KEYWORD EXTRACTION USING PYTHON

-Pujita Sunnapu

-20210701019

INTRODUCTION

The "Keyword Extraction with Python" project focuses on automating the extraction of meaningful keywords from an academic papers dataset. Leveraging Python and NLP libraries, the project preprocesses textual data, applies the TF-IDF algorithm for word weighting, and develops a user-friendly Flask web application. Users can upload documents for keyword extraction or search within the existing dataset. The project's robust system enhances information retrieval efficiency, contributing to Natural Language Processing advancements. Future work includes refining preprocessing, exploring advanced NLP techniques, and improving scalability asnd user interfaces. Overall, the project serves as a valuable tool for researchers and academics dealing with substantial textual data.

STEPS OF PROJECT

Step 1: Data Preprocessing

Step 2: TF-IDF Transformation

Step 3: Keyword Extraction Function

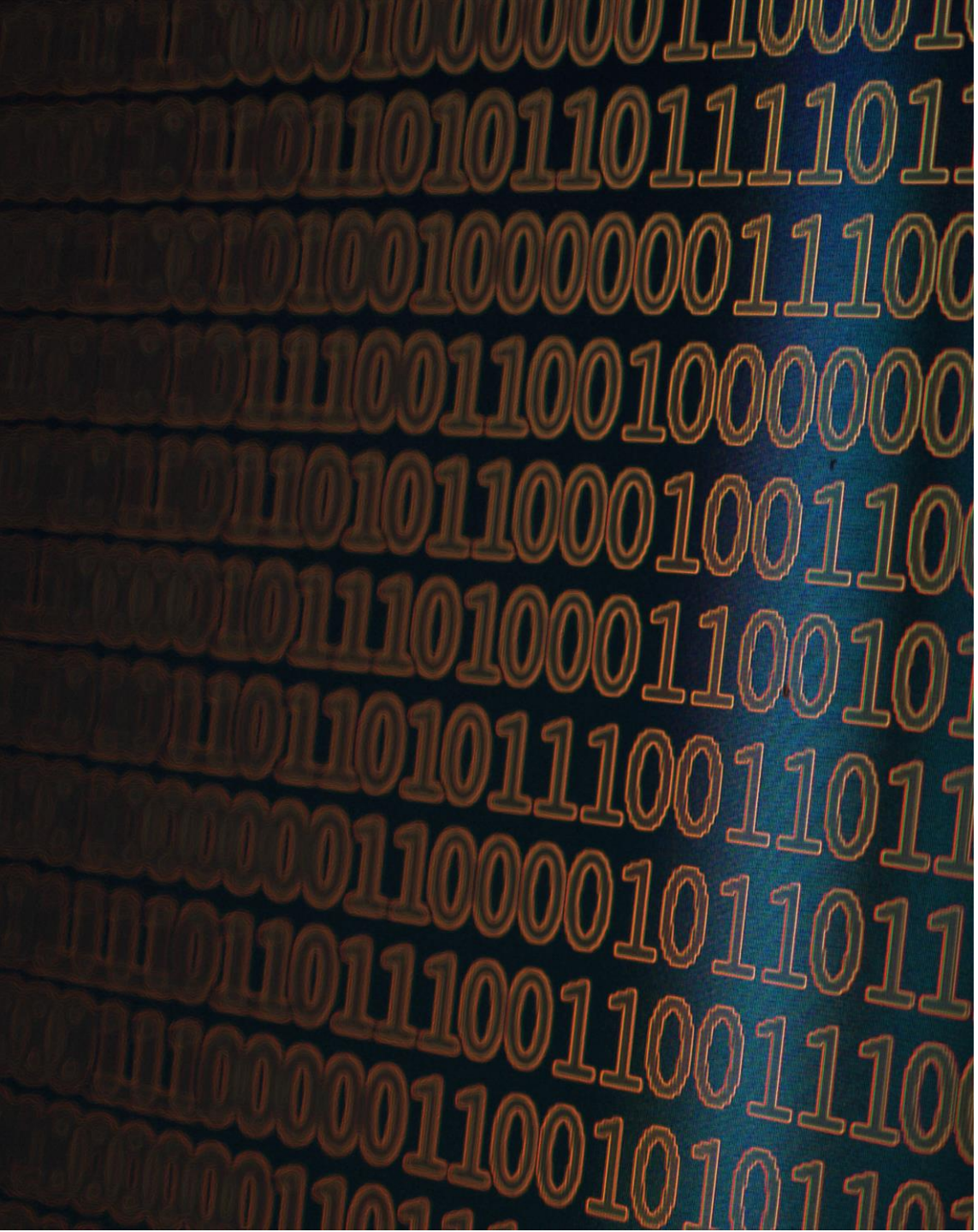
Step 4: Example Keyword Extraction

Step 5: Model Persistence

Step 6: Flask Application (app.py)

