PROJECT REPORT

Conversational Chatbot Using Machine Learning (TF-IDF + Logistic Regression)

Introduction

This project implements a machine learning–based conversational chatbot using Natural Language Processing (NLP) techniques.

Unlike advanced AI chatbots, this system works using:

- Predefined intents
- TF-IDF feature extraction
- Logistic Regression classification
- Streamlit web interface

The chatbot classifies user input into predefined categories and generates appropriate responses.

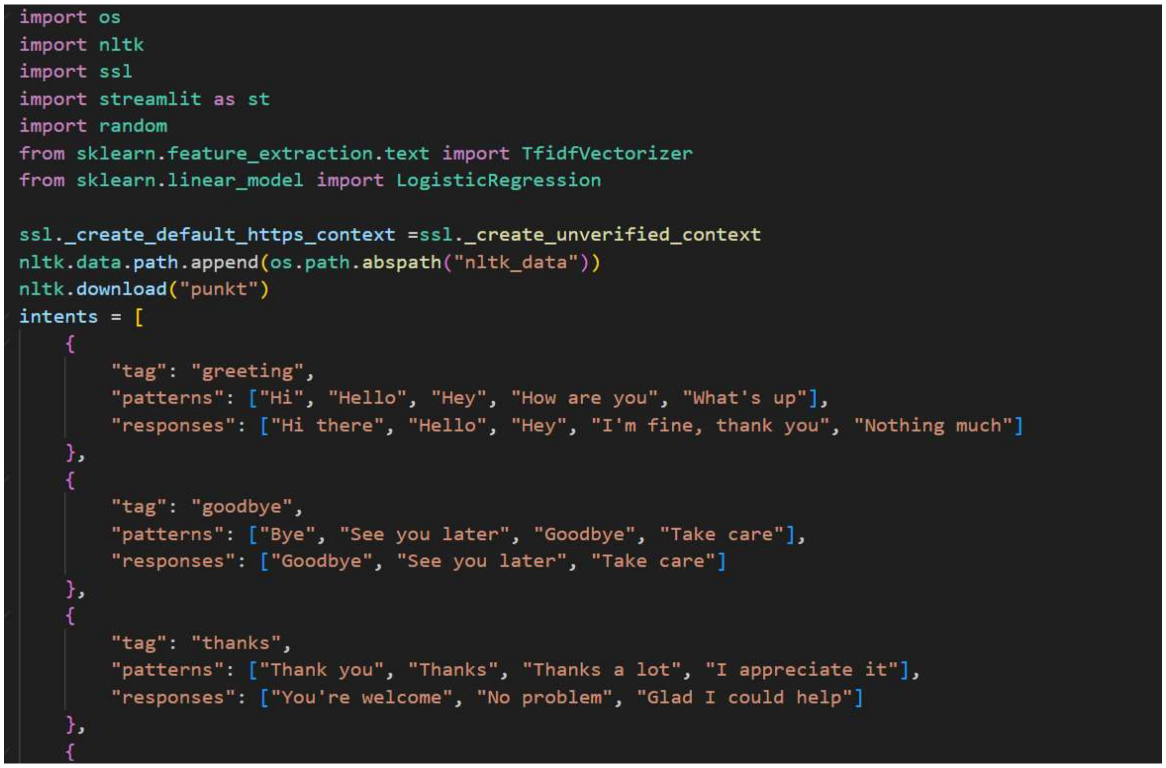
Objective

- To understand NLP basics
- To implement TF-IDF vectorization
- To train a Logistic Regression classifier
- To build a web-based chatbot interface
- To maintain session-based conversation

