MACHINE LEARNING FOR DATA ANALYSIS

PROJECT REPORT

CHATBOT

GROUP MEMBERS

ARESHA ARSHAD

AMTUL HAI BARIA

USAMA ALI HAIDER

SUBMITTED TO

MS. WARDA ASLAM

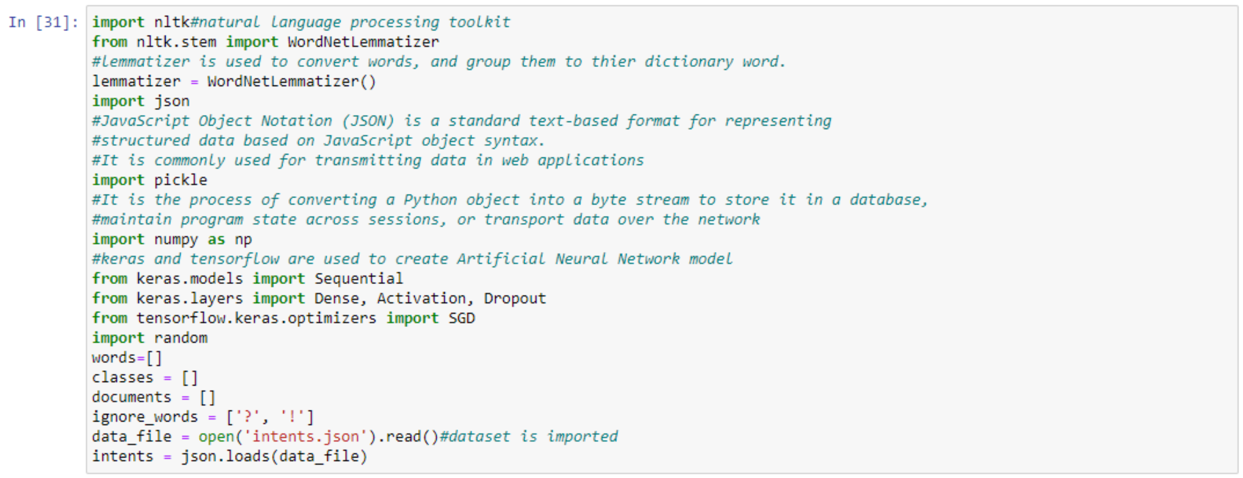
Table of Contents

INTRODUCTION

Chat bot is a computer program designed to stimulate conversation to human users, especially over the internet. We have created a small chat bot as your companion to eliminate loneliness from your life. We have used a JSON format dataset. The purpose of this project is to use NLTK, Keras, and Special recurrent neural networks in python. It is the basic model of the chat bot with a deep learning version to train our dataset according to our requirements and understanding. We have used GUI to predict our responses in less time.

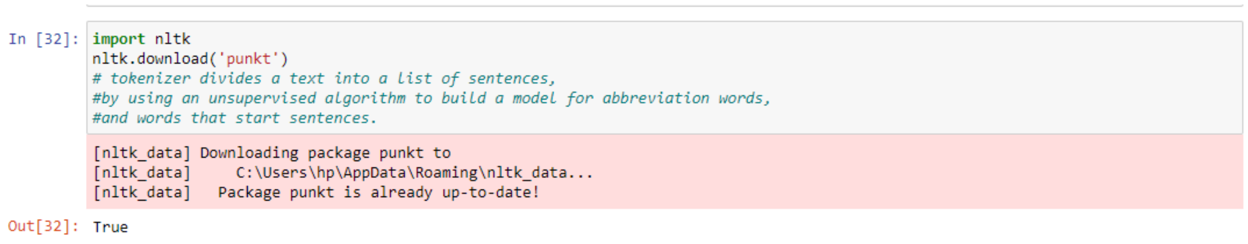
IMPORTING AND LOADING DATA FILES

We import the necessary packages and libraries for our chat bot from python and have initialized the variables (words, documents, ignore words, and classes) to be used to further in our model. Our data file is in JSON format so to make it readable we have also imported JSON package from python for our chat bot.



