Configuration Manual

MSc Research Project
MSc in data Analytics

Suryakanta Sahu
Student ID: X20141513

School of Computing
National College of Ireland

Supervisor: Dr. Majid Latifi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Suryakanta Sahu		
Student ID:	X20141513		
Programme:	Master's in Data Analytics	**Year:**	2021
Module:	MSc Research Project		
Lecturer:	Dr. Majid Latifi		
Submission Due Date:	31/01/2022		
Project Title:	Contextual Healthcare Chatbot using Deep Neural Network – Configuration Manual		
Word Count:	**1229**	**Page Count: 15**	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Suryakanta Sahu

Date: 16/12/2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Suryakanta Sahu
Student ID: X20141513

1 Introduction

This configuration manual report provides a detailed synopsis of the hardware, software, library packages, and programming languages used to implement the project i.e., “Contextual Healthcare Chatbot using Deep Neural Network”. In addition to the hardware and software specifications, the document also shows the functions and procedures used to develop the deep learning model and pre-process the raw training data. By referring to this document, the future researchers will be able to simply replicate this research for further analysis, review, and extension.

This guidebook is divided into the sections below: The section 2 explains the hardware and software configuration of the system on which the project was executed. Methods and procedures executed to clear, transform the input data is outlined in the section 3. The section 4 illustrates the way models are implemented and finally the section 5 shows the evaluation and performance of the models.

2 System Configuration

In this section, the system configurations required to implement and execute the project are described.

2.1 Hardware Specification

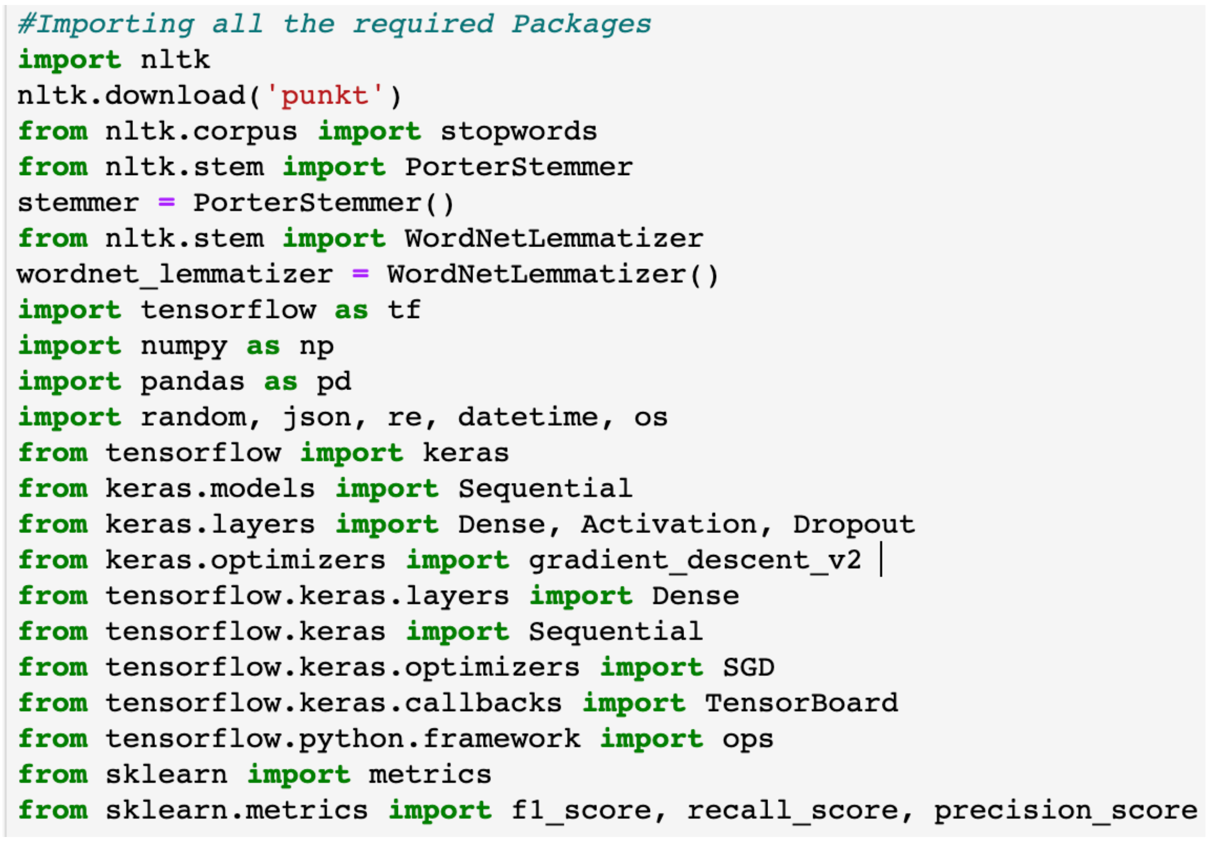
Operating System	macOS Monterey (Version 12.1 (21C52))
Processor	3.5 GHz Dual-Core Intel Core i7
RAM	16 GB 2133 MHz LPDDR3
System Type	64-bit CPU
No. of Processor	1
Total no. of Cores	2

2.2 Software Specification

- Programming Language: Python
- Language Version: Python 3.8.5
- Editor Used: Jupyter Notebook (Google Colab)
- Distribution: Anaconda Distribution

Libraries used:

Library Name	Usage
Numpy	To perform a wide variety of mathematical operations on arrays
Pandas	For loading the data and performing numerous statistical analysis
random	To shuffle the input dataset
Json	To load and work with JSON file
re	To clean the input dataset
nltk	Used to work with natural language, and for per- forming actions like tokenization, stemming, lemmatization on the text data
tensorflow	Numerous sub-packages from tensorflow package were imported to create, fit and predict the Sequential model
datetime	To work with system date and time
os	To work with system folder structure in order to load and save data
sklearn	To create Machine Learning models and evaluate them

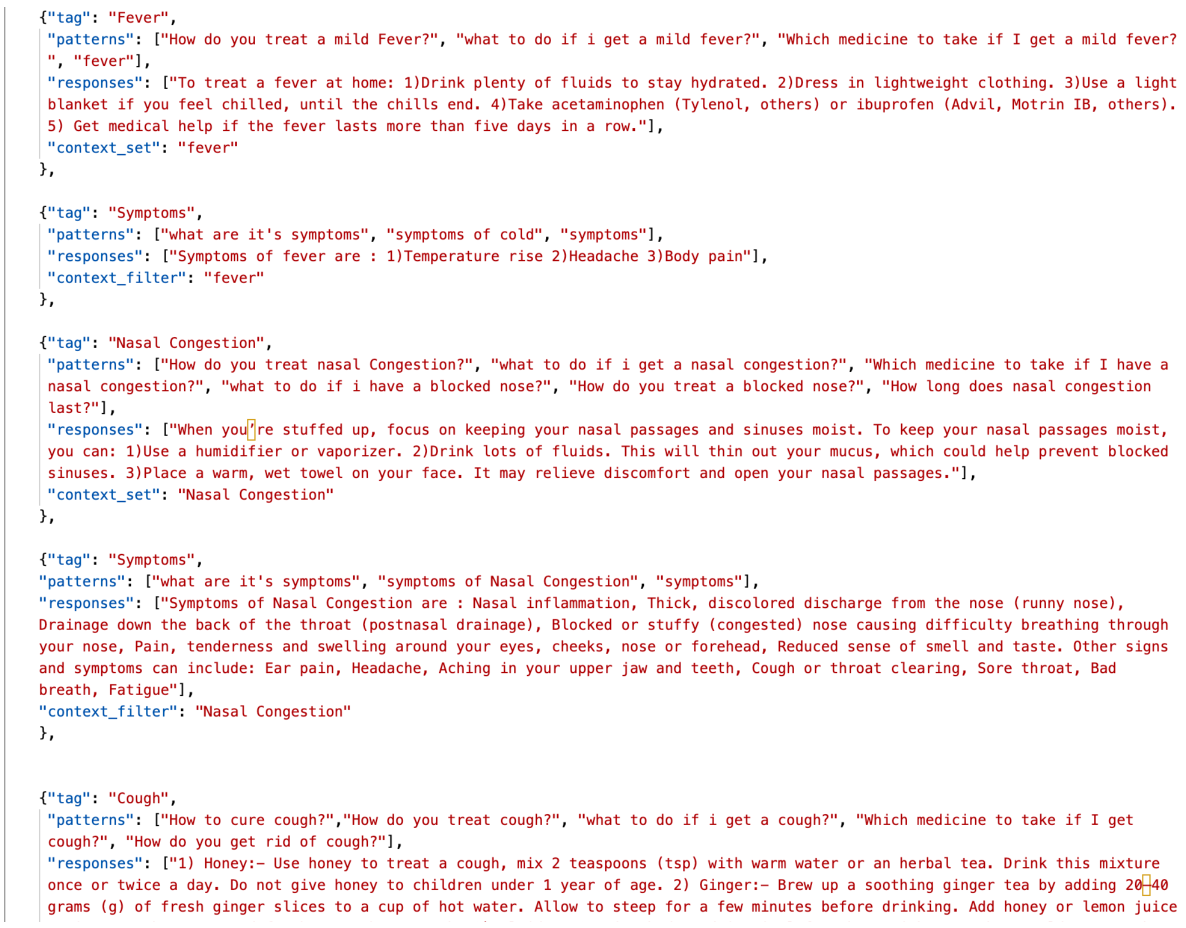


```
#Importing all the required Packages
import nltk
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
import tensorflow as tf
import numpy as np
import pandas as pd
import random, json, re, datetime, os
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import gradient_descent_v2
from tensorflow.keras.layers import Dense
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.python.framework import ops
from sklearn import metrics
from sklearn.metrics import f1_score, recall_score, precision_score
```

Figure 1: Libraries Imported

3 Data Handling

The dataset was sourced from various open-source websites like Wikipedia, WebMD etc and then stored in JSON format.

