Artificial Intelligence
CS F407

Lab Project:
Messenger Bot using Python

Report based on CRISP-DM Methodology

Group Members:

1. Aman Agarwal - 2020B4AA2328H
2. Gauri Tewari - 2020B4A32314H
3. Kartik Chitoor - 2020B4A81617H

In accordance with the CRISP-DM (Cross-Industry Process for Data Mining) model, this report is divided into **6 phases** of presentation which highlight the entire working of the project.
It includes everything from the formulation and explanation of the data pipeline to the execution of the mentioned algorithms.

STAGE ONE: Business Understanding

1.1 Determine Business Objectives

1.1.1 Background:

The messenger bot project aims to develop an intelligent chatbot capable of interacting with users through an interface. The bot will utilize a knowledge base containing multiple sets of random responses to facilitate conversations with users. Additionally, the bot should be able to store and retrieve conversations from a database or the integrated knowledge base, allowing for appropriate responses to prompts. The knowledge base is to be stored using technologies such as MySQL, SQLite, or other flat file storage systems.

1.1.2 Business Objectives:

The primary objective of this project is to enhance user engagement and provide efficient support by developing an intelligent messenger bot. The bot should be able to handle generic interactions, interpret user queries, and provide appropriate responses. The key business objectives include:

- **Improve User Interaction**: Develop a chatbot which is capable of engaging in meaningful conversations with users. The bot should understand user intents, respond accordingly, and provide relevant information through retrieval.

- **Integration with existing web services**: Enable the bot to function with external web services to provide additional

functionalities to users. This includes features like retrieving weather information using an open source API, performing basic math operations, responding to generic prompts about topics like movies or literature.

- **Restaurant Recommendations**: Incorporate a recommendation system within the bot to suggest good eateries within the BITS, Hyderabad campus. To implement this, we create a corpus containing the relevant data and conditions needed to decide which restaurant to recommend.

1.1.3 Business Success Criteria:

The success of this project will be measured based on the achievement of the following criteria:

- **Accuracy and Relevance**: The chatbot should be able to understand user queries adequately and provide relevant responses based on the context. It should demonstrate a high level of understanding and offer helpful information or assistance.

- **Personalized Recommendations**: The restaurant recommendation system should be able to deliver personalized and accurate suggestions to users based on their preferences and the available eateries within the BITS, Hyderabad campus.

- **User Satisfactio**n: The overall success of the project will be ascertained by user feedback and satisfaction. Regular user testing and feedback collection can ensure that the bot meets user expectations and provides a meaningful user experience.

1.2 Assess Situation

1.2.1 Inventory of Resources:

In order to assess the situation effectively, it is essential to identify the available resources that will be utilized throughout the project. The inventory of resources includes:

- **Human Resources**: Identify the team members and the particular roles involved in the development and implementation of the messenger bot.

- **Technical Resources**: The tools and technologies required for the project are: a programming language (Python), an XML based markup language (AIML), framework (Flask) and open source APIs, to name a few.

- **Data Resources**: A custom built corpus for the restaurants available within BITS Pilani, Hyderabad Campus along with few relevant attributes (for each restaurant) is made to include in the chatbot's functionality of recommending restaurants.

- **Infrastructure:** Aside from the web, laptops were needed to run, implement and test the entire project along with building the required dependencies.

1.2.2 Requirements, Assumptions, and Constraints:

To successfully develop and deploy the messenger bot, we briefly outline the requirements, assumptions, and constraints that need to be considered:

- **Functional Requirements:** As mentioned previously, we expect the bot to perform all functions of a general chatbot in addition to recommending a suitable restaurant according to the user's description of their requirements.

- **Non-functional Requirements**: Here, we consider factors such as relevance of response, time taken to respond, ability to respond appropriately to the user prompt and provide a useful experience to the user in terms of the conversation and the outcome.

- **Assumptions**: We are limiting the corpus of restaurants to eateries within the campus and assume that the user would need the recommendation within this subset itself.

- **Constraints**: Even after including lots of sample data, it is possible that the chatbot encounters a prompt that it is not able to respond to due to the complexity of the prompt.

1.2.3 Risks and Contingencies:

During the development and implementation of the project, the following risks and contingencies should be considered:

Integration Issues: Integrating the messenger bot with external web services may be faced with compatibility issues or changes in the functionality of the API.
Contingency: Keeping a constant check for updates and options to employ alternate resources can help to minimize this risk.

1.2.4 Costs and Benefits:

The major cost of this project arises from its implementation. The modeling can be significantly enhanced by obtaining paid APIs which have much more sophisticated data compared to the open source ones. Further, the interface can be improved by developing it into an app or a hosted website.

