Phase 5

PROJECT DOCUMENTATION AND SUBMISSION

Project Name : CREATE A CHATBOT IN PYTHON

Team ID : 8932

Introduction:

Chatbot often powered by sophisticated language models like GPT, have gained popularity in natural language processing tasks. These models can generate human-like responses based on input text and are trained on vast amounts of diverse textual data.

To effectively train such models, a crucial step is the preprocessing of the dataset. In this Python script, we outline a systematic approach to prepare a dataset for training a chatbot.

Problem Statement:

The problem is to build an AI-powered chatbot that uses Natural language processing to convers with the user and answer their queries. The system aims to provide knowledge and information about various topics for easy and effective understanding , provide a user friendly conversations to make it easy.

Objective: To answer inquiries, provide study tools, provide feedback and provide a more interesting and interactive environment for the user.

Data: We have access to a dataset containing queries and their answers and website pages with people's questions in various topics .This data set is used to train put chatbot

Key Challenges faced while creating the chatbot:

Content Knowledge:

Ensuring the chatbot has a comprehensive understanding of various topics and basic knowledge to answer various quires and conversation of the user.

Natural Language Processing (NLP):

Developing robust NLP capabilities to understand and generate human language accurately can be challenging, particularly in the context of code explanations and examples.

Interactivity:

Creating an engaging and interactive learning experience, including code execution and visualizations, can be complex to implement.

Scalability and Performance:

Ensuring the chatbot can handle a large user base, respond quickly, and scale with increasing demand is a technical challenge.

Feedback and Adaptation:

Implementing mechanisms for gathering user feedback and using it to improve the chatbot's performance and content is an ongoing challenge.

Multimodal Learning:

Supporting various learning modes, such as text, images, videos, or voice interactions, can enhance the chatbot's usability but also adds complexity.

Maintenance and Updates:

Python evolves, and so does the best practice .Keeping the chatbot's content and capabilities up-to-date is an ongoing effort.

User Engagement:

Maintaining user interest and motivation to continue learning about certain topics and converse in user friendly way.

Integration:

If the chatbot is part of a broader educational platform, integrating it with other tools and systems seamlessly is essential.

Design Thinking Approach:

Functionality:

The scope of this chatbot is to provide knowledge and information about the topic user is asking for , and it should provide user friendly conversation to make the conversation more encaging.

User Interface:

The chatbot will be integrated in website, this design is a user-friendly interface for interactions which includes text box for text queries and camera lens for image type queries.

Natural Language Processing (NLP):

Using NLP techniques such as keyword extraction, intent recognition, and sentiment analysis, the chatbot is trained to comprehend and respond to user queries.

Response:

The bot should response to the user in the manner that the converser understand the content in one go without any confusion in the content.

Testing and Improvement:

The chatbot is tested and refine continuously with the help of various queries and questions on from different websites and queries dataset.

Integration:

Creating a web interface for users to interact with the chatbot. Building a custom web interface using web development technologies like HTML,CSS and JavaScript, or use existing platforms and tools and appending the chatbot to website.

Steps involved in designing:

Tokenization: The input text is broken down into smaller units called tokens. Tokens can be words or even subwords, depending on the language and model used.

Text Preprocessing: The text is preprocessed to remove noise, like punctuation, capitalization, and stopwords (common words like "the," "and," "in" that don't carry much meaning).

Extraction: The chatbot extracts features from the tokens to represent the input text. These features can include word embeddings, which map words to numerical vectors, making it easier for the model to understand the text's meaning.

NLP Model: The chatbot uses a machine learning or deep learning model, often based on neural networks, to analyze the input. Common models include recurrent neural networks (RNNs), convolutional neural networks (CNNs), or transformer-based models like GPT (Generative Pre-trained Transformer).

Understanding: The model processes the input text and tries to understand the user's intent. This involves recognizing entities (like names, places, or dates) and determining the context.

Matching: The chatbot compares the user's input to predefined patterns or rules to identify the most appropriate response. This might involve searching a database of responses or using predefined conversational templates.Generating Response: Based on the understanding and matching results, the chatbot generates a response. This can be done using template-based responses, rule based responses, or by generating text using the model itself.

Innovation techniques:

Various Natural Language Processing Algorithms are used for reading and replying in ChatBot.

Named Entity Recognition (NER): Identifies and classifies entities like names, dates, and locations in text.

Text Classification: Categorizes text into predefined classes (e.g., sentiment analysis, spam detection).

Text Clustering: Groups similar documents or sentences together based on their content.

Information Retrieval: Retrieves relevant documents from a large corpus in response to a query.

Machine Translation: Translates text from one language to another.

