Phase 5: Project Documentation & Submission

In this phase you will document your project and prepare it for submission. Please refer Below the requirements technology wise:

Submission:

Phase 1: Problem Definition and Design Thinking

Problem Definition: The problem is to build an AI-powered Diabetes prediction system that uses machine learning Algorithms to analyze medical data and predict the likelihood of An individual developing diabetes. The system aims to provide Early risk assessment and personalized preventive measures, Allowing individuals to take proactive actions to manage their Health.

Design Thinking:

1. Functionality: Define the scope of the chatbot's abilities, Including answering common questions, providing Guidance, and directing users to appropriate resources.
2. User Interface: Determine where the chatbot will be Integrated (website, app) and design a user-friendly Interface for interactions.

3. Natural Language Processing (NLP): Implement NLP Techniques to understand and process user input in a Conversational manner.

4. Responses: Plan responses that the chatbot will offer, such As accurate answers, suggestions, and assistance.

5. Integration: Decide how the chatbot will be integrated with The website or app.

6. Testing and Improvement: Continuously test and refine the Chatbot's performance based on user interactions.

Steps Involved:

1. Design: Develop a chatbot for website/apps to provide Instant customer service.

2. Applicability: Enhance user experience by offering Immediate assistance.

3. Technology: NLP and AI algorithms for understanding and Responding to queries.

4. Coding: Python, NLP libraries, and possibly Tensorflow for

Advanced chatbots.

5. Architecture: An AI-Powered concierge guilding users Through interactions.

6. Transformation: Transforms customer service into an Always-available virtual assistant.

7. Real-World Analogy: A knowledgeable hotel concierge Offerings guidance at any time.

INSTRUCTIONS:

Step 1: Download and Install Git

1. Visit the official Git website: https://git-scm.com/
2. Download the appropriate version of Git for your operating System (Windows, macOS, or Linux).
3. Run the installer and follow the on-screen instructions to Complete the installation.
4. Open a terminal or command prompt and verify the Installation by typing git—version.

Step 2: Download and Install Visual Studio Code

1. Go to the Visual Studio Code Website: https://code.visualstudio.com/
2. Download the installer for your operating system (Windows, macOS, or Linux).
3. Run the installer and follow the installation prompts.
4. Launch Visual Studio Code.

Step 3: Create a GitHub Account

1. Open a web browser and go to https://github.com/
2. Click on the “Sign up” button.
3. Follow the registration process, providing your username,

   Email address, and password.

4. Complete the verification process if prompted.

Step 4: Create a GitHub Repository

1. Log in to your GitHub account.
2. Click on your profile icon in the upper right corner and select

   “Your repositories” from the dropdown menu.

3. On the “Repositories” page, click the green “New” button.
4. Fill in the required information for your new repository,

   Including the repository name, description, visibility, and other Settings.

5. Optionally, you can choose to initialize the repository with a README file or add a .gitignore file for your specific project.
6. Click the green “Create repository” button to create your

   GitHub repository.

Step 5: Create a Local Folder

1. Minimize any open windows on your computer to see your

Desktop.

2. Right-click on an empty area of your desktop.
3. Hover over “New” in the context menu.
4. Click on “Folder” to create a new folder.
5. Give your folder a meaningful name, like “MyProject.”

Step 6: Open the Folder in Visual Studio Code

1. Launch Visual Studio Code.
2. Click on “File” in the top-left corner.
3. Select “Open Folder” from the dropdown menu.
4. Browse to your desktop and select the folder you created in Step 5 (e.g., “MyProject”).
5. Click the “Open” button to open the folder in Visual Studio Code.

Step 7: Clone Your GitHub Repository

1. In Visual Studio Code, open the integrated terminal by Clicking on “View” in the top menu and selecting “Terminal” or Using the keyboard shortcut (Ctrl+ on Windows/Linux Or Cmd+ on macOS).
2. Use the git clone command to clone your GitHub repository By pasting the HTTPS URL of your repository. Replace repository_url with the actual URL. Git clone <repository_url>
3. Navigate to the newly created repository folder using The cd command:

Cd <repository_name>

Step 8: Check Git Status

1. To check the status of your local repository, enter the

Following command:

Git status

Step 9: Modify the README File

1. Open the README file in your repository folder using

Visual Studio Code.