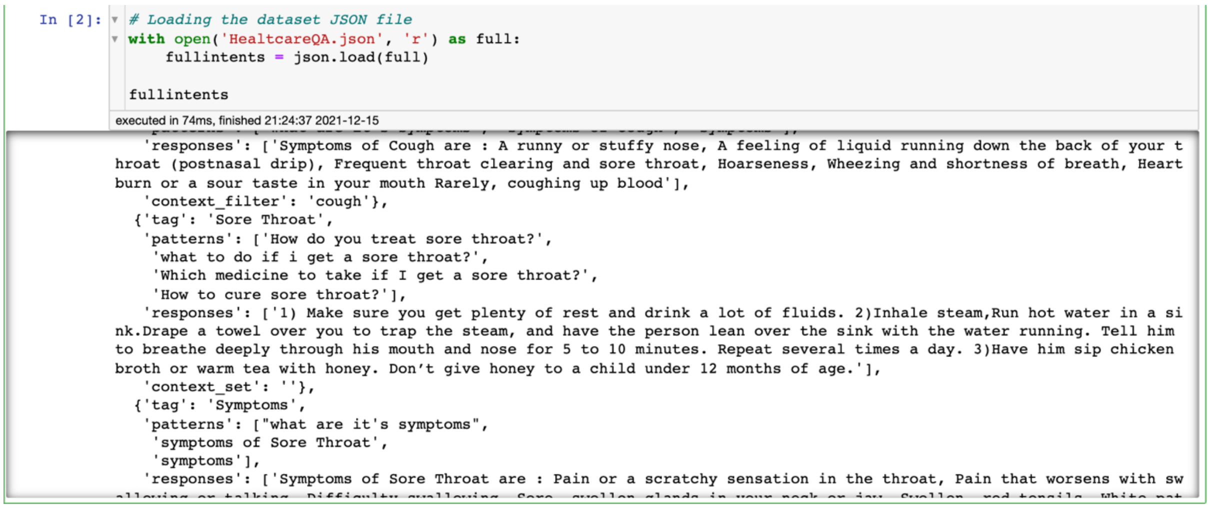


```
{"tag": "Fever",
 "patterns": ["How do you treat a mild Fever?", "what to do if i get a mild fever?", "Which medicine to take if I get a mild fever?
", "fever"],
 "responses": ["To treat a fever at home: 1)Drink plenty of fluids to stay hydrated. 2)Dress in lightweight clothing. 3)Use a light
 blanket if you feel chilled, until the chills end. 4)Take acetaminophen (Tylenol, others) or ibuprofen (Advil, Motrin IB, others).
 5) Get medical help if the fever lasts more than five days in a row."],
 "context_set": "fever"
},

{"tag": "Symptoms",
 "patterns": ["what are it's symptoms", "symptoms of cold", "symptoms"],
 "responses": ["Symptoms of fever are : 1)Temperature rise 2)Headache 3)Body pain"],
 "context_filter": "fever"
},

{"tag": "Nasal Congestion",
 "patterns": ["How do you treat nasal Congestion?", "what to do if i get a nasal congestion?", "Which medicine to take if I have a
 nasal congestion?", "what to do if i have a blocked nose?", "How do you treat a blocked nose?", "How long does nasal congestion
 last?"],
 "responses": ["When you’re stuffed up, focus on keeping your nasal passages and sinuses moist. To keep your nasal passages moist,
 you can: 1)Use a humidifier or vaporizer. 2)Drink lots of fluids. This will thin out your mucus, which could help prevent blocked
 sinuses. 3)Place a warm, wet towel on your face. It may relieve discomfort and open your nasal passages."],
 "context_set": "Nasal Congestion"
},

{"tag": "Symptoms",
"patterns": ["what are it's symptoms", "symptoms of Nasal Congestion", "symptoms"],
"responses": ["Symptoms of Nasal Congestion are : Nasal inflammation, Thick, discolored discharge from the nose (runny nose),
Drainage down the back of the throat (postnasal drainage), Blocked or stuffy (congested) nose causing difficulty breathing through
your nose, Pain, tenderness and swelling around your eyes, cheeks, nose or forehead, Reduced sense of smell and taste. Other signs
and symptoms can include: Ear pain, Headache, Aching in your upper jaw and teeth, Cough or throat clearing, Sore throat, Bad
breath, Fatigue"],
"context_filter": "Nasal Congestion"
},


{"tag": "Cough",
 "patterns": ["How to cure cough?","How do you treat cough?", "what to do if i get a cough?", "Which medicine to take if I get
 cough?", "How do you get rid of cough?"],
 "responses": ["1) Honey:- Use honey to treat a cough, mix 2 teaspoons (tsp) with warm water or an herbal tea. Drink this mixture
 once or twice a day. Do not give honey to children under 1 year of age. 2) Ginger:- Brew up a soothing ginger tea by adding 20–40
 grams (g) of fresh ginger slices to a cup of hot water. Allow to steep for a few minutes before drinking. Add honey or lemon juice
```

Figure 2: Data stored in JSON format

3.1 Loading Dataset

The JSON data file was loaded into the project space using json.load() function.



```
In [2]: # Loading the dataset JSON file
        with open('HealtcareQA.json', 'r') as full:
            fullintents = json.load(full)

        fullintents
```

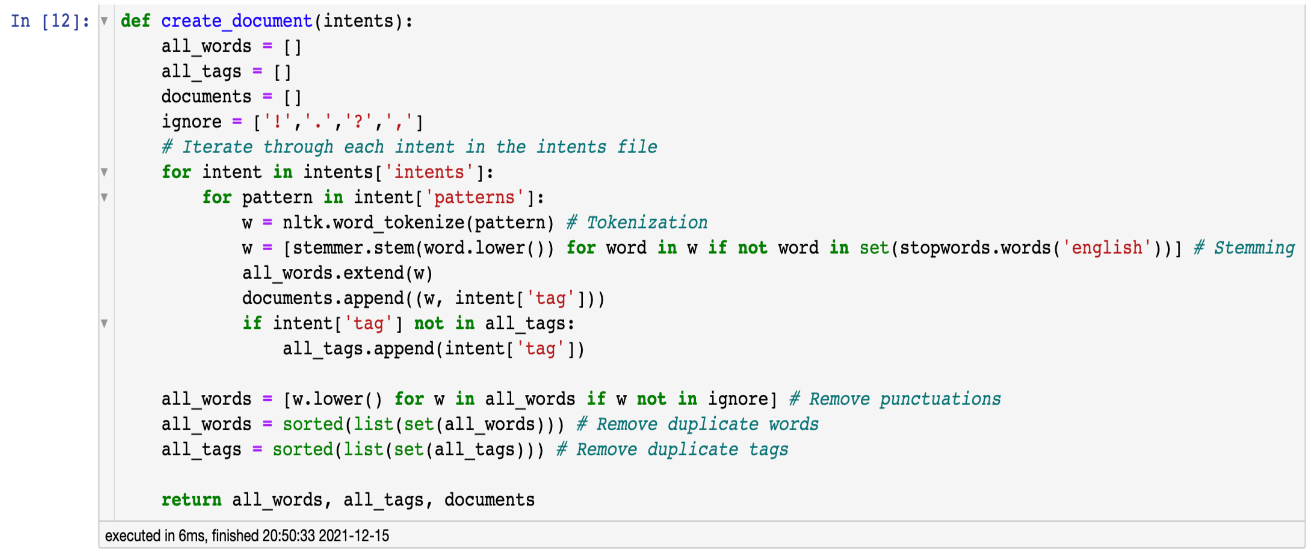
executed in 74ms, finished 21:24:37 2021-12-15

```
   'responses': ['Symptoms of Cough are : A runny or stuffy nose, A feeling of liquid running down the back of your t
hroat (postnasal drip), Frequent throat clearing and sore throat, Hoarseness, Wheezing and shortness of breath, Heart
burn or a sour taste in your mouth Rarely, coughing up blood'],
   'context_filter': 'cough'},
  {'tag': 'Sore Throat',
   'patterns': ['How do you treat sore throat?',
    'what to do if i get a sore throat?',
    'Which medicine to take if I get a sore throat?',
    'How to cure sore throat?'],
   'responses': ['1) Make sure you get plenty of rest and drink a lot of fluids. 2)Inhale steam,Run hot water in a si
nk.Drape a towel over you to trap the steam, and have the person lean over the sink with the water running. Tell him
to breathe deeply through his mouth and nose for 5 to 10 minutes. Repeat several times a day. 3)Have him sip chicken
broth or warm tea with honey. Don’t give honey to a child under 12 months of age.'],
   'context_set': ''},
  {'tag': 'Symptoms',
   'patterns': ["what are it's symptoms",
    'symptoms of Sore Throat',
    'symptoms'],
   'responses': ['Symptoms of Sore Throat are : Pain or a scratchy sensation in the throat, Pain that worsens with sw
```

Figure 3: Loading the data into project space

3.2 Data Description

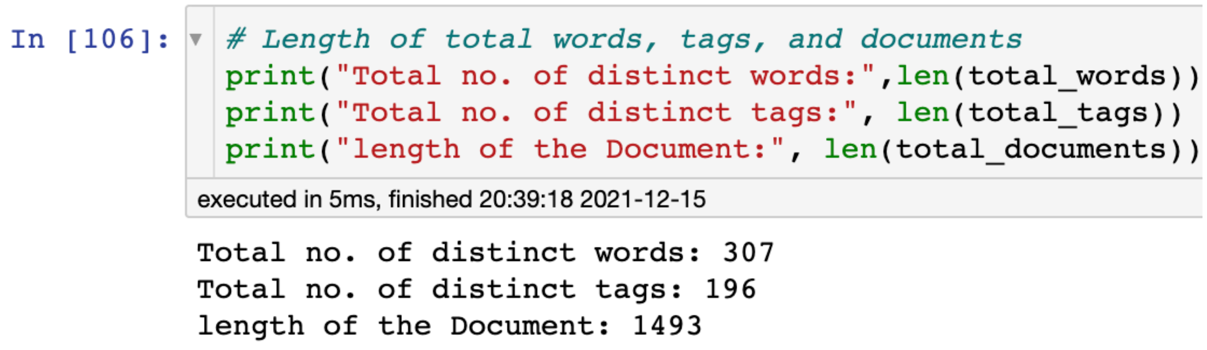
Each tag of the loaded json data was iterated through, and all the tags, classes are stored in list format.



```
In [12]: def create_document(intents):
             all_words = []
             all_tags = []
             documents = []
             ignore = ['!','.','?',',']
             # Iterate through each intent in the intents file
             for intent in intents['intents']:
                 for pattern in intent['patterns']:
                     w = nltk.word_tokenize(pattern) # Tokenization
                     w = [stemmer.stem(word.lower()) for word in w if not word in set(stopwords.words('english'))] # Stemming
                     all_words.extend(w)
                     documents.append((w, intent['tag']))
                     if intent['tag'] not in all_tags:
                         all_tags.append(intent['tag'])

             all_words = [w.lower() for w in all_words if w not in ignore] # Remove punctuations
             all_words = sorted(list(set(all_words))) # Remove duplicate words
             all_tags = sorted(list(set(all_tags))) # Remove duplicate tags

             return all_words, all_tags, documents
executed in 6ms, finished 20:50:33 2021-12-15
```