Word Embeddings: Represent words in a continuous vector space, e.g., Word2Vec, GloVe.

Seq2Seq Models: Used in tasks like machine translation, chatbots, and text summarization.

Keyword Extraction: By definition, keyword extraction is the automated process of extracting the most relevant information from text using AI and machine learning algorithms.

Steps involved in creation of chatbot:

Text processing using NLP:

NLP is a method for computers to intelligently analyse, comprehend, and derive meaning from human language. Developers can use NLP to organise and structure knowledge in order to execute tasks like automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, audio recognition, and topic segmentation**.**

Data preprocessing:

Data preprocessing is a crucial step in the data mining and data analysis process that involves transforming raw data into a format that can be understood and analysed by computers and machine learning algorithms.

```
In [1]:
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.layers import TextVectorization
import re,string
from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization
```

```
In [2]:
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import string
```

Statement segmentation:

Reading the dataset

```
In [11]: with open('Documents/dialogs.txt', 'r', encoding='utf-8') as f:
             raw_data = f.read()
             print(raw_data)
```

```
hi, how are you doing?  i'm fine. how about yourself?
i'm fine. how about yourself?   i'm pretty good. thanks for asking.
i'm pretty good. thanks for asking.     no problem. so how have you been?
no problem. so how have you been?       i've been great. what about you?
i've been great. what about you?        i've been good. i'm in school right now.
i've been good. i'm in school right now.        what school do you go to?
what school do you go to?       i go to pcc.
i go to pcc.    do you like it there?
do you like it there?   it's okay. it's a really big campus.
it's okay. it's a really big campus.    good luck with school.
good luck with school.  thank you very much.
how's it going? i'm doing well. how about you?
i'm doing well. how about you?  never better, thanks.
never better, thanks.   so how have you been lately?
so how have you been lately?    i've actually been pretty good. you?
i've actually been pretty good. you?    i'm actually in school right now.
i'm actually in school right now.       which school do you attend?
which school do you attend?     i'm attending pcc right now.
i'm attending pcc right now.    are you enjoying it there?
```

Question and Answers:

```
In [3]: #paried list of question and corresponding answer
        QA_list=[QA.split('\t') for QA in data.split('\n')]
        print(QA_list[:5])
```

```
[['hi, how are you doing?', "i'm fine. how about yourself?"], ["i'm fine. how about yoursel
f?", "i'm pretty good. thanks for asking."], ["i'm pretty good. thanks for asking.", 'no pro
blem. so how have you been?'], ['no problem. so how have you been?', "i've been great. what
about you?"], ["i've been great. what about you?", "i've been good. i'm in school right no
w."]]
```

Normalization:

The goal of normalizing text is to group related tokens together, where tokens are usually the words in the text.

Depending on the text you are working with and the type of analysis you are doing, you might not need all of the normalization techniques in this post.

Removing Punctuations:

```
In [35]: # Remove stopwords and punctuation
         stop_words = set(stopwords.words('english'))
         tokens = [word for word in tokens if word not in stop_words and word not in string.punctuation]
         print(tokens)
```

```
['hi', "'m", 'fine', "'m", 'fine', "'m", 'pretty', 'good', 'thanks', 'asking', "'m", 'pretty', 'good', 'thanks', 'asking', 'p
roblem', 'problem', "'ve", 'great', "'ve", 'great', "'ve", 'good', "'m", 'school', 'right', "'ve", 'good', "'m", 'school', 'r
ight', 'school', 'go', 'school', 'go', 'go', 'pcc', 'go', 'pcc', 'like', 'like', "'s", 'okay', "'s", 'really', 'big', 'campu
s', "'s", 'okay', "'s", 'really', 'big', 'campus', 'good', 'luck', 'school', 'good', 'luck', 'school', 'thank', 'much', "'s",
'going', "'m", 'well', "'m", 'well', 'never', 'better', 'thanks', 'never', 'better', 'thanks', 'lately', 'lately', "'ve", 'ac
tually', 'pretty', 'good', "'ve", 'actually', 'pretty', 'good', "'m", 'actually', 'school', 'right', "'m", 'actually', 'schoo
l', 'right', 'school', 'attend', 'school', 'attend', "'m", 'attending', 'pcc', 'right', "'m", 'attending', 'pcc', 'right', 'e
njoying', 'enjoying', "'s", 'bad', 'lot', 'people', "'s", 'bad', 'lot', 'people', 'good', 'luck', 'good', 'luck', 'thanks',
'today', "'m", 'great', "'m", 'great', "'m", 'absolutely', 'lovely', 'thank', "'m", 'absolutely', 'lovely', 'thank', 'everyth
ing', "'s", 'good', 'everything', "'s", 'good', "n't", 'better', "n't", 'better', 'started', 'school', 'recently', 'started',
'school', 'recently', 'going', 'school', 'going', 'school', "'m", 'going', 'pcc', "'m", 'going', 'pcc', 'like', 'far', 'lik
e', 'far', 'like', 'far', 'classes', 'pretty', 'good', 'right', 'like', 'far', 'classes', 'pretty', 'good', 'right', 'wish',
'luck', "'s", 'ugly', 'day', 'today', 'know', 'think', 'may', 'rain', 'know', 'think', 'may', 'rain', "'s", 'middle', 'summe
r', "n't", 'rain', 'today', "'s", 'middle', 'summer', "n't", 'rain', 'today', 'would', 'weird', 'would', 'weird', 'yeah', 'es
pecially', 'since', "'s", 'ninety', 'degrees', 'outside', 'yeah', 'especially', 'since', "'s", 'ninety', 'degrees', 'outsid
e', 'know', 'would', 'horrible', 'rained', 'hot', 'outside', 'know', 'would', 'horrible', 'rained', 'hot', 'outside', 'yes',
'would', 'yes', 'would', 'really', 'wish', "n't", 'hot', 'every', 'day', 'really', 'wish', "n't", 'hot', 'every', 'day', 'c
a', "n't", 'wait', 'winter', 'ca', "n't", 'wait', 'winter', 'like', 'winter', 'sometimes', 'gets', 'cold', 'like', 'winter',
'sometimes', 'gets', 'cold', "'d", 'rather', 'cold', 'hot', "'d", 'rather', 'cold', 'hot', "n't", 'look', 'nice', 'outside',
```

```
In [ ]:
```