1.3 Determine Data Mining Goals

1.3.1 Data Mining Goals

For the functionality of recommending restaurants given user preferences, we write a rule-based code which takes the input into consideration, compares it with the conditions available in the knowledge base and gives the output as dictated by the rules.

1.3.2 Data Mining Success Criteria

Relevance, quality and usefulness of the responses of the chatbot to the given user input is the most important criteria to measure the success of the implementation of the program. Additionally, effective utilization of the rules in the knowledge base might be taken into account as well.

1.4 Produce Project Plan

1.4.1 Project Plan

- Creating and incorporating the necessary data (knowledge bases)
- Find the most appropriate techniques to integrate them into the main functionality of the chatbot
- Decide the rules which determine the output/response for a given input/query.
- Employ tools to create the interface which can accept input from the user into a field and then also display the output provided by the chatbot

1.4.2 Initial Assessment of Tools and Techniques

The frameworks and tools selected for the task must be applied at the right stage to successfully compile and execute the purpose of the chatbot. In addition to this, we proceed with rule-based knowledge base which essentially compares the user input to its set of expected

input values and then shows the response based on the condition(s) satisfied.

STAGE TWO: Data Understanding

2.1 Initial Data Collection Report

The data required for recommending restaurants from BITS Hyd was taken from the respective outlets and we have created an aiml file for the same with the data we collected with appropriate tags and statements.

2.2 Data Description Collection Report

AIML is driven by two factors:

- It takes a lot of writing to develop a chatbot that can respond to our questions and engage in conversation.
- Creating a chatbot demands strong programming abilities, as well as a combination of desired conversation flow.

Components of an AIML file:

1. **aiml tag**: The content of the aiml file is contained in this tag.

2. **category tag**: This tag describes the fundamental piece of AIML information. There are often two tags in this tag: a pattern tag and a template tag.

3. **The pattern tag**: This tag is compared to user input. Different types of text can be entered by a user, and those texts are compared to various patterns in aiml files.

Note:Please take note that this tag must be first inside the category tag. Additionally, everything on the pattern tag needs to be written in capital letters. Pattern tags are case-sensitive.

4. **template tag**: This tag is used to save a specific response to a certain user input. The chatbot will react with the appropriate template for a certain pattern.

   Note: It must be the tag immediately following the category tag.

5. **star tag**: This tag is intended to collect user-provided wildcard input. One or more words are indicated by the star wildcard. This enables us to process any user-input type and provide results in accordance with it.

6. **Srai tag**: This tag has a variety of uses. This element allows us to utilize the same template as a response for various user inputs. This is helpful when a user may submit the same inquiry in a variety of formats, and the response is consistent across all of those inputs.

   Note:The srai tag is replaced with the appropriate template when the pattern matches the input.

7. **Random tag**: When the user inputs that specific pattern, the interpreter will randomly choose one of the defined lists of answers for that input.

8. **set tag**: With this tag, a local variable called a predicate that is unique to the client with whom our chatbot is conversing is set. Both preset variables and programmer-created variables are possible.

9. **get tag**: The get tag is used to get a local variable's value that was previously set while communicating with a particular client.

10. **that tag**: When a chatbot is communicating with a client in a certain context, this tag is useful. The previous statement generated by the chatbot is contained within that tag, allowing us to construct a new

template for responding in line with the context. In general, this tag is helpful during an interrogative discourse.

11. **topic tag**: This tag aids in context setting so that all subsequent conversions may be carried out with consideration for that context. This tag is typically used during Yes-or-No conversations.

12. **think tag**:The think tag allows you to set variables privately from the client. In actuality, the set tag also informs the user of the variables' settings, but that is not the case in this instance.

13. **condition tag**:The condition tag performs the same function as a switch case in programming languages. It aids the chatbot's ability to react in accordance with a certain circumstance dependent on a variable and its related values.

STAGE THREE: Data Preparation

3.1 Rationale for Inclusion/exclusion

Some of the topics which we have included for the chatbot to understand and respond about are:

- **Restaurants available within BITS Hyderabad, along with the dishes they serve, based on the preferences which the user specifies.**
- Educational topics like science, history, geography and computers
- Sports and movies
- Psychology, philosophy and personality
- Politics, literature, astrology, music etc.