Figure 4: Creating a Document with all the words, tags

As shown in the Figure 5, a total of 307 distinct words, 196 distinct tags and the whole document contained 1493 records.



```
In [106]: # Length of total words, tags, and documents
          print("Total no. of distinct words:",len(total_words))
          print("Total no. of distinct tags:", len(total_tags))
          print("length of the Document:", len(total_documents))
executed in 5ms, finished 20:39:18 2021-12-15

          Total no. of distinct words: 307
          Total no. of distinct tags: 196
          length of the Document: 1493
```

Figure 5: Total number of words, tags, and length of the document

In the Figure 6: word cloud of the Tags and in the Figure 7: world cloud of the words are shown.

```
In [104]:  import matplotlib.pyplot as plt
           from wordcloud import WordCloud

           unique_string=(" ").join(total_tags)
           wordcloud = WordCloud(width = 1000, height = 500).generate(unique_string)
           plt.figure(figsize=(15,8))
           plt.imshow(wordcloud)
           plt.axis("off")
           plt.savefig("your_file_name"+".png", bbox_inches='tight')
           plt.show()
           plt.close()
```

executed in 1.54s, finished 20:37:27 2021-12-15

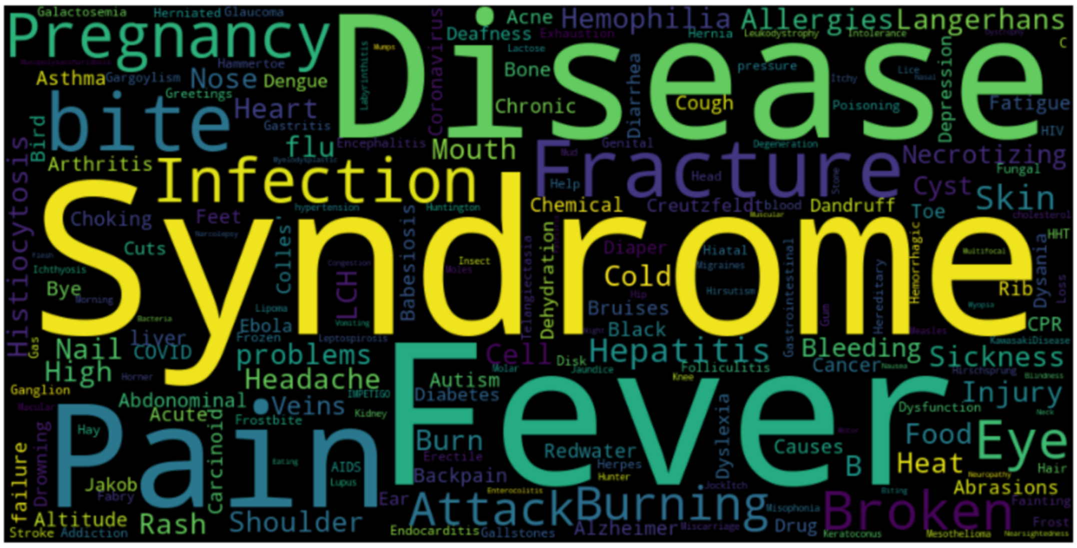


Figure 6: Word Count of Tags

```
In [105]:  import matplotlib.pyplot as plt
           from wordcloud import WordCloud

           unique_string=(" ").join(total_words)
           wordcloud = WordCloud(width = 1000, height = 500).generate(unique_string)
           plt.figure(figsize=(15,8))
           plt.imshow(wordcloud)
           plt.axis("off")
           plt.savefig("your_file_name"+".png", bbox_inches='tight')
           plt.show()
           plt.close()
```

executed in 1.49s, finished 20:37:39 2021-12-15

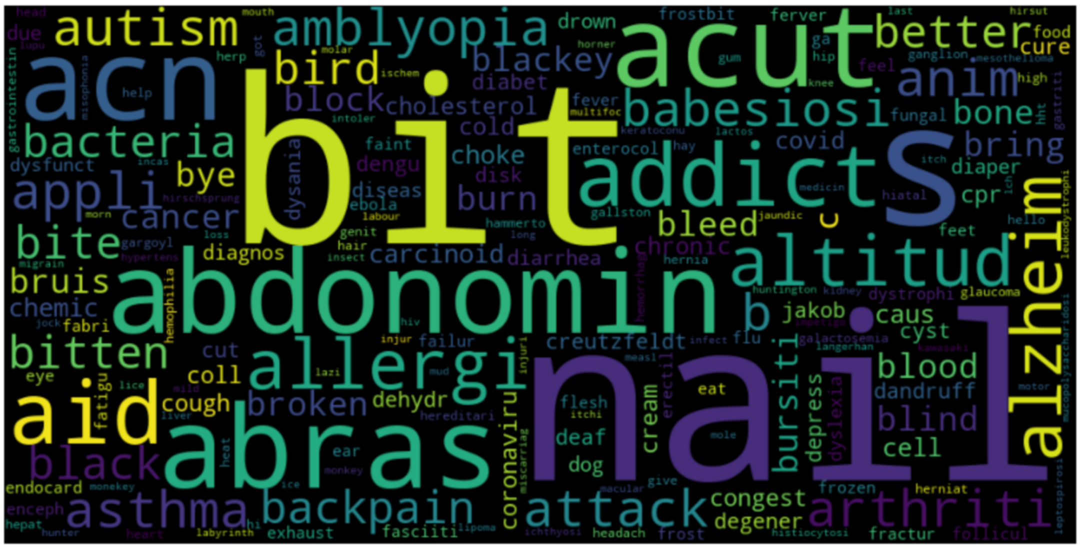
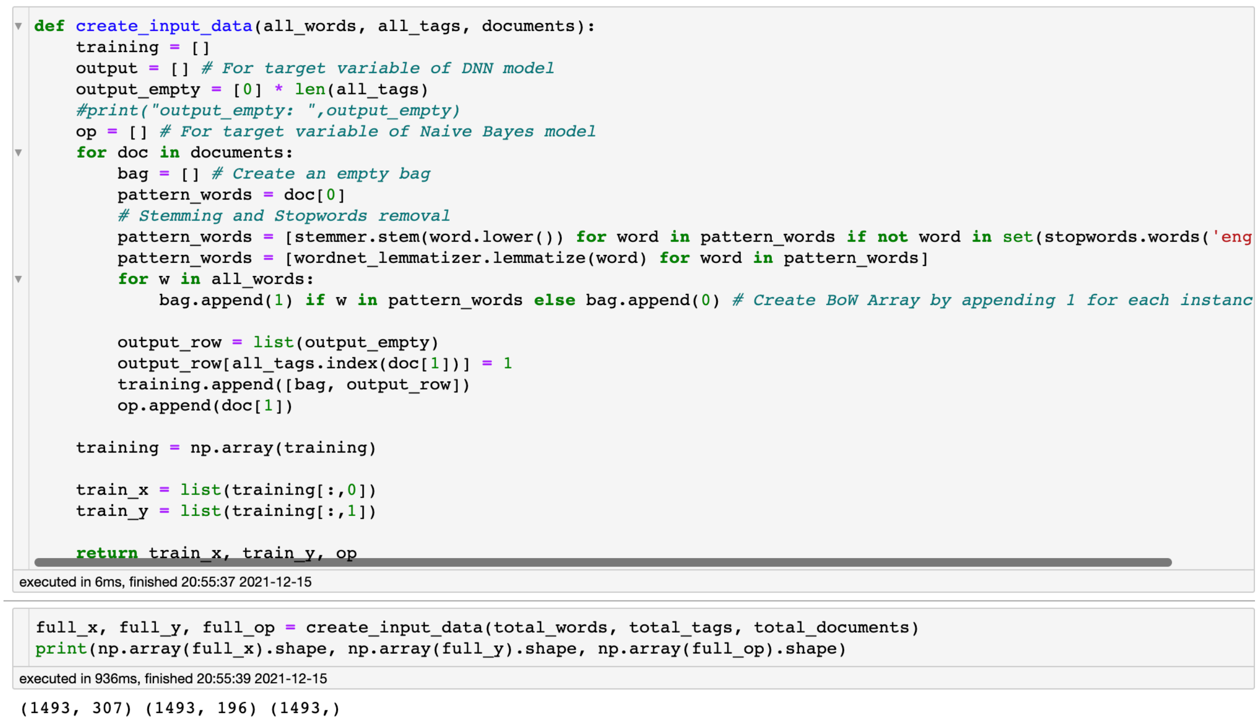


Figure 7: Word Count of Words

3.3 Data Pre-Processing and Transformation

As shown in the Figure: 8, the input text data was first tokenized, stemmed, and Lemmatized. Then a Word Vector using Bag of Word (BoW) technique was created. The word vector sparse matrix was then segregated into Independent and Target variable. The independent variable (X) has 1493 rows and 307 columns. Similarly, the target variable (Y) has 1493 rows and 196 columns.



```
def create_input_data(all_words, all_tags, documents):
    training = []
    output = [] # For target variable of DNN model
    output_empty = [0] * len(all_tags)
    #print("output_empty: ",output_empty)
    op = [] # For target variable of Naive Bayes model
    for doc in documents:
        bag = [] # Create an empty bag
        pattern_words = doc[0]
        # Stemming and Stopwords removal
        pattern_words = [stemmer.stem(word.lower()) for word in pattern_words if not word in set(stopwords.words('eng
        pattern_words = [wordnet_lemmatizer.lemmatize(word) for word in pattern_words]
        for w in all_words:
            bag.append(1) if w in pattern_words else bag.append(0) # Create BoW Array by appending 1 for each instanc

        output_row = list(output_empty)
        output_row[all_tags.index(doc[1])] = 1
        training.append([bag, output_row])
        op.append(doc[1])

    training = np.array(training)

    train_x = list(training[:,0])
    train_y = list(training[:,1])

    return train_x, train_y, op
```

executed in 6ms, finished 20:55:37 2021-12-15

```
full_x, full_y, full_op = create_input_data(total_words, total_tags, total_documents)
print(np.array(full_x).shape, np.array(full_y).shape, np.array(full_op).shape)
```