Converting Uppercase to Lowercase:

```
In [32]: # Lowercase all words
         tokens = [word.lower() for word in tokens]
         print(tokens)
```

```
['hi', ',', 'how', 'are', 'you', 'doing', '?', 'i', "'m", 'fine', '.', 'how', 'about', 'yourself', '?', 'i', "'m", 'fine',
'.', 'how', 'about', 'yourself', '?', 'i', "'m", 'pretty', 'good', '.', 'thanks', 'for', 'asking', '.', 'i', "'m", 'pretty',
'good', '.', 'thanks', 'for', 'asking', '.', 'no', 'problem', '.', 'so', 'how', 'have', 'you', 'been', '?', 'no', 'problem',
'.', 'so', 'how', 'have', 'you', 'been', '?', 'i', "'ve", 'been', 'great', '.', 'what', 'about', 'you', '?', 'i', "'ve", 'bee
n', 'great', '.', 'what', 'about', 'you', '?', 'i', "'ve", 'been', 'good', '.', 'i', "'m", 'in', 'school', 'right', 'now',
'.', 'i', "'ve", 'been', 'good', '.', 'i', "'m", 'in', 'school', 'right', 'now', '.', 'what', 'school', 'do', 'you', 'go', 't
o', '?', 'what', 'school', 'do', 'you', 'go', 'to', '?', 'i', 'go', 'to', 'pcc', '.', 'i', 'go', 'to', 'pcc', '.', 'do', 'yo
u', 'like', 'it', 'there', '?', 'do', 'you', 'like', 'it', 'there', '?', 'it', "'s", 'okay', '.', 'it', "'s", 'a', 'really',
'big', 'campus', '.', 'it', "'s", 'okay', '.', 'it', "'s", 'a', 'really', 'big', 'campus', '.', 'good', 'luck', 'with', 'scho
ol', '.', 'good', 'luck', 'with', 'school', '.', 'thank', 'you', 'very', 'much', '.', 'how', "'s", 'it', 'going', '?', 'i',
"'m", 'doing', 'well', '.', 'how', 'about', 'you', '?', 'i', "'m", 'doing', 'well', '.', 'how', 'about', 'you', '?', 'never',
'better', ',', 'thanks', '.', 'never', 'better', ',', 'thanks', '.', 'so', 'how', 'have', 'you', 'been', 'lately', '?', 'so',
'how', 'have', 'you', 'been', 'lately', '?', 'i', "'ve", 'actually', 'been', 'pretty', 'good', '.', 'you', '?', 'i', "'ve",
'actually', 'been', 'pretty', 'good', '.', 'you', '?', 'i', "'m", 'actually', 'in', 'school', 'right', 'now', '.', 'i', "'m",
'actually', 'in', 'school', 'right', 'now', '.', 'which', 'school', 'do', 'you', 'attend', '?', 'which', 'school', 'do', 'yo
u', 'attend', '?', 'i', "'m", 'attending', 'pcc', 'right', 'now', '.', 'i', "'m", 'attending', 'pcc', 'right', 'now', '.', 'a
re', 'you', 'enjoying', 'it', 'there', '?', 'are', 'you', 'enjoying', 'it', 'there', '?', 'it', "'s", 'not', 'bad', '.', 'the
re', 'are', 'a', 'lot', 'of', 'people', 'there', '.', 'it', "'s", 'not', 'bad', '.', 'there', 'are', 'a', 'lot', 'of', 'peopl
e', 'there', '.', 'good', 'luck', 'with', 'that', '.', 'good', 'luck', 'with', 'that', '.', 'thanks', '.', 'how', 'are', 'yo
```