3.4 Merged Data

We have made the following AIML files with which the chatbot feels real and does have enough things to talk about with the user:

Green:

Adverbs.aiml, Atmoic.aiml, Biography.aiml, Blackjack.aiml, Bot.aiml, Client.aiml, Computers.aiml, Date.aiml, Default.aiml, **Eateries.aiml,** Geography.aiml, Happy.aiml, Inquiry.aiml, Integer.aiml, Knowledge.aiml, Parts.aiml, Personality.aiml, Predicates.aiml, Psychology.aiml, Reduce.aiml, Salutations.aiml, Science.aiml, Spam.aiml, Stack.aiml, update.aiml, Utilities.aiml

Orange:

Astrology.aiml, Dialog.aiml, Drugs.aiml, History.aiml, Humor.aiml, Literature.aiml, Money.aiml, Movies.aiml, Multiple.aiml, Politics.aiml, Religion.aiml, Sports.aiml, Stories.aiml, Wallace.aiml, Wordplay.aiml

Yellow:

AI.aiml, Emotion.aiml, Food.aiml, Gossip.aiml, Human.aiml, Music.aiml, and Philosophy.aiml

After creating the individual AIML files, we merged all of them using the following code written in Python. The code is as follows:

```
def get_all_files(color):
    """
    List all files recursively in the root specified by root
    """
    files_list = []
    root = 'knowledge/' + color
    for path, subdirs, files in os.walk(root):
```

```
    for name in files:
        files_list.append(os.path.join(root, name))
print(files_list)
return files_list[0:-1]
```

We also categorised the files based on the frequency of the topics:
Most commonly asked/referred topics were categorised as Green, then Orange and Yellow.

STAGE FOUR: MODELING

4.1 Select Modeling Techniques

About Artificial Intelligence Markup Language (AIML):

AIML is designed to be simple and easy to understand. It uses a **pattern-matching approach**, where user inputs are matched against predefined patterns.

It provides a **flexible** framework for creating rule-based conversations and handling various user queries and interactions.
Additionally, it provides a quick and efficient way to prototype and develop chatbots, along with enabling the developer to introduce **updates** and modifications to the chatbot's knowledge base without much alteration to the main code files.

AIML can be **integrated** with a variety of **compatible** frameworks and models which enhances its flexibility.
Such chatbots can be deployed across **multiple platforms**, including web apps using Flask, for example.

About XML:

We used XML to deal with our AIML files, and to organize and manage them efficiently. XML provides a structured and hierarchical format that can be utilized to reference and include AIML files easily.

About Flask:

Flask is a lightweight and flexible web framework for Python. It provides the foundation for building web applications, offering a simple yet powerful way to handle requests and responses.

About HTML:

HTML structure and present content on the web. We have leveraged it to create the frontend interface through which users interact with the chatbot (in addition to Flask).

4.2 Generate Test Design

Here, we design a plan to evaluate the chatbot's performance. This involves defining clear test objectives, which means we map the diverse user inputs to appropriate responses from the chatbot. The appropriateness can be determined by the usefulness and relevance of the response.

4.3 Build Model

When starting implementation, the emphasis is on correctly structuring modelling techniques for the chatbot's functioning. This includes using response generation methods like rule-based approaches and integrating them with the necessary frameworks which will process the input, generate the response and show the output to the user.. Training and fine-tuning these models with relevant scenarios is crucial to optimize performance.

4.4 Assess Model

This process involves defining relevant metrics to assess the chatbot's performance, such as quality/relevance of response or user satisfaction ratings. User feedback collection to check if the chatbot behaves as expected can go a long way in determining its success. The results are analyzed to identify areas for improvement, leading to necessary modifications to the knowledge base and associated frameworks. The following code was used to run the bot using localhost in Flask

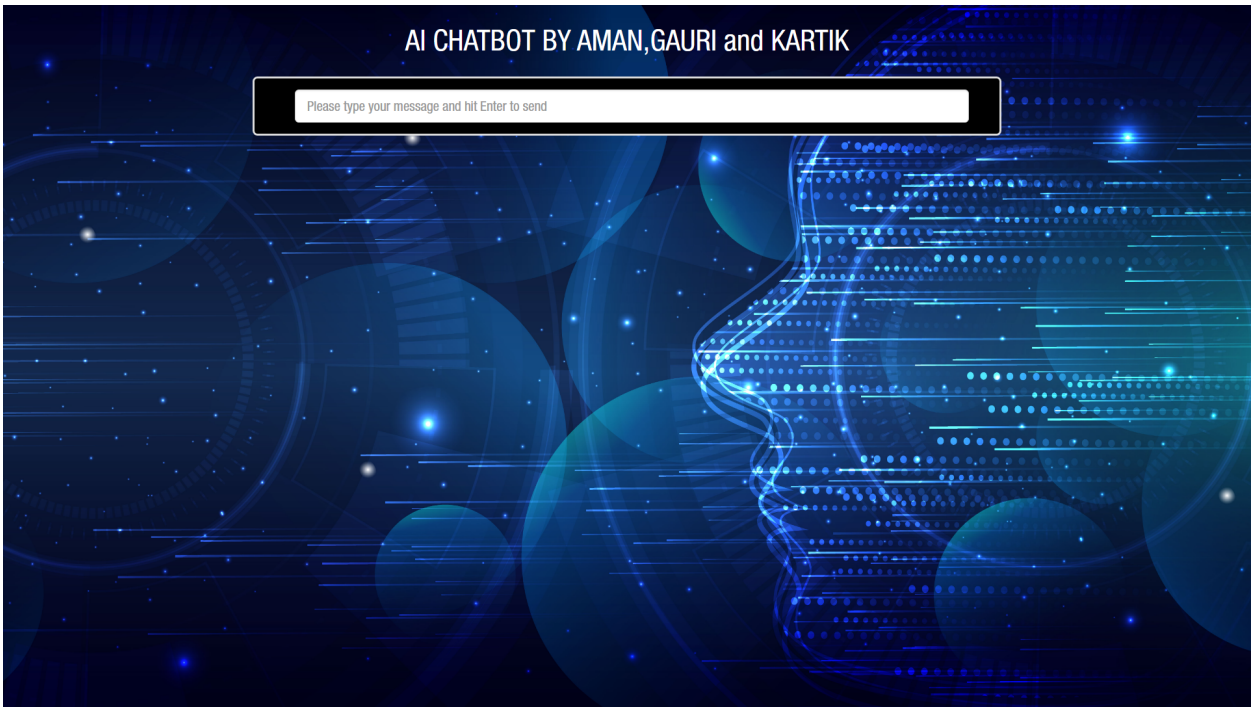
```
app = Flask(__name__)
atomic_kernel = None
if __name__ == '__main__':
    port = int(os.environ.get("PORT", 33507))
    app.run(host='0.0.0.0', port=port, debug=True)
```

