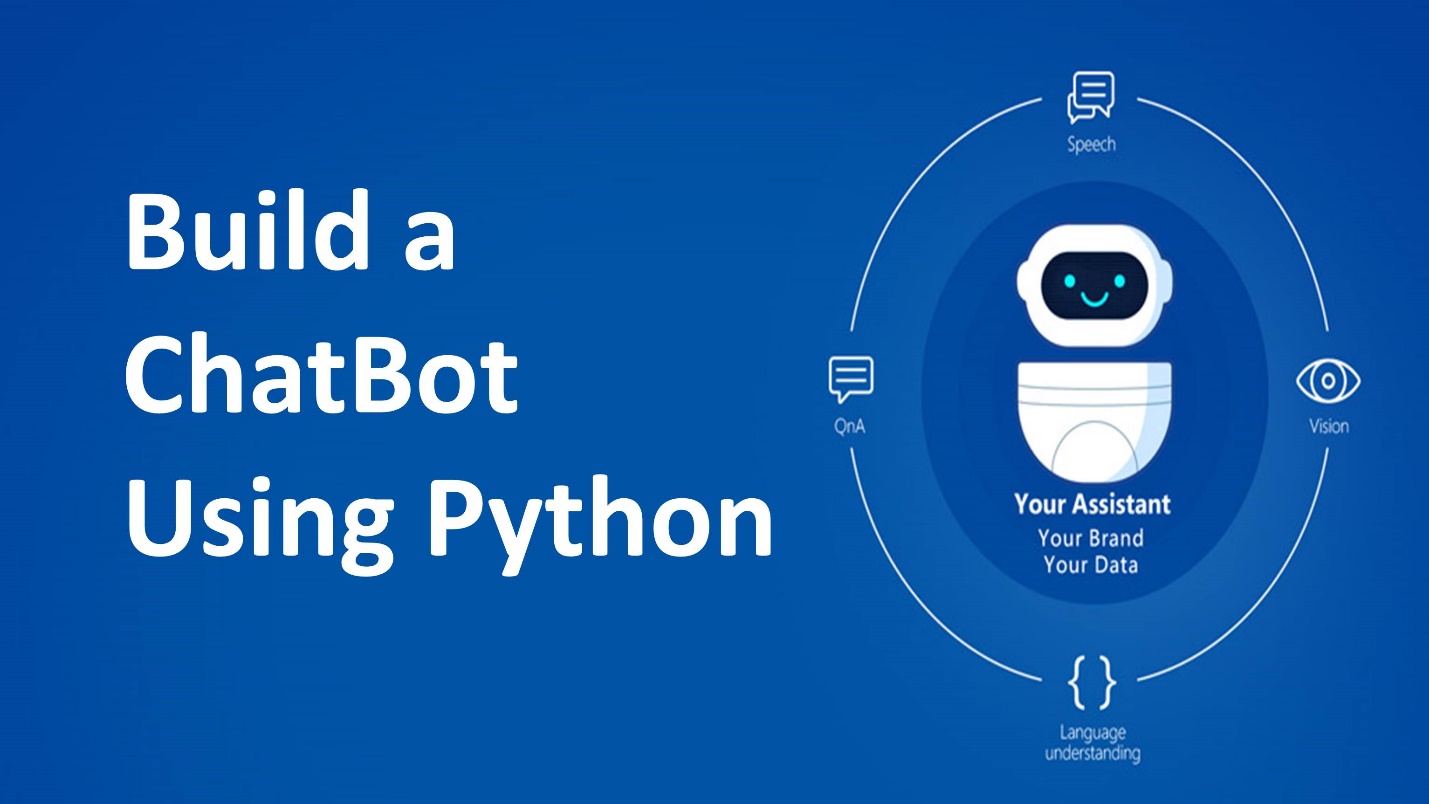
**PHASE 4 DOCUMENT SUBMISSION**

**Project Tittle:** CREATE A CHATBOT IN PYTHON

**PHASE 4:** Development Part 2

**Topic**: In this section continue building the project by performing different activities like feature engineering, model training, evaluation etc as per the instructions in the project.



**INTRODUCTION:**

**Rule-Based Chatbot:**

A rule-based chatbot operates on predefined rules and patterns. It matches user input against a set of rules and generates a response accordingly. Here's a step-by-step guide to building a simple rule-based chatbot in Python:

**Define Responses:**

Create a set of predefined responses for different user inputs. These responses will guide the chatbot's behavior.

**Input Processing:**

Take user input and preprocess it, such as converting it to lowercase for easier handling and understanding.

**Rule Matching:**

Match the preprocessed user input against predefined rules and patterns to determine an appropriate response.

**Response Generation:**

Based on the matched rule, generate a response to the user.

**Interactive Loop:**

Set up a loop where the chatbot continually interacts with the user, taking their input and providing responses until the conversation is ended**.**

**Feature Engineering:**

**Data Collection:**

First, you need to gather data or information that your chatbot will use. In this case, you can use a weather API to fetch weather data based on user queries.

**Text Preprocessing:**

Before performing feature engineering, preprocess the user's input to remove any irrelevant information and convert it to a standard format. You may also need to tokenize and lemmatize the text.

