## CREATE CHATBOT WITH PYTHON

## A PROJECT REPORT

## Submitted by

## 

## 

## VITHYA D - III AI&DS (610521243048)

## 

## JEEVITHA M -III AI&DS (610521243017)

## 

## MEHALA K -III AI&DS (610521243025)

## 

## RAMYA K -III AI&DS (610521243034)

## 

## 

## 

## In partial fulfillment for the award of the degree

## Of

## BACHELOR OF ENGINEERING

## In

## ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

## DHIRAJLAL GANDHI COLLEGE OF TECHNOLOGY

## SALEM – 636 309

## annau logo.png

## ANNA UNIVERSITY::CHENNAI - 600 025

**Table of Content**

**1.Abstract**

**2. Problem definition**

**3.Introduction**

**4.Design thinking approach**

**5.Innovative Design**

* **Architecture Model diagram**
* **Use case model diagram**
* **Activity model diagram**
* **Class Diagram**

**6.Future Innovation Thinking**

**7.Dataset**

**8.Datapreprocessing**

**9.Building the model**

**10.Integrating chatbot with flask**

**11.Source code**

**12.Conclusion**

**ABSTRACT**

**TITLE-CREATE A CHATBOT USING PYTHON**

* In the current digital era, chatbots have become effective instruments for enhancing user interaction and offering prompt customer service.
* The design, development, and deployment of an intelligent chatbot with natural language processing and comprehension capabilities are the main objectives of this study.
* Improving user interaction on several platforms including websites and mobile application is the goal.
* Modern machine learning and natural language processing technologies form the foundation of the chatbot.
* Deep learning algorithms are integrated to understand user inquiries, discern intents, and produce relevant responses.
* Over time, the system's accuracy and responsiveness are enhanced by continual learning methods and a large dataset.
* The chatbot also has sentiment analysis capabilities, which allow it to determine the emotions of users and modify responses accordingly.
* Advanced dialog management techniques are used during the development phase to guarantee smooth and contextually relevant discussions.
* The user interface of the chatbot has been built with ease of use and intuitiveness in mind, catering to a wide range of technological backgrounds.
* Additionally, the backend design of the system is flexible and scalable, making it simple to integrate with a wide range of platforms and applications.

**Problem Definition**

When using an app or website, customers expect outstanding service. They can become disinterested in the app if they can't locate the solution to a question they have. To avoid losing customers and having an adverse effect on your bottom line, you must provide the highest quality service possible while developing a website or application.

**Introduction:**

The way that people and businesses use technology has been changed by chatbots. These conversational agents powered by artificial intelligence are becoming more common in our digital environment and have a variety of us and advantages. We'll examine chatbots in this introduction, their significance, and the potential for change they can bring to a variety of fields and daily life.

**Design thinking approach:**

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.

**Functionality**

Explicitly state the chatbot's main goal. Customer service, information retrieval, or work automation are a few examples. List the precise goals that the chatbot should accomplish. If it's a chatbot for customer service, for instance, goals can include resolving user difficulties, delivering product information, and gathering feedback. Determine the kinds of inquiries or questions the chatbot should be able to answer. These could consist of solutions to problems, suggestions, or frequently asked questions. Indicate the domains or specialties the chatbot should support.

**User Interface**

**Determine the Integration Platform**

Website: Integrating a chatbot into your website is a common choice. It can provide instant assistance, answer frequently asked questions, and guide users through the website's content or services.

App: If you have a mobile app, integrating a chatbot can enhance user engagement, provide personalized recommendations, and offer support within the app's ecosystem.

**Define the Chatbot's Purpose**

Clearly define the primary purpose of the chatbot. Is it for customer support, lead generation, information retrieval, or something else? Understanding its purpose is crucial for designing a user-friendly interface**.**

**User-Centered Design**

Start with user research to understand your target audience, their needs, and preferences. This will inform your design decisions. Create user personas to represent different user types and their goals when interacting with the chatbot.

**Natural Language Processing (NLP)**

**Choose an NLP Framework or Library**

Begin by selecting an NLP framework or library that fits your programming language and platform

Python: NLTK, spaCy, and Transformers (Hugging Face).

**Preprocess User Input**

Before applying NLP techniques, preprocess the user's input to clean and standardize it. Common preprocessing steps include:

Tokenization: Split the input into words or tokens.