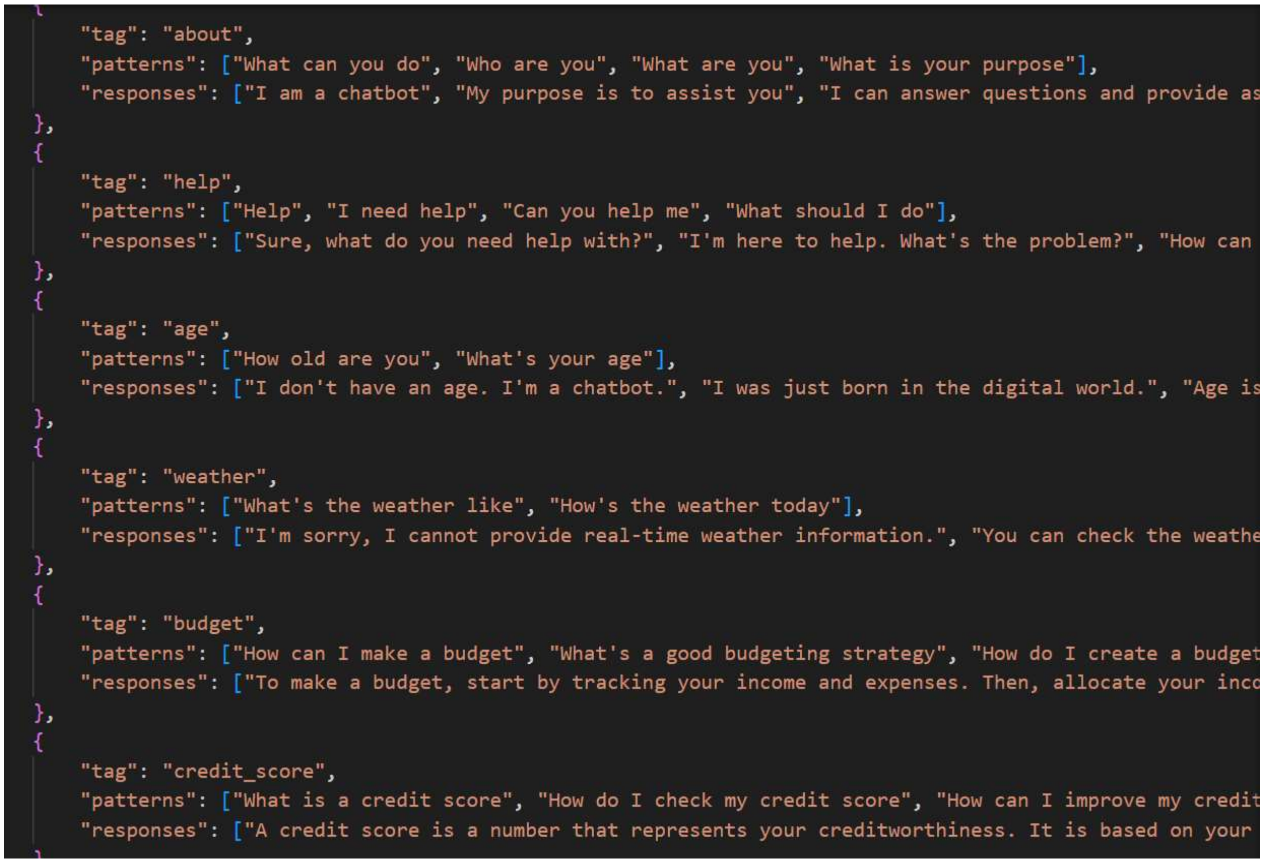
Technologies Used

Tool	Purpose
Python	Programming
Streamlit	Web Interface
NLTK	Text processing
Scikit-learn	ML model
TF-IDF	Text feature extraction
Logistic Regression	Classification