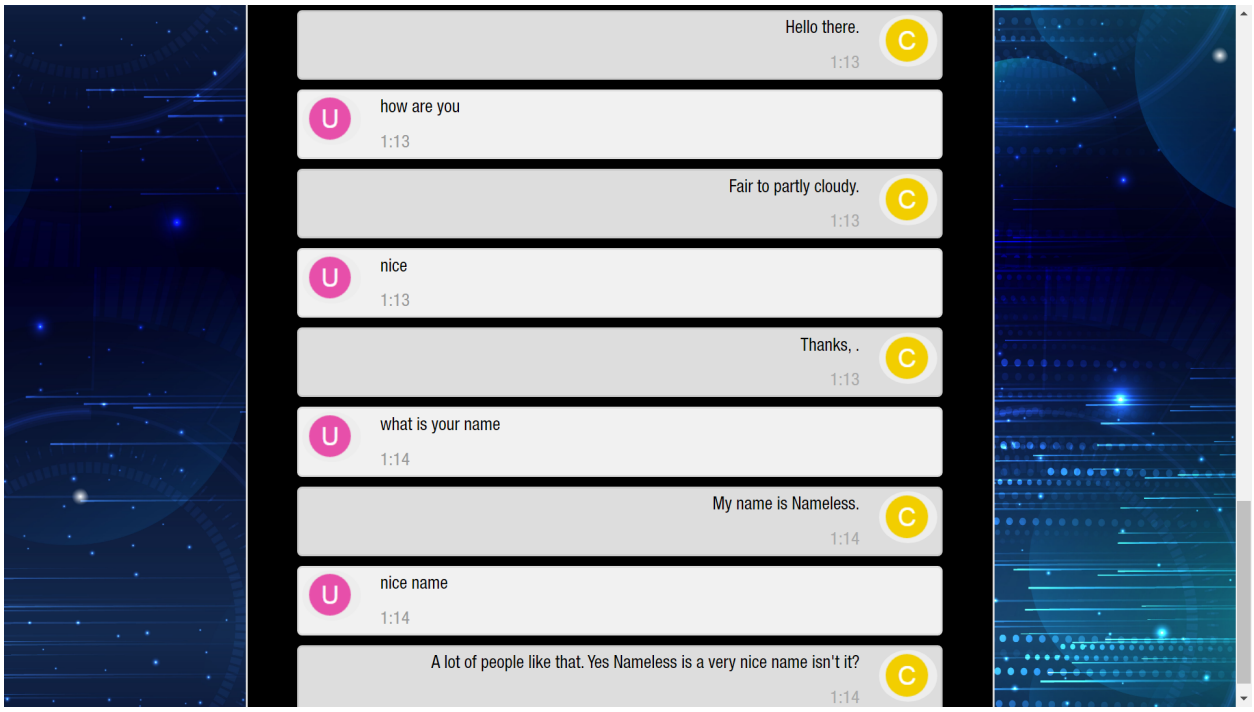
STAGE FIVE: EVALUATION

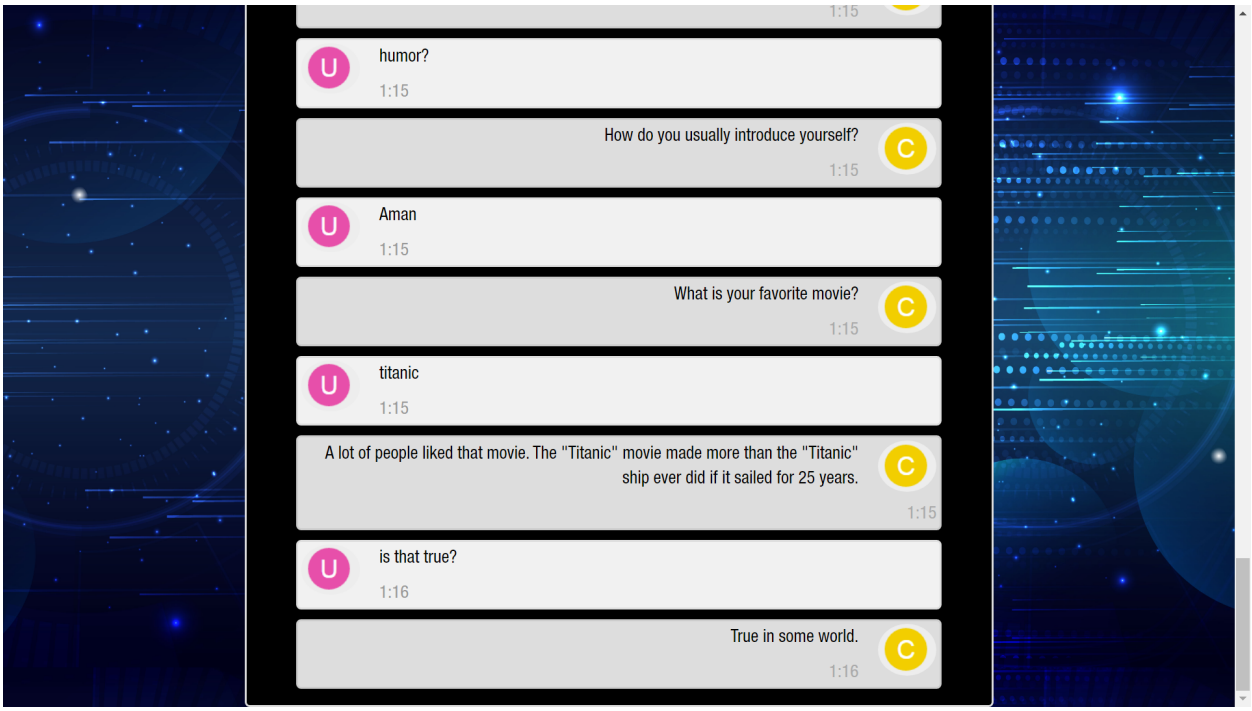
5.1 Evaluate Results

We aim to analyse the outcomes of the chatbot development and testing process. This involves assessing the performance of the chatbot against our success criteria and evaluating the effectiveness of the implemented models and techniques in terms of