Lowercasing: Convert all text to lowercase for uniformity.

Removing punctuation and special characters.

Handling contractions and stemming (if necessary).

**Entity Recognition**

Implement entity recognition to identify and extract specific information from user input. You can use Named Entity Recognition (NER) models or libraries like spaCy for this purpose. Entities might include names, dates, locations, or product names.

**Intent Recognition**

Recognize the user's intent or what they want to achieve. This is often done using techniques like:

Rule-based intent recognition: Define a set of rules or patterns to map user input to specific intents.

Machine learning-based intent recognition: Train a model (e.g., using a supervised learning approach) to classify user input into predefined intents.

**Dialog Management:**

Implement a dialog management system to maintain context and flow of the conversation. This can be achieved using a state machine, where the current state represents the user's progress in the conversation.

**Sentiment Analysis**

Analyze the sentiment of the user's input to gauge their mood or emotions. This can help tailor responses accordingly. You can use pre-trained sentiment analysis models for this purpose.

**Responses**

**Define User Scenarios**

Identify the most common scenarios or user intents that your chatbot will encounter. For example, if your chatbot is for an e-commerce site, common scenarios might include product inquiries, order tracking, and account management**.**

**Response Types**

Determine the types of responses your chatbot will provide. These can include:

Accurate Information: Provide accurate answers to factual questions, such product details, company information, or operating hours.

Suggestions: Offer recommendations based on user queries, preferences, or historical data. For example, suggest products, articles, or services.

Assistance: Help users perform specific tasks or guide them through processes, such as making a reservation or troubleshooting an issue**.**

**Predefined Responses:**

Create predefined responses for common user scenarios. These responses should be well-crafted, concise, and clear. Use a mix of text, images, and links as appropriate.

**Dynamic Responses:**

Develop dynamic responses for scenarios that require real-time or personalized information. This might involve querying databases or external APIs for up-to-date data. For example, displaying the current stock availability of a product.

**Integration**

**Choose Integration Type**

Decide whether the chatbot will be fully integrated or exist as a separate module. Consider the following options:

Fully Integrated: The chatbot is seamlessly integrated into the website or app's interface and interacts directly with users

Separate Module: The chatbot is a separate component, such as a chat window or pop-up, that users can access for specific purposes.

**Platform Compatibility**

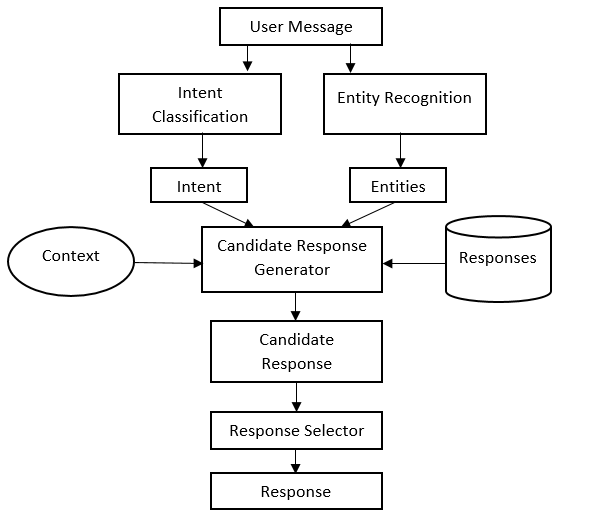
Ensure that the integration method you choose is compatible with the platform (website or app) on which the chatbot will operate. Different platforms may require different approaches**.**

**Testing and Improvement**

Making sure a chatbot is effective requires constant testing and improvement. Through user interactions, we may continuously evaluate its performance in order to pinpoint problem areas and expand its functionalities. We can improve the chatbot's responses through this iterative approach, boost accuracy, and respond to changing user needs, making it a more useful and user-friendly tool. In this cycle of continuous improvement, user feedback is essential because it enables us to address problems, improve dialog flows, and provide users with ever-more rewarding conversational experiences.

**Innovation design**

**Architecture Model Diagram**

****

**Use Case Model Diagram**

Database

Chatbot

Message channel

User

**Activity Model Diagram**

|  |
| --- |
| User Chatbot Information |

Start

if query==Thank

Enter Question

Want to ask question

If string matches

Return response

Fetch response

response

Query

No

Yes

No

[Cite your source here.]

Return byte

[Cite your source here.]

