EMOTIONAL CHATBOT FOR STUDENT MENTAL HEALTH

TEAM MEMBERS:

- GANGARAJU THARUN – 2019103520
- KABILAN P – 2019103532
- RAJAGNANAGURU R – 2019103565

GUIDE'S NAME:

- Dr. Lalitha Devi

PROBLEM STATEMENT:

Nowadays, students are facing a lot of mental health issues due to various reasons like pandemic lockdowns, peer pressure, social media bullying, academic stress, loneliness, sexual harassment, etc. Because of which students are unable to progress well in their life, both emotionally and academically. Due to the battle of life, the students are also unable to receive proper guidance from experienced and knowledgeable humans to solve their personal issues. Therefore, developing a human friendly chatbot that can help students get the right guidance for their issues at the right time is a much needed one.

Also, it would not be feasible for some of the students to pay for a psychologist for mental health advices sessions. They may need to make appointments and it would require time to consult within the time which they may feel low or their mind set may change. So, an application which responds to student feelings correctly helps him/her to resolve or get confident instantly or for the time being.

OVERALL OBJECTIVE:

The objective is to develop a chatbot for students to promote their mental health through emotion recognition technique. These kind of chatbots can play a vital role in reducing the number of suicides in the country due to depression and stress. The chatbot named EmoChatbot helps students solve or prevent any mental health issue in their day-to-day life. converses with the student, understands his/her present emotional state/mental health issue (if any), identifies the cause of that emotion/mental health issue and provides the right guidance based on the reason identified. Emochatbot uses technologies like Neural network training with Natural language processing techniques and SQLite for data storage.

Our project is based on application which helps students to cope with stress and mental health issues. Students are facing a lot of mental health issues due to various reasons like peer pressure, academic stress, loneliness, etc. Due to the above reasons students fail to success both emotionally and academically. In this project, a chat bot is to be developed which chats like a mentor to enhance mindset of students by texting appropriate replies to their queries.

As an extension, there included an extra feature of storage and retrieval of data in database in the form of time table. Time table is the academic period time table which shows the subject if prompted which period in number and the particular day like Monday, Tuesday, etc. These is included so that the student need not search for it if needed, instead just ask in the chat to the chatbot and also may update if needed.

INTRODUCTION:

A Chatbot is a computer program which conducts a conversation with human users. Through auditory or textual methods, chatbots indulge human beings in a basic conversation. These are used in messaging apps like Facebook messenger, Telegram, Slack etc and is a user-friendly method to clear one's queries, retrieve information, give simple directives or to engage oneself in a simple talk. Until recently chatbots were incapable of responding appropriately to the emotions of the user. But now a new development is seen in this area.

Our EmoChatbot is a trained emotional chatbot for friendly chat to express their emotional feelings of their live problems to this chatbot. There are many emotions included in this problem mapping different type of chats to different emotions and training the model. Emotion is a part of Human life. It persists in various situations such as inspiration, thoughts, awareness, imagination, focus, thinking, experience and effective decision- making. People occasionally express their emotions through conscious or subconscious from their face countenances, words and text – messaging. Whereas, most of them choose the text in the form of emails, commenting on items, weblogs as an interface for communication. To politicians, economists, market analysts and social scientists, identification of emotions from text can be of interest.

According to the World Health Organization (WHO), 1 in 10 people need mental healthcare worldwide, and different mental disorders are, portrayed by a combination of perceptions, feelings, and relationships with others. Results of a household survey conducted by National Health Services (NHS) states that 1 in 4 people experience a mental health problem each year, 1 in 6 people face a common mental health problem such as anxiety, depression each week in England. The number of people affected is increasing gradually, and with the isolation that Covid-19 brought, the numbers are much higher. Despite having access to health and social services and the number of people who need care is higher, only 70 per 100,000 mental health professionals are available in high-income nations and 2 per 100,000 in low-income nations.

Chatbots are a part and parcel of messaging apps and it was found that over 18.5 percent of the world's population used messaging apps in 2016 and the figures are expected to go over 25 percent by 2019, indicating how extensively chatbots are being used. A survey conducted by a reliable source found that over 60 percent of the people who use messaging apps are more than willing to talk to chatbots to communicate with a business firm or a brand. Selling and purchasing of goods and services are also done via these virtual assistants apart from receiving recommendations and advice from them. Facebook chatbots used by millions of people in 200 countries can be numerically calculated to be 30,000 in number.

This method in which the chatbots extract the emotion displayed in the conversation can be called sentiment analysis. This is followed by the generation of contextually and emotionally appropriate responses that are selected from the multitude of sentences that have already been provided as input. The development of artificial intelligence has come to a stage where the simulation by chatbots have become too realistic that the human users get confused as to whether they are talking to a programmed machine or a real, live human being. As a matter of fact, Ray Kurzweil, an American author, computer scientist, inventor, and futurist who has written books on AI, futurism, trans-humanism, and likewise, predicts that chatbots will have human-level language ability by the year 2029.

SUMMARY OF RELATED WORK:

At the most basic level, a chatbot is a computer program that simulates and processes human conversation (either written or spoken), allowing humans to interact with digital devices as if they were communicating with a real person. Chatbots can be as simple as rudimentary programs that answer a simple query with a single-line response, or as sophisticated as digital assistants that learn and evolve to deliver increasing levels of personalization as they gather and process information.

You’ve probably interacted with a chatbot whether you know it or not. For example, you’re at your computer researching a product, and a window pops up on your screen asking if you need help. Or perhaps you’re on your way to a concert and you use your smartphone to request a ride via chat. Or you might have used voice commands to order a coffee from your neighborhood café and received a response telling you when your order will be ready and what it will cost. These are all examples of scenarios in which you could be encountering a chatbot.

Driven by AI, automated rules, natural-language processing (NLP), and machine learning (ML), chatbots process data to deliver responses to requests of all kinds.

There are two main types of chatbots.

Task-oriented (declarative) chatbots are single-purpose programs that focus on performing one function. Using rules, NLP, and very little ML, they generate automated but conversational responses to user inquiries. Interactions with these chatbots are highly specific and structured and are most applicable to support and service functions—think robust, interactive FAQs. Task-oriented chatbots can handle common questions, such as queries about hours of business or simple transactions that don't involve a variety of variables. Though they do use NLP so end users can experience them in a conversational way, their capabilities are fairly basic. These are currently the most commonly used chatbots.

Data-driven and predictive (conversational) chatbots are often referred to as virtual assistants or digital assistants, and they are much more sophisticated, interactive, and personalized than task-oriented chatbots. These chatbots are contextually aware and leverage natural-language understanding (NLU), NLP, and ML to learn as they go. They apply predictive intelligence and analytics to enable personalization based on user profiles and past user behavior. Digital assistants can learn a user's preferences over time, provide recommendations, and even anticipate needs. In addition to monitoring data and intent, they can initiate conversations. Apple's Siri and Amazon's Alexa are examples of consumer-oriented, data-driven, predictive chatbots.

Whereas conversations with a "regular" chatbot have a professional tone and are conducted with the aim of problem-solving, talking with emotional chatbots should resemble the feeling of talking to a friend.

The technology behind emotional chatbots strives to make them grasp an understanding of the user's feelings and emotions and to then offer applicable advice or course of action. Arguably, the primary aim of chatbots is to make the user feel heard. And therein lies the potential they have to address mental health issues.

ELIZA and Woebot are examples of chatbots that provide mental healthcare. Assuming that the data privacy of the users is secured, encrypted emotional chatbots like them can be trusted companions. Users could divulge their personal problems, stories, feelings, or mental symptoms they have been experiencing.

What makes chatbots a great tool for addressing mental health issues is the lack of stigma. Robots do not judge, and hopefully, will not exploit your data either. The technology is not yet mature enough to safely allow for the disposition of professionals with chatbots. At the basic level, a chatbot like Replika should just be a companion that listens and talks to you. For professional use; however, the current best implementation would be healthcare professionals complementing their face-to-face (or monitor-to-monitor) interactions with patients with emotional chatbots. This will help them increase their outreach to patients.

In addition, no longer would they have to conduct a preliminary session to learn more about a patient's backstory. All that data would have previously been recorded by the chatbots and be available for review. Advice for mental health professionals: Investing in emotional chatbots for your websites to increase your outreach to patients and reduce your workloads.

The architecture model of a chatbot is decided based on the core purpose of development. There are two types of possible responses of chatbot: it can either generate a response from scratch as per machine learning models or use some heuristic to select an appropriate response from a library of predefined responses.

Generative Models

This model is used for the development of smart bots that are quite advanced in nature. This type of chatbot is very rarely used, as it requires the implementation of complex algorithms.

Generative models are comparatively difficult to build and develop. Training of this type of bot requires investing a lot of time and effort by giving millions of examples. This is how the deep learning model can engage in conversation. However, still, you cannot be sure what responses the model will generate.