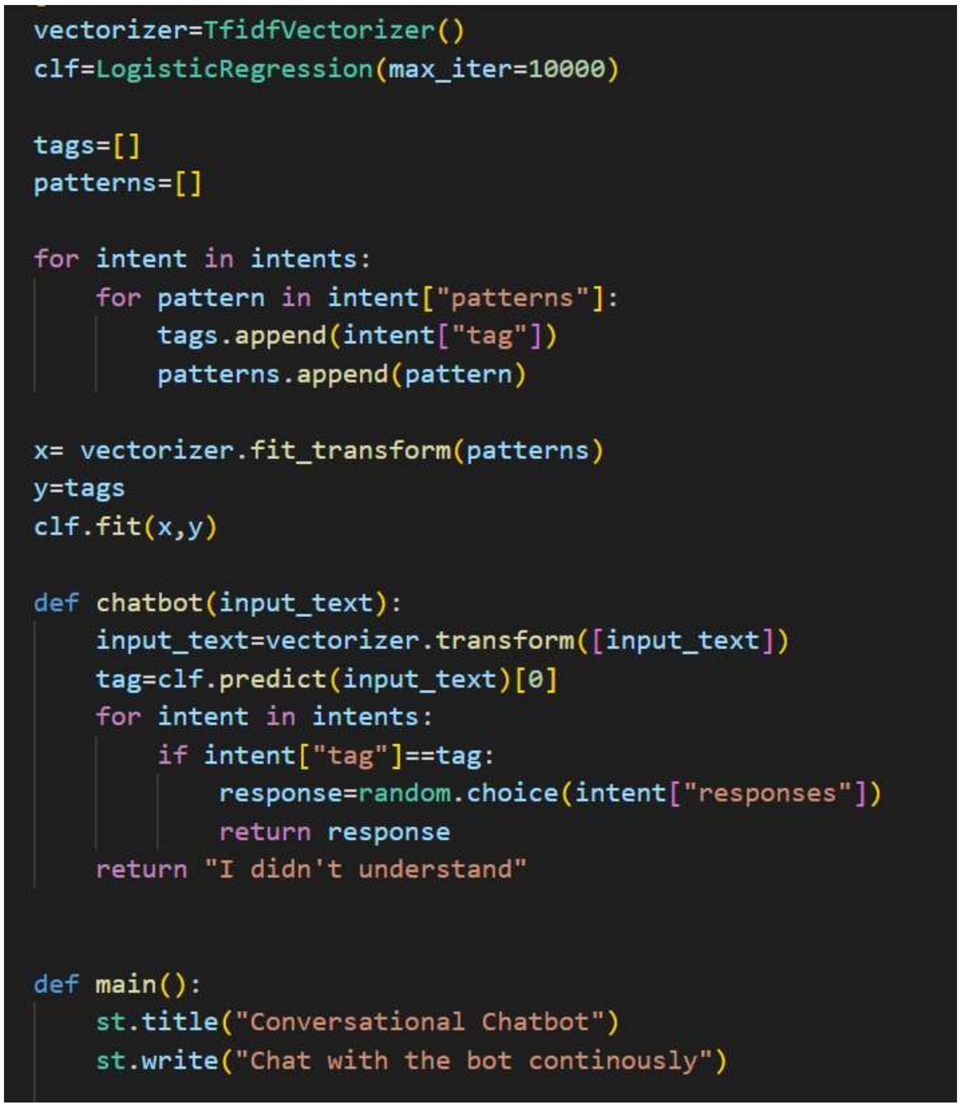
Code Explanation



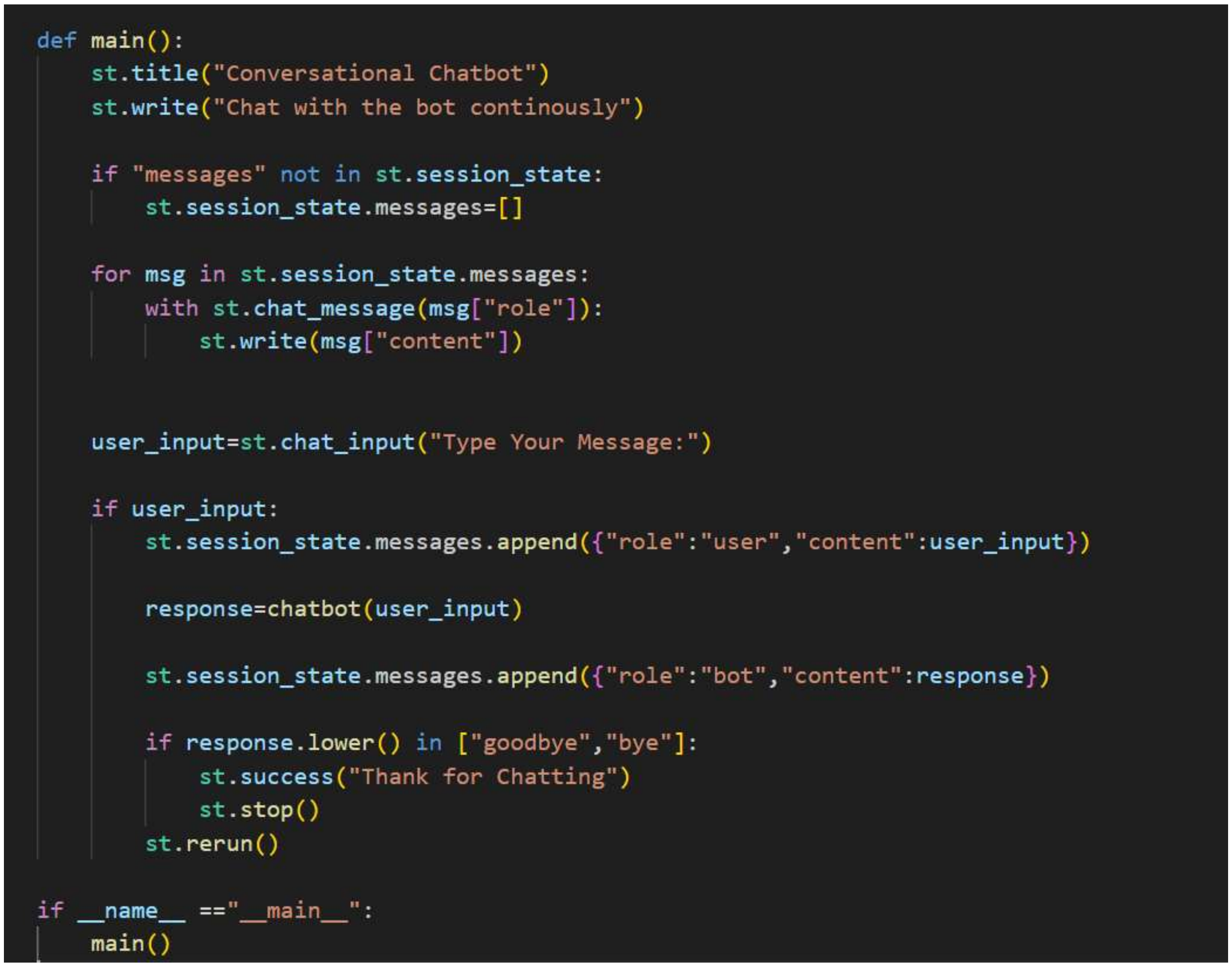
```
import os
import nltk
import ssl
import streamlit as st
import random
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

ssl._create_default_https_context =ssl._create_unverified_context
nltk.data.path.append(os.path.abspath("nltk_data"))
nltk.download("punkt")
intents = [
    {
        "tag": "greeting",
        "patterns": ["Hi", "Hello", "Hey", "How are you", "What's up"],
        "responses": ["Hi there", "Hello", "Hey", "I'm fine, thank you", "Nothing much"]
    },
    {
        "tag": "goodbye",
        "patterns": ["Bye", "See you later", "Goodbye", "Take care"],
        "responses": ["Goodbye", "See you later", "Take care"]
    },
    {
        "tag": "thanks",
        "patterns": ["Thank you", "Thanks", "Thanks a lot", "I appreciate it"],
        "responses": ["You're welcome", "No problem", "Glad I could help"]
    },
    {
```



```
    "tag": "about",
    "patterns": ["What can you do", "Who are you", "What are you", "What is your purpose"],
    "responses": ["I am a chatbot", "My purpose is to assist you", "I can answer questions and provide as
},
{
    "tag": "help",
    "patterns": ["Help", "I need help", "Can you help me", "What should I do"],
    "responses": ["Sure, what do you need help with?", "I'm here to help. What's the problem?", "How can
},
{
    "tag": "age",
    "patterns": ["How old are you", "What's your age"],
    "responses": ["I don't have an age. I'm a chatbot.", "I was just born in the digital world.", "Age is
},
{
    "tag": "weather",
    "patterns": ["What's the weather like", "How's the weather today"],
    "responses": ["I'm sorry, I cannot provide real-time weather information.", "You can check the weathe
},
{
    "tag": "budget",
