Table of Contents

1. Introduction 1

**1.1 Project Objectives 1**

**1.2 Project Overview 3**

**1.3 Project Scope 3**

2. System Analysis 4

**2.1 Existing System 4**

**2.2 Proposed System 2**

**2.3 Software Requirement Specification 3**

3. System Design 7

**3.1 Input-Output Design 7**

**3.2 Database Design 8**

**3.3 System Tools 8**

4. System Implementations 8

5. Results 12

6. System Testing 13

7. Future Scope 13

8. Conclusion 13

9. References 14

# Introduction

## Project Objectives

The purpose of the project is to implement machine learning models to summarize a paragraph which is given as an input to a flask webapp.

## Project Overview

Automatic Text Summarization, is the procedure in which, a text is reduced to create a summary from an original document. Reduction of text is a very complex problem which, in spite of the progress in the area thus far, poses many challenges to the scientific community. It is also a relevant application in today's information society given the exponential growth of textual information online and the need to promptly assess the contents of text collections. It has long been assumed that summarization presupposes to understand the input text, which means for identifying the important point of the document, explicit (semantic) representation of the text must be calculated therefore, text summarization became an interesting application to test the understanding capabilities of artificial systems.

## Project Scope

Text Summarization is a project model based on machine learning models used for summarizing paragraphs using a web console. The model allows users to copy/type text into the box and then the webpage gives out a summary of that text. The aim of this project is to make model to summarise texts which would be very useful for automated systems where summary generation doesn’t require human interference. The project uses HTML, CSS, JavaScript and Bootstrap for the generation of front end while the connectivity has been done using Flask which is well known and widely use framework of python.

With push notifications and article digests gaining more and more traction, the task of generating intelligent and accurate summaries for long pieces of text has become a popular research as well as industry problem. There are two fundamental approaches to text summarization:

* Extractive
* Abstractive.

The former extracts words and word phrases from the original text to create a summary. The latter learns an internal language representation to generate more human-like summaries, paraphrasing the intent of the original text.

# System Analysis

## Existing System

We intend to make a webapp to show the working of the text summarization. The algorithm and working principles behind the app can be used in various other applications like novels summarization, caption generation and recaps. Some of the previous works are as follows:

* [Implementation and Evaluation of Evolutionary Connectionist Approaches to Automated Text Summarization](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.671.7415)
* [Text Summarization in Python: Extractive vs. Abstractive techniques revisited](https://rare-technologies.com/text-summarization-in-python-extractive-vs-abstractive-techniques-revisited/#text_summarization_in_python)
* [Automatic text summarization and it's methods - a review](https://ieeexplore.ieee.org/abstract/document/7508049/keywords#keywords)

## Proposed System

There are two fundamental approaches to text summarization:

* Extractive
* Abstractive.

The main function of the webapp is to take paragraph with more than 200 words, processes it and gives an output which is the summary of that paragraph which uses abstractive model of summarization.

## Software Requirement Specification

# Introduction

## Purpose

The purpose of the project is to utilize various machine learning models to summarize paragraphs from a web application.

## Intended Audience and Reading Suggestions

Our application is intended for

## Document Conventions

NLTK – Natural Language Tool Kit

IDE – Integrated Development Environment

HTML – Hyper Text Markup Language

OS – Operating System

NLP – Natural Language Processing

CSS – Cascading Style Sheets

GPU – Graphics Processing Unit

## Product Scope

Text summarization is an incipient practice for verdict out the summary of the text article. Due to the enormous aggregate of information getting augmented on internet; it is challenging for the user to verve through altogether the information accessible on web. The large availability of internet content partakes constrained a broad research area in the extent of automatic text summarization contained by the Natural Language Processing (NLP), especially statistical machine learning communal. Through this paper, we intend to show two ways by which the text summarization is done by machine learning models. There are two fundamental approaches to text summarization: extractive and abstractive

# Overall Description

## Product Perspective

We intend to make a webapp to show the working of the text summarization. The algorithm and working principles behind the app can be used in various other applications like novels summarization, caption generation and recaps. The product is particularly useful in the field of article summarization where we can have any kind of article that is required to be summarized which can be scraped and summarized by our model.