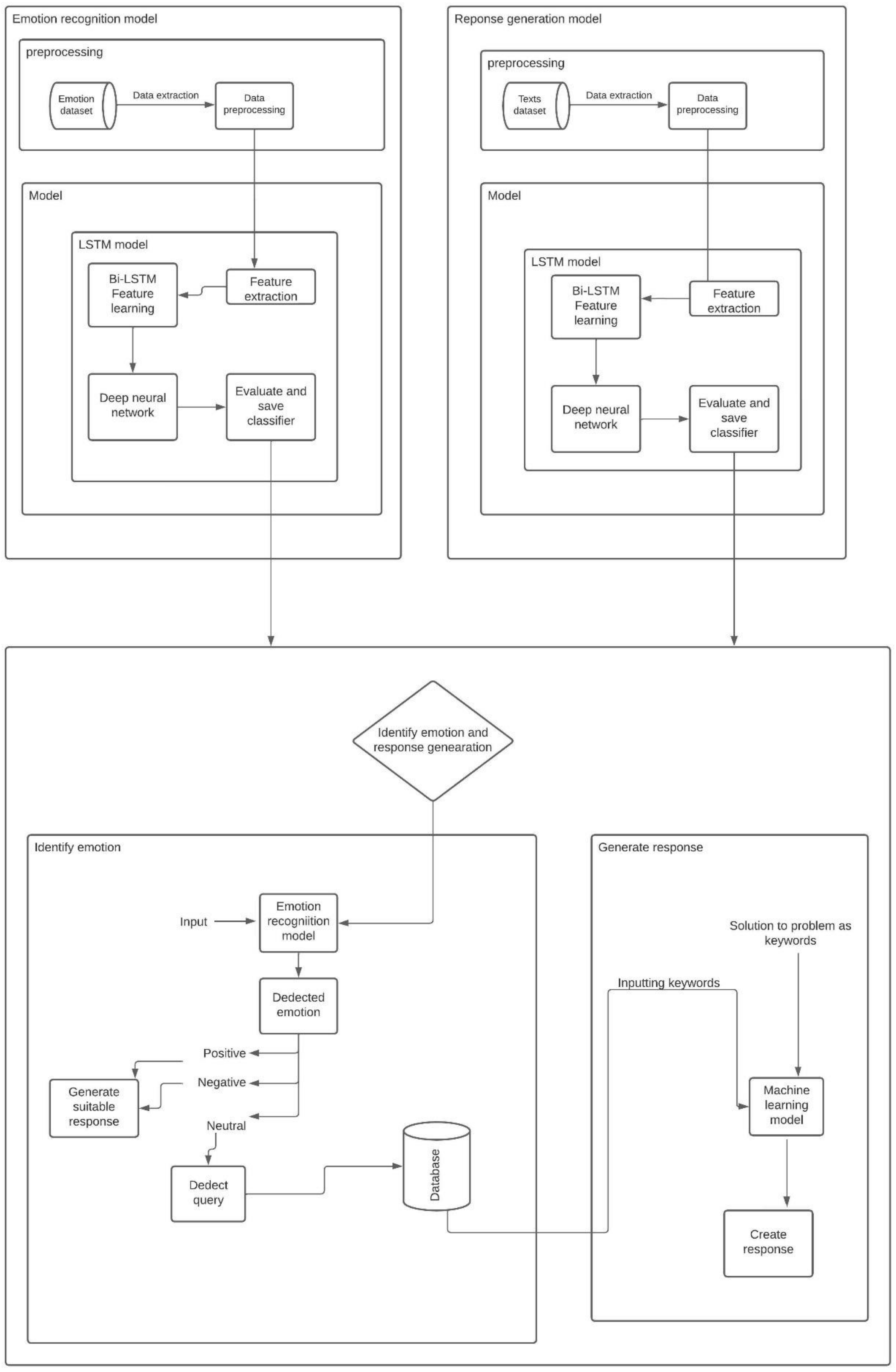
Retrieval-Based Models

This architectural model of a chatbot is easier to build and much more reliable. Though there cannot be 100% accuracy of responses, you can know the possible types of responses and ensure that no inappropriate or incorrect response is delivered by the chatbot.

Retrieval-based models are more in use at the moment. Several algorithms and APIs are readily available for developers to build chatbot on this architectural model. This bot considers the message and context of the conversation to deliver the best response from a predefined list of messages.

Here, in our project we have retrieval-based model for better accuracy and response to the users.

SYSTEM ARCHITECTURE:



The emotion recognition model is trained on Bi-LSTM neural network by using emotion detection dataset, pre-processing, feature extraction, bi-lstm feature learning, deep neural network and then evaluated the classifier and saved the model. Using a retrieval-based response generation from a set of responses given in the storage appropriately.

The input text was given by the user in chat box, then the emotion was deducted and if the emotion predicted to be positive or negative then the corresponding response is generated. If the emotion is neutral which is here is put as database related commands, the request is viewed or updated into or with the database.

DETAILS OF MODULE DESIGN INDICATING THE INTERMEDIATE DELIVERABLES

There are 3 modules in the project. Those are Emotion recognition module, reason finding module and response generation module.

Emotion recognition module:

To find the emotion for an input, the model is feed with many examples sentences with emotion. Hence based on the data provided it predict the emotion like happy, sad, anxiety, fear, etc. as expressed by the user.

Input: Text sentence
Output: Emotions with probability

Response generation module:

Once the segregation of reason is successful, the student issue is found. Each issue will be provided with unique sets of training data of advices. Hence if an issue is predicted, the chatbot will generate valuable response for the issue as output chat.

Input: Emotion and reason from above modules
Output: Response text

Database query module:

Initially as the time table templated into which users can use view and update commands to alter or view the database values.

Input: Chat with db command
Output: Database result or updating acknowledgement

INPUT AND OUTPUT:

Given the input as chat, an appropriate response chat is generated and displayed to the user as output.

The input is feed to chat as text by user. Then the input is pre-processed using different methods like stemming, tokenization and lemmatization, removing stop words, etc.

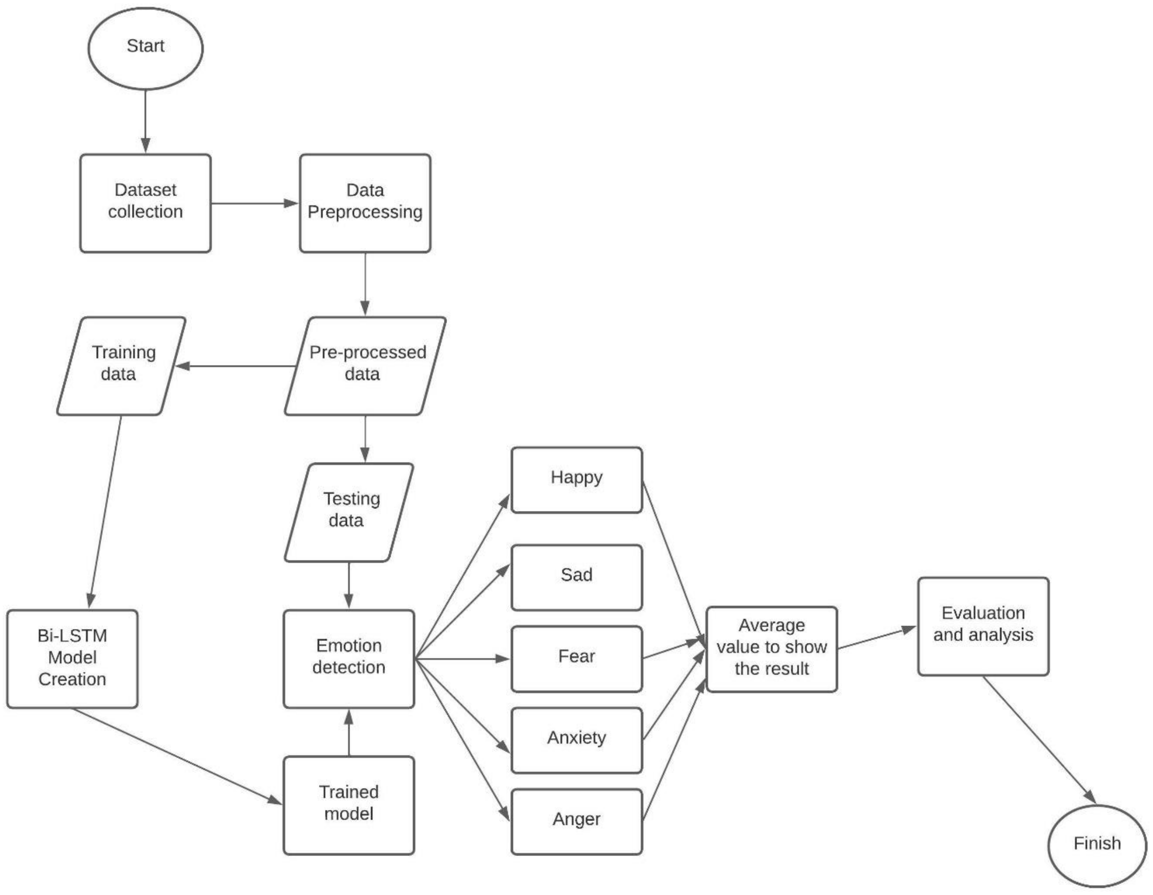
The important texts from the input sentence are stored into an array and sent for training in case of training data and predicted using the model in case of test data.

If the sentence is predicted as positive, then appropriate response is created using response generation module. If predicted as negative, the reason for the negativity is found using the module (by asking the user explicitly if unable to get from the user input) and appropriate response is generated. If the input is predicted as neutral and if it is a query, it is redirected to the database using NLP and result data is responded as chat to user.

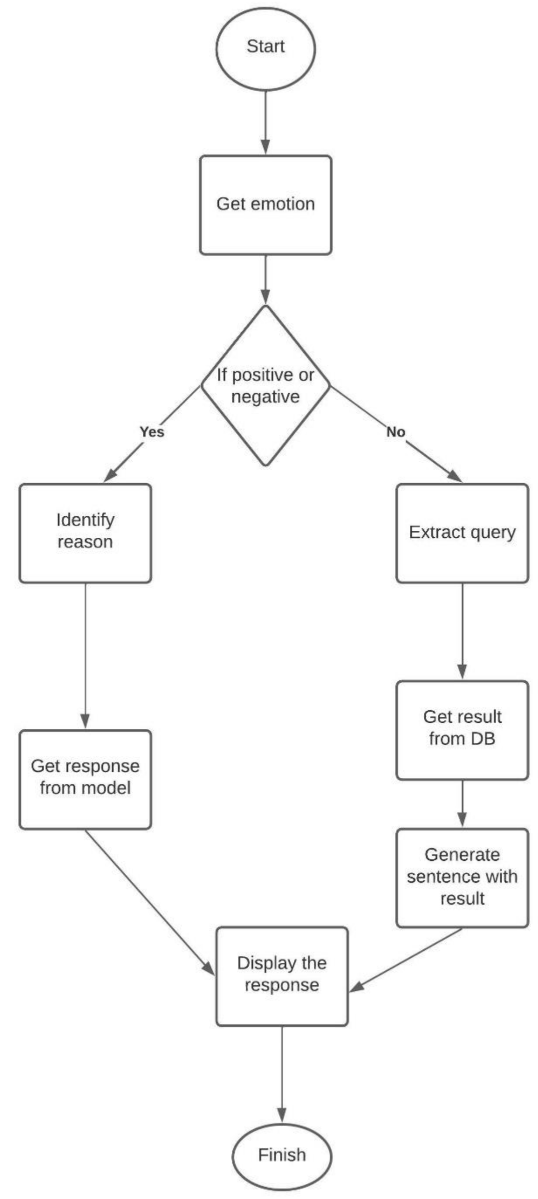
PROCESS FLOWCHART:

The process flowchart is below:

Emotion recognition flow chart

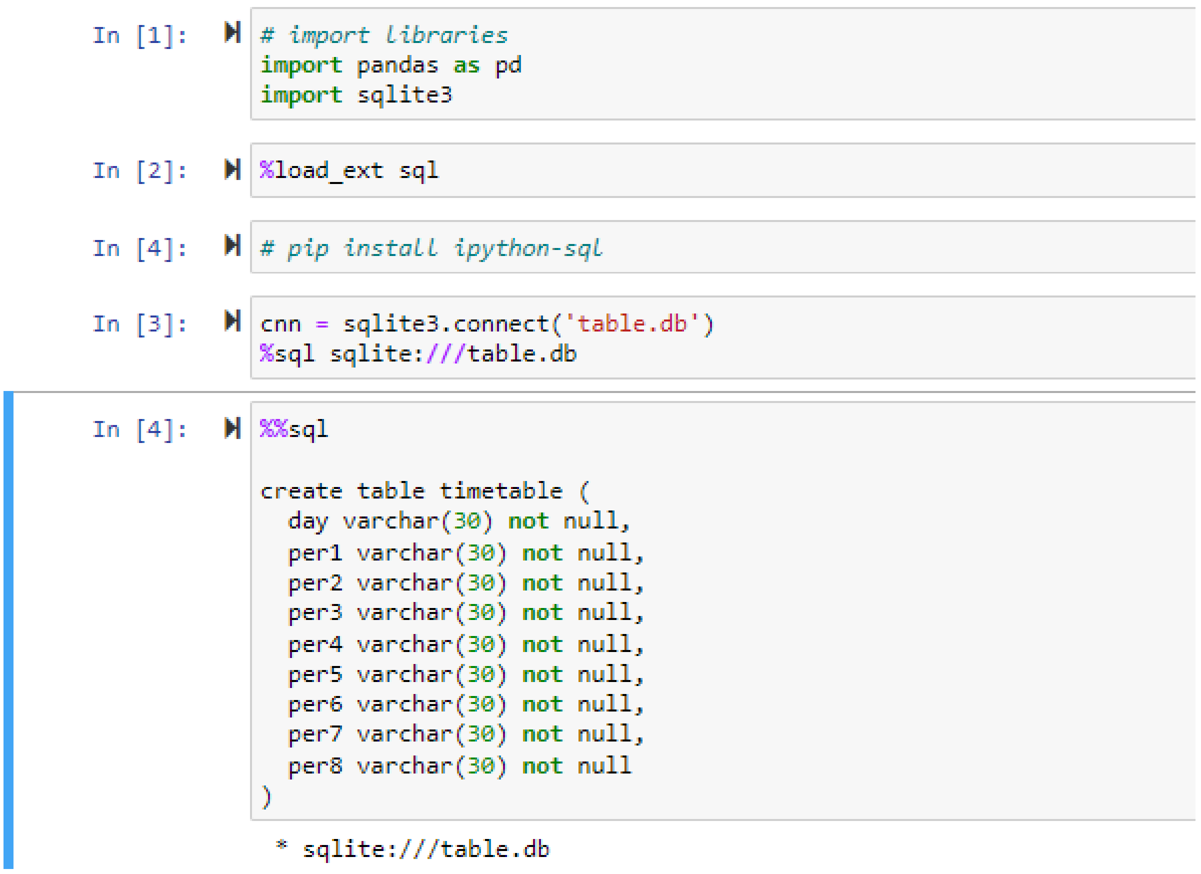


Response generation flow chart



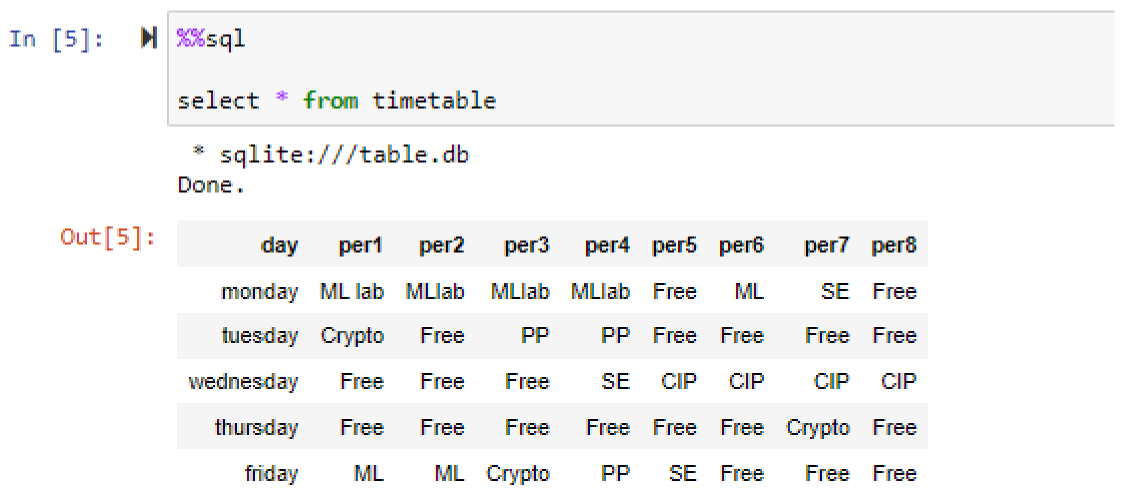
IMPLEMENTATION DETAILS – 80% AND CORRESPONDING DETAILS/SNAPSHOTS

INTEGRATING ALL THE MODULES WITH FEW MODIFICATION



```
In [1]:   # import libraries
          import pandas as pd
          import sqlite3

In [2]:   %load_ext sql

In [4]:   # pip install ipython-sql

In [3]:   cnn = sqlite3.connect('table.db')
          %sql sqlite:///table.db

In [4]:   %%sql

          create table timetable (
            day varchar(30) not null,
            per1 varchar(30) not null,
            per2 varchar(30) not null,
            per3 varchar(30) not null,
            per4 varchar(30) not null,
            per5 varchar(30) not null,
            per6 varchar(30) not null,
            per7 varchar(30) not null,
            per8 varchar(30) not null
          )

            * sqlite:///table.db
```

Setting up the libraries for database query, connecting to the database which contains the table db. Creating the time table and its initiated data for sample.

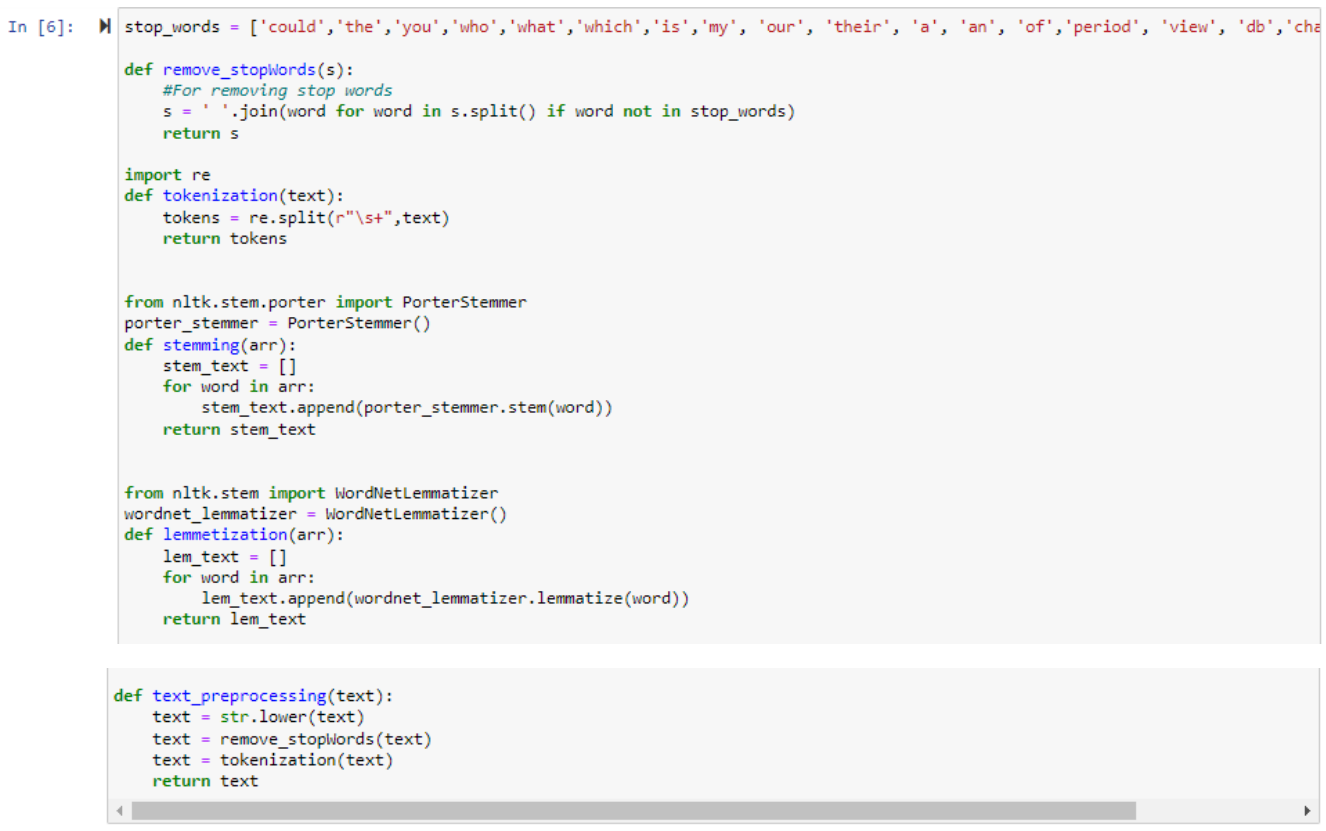


```
In [5]:  %%sql

         select * from timetable

          * sqlite:///table.db
         Done.
```

Out[5]:

