PROJECT TITLE:CREATE A CHATBOT IN PYTHON

Phase-5: Project Documentation and Submission

Objective:

In this part, I will give a brief explanation on what I have done in all the other phases and prepare a finalized document on my chatbot. I will clearly outline the problem statement, design thinking process, and the phases of development and I will describe the libraries used and the integration of NLP techniques and I will explain how the chatbot interacts with users and the web application.

PROBLEM DEFINITION:

The challenge is to create a chatbot in python that provides exceptional customer service,answering user queries on a website or application.The objective is to deliver high quality support to users,ensuring a positive user experience and customer satisfication.

DESIGN THINKING APPROACH:

1.FUNCTIONALITY

Define scope of the chatbot's ability including:

- Natural language toolkit .
- efficient techniques.
- Privacy and data security.

2.USER INTERFACES:

-Design a user-friendly interface.

-Inclusive design principles.

-Being a natural language interface.

3.NATURAL LANGUAGE PROCESSING:

Implement NLP techniques for:

-Conversational understanding of user input.

-Handling user inquiries effectively.

-Sentimental analysis.

4.RESPONSES:

-Plan a range of responses.

-Accurate answers to user questions.

-Clear communication between user and chatbot.

5.INTEGRATION:

-Decide integration details.

-Connecting the chatbot with various platforms.

-Various social platforms and applications.

-Accessibility features to all users.

6.TESTING AND IMPROVEMENTS:

-Continuous testing and improvement plan.

-User feedback collection.

-Regularly update your algorithms.

-Adherence to privacy and security standards

By following these suggestion you can continuously improve your chatbot and ensure that it is providing accurate details to the user.

TO IMPLEMENT CHATBOT USING PYTHON:

Objectives of chatbot:

- Answering user queries: Providing quick and accurate answers to userquestions, whether they are informational queries or troubleshooting requests.

- Customer Support and Services: Assisting users with productrelatedinquiries, technical issues or general customer service queries.

- Continuous learning and Improvement: continuouslylearning from userinteractions to enhance the chatbots capabilities, language understanding and responses.

Conduct User Interface:

Gather insights through,

- Surveys: Create surveys to collect quantitative data about user preference and demographics.

- Observation: Observe user behavior and interaction in relevant code

Functionalities:

-Determine that the key functionalities of the chatbot will offer based on user needs and objectives.

-Prioritize these functionalities and design them to align with chatbot's purpose.

Natural Language processing :

Natural Language Processing plays a crucial role in chatbot development, enabling chatbots to understand, interpret, and generate human language effectively.

- **Language Understanding:** NLP allows chatbots to understand and extract meaning from user inputs. It involves techniques like tokenization, part-of-speech tagging, and named entity recognition to break down and analyze text.

- **Continuous Learning:** Use reinforcement learning and machine learning techniques to continuously improve their language.

TESTING AND DEPLOYMRNT:

- Once the chatbot is implemented I will test the chatbot. During testing, the chatbot undergoes rigorous evaluation, including unit testing to validate individual components, functional testing to assess its performance in various user scenarios, and integration testing to confirm seamless communication with external systems.

- Once the chatbot is thoroughly tested, I will deploy it to the website or app so that users can interacting with them.

EVALUTION:

- Evaluating a chatbot's performance is a critical process to ensure it meets user expectations.

- Key factors include assessing accuracy, response quality, user satisfaction,and error handling.

- Compliance with data privacy and security regulations is essential.

- Scalability and continuous learning mechanisms ensure the chatbot remains effective over time.

Dataset used:

Dataset Link: **https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot**

The dataset I used for my project that is creating a chatbot is given in the above link. I loaded and preprocessed my data using machine learning techniques. I performed different activities like feature engineering, model training and evaluation as per the instructions in the project. I built the chatbot by integrating it into a web app using Flask.

Creating a CSV file named “chatbot_data.csv”:

The first step is to create a CSV file using the given dataset. This is how I created a structure for my chatbot data:

1. Open a text editor or spreadsheet software (e.g., Microsoft Excel, Google Sheets).
2. Create two columns: "Question" and "Category."
3. In the "Question" column, enter a list of user questions or queries.
4. In the "Category" column, specify the corresponding category for each question.
5. Save the file with a ".csv" extension, such as "chatbot_data.csv." Here's a representation of what your CSV file might look like:

Question, Category

How can I track my order?, Order Tracking

What is your return policy?, Returns

Tell me about your product warranty., Product Information

How can I contact customer support?, Contact Information

Can you help me with a technical issue?, Technical Support

Once I've created CSV file with my own data, I used it as input to the chatbot program provided in the previous response. I replaced the

'customer_service_data.csv' path with the path to my newly created CSV file.