## Product Functions

The main function of the webapp is to take paragraph with more than 200 words, processes it and give an output which is the summarization of that paragraph.

## User Classes and Characteristics

Any user looking for ways to summarize huge passages in novels, news articles, papers etc can use this app to have a quick overview of that passage. Technical knowledge required to use our module is little none. Any user with knowledge on how to browse the web will be able to accomplish the task.

Another version of our module is available as a Python library which enables our summarization methods to be used programmatically.

## Operating Environment

It requires a server-side operating system which can be Windows or Linux based OS.

Client can be any device which can access the internet using a JavaScript enabled web browser.

## Design and Implementation Constraints

For extractive summarization, we require a minimum of 200 words with sentences that are separated with commas and full stops in order to get ideal results.

For abstractive summarization, due to lack of GPU resources and a lot of parameters to tune abstractive summarization is able to generate only 2-3 words summary for paragraphs.

## Assumptions and Dependencies

For making the model we used the following libraries and dependencies: -

* numpy
* Pandas
* NLTK – Natural Language Tool Kit
* Bootstrap
* JQuery

# External Interface Requirements

## User Interfaces

The GUI is made using HTML and CSS which clearly defines a text area where the paragraphs can be typed/copied on to. Then a summarize button is provided right below the text paragraph becomes available once the 200 words mark is crossed. Upon clicking this button, below it the original text which was given as into and the summarized paragraph become visible on the page.

## Software Interfaces

We used the following software to develop the app:

* Jupyter Notebook: is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. It was used for making the initial abstractive summarization model
* Google Colaboratory: is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser. It was also used for making the initial extractive summarization model.
* Flask App: is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.
* Bootstrap HTML, CSS libraries along with jQuery libraries were used for making the webpage
* Goorm IDE: **IDE** stands for Integrated Development Environment. It refers to software that provides an environment in which all tasks related to program development, such as coding, debugging, and compilation, are processed in a single program. The cloud IDE eliminates the complexity of building development environment. If you are not dependent on a computer and can only run a web browser, you can use the IDE immediately.

# System Features

The entire program functions on a flask server. Since flask is web framework which helps managing pages, performing functions on a set of inputs, the entire handling requires only python coding which includes receiving POST and GET request from the browser using jinja which is also managed by flask. Jinja is a modern and designer-friendly templating language for Python, modelled after Django’s templates. It is fast, widely used and secure with the optional sandboxed template execution environment. This is how it is written into the html page in order to execute python codes.

<**title**>{% **block** title %}{% **endblock** %}</**title**>

<**ul**>

{% **for** user **in** users %}

<**li**><**a** href="{{ user.url }}">{{ user.username }}</**a**></**li**>

{% **endfor** %}

</**ul**>

# Other Nonfunctional Requirements

## Performance Requirements

We are going to perform the project on UBUNTU platform so we need the OS as UBUNTU. Any version of Ubuntu preferably Ubuntu 18.04 or Ubuntu 16.0 or Windows 10 would suffice the necessary operating system requirements.

The system should have minimum ram of 4GB as well as minimum storage capacity of 100GB.