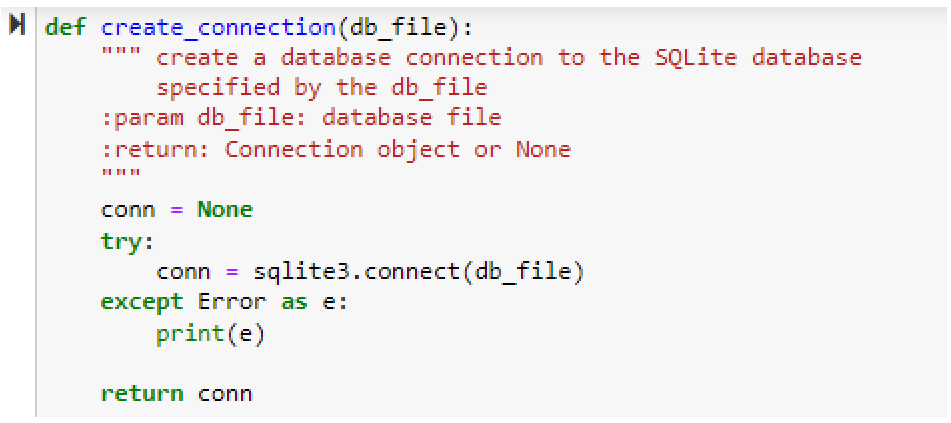
day	per1	per2	per3	per4	per5	per6	per7	per8
monday	ML lab	MLlab	MLlab	MLlab	Free	ML	SE	Free
tuesday	Crypto	Free	PP	PP	Free	Free	Free	Free
wednesday	Free	Free	Free	SE	CIP	CIP	CIP	CIP
thursday	Free	Free	Free	Free	Free	Free	Crypto	Free
friday	ML	ML	Crypto	PP	SE	Free	Free	Free

Viewing the time table data



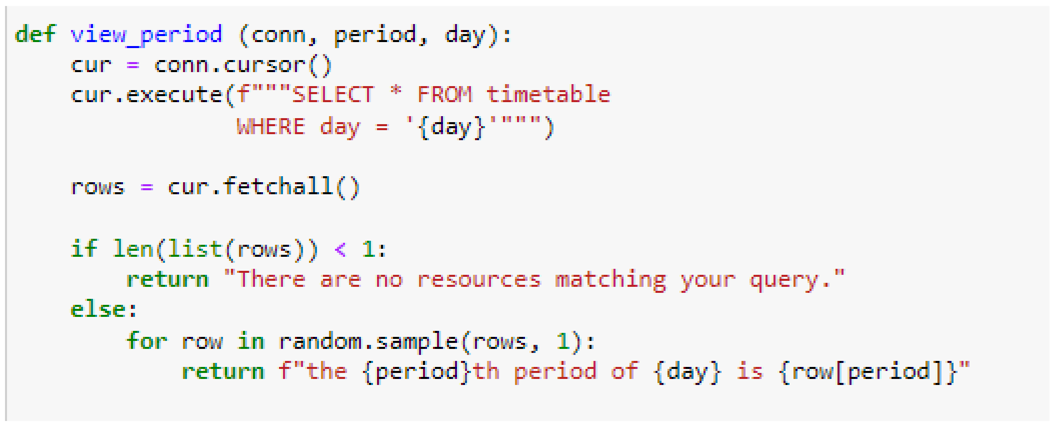
```
In [6]:  stop_words = ['could','the','you','who','what','which','is','my', 'our', 'their', 'a', 'an', 'of','period', 'view', 'db','cha

         def remove_stopWords(s):
             #For removing stop words
             s = ' '.join(word for word in s.split() if word not in stop_words)
             return s

         import re
         def tokenization(text):
             tokens = re.split(r"\s+",text)
             return tokens


         from nltk.stem.porter import PorterStemmer
         porter_stemmer = PorterStemmer()
         def stemming(arr):
             stem_text = []
             for word in arr:
                 stem_text.append(porter_stemmer.stem(word))
             return stem_text


         from nltk.stem import WordNetLemmatizer
         wordnet_lemmatizer = WordNetLemmatizer()
         def lemmetization(arr):
             lem_text = []
             for word in arr:
                 lem_text.append(wordnet_lemmatizer.lemmatize(word))
             return lem_text

        def text_preprocessing(text):
            text = str.lower(text)
            text = remove_stopWords(text)
            text = tokenization(text)
            return text
```

Pre-processing the input data which the database query gets from the user chat which includes removing stop words and tokenization of string into array of words.



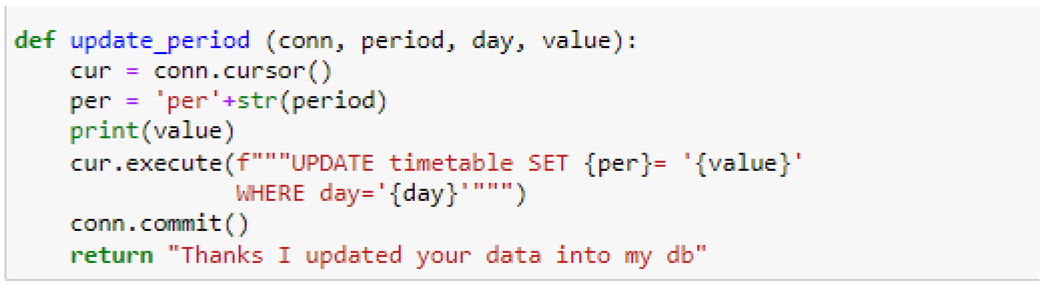
```python
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by the db_file
    :param db_file: database file
    :return: Connection object or None
    """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
    except Error as e:
        print(e)

    return conn
```

Creating connection with the required database



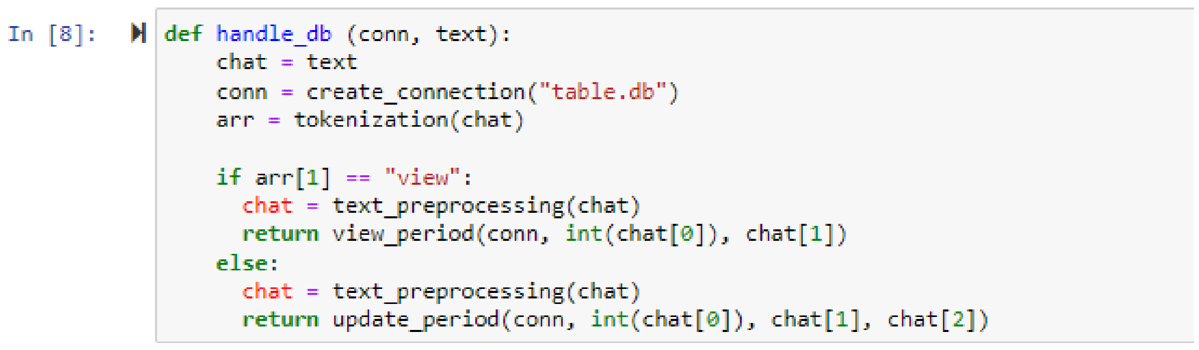
```python
def view_period (conn, period, day):
    cur = conn.cursor()
    cur.execute(f"""SELECT * FROM timetable
                WHERE day = '{day}'""")

    rows = cur.fetchall()

    if len(list(rows)) < 1:
        return "There are no resources matching your query."
    else:
        for row in random.sample(rows, 1):
            return f"the {period}th period of {day} is {row[period]}"
```

Function to view the period in the time table of given period and day



```python
def update_period (conn, period, day, value):
    cur = conn.cursor()
    per = 'per'+str(period)
    print(value)
    cur.execute(f"""UPDATE timetable SET {per}= '{value}'
                WHERE day='{day}'""")
    conn.commit()
    return "Thanks I updated your data into my db"
```

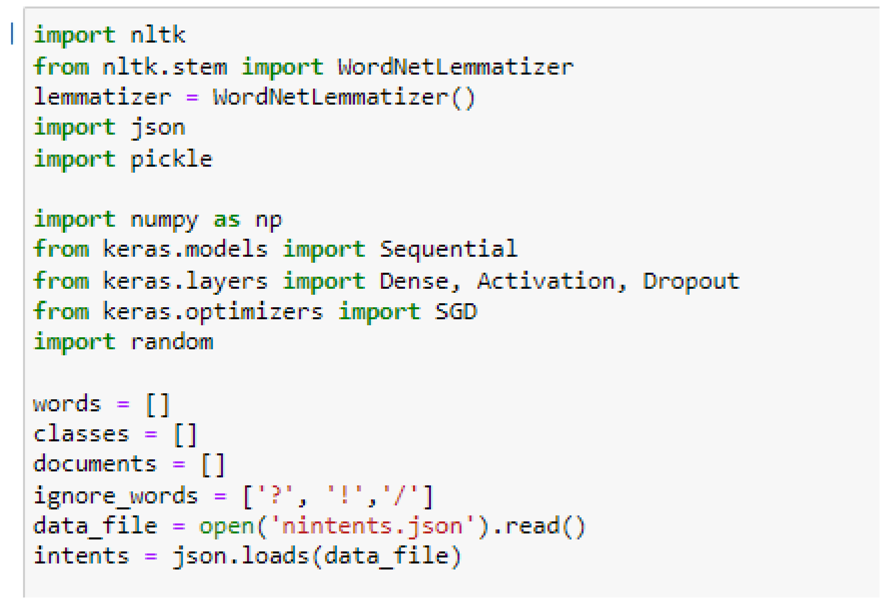
Updating the period of given index of period as number, day and to which period it is needed to be updated



```
In [8]:     def handle_db (conn, text):
                chat = text
                conn = create_connection("table.db")
                arr = tokenization(chat)

                if arr[1] == "view":
                  chat = text_preprocessing(chat)
                  return view_period(conn, int(chat[0]), chat[1])
                else:
                  chat = text_preprocessing(chat)
                  return update_period(conn, int(chat[0]), chat[1], chat[2])
```