    "patterns": ["How can I make a budget", "What's a good budgeting strategy", "How do I create a budget
    "responses": ["To make a budget, start by tracking your income and expenses. Then, allocate your inco
},
{
    "tag": "credit_score",
    "patterns": ["What is a credit score", "How do I check my credit score", "How can I improve my credit
    "responses": ["A credit score is a number that represents your creditworthiness. It is based on your
```



```
vectorizer=TfidfVectorizer()
clf=LogisticRegression(max_iter=10000)

tags=[]
patterns=[]

for intent in intents:
    for pattern in intent["patterns"]:
        tags.append(intent["tag"])
        patterns.append(pattern)

x= vectorizer.fit_transform(patterns)
y=tags
clf.fit(x,y)

def chatbot(input_text):
    input_text=vectorizer.transform([input_text])
    tag=clf.predict(input_text)[0]
    for intent in intents:
        if intent["tag"]==tag:
            response=random.choice(intent["responses"])
            return response
    return "I didn't understand"


def main():
    st.title("Conversational Chatbot")
    st.write("Chat with the bot continously")
```



```python
def main():
    st.title("Conversational Chatbot")
    st.write("Chat with the bot continously")

    if "messages" not in st.session_state:
        st.session_state.messages=[]

    for msg in st.session_state.messages:
        with st.chat_message(msg["role"]):
            st.write(msg["content"])


    user_input=st.chat_input("Type Your Message:")

    if user_input:
        st.session_state.messages.append({"role":"user","content":user_input})

        response=chatbot(user_input)

        st.session_state.messages.append({"role":"bot","content":response})

        if response.lower() in ["goodbye","bye"]:
            st.success("Thank for Chatting")
            st.stop()
        st.rerun()

if __name__ =="__main__":
    main()
```

