Code Overview and Project Report

Project Name: Mental Health Assistant Chatbot

Introduction

Chatbots have become increasingly important in the business world as they provide automated customer support and assistance. This industry-level chatbot project aims to demonstrate how to build a functional chatbot using Python and natural language processing libraries. In this report, we will provide an overview of the code and its functionality.

Code Overview

The chatbot project is implemented in Python and relies on several libraries, including NLTK, Keras, Tkinter. Here is an overview of the key components and functions in the code:

1. **Data Preparation**

The project begins by importing necessary libraries and initializing variables such as words, classes, documents, and ignore_words.

It reads the intent data from an intents.json file. This JSON file contains predefined intents and responses for the chatbot.

```
In [1]: import nltk

In [2]: from nltk.stem import WordNetLemmatizer
        lemmatizer = WordNetLemmatizer()
        import json
        import pickle

In [3]: import numpy as np
        import keras
        from keras.models import Sequential
        from keras.layers import Dense, Activation, Dropout
        from keras.optimizers import SGD
        import random
        import warnings
        warnings.filterwarnings('ignore')

In [4]: words=[]
        classes = []
        documents = []
        ignore_words = ['?', '!']
        data_file = open('intents.json').read()
        intents = json.loads(data_file)
```

2. Text Preprocessing

NLTK's tokenization and lemmatization techniques are used to preprocess the text data. Tokenization splits sentences into words, and lemmatization reduces words to their base or root forms.

The code compiles a list of unique lemmatized words and classes (intents) found in the dataset.

In [7]:
```
nltk.download('all')
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'])
```

In [8]:
```
# lemmatize, 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)
```

```
58 documents
10 classes ['Social_anxiety', 'adddiction_remedy', 'addiction_causes', 'goodbye', 'greetings', 'grief_loss_stress', 'headache_i
ssues', 'name', 'options', 'sleep']
103 unique lemmatized words ['.', 'a', 'abuse', 'according', 'adddiction', 'addiction', 'again', 'alcohol', 'am', 'anxious', 'a
re', 'around', 'at', 'be', 'bye', 'can', 'cause', 'chatbot', 'cigarette', 'concentrate', 'consumption', 'cope', 'could', 'cur
e', 'curing', 'day', 'disturbed', 'divorced', 'do', 'dont', 'drink', 'drug', 'during', 'early', 'evening', 'feel', 'feeling',
'focus', 'for', 'from', 'get', 'good', 'goodbye', 'have', 'headache', 'hello', 'help', 'helpful', 'hi', 'how', 'i', 'im', 'in',
'insecure', 'is', 'later', 'leaving', 'lonely', 'lost', 'lot', 'me', 'middle', 'mom/dad/parents', 'morning', 'my', 'name', 'neg
ative', 'nice', 'night', 'not', 'of', 'offered', 'pain', 'people', 'provide', 'quit', 'recently', 'see', 'sleep', 'sleeping',
'sleepless', 'smoke', 'smoking', 'suffering', 'support', 'suudenly', 'talk', 'the', 'there', 'thought', 'to', 'treatment', 'una
ble', 'up', 'wake', 'way', 'weddings/parties/events', 'what', 'whats', 'while', 'with', 'you', 'your']
```

3. Data Serialization

The unique words and classes are serialized using the pickle library and saved as words.pkl and classes.pkl. These serialized files will be used for model training and response generation.

In [9]:
```
pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```

4. Training Data Creation

The code creates a training dataset by iterating through the intents and patterns defined in the intents.json file. For each pattern, it tokenizes, lemmatizes, and converts the words into a bag-of-words representation. The bag-of-words representation is paired with the corresponding intent as a training sample.The resulting dataset is shuffled for better training results.

```
In [10]: # Create our training data
training_data = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence
for doc in documents:
    # 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 the current pattern
    bag = [1 if w in pattern_words else 0 for w in words]

    # 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_data.append([bag, output_row])

# Shuffle our features
random.shuffle(training_data)

# Separate features (X) and labels (Y)
train_x = np.array([item[0] for item in training_data])
train_y = np.array([item[1] for item in training_data])

print("Training data created")

Training data created
```

```
In [11]: print(train_x.shape)