Step 7: HTML Templates (index.html, keywords.html, keywordslist.html)

Step 8: Running the Application



1. DATA PREPROCESSING

Objective: Ensure the data is clean and ready for analysis.

1. Check for Missing Values

Utilize `df.isnull().sum()` to identify any missing values.

Emphasize the importance of handling missing data for robust analysis.

2. Preprocess the "paper_text" Column

Convert to Lowercase: Standardize text by converting all characters to lowercase.

Remove HTML Tags: Utilize regular expressions to eliminate HTML tags from the text.

Remove Special Characters and Digits: Cleanse the text by removing unwanted characters and numerical digits.

Tokenization: Break down the text into individual tokens or words.

Remove Stopwords: Eliminate common and less informative words using a predefined set.

Remove Short Words: Exclude words with a length of less than three characters.

Lemmatization: Reduce words to their base or root form for better analysis and consistency.

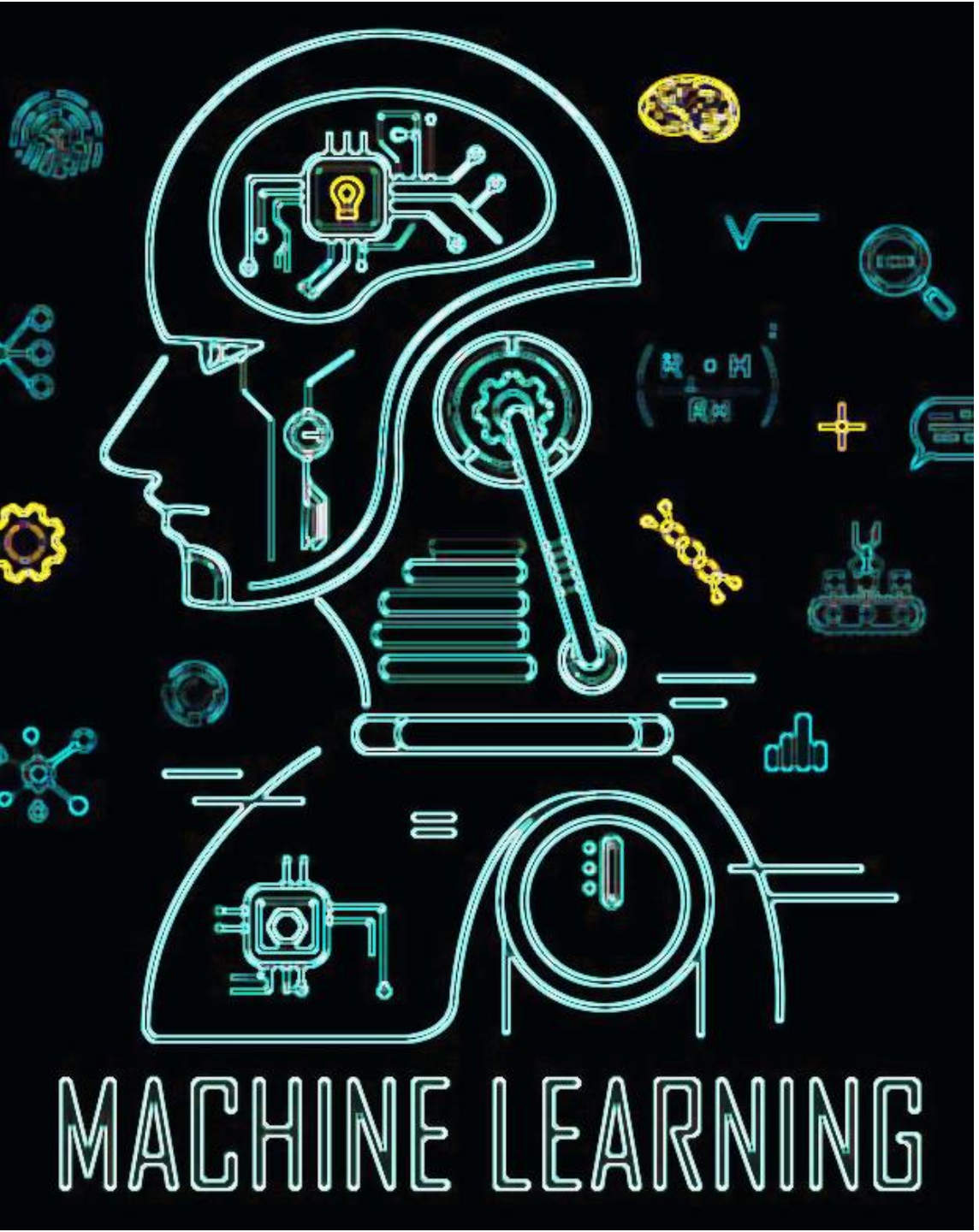
TF-IDF VECTORIZATION

TF-IDF is a numerical statistic tool that evaluates the importance of a word in a document relative to its frequency across a collection of documents, considering both the term frequency (TF) and inverse document frequency (IDF).

TF-IDF vectorization enhances our ability to identify keywords by capturing the distinctive importance of words in each document.

This process serves as a crucial step in our project's goal of automating keyword extraction and uncovering significant insights from academic papers.

TF-IDF weighting helps identify keywords by considering both the frequency of a word in a document and its rarity across the entire dataset. This method highlights words that are important in specific documents but not common throughout the dataset.



TF-IDF VECTORIZATION

Term Frequency (TF): Measures how often a word appears in a document. Calculated as the ratio of the number of times a word appears in a document to the total number of words in the document.

Inverse Document Frequency (IDF): Evaluates the rarity of a word across the entire document collection. Calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the word.