data flow and processing. We also identify strengths and weaknesses, noting areas where the chatbot excels and areas that require further improvement. By thoroughly evaluating the results, we gain insights into the overall performance and effectiveness of the chatbot.

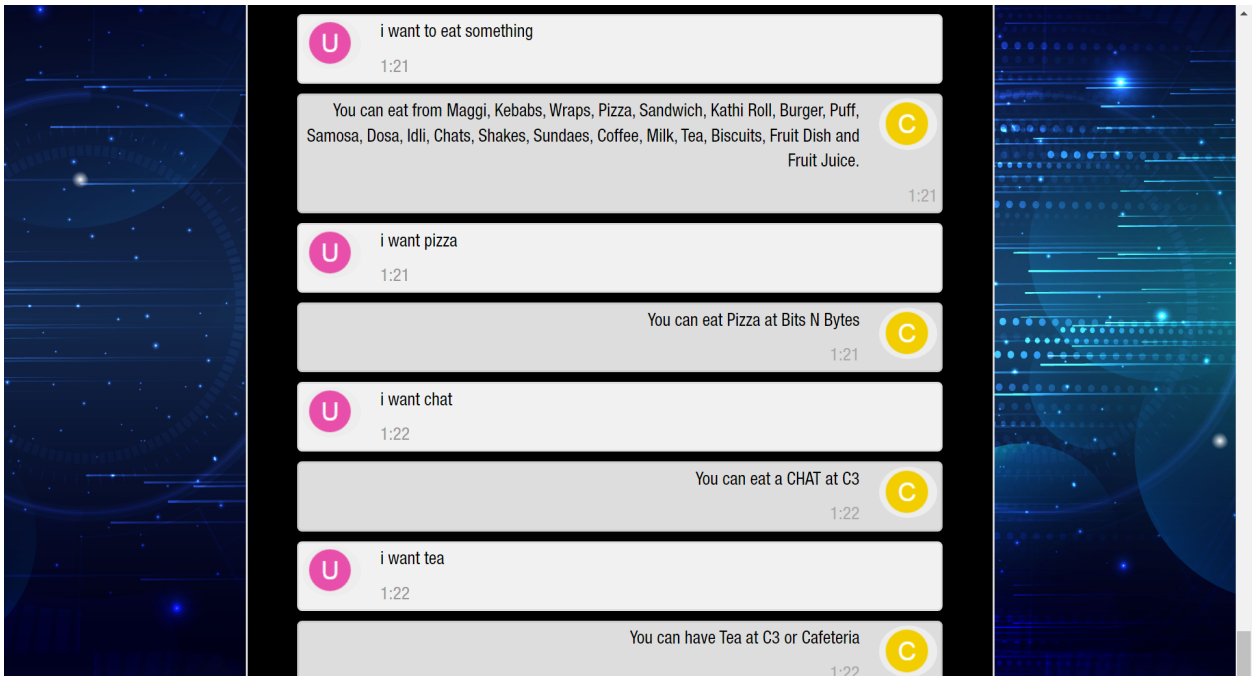
Below are attached some sample conversations with the chatbot:

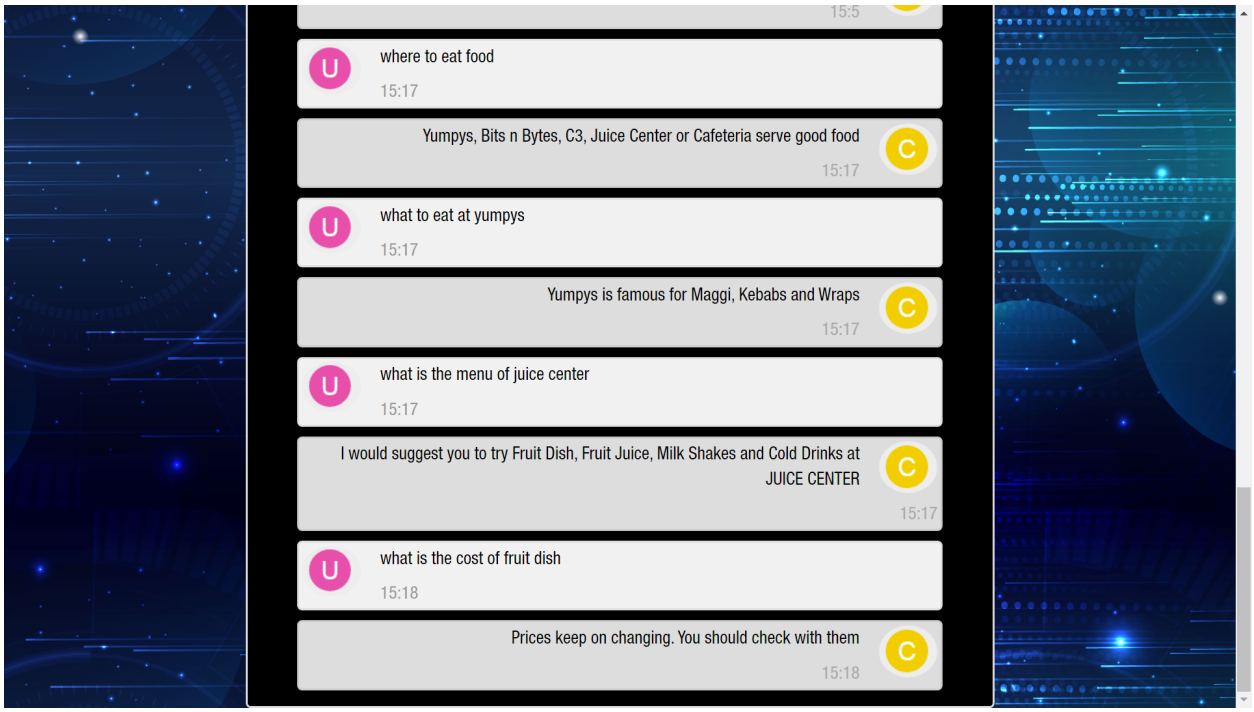

AI CHATBOT BY AMAN,GAURI and KARTIK
Please type your message and hit Enter to send


Hello there.
1:13
C
U
how are you
1:13
Fair to partly cloudy.
1:13
C
U
nice
1:13
Thanks, .
1:13
C
U
what is your name
1:14
My name is Nameless.
1:14
C
U
nice name
1:14
A lot of people like that. Yes Nameless is a very nice name isn't it?
1:14
C

