**QuantumClarity: Cutting-Edge Summarization for Quantum Computing Texts**

In recent years, quantum computing has emerged as a groundbreaking field, offering unprecedented computational capabilities. The "Quantum Processing - Text Dataset" provides valuable text content discussing a significant development in this domain. However, with approximately 482 words, the text is dense and complex, posing challenges for quick comprehension and extraction of key insights. The goal of this project is to develop an effective text summarization solution that can condense the content into a concise and coherent summary, enabling researchers, engineers, and enthusiasts to quickly grasp the essential information without losing the depth and significance of the original discussion.

**Scenario 1- Research Paper Reviews:**

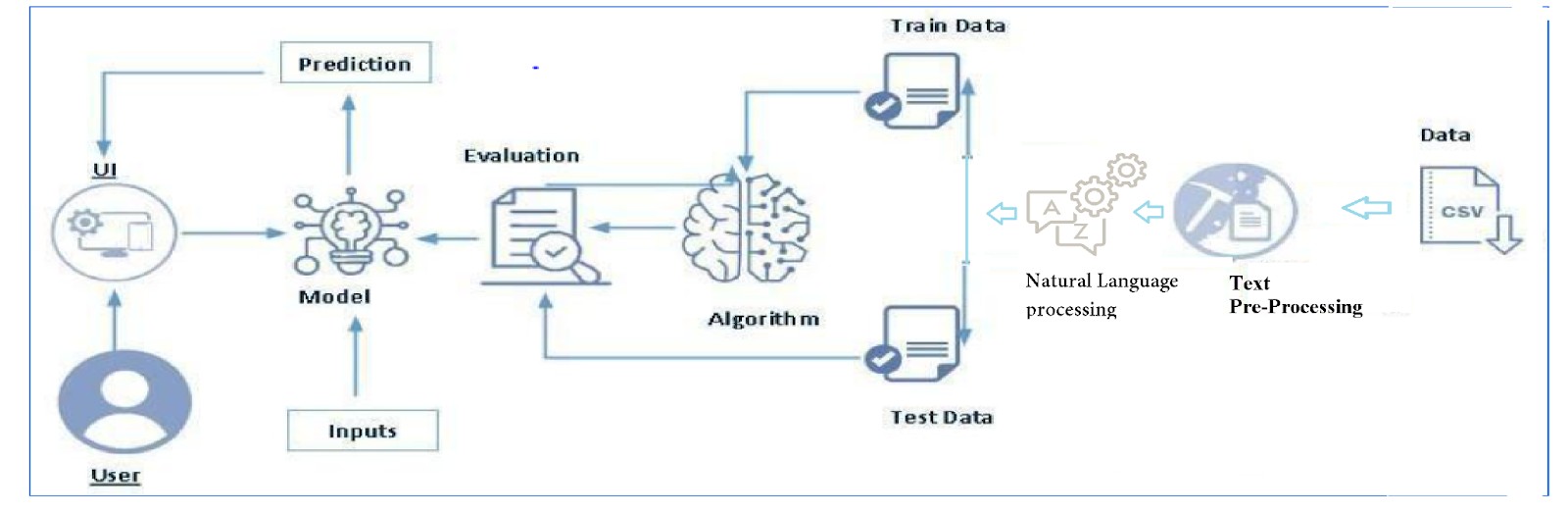
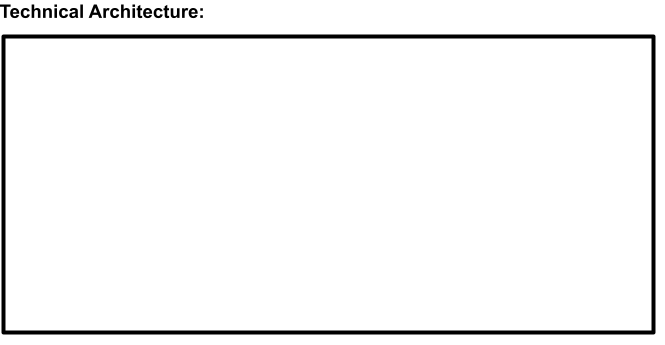
Researchers need to review multiple quantum computing research papers quickly. By summarizing the content, they can rapidly identify which papers are most relevant to their work, saving valuable time.

**Scenario 2- Conference Briefings:**

During quantum computing conferences, participants are often presented with numerous technical papers. Real-time summarization can help organizers provide concise overviews of each paper to attendees, facilitating quicker understanding and discussion.

**Scenario 3- News Aggregation:**

Journalists covering advancements in quantum computing could use the summarization tool to generate brief summaries of technical articles, making complex developments more accessible to a broader audience.



