***TITLE: CREATE A CHATBOT USING*** ***PYTHON***

PROJECT OVERVIEW:

Creating a chatbot using Python can be a fascinating project. Here's an overview of the general steps involved in creating a simple chatbot using Python:

Understanding the Objective:

Define the purpose of your chatbot. Determine the target audience and the specific problems or questions the chatbot will address.

Data Collection:

Gather data relevant to the chatbot's purpose, such as FAQs, previous conversations, or any other relevant text data.

Data Preprocessing:

Clean and preprocess the collected data, including tasks like tokenization, lemmatization, and removing unnecessary characters.

Training Data Preparation:

Prepare the data for training the chatbot, ensuring it is in a suitable format for the chosen machine learning or NLP model.

Choose a Framework or Library:

Select a suitable Python library or framework for creating the chatbot. Popular choices include NLTK, TensorFlow, PyTorch, or spaCy.

Developing the Chatbot Logic:

Implement the core logic of the chatbot, including handling user inputs, understanding and processing the inputs, and generating appropriate responses.

Natural Language Processing (NLP):

Utilize NLP techniques such as tokenization, part-of-speech tagging, and sentiment analysis to enhance the chatbot's understanding of user inputs.

Machine Learning Model Integration:

Integrate machine learning models, such as text classification or sequence-to-sequence models, to improve the chatbot's ability to generate accurate and contextually relevant responses.

Testing and Evaluation:

Test the chatbot with various user inputs to ensure it provides accurate and appropriate responses. Evaluate its performance and refine the model as needed.

Deployment:

Deploy the chatbot to a suitable platform for user interaction, such as a website, messaging platform, or mobile application.

Continuous Improvement:

Collect user feedback and data to continuously improve the chatbot's performance and expand its capabilities over time.

This is a high-level overview of the process. Creating a fully functional and sophisticated chatbot involves deeper knowledge of natural language processing, machine learning, and programming in Python

PROJECT DESCRIPTION:

The goal of this project is to develop a conversational chatbot using Python that can interact with users, provide helpful information, and respond to user queries in a natural and intuitive manner. The chatbot will be built to simulate human-like conversations and provide accurate and contextually relevant responses to user inputs.

Key Features:

1. Natural Language Processing (NLP): Implement NLP techniques to preprocess and understand user inputs, including tokenization, lemmatization, and part-of-speech tagging.

2. Machine Learning Integration: Integrate machine learning models to improve the chatbot's ability to comprehend user queries and generate appropriate responses based on the context.

3. Contextual Understanding: Develop the chatbot's ability to understand and maintain context across multiple user interactions to provide more engaging and relevant responses.

4. User Interaction: Implement a user-friendly interface for users to interact with the chatbot, allowing seamless communication and easy access to information.

5. Integration with External APIs: Enable the chatbot to access external APIs for retrieving real-time information and delivering dynamic and up-to-date responses to user queries.

Technical Details:

1. Programming Language: Python

2. Libraries/Frameworks: NLTK, TensorFlow, or PyTorch for NLP and machine learning integration.

3. Data Collection: Gather relevant datasets or text data to train and improve the chatbot's conversational abilities.

4. Data Preprocessing: Clean and preprocess the collected data for effective training and model building.

5. Model Training: Train the chatbot model using suitable machine learning algorithms and techniques to enhance its conversational capabilities.

6. User Interface: Develop an intuitive and user-friendly interface for users to interact with the chatbot seamlessly.

7. Deployment: Deploy the chatbot on a suitable platform, such as a website, messaging platform, or mobile application, to make it accessible to users.

Project Deliverables:

1. Python scripts for data preprocessing, model training, and chatbot implementation.

2. User interface for seamless interaction with the chatbot.

3. Documentation detailing the project's architecture, implementation details, and instructions for future development and improvements.

4. A working prototype of the chatbot that demonstrates its conversational abilities and user interaction capabilities.

Conclusion:

Through this project, we aim to build an intelligent and user-friendly chatbot using Python that can effectively communicate with users, provide accurate information, and enhance user experience through natural and engaging conversations.