Tokenization:

Tokenization, when applied to data security, is the process of substituting a sensitive data element with a non-sensitive equivalent, referred to as a token, that has no intrinsic or exploitable meaning or value.

In [38]:
```
# Lemmatize words
lemmatizer = WordNetLemmatizer()
tokens = [lemmatizer.lemmatize(word) for word in tokens]
```

In [39]:
```
# Preprocess data
processed_data = [preprocess(qa) for qa in raw_data.split('\n')]
```

In [8]:
```
def sequences2ids(sequence):
    return vectorize_layer(sequence)

def ids2sequences(ids):
    decode=''
    if type(ids)==int:
        ids=[ids]
    for id in ids:
        decode+=vectorize_layer.get_vocabulary()[id]+' '
    return decode

x=sequences2ids(df['encoder_inputs'])
yd=sequences2ids(df['decoder_inputs'])
y=sequences2ids(df['decoder_targets'])

print(f'Question sentence: hi , how are you ?')
print(f'Question to tokens: {sequences2ids("hi , how are you ?")[:10]}')
print(f'Encoder input shape: {x.shape}')
print(f'Decoder input shape: {yd.shape}')
print(f'Decoder target shape: {y.shape}')
```

```
Question sentence: hi , how are you ?
Question to tokens: [1971    9   45   24    8    7    0    0    0    0]
Encoder input shape: (3725, 30)
Decoder input shape: (3725, 30)
Decoder target shape: (3725, 30)
```

In [9]:

```
print(f'Encoder input: {x[0][:12]} ...')
print(f'Decoder input: {yd[0][:12]} ...')    # shifted by one time step of the target as input t
o decoder is the output of the previous timestep
print(f'Decoder target: {y[0][:12]} ...')
```

```
Encoder input: [1971    9   45   24    8  194    7    0    0    0    0    0] ...
Decoder input: [  4   6   5  38 646   3  45  41 563   7   2   0] ...
Decoder target: [  6   5  38 646   3  45  41 563   7   2   0   0] ...
```

In [9]:

```
def tokenize(lang):
    lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
      filters='')
```

Word Embedding:

It is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lowerdimensional space. It allows words with similar meanings to have a similar representation. They can also approximate meaning. A word vector with 50 values can represent 50 unique features.

In [10]:

```
def vectorization(lang_tokenizer,lang):

    #word embedding for training the neural network
    tensor = lang_tokenizer.texts_to_sequences(lang)

    tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                           padding='post')

    return tensor
```

Loading dataset :

In [11]:

```
def load_Dataset(data,size=None):

    if(size!=None):
        y,X=data[:size]
    else:
        y,X=data

    X_tokenizer=tokenize(X)
    y_tokenizer=tokenize(y)

    X_tensor=vectorization(X_tokenizer,X)
    y_tensor=vectorization(y_tokenizer,y)

    return  X_tensor,X_tokenizer, y_tensor, y_tokenizer
```

In [12]:

```
size=30000
data=preprocessed_answers,preprocessed_questions\


X_tensor,X_tokenizer, y_tensor, y_tokenizer=load_Dataset(data,size)
```

In [13]:

```
# Calculate max_length of the target tensors
max_length_y, max_length_X = y_tensor.shape[1], X_tensor.shape[1]
```

Text Cleaning:

[4]:

```
def clean_text(text):
    text=re.sub('-',' ',text.lower())
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
    text=re.sub('[3]',' 3 ',text)
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    text=re.sub('[6]',' 6 ',text)
    text=re.sub('[7]',' 7 ',text)
    text=re.sub('[8]',' 8 ',text)
    text=re.sub('[9]',' 9 ',text)
    text=re.sub('[0]',' 0 ',text)
    text=re.sub('[,]',' , ',text)
    text=re.sub('[?]',' ? ',text)
    text=re.sub('[!]',' ! ',text)
    text=re.sub('[$]',' $ ',text)
    text=re.sub('[&]',' & ',text)
    text=re.sub('[/]',' / ',text)
    text=re.sub('[:]',' : ',text)
    text=re.sub('[;]',' ; ',text)
    text=re.sub('[*]',' * ',text)
```