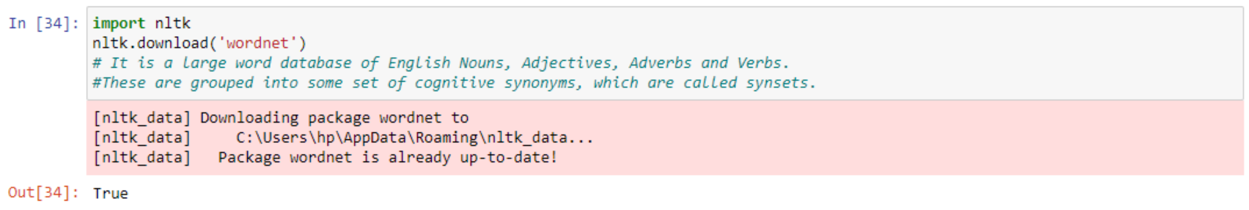
```
In [31]: import nltk#natural language processing toolkit
         from nltk.stem import WordNetLemmatizer
         #lemmatizer is used to convert words, and group them to thier dictionary word.
         lemmatizer = WordNetLemmatizer()
         import json
         #JavaScript Object Notation (JSON) is a standard text-based format for representing
         #structured data based on JavaScript object syntax.
         #It is commonly used for transmitting data in web applications
         import pickle
         #It is the process of converting a Python object into a byte stream to store it in a database,
         #maintain program state across sessions, or transport data over the network
         import numpy as np
         #keras and tensorflow are used to create Artificial Neural Network model
         from keras.models import Sequential
         from keras.layers import Dense, Activation, Dropout
         from tensorflow.keras.optimizers import SGD
         import random
         words=[]
         classes = []
         documents = []
         ignore_words = ['?', '!']
         data_file = open('intents.json').read()#dataset is imported
         intents = json.loads(data_file)
```

From NLTK (toolkit for NLP) we have downloaded **punkt** and **wordnet** files to be used in our model.



```
In [32]: import nltk
         nltk.download('punkt')
         # tokenizer divides a text into a list of sentences,
         #by using an unsupervised algorithm to build a model for abbreviation words,
         #and words that start sentences.

         [nltk_data] Downloading package punkt to
         [nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
         [nltk_data]   Package punkt is already up-to-date!

Out[32]: True
```

PUNKT allows for incremental training and modification of the hyper-parameters used to decide what is considered as abbreviation. This tokenizes a text into a list of sentences, by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plain text in the target language before it can be used.

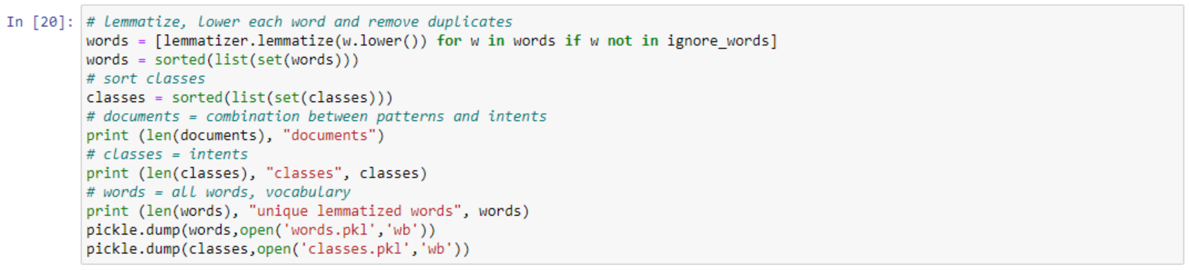


```
In [34]: import nltk
         nltk.download('wordnet')
         # It is a large word database of English Nouns, Adjectives, Adverbs and Verbs.
         #These are grouped into some set of cognitive synonyms, which are called synsets.

         [nltk_data] Downloading package wordnet to
         [nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
         [nltk_data]   Package wordnet is already up-to-date!

Out[34]: True
```

WORDNET is a large word database of English nouns, adjectives, adverbs, and verbs. These are grouped into some set of cognitive synonyms, which are called synsets.