**Project Flow:**

* A user interacts with the UI to enter the input.
* Entered input is analyzed by the model which is integrated.
* Once the model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

* Data Collection & Preparation
  + Collect the dataset
  + Text Preprocessing
* Model Building
  + Vectorize text
  + Summarizing model algorithms
  + Testing the model
* Model Deployment
  + Integrate with Web Framework

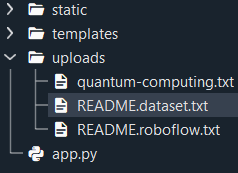
**Prior Knowledge:**

You must have prior knowledge of the following topics to complete this project.

* NLP:-https://[www.javatpoint.com/nlp](http://www.javatpoint.com/nlp)
* Flask Basics: <https://www.youtube.com/watch?v=lj4I_CvBnt0>

**Project Structure:**

Create the Project folder which contains files as shown below



* We are building a Flask application that needs HTML pages stored in the templates folder and a Python script app.py for scripting.
* mnb\_model. joblib is our saved model. Further, we will use this model for flask integration.
* The training folder contains a model training file

# Milestone 1: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

## Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project, we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/datasets/aruneshhh/quantum-processing-text-dataset>

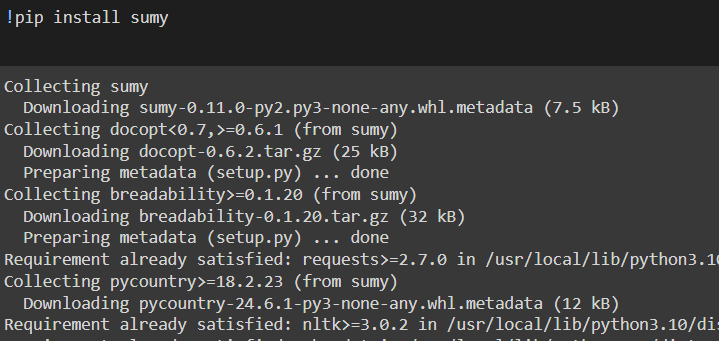
As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

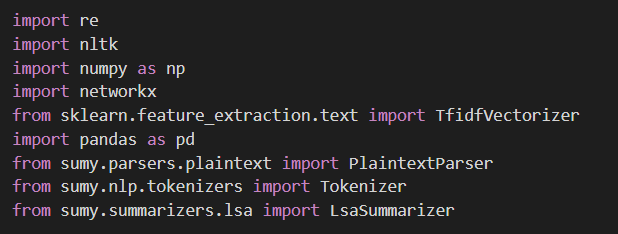
**Note:** There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

## Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.





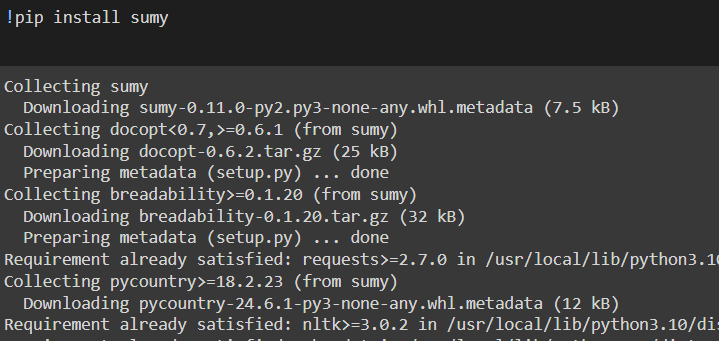


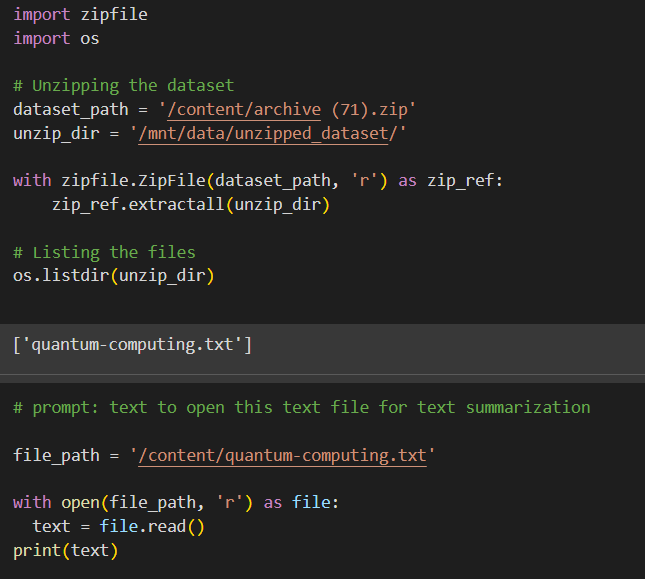
* pandas as pd: Imports the pandas library, which provides data structures and data analysis tools for Python.
* seaborn as sns: Imports the seaborn library, which is used for statistical data visualization based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
* matplotlib.pyplot as plt: Imports the pyplot module from the matplotlib library, which is used for creating static, animated, and interactive visualizations in Python.
* nltk: Imports the Natural Language Toolkit (NLTK), which is a leading platform for building Python programs to work with human language data.
* nltk.tokenize.word\_tokenize: Imports the word\_tokenize function from NLTK, which is used to split text into words or tokens.
* sklearn.feature\_extraction.text.TfidfVectorizer: Imports the TfidfVectorizer class from scikit-learn, which is used to convert a collection of raw documents into a matrix of TF-IDF features.

## Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas, we have a function called read\_csv() to read the dataset. As a parameter, we have to give the directory of the CSV file



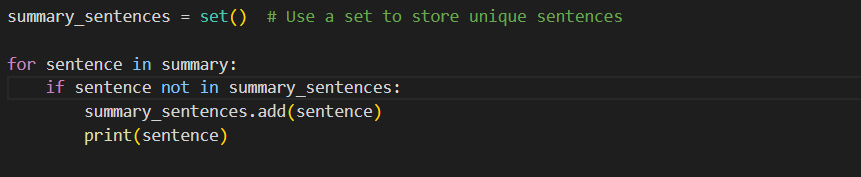


## MILESTONE 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly to fetch good results. This activity includes the following steps.

Keeping the unique sentences:



## Activity 2.1: Text Preprocessing

## Text pre-processing (python packages)

Text pre-processing is a crucial step in Natural Language Processing (NLP) and Information Retrieval (IR) tasks. The goal is to convert raw text into a more meaningful and manageable representation for further analysis.

In Python, several packages provide support for text pre-processing operations. Some of the most common ones are:

**NLTK (Natural Language Toolkit)**-

It is one of the most widely used NLP libraries in Python. It provides tools for tokenization, stemming, lemmatization, stop-word removal, and more.



In text preprocessing, we first convert all text to lowercase for uniformity. Then, we tokenize the

## text into smaller units like words or sentences. Special characters and numbers are removed to

## eliminate noise. Common words, known as stopwords, are also removed. Lemmatization

## further refines the text by reducing words to their base forms. Tokens are rejoined into text,

## and empty rows are handled. Finally, the preprocessed data is ready for analysis.

## Lowercasing: Converting all text to lowercase to ensure consistency in word representations.

## Tokenization: Breaking down the text into smaller units such as words, phrases, or sentences.

## Removing Stopwords: Removing common words like "and", "the", and "is" that occur

## frequently in the language but usually don't contribute much to the meaning of the text.

## Stemming or Lemmatization: Reducing words to their base or root form to normalize

## variations. Stemming chops off prefixes or suffixes, while lemmatization maps words to

## their dictionary form.

## Removing special characters and numbers: Dealing with numerical values by either

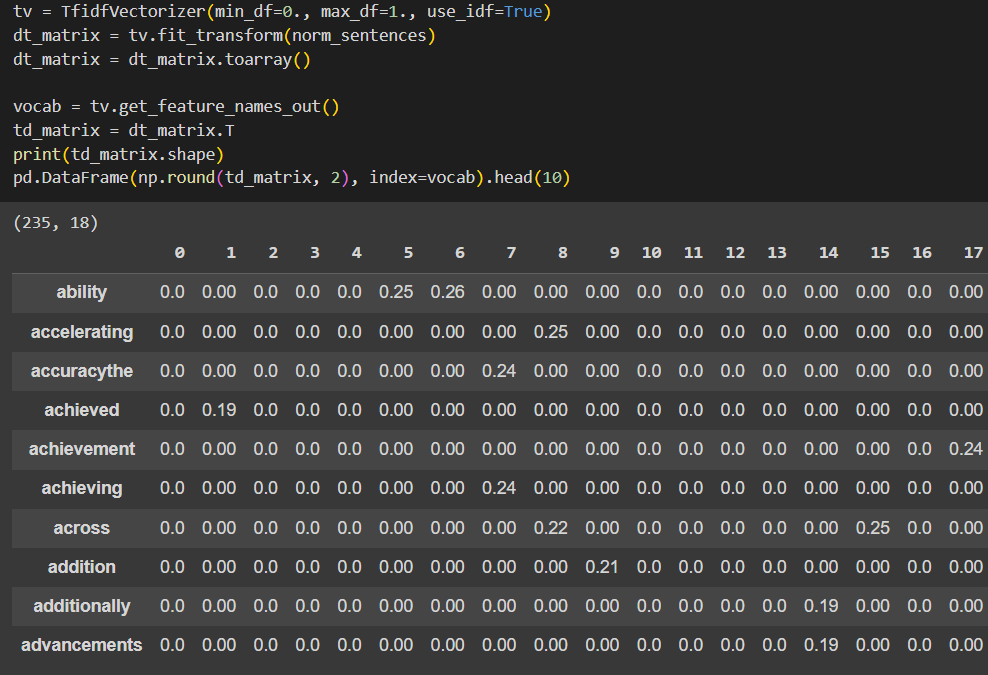
## replacing them with placeholders or converting them into text representations, Addressing special characters, emojis, or symbols that might not be relevant to the analysis.