(58, 103)
```

5. Neural Network Model

A neural network model is constructed using the Keras library. The model has an input layer with the same number of neurons as there are unique words.

The model includes hidden layers with dropout to prevent overfitting and an output layer with neurons corresponding to the number of unique intents.

The model is compiled using categorical cross-entropy as the loss function and stochastic gradient descent (SGD) as the optimizer.

6. Training the Model

The model is trained on the prepared training dataset with a specified number of epochs and batch size.

The training process aims to optimize the model's ability to classify user inputs into predefined intents.

In [12]:
```
from keras.layers import Dense, Dropout
# Assuming your input data has 103 features
input_shape = (103,)
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of
# neurons equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(128, input_shape=input_shape, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
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(learning_rate=0.01, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, 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('chatbot_model.h5', hist)

print("model created")
```

```
12/12 [==============================] - 0s 3ms/step - loss: 0.0209 - accuracy: 1.0000
Epoch 192/200
12/12 [==============================] - 0s 4ms/step - loss: 0.0131 - accuracy: 1.0000
Epoch 193/200
12/12 [==============================] - 0s 3ms/step - loss: 0.0407 - accuracy: 0.9828
Epoch 194/200
12/12 [==============================] - 0s 3ms/step - loss: 0.0317 - accuracy: 1.0000
Epoch 195/200
12/12 [==============================] - 0s 3ms/step - loss: 0.0044 - accuracy: 1.0000
Epoch 196/200
12/12 [==============================] - 0s 3ms/step - loss: 0.0451 - accuracy: 0.9828
Epoch 197/200
12/12 [==============================] - 0s 3ms/step - loss: 0.0064 - accuracy: 1.0000
```

7. User Input Processing

User inputs are preprocessed using the clean_up_sentence and bow functions, which convert input text into a bag-of-words representation.

In [14]:
```
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))

