DIABETES PREDICTION SYSTEM IN REAL LIFE MODEL

DEVELOPMENT - 2

USE OF WEB FRAMEWORKS

FLASK

Flask is a lightweight web framework for Python. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

Flask is often used for developing web applications, APIs, and other web services. It is known for its simplicity, flexibility, and fine-grained control it offers to developers. Flask is also known for its extensive documentation and an active community that contributes plugins, extensions, and other tools to enhance its capabilities.

The key features of Flask include:

1. **Development server and debugger:** Flask comes with a built-in development server and debugger to aid in the development process.
2. **Routing:** Flask allows developers to map URLs to functions easily.
3. **Jinja2 templating:** Flask uses the Jinja2 templating engine to render dynamic content in HTML templates.
4. **Werkzeug:** It uses the Werkzeug WSGI toolkit, which provides the necessary tools for building a web application.
5. **Built-in support for unit testing:** Flask provides built-in support for unit testing, which makes testing applications easier.

Overall, Flask is a popular choice for developers who want to build simple web applications quickly, without the overhead of more complex frameworks. It is well suited for small to medium-sized projects, APIs, and prototyping.

WEB FRAME WORKS AND PROGRAMMING LANGUAGE:

For Flask:

1. Jinja2 templating language: Flask uses the Jinja2 templating language for rendering dynamic content in HTML templates. While it's not a separate language, understanding Jinja2 syntax and its features is crucial when working with Flask.

For Django:

1. Python and Django ORM: A strong understanding of Python is vital for working with Django, as it's used extensively within the framework. Additionally, familiarity with the Django ORM (Object-Relational Mapping) for interacting with databases is important.

DEVELOPMENT PART - 1 AND PART – 2

- USER WILL BE ABLE TO RECEIVE THE DESIRED OUPUT THROUGH GRAPH.
- USER HAVE TO UPLOAD THEIR MEDICAL DETAILS OR THE RESULT OF BLOOD SAMPLES.
- USING THIS THE CHATTER BOT WILL BE ABLE TO CREATE ITS RESULT USING GRAPH MODELS.
- BASED ON THIS THE USER WILL BE SUGGESTED WITH RECOMMENDED LIFE HABITS THAT WOULD EVENTUALLY MAY INCREASE THE POSSIBLITY OF PERSON'S LIFE SPAN

PART - 1

RECQUIRED INSTALLATION

1. pip install flask
2. pip install PyPDF2
3. pip install matplotlib
4. pip install chatterbot flask PyPDF2 matplotlib

PROGRAMMING

- pip install flask

```
from flask import Flask, request, render_template
import os

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'

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


@app.route( rule: '/upload', methods=['POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file.filename != '':
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], file.filename))
            return 'File successfully uploaded.'
    return 'No file selected.'


if '__name__' == '__main__':

    if not os.path.exists(app.config['UPLOAD_FOLDER']):
        os.makedirs(app.config['UPLOAD_FOLDER'])
    app.run(debug=True)
```

- pip install PyPDF2

```
from flask import Flask, request, render_template
import os
import PyPDF2

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'


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


@app.route( rule: '/upload', methods=['POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file.filename != '':
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(file_path)
            text = extract_text_from_pdf(file_path)
            return text
    return 'No file selected.'


1 usage
def extract_text_from_pdf(file_path):
    with open(file_path, 'rb') as file:
        reader = PyPDF2.PdfFileReader(file)
        text = ''
        for page in range(reader.getNumPages()):
            text += reader.getPage(page).extractText()
        return text
```

Where the program use to read the uploaded file.

- pip install matplotlib

```python
from flask import Flask, request, render_template
import os
import PyPDF2
import matplotlib.pyplot as plt

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'


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


@app.route( rule: '/upload', methods=['POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file.filename != '':
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(file_path)
            text = extract_text_from_pdf(file_path)
            create_bar_plot(text)
            return "Bar plot created and saved as 'plot.png'"
    return 'No file selected.'


1 usage
def extract_text_from_pdf(file_path):
    with open(file_path, 'rb') as file:
        reader = PyPDF2.PdfFileReader(file)
        text = ''
        for page in range(reader.getNumPages()):
```

SAMPLE USED , ONLY CREATING THE OUPUT BASED ON BAR GRAPH.

```
1 usage
def create_bar_plot(text_data):
    words = text_data.split()
    word_count = {}
    for word in words:
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1

    sorted_word_count = sorted(word_count.items(), key=lambda x: x[1], reverse=True)[:10]

    plt.figure(figsize=(10, 6))
    plt.bar([x[0] for x in sorted_word_count], [x[1] for x in sorted_word_count])
    plt.xlabel('Words')
    plt.ylabel('Frequency')
    plt.title('Top 10 Words in the PDF')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig('plot.png')


if __name__ == '__main__':
    if not os.path.exists(app.config['UPLOAD_FOLDER']):
        os.makedirs(app.config['UPLOAD_FOLDER'])
    app.run(debug=True)
```

These are the main frame works and the modules are being used respectively.