Implementing Basic User Interactions:

I provided a code using Python's NLTK for natural language processing and a rule based approach to respond to common customer queries.

To create a customer service chatbot, follow these steps:

1. Install the necessary libraries:

pip install nltk

2. Create a Python script for your chatbot:

import nltk from nltk.chat.util import Chat, reflections # Define

patterns and responses for the chatbot patterns = [

(r'hi|hello|hey', ['Hello!', 'Hi there!', 'Hey!']),

(r'help', ['How can I assist you?', 'What do you need help with?']),

(r'order status', ['Please provide your order number, and I will check the status.']),

(r'(\d{5})', ['Your order {} is currently being processed.']),

(r'bye|quit', ['Goodbye!', 'See you later!']),

]

Define reflection pairs for transforming user inputs ***custom_reflections = {***

"my": "your",

"your": "my",

"you": "I",

"I": "you",

}

Create an instance of the Chat ***class chatbot =***

Chat(patterns, reflections=reflections) # Main

loop for interacting with the chatbot

print("Customer Service Chatbot: How can I assist

you today? (Type 'bye' to exit)") while True:

user_input = input("You: ") if

user_input.lower() == 'bye':

print("Customer Service Chatbot: Goodbye!")

break response = chatbot.respond(user_input)

print("Customer Service Chatbot:", response)

I used NLTK's `Chat` class to define patterns and responses for common customer service queries. The chatbot can respond to greetings, requests for help, and queries about order status.

Data Loading:

import pandas as pd data = pd.read_csv('chatbot_data.csv')

print(data.head()) # Display the first few rows of the dataset

customer_queries = data['customer_query'].values

bot_responses = data['bot_response'].values

Data Preprocessing:

Data preprocessing is a crucial step in building a chatbot. It involves cleaning and preparing the text data so that it can be used effectively in a machine learning model. Above is a Python code for data preprocessing in my chatbot. import pandas as pd import re import string from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer from

sklearn.preprocessing import LabelEncoder data =

pd.read_csv('chatbot_data.csv') def preprocess_text(text):

text = text.lower() text = text.translate(str.maketrans('', '',

string.punctuation)) text = ' '.join(text.split()) return text

data['customer_query'] = data['customer_query'].apply(preprocess_text)

data['bot_response'] = data['bot_response'].apply(preprocess_text)

customer_queries = data['customer_query'].values bot_responses =

data['bot_response'].values

X_train, X_test, y_train, y_test = train_test_split(customer_queries, bot_responses, test_size=0.2, random_state=42) tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

X_test_tfidf = tfidf_vectorizer.transform(X_test)

Data Loading and Preprocessing Using a CSV file:

Building a customer service chatbot that loads and preprocesses data from a CSV file using machine learning involves several steps. Here's a Python program that explains the process using the pandas library for data handling and scikit-learn for machine learning. This will focus on data loading, preprocessing, and training a simple classification model to respond to customer queries based on the loaded CSV data.

Make sure you have the necessary libraries installed: pip

install pandas scikit-learn nltk

Now, create a Python script for your chatbot:

import pandas as pd from sklearn.feature_extraction.text import

TfidfVectorizer from sklearn.naive_bayes import MultinomialNB

from sklearn.pipeline import Pipeline from

sklearn.preprocessing import LabelEncoder from

sklearn.model_selection import train_test_split import nltk from

nltk.corpus import stopwords from nltk.tokenize import

word_tokenize # Load and preprocess data from the CSV file def

load_and_preprocess_data(csv_file):

data = pd.read_csv(csv_file)

X = data['Question'].values y =

data['Category'].values

Text preprocessing: remove stopwords and tokenize stop_words =

set(stopwords.words('english'))

X = [" ".join([word for word in word_tokenize(sentence) if word.lower() not in

stop_words]) for sentence in X] return X, y

Create a classification model and train it def

train_classification_model(X, y):