Importing Libraries

`import os`

Used to interact with the operating system (file paths).

`import nltk`

Natural Language Toolkit – used for text processing.

`import ssl`

Handles SSL certificate verification (used while downloading NLTK data).

`import streamlit as st`

Creates the chatbot web interface.

```
import random
```

Used to select random responses from predefined replies.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Imports TF-IDF tool to convert text into numerical format.

```
from sklearn.linear_model import LogisticRegression
```

Imports Logistic Regression classifier.

Fixing SSL Issue

```
ssl._create_default_https_context =
ssl._create_unverified_context
```

This avoids SSL certificate errors while downloading NLTK data.

Setting NLTK Data Path

```
nltk.data.path.append(os.path.abspath("nltk_data"))
```

Stores NLTK files locally inside project folder.

Downloading Tokenizer

```
nltk.download("punkt")
```

Downloads tokenizer for splitting sentences into words.

Defining Intents

```
intents = [ ... ]
```

This is the chatbot knowledge base.

Each intent contains:

- `tag` → **Category name**
- `patterns` → **Example user inputs**
- `responses` → **Possible replies**

Example:

```
tag: greeting
patterns: ["Hi", "Hello"]
responses: ["Hi there", "Hello"]
```

Creating Vectorizer and Model

```
vectorizer = TfidfVectorizer()
```

Creates TF-IDF object to convert text into numbers.

```
clf = LogisticRegression(max_iter=10000)
```

Creates Logistic Regression model.

`max_iter=10000` **ensures proper training convergence.**

Preparing Training Data

```
tags = []
patterns = []
```

Empty lists to store training data.

```
for intent in intents:
    for pattern in intent["patterns"]:
        tags.append(intent["tag"])
        patterns.append(pattern)
```

This loop:

- **Collects all example sentences**
- **Stores corresponding tags**

Example:

patterns = ["Hi", "Hello", "Bye"]
tags = ["greeting", "greeting", "goodbye"]

Converting Text to Numerical Data

```
x = vectorizer.fit_transform(patterns)
```

Steps:

1. **Learn vocabulary**
2. **Convert text into TF-IDF matrix**

```
y = tags
```

Labels for classification.

Training Model

```
clf.fit(x, y)
```

Model learns relationship between text patterns and tags.

Now model can predict intent for new text.

Chatbot Function

```
def chatbot(input_text):
```

Defines chatbot response function.

```
input_text = vectorizer.transform([input_text])
```

Converts new user input into TF-IDF format.

```
tag = clf.predict(input_text)[0]
```

Predicts intent tag.

Example:
User: "Hi"
Prediction: "greeting"

```
for intent in intents:
    if intent["tag"] == tag:
        response = random.choice(intent["responses"])
        return response
```

- **Finds matching intent**
- **Picks random response**
- **Returns reply**

```
return "I didn't understand"
```

Fallback message if no match found.

Main Function (Streamlit UI)

```
def main():
```

Defines main app logic.

```
st.title("Conversational Chatbot")
st.write("Chat with the bot continously")
```

Displays heading and description.

Session State Initialization

```
if "messages" not in st.session_state:
    st.session_state.messages = []
```

Stores conversation history.

Display Previous Messages

```
for msg in st.session_state.messages:
    with st.chat_message(msg["role"]):
        st.write(msg["content"])
```

Shows stored chat messages.