Return sorry don’t known

Yes

Return response

t

stop

[Cite your source here.]

**CLASS DIAGRAM**

+getuser query:string

+out response()

+getquery

+train model()

+string model()

+string match()

+response()

**INFORMATION**

**CHATBOT**

+user query:string

+Response:String

**USER**

**FUTURE INNOVATION THINKING**

**Selecting the Right Pre-trained Model**

Choose a pre-trained language model that fits the requirements of your application. Models like GPT-3, BERT, or RoBERT a offer different capabilities, so understanding your specific needs is crucial.

**Fine-tuning for Specific Tasks**

Fine-tune the chosen pre-trained model on a dataset relevant to your application. Fine-tuning helps the model to adapt to the specific context and domain, making it more proficient in generating accurate and contextually relevant responses.

**Understanding User Intent**

Utilize the pre-trained model to understand the intent behind the user's query. This can involve using techniques such as intent recognition to identify what the user is asking or trying to achieve**.**

**Generating Responses**

Use the pre-trained model to generate responses based on the identified user intent. Pre-trained models excel at understanding the context of the conversation and can generate responses that are coherent, contextually appropriate, and grammatically correct

**Handling Multiturn Conversations**

Utilize the pre-trained model's ability to maintain context over multiple turns in a conversation. This ensures that the chatbot's responses are consistent and relevant to the ongoing dialogue

**Incorporating Knowledge Base**

Integrate a knowledge base with the pre-trained model. This could be a database of frequently asked questions, product information, or any relevant data. The chatbot can use the pre-trained model to understand user queries and then refer to the knowledge base for accurate and up-to-date information**.**

**Contextual Understanding**

Leverage the contextual understanding of the pre-trained model. It can understand the nuances of human language, including idiomatic expressions, sarcasm, and ambiguous queries, leading to more accurate responses.

**Error Handling and Clarifications**

Use the pre-trained model to identify ambiguous queries or errors in user input. The chatbot can then ask clarifying questions to ensure it understands the user's intent correctly before generating a response.

**Personalization**

Utilize the pre-trained model to analyse user data (within ethical boundaries and respecting privacy) to personalize responses. Personalization can enhance user engagement by tailoring responses to individual preferences and behaviours.

**Continuous Monitoring and Improvement**

Continuously monitor the chatbot's interactions and user feedback. Use this data to iteratively improve the pre-trained model and the chatbot's responses. Regular updates and re-training ensure that the chatbot stays relevant and accurate over time.

**IMPORT LIBRARIES**

We need to import all the libraries which required for the project

**import tensorflow as tf**

**from sklearn.model\_selection import train\_test\_split**

**import unicodedata**

**import re**

**import numpy as np**

**import warnings**

**warnings.filterwarnings('ignore')**

**Read the dataset**

Depending on the format of your dataset, use appropriate libraries or functions in Python (such as pandas for CSV or Excel files) to load the data into your programming environment.

**DATA LINK :** www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

**df=open('dialogs1.txt','r').read()**

**print(df)**

**SAMPLE OUTPUT**

hi, how are you doing? i'm fine. how about yourself?

i'm fine. how about yourself? i'm pretty good. thanks for asking.

i'm pretty good. thanks for asking. no problem. so how have you been?

no problem. so how have you been? i've been great. what about you?

i've been great. what about you? i've been good. i'm in school right now.

i've been good. i'm in school right now. what school do you go to?

what school do you go to? i go to pcc.

i go to pcc. do you like it there?

do you like it there? it's okay. it's a really big campus.

it's okay. it's a really big campus. good luck with school.

good luck with school. thank you very much.

how's it going? i'm doing well. how about you?

i'm doing well. how about you? never better, thanks.

never better, thanks. so how have you been lately?

so how have you been lately? i've actually been pretty good. you?

i've actually been pretty good. you? i'm actually in school right now.

i'm actually in school right now. which school do you attend?

which school do you attend? i'm attending pcc right now.

i'm attending pcc right now. are you enjoying it there?

are you enjoying it there? it's not bad. there are a lot of people there.

it's not bad. there are a lot of people there. good luck with that.

good luck with that. thanks.

how are you doing today? i'm doing great. what about you?

i'm doing great. what about you? i'm absolutely lovely, thank you.

i'm absolutely lovely, thank you. everything's been good with you?

everything's been good with you? i haven't been better. how about yourself?