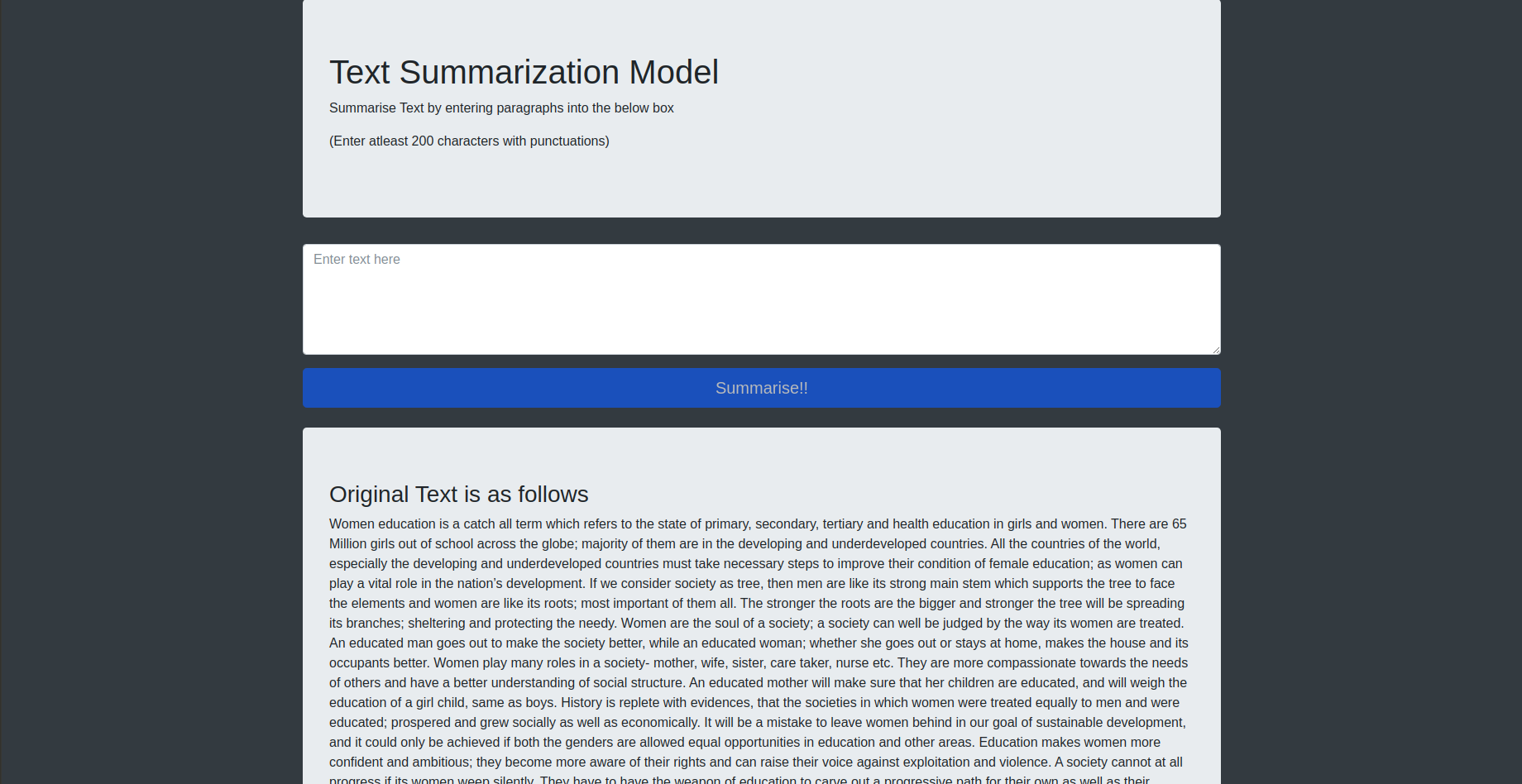
The various software, programs, packages and libraries used in the project should also be installed in the system including NLTK, Flask and required compilers and editors for Python, HTML, CSS, JavaScript and Bootstrap.

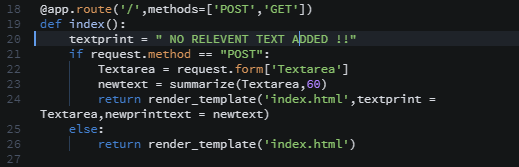
# System Design

## Input-Output Design

The input is taken in the form of a text box in HTML and handled by a python script which runs preprocessing and feeds it to a model which generates a text summary for the same which is displayed as text on the webpage as a summary output. This text input is then processed through the summarize module. The following code snippets will elaborate clearly how the text is taken in:







## Database Design

The only channel of data used is a text box which passes the text from client to the server-side script. There is no type of data collection happening in order to respect the privacy of the users.

## System Tools

The data is communicated over the internet or a localhost through recent TCP/IP protocols. The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server. A web browser may be the client, and an application on a computer that hosts a web site may be the server. A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content. This is done via POST and GET methods.

# System Implementation

The system has two modules:

* The algorithm to summarize any given string.
* The app that accepts a string calls upon that algorithm to summarize that string.

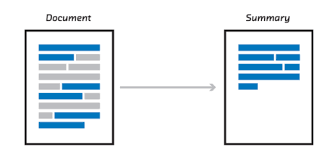
Before the algorithm is called upon to summarize the string, a set of preprocessing operations. After the preprocessing the data is given to the model which generates the summary.