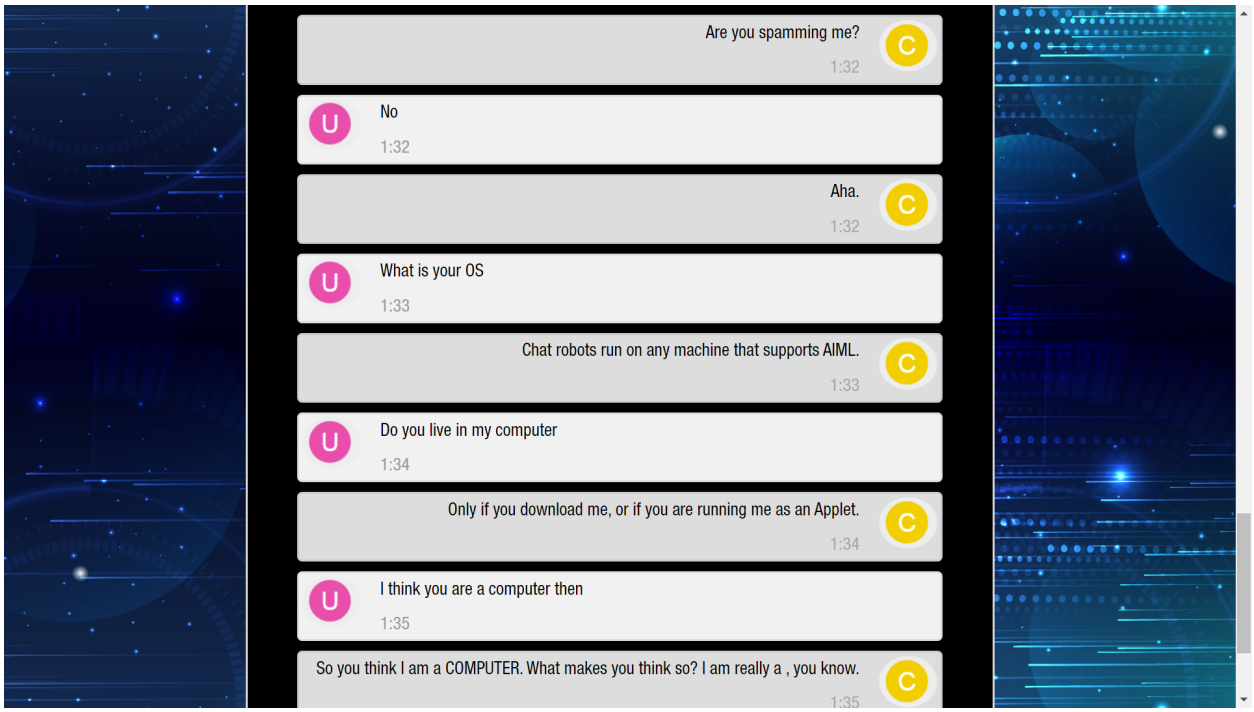


Now if we ask the bot about eateries, it gives us all the options available in BITS and then answers us by suggesting us a place to eat/drink depending on our input. Following is a sample conversation about the topic:





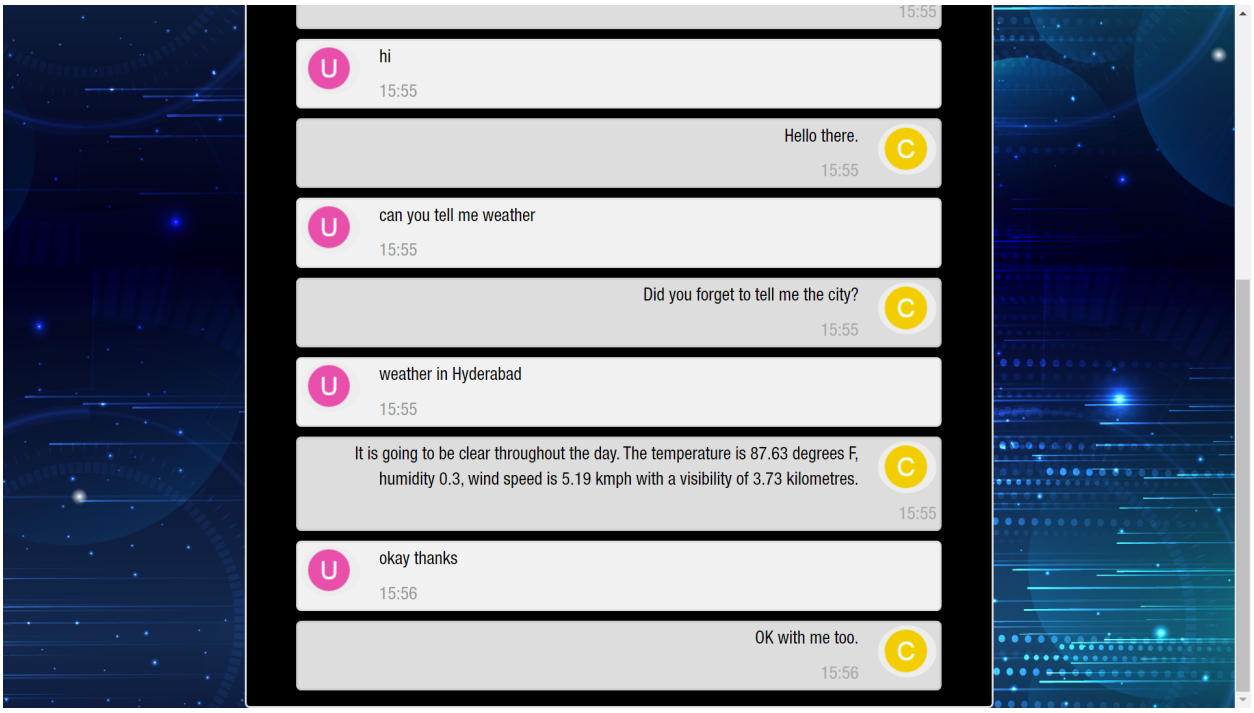
Some random humour is also shown in the below conversation:



We also integrated Weather API from Weatherbit using RapidAPI.com; it takes the Place or Latitude and Longitude of the location, sends a request to the API using my personal API key and fetches the result, after that it displays the result using the following piece of code.

```
def weather_parser(response, place):
    resp = json.loads(response)
    current = resp['currently']
    summary = resp['hourly']['summary']
    statement = 'It is going to be ' + str(summary.lower()) + ' The
temperature is ' + str(current['temperature']) + ' degrees F, humidity ' + str(
        current['humidity']) + ', wind speed is ' + str(current['windSpeed']) +
' kmph with a visibility of ' + str(current['visibility']) + ' kilometres.'
    return statement
```

Attached below is a sample conversation using the Waether API:



5.2 Review Process

To summarise, we first created a knowledge base for the eateries inside the BITS Hyderabad campus and incorporated it into the chatbot. Then, we added other knowledge bases to provide some basic functionality to the program. Then we created an interface which will let the end user prompt the chatbot and view the responses generated by it. This way, we have a functional chatbot that can answer queries when prompted.

5.3 Determine the Next Steps

The following steps can be implemented or enhanced in the future:

- Include more APIs and packages to expand the horizon of conversation, which the chatbot can uphold
- Creating and using larger knowledge bases to enable the program to recommend restaurants spanning an entire city or state
- Improving the interface to make an Android app using Kotlin
- Integrate the feature of remembering previous conversations with a specific user, which can be done by asking the user to log in to the app/portal
- Asking for user feedback/ratings after a conversation to see how much the chatbot can meet up to the user's expectations of an appropriate/meaningful response.

STAGE SIX: DEPLOYMENT

6.1 Plan Deployment

We have implemented the project using AIML, Python, Flask, XML and HTML, and displayed the web app when the program is run. In the window,

the user can enter his/her prompt and obtain the response to it. As of now, this works on a one prompt-one answer method where the chatbot attempts to answer each separate message which the user sends to it.

Various runs of the program were done to see how well the program is able to handle different kinds of prompts from the set of topics which have been integrated into it.

The chatbot is also asked to recommend eateries to the user and hence we have tested that functionality too.

The following piece of python code was used to deploy the model using aiml and html files:

```
@app.route('/computers/<string:user_text>')
def computers(user_text, methods=['GET']):
    computers_kernel = spin_kernel('computers')
    response = computers_kernel.respond(user_text.upper())
    return response
@app.route('/eateries/<string:user_text>')
def eateries(user_text, methods=['GET']):
    eateries_kernel = spin_kernel('eateries')
    response = eateries_kernel.respond(user_text.upper())
    return response

if __name__ == '__main__':
    port = int(os.environ.get("PORT", 33507))
    app.run(host='0.0.0.0', port=port, debug=True)
```

6.2 Plan Monitoring and Maintenance

We are required to monitor the responses of the chatbot and it can be an option to collect user feedback to decide upon improvements to be made (if necessary). It would also be beneficial to consistently check and update the knowledge base which is used in order to incorporate newer additions and accommodate more flexibility in the system.

6.3 Produce Final Report

- Executive summary
- Project Methodology
- Creating the knowledge base(s)
- Integrating the required frameworks
- Modelling techniques and implementation
- Evaluation and results
- Implementation of improvements based on feedback and requirements
- Conclusive note

6.4 Review Project

Hence, by the methodologies and techniques described above, this project has been implemented and executed successfully. Chatbots can prove to be useful for all kinds of businesses, and especially for those which run a majority share it online. Chatbots can be very helpful in customer care services and recommendation systems owing to their ability of assessing and processing user inputs effectively.