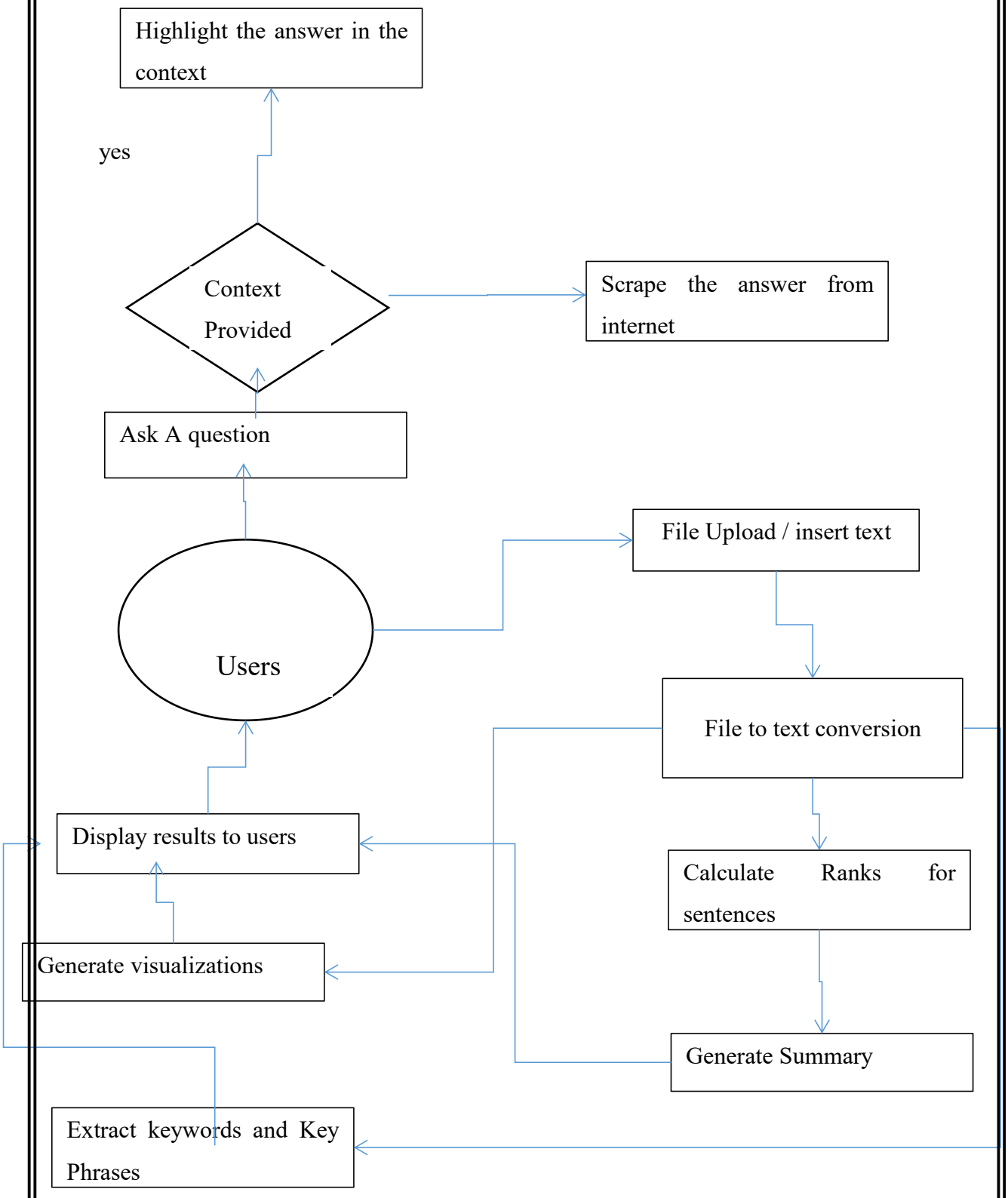

Fig 6.1. System Architecture

6.2. Data flow diagram

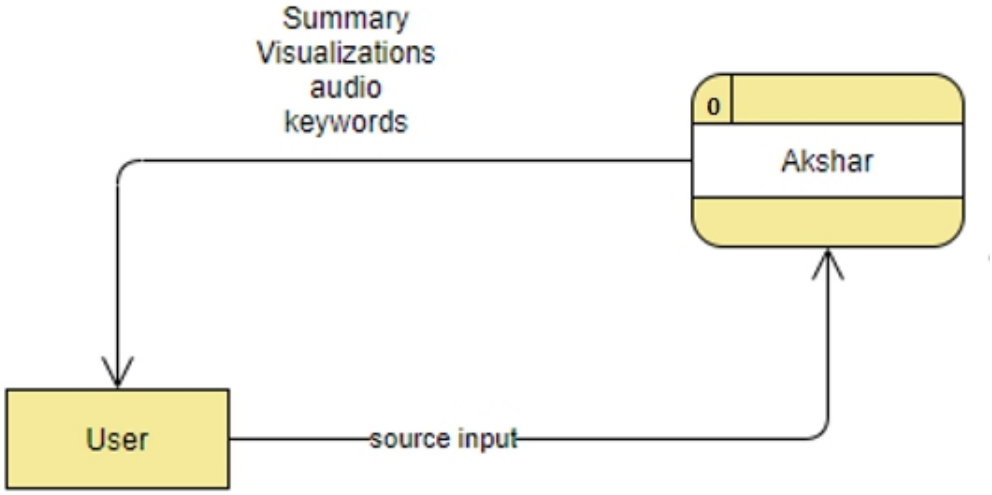

Fig 6.2. Data Flow Diagram - Level 0

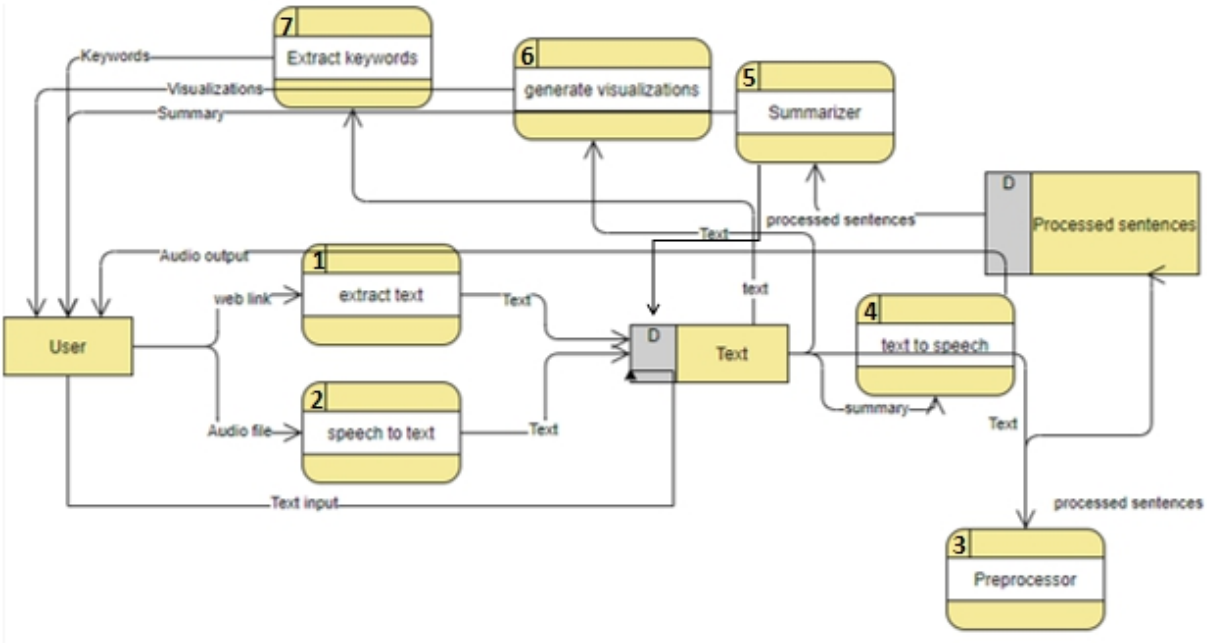

Fig 6.3. Data Flow Diagram - Level 1

6.3. Use case Diagram

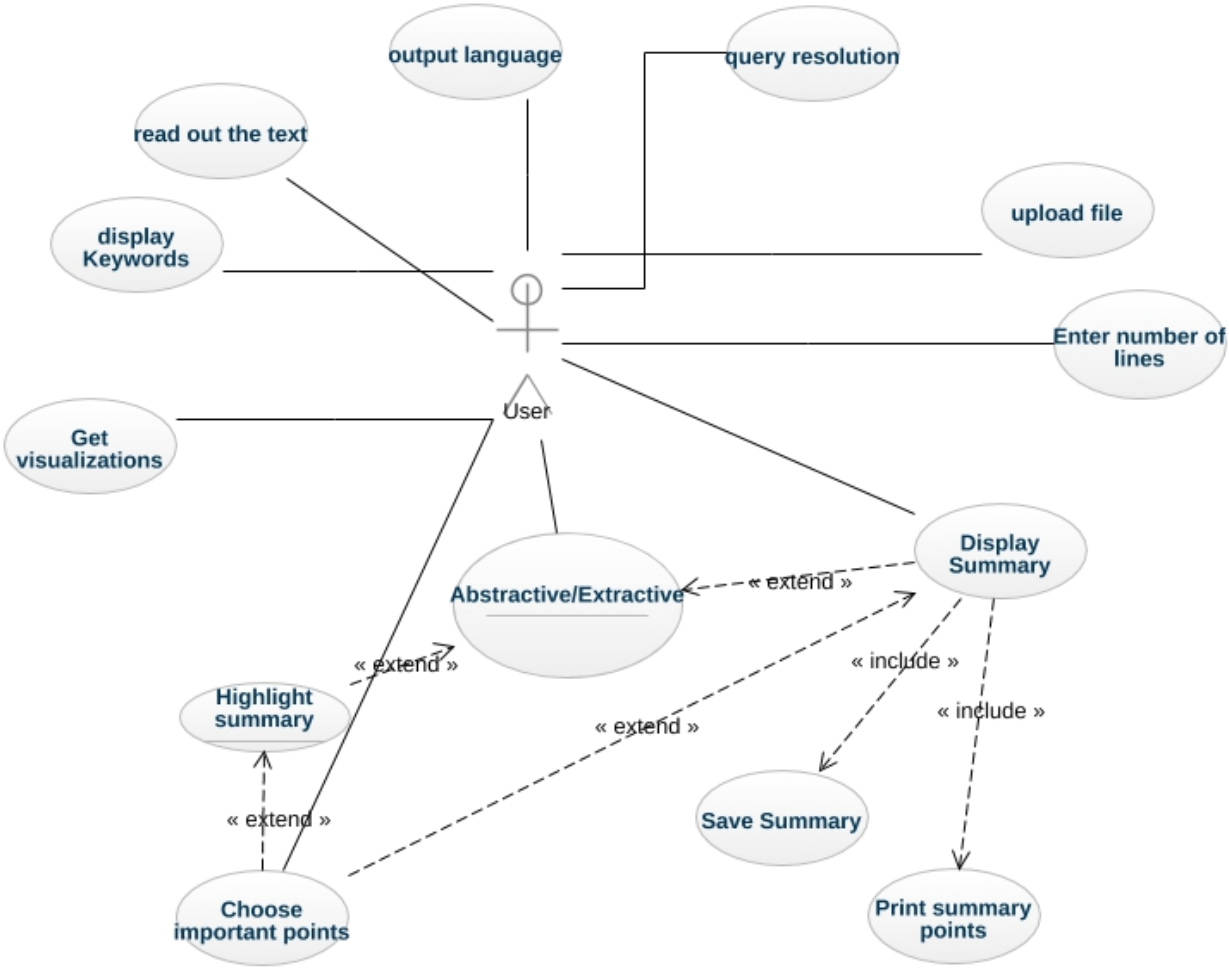

Fig 6.4. Use case Diagram - User

6.4. Class Diagram

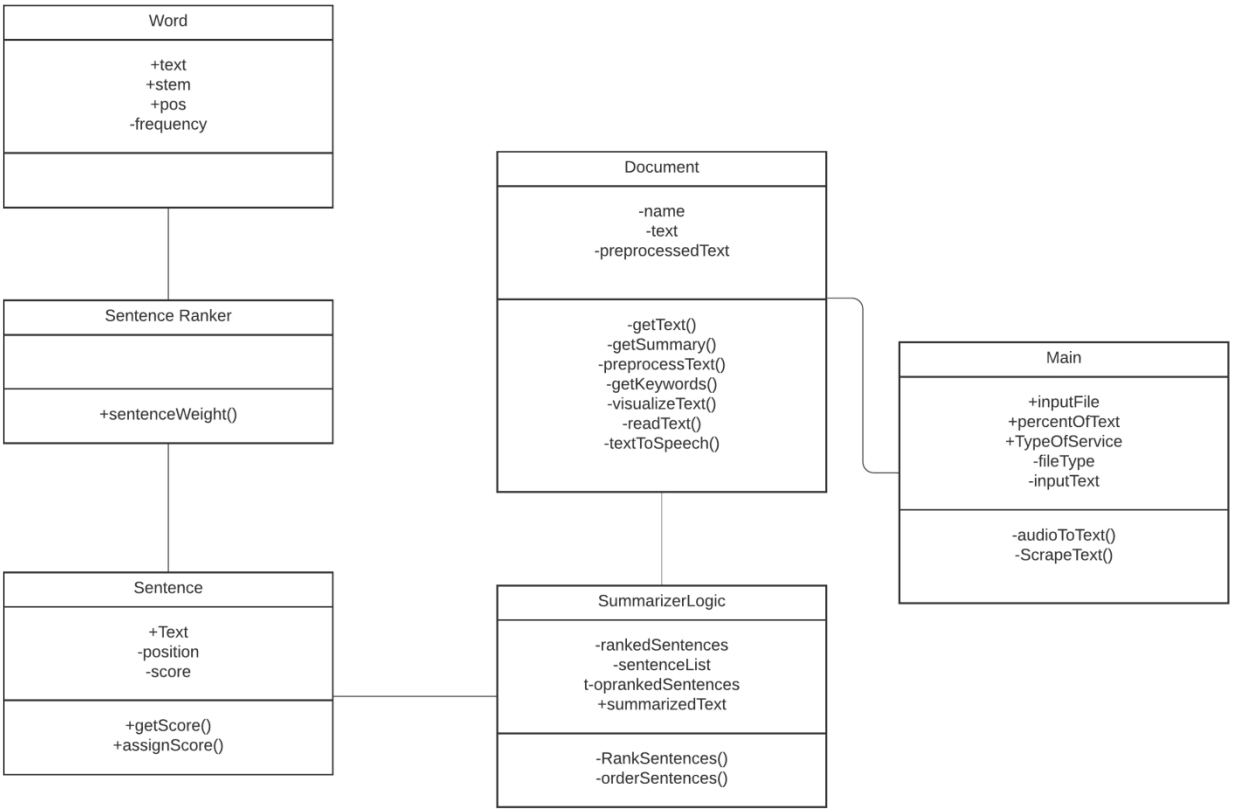

Fig 6.5. Class diagram

7. CODE

7.1. Script Module

```
from flask import *
import TextVisualizations as txtv
import HelperTools as ht
import ExtractiveTextSum as ets
import ATS as ats
import webbasedsearch as wbs
app = Flask(__name__)

txt = ""
howmuchkeyphrases = 1
howmuchkeywords = 2
howmuch = 1
lang = 'en'


@app.route('/file_upload_form', methods=['GET', 'POST'])
def file_upload_form():
	return render_template("file_upload_form.html")


@app.route('/success', methods=['POST'])
def success():
	if request.method == 'POST':
		f = request.files['file']
		f.save(f.filename)
		local_txt = ht.text_file(f.filename)

		global howmuch
		global lang
		global howmuchkeyphrases
		global howmuchkeyphrases
		global txt

		howmuch = request.form['howmuch']
		lang = request.form['languages']
		howmuchkeyphrases = request.form['howmuchkeyphrases']
		howmuchkeywords = request.form['howmuchkeywords']
		txt = local_txt
		txtv.run(local_txt)

		return render_template("visualizations.html")
```

```
@app.route('/')
def message():
        return render_template('message.html')


@app.route('/inserttext', methods=['GET', 'POST'])
def inserttext():
        return render_template('inserttext.html')


@app.route('/keywords', methods=['GET', 'POST'])
def keywords():
        data = ht.get_keywords(txt, howmuchkeywords)
        return render_template('keywords.html', data=data, context=txt)


@app.route('/keyphrases', methods=['GET', 'POST'])
def keyphrases():
        data = ht.get_keyphrases(txt, int(howmuchkeyphrases))
        return render_template('keyphrases.html', data=data, context=txt)


@app.route('/summary', methods=['GET', 'POST'])
def summary():
        data = ets.Word_weight(txt, int(howmuch), lang)
        return render_template('summary.html', data=data)


@app.route('/insertlink', methods=['GET', 'POST'])
def insertlink():
        return render_template('insertlink.html')


@app.route('/submit', methods=['POST'])
def submit():
        global    howmuch
        global lang
        howmuch = request.form['howmuch']
        lang = request.form['languages']
        global howmuchkeyphrases
        global howmuchkeyphrases
        howmuchkeyphrases = request.form['howmuchkeyphrases']
        howmuchkeywords = request.form['howmuchkeywords']

        global txt
        txt = request.form['text']

        txtv.run(request.form['text'])
        return render_template("visualizations.html")
```

```
@app.route('/submitlink', methods=['POST', 'GET'])
def submitlink():
	global howmuch
	global lang
	got_the_text = ht.get_text_from_link(request.form['text'])
	howmuch = request.form['howmuch']
	lang = request.form['languages']
	global howmuchkeyphrases
	global howmuchkeyphrases
	howmuchkeyphrases = request.form['howmuchkeyphrases']
	howmuchkeywords = request.form['howmuchkeywords']

	global txt
	txt = request.form['text']
	txtv.run(got_the_text)
	return render_template("visualizations.html")



@app.route('/file_upload_formats', methods=['GET', 'POST'])
def file_upload_formats():
	return render_template("file_upload_formats.html")


@app.route('/successats', methods=['POST', 'GET'])
def successats():
	if request.method == 'POST':
		global howmuch
		global lang
		howmuch = request.form['howmuch']
		lang = request.form['languages']

		f = request.files['file']
		f.save(f.filename)