2. Make the desired modifications to the README file.

Step 10: Check Git Status Again

1. Return to the terminal in Visual Studio Code.
2. Use the gitstatus command again to see the changes you

Made:

Git status

Step 11: Add Modifications to Staging Area

1. To stage your changes for a commit, use the gitadd command:

Git add README.md

Step 12: Commit Your Changes

1. Commit your staged changes with a descriptive message:

Git commit -m “Updated README file”

Step 13: Push Changes to GitHub

1. Push your committed changes to your GitHub repository:

Git push

Step 14: Create a New Branch

1. To create a new branch, use the gitbranch command followed By the desired branch name: Git branch branch_name

Step 15: Switch to the New Branch

1. To switch to the newly created branch, use The gitcheckout command:Git checkout branch_name

Step 16: Check Your Current Branch

1. To confirm the branch you're currently working on, use The gitbranch command:

Git branch

Innovation:

Chat Bot in Python with Chatter Bot Module

Nobody likes to be alone always, but sometimes loneliness could be a better medicine to hunch the thirst for a peaceful environment. Even during such lonely quarantines, we may ignore humans but not humanoids. Yes, if you have guessed this article for a chatbot, then you have cracked it right. We won't require 6000 lines of code to create a chatbot but just a six-letter word "Python" is enough. Let us

have a quick glance at Python's ChatterBot to create our bot. ChatterBot is a Python library built based on machine learning with an inbuilt conversational dialog flow and training engine. The bot created using this library will get trained automatically with the response it gets from the user.

Understanding the Chatbot:

A Chatbot is an Artificial Intelligence-based software developed to interact with humans in their natural languages. These chatbots are generally converse through auditory or textual methods, and they can effortlessly mimic human languages to communicate with human beings in a human-like way. A chatbot is considered one of the best applications of natural languages processing.

We can categorize the Chatbots into two primary variants: Rule-Based Chatbots and Self-Learning Chatbots.

1. Rule-based Chatbots:The Rule-based approach trains a chatbot to answer questions based on a list of pre-determined rules on which it was primarily trained. These set rules can either be pretty simple or quite complex, and we can use these rule-based chatbots to handle simple queries but not process more complicated requests or queries.
2. Self-learning Chatbots:Self-learning chatbots are chatbots that can learn on their own. These leverage advanced technologies such as Artificial Intelligence (AI) and Machine Learning (ML) to train themselves from behaviours and instances. Generally, these chatbots are

quite smarter than rule-based bots. We can classify the Self-learning chatbots furtherly into two categories – Retrieval-based Chatbots and Generative Chatbots.

1. Retrieval-based Chatbots:A retrieval-based chatbot works on pre-defined input patterns and sets responses. Once the question or pattern is inserted, the chatbot utilizes a heuristic approach to deliver the relevant response. The model based on retrieval is extensively utilized to design and develop goal-oriented chatbots using customized features such as the flow and tone of the bot in order to enhance the experience of the customer.
2. Generative Chatbots:Unlike retrieval-based chatbots, generative chatbots are not based on pre-defined responses – they leverage seq2seq neural networks. This is constructed on the concept of machine translation, where the source code is converted from one language to another language. In the seq2seq approach, the input is changed into an output.

The first chatbot named ELIZA was designed and developed by Joseph Weizenbaum in 1966 that could imitate the language of a psychotherapist in only 200 lines of code. But as the technology gets more advance, we have come a long way from scripted chatbots to chatbots in Python today.

Chatbot in present Generation:

Today, we have smart Chatbots powered by Artificial Intelligence that utilize natural language processing (NLP) in order to understand the commands from humans (text and voice) and learn from experience. Chatbots

have become a staple customer interaction utility for companies and brands that have an active online existence (website and social network platforms).