User Input

```
user_input = st.chat_input("Type Your Message:")
```

Creates input box.

When User Sends Message

```
if user_input:
```

Runs only when user types something.

```
st.session_state.messages.append({"role":"user","content":us
er_input})
```

Stores user message.

```
response = chatbot(user_input)
```

Calls ML chatbot function.

```
st.session_state.messages.append({"role":"bot","content":res
ponse})
```

Stores bot reply.

Exit Condition

```
if response.lower() in ["goodbye","bye"]:
    st.success("Thank for Chatting")
    st.stop()
```

Stops app when user says goodbye.

Rerun App

```
st.rerun()
```

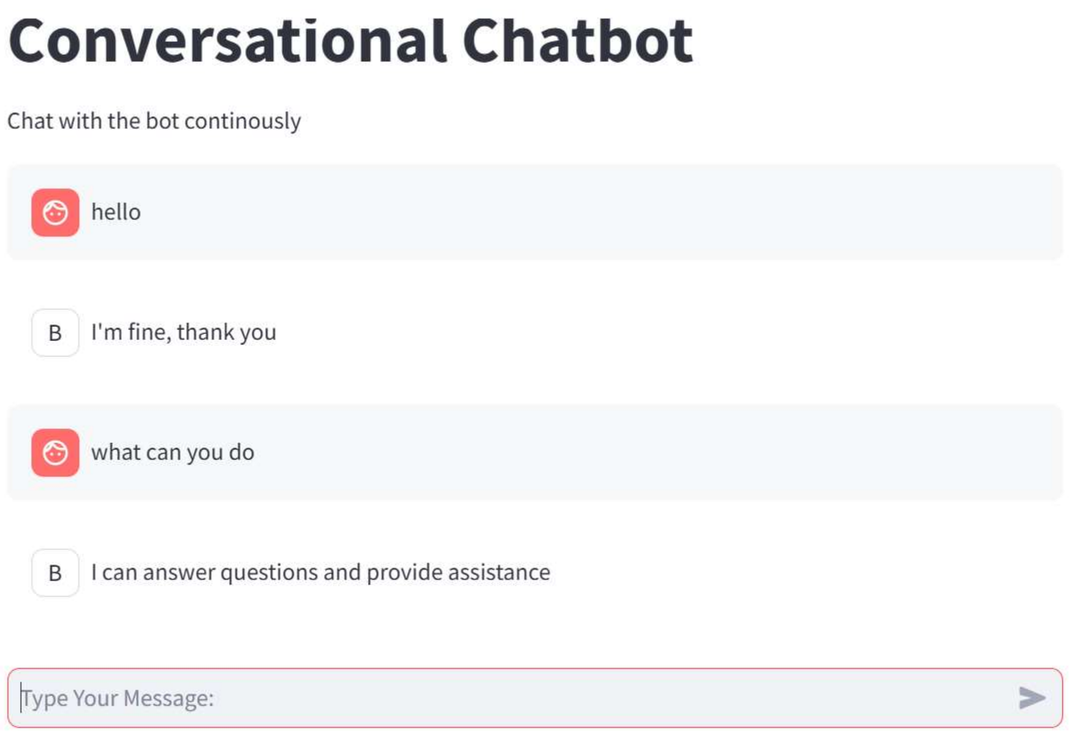
Refreshes app to update chat UI.

Run Main Function

```
if __name__ == "__main__":
    main()
```

Ensures program runs only when executed directly.

Output


Conversational Chatbot

Chat with the bot continously

hello

B I'm fine, thank you

what can you do

B I can answer questions and provide assistance

Working Flow Summary

1. **Load intents**
2. **Train ML model**
3. **User enters message**
4. **Convert to TF-IDF**
5. **Predict tag**
6. **Select response**

7. Display reply
8. Repeat

Advantages

- Runs offline
- Lightweight
- Beginner-friendly
- No API needed
- Fast response

Limitations

- Limited intelligence
- Cannot generate new sentences
- Works only on predefined patterns

Conclusion

This project successfully demonstrates how a chatbot can be built using traditional machine learning techniques. By combining TF-IDF vectorization with Logistic Regression classification, the chatbot identifies user intent and responds appropriately.

It is an excellent beginner NLP project suitable for academic submission, internships, and portfolio development.