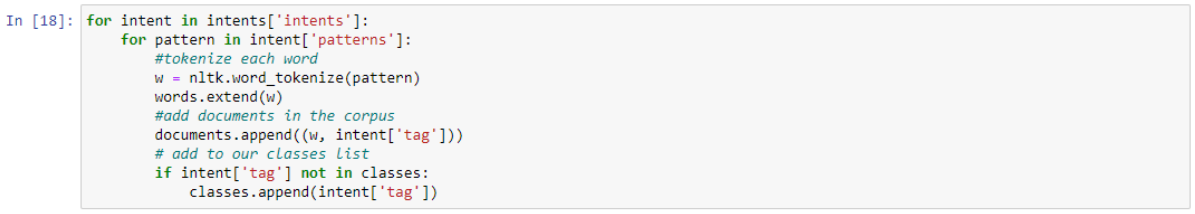
PREPROCESSING

Preprocessing of the dataset is one of the most important steps. It is used to clean our data and extract the best version of dataset to be further used in ANN. We used NLP toolkit to preprocess the text data. For this purpose, we've used the NLTK in our code. Text preprocessing use both stemming and lemmatization, but we have preferred lemmatization over stemming because lemmatization itself do more morphological analysis of words.



```
In [20]: # 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)
         pickle.dump(words,open('words.pkl','wb'))
         pickle.dump(classes,open('classes.pkl','wb'))
```

We have used WordNetLemmatizer for preprocessing. WordNetLemmatizer is used for lemmatization which is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item, basically it converges to its stem word. We have used pickle files to convert our classes and words file (i-e python object) into a byte stream to store it in a file, to maintain a program state across sessions or transport data over the network.

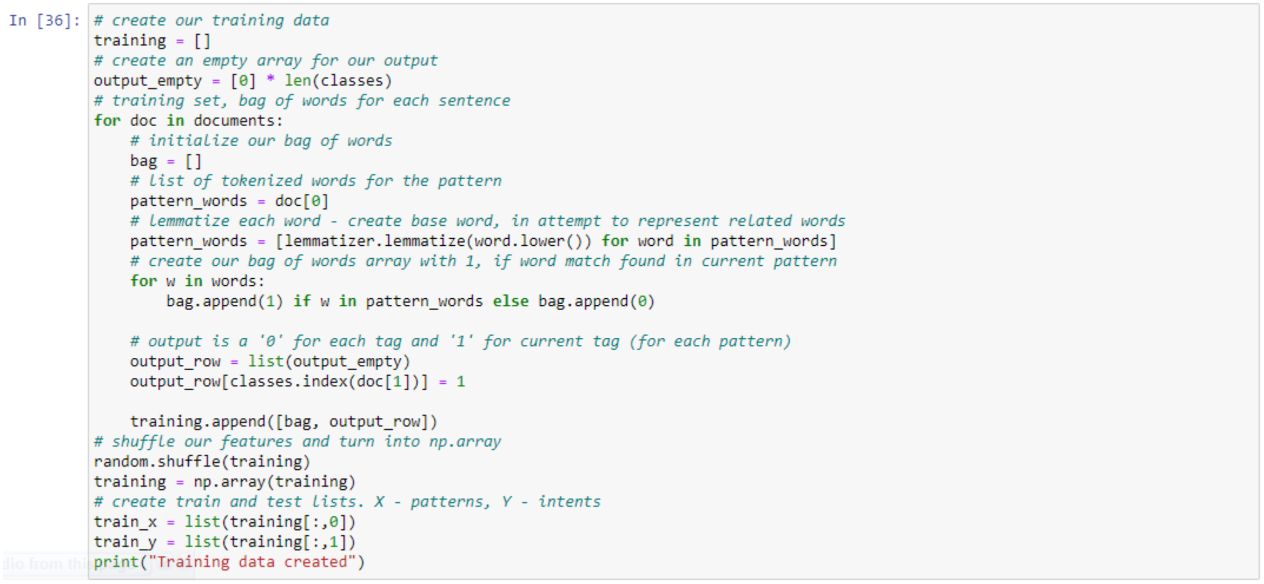


```
In [18]: for intent in intents['intents']:
             for pattern in intent['patterns']:
                 #tokenize each word
                 w = nltk.word_tokenize(pattern)
                 words.extend(w)
                 #add documents in the corpus
                 documents.append((w, intent['tag']))
                 # add to our classes list
                 if intent['tag'] not in classes:
                     classes.append(intent['tag'])
```

We have used nested for loop for tokenization of our tags in the data file of intents.

TRAINING DATA

We are creating our training data now on which we will implement our model. For this purpose, we have created an empty array of training set and an empty array for output, to store the bag of required words in these empty arrays. In our dataset, input will be our patterns and output will be classes to which these patterns belong. Since computer does not understand words so we are assigning numbers to our text.