With the help of Python, Chatbots are considered a nifty utility as they facilitate rapid messaging between the brand and the customer. Let us think about Microsoft's Cortana, Amazon's Alexa, and Apple's Siri. Aren't these chatbots wonderful? It becomes quite interesting to learn how to create a chatbot using the Python programming language.

Fundamentally, the chatbot utilizing Python is designed and programmed to take in the data we provide and then analyze it using the complex algorithms for Artificial Intelligence. It then delivers us either a written response or a verbal one. Since these bots can learn from experiences and behavior, they can respond to a large variety of queries and commands.

Although chatbot in Python has already started to rule the tech scenario at present, chatbots had handled approximately 85% of the customer-brand interactions by 2020 as per the prediction of Gartner.

In light of the increasing popularity and adoption of chatbots in the industry, we can increase the market value by learning how to create a chatbot in Python – among the most extensively utilized programming languages globally.

Understanding the ChatterBot Library:

ChatterBot is a Python library that is developed to provide automated responses to user inputs. It makes utilization of a combination of Machine Learning algorithms in order to generate multiple types of responses. This feature enables developers to construct chatbots using Python that can communicate with humans and provide relevant and appropriate responses. Moreover, the ML algorithms support the bot to improve its performance with experience.

Another amazing feature of the ChatterBot library is its language independence. The library is developed in such a manner that makes it possible to train the bot in more than one programming language.

Why Chatbots are important for a Business or a Website

- Quick resolution for a complaint or a problem.
- Improve business branding thereby achieving great customer satisfaction.
- Answering questions and answers for customers.
- Making a reservation at hotel or at restaurant.
- Save human effort 24×7.
- Enhance business revenue by providing ideas and inspirations.
- Finding details about business such as hours of operation, phone number and address.
- Automate sales and lead generation process.
- Reduce customer agents waiting time answering phone calls.

Benefits of using Chatbots:

- 24×7 availability.
- Instant answers to queries.
- Support multi-language to enhance businesses.
- Simple and Easy to Use UI to engage more customers.
- Cost effective and user interactive.
- Avoid communication with call agents thereby reducing the time consuming tasks.
- Understand the Customer behavior
- Increase sales of business by offering promo codes or gifts.

Types of Chatbots:

Chatbots deliver instantly by understanding the user requests with pre-defined rules and AI based chatbots. There are two types of chatbots.

- Rule Based Chatbots: This type of chatbots answer the customer queries using the pre-defined rules. These bots answer common queries such as hours of operation of business, addresses, phone numbers and tracking status.
- Conversational AI Chatbots: This type of chatbots using Natural language Processing(NLP) to understand the context and intent of a user input before providing the response. These Bots train

themselves as per the user inputs and more they learn, more they become user interactive.

Installation:

Install chatterbot using Python Package Index(PyPi) with this command

Pip install chatterbot Below is the implementation.

Example program:

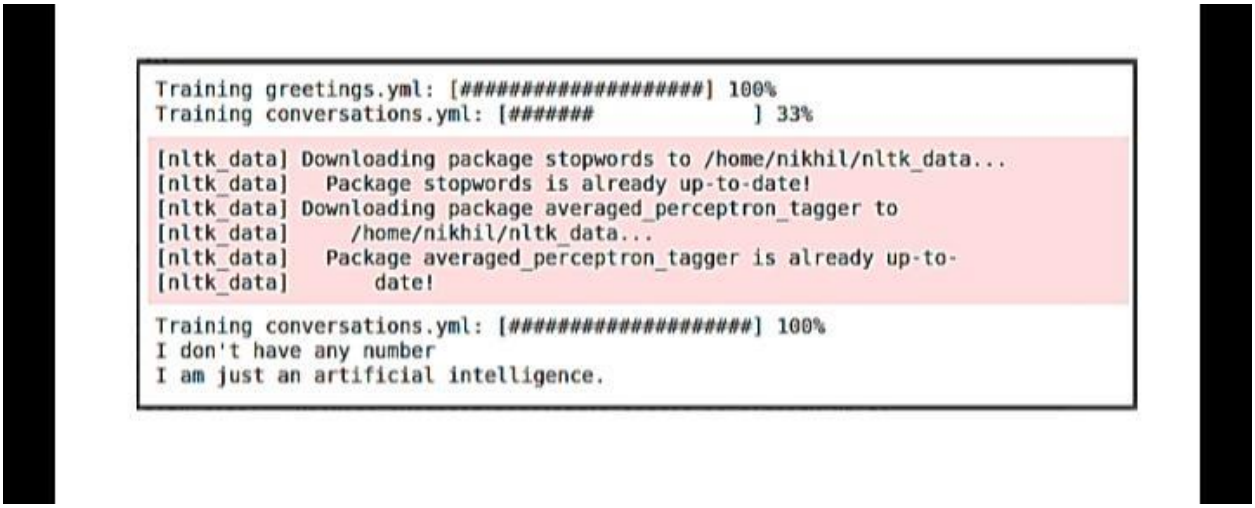
```
# Import “chatbot” from
# chatterbot package.
From chatterbot import ChatBot
# Inorder to train our bot, we have
# to import a trainer package

# “ChatterBotCorpusTrainer”
From chatterbot.trainers import ChatterBotCorpusTrainer
# Give a name to the chatbot “corona bot”
# and assign a trainer component.
Chatbot=ChatBot(‘corona bot’)
# Create a new trainer for the chatbot
```

```
Trainer = ChatterBotCorpusTrainer(chatbot)
# Now let us train our bot with multiple corpus
Trainer.train("chatterbot.corpus.english.greetings",
                "chatterbot.corpus.english.conversations" )
Response = chatbot.get_response('What is your Number')

Print(response)
Response = chatbot.get_response('Who are you?')
Print(response
```

Output:



Phase 4

Introduction:

Building a chatbot and integrating it into a web app using Flask is a common and practical application of chatbot technology. Flask is a lightweight Python web framework that's well-suited for creating web applications, and you can easily integrate a chatbot into a Flask-based web app.

1. Install Required Libraries:

Make sure you have Flask and any other necessary libraries installed. You may want to use a chatbot framework or library, like ChatterBot or Rasa, to simplify chatbot development. Install them using pip:

Bash pip install Flask pip install chatterbot

2. Create a Flask Web App:

Create a Flask web app by creating a Python file, e.g., `app.py`. Here's a basic example:

Python from flask import Flask, render_template, request

```
App = Flask(__name__)
@app.route('/')    def index():
    Return render_template('index.html')
If __name__ == '__main__':
    App.run()
```

3. Create HTML Template:

Create an HTML template for your chat interface. You can use this template to collect user input and display chatbot responses. Save this template as `index.html` in a `templates` directory in your project folder.

```
Html
<!DOCTYPE html>
<html>
<head>
    <title>Chatbot Example</title>
</head>
<body>
    <h1>Chatbot Example</h1>
    <div id="chatbox">
        <div id="chatlog">
```

<!—Chat messages will appear here →

<input type=”text” id=”user_input” placeholder=”Type your message…”>

<button id=”send”>Send</button>

</body>

</html>

4. Implement Chatbot Logic:

Implement your chatbot logic in Python. You can use ChatterBot, Rasa, or any other chatbot framework of your choice. Define a function in your Flask app that handles the chatbot interaction, taking user input and returning bot responses.

5. Handle User Input:

Add JavaScript to your HTML template to handle user input and display chatbot responses. You can use AJAX or WebSocket to communicate with your Flask app. Here’s a simple example using jQuery:

Html

<script src=https://code.jquery.com/jquery-3.6.0.min.js></script>

```
<script>
    $(document).ready(function () {
  $("#send").click(function () {
 var user_input = $("#user_input").val();
            $("#chatlog").append("<p>User: " + user_input + "</p>");

            $("#user_input").val("");

            $.ajfunction
 type: "POST",  url: "/get_response",   data: JSON.stringify({ user_input: user_input }),
  contentType: "application/json; charset=utf-8", dataType: "json",                success: function (data) {

                $("#chatlog").append("<p>Bot: " + data.bot_response + "</p>");

            }

        });
```

```
        });

    });
</script>
```