TF-IDF Weighting Scheme: Combines TF and IDF to assign weights to words, emphasizing those that are frequent in a document but rare across the entire collection. Words with higher TF-IDF scores are considered more significant in the context of a specific document.

CountVectorizer Implementation: In our project, we utilize Scikit-learn's `CountVectorizer` to generate word count vectors. The vectorizer creates a vocabulary of words and transforms the processed text into a matrix of word counts.

TF-IDF Transformation: The word count matrix is then transformed using Scikit-learn's `TfidfTransformer`.This transformation applies the TF-IDF weighting scheme to calculate the final TF-IDF weighted vectors for each document.

In [14]:
```
from sklearn.feature_extraction.text import CountVectorizer

# Reduce max_features and adjust n-gram range
cv = CountVectorizer(max_features=6000, ngram_range=(1, 2))

# Create a vocabulary and word count vectors
word_count_vectors = cv.fit_transform(docs)
```

In [16]:
```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count_vectors)
```

Out[16]:
```
TfidfTransformer()
```

Keyword Extraction

KEYWORD EXTRACTION

Objective: Uncover the most significant terms in a document based on TF-IDF scores.

Keyword extraction is performed through a series of steps utilizing the TF-IDF vectorization scheme. First, after preprocessing the text data, a TF-IDF matrix is generated. This matrix encapsulates the importance of each term within the document and the broader context of the entire corpus.

1. TF-IDF Vectorization: Utilizing the TF-IDF matrix, each term in the document is assigned a numerical score, reflecting its relevance to the specific document and the entire dataset.

2. Sorting TF-IDF Scores: The TF-IDF scores are sorted in descending order. This step ensures that the most important terms are identified first.

3. Top Keywords Extraction: The top N keywords are extracted based on their TF-IDF scores. This step is crucial for obtaining a concise and relevant set of keywords associated with the document.

4. Displaying Results: The extracted keywords, along with their respective TF-IDF scores, are presented as the output. This provides a clear understanding of the terms that contribute significantly to the document's content.