Define a text classification pipeline with TF-IDF vectorization and a Naive

Bayes classifier

text_clf = Pipeline([

```
        ('tfidf', TfidfVectorizer()),
        ('clf', MultinomialNB())
    ])
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    # Fit the model on the training data
text_clf.fit(X_train, y_train)     # Evaluate the
model     accuracy = text_clf.score(X_test,
y_test)     print(f"Model Accuracy:
{accuracy:.2f}")      return text_clf
# Main function to interact with the chatbot def
chatbot(csv_file):
    X, y = load_and_preprocess_data(csv_file)     model = train_classification_model(X, y)
print("Customer Service Chatbot: How can I assist you today? (Type 'exit' to quit)")
while True:
        user_input = input("You: ")
if user_input.lower() == 'exit':
            print("Customer Service Chatbot: Goodbye!")            break
        predicted_category = model.predict([user_input])        print(f"Customer
Service Chatbot: This falls under the category:
{predicted_category[0]}")  if
__name__ == '__main__':
    csv_file = 'customer_service_data.csv'  # Replace with the path to your CSV file
chatbot(csv_file)
```

Model Training in Chatbot:

Training a custom customer service chatbot typically involves using Natural Language Processing (NLP) models and datasets. In this part, I used a pre-trained model (GPT-3) for chatbot functionality using the OpenAI GPT-3 API.

Before running this code, make sure you have an API key from OpenAI and have the `openai` Python package installed. I installed it using `pip install openai`.

Here's a Python script for a Flask app that uses the OpenAI GPT-3 API to create a customer service chatbot:

```
from flask import Flask, render_template, request, jsonify
import openai app = Flask(__name__) # Your OpenAI API key
api_key = "YOUR_API_KEY"
# Initialize the OpenAI API client openai.api_key =
api_key
@app.route('/')
def index():
   return    render_template('index.html')
@app.route('/chat',    methods=['POST'])
def chat():
   user_message = request.form['user_message']    chatbot_response
= generate_chatbot_response(user_message)    return
jsonify({'response': chatbot_response}) def
generate_chatbot_response(user_message):
   prompt = f"Customer: {user_message}\nChatbot:"    response =
openai.Completion.create(       engine="text-davinci-002",  # You can
choose a different engine       prompt=prompt,       max_tokens=50,  #
Adjust this as needed       stop=["\n"]  # Stop generating after a newline
character
   )
   return response.choices[0].text.strip() if
__name__          ==          '__main__':
app.run(debug=True)
```

This code sets up a Flask web application that communicates with the OpenAI GPT-3 API to generate chatbot responses. I replaced `"YOUR_API_KEY"` with my actual OpenAI API key.

I also created HTML templates as shown in the previous response to create a user interface for the chatbot.

Program Code for Evaluation in Chatbot:

When evaluating the output of a customer service chatbot using Flask and a CSV file, I examined how the chatbot interacts with users and provides responses. Here's a code for evaluating the output and collecting feedback from users: from flask import Flask, render_template, request, jsonify, redirect, url_for app = Flask(__name__)

Load the CSV file with user queries and chatbot responses

responses = [] @app.route('/') def index():

return render_template('index.html')

@app.route('/chat', methods=['POST'])

def chat():

user_message = request.form['user_message'] chatbot_response = get_chatbot_response(user_message)

Log the conversation for evaluation responses.append({'user_message': user_message, 'chatbot_response': chatbot_response})

return jsonify({'response': chatbot_response})

@app.route('/feedback', methods=['GET', 'POST']) def feedback(): if request.method == 'GET':

return render_template('feedback.html', responses=responses) elif request.method == 'POST':

Collect user feedback from the form user_feedback = request.form['user_feedback']

Store feedback in a database or file store_user_feedback(user_feedback)

return redirect(url_for('index')) def get_chatbot_response(user_message):

Implement chatbot logic here

Return the chatbot's response based on the user's input

return "This is a sample chatbot response." def store_user_feedback(feedback):

```
    # Implement code to store user feedback in a database or file     # I
can save feedback with timestamps and user identifiers if __name__
== '__main__':
    app.run(debug=True)
```

In this code, I have added the ability to log user interactions and collect feedback. I created HTML templates (`index.html` and `feedback.html`) for the chat interface and feedback form, respectively.