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
```

8. Intent Prediction and Response Generation

The predict_class function predicts the intent of the user's input using the trained model.

The getResponse function selects a random response from the predefined responses based on the predicted intent.

The selected response is displayed to the user.

In [15]:
```
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(text):
    ints = predict_class(text, model)
    res = getResponse(ints, intents)
    return res
```

9. User Interface (UI) with Tkinter

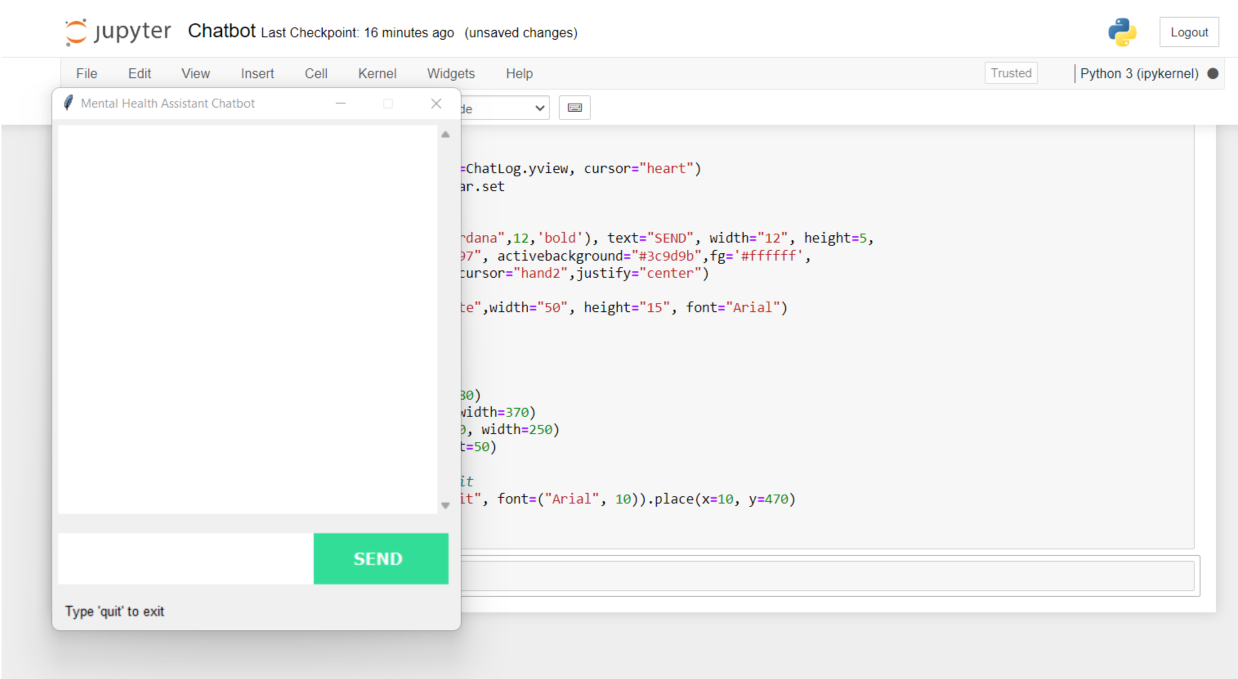
Tkinter is used to create a simple UI for interacting with the chatbot.

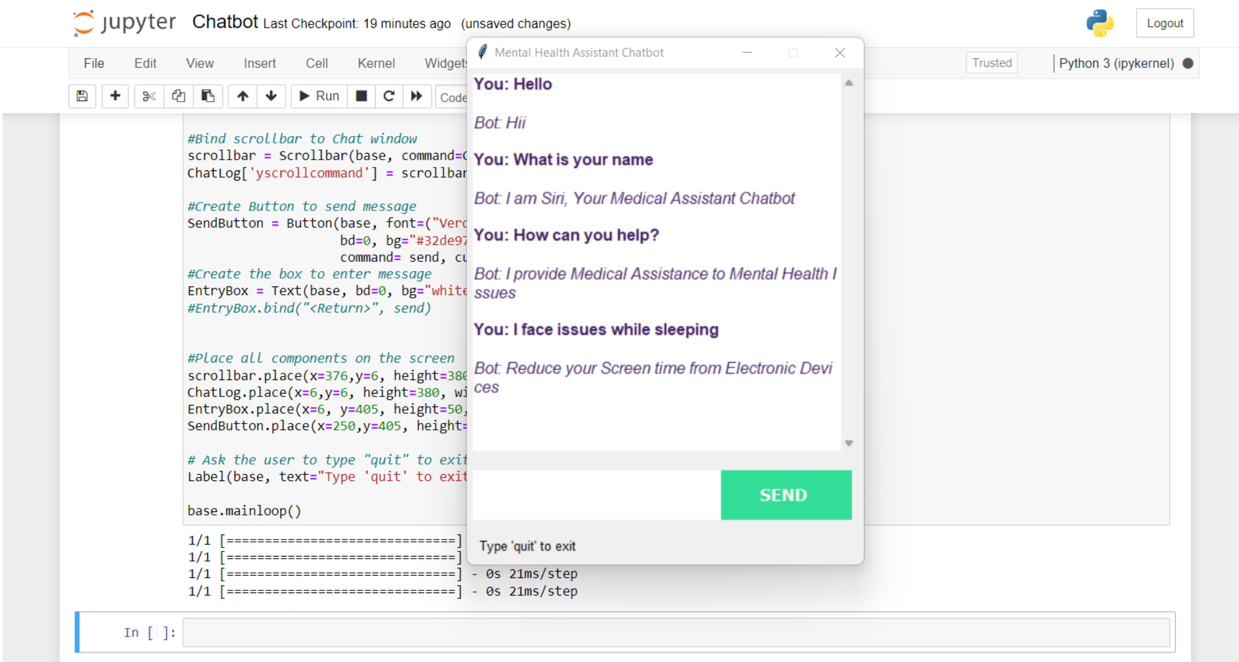
Users can type messages in a text box and receive responses from the chatbot.

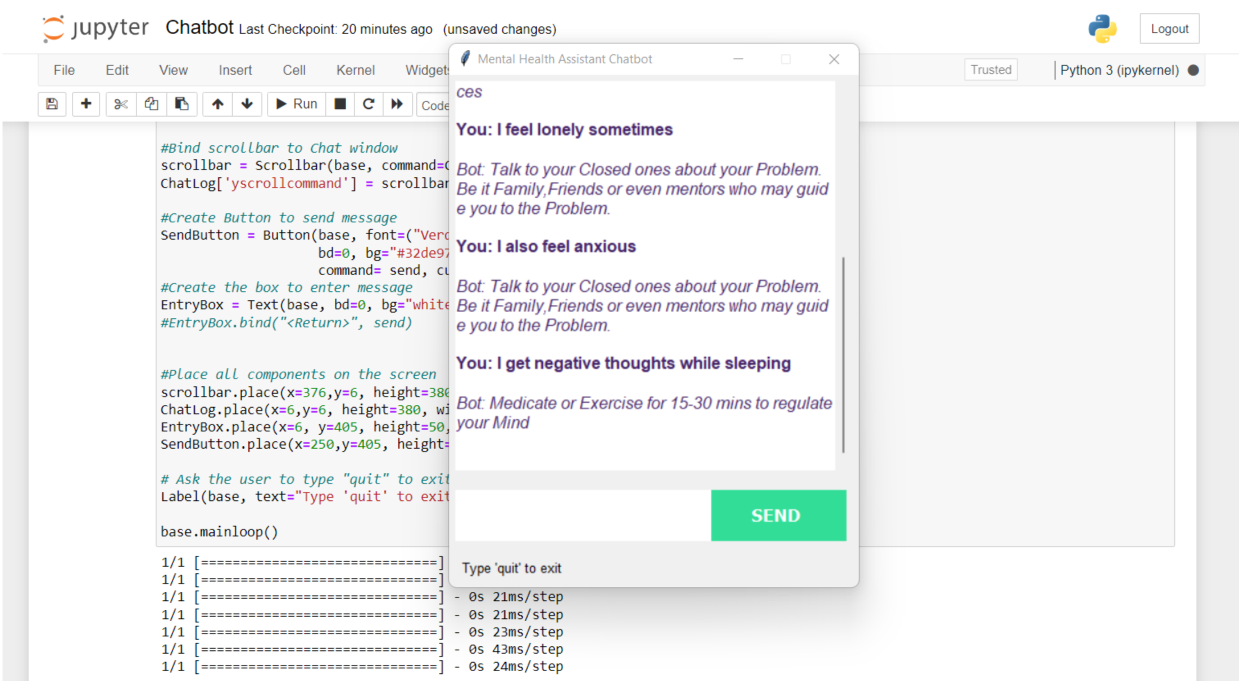
In [19]:
```
#Creating GUI with tkinter
import tkinter
from tkinter import *
import time