STEPS TO CREATE CHATBOT IN PYTHON:

1. Install Python: If you haven't already, install Python on your system. You can download the latest version from the official Python website.

2. Choose a Framework or Library: Select a suitable library for natural language processing (NLP) tasks. Common choices include NLTK, TensorFlow, PyTorch, or spaCy.

3. Data Collection and Preprocessing:

- Collect relevant data or create a dataset for training the chatbot.

- Preprocess the data by cleaning, tokenizing, and removing unnecessary information.

4. Build the Model:

- Define the structure of your chatbot, including the data processing pipeline.

- Implement algorithms for text processing, including tokenization, lemmatization, and part-of-speech tagging.

5. Train the Model:

- Use the collected and preprocessed data to train the chatbot model.

- Fine-tune the model parameters for improved performance and accuracy.

6. Define Responses:

- Create a set of responses for the chatbot to provide to user inputs.

- Implement logic to generate appropriate and contextually relevant responses based on user queries.

7. Integrate NLP and ML Techniques:

- Implement natural language processing techniques to enhance the chatbot's understanding of user inputs and generate appropriate responses.

- Integrate machine learning models to improve the chatbot's conversational abilities and response accuracy.

8. Create User Interaction:

- Develop a user-friendly interface for users to interact with the chatbot

- Enable the chatbot to receive user inputs and provide suitable responses in a conversational format.

9. Testing and Debugging:

- Test the chatbot with various user inputs to ensure it responds accurately and appropriately.

- Debug and refine the chatbot's logic and responses to improve its performance.

10. Deployment:

- Deploy the chatbot on a suitable platform for user interaction, such as a website, messaging platform, or mobile application.

- Ensure the deployed chatbot is accessible and responsive to user queries.

11. Maintenance and Upgrades:

- Collect user feedback and data to continuously improve the chatbot's performance and enhance its capabilities.

- Implement regular updates and upgrades to incorporate new features and improve the chatbot's conversational abilities.

Following these steps will help you create a basic chatbot using Python. For more advanced functionalities, consider exploring additional NLP techniques and machine learning algorithms.

PROGRAM CODE:

import nltk

import numpy as np

import random

import string

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

nltk.download('punkt')

nltk.download('wordnet')

f=open('chatbot.txt','r',errors = 'ignore')

raw=f.read()

raw=raw.lower()# converts to lowercase

sent\_tokens = nltk.sent\_tokenize(raw)# converts to list of sentences

word\_tokens = nltk.word\_tokenize(raw)# converts to list of words

lemmer = nltk.stem.WordNetLemmatizer()

#WordNet is a semantically-oriented dictionary of English included in NLTK.

def LemTokens(tokens):

return [lemmer.lemmatize(token) for token in tokens]

remove\_punct\_dict = dict((ord(punct), None) for punct in string.punctuation)

def LemNormalize(text):

return LemTokens(nltk.word\_tokenize(text.lower().translate(remove\_punct\_dict)))

GREETING\_INPUTS = ("hello", "hi", "greetings", "sup", "what's up", "hey")

GREETING\_RESPONSES = ["hi", "hey", "\*nods\*", "hi there", "hello", "I am glad you are talking to me"]

def greeting(sentence):

for word in sentence.split():

if word.lower() in GREETING\_INPUTS:

return random.choice(GREETING\_RESPONSES)

from sklearn.feature\_extraction.text import TfidfVectorizer

def response(user\_response):

robo\_response=''

sent\_tokens.append(user\_response)

TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop\_words='english')

tfidf = TfidfVec.fit\_transform(sent\_tokens)

vals = cosine\_similarity(tfidf[-1], tfidf)

idx=vals.argsort()[0][-2]

flat = vals.flatten()

flat.sort()

req\_tfidf = flat[-2]

if(req\_tfidf==0):

robo\_response=robo\_response+"I am sorry! I don't understand you"

return robo\_response

else:

robo\_response = robo\_response+sent\_tokens[idx]

return robo\_response

flag=True

print("ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!")