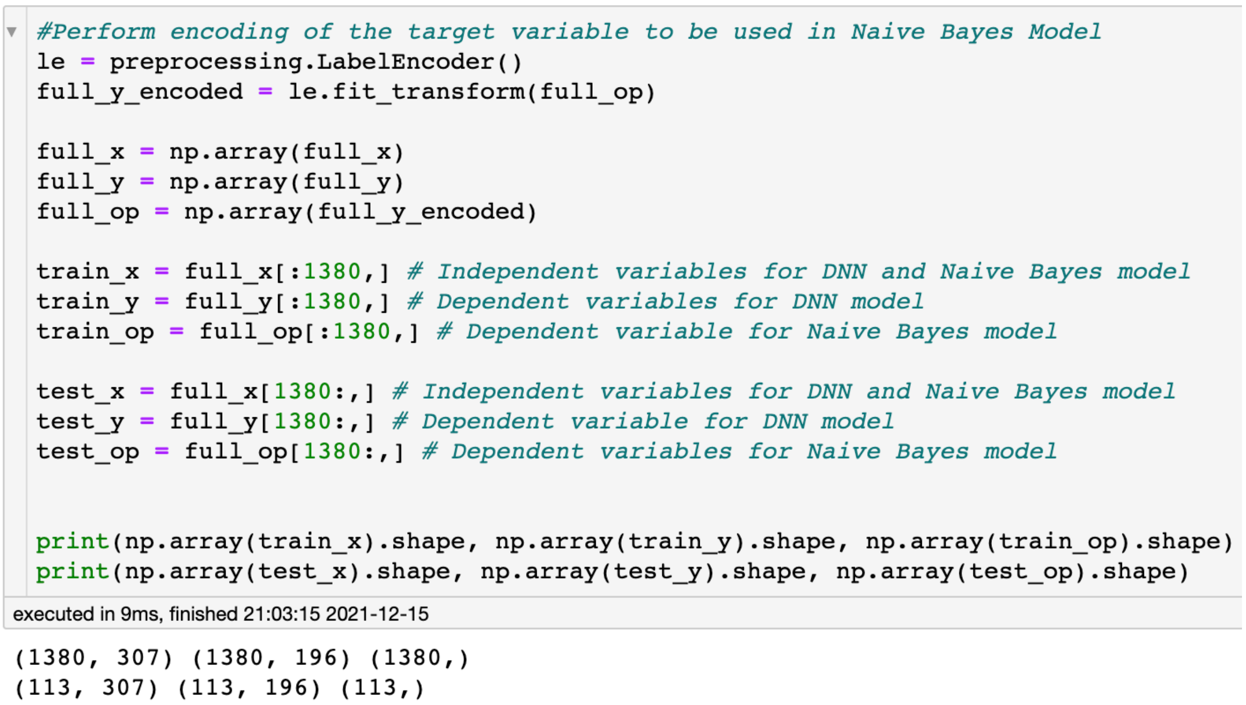
executed in 936ms, finished 20:55:39 2021-12-15

```
(1493, 307) (1493, 196) (1493,)
```

Figure 8: Create a Word Vector for training

3.4 Train and Validation split

The input data is then split into train and validation set as shown in Figure: 9.



```
#Perform encoding of the target variable to be used in Naive Bayes Model
le = preprocessing.LabelEncoder()
full_y_encoded = le.fit_transform(full_op)

full_x = np.array(full_x)
full_y = np.array(full_y)
full_op = np.array(full_y_encoded)

train_x = full_x[:1380,] # Independent variables for DNN and Naive Bayes model
train_y = full_y[:1380,] # Dependent variables for DNN model
train_op = full_op[:1380,] # Dependent variable for Naive Bayes model

test_x = full_x[1380:,] # Independent variables for DNN and Naive Bayes model
test_y = full_y[1380:,] # Dependent variable for DNN model
test_op = full_op[1380:,] # Dependent variables for Naive Bayes model


print(np.array(train_x).shape, np.array(train_y).shape, np.array(train_op).shape)
print(np.array(test_x).shape, np.array(test_y).shape, np.array(test_op).shape)
```

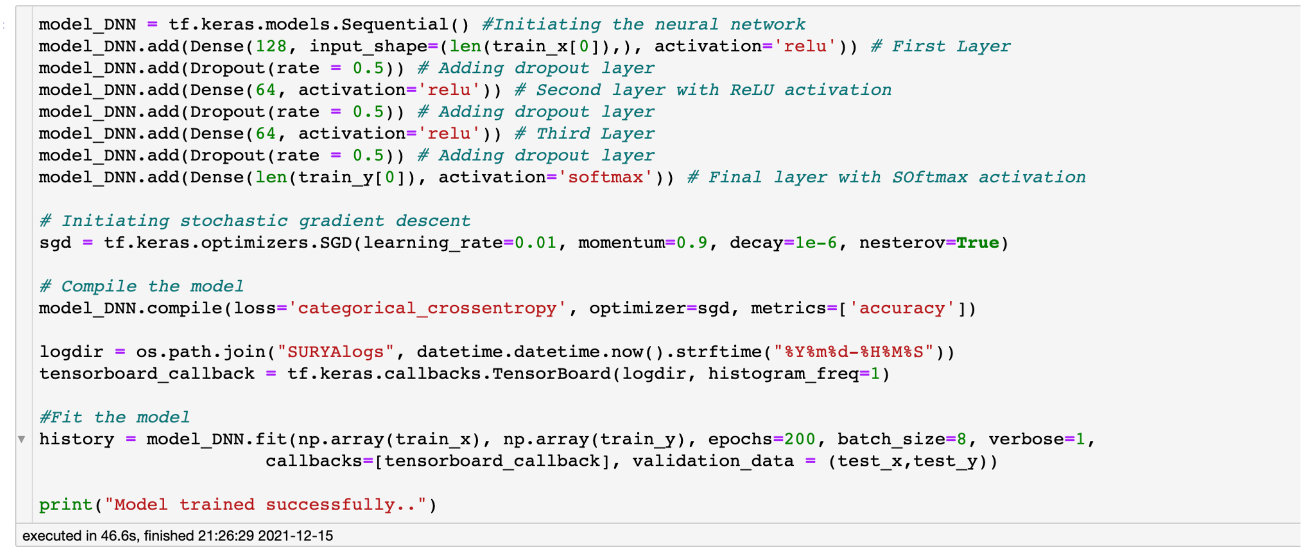
executed in 9ms, finished 21:03:15 2021-12-15

```
(1380, 307) (1380, 196) (1380,)
(113, 307) (113, 196) (113,)
```

Figure 9: Create Training and Validation Dataset for the DNN and Naive Bayes model

4 Model Initialisation and Implementation

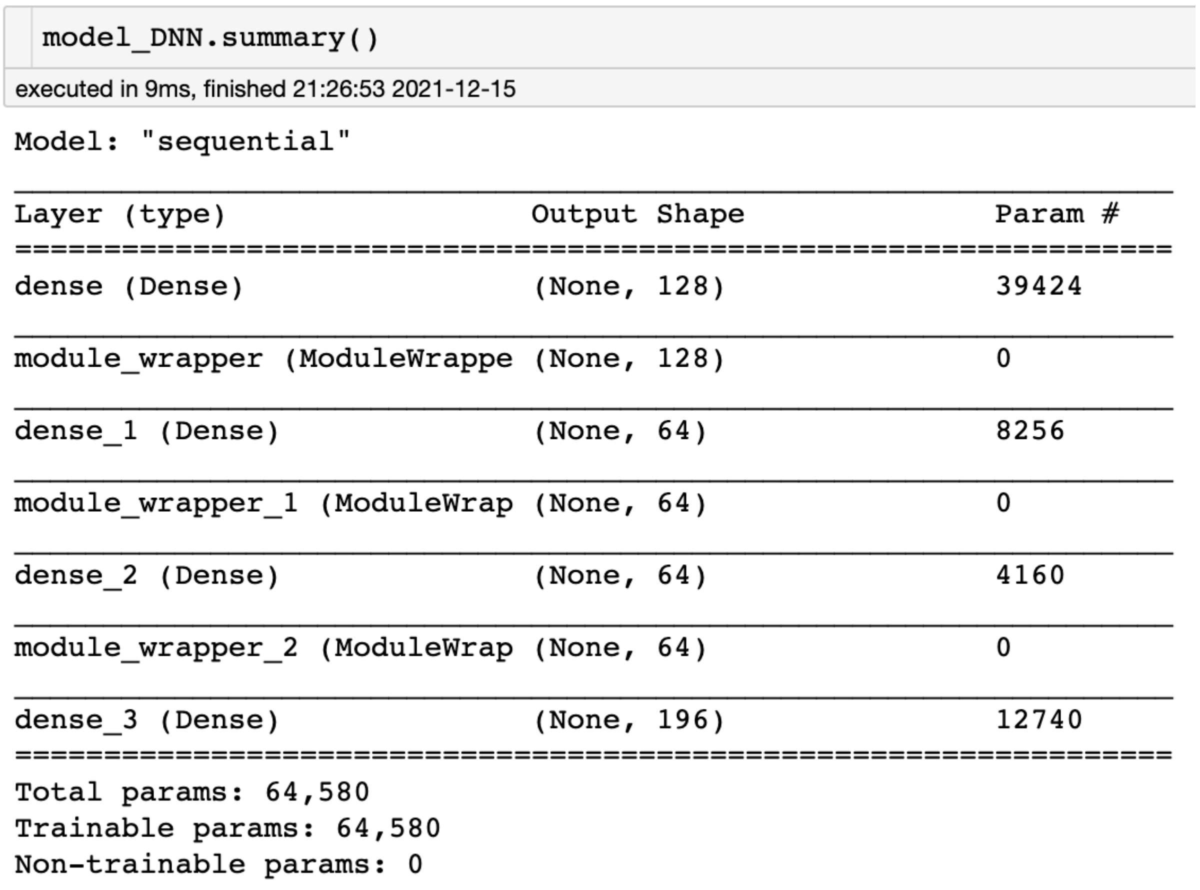
A multi-layered deep neural network with 4 layers (Figure:10) was created. A total of 64,580 trainable parameters initialized (Figure: 11) in the model.



```
model_DNN = tf.keras.models.Sequential() #Initiating the neural network
model_DNN.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu')) # First Layer
model_DNN.add(Dropout(rate = 0.5)) # Adding dropout layer
model_DNN.add(Dense(64, activation='relu')) # Second layer with ReLU activation
model_DNN.add(Dropout(rate = 0.5)) # Adding dropout layer
model_DNN.add(Dense(64, activation='relu')) # Third Layer
model_DNN.add(Dropout(rate = 0.5)) # Adding dropout layer
model_DNN.add(Dense(len(train_y[0]), activation='softmax')) # Final layer with SOftmax activation

# Initiating stochastic gradient descent
sgd = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, decay=1e-6, nesterov=True)

# Compile the model
model_DNN.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

logdir = os.path.join("SURYAlogs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

#Fit the model
history = model_DNN.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=8, verbose=1,
                    callbacks=[tensorboard_callback], validation_data = (test_x,test_y))

print("Model trained successfully..")
```