6. Create a Flask Route for Chatbot Interaction:

In your Flask app, create a route to handle chatbot interactions. This route should take user input, process it, and return the chatbot's response.

```
Python    from flask import request, jsonify
@app.route('/get_response', methods=['POST'])    def get_response():

    User_input = request.json['user_input']
    # Process user_input and get the chatbot's response
bot_response = get_chatbot_response(user_input)        return jsonify({'bot_response': bot_response})
```

7. Run the Flask App:

Run your Flask app using the command `python app.py`. You can access your chatbot web app at `http://localhost:5000` in your web browser.

8. Test and Refine:

Test your chatbot and make any necessary refinements to improve its functionality and user experience.

These are the basic steps to integrate a chatbot into a Flask web app. Depending on your chatbot's Complexity, you may need to add features like natural language processing, user authentication, and more to create a fully functional and secure chat application.

Program:

```
Import pickle from flask import Flask, request, jsonify import pandas as pd from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Embedding, LSTM, Dense from tensorflow.keras.preprocessing.text import Tokenizer from tensorflow.keras.preprocessing.sequence import pad_sequences from spellchecker import SpellChecker  # You may need to install this library


App = Flask(__name)

# Load your dialog dataset dataset = pd.read_csv('dialog.csv')
```

```
# Preprocess the dataset tokenizer = Tokenizer()
tokenizer.fit_on_texts(dataset[‘a’]) total_words =
len(tokenizer.word_index) + 1

# Tokenize and pad the sequences input_sequences = [] for line
in dataset[‘a’]:


    Token_list = tokenizer.texts_to_sequences([line])[0]      for I
in range(1, len(token_list)):


        N_gram_sequence = token_list[:I + 1]
input_sequences.append(n_gram_sequence)
max_sequence_length = max([len(x) for x in input_sequences])
input_sequences = pad_sequences(input_sequences,
maxlen=max_sequence_length, padding=’pre’)

# Separate input and target sequences X = input_sequences[:, :-
1] y = input_sequences[:, -1]

# Create and compile the model model = Sequential()
model.add(Embedding(total_words, 100,
input_length=max_sequence_length – 1))
model.add(LSTM(150)) model.add(Dense(total_words,
activation=’softmax’))
model.compile(loss=’categorical_crossentropy’,
optimizer=’adam’)
```

Train the model with a specified number of epochs
model.fit(X, y, epochs=10) # You can specify the number of epochs here

Save the trained model to a pkl file with open('model.pkl', 'wb') as model_file:

    Pickle.dump(model, model_file)

Initialize a spell checker spell = SpellChecker()

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

    User_input = request.json['user_input'] # Assuming you're receiving JSON input

    # Load the trained model from the pkl file

    With open('model.pkl', 'rb') as model_file:

        Model = pickle.load(model_file)

    # Preprocess user input   user_input = user_input.lower() # Convert to lowercase   user_input = spell.correction(user_int) # Correct spelling   input_sequence = tokenizer.texts_to_sequences([user_input])[0]   input_sequence = pad_sequences([input_sequence], maxlen=max_sequence_length – 1, padding='pre')

    # Generate a response using the trained model response_sequence = []   for _ in range(max_sequence_length – 1):

```
        Predicted_word_index =
model.predict_classes(input_sequence, verbose=0)
predicted_word = “”         for word, index in
tokenizer.word_index.items():            if index ==
predicted_word_index:


                Predicted_word = word

                Break

        Input_sequence       =
        pad_sequences([input_sequence.tolist()        +
        [predicted_word_index]], maxlen=max_sequence_length –
1, padding=’pre’)
response_sequence.append(predicted_word)

    Response = ‘ ‘.join(response_sequence)

    Return jsonify({‘response’: response}) if _name_ ==
‘_main_’:     app.run(debug=True)
```

Report:

All the above instructions are installed and executed

Successfully.

Project by:

Name: RAMESH .N

Dept: CSE III YEAR

Reg No: 621421104041

College code: 6214

Group:IBM-GROUP -5