		local_txt = ht.text_file(f.filename)
		global howmuchkeyphrases
		global howmuchkeyphrases
		howmuchkeyphrases = request.form['howmuchkeyphrases']
		howmuchkeywords = request.form['howmuchkeywords']

		global txt
		txt = local_txt
		txtv.run(local_txt)

		return render_template("visualizations.html")


@app.route('/inserttextats', methods=['GET', 'POST'])
def inserttextats():
```

```
        return render_template('inserttextats.html')


@app.route('/insertlinkats', methods=['GET', 'POST'])
def insertlinkats():
        return render_template('insertlinkats.html')


@app.route('/submitats', methods=['POST', 'GET'])
def submitats():
        global howmuch
        global lang
        howmuch = request.form['howmuch']
        lang = request.form['languages']

        global howmuchkeyphrases
        global howmuchkeyphrases
        howmuchkeyphrases = request.form['howmuchkeyphrases']
        howmuchkeywords = request.form['howmuchkeywords']

        global txt
        txt = request.form['text']
        txtv.run(request.form['text'])
        return render_template("visualizations.html")


@app.route('/submitlinkats', methods=['POST', 'GET'])
def submitlinkats():
        global howmuchkeyphrases
        global howmuchkeyphrases
        global howmuch
        global lang
        howmuchkeyphrases = request.form['howmuchkeyphrases']
        howmuchkeywords = request.form['howmuchkeywords']
        howmuch = request.form['howmuch']
        lang = request.form['languages']

        got_the_text = ht.get_text_from_link(request.form['text'])
        global txt
        txt = request.form['text']
        txtv.run(got_the_text)
        return render_template("visualizations.html")

@app.route('/answering', methods=['GET', 'POST'])
def answering():
        quest = request.form['prasna']
        con = request.form['nepadyam']
        ans = ats.question_answering(quest, con)
        return render_template("Question_Answering.html", question=quest,
answer=ans, context=con)
```

```
@app.route('/question_answering', methods=['GET', 'POST'])
def question_answering():
    return render_template("Question_Answering.html")


@app.route('/file_upload_formds', methods=['GET', 'POST'])
def file_upload_formds():
    return render_template("file_upload_formds.html")


@app.route('/successds', methods=['POST'])
def successds():
    if request.method == 'POST':
        f = request.files['file']
        f.save(f.filename)
        local_txt = ht.text_file(f.filename)
        return render_template("Question_Answering.html", context=local_txt)


@app.route('/inserttextds', methods=['GET', 'POST'])
def inserttextds():
    return render_template('inserttextds.html')


@app.route('/insertlinkds', methods=['GET', 'POST'])
def insertlinkds():
    return render_template('insertlinkds.html')


@app.route('/submitds', methods=['POST'])
def submitds():
    txt = request.form['text']
    return render_template("Question_Answering.html", context=txt)


@app.route('/submitlinkds', methods=['POST'])
def submitlinkds():
    got_the_text = ht.get_text_from_link(request.form['text'])
    return render_template("Question_Answering.html", context=got_the_text)


@app.route('/successws', methods=['POST'])
def successws():
    txt = request.form['description']
    type = request.form['type']
    output = wbs.mainu(txt , type )
    print
    return render_template("websearch.html", data=output)
```

```
@app.route('/websearch', methods=['GET', 'POST'])
def websearch():
        return render_template("websearch.html" , data ={})


if __name__ == '__main__':
        app.config['TEMPLATES_AUTO_RELOAD'] = True
        app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
        app.run(debug=True)
```