executed in 46.6s, finished 21:26:29 2021-12-15

Figure 10: Deep Neural Network Model initialization and Fitting



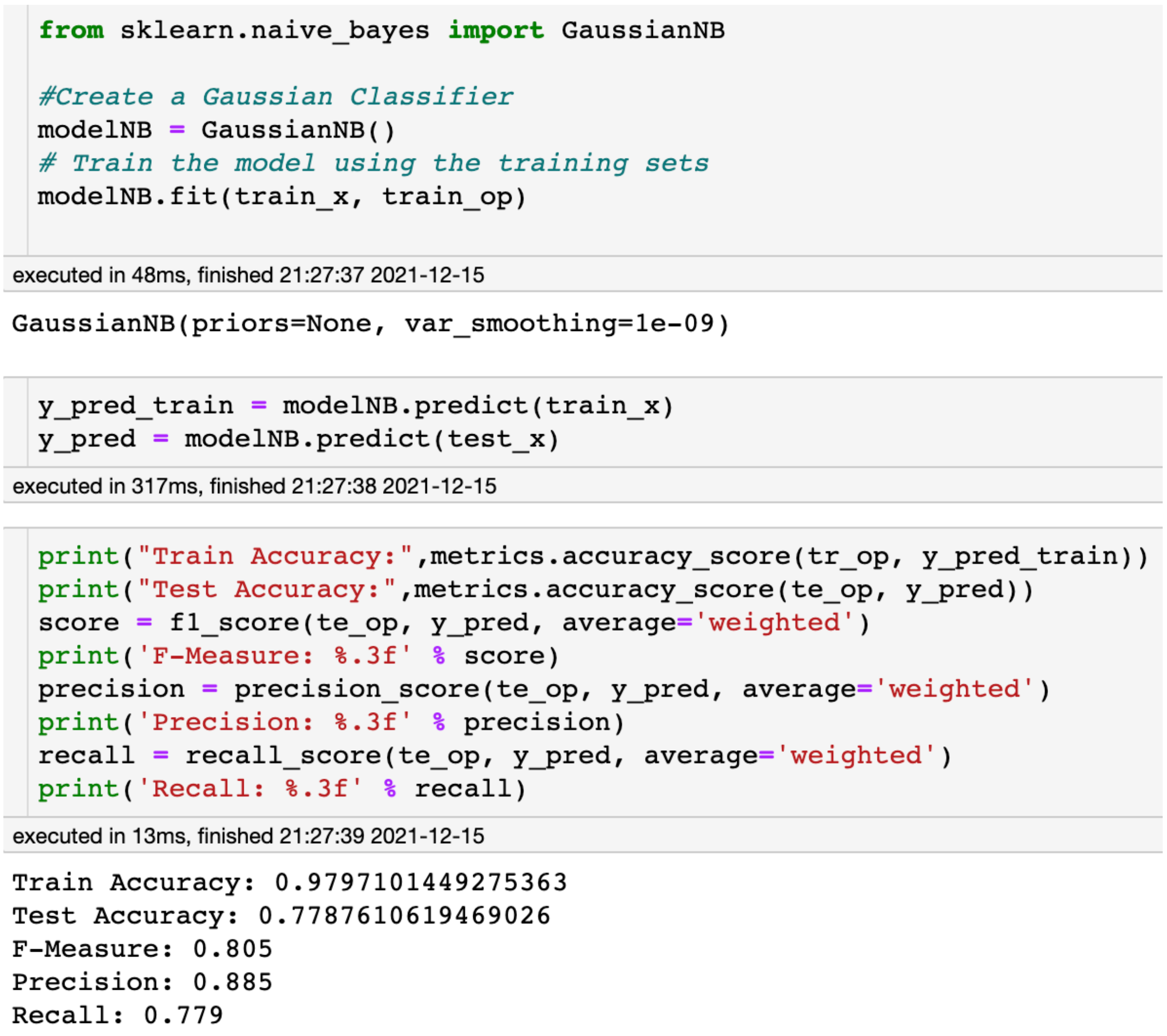
```
model_DNN.summary()
```

executed in 9ms, finished 21:26:53 2021-12-15

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               39424
_________________________________________________________________
module_wrapper (ModuleWrappe (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
module_wrapper_1 (ModuleWrap (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
module_wrapper_2 (ModuleWrap (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 196)               12740
=================================================================
Total params: 64,580
Trainable params: 64,580
Non-trainable params: 0
```

Figure 11: Deep Neural Network Model Summary

For comparing the performance and accuracy, a Naïve Bayes model was created and trained (Figure: 12) with the same input data.



```
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
modelNB = GaussianNB()
# Train the model using the training sets
modelNB.fit(train_x, train_op)
```

executed in 48ms, finished 21:27:37 2021-12-15

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
y_pred_train = modelNB.predict(train_x)
y_pred = modelNB.predict(test_x)
```

executed in 317ms, finished 21:27:38 2021-12-15

```
print("Train Accuracy:",metrics.accuracy_score(tr_op, y_pred_train))
print("Test Accuracy:",metrics.accuracy_score(te_op, y_pred))
score = f1_score(te_op, y_pred, average='weighted')
print('F-Measure: %.3f' % score)
precision = precision_score(te_op, y_pred, average='weighted')
print('Precision: %.3f' % precision)
recall = recall_score(te_op, y_pred, average='weighted')
print('Recall: %.3f' % recall)
```

executed in 13ms, finished 21:27:39 2021-12-15

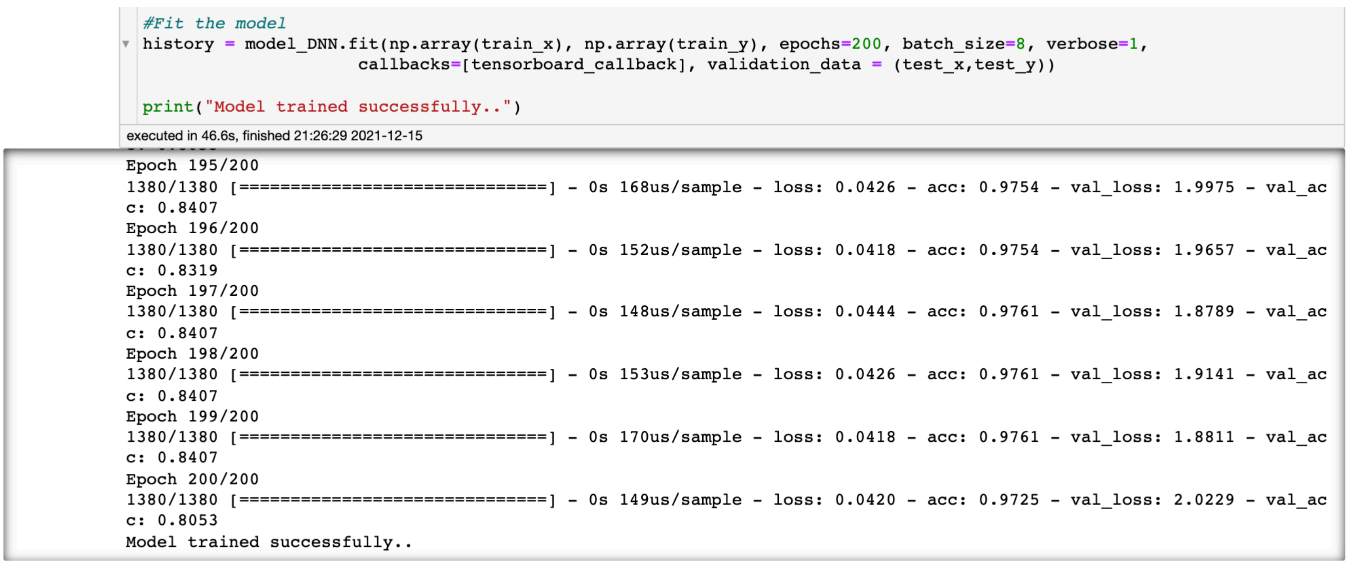
```
Train Accuracy: 0.9797101449275363
Test Accuracy: 0.7787610619469026
F-Measure: 0.805
Precision: 0.885
Recall: 0.779
```

Figure 12: Naive Bayes Model

5 Evaluation

Figure: 12 shows that after running for 200 epochs the DNN model attained training accuracy of 97.25 % and validation accuracy of 80.53 %. Similarly, after 200 epochs the training loss was 0.04 and validation loss was 2.02. Figure: 13 illustrates the Deep Neural Network Model’s Accuracy vs Epoch plot and Figure: 14 displays the Loss vs Epoch plot.