while(flag==True):

user\_response = input()

user\_response=user\_response.lower()

if(user\_response!='bye'):

if(user\_response=='thanks' or user\_response=='thank you' ):

flag=False

print("ROBO: You are welcome..")

else:

if(greeting(user\_response)!=None):

print("ROBO: "+greeting(user\_response))

else:

print("ROBO: ",end="")

print(response(user\_response))

sent\_tokens.remove(user\_response)

else:

flag=False

print("ROBO: Bye! take care..")

PROGRAM:

main.py:

import json

import re

import random\_responses

# Load JSON data

def load\_json(file):

with open(file) as bot\_responses:

print(f"Loaded '{file}' successfully!")

return json.load(bot\_responses)

# Store JSON data

response\_data = load\_json("bot.json")

def get\_response(input\_string):

split\_message = re.split(r'\s+|[,;?!.-]\s\*', input\_string.lower())

score\_list = []

# Check all the responses

for response in response\_data:

response\_score = 0

required\_score = 0

required\_words = response["required\_words"]

# Check if there are any required words

if required\_words:

for word in split\_message:

if word in required\_words:

required\_score += 1

# Amount of required words should match the required score

if required\_score == len(required\_words):

# print(required\_score == len(required\_words))

# Check each word the user has typed

for word in split\_message:

# If the word is in the response, add to the score

if word in response["user\_input"]:

response\_score += 1

# Add score to list

score\_list.append(response\_score)

# Debugging: Find the best phrase

# print(response\_score, response["user\_input"])

# Find the best response and return it if they're not all 0

best\_response = max(score\_list)

response\_index = score\_list.index(best\_response)

# Check if input is empty

if input\_string == "":

return "Please type something so we can chat :("

# If there is no good response, return a random one.

if best\_response != 0:

return response\_data[response\_index]["bot\_response"]

return random\_responses.random\_string()

while True:

user\_input = input("You: ")

print("Bot:", get\_response(user\_input))

bot.json:

[

{

"response\_type": "greeting",

"user\_input": ["hello", "hi", "hey"],

"bot\_response": "Hey there!",

"required\_words": []

},

{

"response\_type": "greeting",

"user\_input": ["see you", "goodbye", "bye"],

"bot\_response": "See you later!",

"required\_words": []

},

{

"response\_type": "greeting",

"user\_input": ["nice", "to", "meet", "you"],

"bot\_response": "The pleasure is all mine!",

"required\_words": ["nice", "meet", "you"]

},

{

"response\_type": "question",

"user\_input": ["how", "to", "learn", "code", "coding", "apps"],

"bot\_response": "Start by typing: 'How to learn coding' on Google.",

"required\_words": ["learn", "code"]

},

{

"response\_type": "question",

"user\_input": ["refund", "how", "can", "I", "get"],

"bot\_response": "We don't offer refunds for free education.",

"required\_words": ["refund", "I"]

"required\_words": ["refund", "i"]

},

{

"response\_type": "question",

"user\_input": ["how", "are", "you"],

"bot\_response": "I'm great! Thanks for asking.",

"required\_words": ["how", "are", "you"]

}

]

Random\_responses.py:

import random

def random\_string():

random\_list = [

"Please try writing something more descriptive.",

"Oh! It appears you wrote something I don't understand yet",

"Do you mind trying to rephrase that?",

"I'm terribly sorry, I didn't quite catch that.",

"I can't answer that yet, please try asking something else."

]

list\_count = len(random\_list)

random\_item = random.randrange(list\_count)

return random\_list[random\_item]

OUTPUT:

Loaded ‘bot.json’ successfully!

You : How are you?

Bot : I’m great! Thanks for asking.

You : I want to have a refund please!

Bot : We don’t offer refunds for free education