i haven't been better. how about yourself? i started school recently.

i started school recently. where are you going to school?

where are you going to school? i'm going to pcc.

i'm going to pcc. how do you like it so far?

how do you like it so far? i like it so far. my classes are pretty good right now.

i like it so far. my classes are pretty good right now. i wish you luck.

it's an ugly day today. i know. i think it may rain.

i know. i think it may rain. it's the middle of summer, it shouldn't rain today.

it's the middle of summer, it shouldn't rain today. that would be weird.

that would be weird. yeah, especially since it's ninety degrees outside.

yeah, especially since it's ninety degrees outside. i know, it would be horrible if it rained and it was hot outside.

i know, it would be horrible if it rained and it was hot outside. yes, it would be.

yes, it would be. i really wish it wasn't so hot every day………….

**Split the dataset as question and answer**

We need to split the dataset into question and answer for easy processing

**QA\_list=[QA.split('\t') for QA in data.split('\n')]**

**print(QA\_list[:5])**

**questions=[row[0] for row in QA\_list]**

**answers=[row[1] for row in QA\_list]**

**print(questions[:])**

**print(answers[:])**

**Data preprocessing**

**Lowercasing:**

Convert all text to lowercase. This ensures that the chatbot treats "hello," "Hello," and "HELLO" as the same word.

**Tokenization:**

Split text into smaller units, such as words or subwords. Tokenization helps the chatbot understand the structure of sentences, making it easier to process and respond to user input.

**Removing Special Characters and Punctuation:**

Remove unnecessary characters like punctuation marks, special symbols, or emojis. This simplifies the text and prevents the chatbot from misinterpreting special characters.

**import nltk**

**from nltk.tokenize import word\_tokenize**

**from nltk.corpus import stopwords**

**import string**

**nltk.download('punkt')**

**nltk.download('stopwords')**

**def normalize\_text(text):**

**# Convert to lowercase**

**text = text.lower()**

**# Tokenize the text**

**tokens = word\_tokenize(text)**

**sentences = nltk.sent\_tokenize(text)**

**# Remove punctuation and special characters**

**tokens = [word for word in tokens if word.isalnum()]**

**Removing Stop Words:**

Remove common words like "and," "the," "is," etc., which don’t contribute significantly to the meaning of the text. Removing stop words can reduce the dimensionality of the data and improve processing efficiency

**# Remove stopwords**

**stop\_words = set(stopwords.words('english'))**

**tokens = [word for word in tokens if word not in stop\_words]**

**# Join the tokens back into a normalized string**

**normalized\_text = ' '.join(tokens)**

**return normalized\_text**

**# Example usage**

**input\_text = open('/content/dialogs1.txt','r').read()**

**normalized\_text = normalize\_text(input\_text)**

**print("Normalized Text:", normalized\_text)**

**OUTPUT**

**Normalized Text:** hi fine fine pretty good thanks asking pretty good thanks asking problem problem great great good school right good school right school go school go go pcc go pcc like like okay really big campus okay really big campus good luck school good luck school thank much going well well never better thanks never better thanks lately lately actually pretty good actually pretty good actually school right actually school right school attend school attend attending pcc right attending pcc right enjoying enjoying bad lot people bad lot people good luck good luck thanks today great great absolutely lovely thank absolutely lovely thank everything good everything good better better started school recently started school recently going school going school going pcc going pcc like ……..

**Stemming and Lemmatization:**

Reduce words to their root form. Stemming and lemmatization help in reducing words to their base or root form (e.g., "running" to "run"). This process ensures that different forms of the same word are treated as identical, enhancing the chatbot's understanding.

**import nltk**

**from nltk.stem import PorterStemmer**

**from nltk.stem import WordNetLemmatizer**

**# Download NLTK resources (only needed once)**

**nltk.download('punkt')**

**nltk.download('wordnet')**

**# Example text**

**text = open('/content/dialogs1.txt','r').read()**

**# Tokenize the text**

**words = nltk.word\_tokenize(text)**

**# Perform Stemming**

**stemmer = PorterStemmer()**

**stemmed\_words = [stemmer.stem(word) for word in words]**

**print("Stemmed Words:")**

**print(stemmed\_words)**

**# Perform Lemmatization**

**lemmatizer = WordNetLemmatizer()**

**lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]**

**print("\nLemmatized Words:")**

**print(lemmatized\_words)**

**OUTPUT:**

**Stemmed and Lemmatized Words:**

['hi', ',', 'how', 'are', 'you', 'do', '?', 'i', "'m", 'fine', '.', 'how', 'about', 'yourself', '?', 'i', "'m", 'fine', '.', 'how', 'about', 'yourself', '?', 'i', "'m", 'pretti', 'good', '.', 'thank', 'for', 'ask', '.', 'i', "'m", 'pretti', 'good', '.', 'thank', 'for', 'ask', '.', 'no', 'problem', '.', 'so', 'how', 'have', 'you', 'been', '?', 'no', 'problem……]

