PHASE 5: PROJECT DOCUMENTATION AND SUBMISSION

PROJECT TITLE: CREATE A CHATBOT IN PYTHON

In today's digital age, chatbots have emerged as invaluable tools for streamlining communication and aiding users. Chatbots encompass various categories, and among them are rule-based chatbots, which supply predefined responses grounded commands or trigger keywords. For a range of applications, including customer support, handling frequently asked questions (FAQs), or facilitating interactive information retrieval, creating a rule-based chatbot proves to be a cost-effective and efficient solution.

This project's objective is to conceptualize, construct, and deploy a rule-based chatbot using the Python programming language, specifically tailored to address distinct user queries and commands**.**

PROBLEM STATEMENT:

The essence of this project entails crafting a rule-based chatbot with the capability to comprehend and furnish responses to a defined set of user commands or inquiries. This chatbot should possess the ability to discern specific keywords or phrases within a user's input and generate suitable responses in line with those inputs. The primary emphasis lies in the implementation of the chatbot's underlying logic, the creation of an intuitive user interface when applicable, and ensuring the seamless deployment of the chatbot.

By addressing this challenge, the objective is to develop an effective rule-based chatbot that enhances user interactions by automating responses to specific queries or commands, ultimately improving user satisfaction and operational efficiency in the designated domain.

DESIGN THINKING:

Design thinking is a creative and user-centric problem-solving approach that is particularly useful in the development of chatbots. It involves several iterative stages to ensure that the chatbot is designed with the end user in mind. Here is how the design thinking process can be applied to creating a rule-based chatbot in Python:

1.Data repository

Action: A Kaggle dataset can be utilized to establish a repository or catalogue of predetermined replies suitable for various user inputs.

Rationale: This approach is logical because Kaggle is a reputable platform that hosts diverse datasets, some of which might contain text or conversational data that can be repurposed for creating a database of predefined responses in a chatbot or similar application.

2.Develop function to assess and furnish a reply

Action: Create a function that, given user input and an existing response database, will determine and provide a response.

Rationale: It enables the chatbot to dynamically select the most appropriate response from the database based on the user's input. This mechanism ensures the chatbot's ability to engage in meaningful conversations. Build a simple command-line interface (CLI) to facilitate user interactions.

3.Developing a loop

Action: Develop an ongoing loop that consistently takes in user input and produces responses from the chatbot until the predefined exit condition is satisfied.

Rationale: The need for a continuous interaction loop in the chatbot, which remains active to accept user input and generate responses until a specified condition (e.g., user-initiated exit) is met. This design ensures the chatbot's responsiveness throughout the conversation.

4.Response Crafting

Action: Establish a method for generating responses that are contextually appropriate and pertinent to user interactions.

Rationale: Emphasizing the importance of delivering meaningful and context aware responses to users. It highlights the flexibility in response generation methods, ranging from rule-based responses to more advanced machine learning approaches, depending on the chatbot's specific requirements and capabilities.

5.Error Management

Action: Create effective procedures to gracefully manage unexpected or incorrect user inputs, ensuring that the chatbot can respond intelligently and maintain a smooth conversation flow.

Rationale: It is crucial for maintaining a positive user experience. Error handling mechanisms should prevent the chatbot from crashing or giving incoherent responses when faced with unexpected or incorrect input, providing users with meaningful feedback or guidance instead.

6. Contextual Flow Control

Action: Incorporate context tracking to uphold the conversation's continuity and grasp references to prior interactions.

Rationale: Context management is essential for the chatbot to comprehend and respond coherently to user queries or statements based on the conversation's history.

7.Seamless Integration

Action: When incorporating the chatbot into a broader system, guarantee a smooth harmonization with other elements like databases, CRM systems, or web services.

Rationale: Ensuring that the chatbot operates effectively within a larger ecosystem, facilitating data exchange and communication with other system components.

8. Sustainability and Enhancement

Action: Develop a structured approach for consistently upkeeping, enhancing,

and adapting the chatbot.

Rationale: Acknowledging the dynamic nature of chatbot development. Maintenance ensures the chatbot remains functional, while regular updates and improvements guarantee its relevance and effectiveness as user. User feedback serves as a valuable source for fine-tuning and enhancing the chatbot's capabilities.