7.2. Extractive Text Summarization Module

```
import heapq
import re
import nltk
from nltk.tokenize import sent_tokenize
import HelperTools as ht

f = open('SmartStoplist.txt', 'r')

try:
        smartstoplist = f.read()
        # print(text)
except UnicodeDecodeError:
        pass

stopwords = nltk.corpus.stopwords.words('english')
stopwords.extend(smartstoplist.split('\n'))
stopwords = set(stopwords)
def Word_weight(article_text, p, lang):
        count = 1
        try:
                article_text = re.sub(r'\[[0-9]*\]', ' ', article_text)
                article_text = re.sub(r'\s+', ' ', article_text)
                formatted_article_text = re.sub('[^a-zA-Z]', ' ', article_text)
                formatted_article_text = re.sub(r'\s+', ' ', formatted_article_text)
                # print(article_text)
                sentence_list = (sent_tokenize(article_text))
                # print(len(sentence_list))
                sentence_list = set(sentence_list)
                sentence_list = list(sentence_list)
                noofsentences = len(sentence_list)
                howmuch = p * noofsentences // 100

                new_list = []
                for sentence in sentence_list:
```

```
                new_list.append((str(count) + ' : ' + sentence))
                count += 1
        sentence_list = new_list

        word_frequencies = {}
        for word in nltk.word_tokenize(formatted_article_text):
                if word not in stopwords:
                        if word not in word_frequencies.keys():
                                word_frequencies[word] = 1
                        else:
                                word_frequencies[word] += 1
        maximum_frequency = max(word_frequencies.values())

        for word in word_frequencies.keys():
                word_frequencies[word] = (word_frequencies[word] /
maximum_frequency)
        sentence_scores = {}
        for sent in sentence_list:
                for word in nltk.word_tokenize(sent.lower()):
                        if word in word_frequencies.keys():
                                if sent not in sentence_scores.keys():
                                        sentence_scores[sent] = word_frequencies[word]
                                else:
                                        sentence_scores[sent] += word_frequencies[word]

        summary_sentences = heapq.nlargest(howmuch, sentence_scores,
key=sentence_scores.get)

        summary_dict = {}
        for sent in summary_sentences:
                index = sent.find(':')
                summary_dict[int(sent[:index - 1])] = sent[index + 2:]

        order_details = sorted(summary_dict.keys())
        i = 1
        string = ''
        print(order_details)
        lst=[]
        if lang == 'en':

                '''for j in order_details:
                        print(i, ' : ', summary_dict[j])
                        string += str(i) + " : " + summary_dict[j] + "&#13;&#10;"
                        i += 1'''
                for j in order_details:
                        print(i, ' : ', summary_dict[j])
                        lst.append( str(i) + " : " + summary_dict[j] )
                        i += 1

        else:
```

```
                '''for j in order_details:
                        print(i, ' : ', summary_dict[j])
                        string += str(i) + " : " + ht.langtranslate(summary_dict[j], lang) +
"&#13;&#10;"
                        i += 1'''
                for j in order_details:
                        print(i, ' : ', summary_dict[j])
                        lst.append( str(i) + " : " + ht.langtranslate(summary_dict[j], lang))
                        i += 1

        except ValueError:
                pass
        return lst
```