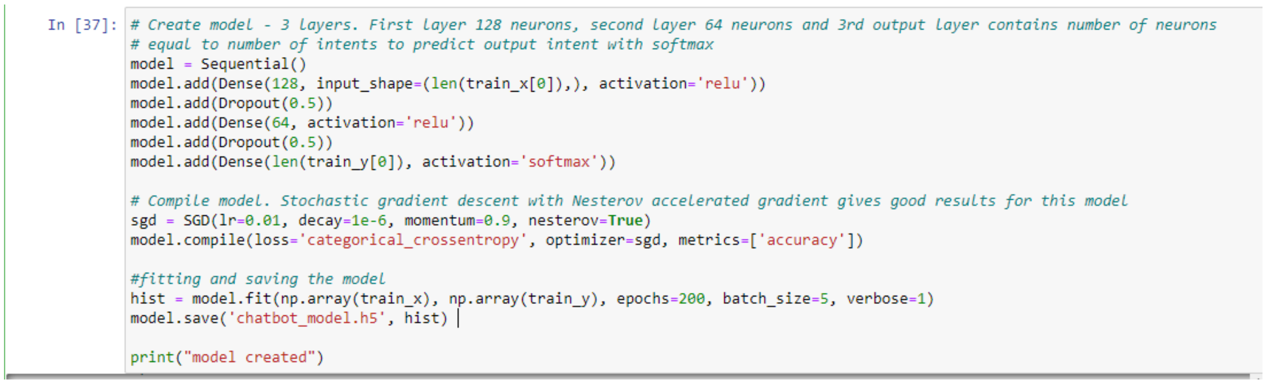


```
In [36]: # create our training data
         training = []
         # create an empty array for our output
         output_empty = [0] * len(classes)
         # training set, bag of words for each sentence
         for doc in documents:
             # initialize our bag of words
             bag = []
             # 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")
```

We have used lemmatization here again to convert our datasets words into their base words and then assigning numbers to them. At the end, we have shuffled our training set and created an array out of them and then separated our training features and labels.

CREATING MODEL

Now, finally its time to create our model using Nueral Network. We have used keras sequential for this purpose. We have created total of three layers, input, hidden and output, along with 2 dropout layers. Relu and softmax are used as an activation functions. We have compiled our model using stochastic gradient descent as our optimizer for more accuracy.

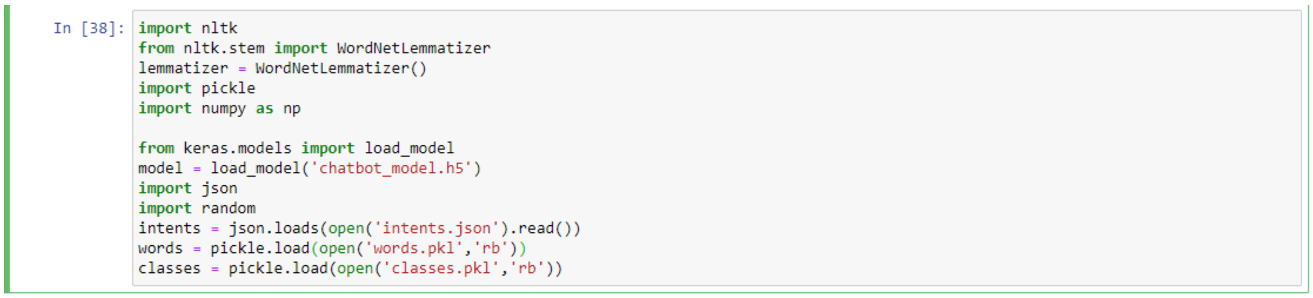


```
In [37]: # Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons
         # equal to number of intents to predict output intent with softmax
         model = Sequential()
         model.add(Dense(128, input_shape=(len(train_x[0]),), 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(lr=0.01, decay=1e-6, 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")
```

Further, we fitted our training data into our model as arrays with 200 epochs (i-e number of iteration for back and forth propogation) and batch size of 5 (i-e random data samples going in first iteration). Further we saved our model with name ‘chatbot_model.h5’.