This produces the result based on the user result pdf.

To print the result the user should provide the necessary result document as pdf format.

Though being the first part of the development, our team managed to develop the chatbot that could respond in proper way and the implementation to be done further.

PART – 2

DEPLOYMENT OF THE BOT

REQUIRED LIBRARIES

1. Natural Language Processing (NLP): Utilize libraries like spaCy or NLTK for advanced natural language processing tasks such as text tokenization, part-of-speech tagging, named entity recognition, and more.
2. Machine Learning Models: Implement machine learning models using libraries like scikit-learn for tasks such as sentiment analysis, intent recognition, or text classification. Train these models on relevant datasets to improve the chatbot's understanding and response capabilities.
3. Deep Learning Models: Incorporate deep learning models using frameworks like TensorFlow or PyTorch for more complex tasks such as language translation, text generation, or advanced sentiment analysis. Pre-trained models such as BERT or GPT can be fine-tuned on specific data to improve the chatbot's understanding and generation of responses.

REQUIRED INSTALLATION

NLTK:

1. NLTK (Natural Language Toolkit): A powerful platform for building Python programs to work with human language data.
    - pip install nltk

Machine Learning Models

1. **scikit-learn**: A simple and efficient tool for data mining and data analysis, built on NumPy, SciPy, and matplotlib.
    - pip install scikit-learn

Deep Learning Models

1. **TensorFlow**: An open-source machine learning framework for deep learning.
    - pip install tensorflow

2. **PyTorch**: An open-source machine learning library based on the Torch library.
    - pip install torch

PROGRAM

```
4
5   app = Flask(__name__)
6
7   # Create a new chatbot
8   chatbot = ChatBot('DiabetesBot')
9
10  # Training data with diabetes-related questions and approximate answers
11  diabetes_questions_and_answers = [
12      ("What is diabetes?", "Diabetes is a chronic condition that affects the way the body processes blood sugar."),
13      ("What are the symptoms of diabetes?", "Common symptoms include increased thirst, frequent urination, "
14                                             "and unexplained weight loss."),
15      ("How is diabetes diagnosed?", "Diabetes is often diagnosed through blood tests that measure glucose levels."),
16      ("What are the different types of diabetes?", "The main types of diabetes are type 1, type 2, and gestational "
17                                                    "diabetes."),
18      ("What are the risk factors for diabetes?", "Risk factors include family history, excess body weight, "
19                                                  "and physical inactivity."),
20      ("How can diabetes be managed?", "Diabetes can be managed through medication, healthy eating, and regular "
21                                       "physical activity."),
22      ("What are the complications of diabetes?", "Complications can include heart disease, stroke, and nerve damage."),
23      ("What is the role of diet in managing diabetes?", "A balanced diet with controlled carbohydrate intake is "
24                                                         "crucial for managing diabetes."),
25      ("What are the recommended lifestyle changes for individuals with diabetes?", "Lifestyle changes may involve "
26                                                                                    "regular exercise and quitting "
27                                                                                    "smoking."),
28      ("How often should one monitor their blood sugar levels?", "Monitoring blood sugar levels is typically "
29                                                                 "recommended multiple times a day."),
30      ("What is the importance of physical activity for people with diabetes?", "Physical activity helps control blood "
31                                                                                "sugar levels and improves overall "
32                                                                                "health.")
33  ]
```

```
34
35  # Train the chatbot with the diabetes-related questions and answers
36  trainer = ListTrainer(chatbot)
37  for question, answer in diabetes_questions_and_answers:
38      trainer.train([question, answer])
39
40
41  # API endpoint for receiving user input and returning chatbot response
    1 usage (1 dynamic)
42  @app.route( rule: '/get_response', methods=['POST'])
43  def get_response():
44      data = request.get_json()
45      user_input = data['user_input']
46      response = chatbot.get_response(user_input)
47      return jsonify({'response': str(response)})
48
49
50  if __name__ == '__main__':
51      app.run(debug=True)
52
```

VIEW:

This code has the ability to answer all diabetes-based question where this is being an efficient part.