In [17]:

```python
def sort_coo(coo_matrix):
    tuples = zip(coo_matrix.col, coo_matrix.data)
    return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True)

def extract_topn_from_vector(feature_names, sorted_items, topn=10):
    #taking top items from vector
    sorted_items = sorted_items[:topn]

    score_vals = []
    feature_vals = []
    for idx, score in sorted_items:
        fname = feature_names[idx]
        score_vals.append(round(score,3))
        feature_vals.append(feature_names[idx])

    #create a tuples of features,score
    results = {}
    for idx in range(len(feature_vals)):
        results[feature_vals[idx]] = score_vals[idx]  # Fix: Changed '==' to '='
    return results


# get feature names
feature_names=cv.get_feature_names_out()

def get_keywords(idx, docs):

    #generate tf-idf for the given document
    tf_idf_vector=tfidf_transformer.transform(cv.transform([docs[idx]]))

    #sort the tf-idf vectors by descending order of scores
    sorted_items=sort_coo(tf_idf_vector.tocoo())

    #extract only the top n; n here is 10
    keywords=extract_topn_from_vector(feature_names,sorted_items,10)

    return keywords


def print_results(idx,keywords, df):
    # now print the results
    print("\n=====Title=====")
    print(df['title'][idx])
    print("\n=====Abstract=====")
    print(df['abstract'][idx])
    print("\n===Keywords===")
    for k in keywords:
        print(k,keywords[k])
idx=941
keywords=get_keywords(idx, docs)
print_results(idx,keywords, df)
```

MODEL PERSISTENCE

Objective: Ensure the longevity and reusability of the trained model.

1. Importance of Model Persistence

- After training a model, it is crucial to save its state for future use without retraining.
- Model persistence ensures that the TF-IDF vectorizer and transformer can be used in other environments or applications.

2. Pickle Necessary Files

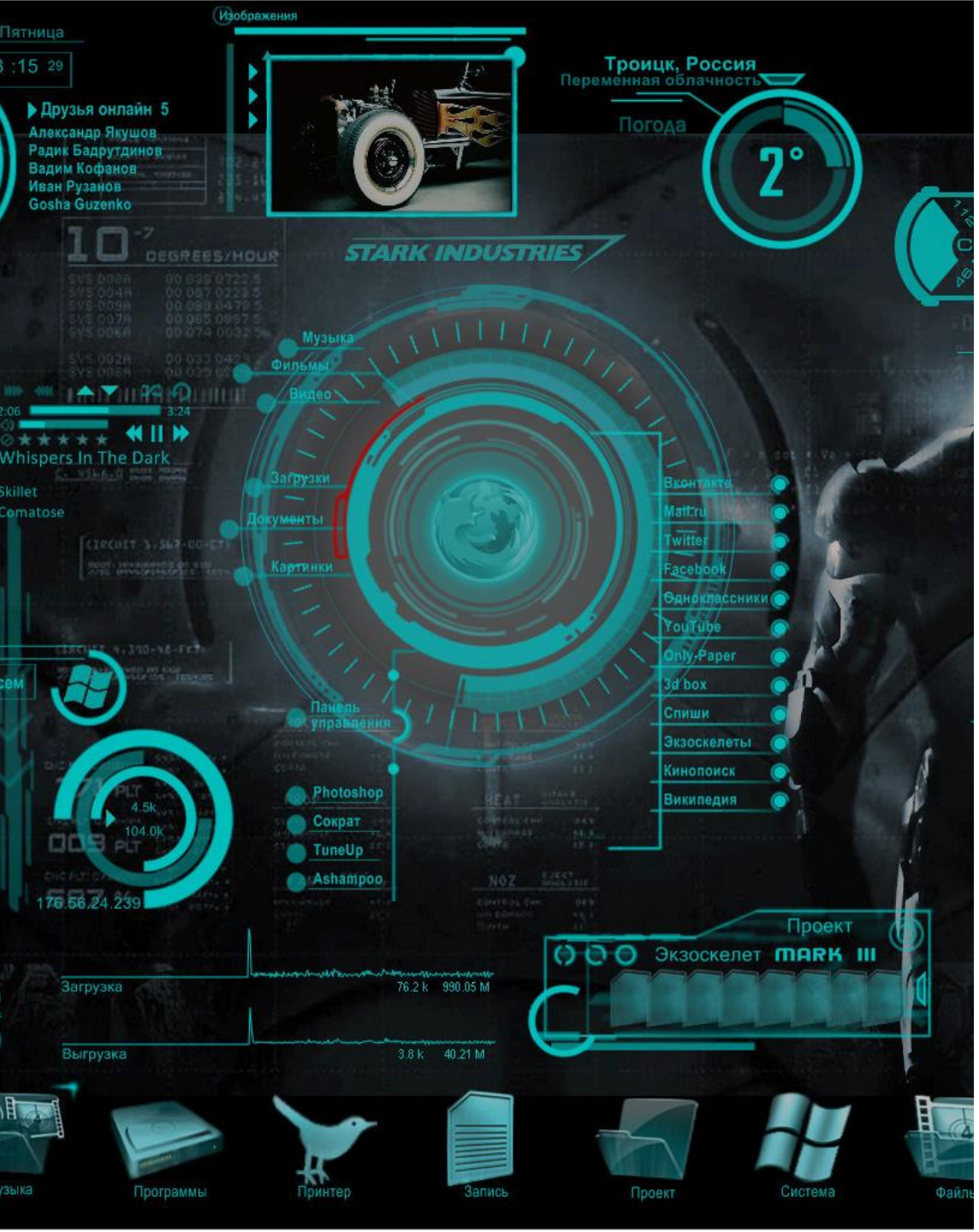
`pickle` library is employed to serialize and save Python objects.