## 

## Activity 2.2: Vectorize Text Data

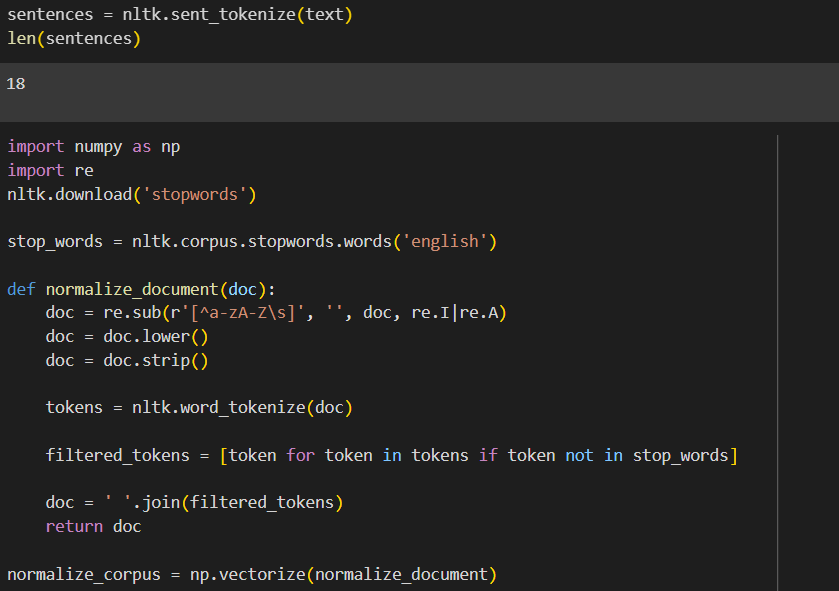
## 

TF-IDF vectorization converts text data into numerical format. It assigns weights to words based on their frequency in a document and across all documents. This process captures the significance of words within individual documents while considering their prevalence in the entire dataset. The resulting numerical representation facilitates the application of machine learning algorithms to analyze and extract insights from textual data



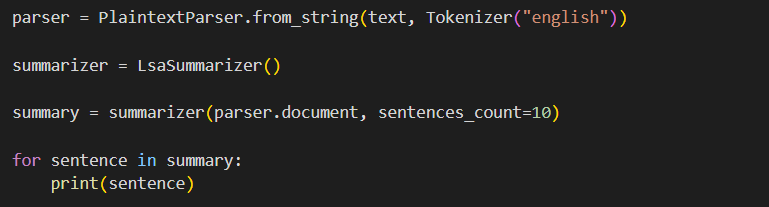
**Activity 2.3: Text normalization**

Text normalization is a crucial preprocessing step in Natural Language Processing (NLP) that involves converting text into a standard, consistent format. The goal of text normalization is to reduce the variability in the text data while preserving the meaning, making it easier to analyze and process by algorithms.