PREDICTING RESPONSES

Now, its time to predict reponses from chatbot using our saved model. For this, we imported our model and then used GUI that will predict the response from the bot, loaded our intents, classes and words as pickle files we created when we trained our model, also imported word net lemmatizer for lemmatization of words given as input further. The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.



```
In [38]: 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('intents.json').read())
         words = pickle.load(open('words.pkl','rb'))
         classes = pickle.load(open('classes.pkl','rb'))
```

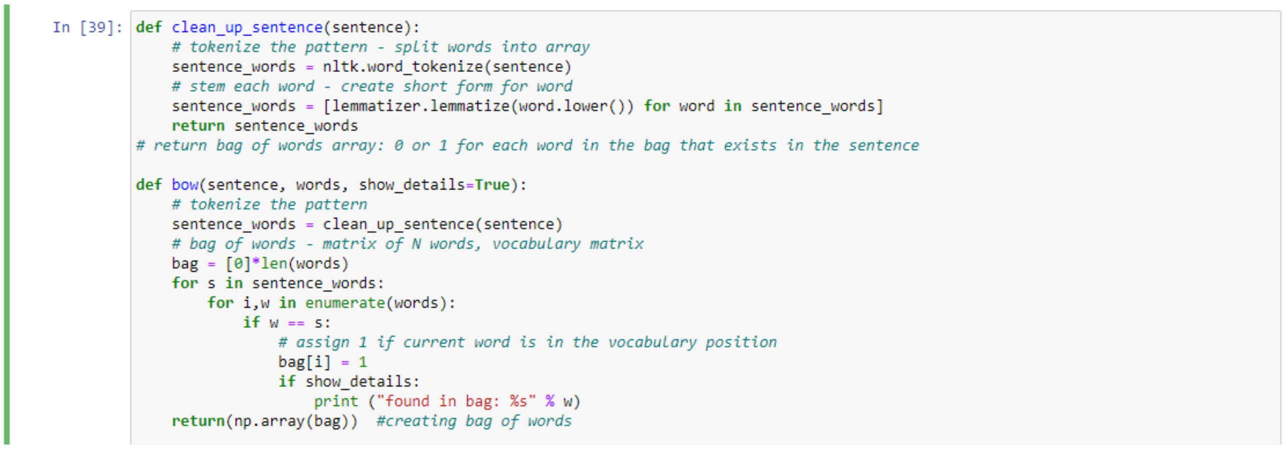
To predict the class, we will need to provide input in the same way as we did while training. So, we will create some functions that will perform text preprocessing and then predict the class. Use of following functions:

- **CLEAN_UP_SENTENCE**

  This function cleans the input given from the user to be further used for prediction of input. So whenever called it uses tools from NLTK to clean input.

- **BOW**

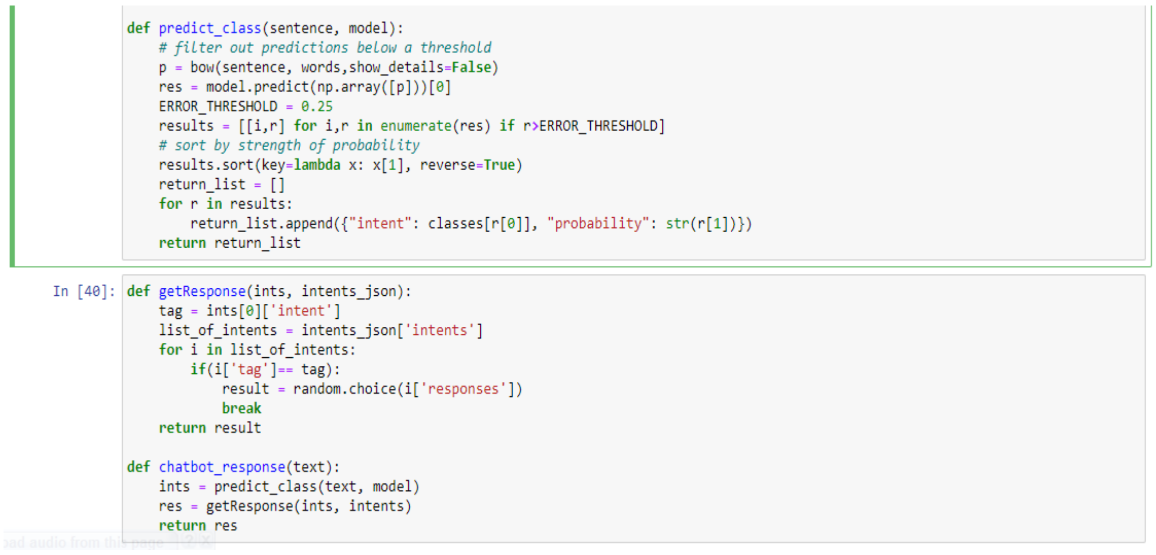
  This is the one which creates bag of words. Here we call our above function. It returns an array of words. This function assigns 1 to the input words if they get positioned with words in our dataset.



```
In [39]: 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))  #creating bag of words
```