5.1 Evaluating the DNN model with Accuracy metrics

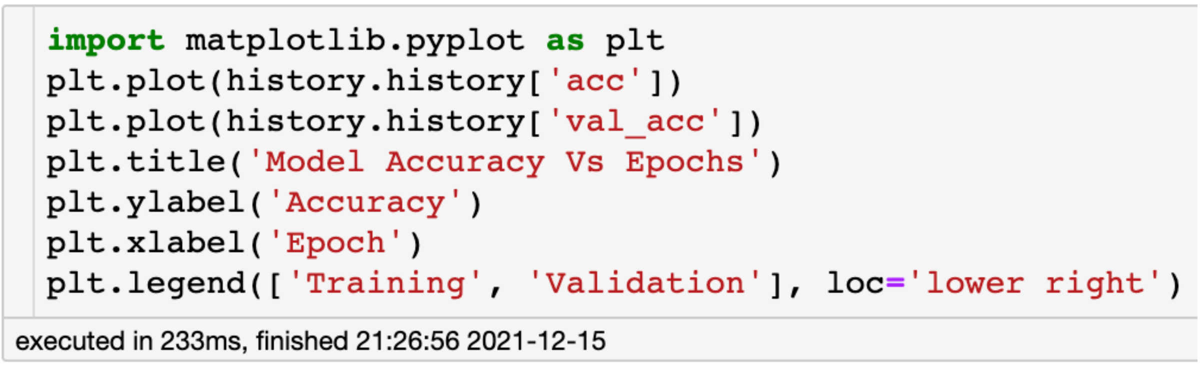


```
#Fit the model
history = model_DNN.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=8, verbose=1,
                        callbacks=[tensorboard_callback], validation_data = (test_x,test_y))

print("Model trained successfully..")
```

executed in 46.6s, finished 21:26:29 2021-12-15

```
Epoch 195/200
1380/1380 [==============================] - 0s 168us/sample - loss: 0.0426 - acc: 0.9754 - val_loss: 1.9975 - val_ac
c: 0.8407
Epoch 196/200
1380/1380 [==============================] - 0s 152us/sample - loss: 0.0418 - acc: 0.9754 - val_loss: 1.9657 - val_ac
c: 0.8319
Epoch 197/200
1380/1380 [==============================] - 0s 148us/sample - loss: 0.0444 - acc: 0.9761 - val_loss: 1.8789 - val_ac
c: 0.8407
Epoch 198/200
1380/1380 [==============================] - 0s 153us/sample - loss: 0.0426 - acc: 0.9761 - val_loss: 1.9141 - val_ac
c: 0.8407
Epoch 199/200
1380/1380 [==============================] - 0s 170us/sample - loss: 0.0418 - acc: 0.9761 - val_loss: 1.8811 - val_ac
c: 0.8407
Epoch 200/200
1380/1380 [==============================] - 0s 149us/sample - loss: 0.0420 - acc: 0.9725 - val_loss: 2.0229 - val_ac
c: 0.8053
Model trained successfully..
```

Figure 13: DNN Model Accuracy Vs Epoch



```
import matplotlib.pyplot as plt
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy Vs Epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
```

executed in 233ms, finished 21:26:56 2021-12-15

```
<matplotlib.legend.Legend at 0x7ff24558f940>
```

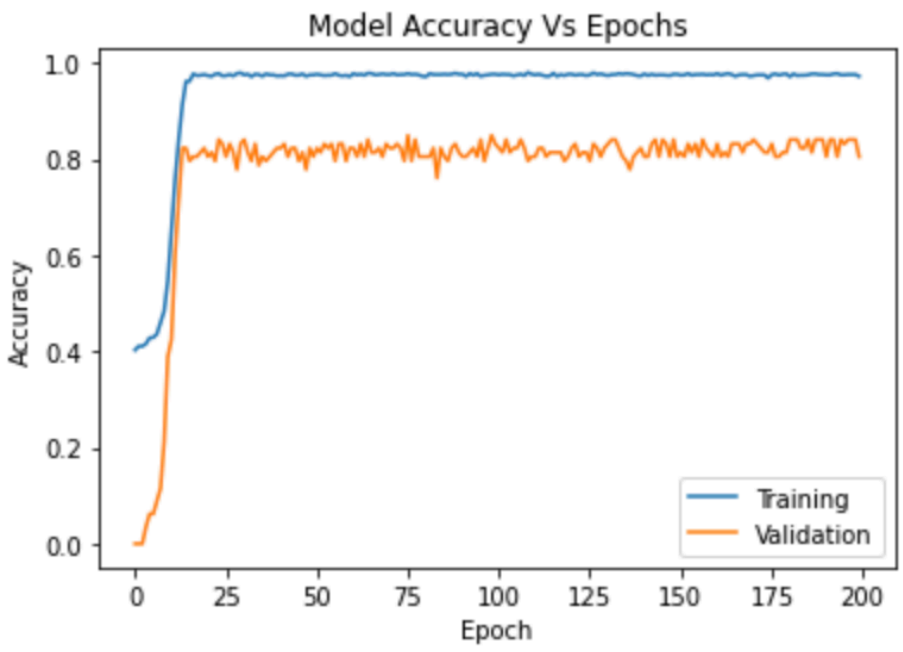
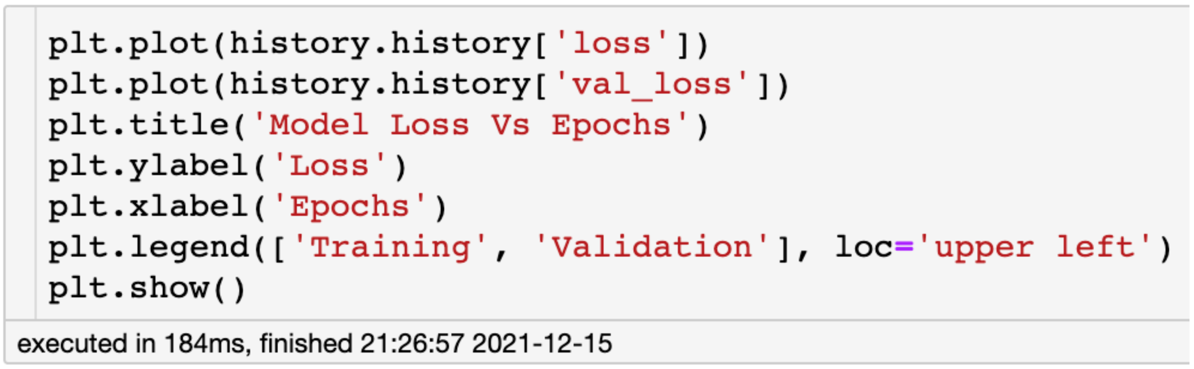


Figure 14: DNN Model Accuracy Vs Epoch



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss Vs Epochs')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Training', 'Validation'], loc='upper left')
plt.show()
```

```
executed in 184ms, finished 21:26:57 2021-12-15
```

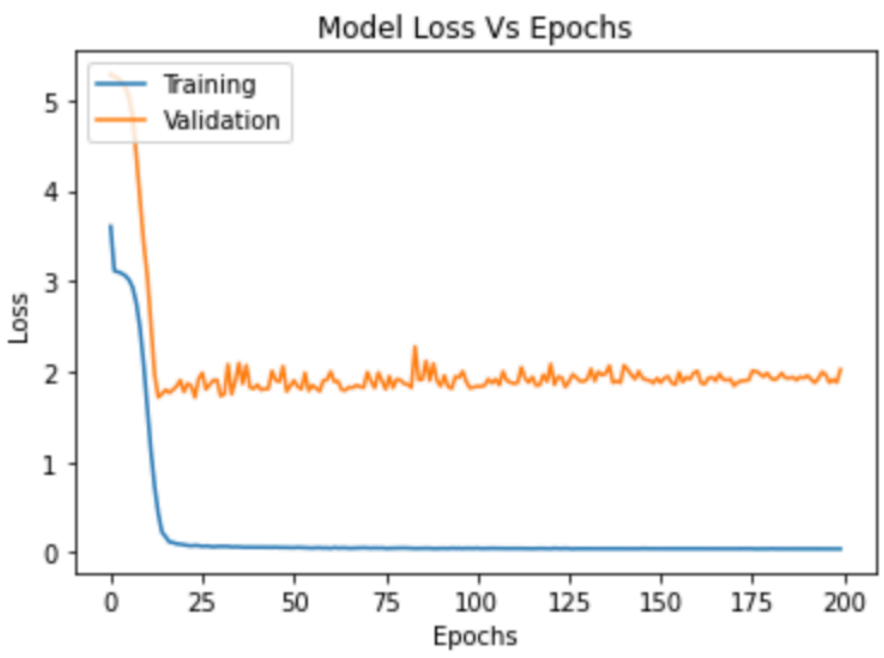
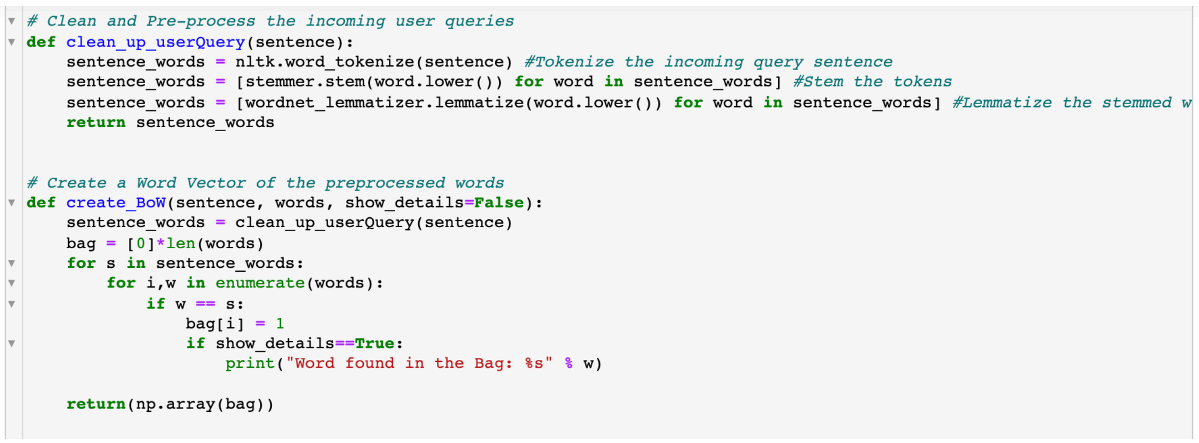


Figure 15: DNN Model Loss Vs Epoch

5.2 Evaluating the DNN model with real-time user queries

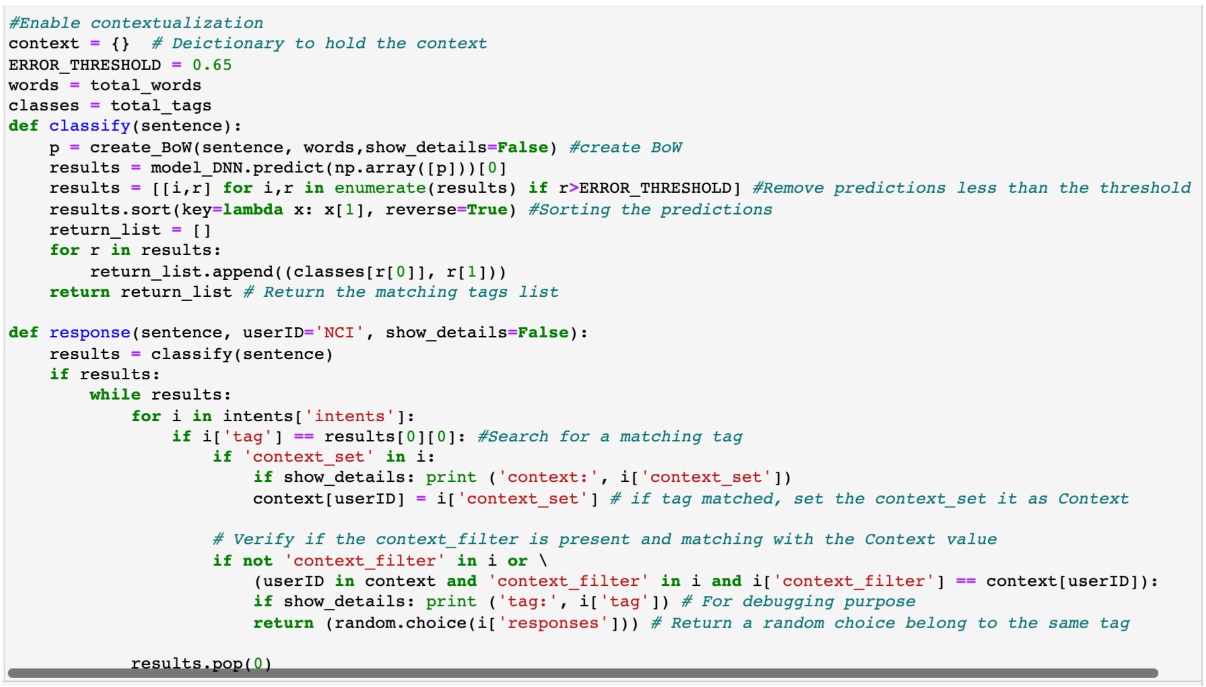
At first the incoming user query sentence was cleaned, pre-processed and converted into a Bag of Words (Figure: 16).