**Extractive Text Summarization**

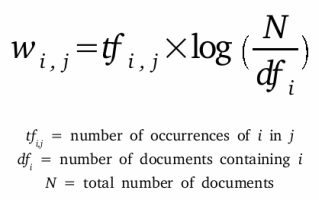
Text Summarization in Gensim

Gensim summarization module implements TextRank, an unsupervised algorithm based on weighted-graphs. TextRank works as follows:

* Pre-process the text: remove stop words and stem the remaining words.
* Create a graph where vertices are sentences.
* Connect every sentence to every other sentence by an edge. The weight of the edge is how similar the two sentences are.
* Run the PageRank algorithm on the graph.
* Pick the vertices(sentences) with the highest PageRank score



The score for each word is calculated by TF-IDF (Term Frequency Inverse Document Frequency). TF-IDF is calculated as follows:



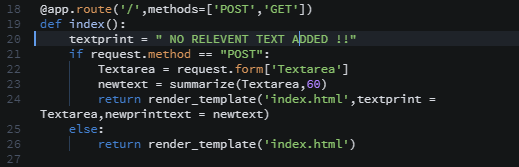
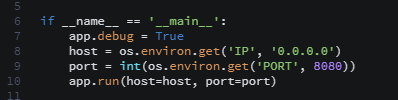
First, TF-IDF measures the number of times that words appear in a given document (that’s “term frequency”). But because words such as “and” or “the” appear frequently in all documents, those must be systematically discounted. That’s the inverse-document frequency part. The more documents a word appears in, the less valuable that word is as a signal to differentiate any given document. That’s intended to leave only the frequent AND distinctive words as markers. Each word’s TF-IDF relevance is a normalized data format that also adds up to one.

**Flask Web Framework**

Flask (source code) is a Python web framework built with a small core and easy-to-extend philosophy.

Flask is considered more Pythonic than the Django web framework because in common situations the equivalent Flask web application is more explicit. Flask is also easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running.

For example, here is a example from our web application with Flask:



The above code shows ‘index.html’ on localhost port 5000 in a web browser when run with the python app.py command and the Flask library installed.

* @app.route: Route definition is the simple act of assigning URLs to functions containing view logic.
* Render\_template( ): Templates are files that contain static data as well as placeholders for dynamic data. A template is rendered with specific data to produce a final document. Flask uses the Jinja template library to render templates.

So, we get the input text through the HTML using jinja using the following snippet of code:

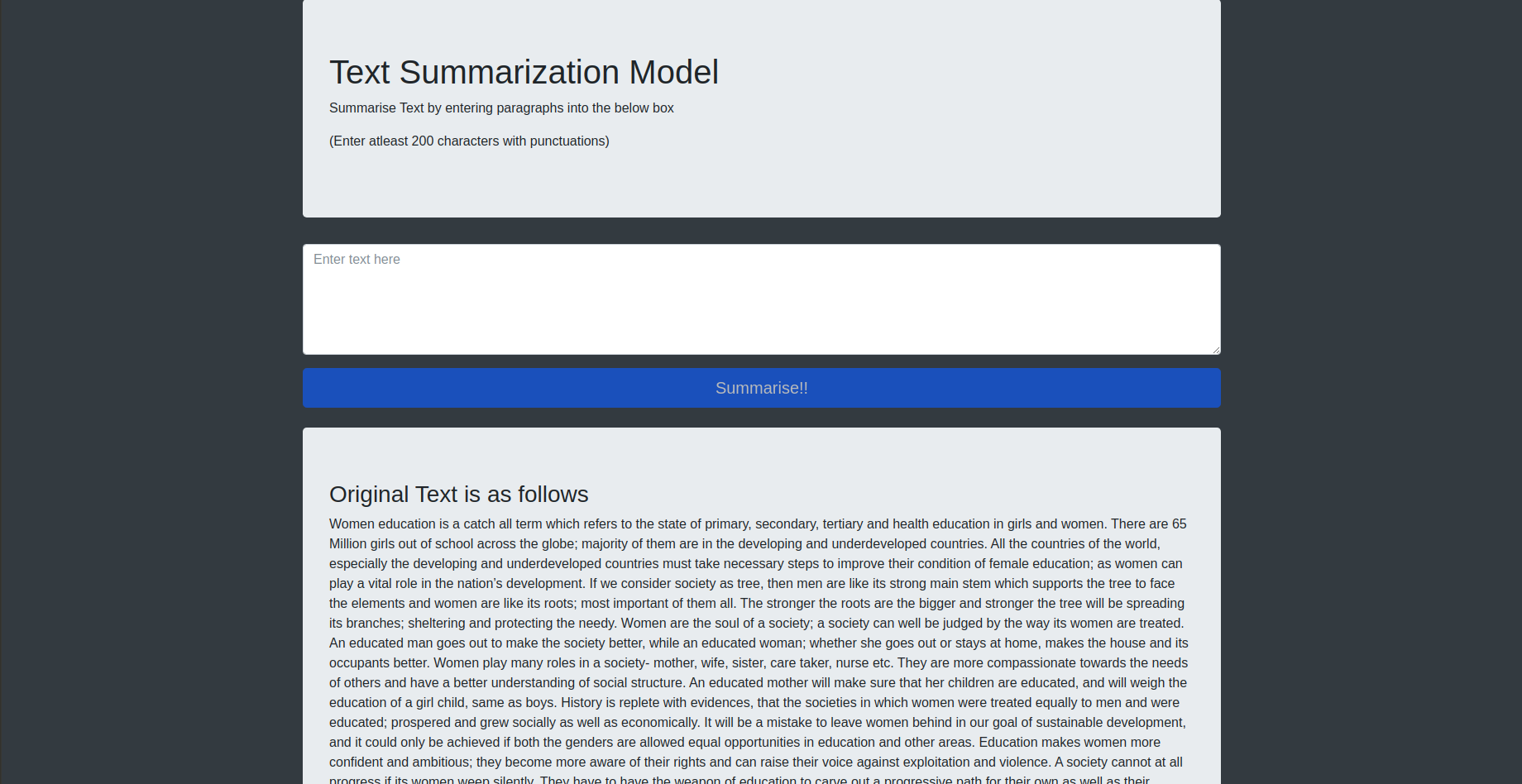
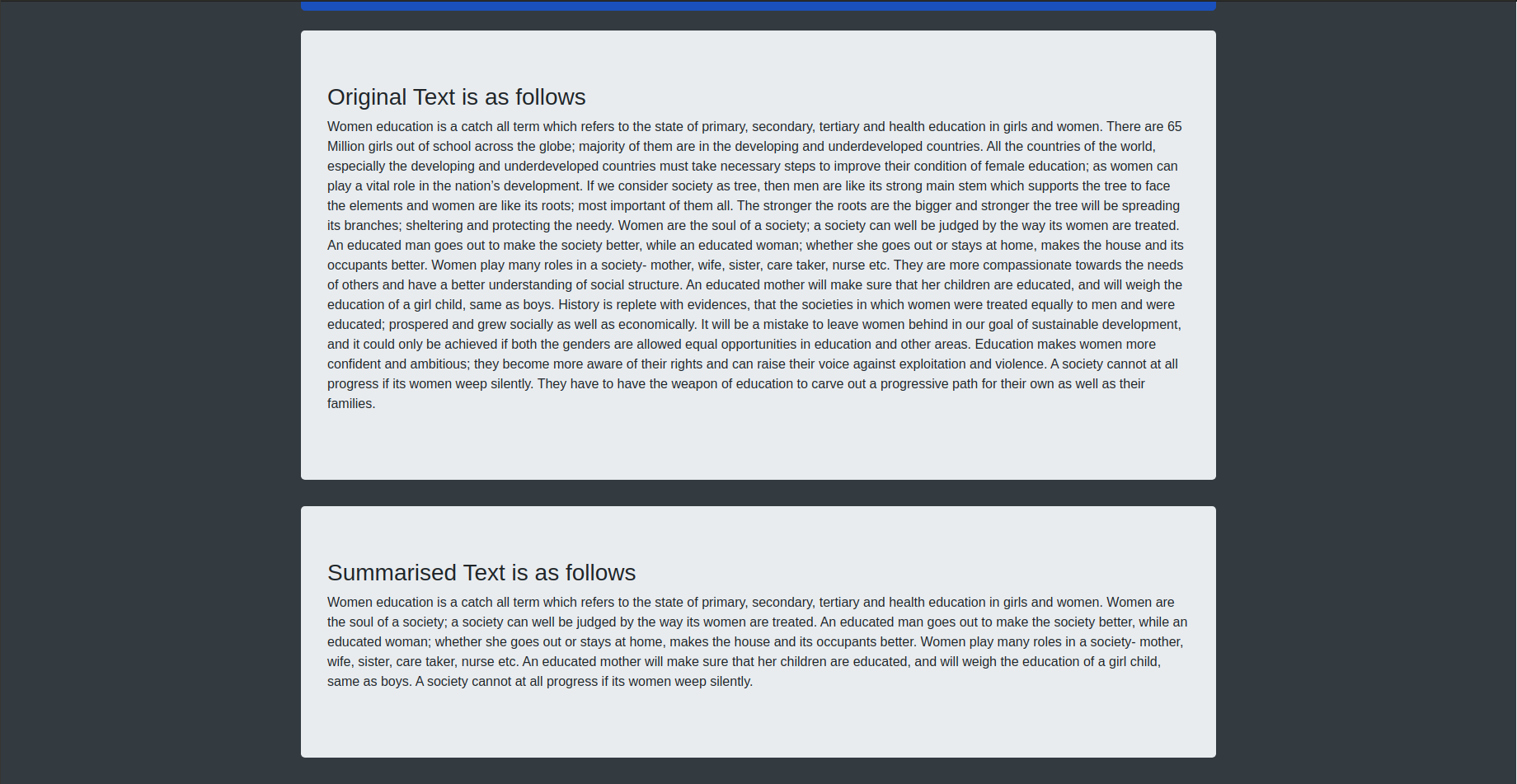
<textarea value="{{request.form.Textarea}}"></textarea>