7.3. Text Visualization Module

```
import collections
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import mpld3
import nltk
from nltk.tokenize import sent_tokenize
import re
import unicodedata
from nltk.corpus import stopwords
from collections import Counter
import mpld3
from matplotlib.pyplot import figure
from wordcloud import WordCloud, STOPWORDS
from textblob import TextBlob

ADDITIONAL_STOPWORDS = ['covfefe']
stopwords = nltk.corpus.stopwords.words('english') + ADDITIONAL_STOPWORDS


def preprocess(ReviewText):
        ReviewText = ReviewText.replace("(<br/>)", "")
        ReviewText = ReviewText.replace('(<a).*(>).*(</a>)', '')
        ReviewText = ReviewText.replace('(&amp)', '')
        ReviewText = ReviewText.replace('(&gt)', '')
        ReviewText = ReviewText.replace('(&lt)', '')
        ReviewText = ReviewText.replace('(\xa0)', ' ')
        return ReviewText


def get_top_n_words_wo_stopwords(corpus, n_print=None):
        corpus = preprocess(corpus)
        wordcount = {}
```

```
    a = corpus
    fig = figure()
    ax = fig.gca()

    for word in a.lower().split():
        word = word.replace(".", "")
        word = word.replace(",", "")
        word = word.replace(":", "")
        word = word.replace("\"", "")
        word = word.replace("!", "")
        word = word.replace("â€œ", "")
        word = word.replace("â€˜", "")
        word = word.replace("*", "")
        if word not in stopwords:
            if word not in wordcount:
                wordcount[word] = 1
            else:
                wordcount[word] += 1
    word_counter = collections.Counter(wordcount)
    # for word, count in word_counter.most_common(n_print):
    #     print(word, ": ", count)
    lst = word_counter.most_common(n_print)
    df = pd.DataFrame(lst, columns=['Word', 'Count'])
    df.plot.bar(x='Word', y='Count', ax=ax, figsize=(14, 6))
    plt.xlabel('Words')
    plt.ylabel('Word Count')

    plt.title(label='Most Common Words (Unigrams)',
              fontweight=10,
              pad='7.0')
    plt.tight_layout()
    mpld3.save_html(fig, "static/unigram_wo_stopwords.html")


def get_top_n_words(corpus, n_print=None):
    fig = figure()
    ax = fig.gca()

    corpus = preprocess(corpus)
    wordcount = {}
    a = corpus
    for word in a.lower().split():
        word = word.replace(".", "")
        word = word.replace(",", "")
        word = word.replace(":", "")
        word = word.replace("\"", "")
        word = word.replace("!", "")
        word = word.replace("â€œ", "")
        word = word.replace("â€˜", "")
        word = word.replace("*", "")
```

```
            if word not in wordcount:
                wordcount[word] = 1
            else:
                wordcount[word] += 1
    word_counter = Counter(wordcount)
    lst = word_counter.most_common(n_print)
    df = pd.DataFrame(lst, columns=['Word', 'Count'])
    df.plot.bar(x='Word', y='Count', ax=ax, figsize=(14, 6))
    plt.xlabel('Words')
    plt.ylabel('Word Count')

    plt.title(label='Most Common Words with stop words (Unigrams)',
                fontweight=10,
                pad='7.0')
    plt.tight_layout()
    mpld3.save_html(fig, "static/unigram_stopwords.html")


def get_top_n_bigram_stopwords(corpus, n_print=None):
    fig = figure()
    ax = fig.gca()

    corpus = preprocess(corpus)
    a = corpus
    for word in a.lower().split():
        word = word.replace(".", "")
        word = word.replace(",", "")
        word = word.replace(":", "")
        word = word.replace("\"", "")
        word = word.replace("!", "")
        word = word.replace("â€œ", "")
        word = word.replace("â€˜", "")
        word = word.replace("*", "")

    bigrams = zip(a.split(), a.split()[1:])
    counts = Counter(bigrams)
    # print(counts.most_common())

    lst = counts.most_common(n_print)
    another_lst = []

    for i in lst:
        another_lst.append((i[0][0] + " " + i[0][1], i[1]))

    lst = another_lst

    df = pd.DataFrame(lst, columns=['Word', 'Count'])
    df.plot.bar(x='Word', y='Count', ax=ax, figsize=(14, 6))
    plt.xlabel('Word Pair')
    plt.ylabel('Word Pair Count')
```

```
    plt.title(label='Most Common Word Pair with stop words (Bigrams)',
                  fontweight=10,
                  pad='7.0')
    plt.tight_layout()

    mpld3.save_html(fig, "static/bigram_stopwords.html")


def get_top_n_bigram_wo_stop_words(corpus, n_print=None):
    corpus = preprocess(corpus)

    a = corpus
    padalu = []
    fig = figure()
    ax = fig.gca()

    for word in a.lower().split():
        word = word.replace(".", "")
        word = word.replace(",", "")
        word = word.replace(":", "")
        word = word.replace("\"", "")
        word = word.replace("!", "")
        word = word.replace("â€œ", "")
        word = word.replace("â€˜", "")
        word = word.replace("*", "")
        if word not in stopwords:
            padalu.append(word)
    bigrams = zip(padalu, padalu[1:])
    counts = Counter(bigrams)
    # print(counts.most_common())

    lst = counts.most_common(n_print)
    another_lst = []

    for i in lst:
        another_lst.append((i[0][0] + " " + i[0][1], i[1]))

    lst = another_lst

    df = pd.DataFrame(lst, columns=['Word', 'Count'])
    df.plot.bar(x='Word', y='Count', ax=ax, figsize=(14, 6))
    plt.xlabel('Word Pair')
    plt.ylabel('Word Pair Count')

    plt.title(label='Most Common Word Pair with out stop words (Bigrams)',
                  fontweight=10,
                  pad='7.0')
    plt.tight_layout()
    mpld3.save_html(fig, "static/bigram_wo_stopwords.html")
```

```
def get_top_n_trigram_stopwords(corpus, n_print=None):
    corpus = preprocess(corpus)
    a = corpus
    fig = figure()
    ax = fig.gca()

    for word in a.lower().split():
        word = word.replace(".", "")
        word = word.replace(",", "")
        word = word.replace(":", "")
        word = word.replace("\"", "")
        word = word.replace("!", "")
        word = word.replace("â€œ", "")
        word = word.replace("â€˜", "")
        word = word.replace("*", "")

    padalu = a.split()
    trigrams = zip(padalu, padalu[1:], padalu[2:])
    counts = Counter(trigrams)
    # print(counts.most_common())

    lst = counts.most_common(n_print)
    another_lst = []

    for i in lst:
        another_lst.append((i[0][0] + " " + i[0][1] + " " + i[0][2], i[1]))

    lst = another_lst

    df = pd.DataFrame(lst, columns=['Word', 'Count'])
    df.plot.bar(x='Word', y='Count', ax=ax, figsize=(14, 6))
    plt.xlabel('Word Pair')
    plt.ylabel('Word Pair Count')

    plt.title(label='Most Common Word trio with stop words (trigrams)',
              fontweight=10,
              pad='7.0')
    plt.tight_layout()

    mpld3.save_html(fig, "static/trigram_stopwords.html")


def get_top_n_trigram_wo_stop_words(corpus, n_print=None):
    corpus = preprocess(corpus)

    fig = figure()
    ax = fig.gca()
```

```
    a = corpus
    padalu = []
    for word in a.lower().split():
        word = word.replace(".", "")
        word = word.replace(",", "")
        word = word.replace(":", "")
        word = word.replace("\"", "")
        word = word.replace("!", "")
        word = word.replace("â€œ", "")
        word = word.replace("â€˜", "")
        word = word.replace("*", "")
        if word not in stopwords:
            padalu.append(word)
    trigrams = zip(padalu, padalu[1:], padalu[2:])
    counts = Counter(trigrams)
    # print(counts.most_common())

    lst = counts.most_common(n_print)
    another_lst = []

    for i in lst:
        another_lst.append((i[0][0] + " " + i[0][1] + " " + i[0][2], i[1]))

    lst = another_lst

    df = pd.DataFrame(lst, columns=['Word', 'Count'])
    df.plot.bar(x='Word', y='Count', ax=ax, figsize=(14, 6))
    plt.xlabel('Word trio')
    plt.ylabel('Word trio Count')

    plt.title(label='Most Common Word trio with out stop words (trigrams)',
              fontweight=10,
              pad='7.0')
    plt.tight_layout()
    mpld3.save_html(fig, "static/trigram_wo_stopwords.html")


def plot_pos(corpus):
    blob = TextBlob(corpus)
    fig = figure()
    ax = fig.gca()

    df = pd.DataFrame(blob.tags, columns=['word', 'pos'])
    df = df.pos.value_counts()[:20]
    df.plot.bar(x='word', y='pos', ax=ax, figsize=(14, 6))
    plt.xlabel('Parts Of Speech')
    plt.ylabel('Tags Count')

    plt.title(label='Top 20 Part-of-speech tagging for given text',
              fontweight=10,
```

```
                    pad='7.0')
    plt.tight_layout()
    mpld3.save_html(fig, "static/pos.html")


def generate_wordcloud(corpus):
    fig = figure()
    ax = fig.gca()

    wordcloud = WordCloud(width=3000, height=2000, random_state=1,
background_color='white', collocations=False,
                          stopwords=STOPWORDS).generate(corpus)
    plt.figure(figsize=(40, 30))
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout()
    plt.savefig("static/wordcloud.jpg")


def run(corpus):
    get_top_n_words_wo_stopwords(corpus, 20)
    get_top_n_words(corpus, 20)
    get_top_n_bigram_stopwords(corpus, 20)
    get_top_n_bigram_wo_stop_words(corpus, 20)
    get_top_n_trigram_stopwords(corpus, 20)
    get_top_n_trigram_wo_stop_words(corpus, 20)
    plot_pos(corpus)
    #generate_wordcloud(corpus)
```