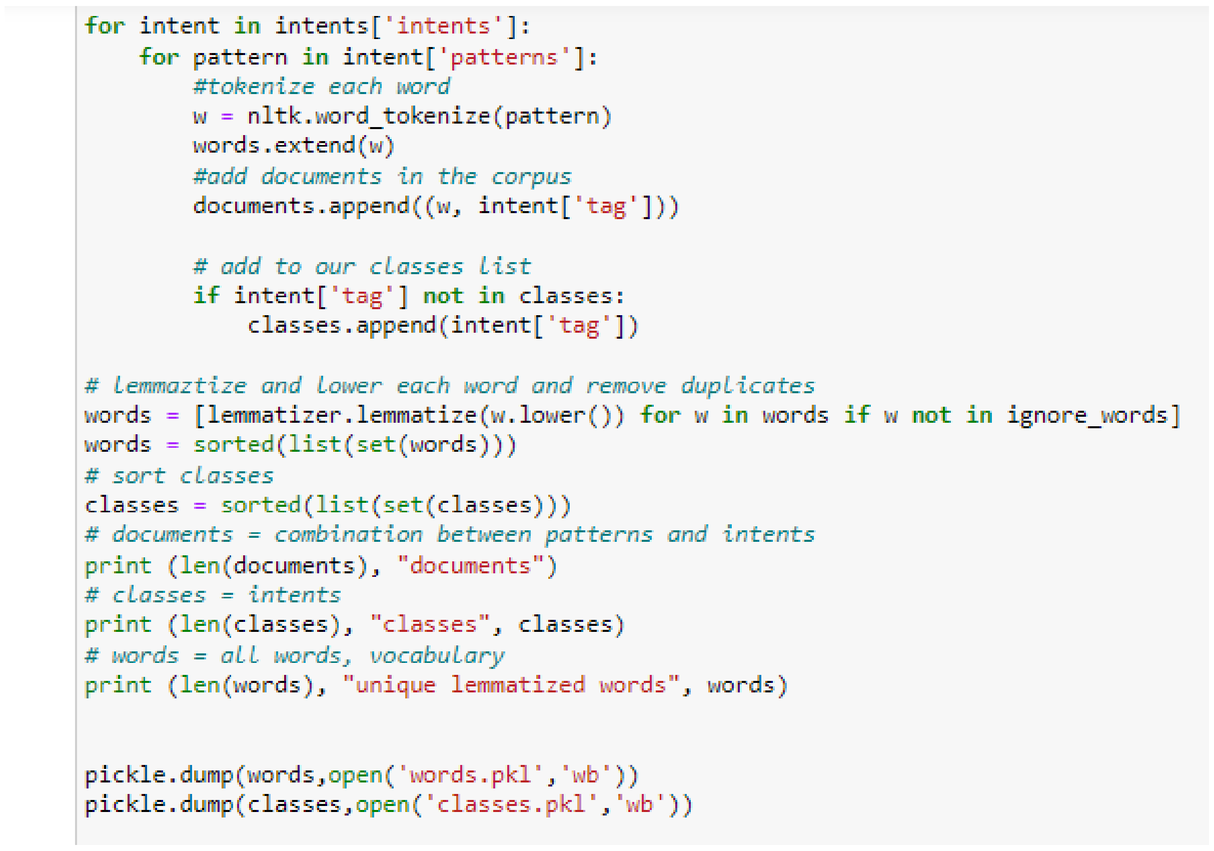
Database function to handle the request from emotion recognition module, if the query is neutral here DB related

Training the models

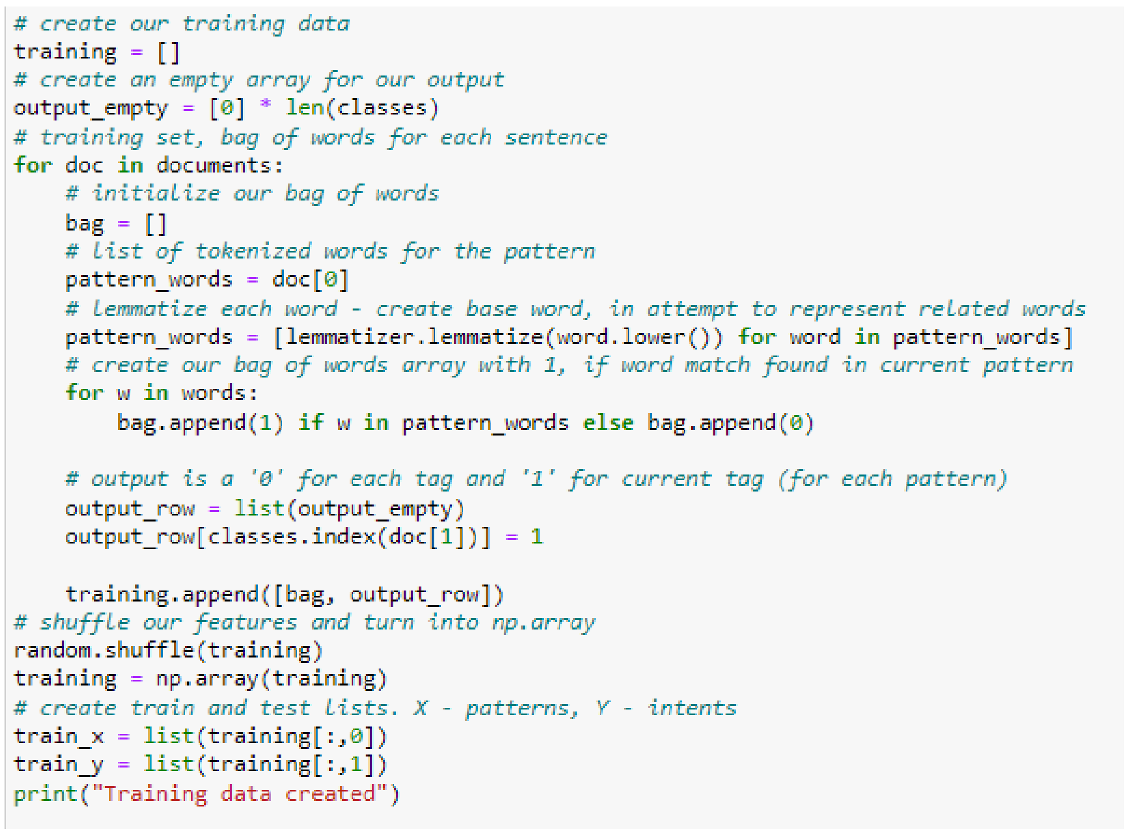


```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random

words = []
classes = []
documents = []
ignore_words = ['?', '!','/']
data_file = open('nintents.json').read()
intents = json.loads(data_file)
```

Importing the required libraries and initialising few variables and opening the intent.json file which contains the datasets.



```python
for intent in intents['intents']:
    for pattern in intent['patterns']:
        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        #add documents in the corpus
        documents.append((w, intent['tag']))

        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(documents), "documents")
# classes = intents
print (len(classes), "classes", classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)


pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```



```python
# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # lemmatize each word - create base word, in attempt to represent related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    # create our bag of words array with 1, if word match found in current pattern
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag (for each pattern)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
print("Training data created")
```

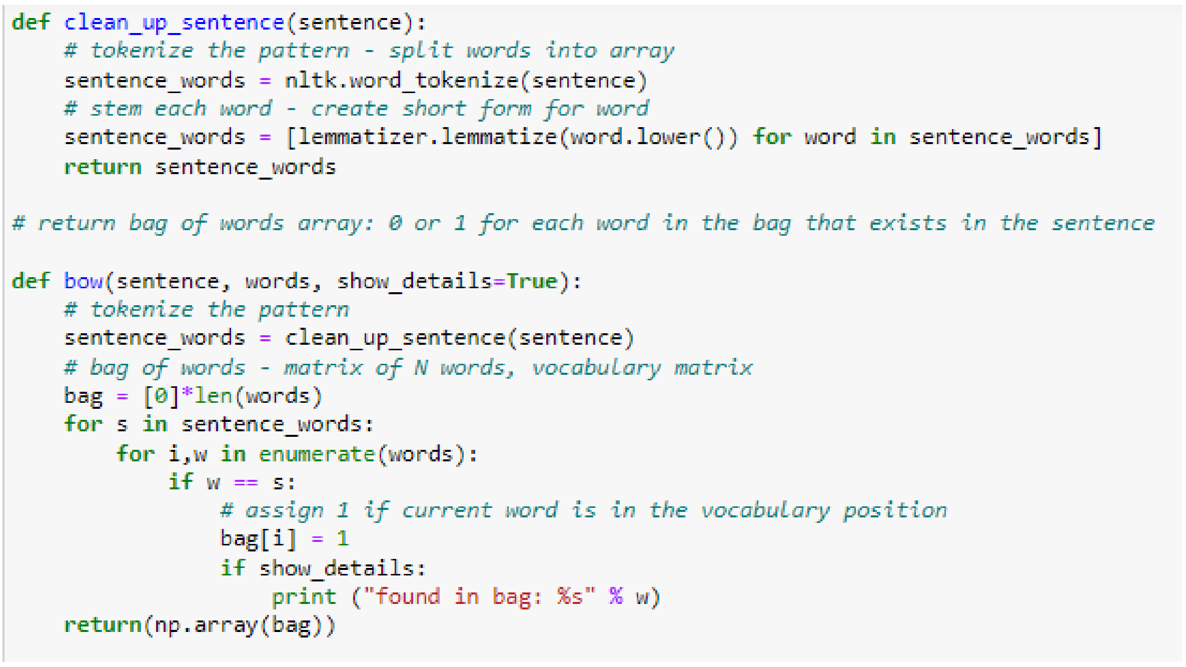
Processing the dataset in intent and creating trainable x, y values by pre-processing in the training texts. Then saving the pre-processed training data in easily retrievable format.