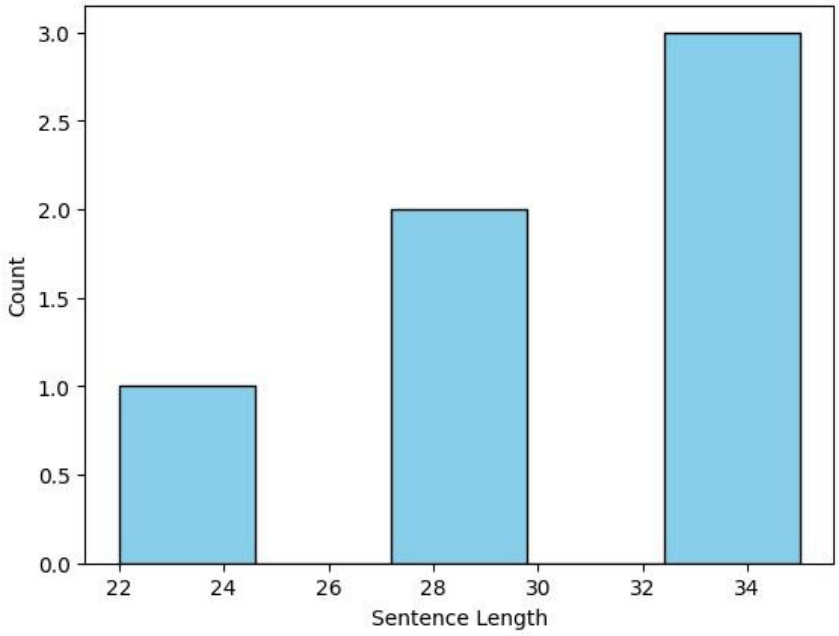
Dataset Link:

https://www.kaggle.com/datasets/grafstor/simple-dialogs-forchatbot

Data Visualization:

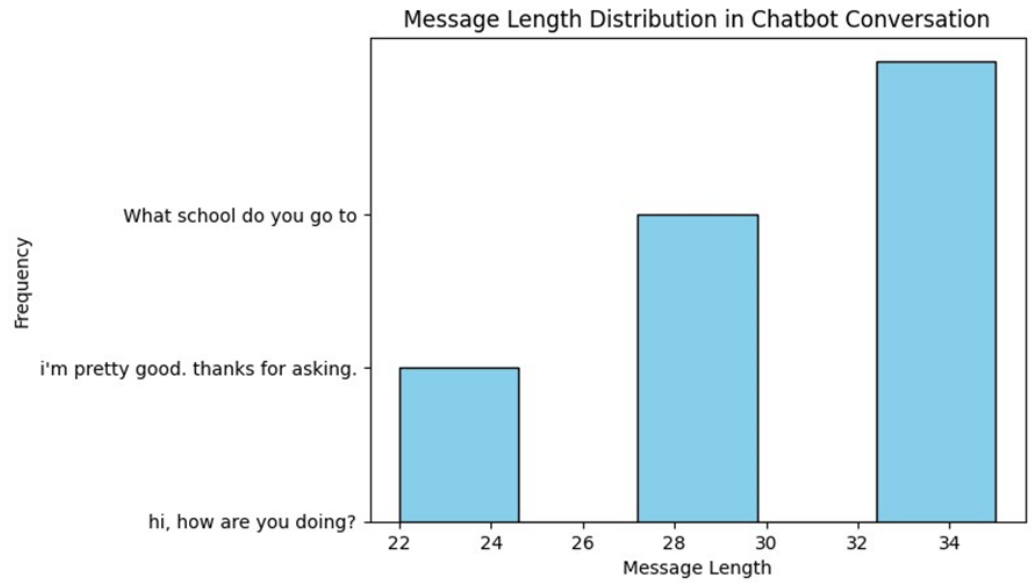
You can create visualizations to understand the distribution of data. For instance, you can visualize the sentence length distribution:

** Using Matplotlib



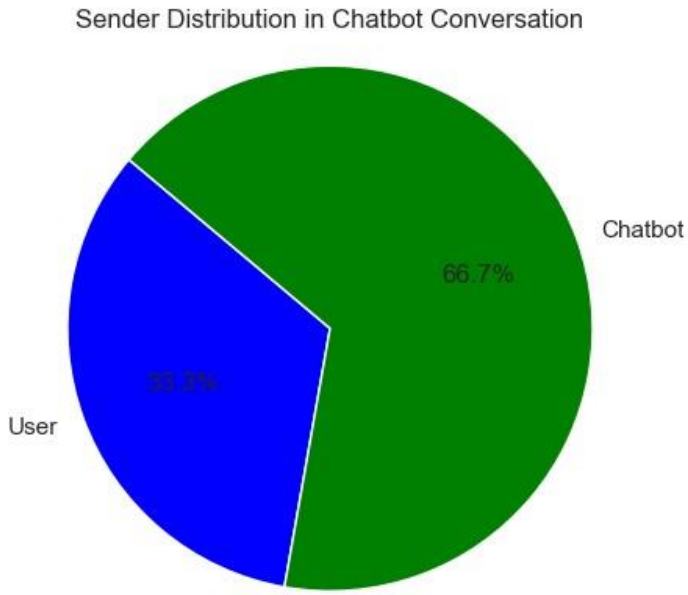

```
# Customize the plot
plt.yticks(range(len(senders)), messages)
plt.xlabel("Message Length")
plt.ylabel("Frequency")
plt.title("Message Length Distribution in Chatbot Conversation")

# Show the plot
plt.show()
```



Pie chat with Matplotlib:

```
# Create a pie chart
labels = 'User', 'Chatbot'
sizes = [user_messages, chatbot_messages]
colors = ['blue', 'green']
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title("Sender Distribution in Chatbot Conversation")
plt.show()
```



building testing and training model

```
class ChatBotTrainer(tf.keras.models.Model):    def
__init__(self,encoder,decoder,*args,**kwargs):
      super().__init__(*args,**kwargs)
self.encoder=encoder
self.decoder=decoder


   def loss_fn(self,y_true,y_pred):
loss=self.loss(y_true,y_pred)
mask=tf.math.logical_not(tf.math.equal(y_true,0))
mask=tf.cast(mask,dtype=loss.dtype)        loss*=mask
return tf.reduce_mean(loss)

      def
accuracy_fn(self,y_true,y_pred):

      pred_values = tf.cast(tf.argmax(y_pred, axis=-1), dtype='int64')        correct =
tf.cast(tf.equal(y_true, pred_values), dtype='float64')        mask =
tf.cast(tf.greater(y_true, 0), dtype='float64')        n_correct =
tf.keras.backend.sum(mask * correct)        n_total = tf.keras.backend.sum(mask)
return n_correct / n_total


   def call(self,inputs):

      encoder_inputs,decoder_inputs=inputs
encoder_states=self.encoder(encoder_inputs)        return
self.decoder(decoder_inputs,encoder_states)


   def train_step(self,batch):

      encoder_inputs,decoder_inputs,y=batch
with tf.GradientTape() as tape:

         encoder_states=self.encoder(encoder_inputs,training=True)
y_pred=self.decoder(decoder_inputs,encoder_states,training=True)
loss=self.loss_fn(y,y_pred)

         acc=self.accuracy_fn(y,y_pred)



variables=self.encoder.trainable_variables+self.decoder.trainable_variables
grads=tape.gradient(loss,variables)
self.optimizer.apply_gradients(zip(grads,variables))
metrics={'loss':loss,'accuracy':acc}        return metrics

      def
test_step(self,batch):
```

```
        encoder_inputs,decoder_inputs,y=batch

        encoder_states=self.encoder(encoder_inputs,training=True)
y_pred=self.decoder(decoder_inputs,encoder_states,training=True)
loss=self.loss_fn(y,y_pred)        acc=self.accuracy_fn(y,y_pred)
metrics={'loss':loss,'accuracy':acc}        return metrics
model=ChatBotTrainer(encoder,decoder,name='chatbot_trainer')
model.compile(

    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
weighted_metrics=['loss','accuracy']

)

model(_[:2])
```

Evaluating a chatbot typically involves assessing its performance in terms of response quality, correctness, and user satisfaction.