- **PREDICT_CLASS**

  This function takes the input sentence and our model as input and predict the response as output. We have loaded the result of bow function into ‘p’ and then loaded that array in ‘res’ variable. Now, we have set a threshold and filter out prediction below the threshold. Then we will append the results from intents into ‘return_list’ that basically contains all the possible responses of the input sentence. After predicting now, we have number of random responses from the file of intents.



```
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
```

```
In [40]: 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
```

- **GETRESPONSE**

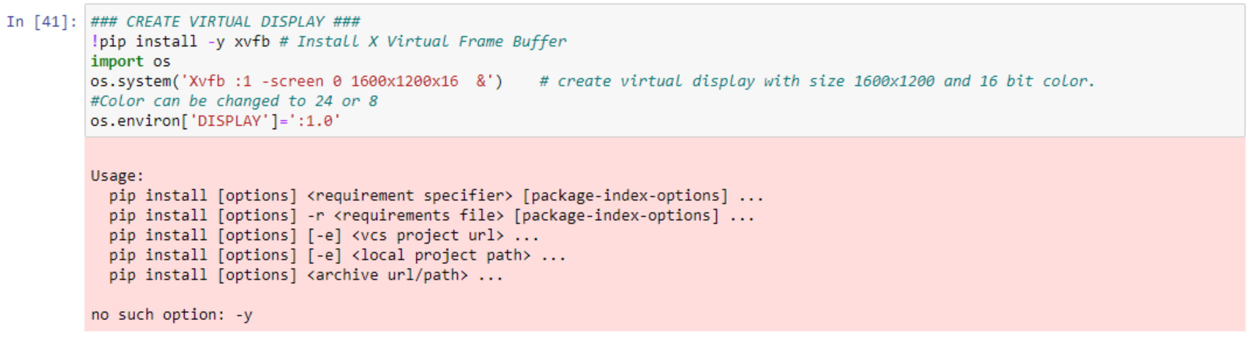
  This function takes intents file and itns as input, now we will use tags to get the best response out of all possible responses. We will match the tag of predicted class with the tags present in our intent file to randomly choose the best response.

- **CHATBOT_RESPONSE**

  Here, we are finally printing the result using text, model, and intents.

WORKING WITH OPERATING SYSTEM

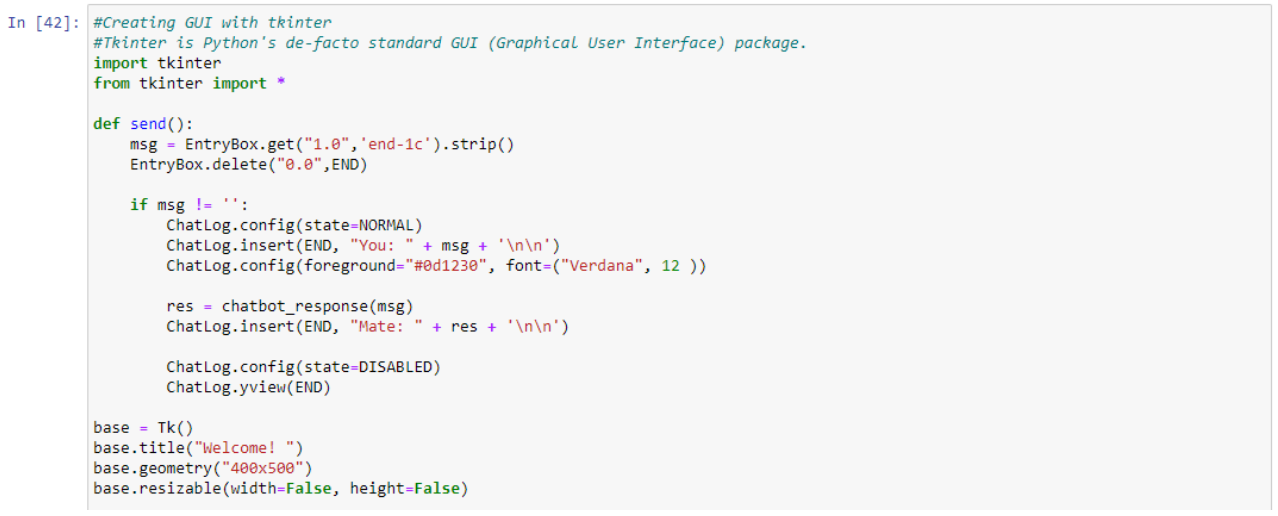
Now, finally it’s time to display our chat bot after training and predicting the best output of all. For this purpose, first we will import “os” which is the platform in python to interact with the operating system. We have installed virtual frame buffer (it basically performs all graphical operations in virtual memory without showing any screen output).



```
In [41]: ### CREATE VIRTUAL DISPLAY ###
         !pip install -y xvfb # Install X Virtual Frame Buffer
         import os
         os.system('Xvfb :1 -screen 0 1600x1200x16  &')    # create virtual display with size 1600x1200 and 16 bit color.
         #Color can be changed to 24 or 8
         os.environ['DISPLAY']=':1.0'
```

```
Usage:
  pip install [options] <requirement specifier> [package-index-options] ...
  pip install [options] -r <requirements file> [package-index-options] ...
  pip install [options] [-e] <vcs project url> ...
  pip install [options] [-e] <local project path> ...
  pip install [options] <archive url/path> ...

no such option: -y
```