```
model = Sequential()
model.add(Embedding(input_dim=len(train_x[0]), output_dim = 100, trainable = False))
model.add(Dropout(0.4))
model.add(Bidirectional(LSTM(100,return_sequences=True)))
model.add(Dropout(0.4))
model.add(Bidirectional(LSTM(100,return_sequences=False)))
model.add(Dense(len(train_y[0]), activation = 'softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model
# sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
# model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam',metrics = ['accuracy'])

#fitting and saving the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('drive/MyDrive/Colab Notebooks/CIP/ours - Copy/chatbot_model3.h5', hist)

print("model created")

import matplotlib.pyplot as plt
plt.plot(hist.history['accuracy'],c='b',label='train accuracy')
plt.legend(loc='lower right')
plt.show()
```

GUI

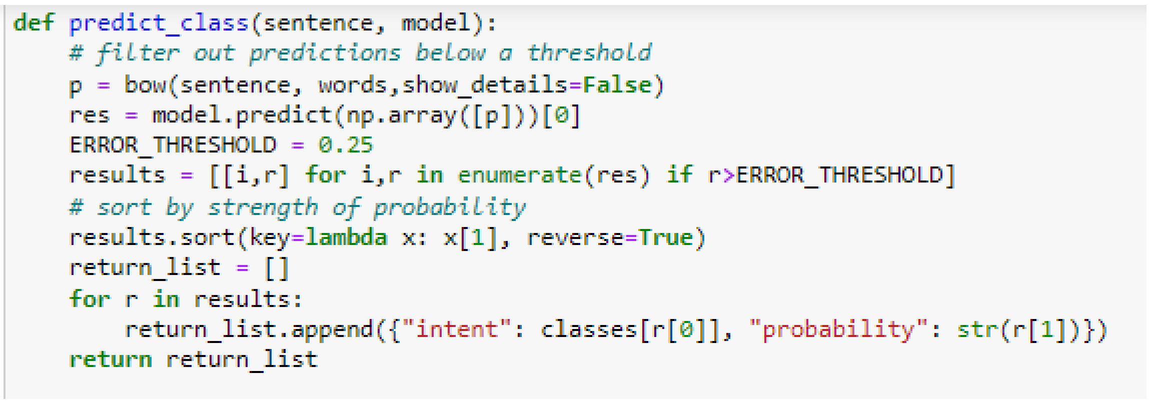
```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np

from keras.models import load_model
model = load_model('chatbot_model.h5')
import json
import random

intents = json.loads(open('nintents.json').read())
words = pickle.load(open('words.pkl','rb'))
classes = pickle.load(open('classes.pkl','rb'))
```

Importing required libraries and loading the model and required pre-processed training data set.



```python
def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))
```

Cleaning the text which the GUI texts from the user and pre-processing it with the above functions. bow() function returns an array of 0s and 1s which shows whether the given word in input is present in the bag of words.



```python
def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list
```

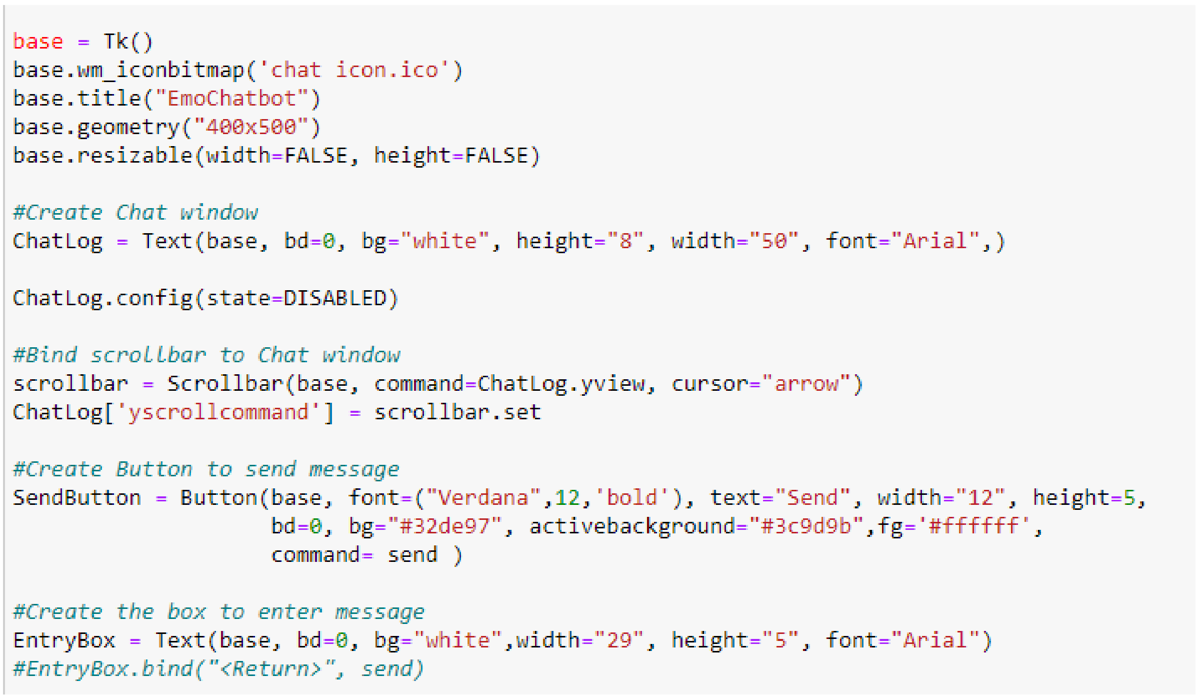
Function to predict the class which uses the loaded model feeding the input from bow() function and getting the results of class names which has less error threshold and sorting this array and showing the first result as output which contains the intent/predicted output and probability of it.

```python
#Creating GUI with tkinter
import tkinter
from tkinter import *


def send():
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)

    if msg != '':
        ChatLog.config(state=NORMAL)
        ChatLog.insert(END, "You: " + msg + '\n\n')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

        res = chatbot_response(msg)
        ChatLog.insert(END, "Bot: " + res + '\n\n')

        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)
```

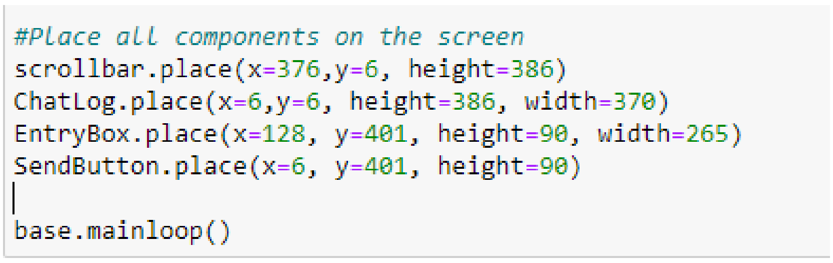
Creating tkinter window for gui with fonts and view properties set. Getting input from user and getting response from chatbot_response functions into res variable and rendering the output to the user.

```python
def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result

def chatbot_response(msg):
    arr = tokenization(msg)
    if(arr[0] == "db"):
        return handle_db(create_connection('table.db'), msg)

    ints = predict_class(msg, model)

    print("ints: ",ints)
    if(len(ints) == 0):
        return "Retry again please!"
    res = getResponse(ints, intents)
    return res
```

Function chatbot_response is the function which respond to the user input, this identifies whether the chat is database related if yes redirect the response else proceed to predict_class() function. If it does not match to any of the functions.

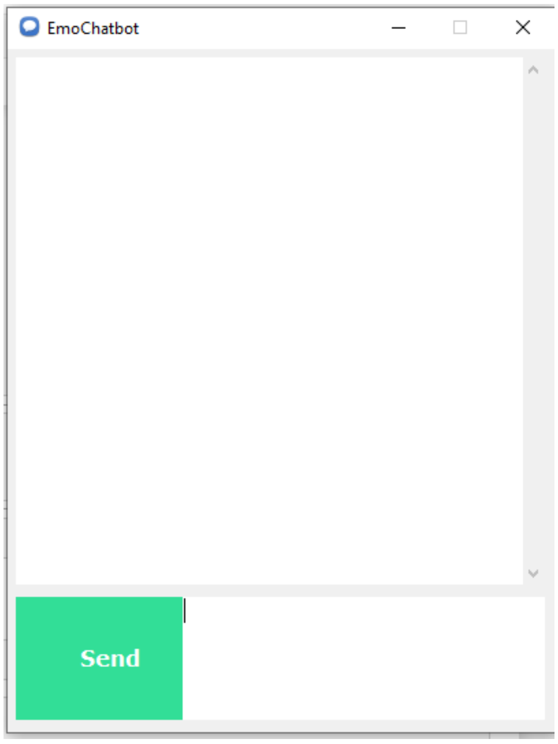


```python
base = Tk()
base.wm_iconbitmap('chat icon.ico')
base.title("EmoChatbot")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="arrow")
ChatLog['yscrollcommand'] = scrollbar.set

#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)
```

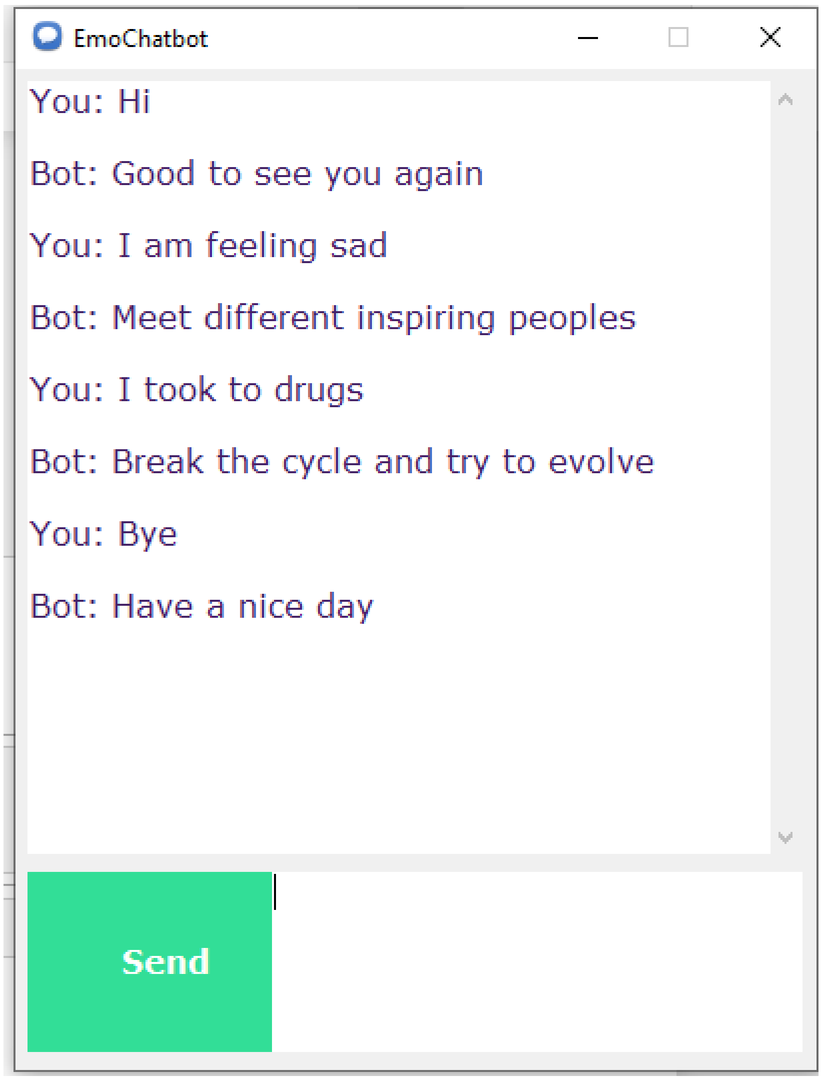


```python
#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```

Setting the properties of gui window and customizing it.



