Experiment No. 2
Aim: To build a cognitive text-based application to understand context for insurance help.

Objective:

The main objective of this experiment is to develop a simple chatbot that can assist users with insurance-related queries. The chatbot should be able to understand the user's message, identify important keywords, and respond with helpful and relevant answers. This experiment demonstrates the basic concept of creating a text-based AI assistant using natural language processing and keyword matching.

Theory:

A **cognitive text-based application** is a system that uses Artificial Intelligence (AI), particularly **Natural Language Processing (NLP)**, to understand, analyze, and respond to user inputs in natural (human) language. In this experiment, the chatbot is built using Python, and it uses two key NLP libraries — `spaCy` and `nltk`.

The chatbot can identify greetings (like "hello") and farewells (like "bye") and can also match specific insurance-related keywords (like "claim", "premium", "cancel policy") with predefined responses.

Theoretical Background:

1. **Natural Language Processing (NLP):**
   NLP is a branch of AI that deals with how machines understand and process human language. It includes tasks like tokenization, lemmatization, removing stop words, and part-of-speech tagging. In this experiment, NLP allows the chatbot to process and understand the user's text input.

2. **Keyword-Based Text Analysis:**
   This is a simple method to understand the context of a user query. It involves scanning the input text for specific keywords or phrases. If the input contains known keywords, the chatbot returns a predefined response. This is an easy and effective way to build a basic chatbot.

Libraries Used:

- **spaCy:** An advanced NLP library used for tasks like tokenization, stop word removal, and lemmatization.
- **nltk (Natural Language Toolkit):** A widely used Python library for NLP. It provides tools for text processing, classification, and analysis.

Steps Involved:

1. **Data Preparation:**
   A list of sample insurance queries and their responses is defined in the code. These are used as reference data for the chatbot to match against user queries.

2. **Text Preprocessing:**
   The user input is preprocessed using NLP techniques like converting to lowercase, removing punctuation, and removing common stop words. This helps the chatbot better understand the key words in a sentence.

3. **Keyword Matching:**
   The chatbot checks the user's query for specific keywords or phrases that match any of the sample queries. If a match is found, the corresponding response is returned.

4. **User Interaction:**
   The chatbot waits for user input in a loop. Based on the input, it responds with an appropriate answer, greeting, farewell, or default message if no match is found.

Expected Outcome:

By the end of this experiment, the user will have created a working chatbot that can:

- Understand and respond to user greetings and farewells.
- Identify keywords related to insurance (like "claim", "cancel policy", "premium") and provide correct responses.
- Show basic NLP techniques such as keyword extraction and pattern matching.

- Demonstrate how AI can be used to build simple, real-world applications.

Conclusion:

In this experiment, we successfully built a cognitive chatbot for answering insurance-related queries. The chatbot uses keyword-based matching and natural language processing to understand and respond to user inputs. This experiment provides a foundation for building more advanced AI systems and shows how NLP can be applied in real-world situations like customer service and insurance support.

7/31/25, 12:07 AM Untitled2.ipynb - Colab

HARSHIT SHARMA/51

```
!pip install spacy
!python -m spacy download en_core_web_sm
!pip install nltk
```

```
Requirement already satisfied: spacy in /usr/local/lib/python3.11/dist-packages (3.8.7)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.13)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.11) Requirement
already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.10) Requirement already
satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (8.3.6) Requirement already satisfied:
wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.1.3) Requirement already satisfied:
srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.5.1) Requirement already satisfied:
catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.10) Requirement already satisfied:
weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.4.1) Requirement already satisfied:
typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.16.0) Requirement already satisfied:
tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (4.67.1) Requirement already satisfied: numpy>=1.19.0 in
/usr/local/lib/python3.11/dist-packages (from spacy) (2.0.2) Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from spacy) (2.32.3) Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in
/usr/local/lib/python3.11/dist-packages (from spacy) (2.11.7 Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from spacy) (75.2.0) Requirement already
satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (25.0) Requirement already satisfied:
langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.5.0) Requirement already satisfied:
language-data>=1.2 in /usr/local/lib/python3.11/dist-packages (from langcodes<4.0.0,>=3.2.0->spacy)  Requirement already satisfied:
annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0, Requirement already satisfied:
pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,> Requirement already satisfied:
typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0 Requirement already satisfied:
typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0 Requirement already satisfied:
charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->sp Requirement already satisfied:
idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.10) Requirement already satisfied:
urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy)  Requirement already satisfied:
certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy)  Requirement already satisfied:
blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (1.3 Requirement already satisfied:
confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy Requirement already satisfied:
click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (8.2.1) Requirement already satisfied:
shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (1.5 Requirement already satisfied:
rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (13.9.4) Requirement already satisfied:
cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spa Requirement already satisfied:
smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spacy Requirement already satisfied:
MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->spacy) (3.0.2) Requirement already satisfied:
marisa-trie>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from language-data>=1.2->langcodes<4.0 Requirement already satisfied:
markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0 Requirement already satisfied:
pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,> Requirement already satisfied:
wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0 Requirement already satisfied:
mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typ Collecting
en-core-web-sm==3.8.0
 Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.whl
                                                                                                    12.8/12.8 MB 90.4 MB/s eta 0:00:00
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
 'Restart kernel' or 'Restart runtime' option.
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
```

```
import nltk
import spacy

# Load spaCy model for advanced text processing
nlp = spacy.load("en_core_web_sm")

# Sample customer queries and responses with keywords
queries_and_responses = [
    ("policy coverage", "Your policy covers damage to your vehicle caused by accidents."),
```

```
        ("file a claim", "You can file a claim online through our website."),
        ("premium for auto insurance", "The premium for your auto insurance is ₹500 per month."),
        ("rental car expenses", "Yes, rental car expenses are covered under your policy."),
        ("update contact information", "You can update your contact information in your online account."),
        ("grace period for premium payment", "The grace period for premium payment is 30 days."),
        ("add a new driver to my policy", "Yes, you can add a new driver to your policy. Contact our support for details."),
        ("cancel my policy", "To cancel your policy, please contact our customer service department."),
        ("discounts", "We offer various discounts based on your driving history and more."),
    ]

    # Default responses for various scenarios
```



```
    default_responses = {
        "greeting": "Hello! How can I assist you today?",
        "farewell": "Goodbye! Have a great day!",
        "default": "I'm sorry, I didn't understand your question. Please ask something else.",
    }

    # Function to classify user queries
    def classify_query(user_query):
        user_query = user_query.lower()

        if any(greeting in user_query for greeting in ["hi", "hello"]):
            return "greeting"
        elif any(farewell in user_query for farewell in ["bye", "goodbye"]):
            return "farewell"

        for keywords, response in queries_and_responses:
            for keyword in keywords.split():
                if keyword in user_query:
                    return response

        return "default"

    # Interactive user interface (you can simulate inputs here in Colab)
    def chatbot():
        print("Chatbot: Hello! I'm your insurance assistant. Type 'bye' to exit.")
        while True:
            user_query = input("You: ")
            query_type = classify_query(user_query)

            if query_type == "greeting":
                print("Chatbot:", default_responses["greeting"])
            elif query_type == "farewell":
                print("Chatbot:", default_responses["farewell"])
                break
            elif query_type != "default":
                print("Chatbot:", query_type)
            else:
                print("Chatbot:", default_responses["default"])

    # Run the chatbot
    chatbot()
```

```
 Chatbot: Hello! I'm your insurance assistant. Type 'bye' to exit.
 You: BYE
 Chatbot: Goodbye! Have a great day!
```