7.4. Web based Search Module

```
import urllib
from fake_useragent import UserAgent
import re
from urllib.request import Request, urlopen
import wptools
import requests
import nltk
from bs4 import BeautifulSoup
import HelperTools as ht
import ExtractiveTextSum as ets


def get_google_box(query):
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186 Safari/537.36'}
    r = requests.get('https://www.google.com/search?q=' + query, headers=headers)
```

```
    soup = BeautifulSoup(r.text, 'lxml')

    result = soup.find('div', class_='Z0LcW')
    # result = soup.find('div', class_='gL9Hy')

    return (result.text)


def spelling_corrector(query):
    try:
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186 Safari/537.36'}
        r = requests.get('https://www.google.com/search?q=' + query,
headers=headers)

        soup = BeautifulSoup(r.text, 'lxml')
        result = soup.find('a', class_='gL9Hy')
        return (result.text)
    except:
        pass


def google_results(keyword, n_results):
    query = keyword
    query = urllib.parse.quote_plus(query)    # Format into URL encoding
    number_result = n_results
    ua = UserAgent()
    google_url = "https://www.google.com/search?q=" + query + "&num=" +
str(number_result)
    response = requests.get(google_url, {"User-Agent": ua.random})
    soup = BeautifulSoup(response.text, "html.parser")
    result = soup.find_all('div', attrs={'class': 'ZINbbc'})
    results = [re.search('\/url\?q\=(.*)\&sa', str(i.find('a', href=True)['href'])) for i in
result if "url" in str(i)]
    links = [i.group(1) for i in results if i != None]
    # remove youtube links
    sublinks = []
    for link in links:
        if link.find('www.youtube.com') == -1:
            sublinks.append(link)
    return (sublinks)


def infobox(item):
    so = wptools.page(item).get_parse()
    infobox = so.data['infobox']
    return infobox
```

```
def wikiresult(query):
        dic = infobox(query)
        for i in dic.keys():
                print(i, " : ", dic[i])


def mainu(query , flag):
        links = google_results(query, 25)
        print(links)
        for link in links:
                index = link.find("en.wikipedia.org/wiki/")
                print(link)
                if flag != '$' and index != -1:
                        wiki = link[index + 22:]
                        return infobox(wiki)

        else:
                try:
                        return {query: get_google_box(query)}
                except:
                        total_text = ""
                        for link in links:
                                try:

                                        total_text += ht.get_text_from_link(link)
                                except:
                                        pass
                        return {query: ets.Word_weight(total_text , 10 , 'en' )}
```

7.5. Helper Functions Module

```
import re
from googletrans import Translator
import nltk
from docx import Document
from pdfminer.high_level import extract_text
import bs4 as bs
import urllib.request
import wikipedia
from gensim.summarization import keywords
import RAKE
from nltk import tokenize
from operator import itemgetter
import math
import operator
import nltk
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

f = open('SmartStoplist.txt', 'r')
try:
	smartstoplist = f.read()
	# print(text)
except UnicodeDecodeError:
	pass

stopwords = nltk.corpus.stopwords.words('english')
stopwords.extend(smartstoplist.split('\n'))
stopwords = set(stopwords)


def text_file(file):
	rf = file[::-1].index('.')
	extension = file[-1 * rf:]
	print(extension)
	if extension == 'txt':

		f = open(file, 'r')
		try:
			text = f.read()
			# print(text)
		except UnicodeDecodeError:
			pass

	elif extension == 'docx':

		text = ''
		f = Document(file)
		for i in f.paragraphs:
			text += i.text

	elif extension == 'pdf':

		text = extract_text(file)

	else:
		print("Please upload correct file !!!")
		return 0

	return text


def get_text_from_link(urllink):
	if urllink.find('wikipedia.org') != -1:
		result = urllink.find('wikipedia.org/wiki/')
		wiki = wikipedia.page(urllink[result + 19:])
```

```
            text = wiki.content
            text = text.replace('==', '')
            text = re.sub(r'\[[0-9]*\]', ' ', text)
            text = re.sub(r'\s+', ' ', text)
            text = re.sub('[^a-zA-Z]', ' ', text)
            text = re.sub(r'\s+', ' ', text)

            # print(text)
            return text

    scraped_data = urllib.request.urlopen(urllink)
    article = scraped_data.read()

    parsed_article = bs.BeautifulSoup(article, 'lxml')

    paragraphs = parsed_article.find_all('p')

    article_text = ""

    for p in paragraphs:
        article_text += p.text

    # article_text = re.sub(r'\[[0-9]*\]', ' ', article_text)
    # article_text = re.sub(r'\s+', ' ', article_text)
    # formatted_article_text = re.sub('[^a-zA-Z]', ' ', article_text )
    # formatted_article_text = re.sub(r'\s+', ' ', formatted_article_text)
    print(article_text)
    return article_text


def keywords_code(text):
    return keywords(text).split('\n')


import TextVisualizations as tv
def get_keywords(doc , howmuch):
    doc=doc.lower()
    doc=re.sub('[^a-zA-Z" ".]+', '', doc)
    total_words = doc.split()
    total_word_length = len(total_words)
    # print(total_word_length, " twl")
    total_sentences = tokenize.sent_tokenize(doc)
    total_sent_len = len(total_sentences)
    # print(total_sent_len, " tsl")
    tf_score = {}
    for each_word in total_words:
        each_word = each_word.replace('.', '')
        if each_word not in stopwords:
            if each_word in tf_score:
                tf_score[each_word] += 1
```

```
                else:
                    tf_score[each_word] = 1
    # print(tf_score, " tf score")

    # Dividing by total_word_length for each dictionary element
    tf_score.update((x, y / int(total_word_length)) for x, y in tf_score.items())

    # print(tf_score, " update tf score")

    def check_sent(word, sentences):
        final = [all([w in x for w in word]) for x in sentences]
        sent_len = [sentences[i] for i in range(0, len(final)) if final[i]]
        return int(len(sent_len))

    # Step 4: Calculate IDF for each word
    idf_score = {}
    for each_word in total_words:
        each_word = each_word.replace('.', '')
        if each_word not in stopwords:
            if each_word in idf_score:
                idf_score[each_word] = check_sent(each_word, total_sentences)
            else:
                idf_score[each_word] = 1

    # Performing a log and divide
    idf_score.update((x, math.log(int(total_sent_len) / y)) for x, y in
idf_score.items())

    # print(idf_score, " update idf score ")
    tf_idf_score = {key: tf_score[key] * idf_score.get(key, 0) for key in
tf_score.keys()}

    # print(tf_idf_score, " tf idf score")

    # Get top N important words in the document
    def get_top_n(dict_elem, n):
        result = dict(sorted(dict_elem.items(), key=itemgetter(1), reverse=True)[:n])
        return result



    total=len(tf_idf_score.keys())
    howmuch= howmuch * total // 100
    return get_top_n(tf_idf_score , howmuch)

def sort_tup(tup):
    tup.sort(key=lambda x: x[1])
    return tup

def get_keyphrases(text , howmuch):
```

```
    stop_dir = "smartstoplist.txt"
    rakeobj = RAKE.Rake(stop_dir)
    keywords = sort_tup(rakeobj.run(text))
    keywords=keywords[::-1]
    total=len(keywords)
    howmuch= howmuch * total // 100
    lst=[]
    for i in range(howmuch):
        lst.append(keywords[i][0])
    return lst

def langtranslate(text , des):

    translator = Translator()
    translation = translator.translate(text, dest=des)

    return (translation.text)

def detect_lang(text):
    translator=Translator()
    return (translator.detect(text)).lang
```