Now we create user interference for the chat bot. For this purpose, Tkinter, i.e., python package used for graphical user interface. Tkinter includes tons of useful libraries for GUI. We have created user appealing and interactive chat bot in order to get the user-friendly experience. We will take the input message from the user and then use the helper functions we have created to get the response from the bot and display it on the graphical user interface.

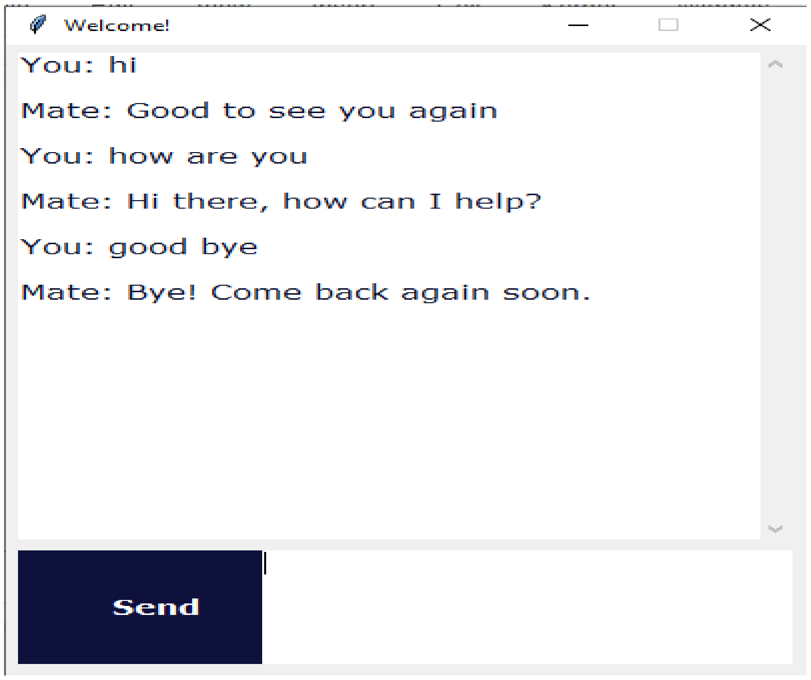


```
In [42]: #Creating GUI with tkinter
         #Tkinter is Python's de-facto standard GUI (Graphical User Interface) package.
         import tkinter
         from tkinter import *

         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="#0d1230", font=("Verdana", 12 ))

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

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

         base = Tk()
         base.title("Welcome! ")
         base.geometry("400x500")
         base.resizable(width=False, height=False)
```

```
#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="10", width="50", font="Ariel",)

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="#0d113b", activebackground="#3c649d",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="30", height="7", font="Ariel")
#EntryBox.bind("<Return>", send)


#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()
```

CONCLUSION

In this project, we understood about chat bots and implemented a deep learning version of a chatbot in Python which is accurate. One can customize the data according to their business requirements and train the chatbot with great accuracy. Chat bots are used everywhere and all businesses are looking forward to implementing chat bot in their workflow.



```
You: hi
Mate: Good to see you again
You: how are you
Mate: Hi there, how can I help?
You: good bye
Mate: Bye! Come back again soon.
```