**Feature Engineering:**

Create features from the preprocessed text that the chatbot can use to determine the appropriate response. In this case, you might want to extract keywords like "weather," "temperature," and the location (city) from the user's query.

import re

def extract\_weather\_features(user\_input):

# Regular expressions to identify keywords and location

weather\_keywords = r"(weather|temperature)"

location\_pattern = r"in (\w+)"

# Extract keywords

keywords = re.search(weather\_keywords, user\_input, re.I)

location = re.search(location\_pattern, user\_input, re.I)

return {

"weather\_keywords": bool(keywords),

"location": location.group(1) if location else None

}

**Response Generation:**

Based on the extracted features, you can generate responses. If the user asks about the weather, use the location feature to fetch weather data from your API and provide a response.

def get\_weather\_response(features):

if features["weather\_keywords"]:

if features["location"]:

# Fetch weather data for the specified location from your API

weather\_data = fetch\_weather\_data(features["location"])

if weather\_data:

return f"The weather in {features['location']} is {weather\_data['temperature']}°C."

else:

return "I'm sorry, I couldn't find information for that location."

else:

return "Please specify a location for the weather query."

else:

return "I'm sorry, I can't help with that request."

def fetch\_weather\_data(location):

# Use an API to get the weather data for the location

# Return the data in a structured format

# You might want to use libraries like requests to make API calls

Pass

**Chatbot Interaction:**

Finally, you can use the extract\_weather\_features and get\_weather\_response functions to interact with users:

while True:

user\_input = input("You: ")

user\_input = user\_input.lower() # Convert to lowercase for case-insensitivity

features = extract\_weather\_features(user\_input)

response = get\_weather\_response(features)

print("Chatbot:", response)

**Model Training:**

* **Install Dependencies:**

You will need Python and a few libraries, such as nltk for natural language processing. Install them using pip:

pip install nltk

* **Data Collection:**

Collect and prepare a dataset of user inputs and responses. For a rule-based chatbot, you'll define rules for specific user inputs and map them to responses.

* **Preprocessing:**

Clean and preprocess the text data. You may want to lowercase the text, remove punctuation, and tokenize it.

* **NLP Library:**

You can use the Natural Language Toolkit (NLTK) for text processing. Here's an example of preprocessing and tokenization.

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

def preprocess(text):

# Tokenization

tokens = word\_tokenize(text)

# Remove stopwords

tokens = [word for word in tokens if word.lower() not in stopwords.words('english')]

# Lemmatization

lemmatizer = WordNetLemmatizer()

tokens = [lemmatizer.lemmatize(word) for word in tokens]

return tokens

* **Define Rules**:

Create a set of rules that match user input to predefined responses. Here's a

simple example:

# Define rules

rules = {

'hi': 'Hello!',

'how are you': 'I am just a computer program, but thanks for asking!',

'bye': 'Goodbye!',

}

* **Chatbot Logic**:

Implement the chatbot's logic to find a matching rule and generate a response.

def chatbot\_response(user\_input):

user\_input = user\_input.lower()

for rule, response in rules.items():

if rule in user\_input:

return response

return "I'm sorry, I don't understand."

# Example usage

while True:

user\_input = input("You: ")

if user\_input.lower() == 'exit':

break

response = chatbot\_response(user\_input)

print("Chatbot: " + response)

* **Testing**:

Test your chatbot by running it and engaging in a conversation. You can add more rules and responses to make it more interactive

**Evaluation as per the Instructions:**

**1. Evaluation Metrics**:

Decide on the evaluation metrics to measure your chatbot's performance. Some common metrics for chatbots include accuracy, precision, recall, F1-score, and human evaluation (e.g., using crowdworkers to assess responses).

**2. Test Data:**

Prepare a test dataset with various user inputs and the expected chatbot responses. This dataset should cover a wide range of scenarios to assess how well your chatbot handles different types of input.

**3. Automated Testing:**

If your chatbot is rule-based, you can automate the testing process by comparing the chatbot's responses with the expected responses. You can use a function like this to automate testing:

def test\_chatbot(chatbot\_function, test\_data):

correct = 0

total = 0

for user\_input, expected\_response in test\_data:

chatbot\_response = chatbot\_function(user\_input)

if chatbot\_response == expected\_response:

correct += 1

total += 1

accuracy = correct / total

return accuracy

**4. Human Evaluation:**

For more qualitative assessments, you can involve human evaluators to judge the quality of chatbot responses. This can involve asking human evaluators to rate responses for fluency, relevance, and correctness.

**5. Benchmarking:**

Compare your chatbot's performance to existing chatbots or industry standards if applicable. This can help you understand where your chatbot stands in terms of its capabilities.

**6. A/B Testing:**

If your chatbot is used in a real-world application, conduct A/B testing to compare the chatbot's performance against different versions, chatbot parameters, or approaches. Monitor user interactions and gather feedback.

**7. Log Analysis:**

Analyze chatbot logs to understand user interactions, frequently asked questions, and user satisfaction. Use this information to make improvements.

**8. Continuous Improvement:**

Use the evaluation results to identify areas for improvement. You can refine the rules, add more responses, or explore machine learning approaches if necessary.

**9. User Feedback:**

Gather user feedback and incorporate it into your chatbot's development. Real user feedback is invaluable for enhancing the chatbot's performance.

**10. Documentation:**

Document the results of your evaluation and any improvements you make. This documentation will help you track the chatbot's progress over time and guide future development.