Here's an overview of how this code works:

1. Users interact with the chatbot on the website.
2. The chatbot's responses and user messages are logged in the `responses` list for evaluation.
3. Users can provide feedback through the feedback form. The feedback can be about the chatbot's responses, user experience, or any other relevant aspect.
4. The feedback is stored in a database or file, as I needed a mechanism to collect and analyze it.
5. Regularly review the chatbot's interactions, user feedback, and user satisfaction to make improvements and enhancements.

This code provides the foundation for evaluating the output and collecting feedback, but it's essential to customize specific requirements and how to handle and store feedback data.

Generating Chatbot Responses Using CSV File:

To generate output from a customer service chatbot using Flask and a CSV file for a website, I implemented the logic for the chatbot to respond to user messages. Here's how I generated chatbot responses based on user input and the CSV file: from flask import Flask, render_template, request, jsonify import csv import random app = Flask(__name__) # Load CSV data with open('chatbot_responses.csv', 'r') as csvfile:

```
    reader = csv.reader(csvfile)     responses
= [row[0] for row in reader]
```

Define a function to get a random response from the CSV def get_res In conclusion, creating a chatbot is a dynamic and iterative process that involves a combination of technology, design, user experience, and data management. A well implemented chatbot can provide significant benefits to both your organization and your users. It can enhance

customer service, streamline interactions, and provide a convenient and efficient way for users to get the information they need.

```
ponse():
    return random.choice(responses)
@app.route('/')
def index():
    return    render_template('index.html')
@app.route('/chat',    methods=['POST'])
def chat():
    user_message = request.form['user_message']
chatbot_response = get_response()    return
jsonify({'response': chatbot_response}) if __name__
== '__main__':
    app.run(debug=True)
```

In this the `/chat` route receives user messages via a POST request, and it generates chatbot responses using the `get_response()` function, which selects a random response from the CSV file.

My HTML template for the chat interface (`index.html`) should include a form for users to input their messages, and JavaScript code to send the user's message to the `/chat` endpoint and display the chatbot's response.

Here's a code of an HTML template:

```
<!DOCTYPE html>
<html>
<head>
    <title>Customer Service Chatbot</title>
</head>
<body>
    <div id="chat-container">
```

```
    <div id="chat-log"></div>
    <input type="text" id="user-message" placeholder="Type a message...">
    <button id="send-button">Send</button>
  </div>
  <script>  const chatLog = document.getElementById('chat-log');  const
userMessage = document.getElementById('user-message');  const sendButton =
document.getElementById('send-button');  sendButton.addEventListener('click', () =>
{  const message = userMessage.value;  userMessage.value = '';
chatLog.innerHTML += `<p><strong>You:</strong> ${message}</p>`;  fetch('/chat',
{  method: 'POST',  body: new URLSearchParams({ user_message:
message }),  headers: { 'Content-Type': 'application/x-www-form-urlencoded' }
    })
    .then(response => response.json())
    .then(data => {  chatLog.innerHTML +=
`<p><strong>Chatbot:</strong> ${data.response}</p>`;
    });
  });
  </script>
</body>
</html>
```

With this setup, when a user enters a message and clicks "Send," the user's message is sent to the `/chat` endpoint, and the chatbot's response is displayed on the web page. The chatbot responses are randomly selected from the CSV file.

Output for a Customer Service Chatbot:

Below is a chatbot conversation, assuming that the chatbot responses are randomly selected from a CSV file, as mentioned in the previous code:

User: Hello, I need help with my account.

Chatbot: Sure, I can help you with that.

User: My password isn't working. Can you reset it?

Chatbot: I'm sorry to hear that. Let me assist you with resetting your password.

User: Thank you. How long does it usually take to reset the password?

Chatbot: Password resets usually take about 10-15 minutes to process.

User: Great, thanks for the information.

Chatbot: You're welcome! If you have any more questions, feel free to ask. User: What are your customer support hours?

Chatbot: Our customer support is available 24/7, so you can reach out to us anytime.

User: That's perfect! Thanks for your help.

Chatbot: No problem. If you have any other questions, just let me know.

This is a conversation between a user and a customer service chatbot. Since I have loaded and preprocessed my data and I have also implemented specific functions to train and model my chatbot in the previous phases, my chatbot is ready.

CONCLUSION:

In conclusion, creating a chatbot is a dynamic and iterative process that involves a combination of technology, design, user experience, and data management. A well implemented chatbot can provide significant benefits to both your organization and your users. It can enhance customer service, streamline interactions, and provide a convenient and efficient way for users to get the information they need.