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

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


        if msg.lower() == "quit":
            # If the user types "quit," close the chat window
            base.destroy()
        else:
            # Delay the bot's response by a few seconds (e.g., 1 seconds)
            base.after(1000, lambda: display_bot_response(msg))

def display_bot_response(user_msg):
    res = chatbot_response(user_msg)
    ChatLog.config(state=NORMAL)
    ChatLog.tag_configure("italic", font=("Arial", 12, "italic"))
    ChatLog.insert(END, "Bot: " + res + '\n\n', "italic")

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

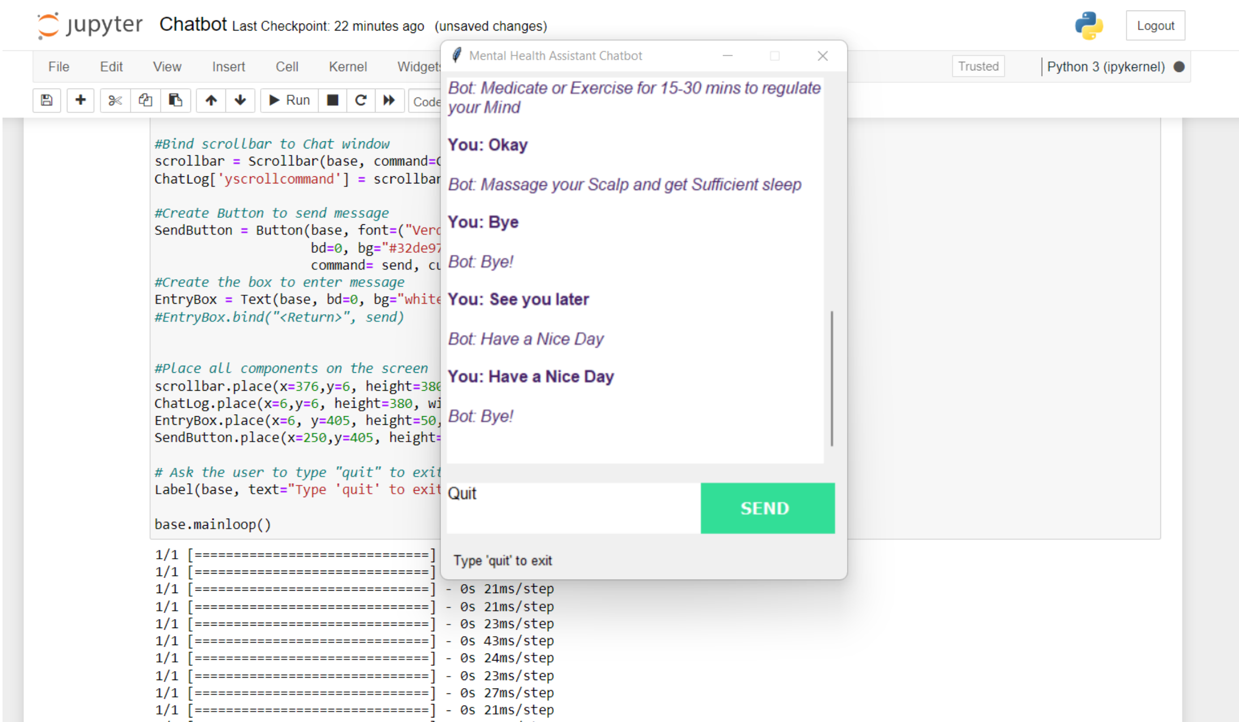
```
base = Tk()
base.title("Mental Health Assistant Chatbot")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="70", font="Arial",)

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
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, cursor="hand2",justify="center")
#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="50", height="15", font="Arial")
#EntryBox.bind("<Return>", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=380)
ChatLog.place(x=6,y=6, height=380, width=370)
EntryBox.place(x=6, y=405, height=50, width=250)
SendButton.place(x=250,y=405, height=50)

# Ask the user to type "quit" to exit
Label(base, text="Type 'quit' to exit", font=("Arial", 10)).place(x=10, y=470)

base.mainloop()
```