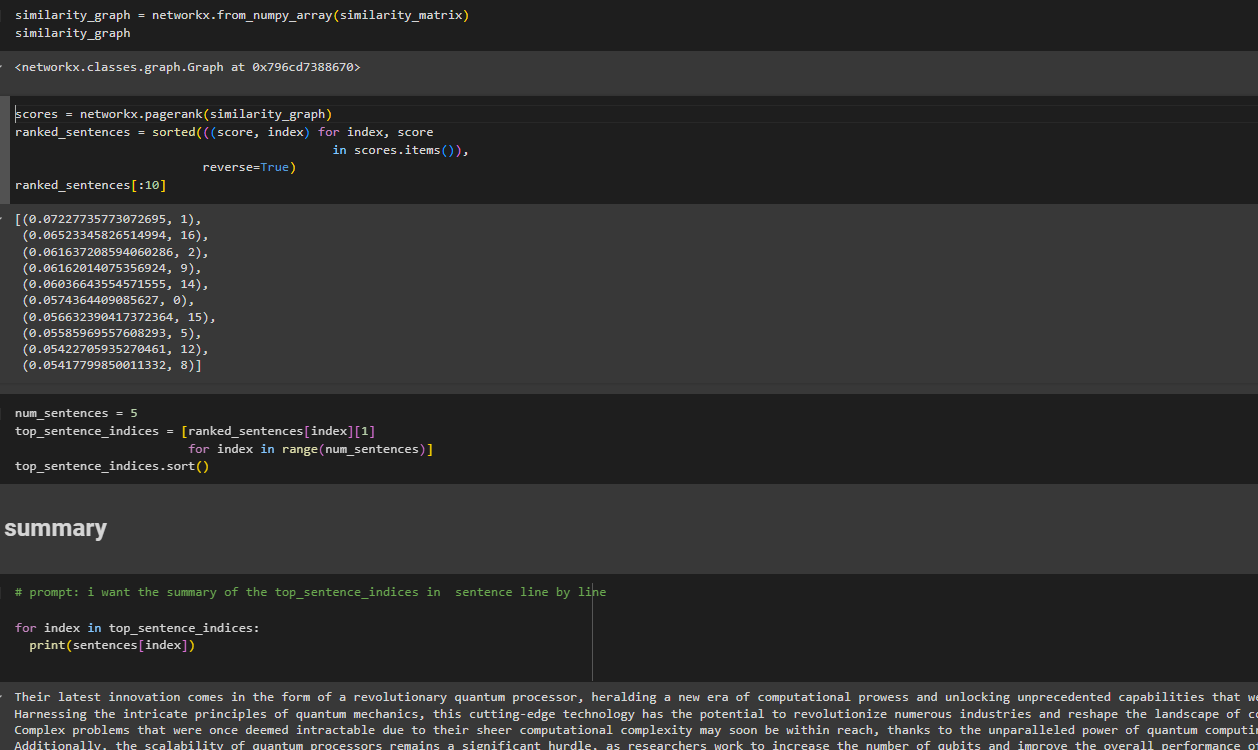


## MILESTONE 3: TEXT SUMMARIZER MODEL

Latent Semantic Analysis (LSA) Summarizer is a text summarization technique that leverages the principles of Latent Semantic Analysis, a natural language processing method that uncovers the hidden (latent) relationships between words in a text. LSA is based on the concept of reducing the dimensionality of the text data, making it possible to capture the most important topics or concepts.



**Summary:**

****

# Milestone 4: Model Deployment

**Activity 1: Integrate with Web Framework**

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

* + Building HTML Pages
  + Building server-side script
  + Run the web application

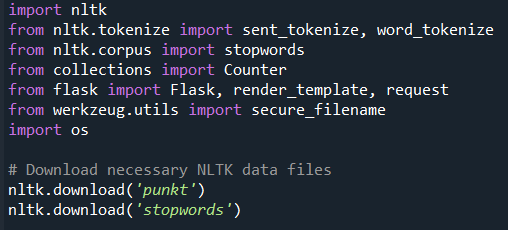
## Activity 2.1: Building Html Pages:

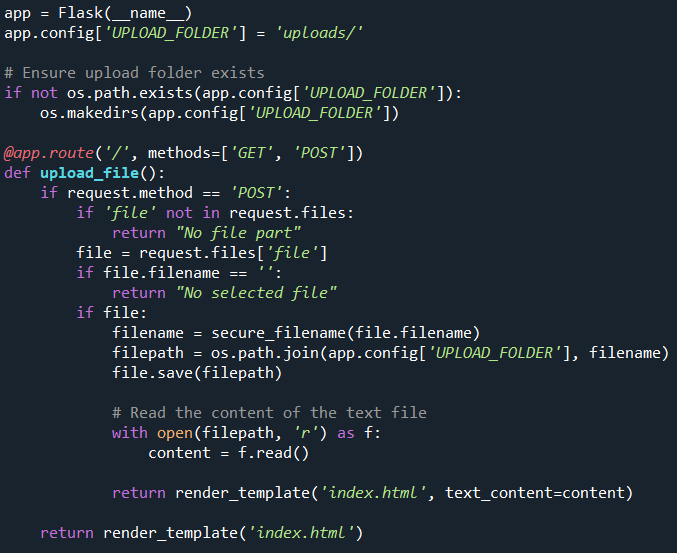
For this project create two HTML files namely