**Entity Recognition**

It involves identifying specific entities (such as names, dates, locations, products, etc.) within the user's input. Proper entity recognition enhances the chatbot's ability to understand user queries and provide accurate responses.

**import spacy**

**# Load the English language model**

**nlp = spacy.load("en\_core\_web\_sm")**

**# Example text for entity recognition**

**text = open('/content/dialogs1.txt','r').read()**

**# Process the text using spaCy**

**doc = nlp(text)**

**# Extract and print entities**

**entities = [(ent.text, ent.label\_) for ent in doc.ents]**

**print("Entities in the text:")**

**for entity, label in entities:**

**print(f"Entity: {entity}, Label: {label}")**

**BUILDING MODEL**

Using Machine Learning algorithm we can build the chatbot

For building the chatbot We convert the text data into csv data then we apply the scikit-learn library for text vectorization and a simple Linear Support Vector Machine (SVM) model for training the chatbot

**CODE**

**import pandas as pd**

**from sklearn.feature\_extraction.text import TfidfVectorizer**

**from sklearn.svm import LinearSVC**

**#Load the data and prepare the dataset**

**df = pd.read\_csv('/content/dialogs .csv')**

**questions = data['question'].tolist()**

**answers = data['answer'].tolist()**

**#Vectorize the text data for the question**

**vectorizer = TfidfVectorizer()**

**X = vectorizer.fit\_transform(questions)**

**y = answers**

**#Train the chatbot using Linear support vector Machine**

**model = LinearSVC()**

**model.fit(X, y)**

**#Implement the chatbot functionality**

**def chatbot\_response(user\_input):**

**input\_vector = vectorizer.transform([user\_input])**

**predicted\_answer = model.predict(input\_vector)**

**return predicted\_answer[0]**

**#Test the chatbot**

**print("Chatbot: Hello! Ask me a question (type 'exit' to end)")**

**while True:**

**user\_input = input("User: ")**

**if user\_input.lower() in ['exit', 'quit', 'bye', 'goodbye']:**

**print("Chatbot: Goodbye! Have a great day!")**

**break**

**response = chatbot\_response(user\_input)**

**print("Chatbot:", response)**

**OUTPUT:**

Chatbot: Hello! Ask me a question (type 'exit' to end)

User: hi, how are you doing?

Chatbot: i'm fine. how about yourself?

User: i'm pretty good. thanks for asking.

Chatbot: no problem. so how have you been?

User: i've been great. what about you?

Chatbot: i've been good. i'm in school right now.

User: what school do you go to?

Chatbot: i go to pcc.

User: do you like it there?

Chatbot: it's okay. it's a really big campus.

User: bye

Chatbot: Goodbye! Have a great day!

**INTEGRATING CHATBOT WITH FLASK**

**CODE**

**from flask import Flask, render\_template, request**

**import pandas as pd**

**from sklearn.feature\_extraction.text import TfidfVectorizer**

**from sklearn.svm import LinearSVC**

**app = Flask(\_\_name\_\_)**

**data = pd.read\_csv('your\_dataset.csv')e**

**questions = data['Question'].tolist()**

**answers = data['Answer'].tolist()**

**vectorizer = TfidfVectorizer()**

**X = vectorizer.fit\_transform(questions)**

**y = answers**

**model = LinearSVC()**

**model.fit(X, y)**

**def chatbot\_response(user\_input):**

**input\_vector = vectorizer.transform([user\_input])**

**predicted\_answer = model.predict(input\_vector)**

**return predicted\_answer[0]**

**# Flask Routes**

**@app.route('/')**

**def index():**

**return render\_template('index.html')**

**@app.route('/get\_response', methods=['POST'])**

**def get\_response():**

**user\_input = request.form['user\_input']**

**response = chatbot\_response(user\_input)**

**return render\_template('index.html',user\_input=user\_input, bot\_response=response)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(debug=True)**