9. Documentation Preparation

Action: Generate thorough and user-friendly documentation, serving as a valuable resource for both end-users and developers to comprehend the chatbot's usage and upkeep procedures.

Rationale: Documentation is pivotal for ensuring that users can effectively interact with the chatbot, while developers can maintain and enhance its functionality. It provides clarity, guidance, and essential information for all.

By applying the design thinking process, you can create a user-centric rule-based chatbot that effectively addresses user needs and provides a valuable and user-friendly experience. This iterative approach ensures that the chatbot evolves and improves over time based on real user feedback and insights.

PHASES OF DEVELOPMENT:

DEVELOPMENT PART 1:

In this part you will begin building your project by loading and preprocessing the dataset. Start building the chatbot by preparing the environment and implementing basic user interactions. Install required libraries, like transformers for GPT-3 integration and flask for web app development.

DATASET USED: https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

ENVIRONMENT: VISUAL STUDIO, GOOGLE COLLAB

DEVELOPING THE CHATBOT

(TRAINING AND TESTING DATASET)

CODE:

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
```

These lines import the necessary Python libraries for data manipulation and natural language processing (NLP).

- `pandas` is a powerful library for data analysis and manipulation.

- `nltk` (Natural Language Toolkit) is a popular library for NLP.

- `nltk.corpus` provides access to stop words, and `nltk.tokenize` is used for text tokenization.

- `string` is a Python standard library for working with string manipulation.

```
dataset_file = "/kaggledataset.csv"
```

- This line defines the path to the CSV file containing your dataset. Make sure to replace `"/kaggledataset.csv"` with the actual path to your dataset file.

```
data = pd.read_csv(dataset_file)
```

- This line reads the dataset from the CSV file and loads it into a Pandas DataFrame called `data`. It assumes that the CSV file is properly formatted and contains the data you want to work with.

```
print(data.head())
```

- This line displays the first few rows of the dataset using the `head()` function. It provides a quick preview of the data to help you understand its structure.

```
print(data.isnull().sum())
```

- This line checks for missing values in the dataset using the `isnull()` function. It counts the number of missing values for each column using `sum()`. This can help identify areas of the dataset where data might be missing.

```
data.fillna(data.mean(), inplace=True)
```

s with the mean of their respective columns. It fills in missing data in numerical columns with the mean value of that column.

```
data.to_csv('preprocessed_dataset.csv', index=False)
```

- These lines save the preprocessed data to a new CSV file named `'preprocessed_dataset.csv'`. The `index=False` parameter prevents the DataFrame's index from being saved as a separate column in the CSV file.