Code:

```
history=model.fit(
train_data,
epochs=100,
validation_data=val_
data,     callbacks=[

        tf.keras.callbacks.TensorBoard(log_dir='logs'),
tf.keras.callbacks.ModelCheckpoint('ckpt',verbose=1,save_best_only=True)     ] )


model.load_weights('ckpt') model.save('models',save_format='tf')

for idx,i in enumerate(model.layers):

    print('Encoder layers:' if idx==0 else 'Decoder layers: ')     for j in
i.layers:
        print(j)     print('----------------------')
```

Encoder layers:

<keras.layers.core.embedding.Embedding object at 0x782084b9d190>

<keras.layers.normalization.layer_normalization.LayerNormalization object at

0x7820e56f1b90>

<keras.layers.rnn.lstm.LSTM object at
0x7820841bd650> ---------- Decoder layers:

<keras.layers.core.embedding.Embedding object at 0x78207c258590>

<keras.layers.normalization.layer_normalization.LayerNormalization object at

0x78207c78bd10>

<keras.layers.rnn.lstm.LSTM object at 0x78207c258a10>

<keras.layers.core.dense.Dense object at 0x78207c2636d0> ----------

Inference Model

```
class ChatBot(tf.keras.models.Model):    def
__init__(self,base_encoder,base_decoder,*args,**kwargs):

      super().__init__(*args,**kwargs)


self.encoder,self.decoder=self.build_inference_model(base_encoder,base_decoder)


   def build_inference_model(self,base_encoder,base_decoder):

      encoder_inputs=tf.keras.Input(shape=(None,))
x=base_encoder.layers[0](encoder_inputs)       x=base_encoder.layers[1](x)
x,encoder_state_h,encoder_state_c=base_encoder.layers[2](x)


encoder=tf.keras.models.Model(inputs=encoder_inputs,outputs=[encoder_state_h,enco
der_state_c],name='chatbot_encoder')


      decoder_input_state_h=tf.keras.Input(shape=(lstm_cells,))
decoder_input_state_c=tf.keras.Input(shape=(lstm_cells,))
decoder_inputs=tf.keras.Input(shape=(None,))
x=base_decoder.layers[0](decoder_inputs)       x=base_encoder.layers[1](x)


x,decoder_state_h,decoder_state_c=base_decoder.layers[2](x,initial_state=[decoder

_input_state_h,decoder_input_state_c])       decoder_outputs=base_decoder.layers[-1](x)
decoder=tf.keras.models.Model(
inputs=[decoder_inputs,[decoder_input_state_h,decoder_input_state_c]],


outputs=[decoder_outputs,[decoder_state_h,decoder_state_c]],name='chatbot_decoder
'
      )
      return encoder,decoder


   def summary(self):

      self.encoder.summary()
self.decoder.summary()
```

```
  def softmax(self,z):
    return np.exp(z)/sum(np.exp(z))

  def sample(self,conditional_probability,temperature=0.5):
    conditional_probability =
np.asarray(conditional_probability).astype("float64")       conditional_probability =
np.log(conditional_probability) / temperature       reweighted_conditional_probability =
self.softmax(conditional_probability)       probas = np.random.multinomial(1,
reweighted_conditional_probability, 1)       return np.argmax(probas)

  def preprocess(self,text):       text=clean_text(text)
seq=np.zeros((1,max_sequence_length),dtype=np.int32)
for i,word in enumerate(text.split()):
       seq[:,i]=sequences2ids(word).numpy()[0]
return seq

    def
postprocess(self,text):
     text=re.sub(' - ','-',text.lower())
text=re.sub(' [.] ','. ',text)       text=re.sub(' [1]
','1',text)       text=re.sub(' [2] ','2',text)
text=re.sub(' [3] ','3',text)       text=re.sub(' [4]
','4',text)       text=re.sub(' [5] ','5',text)
text=re.sub(' [6] ','6',text)       text=re.sub(' [7]
','7',text)       text=re.sub(' [8] ','8',text)
text=re.sub(' [9] ','9',text)       text=re.sub(' [0]
','0',text)       text=re.sub(' [,] ',', ',text)
text=re.sub(' [?] ','? ',text)       text=re.sub(' [!] ','!
',text)       text=re.sub(' [$] ','$ ',text)
text=re.sub(' [&] ','& ',text)       text=re.sub(' [/]
','/ ',text)       text=re.sub(' [:] ',': ',text)
text=re.sub(' [;] ','; ',text)       text=re.sub(' [*] ','*
',text)       text=re.sub(' [\'] ','\'',text)
text=re.sub(' [\"] ','\"',text)       return text

  def call(self,text,config=None):
input_seq=self.preprocess(text)
states=self.encoder(input_seq,training=False)
target_seq=np.zeros((1,1))
target_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]
stop_condition=False       decoded=[]       while not
stop_condition:
```

```
        decoder_outputs,new_states=self.decoder([target_seq,states],training=False) #
index=tf.argmax(decoder_outputs[:,-1,:],axis=-1).numpy().item()
index=self.sample(decoder_outputs[0,0,:]).item()          word=ids2sequences([index])
if word=='<end> ' or len(decoded)>=max_sequence_length:

            stop_condition=True
else:

            decoded.append(index)
target_seq=np.zeros((1,1))
target_seq[:,:]=index              states=new_states
return self.postprocess(ids2sequences(decoded))


chatbot=ChatBot(model.encoder,model.decoder,name='chatbot') chatbot.summary()
```