**INDEX.HTML CODE**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Chatbot</title>**

**</head>**

**<body>**

**<h1>Chatbot</h1>**

**<form action="/get\_response" method="post">**

**<label for="user\_input">You:</label>**

**<input type="text" id="user\_input" name="user\_input" required>**

**<input type="submit" value="Send">**

**</form>**

**<div>**

**<p>You: {{ user\_input }}</p>**

**<p>Chatbot: {{ bot\_response }}</p>**

**</div>**

**</body>**

**</html>**

**SOURCE CODE**

**#importing libraries**

import tensorflow as tf

import unicodedata

import re

import numpy as np

import warnings

warnings.filterwarnings('ignore')

**#Load the data**

df=open('/content/dialog.txt','r').read()

**#Split the question and answer**

QA\_list=[QA.split('\t') for QA in df.split('\n')]

print(QA\_list[:5])

**#Data processing**

pip install nltk

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

import string

# Download NLTK resources (only needed once)

nltk.download('punkt')

nltk.download('stopwords')

def normalize\_text(text):

# Tokenize the text

tokens = word\_tokenize(text)

sentences = nltk.sent\_tokenize(text)

# Remove punctuation and special characters

tokens = [word for word in tokens if word.isalnum()]

# Remove stopwords

stop\_words = set(stopwords.words('english'))

tokens = [word for word in tokens if word not in stop\_words]

# Join the tokens back into a normalized string

normalized\_text = ' '.join(tokens)

return normalized\_text

# Example usage

input\_text = open('/content/dialog.txt','r').read()

normalized\_text = normalize\_text(input\_text)

print("Normalized Text:", normalized\_text)

from nltk.stem import PorterStemmer

from nltk.stem import WordNetLemmatizer

# Download NLTK resources (only needed once)

nltk.download('punkt')

nltk.download('wordnet')

# Example text

text = open('/content/dialogs .csv','r').read()

# Tokenize the text

words = nltk.word\_tokenize(text)

# Perform Stemming

stemmer = PorterStemmer()

stemmed\_words = [stemmer.stem(word) for word in words]

print("Stemmed Words:")

print(stemmed\_words)

# Perform Lemmatization

lemmatizer = WordNetLemmatizer()

lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]

print("\nLemmatized Words:")

print(lemmatized\_words)

def preprocessing(text):

#Case folding and removing extra whitespaces

text=remove\_diacritic(text.lower().strip())

#Ensuring punctuation marks to be treated as tokens

text=re.sub(r"([?.!,¿])", r" \1 ", text)

#Removing redundant spaces

text= re.sub(r'[" "]+', " ", text)

#Removing non alphabetic characters

text=re.sub(r"[^a-zA-Z?.!,¿]+", " ", text)

text=text.strip()

#Indicating the start and end of each sentence

text='<start> ' + text + ' <end>'

return text

**#Building the models**

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.svm import LinearSVC

data = pd.read\_csv('/content/dialogs .csv')

questions = data['question'].tolist()

answers = data['answer'].tolist()

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(questions)

y = answers

model = LinearSVC()

model.fit(X, y)

def chatbot\_response(user\_input):

input\_vector = vectorizer.transform([user\_input])

predicted\_answer = model.predict(input\_vector)

return predicted\_answer[0]

print("Chatbot: Hello! Ask me a question (type 'exit' to end)")

while True:

user\_input = input("User: ")

if user\_input.lower() in ['exit', 'quit', 'bye', 'goodbye']:

print("Chatbot: Goodbye! Have a great day!")

break

response = chatbot\_response(user\_input)

print("Chatbot:", response)

**CONCLUSION**

In conclusion, a well-thought-out and properly built chatbot can be a valuable tool for companies and organizations. The use of chatbots has the potential to improve customer service, expedite procedures, and give customers useful information in a conversational setting. Planning the objective, responses, and integration of the chatbot with user wants and preferences in mind is essential to maximizing their impact. A high-quality chatbot must undergo continuous improvement to remain relevant and respond to changing user needs. This progress must be driven by user feedback and testing. Chatbots can be effective tools for offering users quick and interesting interactions if they are used with the appropriate approach and a constant dedication to improvement.