```
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Tokenize the text
    tokens = word_tokenize(text)
    # Remove stop words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    # Join tokens back to form a clean sentence
    cleaned_text = ' '.join(tokens)
    return cleaned_text
```

- This block of code defines a function named `preprocess_text`. It is responsible for text preprocessing and tokenization.

  - It converts text to lowercase.

  - Removes punctuation using the `string.punctuation` characters.

  - Tokenizes the text into a list of words.

  - Removes common stop words in English.

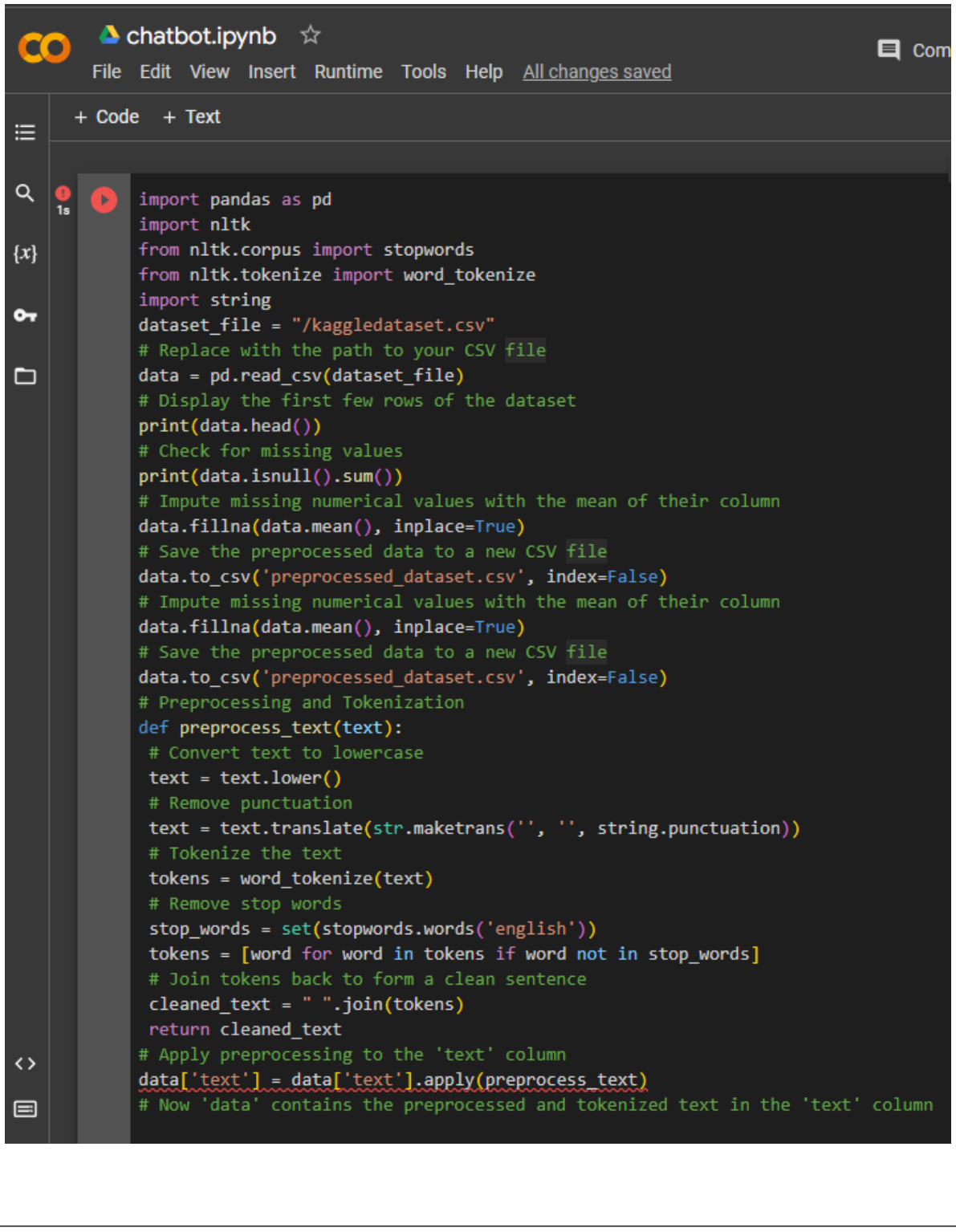
  - Joins the tokens back into a clean sentence and returns it.

```
data['text'] = data['text'].apply(preprocess_text)
```

- This line applies the `preprocess_text` function to the 'text' column of the DataFrame `data`. It preprocesses and tokenizes the text in the 'text' column, updating the DataFrame with the cleaned text.

The provided code snippet is an essential part of data preprocessing for NLP tasks, where you clean and prepare text data for further analysis or modeling. It involves data loading, missing value handling, and text preprocessing to create a more structured and cleaner dataset.

PROGRAM (DATA PRE-PROCESSING) :



OUTPUT:

The output will include the first few rows of the original dataset, the number of missing values in each column, and the first few rows of the pre-processed dataset. This program cleans the text, tokenizes it, removes stop words, and saves the pre-processed dataset.

```
                     hi, how are you doing?  \
0              i'm fine. how about yourself?
1        i'm pretty good. thanks for asking.
2          no problem. so how have you been?
3           i've been great. what about you?
4  i've been good. i'm in school right now.