Three key components are pickled:

TF-IDF Transformer: `tfidf_transformer`

Count Vectorizer: `cv`

Feature Names: `feature_names`



USER INTERFACE

Our user interface, developed using Flask, offers a seamless experience for keyword extraction and exploration. The interface consists of two main functionalities:

1. Document Upload: Users can upload their documents for keyword extraction. The system processes the document, extracts keywords, and presents them on the interface.
2. Keyword Search: Users can search for keywords within the existing dataset. The system responds with matching keywords and their relevance.

FLASK APPLICATION

Objective: Introduce the Flask web application for real-time keyword extraction.

1. Flask Overview

- Flask is a micro web framework for Python.
- Ideal for building web applications and APIs.
- Flask provides a robust foundation for deploying machine learning models and interacting with them through a web interface.

2. Components of the Flask Application

- Loading Pickled Files: Utilize `pickle` to load the TF-IDF transformer, Count Vectorizer, and Feature Names.
- Cleaning Data: Apply text preprocessing steps to incoming documents.
- Keyword Extraction Routes: Define routes for document upload, keyword extraction, and keyword search.

4. Key Functions in Flask Application

- `preprocess_text`: Cleans and preprocesses incoming text data.
- `extract_keywords`: Extracts keywords from uploaded documents.
- `search_keywords`: Searches for keywords based on user input.

5. Integration with HTML Templates

- The Flask application integrates with HTML templates for a user-friendly interface.

app.py

```python
import pickle
from flask import Flask, render_template, request
import re
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords

app = Flask(__name__)

# Load pickled files & data
with open('count_vectorizer.pkl', 'rb') as f:
    cv = pickle.load(f)

with open('tfidf_transformer.pkl', 'rb') as f:
    tfidf_transformer = pickle.load(f)

with open('feature_names.pkl', 'rb') as f:
    feature_names = pickle.load(f)

# Cleaning data:
stop_words = set(stopwords.words('english'))
new_stop_words = ["fig","figure","image","sample","using",
                  "show", "result", "large",
                  "also", "one", "two", "three",
                  "four", "five", "seven","eight","nine"]
stop_words = list(stop_words.union(new_stop_words))
```

HTML TEMPLATES

Objective: Showcase the user interface through HTML templates.

1. Role of HTML Templates

- HTML templates provide the structure and layout for the user interface.
- They seamlessly integrate with the Flask application for a dynamic and interactive experience.
- HTML templates play a crucial role in shaping the user experience, allowing users to interact with the Flask application seamlessly.

2. Design Philosophy

- Responsive Design: Templates are designed to be accessible and usable across various devices.
- Clean and Intuitive: User-friendly interfaces to enhance the user experience.

3. HTML Templates in the Project

Three primary templates:

1. `index.html`: Main landing page for document upload and keyword search.
2. `keywords.html`: Results page displaying extracted keywords.
3. `keywordslist.html`: Results list page for keyword search.

4. CSS Styling

- Consistent styling for a cohesive and professional appearance.
- Bootstrap or other CSS frameworks may be employed for quick and effective styling.

5. Dynamic Elements: Elements are dynamically updated based on user interactions, creating a responsive and real-time experience.


THANK YOU