The highlighted text is jinja snippet for holding strings in variables in the flask web framework. This string is passed on using POST method. Then this string is processed through the machine learning model and then sent back through GET method into the HTML page to display the processed result.

The following jinja snippet code shows how to get the info from the flask web framework:



# Results

**

# System Testing

We performed unit testing on the program where in module was tested on Jupyter notebook before it was integrated with the web framework.

The flask web frameworks testing was done on Goorm IDE to check whether

* The GUI is suitable and eligible to the users.
* The instructions provided on how the system is to be used is clearly understood
* The webpage works smoothly and as intended.

Then an acceptance testing was performed where the end users where our own batchmates. Many bugs where identified and rectified immediately.

# Future Scope

The proposed work does not involve a knowledge base and therefore can be used to summarize articles from fields as diverse as politics, sports, current affairs, and finance. However, it does cause a tradeoff between domain independence and a knowledge-based summary which would provide data in a form more easily understandable to the human user. A possible application of this work can be made to make data available on the move on a mobile network by even shortening the sentences produced by our algorithm and then shortening it. Various NLP based algorithms can be used to achieve this objective. Thus, we would first produce a summary by sentence extraction from various documents and then abstractive methods are employed to shorten those sentences produced. This will ensure that the summary produced is to the highest condensed form which can be made in the mobile industry.

# Conclusion

The system has developed an automatic text summarization system using natural language

processing technique.

The state-of-the-art summarization systems are all extractive in nature, but the community

is gradually progressing towards abstractive summarization. Although a complete

abstractive summarization would require deeper natural language understanding and

processing, a hybrid or shallow abstractive summarization can be achieved through

sentence compression and textual entailment techniques.

Research in summarization continues to enhance the diversity and information richness,

and strive to produce coherent and focused answers to user’s information need.

# References

The following links played an important role in completing our project:

* <https://rare-technologies.com/text-summarization-in-python-extractive-vs-abstractive-techniques-revisited/#text_summarization_in_python>
* <https://ieeexplore.ieee.org/abstract/document/7508049/keywords#keywords>
* <https://www.hindawi.com/journals/tswj/2016/1784827/>
* <http://home.iitk.ac.in/~soumye/cs498a/report.pdf>
* <http://flask.palletsprojects.com/en/1.1.x/tutorial/templates/>