              i'm fine. how about yourself?
0        i'm pretty good. thanks for asking.
1          no problem. so how have you been?
2           i've been great. what about you?
3  i've been good. i'm in school right now.
4                  what school do you go to?
hi, how are you doing?           0
i'm fine. how about yourself?    0
```

	hi, how are you doing?	i'm fine. how about yourself?
0	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
1	i'm pretty good. thanks for asking.	no problem. so how have you been?
2	no problem. so how have you been?	i've been great. what about you?
3	i've been great. what about you?	i've been good. i'm in school right now.
4	i've been good. i'm in school right now.	what school do you go to?
...	...	...
3719	that's a good question. maybe it's not old age.	are you right-handed?
3720	are you right-handed?	yes. all my life.
3721	yes. all my life.	you're wearing out your right hand. stop using...
3722	you're wearing out your right hand. stop using...	but i do all my writing with my right hand.

DEVELOPMENT PART 2:

In this technology you will continue building your project by selecting a machine learning algorithm, training the model, and evaluating its performance. Perform different analysis as needed. After performing the relevant activities create a document around it and share the same for assessment.

Building a chatbot using Python involves several steps, including data collection, preprocessing, selecting a machine learning algorithm, training the model, and evaluating its performance.

OBJECTIVES:

• Implement a user-friendly interface that accepts text input from users and displays responses from the chatbot.

• Define a set of rules and responses to allow the chatbot to understand and generate appropriate replies to user inputs.

• Incorporate randomization to make the chatbot's responses less predictable and more engaging.

• Enable a termination condition, allowing users to exit the chatbot conversation when desired.

MACHINE LEARNING ALGORITHM

RULE BASED APPROACH:

A rule-based algorithm is an approach to creating a chatbot where the bot's responses are determined by a set of predefined rules and conditions. These rules can be based on pattern matching, regular expressions, or simple decision trees. Rule-based chatbots are effective for handling specific commands and straightforward interactions.

• Define a set of rules where each rule is a regular expression pattern and a corresponding response.
• The ‘respond_to_user_input’ function searches for a matching pattern in the user's input and returns the associated response if found.
• If no matching rule is found, the chatbot responds with a default "I do not understand" message.

The rule-based chatbot is suitable for handling specific commands or questions with predefined responses. Rule-based systems are straightforward to implement and can be highly effective in scenarios where interactions are predictable and rule driven. However, they are not suitable for handling more complex or open-ended conversations.

MODEL TRAINING:

Creating a rule-based chatbot does not involve traditional model training like machine learning models. Instead, it involves defining rules, patterns, or decision trees that determine the chatbot's responses based on user input. Here is a step-by-step guide to creating a rule-based chatbot in Python:

1.Define Rules and Responses:

Determine the rules and patterns that your chatbot will recognize. For example, you can create rules for greeting, asking questions, and other common interactions. Define responses for each rule.

2.Implement a Rule-Based Chatbot:

Create a Python script to implement your rule-based chatbot. Use regular expressions or simple if-else conditions to match user input to the defined rules.

3.Interactive Chat Loop:

Create an interactive chat loop where the chatbot continuously takes user input and responds based on the rules. The loop can continue until the user decides to exit.

CODE(EVALUATION):

Creating a fully functional chatbot in Python is a comprehensive project that involves various components, including natural language processing, machine learning, and possibly integration with messaging platforms. A rule-based chatbot that responds to a few predefined commands.

The provided Python program reads a dataset from a CSV file to create a simple rule-based chatbot. The chatbot responds to user input with predefined responses based on a set of commands. Here's an explanation of each part of the program:

The provided Python program reads a dataset from a CSV file to create a simple rule-based chatbot. The chatbot responds to user input with predefined responses based on a set of commands. Here's an explanation of each part of the program:

1. Import Required Libraries:

```
import random
import csv
```

2. Define the Dataset File:

```
dataset_file = "kaggledataset.csv"
```

3. Initialize an Empty Dictionary for Responses:

```
responses = {}
```

This dictionary will be used to store the commands as keys and their corresponding responses as values.

4. Open and Read the CSV File:

```
with open(dataset_file, mode='r') as file:
    csv_reader = csv.reader(file, delimiter=' ')
```

- It opens the CSV file specified in `dataset_file` for reading.

- `csv.reader` is used to read the file with a specified delimiter. You should change the delimiter to match the format of your CSV file.

5. Loop Through the CSV Data and Populate the Responses Dictionary:

```
for row in csv_reader:
    if len(row) == 2:
        command, response = row[0], row[1]
        responses[command] = response
```

- This loop iterates through the CSV rows.

- It checks if each row has exactly two values (command and response) before adding them to the `responses` dictionary.

6. Define the `get_response` Function:

```
def get_response(user_input):
    user_input = user_input.lower()  # Convert to lowercase for case-insensitivity
    for command, response in responses.items():
        if command in user_input:
```