Below are some screenshots of the User Interface of My Chatbot for your Reference:




Jupyter Chatbot Last Checkpoint: 19 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets
Trusted
Python 3 (ipykernel)
Run
Mental Health Assistant Chatbot
You: Hello
Bot: Hii
You: What is your name
Bot: I am Siri, Your Medical Assistant Chatbot
You: How can you help?
Bot: I provide Medical Assistance to Mental Health I ssues
You: I face issues while sleeping
Bot: Reduce your Screen time from Electronic Devi ces
SEND
Type 'quit' to exit
#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=
ChatLog['yscrollcommand'] = scrollbar
#Create Button to send message
SendButton = Button(base, font=("Ver
bd=0, bg="#32de97
command= send, cu
#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white
#EntryBox.bind("<Return>", send)
#Place all components on the screen
scrollbar.place(x=376,y=6, height=380
ChatLog.place(x=6,y=6, height=380, wi
EntryBox.place(x=6, y=405, height=50,
SendButton.place(x=250,y=405, height=
Ask the user to type "quit" to exit
Label(base, text="Type 'quit' to exit
base.mainloop()
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
In [ ]:


Jupyter Chatbot Last Checkpoint: 20 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets
Trusted
Python 3 (ipykernel)
Run
Mental Health Assistant Chatbot
ces
You: I feel lonely sometimes
Bot: Talk to your Closed ones about your Problem. Be it Family,Friends or even mentors who may guid e you to the Problem.
You: I also feel anxious
Bot: Talk to your Closed ones about your Problem. Be it Family,Friends or even mentors who may guid e you to the Problem.
You: I get negative thoughts while sleeping
Bot: Medicate or Exercise for 15-30 mins to regulate your Mind
SEND
Type 'quit' to exit
#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=
ChatLog['yscrollcommand'] = scrollbar
#Create Button to send message
SendButton = Button(base, font=("Ver
bd=0, bg="#32de97
command= send, cu
#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white
#EntryBox.bind("<Return>", send)
#Place all components on the screen
scrollbar.place(x=376,y=6, height=380
ChatLog.place(x=6,y=6, height=380, wi
EntryBox.place(x=6, y=405, height=50,
SendButton.place(x=250,y=405, height=
Ask the user to type "quit" to exit
Label(base, text="Type 'quit' to exit
base.mainloop()
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 24ms/step



Conclusion:

This Mental Health Assistant Chatbot project provides a robust example of how to build a chatbot capable of understanding user inputs, determining intents, and generating appropriate responses. It leverages natural language processing techniques, machine learning, and a user-friendly interface to create an interactive chatbot.

The project's modular structure allows for easy customization and extension with additional intents and responses. It demonstrates best practices for text preprocessing, model training, and response selection.

Overall, this chatbot project serves as a valuable foundation for businesses looking to implement automated customer support and engagement systems. By integrating it with domain-specific knowledge and expanding the intents, it can become a powerful tool for improving user experiences and providing timely assistance.