The left side window is the GUI of the emotional chatbot. In the below text box need to enter the input and press the Send button.



EmoChatbot

You: Hi

Bot: Good to see you again

You: I am feeling sad

Bot: Meet different inspiring peoples

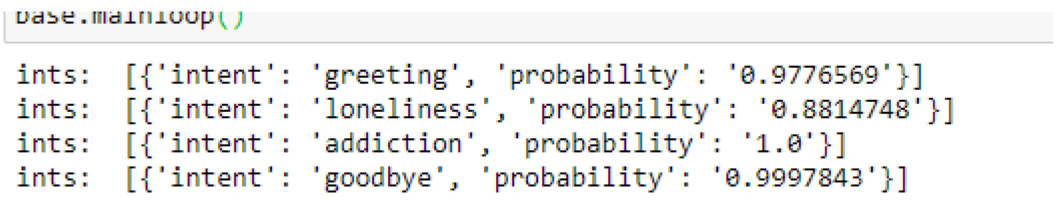
You: I took to drugs

Bot: Break the cycle and try to evolve

You: Bye

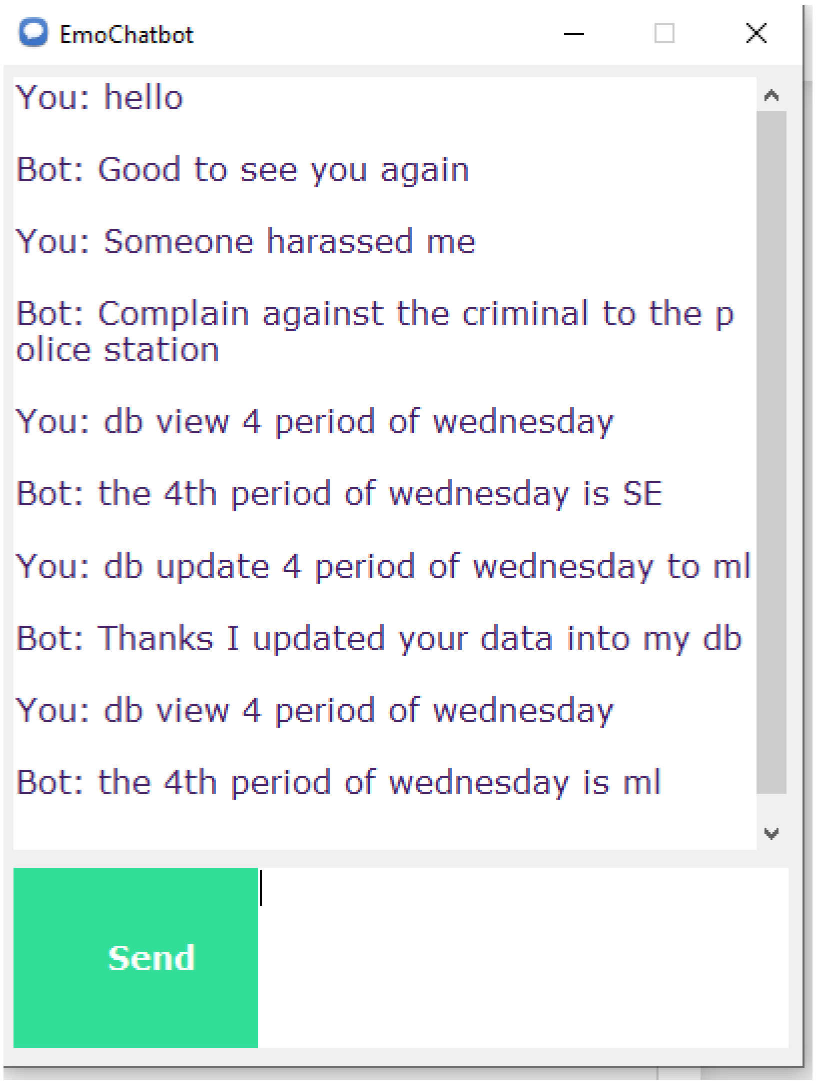
Bot: Have a nice day

Sample input and output



```
base.mainloop()

ints:  [{'intent': 'greeting', 'probability': '0.9776569'}]
ints:  [{'intent': 'loneliness', 'probability': '0.8814748'}]
ints:  [{'intent': 'addiction', 'probability': '1.0'}]
ints:  [{'intent': 'goodbye', 'probability': '0.9997843'}]
```

Backend showing intents which were deducted


EmoChatbot

You: hello

Bot: Good to see you again

You: Someone harassed me

Bot: Complain against the criminal to the p
olice station

You: db view 4 period of wednesday

Bot: the 4th period of wednesday is SE

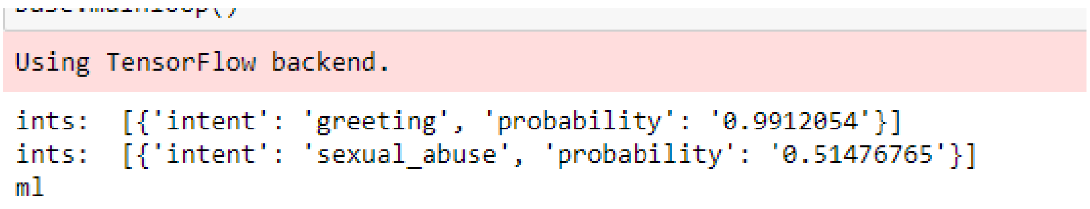
You: db update 4 period of wednesday to ml

Bot: Thanks I updated your data into my db

You: db view 4 period of wednesday

Bot: the 4th period of wednesday is ml

Normal chat with database querying chats

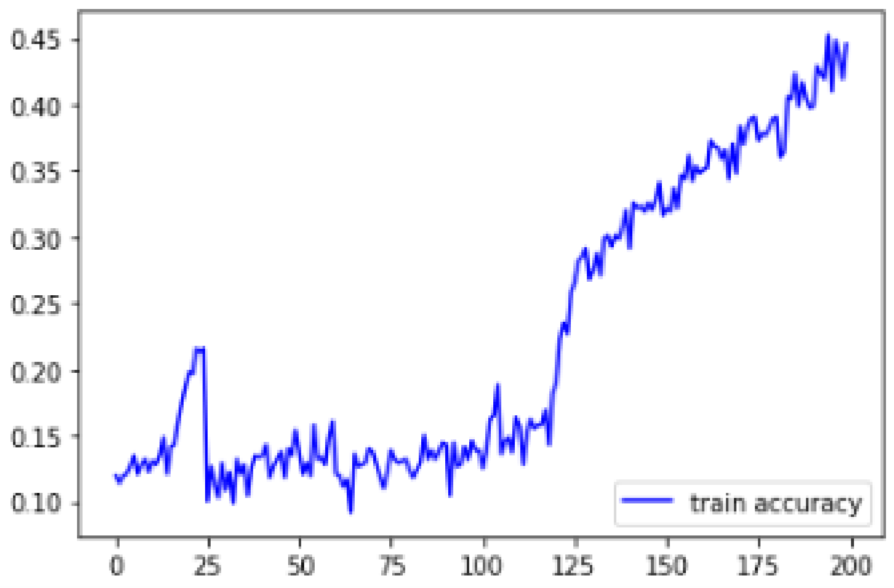


```
Using TensorFlow backend.

ints:  [{'intent': 'greeting', 'probability': '0.9912054'}]
ints:  [{'intent': 'sexual_abuse', 'probability': '0.51476765'}]
ml
```

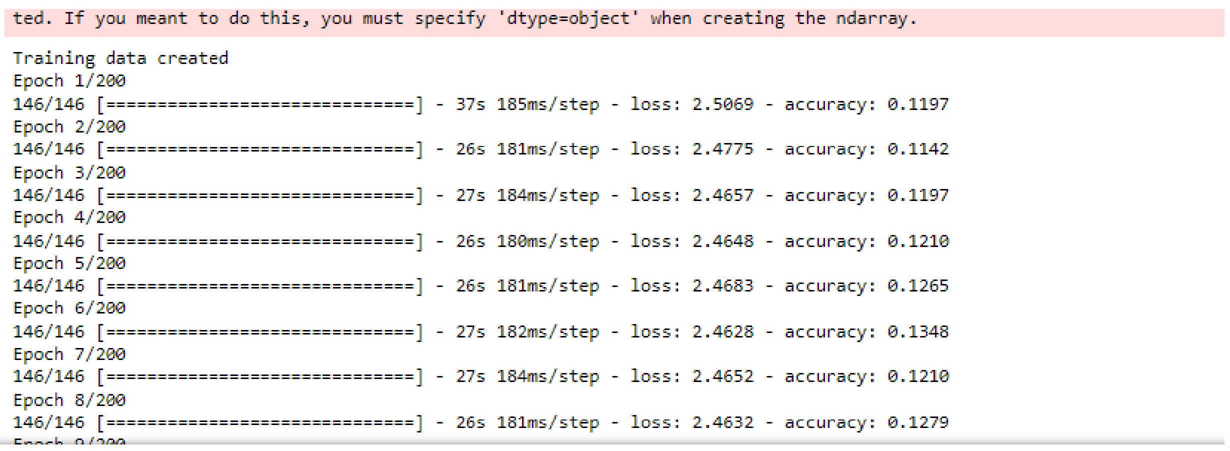
Corresponding intents predicted in backend

METRICS OF EVALUATION:

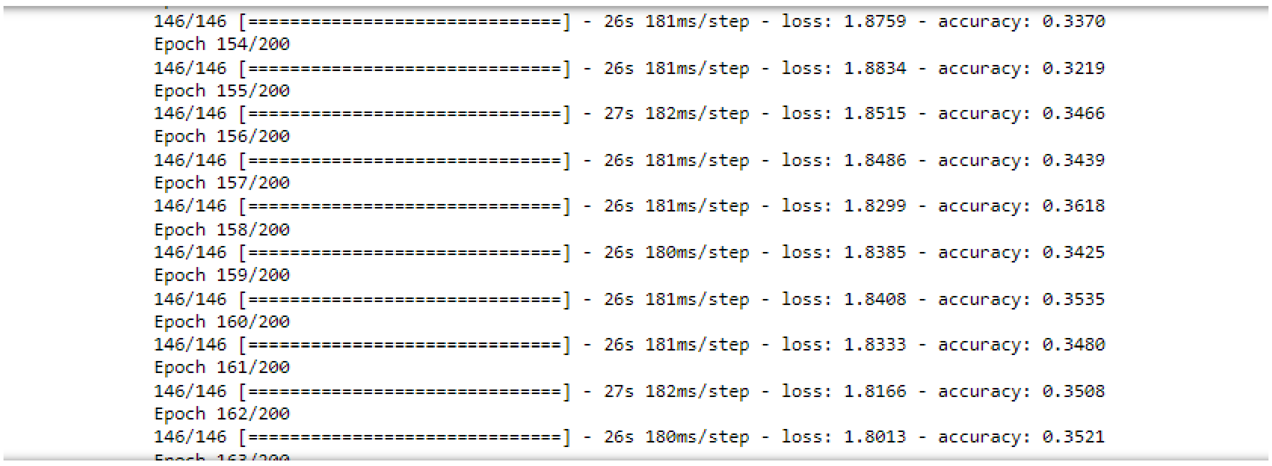
```
model created
```



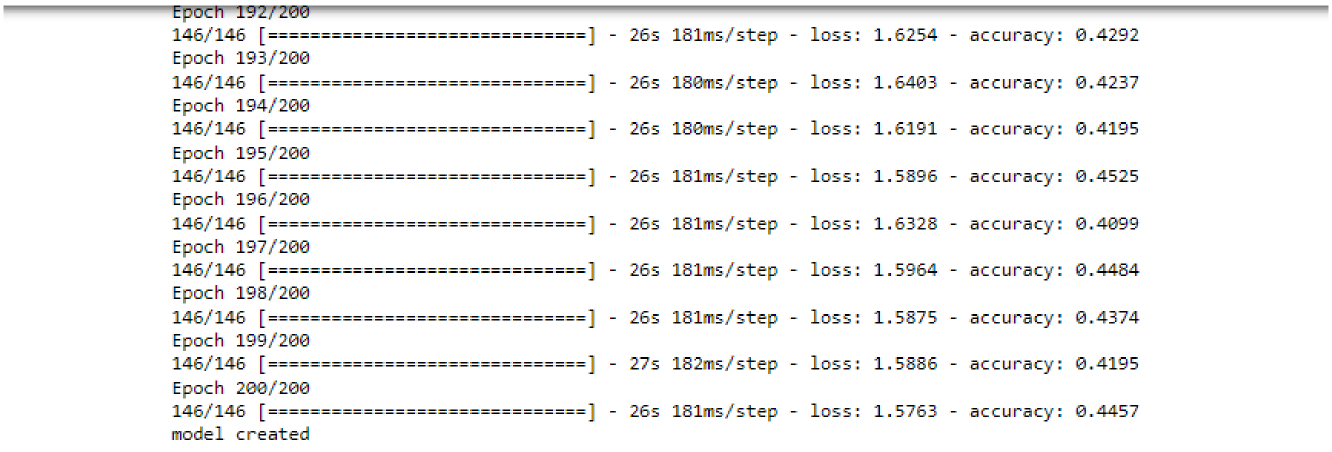
Model accuracy graph



```
ted. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.

Training data created
Epoch 1/200
146/146 [==============================] - 37s 185ms/step - loss: 2.5069 - accuracy: 0.1197
Epoch 2/200
146/146 [==============================] - 26s 181ms/step - loss: 2.4775 - accuracy: 0.1142
Epoch 3/200
146/146 [==============================] - 27s 184ms/step - loss: 2.4657 - accuracy: 0.1197
Epoch 4/200
146/146 [==============================] - 26s 180ms/step - loss: 2.4648 - accuracy: 0.1210
Epoch 5/200
146/146 [==============================] - 26s 181ms/step - loss: 2.4683 - accuracy: 0.1265
Epoch 6/200
146/146 [==============================] - 27s 182ms/step - loss: 2.4628 - accuracy: 0.1348
Epoch 7/200
146/146 [==============================] - 27s 184ms/step - loss: 2.4652 - accuracy: 0.1210
Epoch 8/200
146/146 [==============================] - 26s 181ms/step - loss: 2.4632 - accuracy: 0.1279
Epoch 9/200
```

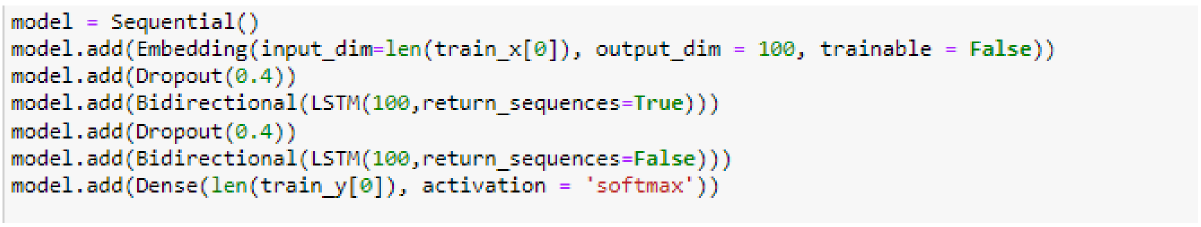


```
146/146 [==============================] - 26s 181ms/step - loss: 1.8759 - accuracy: 0.3370
Epoch 154/200
146/146 [==============================] - 26s 181ms/step - loss: 1.8834 - accuracy: 0.3219
Epoch 155/200
146/146 [==============================] - 27s 182ms/step - loss: 1.8515 - accuracy: 0.3466
Epoch 156/200
146/146 [==============================] - 26s 181ms/step - loss: 1.8486 - accuracy: 0.3439
Epoch 157/200
146/146 [==============================] - 26s 181ms/step - loss: 1.8299 - accuracy: 0.3618
Epoch 158/200
146/146 [==============================] - 26s 180ms/step - loss: 1.8385 - accuracy: 0.3425
Epoch 159/200
146/146 [==============================] - 26s 181ms/step - loss: 1.8408 - accuracy: 0.3535
Epoch 160/200
146/146 [==============================] - 26s 181ms/step - loss: 1.8333 - accuracy: 0.3480
Epoch 161/200
146/146 [==============================] - 27s 182ms/step - loss: 1.8166 - accuracy: 0.3508
Epoch 162/200
146/146 [==============================] - 26s 180ms/step - loss: 1.8013 - accuracy: 0.3521
Epoch 163/200
```



```
Epoch 192/200
146/146 [==============================] - 26s 181ms/step - loss: 1.6254 - accuracy: 0.4292
Epoch 193/200
146/146 [==============================] - 26s 180ms/step - loss: 1.6403 - accuracy: 0.4237
Epoch 194/200
146/146 [==============================] - 26s 180ms/step - loss: 1.6191 - accuracy: 0.4195
Epoch 195/200
146/146 [==============================] - 26s 181ms/step - loss: 1.5896 - accuracy: 0.4525
Epoch 196/200
146/146 [==============================] - 26s 181ms/step - loss: 1.6328 - accuracy: 0.4099
Epoch 197/200
146/146 [==============================] - 26s 181ms/step - loss: 1.5964 - accuracy: 0.4484
Epoch 198/200
146/146 [==============================] - 26s 181ms/step - loss: 1.5875 - accuracy: 0.4374
Epoch 199/200
146/146 [==============================] - 27s 182ms/step - loss: 1.5886 - accuracy: 0.4195
Epoch 200/200
146/146 [==============================] - 26s 181ms/step - loss: 1.5763 - accuracy: 0.4457
model created
```

Model details:



```
model = Sequential()
model.add(Embedding(input_dim=len(train_x[0]), output_dim = 100, trainable = False))
model.add(Dropout(0.4))
model.add(Bidirectional(LSTM(100,return_sequences=True)))
model.add(Dropout(0.4))
model.add(Bidirectional(LSTM(100,return_sequences=False)))
model.add(Dense(len(train_y[0]), activation = 'softmax'))
```

Sequential model having bi-lstm layers for effective learning in both directions.
Accuracy => 44.5%

ALL POSSIBLE TEST CASE (TABULAR FORM) AND VALIDATION

Input text	Replied text	Expected emotion	Returned emotion	Probability
Hi hello	Good to see you again	Greeting	Greeting	0.99
Hi	Hi there, how can I help?	Greeting	Greeting	0.97
I am addicted to games	Break the cycle and try to evolve	Addiction	Addiction	1.0
I was mocked of being black	Prove others that you are more than a body	Body Shaming	Body Shaming	0.76
My friends are fighting with each other	Communicate the reasons for conflict from each side to understand and resolve the problem	Family issue	Family issue	1.0
i am feeling not good	Consult to a doctor immediately	Physical ill	Physical ill	0.99
I am not good with the marks	Try to First aid if possible	Less competent	Physical ill	0.54

LIST OF REFERENCES IN STANDARD IEEE FORMAT: (BASE PAPER REFERENCES, CROSS REFERENCES, ADDITIONAL REFERENCES)

REFERENCES:

1) Dhanasekar, V., Preethi, Y., Vishali, S., & IR, P. J. (2021, September). A Chatbot to promote Students Mental Health through Emotion Recognition. In *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)* (pp. 1412-1416). IEEE.

2) Karna, M., Juliet, D. S., & Joy, R. C. (2020, June). Deep learning based text emotion recognition for chatbot applications. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)* (pp. 988-993). IEEE.

3) Lee, D., Oh, K. J., & Choi, H. J. (2017, February). The chatbot feels you-a counseling service using emotional response generation. In *2017 IEEE international conference on big data and smart computing (BigComp)* (pp. 437-440). IEEE.