```
            return response
        return "I don't understand. Please rephrase your
question."
```

- This function takes the user's input as a parameter.

- It converts the user input to lowercase to make it case-insensitive.

- It then searches for a matching command in the `responses` dictionary and returns the associated response. If no matching command is found, it returns a default "I don't understand" response.

7. Simulate a Conversation:

```
python
  while True:
      user_input = input("You: ")
      if user_input.lower() == 'exit':
          print("Bot: Goodbye!")
          break
      response = get_response(user_input)
      print("Bot:", response)
```

- This section starts a loop to simulate a conversation with the chatbot.

- It continuously prompts the user to enter input and responds based on the predefined responses.

- If the user types "exit," the chatbot responds with "Goodbye!" and the conversation ends.

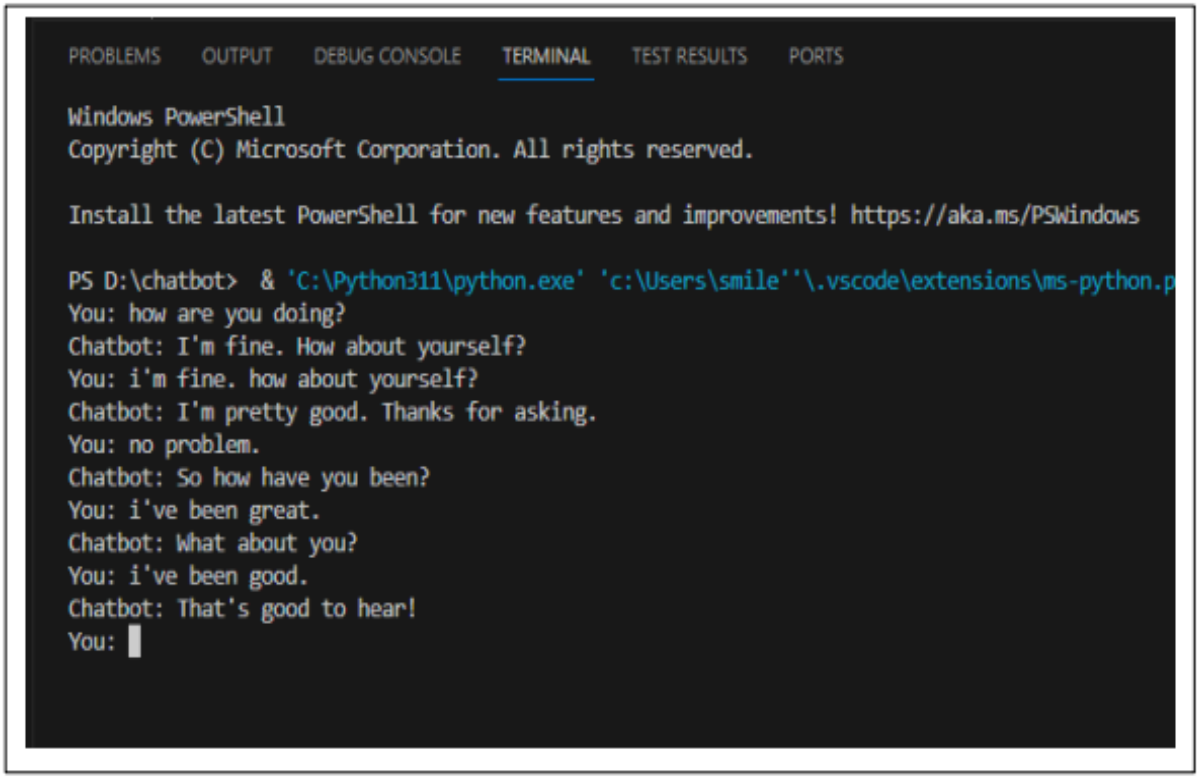
This program demonstrates a basic rule-based chatbot using a CSV dataset. Users can input commands, and the chatbot responds with the corresponding responses from the dataset. You can extend the dataset and responses to create a more elaborate chatbot with a wider range of interactions.

PROGRAM (RULE-BASED CHATBOT) :