```
# Clean and Pre-process the incoming user queries
def clean_up_userQuery(sentence):
    sentence_words = nltk.word_tokenize(sentence) #Tokenize the incoming query sentence
    sentence_words = [stemmer.stem(word.lower()) for word in sentence_words] #Stem the tokens
    sentence_words = [wordnet_lemmatizer.lemmatize(word.lower()) for word in sentence_words] #Lemmatize the stemmed w
    return sentence_words


# Create a Word Vector of the preprocessed words
def create_BoW(sentence, words, show_details=False):
    sentence_words = clean_up_userQuery(sentence)
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                bag[i] = 1
                if show_details==True:
                    print("Word found in the Bag: %s" % w)

    return(np.array(bag))
```

Figure 16: Clean and transform the User Input

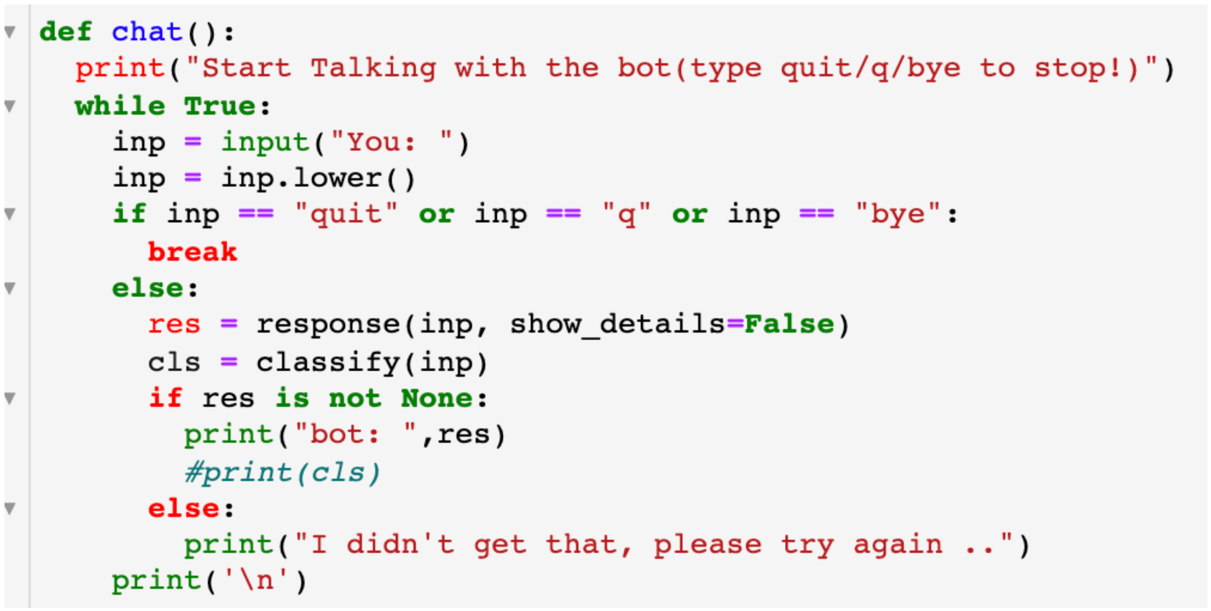
Secondly, the user query was predicted by the trained DNN model, and the prediction was matched against the intent tags. Once a matching tag found, a sample response from that intent is sent to the user. Also, the contextualization capability was added to the chatbot using python dictionary data structure (Figure: 17).



```python
#Enable contextualization
context = {}  # Deictionary to hold the context
ERROR_THRESHOLD = 0.65
words = total_words
classes = total_tags
def classify(sentence):
    p = create_BoW(sentence, words,show_details=False) #create BoW
    results = model_DNN.predict(np.array([p]))[0]
    results = [[i,r] for i,r in enumerate(results) if r>ERROR_THRESHOLD] #Remove predictions less than the threshold
    results.sort(key=lambda x: x[1], reverse=True) #Sorting the predictions
    return_list = []
    for r in results:
        return_list.append((classes[r[0]], r[1]))
    return return_list # Return the matching tags list

def response(sentence, userID='NCI', show_details=False):
    results = classify(sentence)
    if results:
        while results:
            for i in intents['intents']:
                if i['tag'] == results[0][0]: #Search for a matching tag
                    if 'context_set' in i:
                        if show_details: print ('context:', i['context_set'])
                        context[userID] = i['context_set'] # if tag matched, set the context_set it as Context

                    # Verify if the context_filter is present and matching with the Context value
                    if not 'context_filter' in i or \
                        (userID in context and 'context_filter' in i and i['context_filter'] == context[userID]):
                        if show_details: print ('tag:', i['tag']) # For debugging purpose
                        return (random.choice(i['responses'])) # Return a random choice belong to the same tag

            results.pop(0)
```

Figure 17: Classify and add Contextualization capabilities

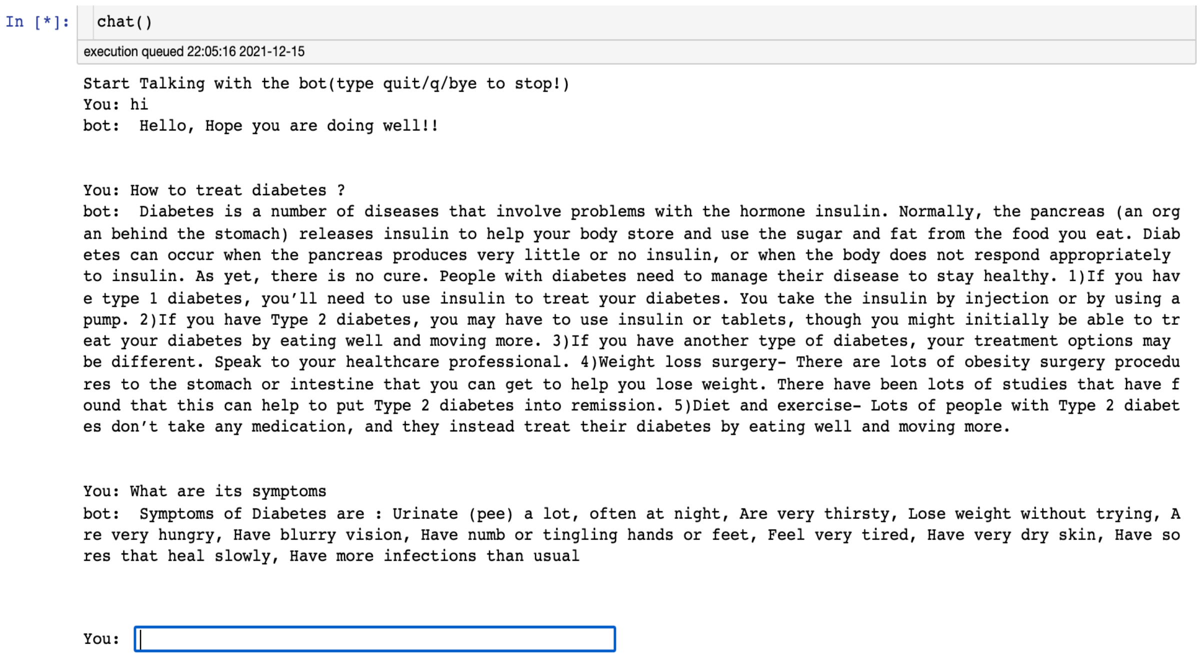
The customized function chat() (Figure: 18) simulates the interaction between the chatbot and user by providing a input box.