7.6. User Interface Images

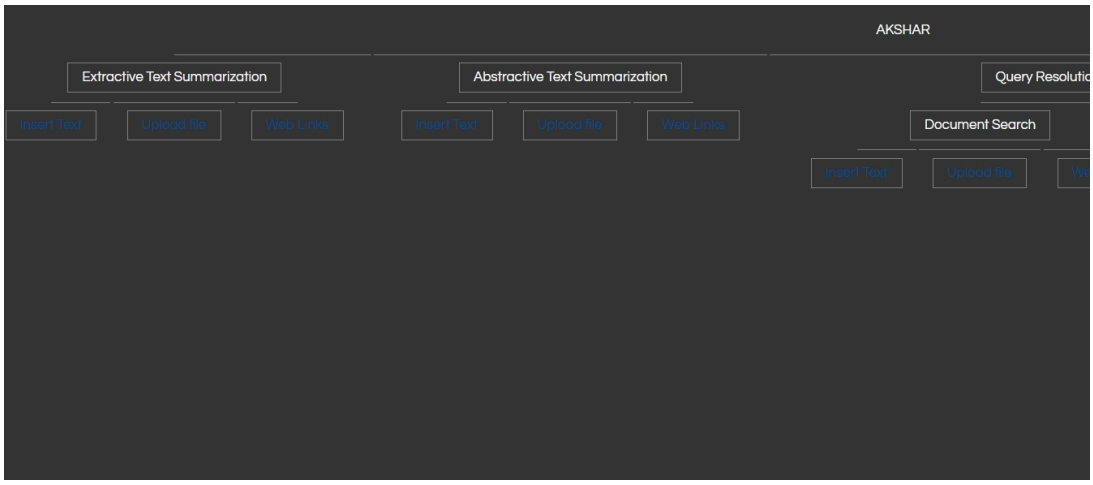

Fig 7.1. Home page part - 1

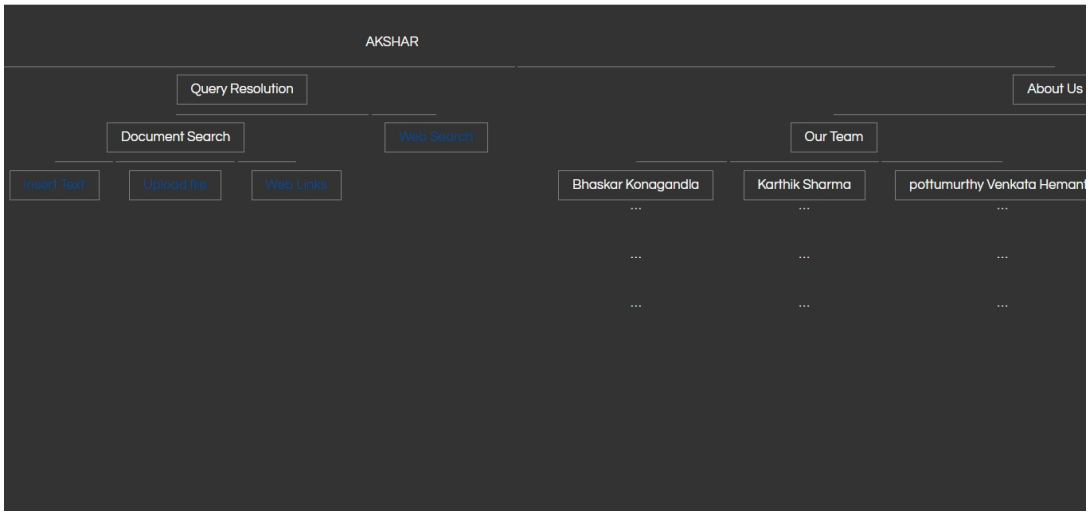

Fig 7.2. Home page part 2

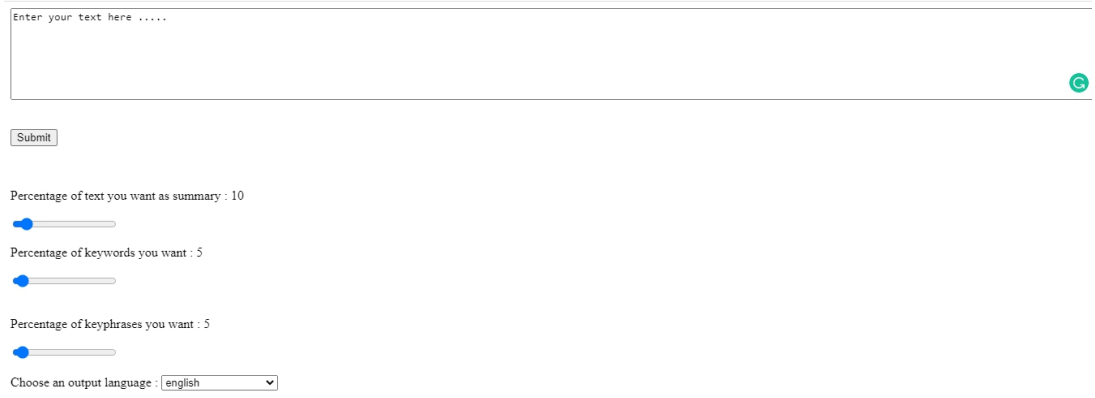

Fig 7.3. Text summarization - Iserting text

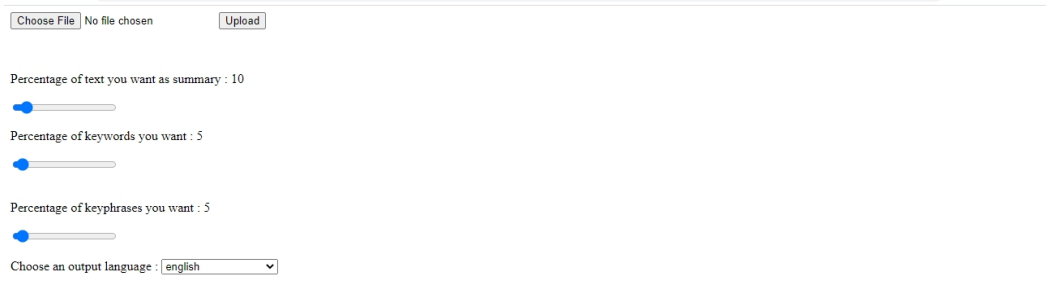

Fig 7.4. Text Summarization - upload file type

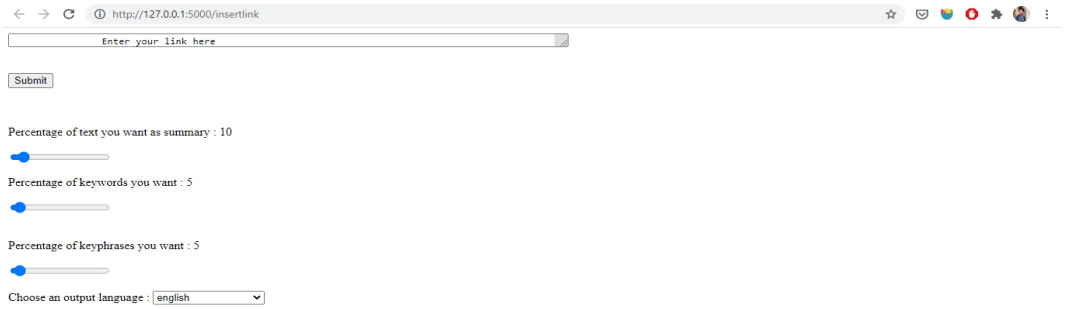

Fig 7.5. Text Summarization - Insert Link type

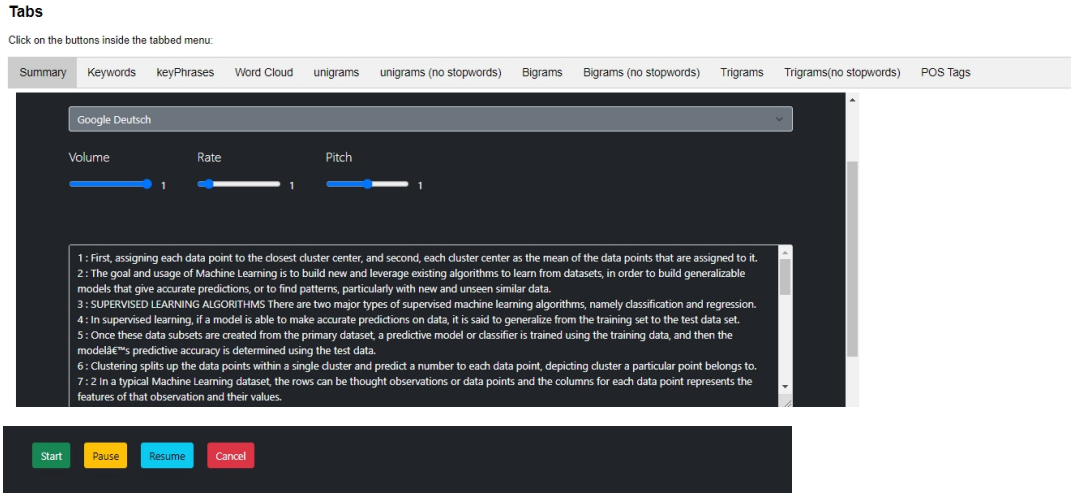

Fig 7.6. Text Summarization - Result page

Fig 7.7. keywords

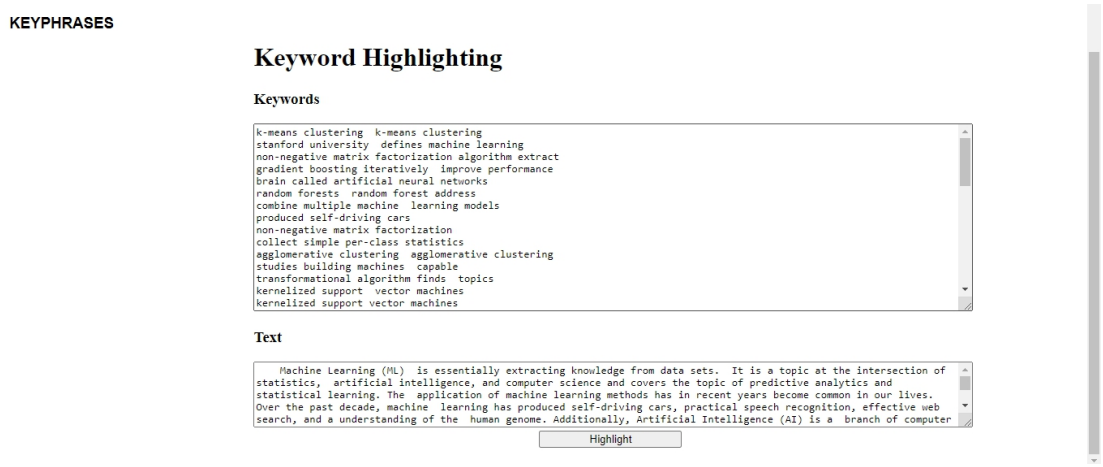

Fig 7.8. Keyphrases

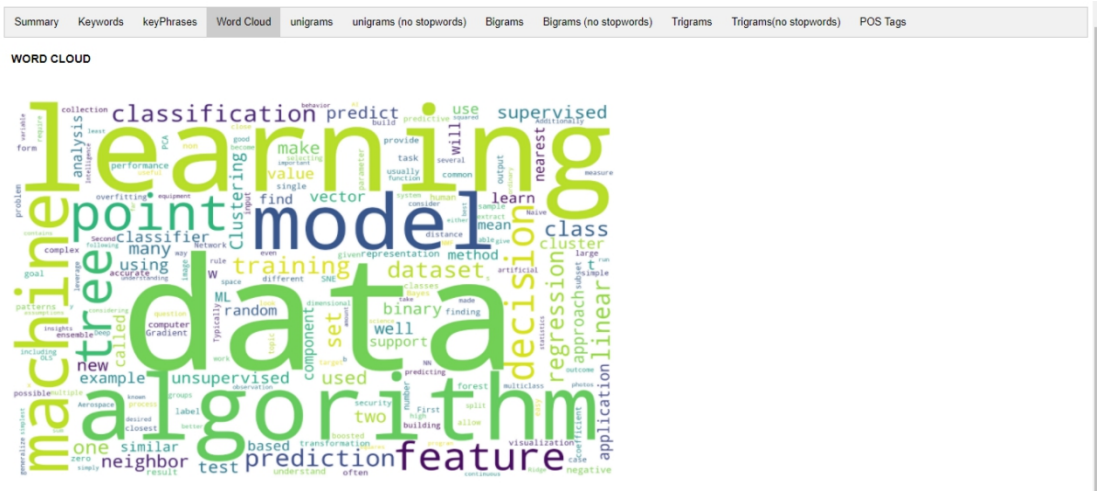

Fig 7.9. Word cloud

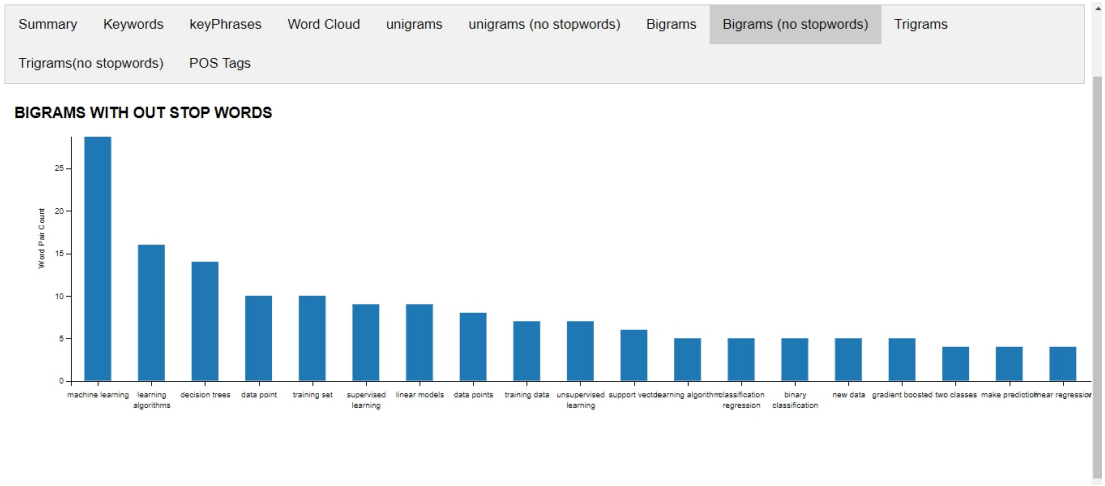