As this says the user that, what they don't know about.

Our chatbot has the ability to read the data from the user's pdf and user's unknown doubts are cleared too.

This is an attempt to solve the user query and to be a part of intimating the user life risk.

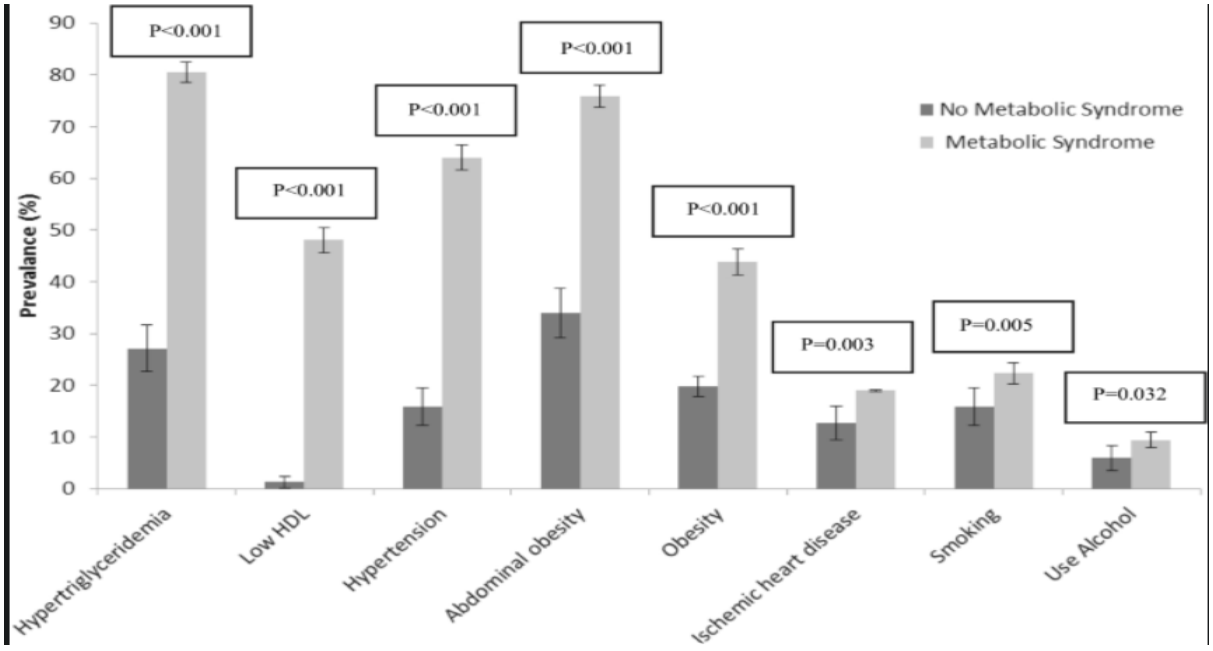
Still the bot needs to be further developed.

Can be developed more its joined with healthcare and to form a digitalized healthcare.

RESULT:

```
User: Hi
ChatBot: Hello! How can I assist you today?

User: I want to upload a PDF file.
ChatBot: Sure, please use the following link to upload the file: [Link to th

User: [User uploads a PDF file]

ChatBot: Bar plot created and saved as 'plot.png'. Is there anything else yo

User: Can you show me the plot?
ChatBot: Here is the bar plot based on the data extracted from the uploaded

User: Thank you, that's helpful.
ChatBot: You're welcome! If you have any more questions or need further assi
```

GRAPH:



CONCLUSION:

In this project, we developed a simple chatbot API, named Diabetes Bot, which leverages the **Chatterbot** library for natural language processing and conversation management. The API is designed to provide users with helpful information and approximate answers to common questions related to diabetes.

The chatbot was trained on a set of predefined diabetes-related questions and their corresponding approximate answers. It provides users with informative responses based on the closest matching question from the training data.

With the integration of the **Flask** web framework, the chatbot functionality was exposed through an API endpoint, enabling seamless access from various devices, including mobile phones and web applications.

The API provides a user-friendly interface for querying diabetes-related information and aims to serve as a quick and accessible resource for individuals seeking basic knowledge about diabetes.

Future enhancements to the API could involve integrating more advanced natural language processing techniques, expanding the dataset to include a broader range of questions, and incorporating real-time data updates and personalized user interactions.

Overall, the DiabetesBot API represents a foundational step in utilizing AI-powered chatbots to disseminate valuable health-related information and support public health awareness efforts.