```python
def chat():
  print("Start Talking with the bot(type quit/q/bye to stop!)")
  while True:
    inp = input("You:  ")
    inp = inp.lower()
    if inp == "quit" or inp == "q" or inp == "bye":
      break
    else:
      res = response(inp, show_details=False)
      cls = classify(inp)
      if res is not None:
        print("bot:  ",res)
        #print(cls)
      else:
        print("I didn't get that, please try again ..")
    print('\n')
```

Figure 18: Function to interact with User

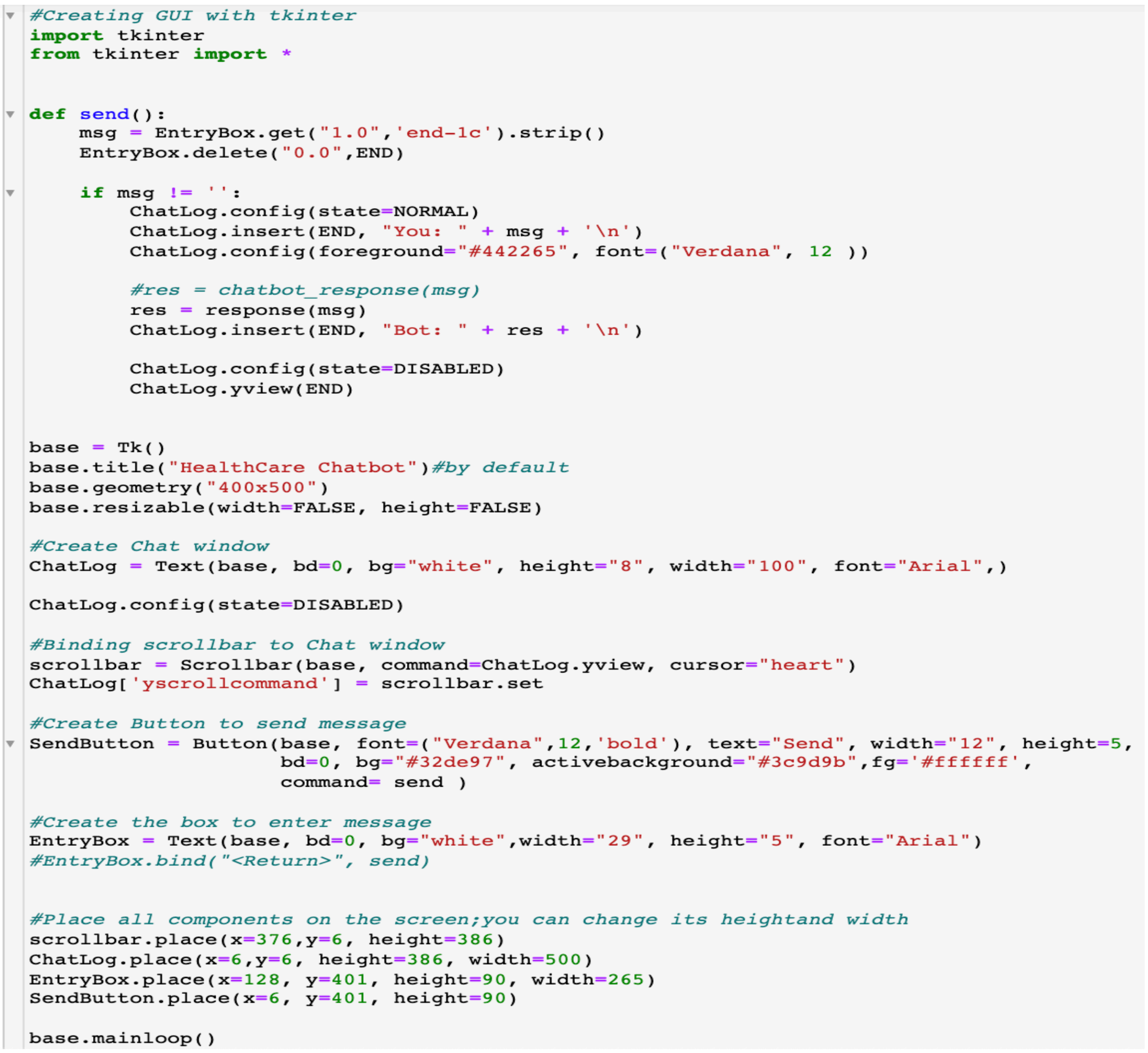
As shown in Figure:19, the user is able to ask healthcare related queries to the chatbot, and the model is providing the user appropriate responses.



```
In [*]:  chat()
execution queued 22:05:16 2021-12-15

Start Talking with the bot(type quit/q/bye to stop!)
You: hi
bot:  Hello, Hope you are doing well!!


You: How to treat diabetes ?
bot:  Diabetes is a number of diseases that involve problems with the hormone insulin. Normally, the pancreas (an org
an behind the stomach) releases insulin to help your body store and use the sugar and fat from the food you eat. Diab
etes can occur when the pancreas produces very little or no insulin, or when the body does not respond appropriately
to insulin. As yet, there is no cure. People with diabetes need to manage their disease to stay healthy. 1)If you hav
e type 1 diabetes, you'll need to use insulin to treat your diabetes. You take the insulin by injection or by using a
pump. 2)If you have Type 2 diabetes, you may have to use insulin or tablets, though you might initially be able to tr
eat your diabetes by eating well and moving more. 3)If you have another type of diabetes, your treatment options may
be different. Speak to your healthcare professional. 4)Weight loss surgery- There are lots of obesity surgery procedu
res to the stomach or intestine that you can get to help you lose weight. There have been lots of studies that have f
ound that this can help to put Type 2 diabetes into remission. 5)Diet and exercise- Lots of people with Type 2 diabet
es don't take any medication, and they instead treat their diabetes by eating well and moving more.


You: What are its symptoms
bot:  Symptoms of Diabetes are : Urinate (pee) a lot, often at night, Are very thirsty, Lose weight without trying, A
re very hungry, Have blurry vision, Have numb or tingling hands or feet, Feel very tired, Have very dry skin, Have so
res that heal slowly, Have more infections than usual


You:
```

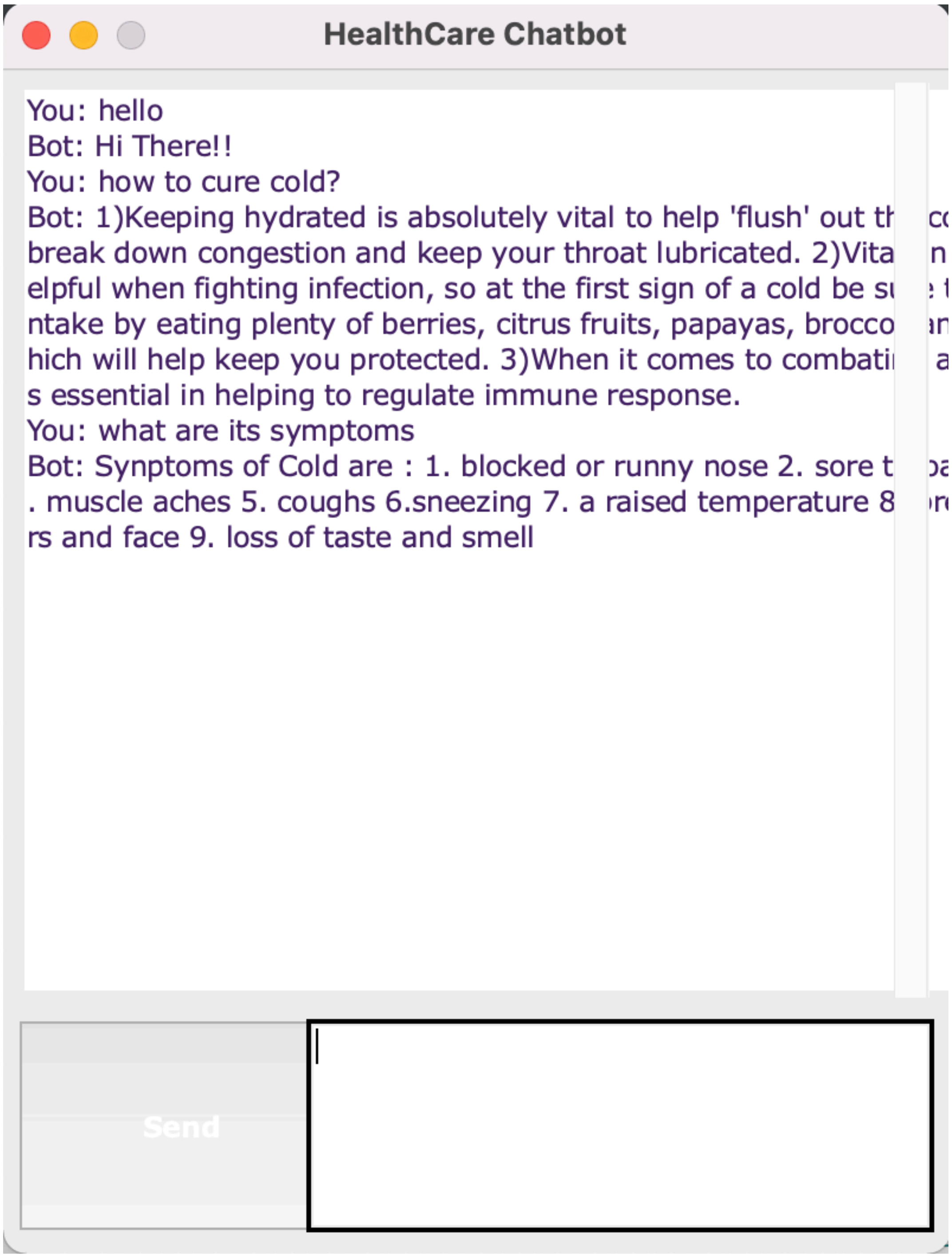
Figure 19: Sample conversation with the user

Figure: 20 shows a user defined GUI function which takes user input through and interactive window (Figure: 21) and provide suitable responses.



```
#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')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

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

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


base = Tk()
base.title("HealthCare Chatbot")#by default
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="100", font="Arial",)

ChatLog.config(state=DISABLED)

#Binding 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 )

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


#Place all components on the screen;you can change its heightand width
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=500)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```

Figure 20: Function to create a Chatbot GUI



```
You: hello
Bot: Hi There!!
You: how to cure cold?
Bot: 1)Keeping hydrated is absolutely vital to help 'flush' out th  co
break down congestion and keep your throat lubricated. 2)Vita  n
elpful when fighting infection, so at the first sign of a cold be su  e t
ntake by eating plenty of berries, citrus fruits, papayas, brocco  an
hich will help keep you protected. 3)When it comes to combati  a
s essential in helping to regulate immune response.
You: what are its symptoms
Bot: Synptoms of Cold are : 1. blocked or runny nose 2. sore t  pa
. muscle aches 5. coughs 6.sneezing 7. a raised temperature 8  pr
rs and face 9. loss of taste and smell
```

Figure 21: Sample conversation with the user in chatbot GUI