Fig 7.10. Histograms for n - grams (here n = 1 , 2, 3)

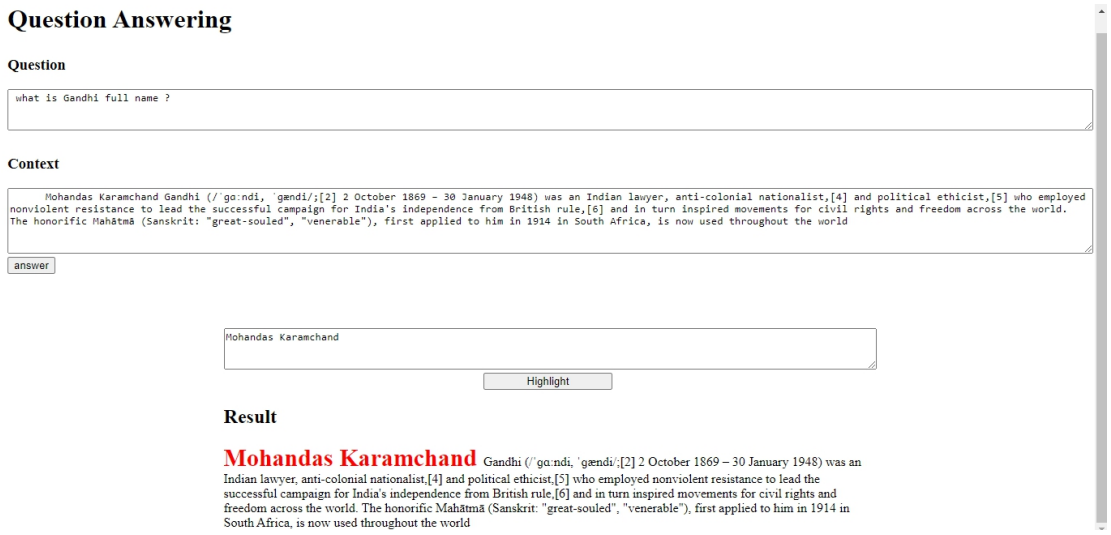

Fig 7.11. Extractive Question Answering

Fig 7.12. Web Based Question Answering

8. TESTING

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defect of application. The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing furnishes comparison that compares the behavior and state of software against mechanisms because the problem can be recognized by the mechanism. The mechanism may include past versions of the same specified product, comparable products, and interfaces of expected purpose, relevant standards, or other criteria but not limited up to these.