* index.html
* portfolio-details.html

## Activity 2.2: Build Python code:

Import the libraries



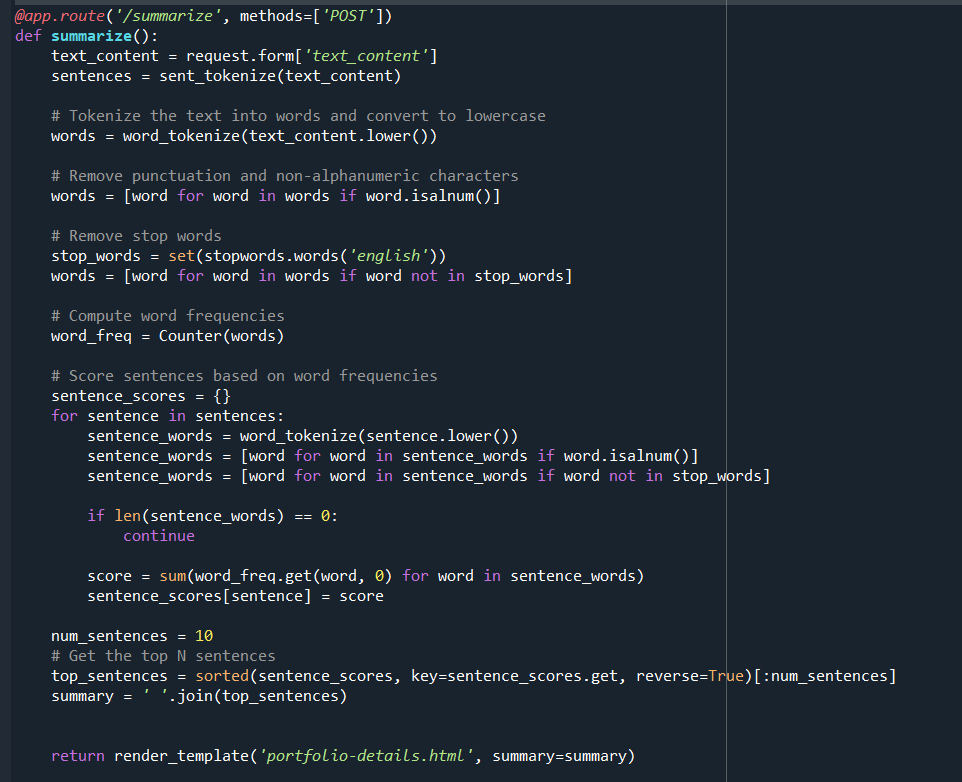


Render HTML page:

Here we will be using a declared constructor to route to the HTML page that we have created earlier.In the above example, ‘/’ URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.

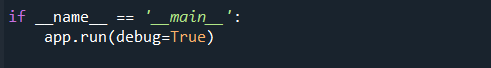
Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:



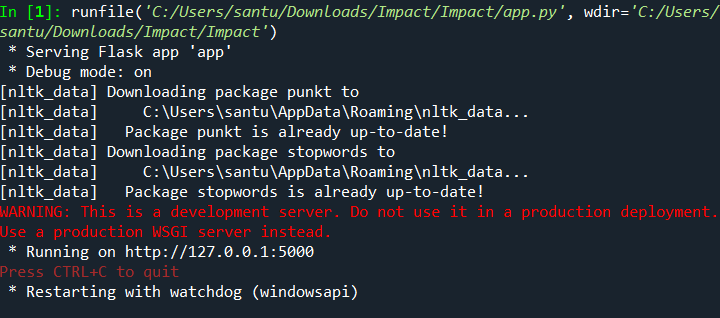
Here we are routing our app to predict the () function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model. predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the input.html page earlier.

Main Function:

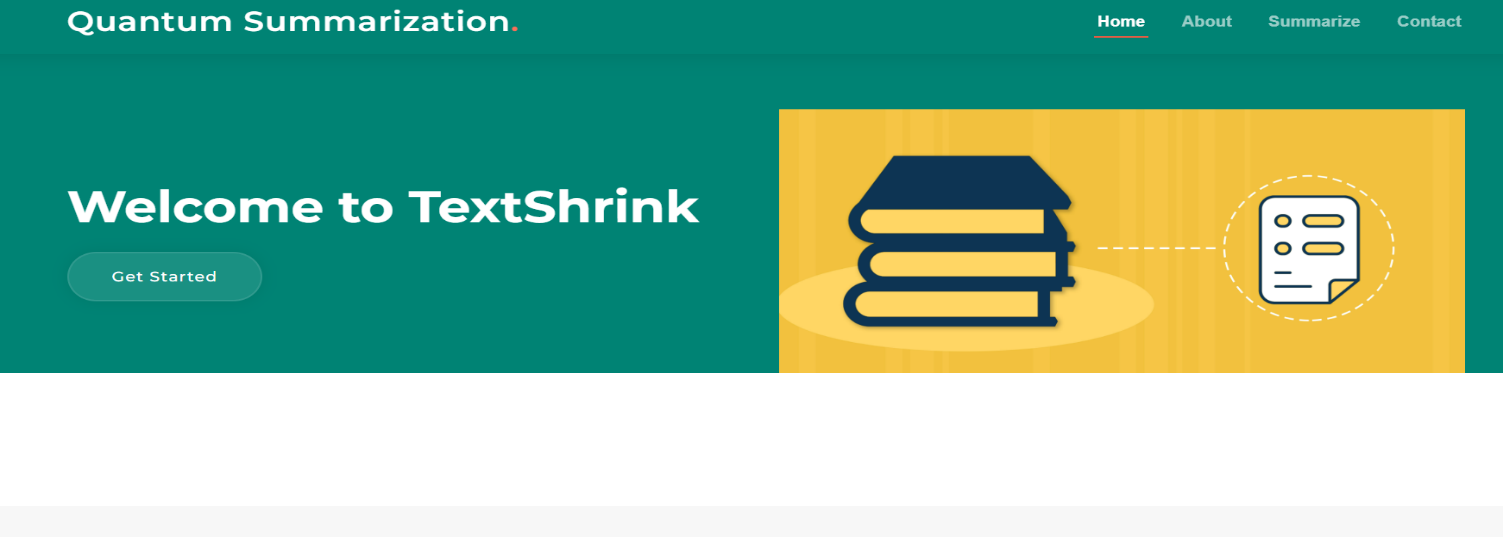


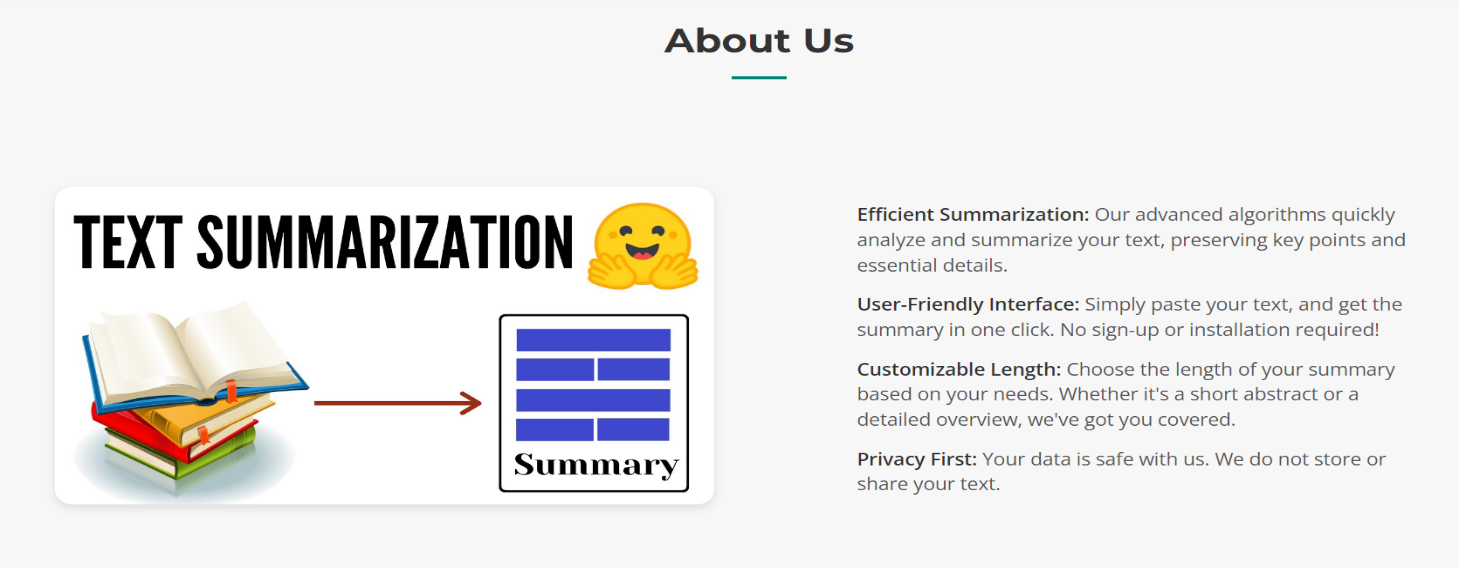
## Activity 2.3: Run the web application