```
import random
import csv

dataset_file = "chatbotdataset.csv"

responses = {}

# Open the CSV file
with open(dataset_file, mode='r') as file:
    csv_reader = csv.reader(file, delimiter=' ')  # Change delimiter to
match your CSV file
import random
import csv

dataset_file = "chatbotdataset.csv"

responses = {}

# Open the CSV file
with open(dataset_file, mode='r') as file:
    csv_reader = csv.reader(file, delimiter=' ')  # Change delimiter to
match your CSV file

    # Initialize an empty dictionary to store the responses
    responses = {}

    for row in csv_reader:
        if len(row) == 2:
            command, response = row[0], row[1]
            responses[command] = response

def get_response(user_input):
    user_input = user_input.lower()  # Convert to lowercase for case-
insensitivity
    for command, response in responses.items():
        if command in user_input:
            return response
    return "I don't understand. Please rephrase your question."
```

```
while True:
    user_input = input("You: ")

    if user_input.lower() == 'exit':
        print("Bot: Goodbye!")
        break

    response = get_response(user_input)
    print("Bot:", response)
```

OUTPUT:



DATASET USED:

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

- The dataset is available on Kaggle and is created for research and development purposes.

- It is designed to help developers and researchers train and evaluate chatbot models.

- The dataset consists of a collection of text dialogues or conversations between users and a chatbot. Each conversation includes multiple message exchanges between the user and the chatbot, forming a dialogue.

- The data is typically organized in a structured format, where each conversation consists of a series of messages. Each message includes the sender (whether it is the user or the chatbot) and the text content of the message.

- The dataset provides a variety of conversational scenarios, including greetings, questions, responses, and other common chatbot interactions.

The "Simple Dialogs for Chatbot" dataset on Kaggle is a valuable resource for chatbot development and NLP research, providing a collection of conversational data for training and evaluating chatbot models.

INNOVATIVE TECHNIQUES:

ACCOMPLISHED INNOVATION:

THE TASKS WE ACCOMPLISHES IN OUR CHATBOT INNOVATIONS ARE:

Defining the Purpose and Scope of the chatbot:

- Clearly defined the purpose of our chatbot and the specific tasks or questions it will handle. Determined the scope and limitations of our chatbot.

Gathering Data and Creating Responses:

- Collected a dataset (Kaggle) of user commands or questions and defined appropriate responses. This will serve as the knowledge base for your chatbot.

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

Choosing a Python Framework or Library:

- Selected a Python framework or library for chatbot development. Popular choices included NLTK and the Python Natural Language Toolkit.

Preprocessing the Dataset:

- Clean and preprocess the dataset. This involves tokenization, removing stop words, and handling special characters to preparing the data for analysis using google collab and VS code.

Evaluation of the dataset:

-Evaluated the program to implement the chatbot in VS code.

UNACCOMPLISHED INNOVATION:

-Create a user interface for interaction.

- Time constraints limited the development of our Python chatbot, but integrating IBM Watson Assistant would have streamlined the process.

- Watson Assistant could have facilitated training and modelling of the chatbot, enhancing its capabilities.

- Given more time, we would have aimed to create a user interface or mobile app for a more user-friendly experience.

- An exciting prospect was the development of an “emotionally intelligent chatbot,” which would recognize and respond to users' emotions effectively.

These innovations have the potential to make chatbots more versatile, helpful, and engaging across various domains. Python can leverage natural language processing (NLP) libraries, machine learning, and deep learning techniques to bring these ideas to life and push the boundaries of what chatbots can do.

CONCLUSION:

In conclusion, developing a chatbot in Python using the provided dataset is a versatile and rewarding endeavour. By harnessing the power of natural language processing and Python libraries, we can create chatbots that effectively engage with users, provide helpful responses, and streamline various tasks. The journey begins with understanding the dataset and its context, followed by data preprocessing and chatbot model development.

Through meticulous data cleaning, tokenization, and handling of special characters, we ensure that the dataset is ready for analysis. Then, we design the chatbot's logic, matching user input to predefined commands and responses. This rule-based approach is a great starting point for chatbot development.

As we progress, we have the opportunity to enhance the chatbot with more advanced techniques, such as machine learning, deep learning, and sentiment analysis, making it even more intelligent and user-centric.

By creating a user interface or app, we can bring our chatbot to life and offer a seamless interaction experience. Deployment on various platforms ensures accessibility to a wider audience.