Testing includes an examination of code and also the execution of code in various environments, conditions as well as all the examining aspects of the code. In the current scenario of software development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development.

The success of software depends upon acceptance of its targeted audience, easy graphical user interface, strong functionality load test, etc.

Manual testing is done in this project .

The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing. While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

8.1. White Box Testing

In white-box testing, the developer will inspect every line of code before handing it over to the testing team or the concerned test engineers.
Subsequently, the code is noticeable for developers throughout testing; that's why this process is known as WBT (White Box Testing).
In other words, we can say that the developer will execute the complete white-box testing for the particular software and send the specific application to the testing team.
The purpose of implementing the white box testing is to emphasize the flow of inputs and outputs over the software and enhance the security of an application.
In white box testing the following tests are done :

- ✓ Path testing
- ✓ Loop testing
- ✓ Condition testing
- ✓ Testing based on the memory perspective
- ✓ Test performance of the program

8.2. Black box testing

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.
In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Decision table technique in Black box testing

Input by user	T	T	F	F
Correct format	T	F	T	F
Expected Behaviour	Visualizations page / Answer page	Error page	Not possible	Not possible

Table 8.1. Decision Table Testing

8.3. Functional Testing

It is a type of software testing which is used to verify the functionality of the software application, whether the function is working according to the requirement specification. In functional testing, each function tested by giving the value, determining the output, and verifying the actual output with the expected value. Functional testing performed as black-box testing which is presented to confirm that the functionality of an application or system behaves as we are expecting. It is done to verify the functionality of the application.

The purpose of the functional testing is to check the primary entry function, necessarily usable function, the flow of screen GUI. Functional testing displays the error message so that the user can easily navigate throughout the application.

8.4. Unit Testing

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with its performance.

A unit is a single testable part of a software system and tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. White box testing approach used for unit testing and usually done by the developers.

Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or one by one, and this process is known as Unit testing or components testing.

8.5. Integration testing

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.
Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.

8.6. System Testing

System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software (any software is only a single element of a computer system). The software is developed in units and then interfaced with other software and hardware to create a complete computer system. In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

To check the end-to-end flow of an application or the software as a user is known as System testing. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.
It is end-to-end testing where the testing environment is similar to the production environment

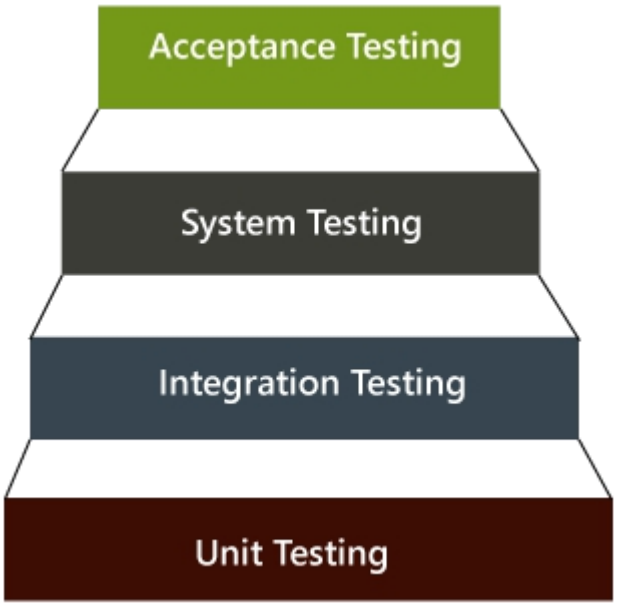

Fig 8.1. Testing

8.7. Acceptance testing

Acceptance testing is formal testing based on user requirements and function processing. It determines whether the software is conforming specified requirements and user requirements or not. It is conducted as a kind of Black Box testing where the number of required users involved testing the acceptance level of the system.User acceptance testing (UAT) is a type of testing, which is done by the customer before accepting the final product. Generally, UAT is done by the customer (domain expert) for their satisfaction, and check whether the application is working according to given business scenarios, real-time scenarios.

In this, we concentrate only on those features and scenarios which are regularly used by the customer or mostly user scenarios for the business or those scenarios which are used daily by the end-user or the customer.

However, the software has passed through three testing levels (Unit Testing, Integration Testing, System Testing) But still there are some minor errors which can be identified when the system is used by the end user in the actual scenario.

Acceptance testing is the squeezing of all the testing processes that have done previously.

8.8. Test Cases

Test case id	Test case name	Test case description	Steps	Expected	actual	Priority
1	Upload file	Need to identify the file type	Identify and extract	Text is returned as output	Works fine	high
2	Insert link	Can scrape the data or not	Go to the web page and retrieve	Text is returned	Works fine	High
3	Insert text	Able to take it	Receive it from user	Text received	Works fine	High
4	Insert keywords	Will it process it and highlight in the context	Highlight each word in the context	Display highlighted words	Works fine	Medium
5	Insert keyphrases	Will it process it and highlight in the context	Highlight each phrase in the context	Display highlighted phrases	Works fine	Medium
6	Pre process text	Remove unwanted source related formatting	Remove using regular expressions	Receive plain text without source formatting	There are small residues.	Medium
7	Illegal web links	Check the behaviour	Give wrong links	Error page	Works fine	Low

Table 8.2 Test cases

9. OVERHEADS OF THE PROJECT

9.1. Challenges

As the text summarization provides the reduced subtext of the original text there are always a lot of challenges there measure if we get the required text summary from the large document or not. If, the summary contains important sentences and words or not. Some of the challenge of text summarization are discussed as follow:

Extract hidden semantic relationship

In the text, there are many sentences that are related to each other. The text includes many sentences. Some sentences in the document related to each based on have semantic relationships between concepts in the text. Many sentences are related to each other that depict some particular details of the text document. So, capturing such sentences in the summary is always challenging task.

Relevance detection

While generating a summary of the text it is also important to find the relevant sentences in our text documents so that they can be selected to make up the final summaries. This is always a challenging task to find out the relevant sentences in text so that they can be included in our final summary. As it highly affects the quality of our summary. “The Code Quality Principle” can be used to detect important sentences in our text document. Different criterion like sentence position within the text, word and phrases frequencies and title overlap are some of the examples to ensure the relevance of the sentence.

Appearance of pronouns during extractive text summarization which will make the summary unclear.

If unnecessary topic is also present in the source text , there are chances of getting them during the summary generation if topic is taken care then this problem can be taken care to some extent.

10. CONCLUSION & FUTURE SCOPE

As the availability of data in the form of text increasing day by day. It becomes so difficult to read the whole textual data in order to find the required information which is both difficult as well as a time-consuming task for a human being. So, at that time ATS performs an important role by providing a summary of a whole text document by extracting only the useful information and sentences. There are different approaches of text summarization. The real-world applications of text summarization can be: documents summarization, news and articles summarization, review systems, recommendation systems, social media monitoring, survey responses systems.

The future goals of this project is to reduce the inefficiencies in the summaries by enhancing the the techniques used for overcoming the challenges as stated, by employing more effective methods. Make much more useful visualizations for the gaining the best out of the context. Better the techniques used for scraping the web during text extraction form internet. Train state of the art models for extractive answering and query resolution for appropriate answering of the questions.The ultimate aim is fine tune every aspect of the project and make overall performance as close as possible to humans.

11. BIBLIOGRAPHY

1) https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/
2) https://en.wikipedia.org/wiki/Automatic_summarization
3) https://www.kdnuggets.com/2019/01/approaches-text-summarization-overview.html