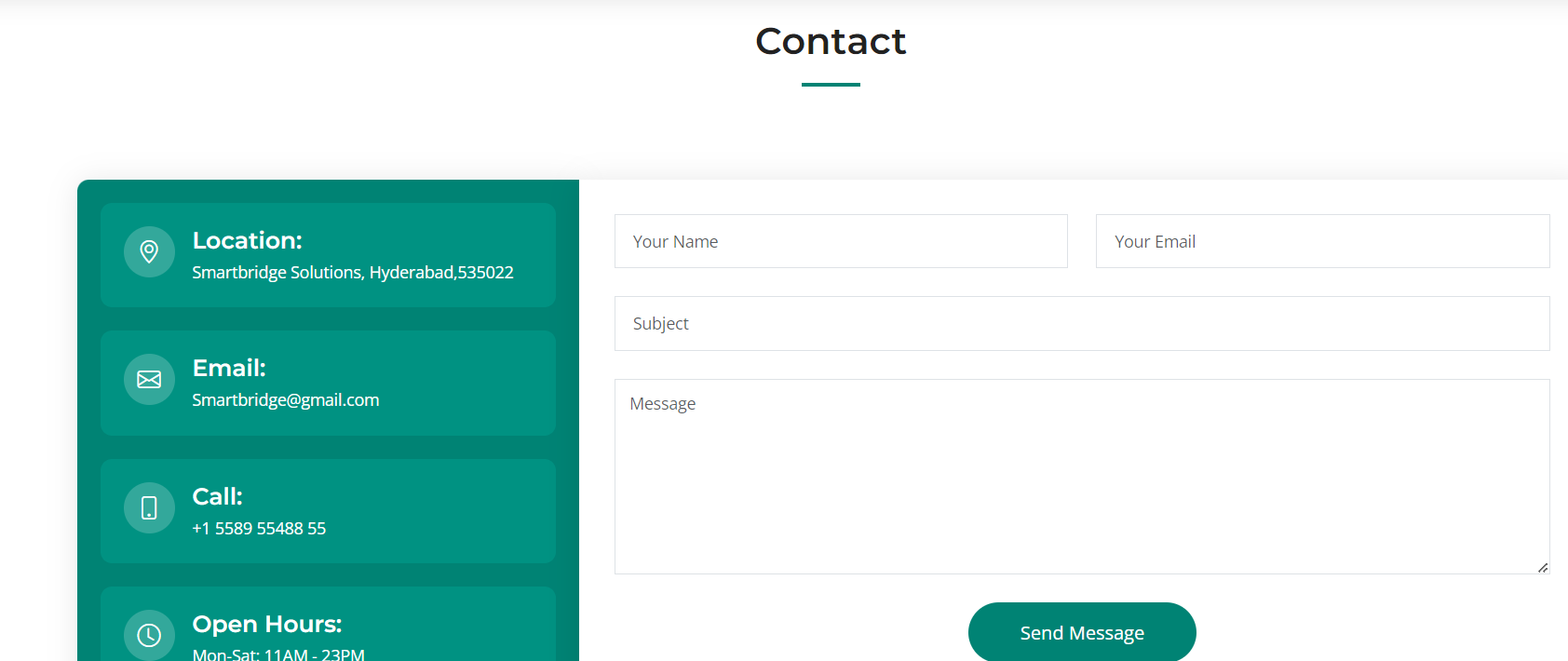
* Open the anaconda prompt from the start menu
* Navigate to the folder where your Python script is.
* Now type “app.py” command
* Navigate to the localhost where you can view your web page.
* Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



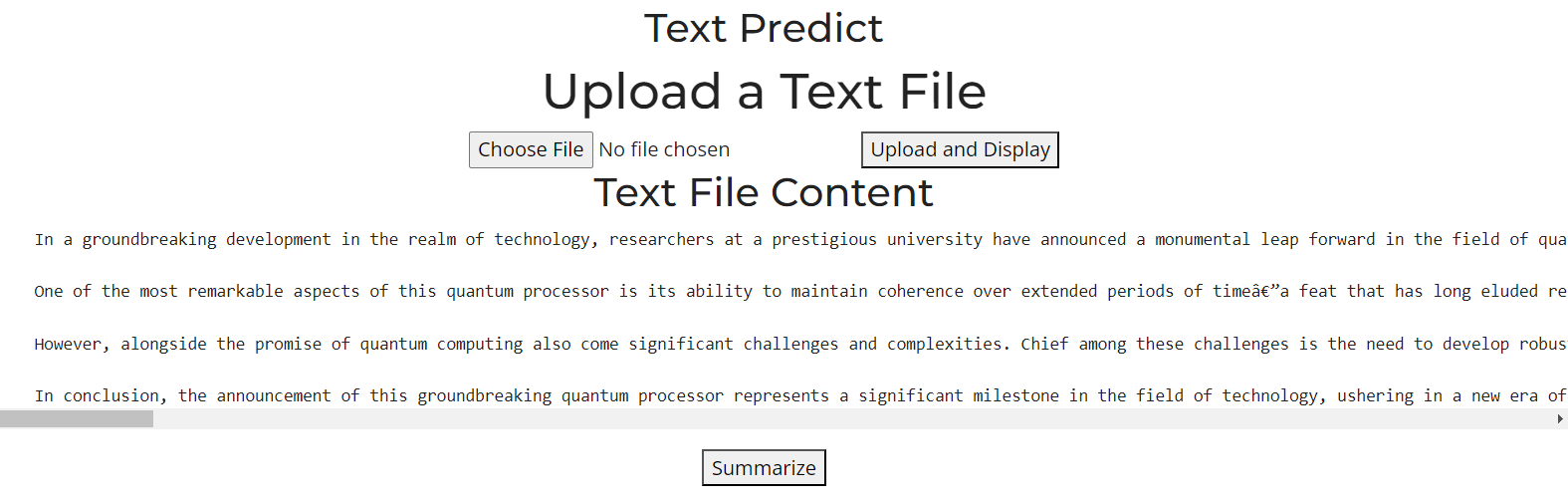
Now, Go to the web browser and write the localhost URL (http://127.0.0.1:5000) to get the below result







Give input to predict (AI Class):



Output page :