Integration of chatbot with website:

Create a website to integrate the chatbot:

To integrate the chatbot creation of website is important

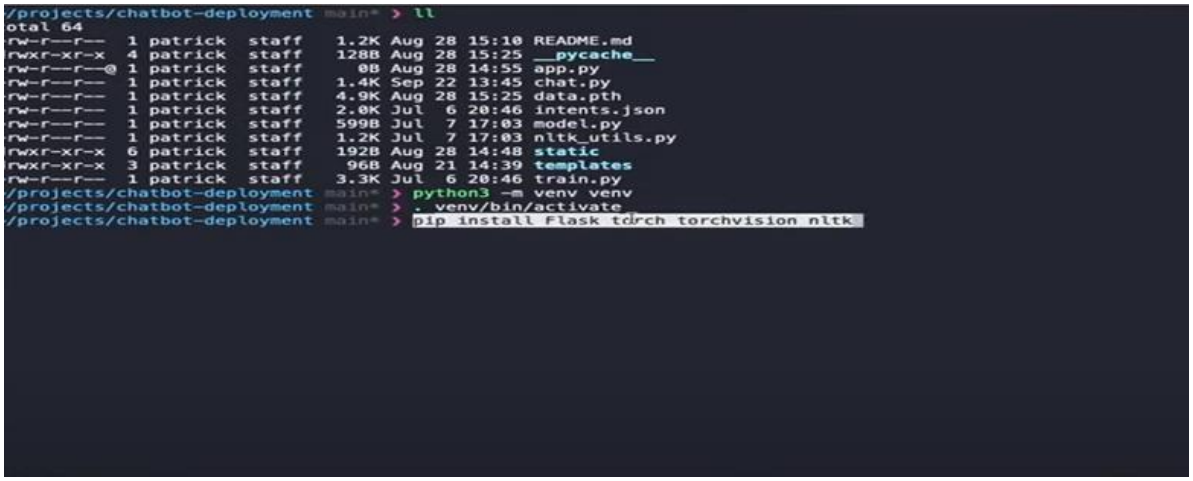
HTML and javascript is used to create a website of our own.

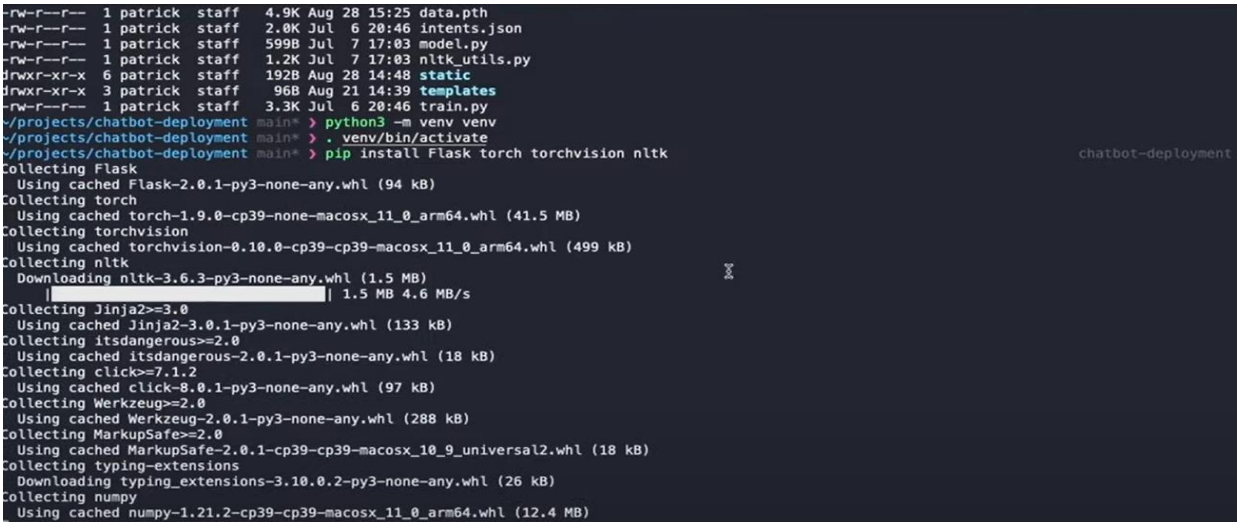
To add the chatbot use our custom design for user friendly UI for great experience.

Set up Flask:

Install Flask using pip. command: pip install flask

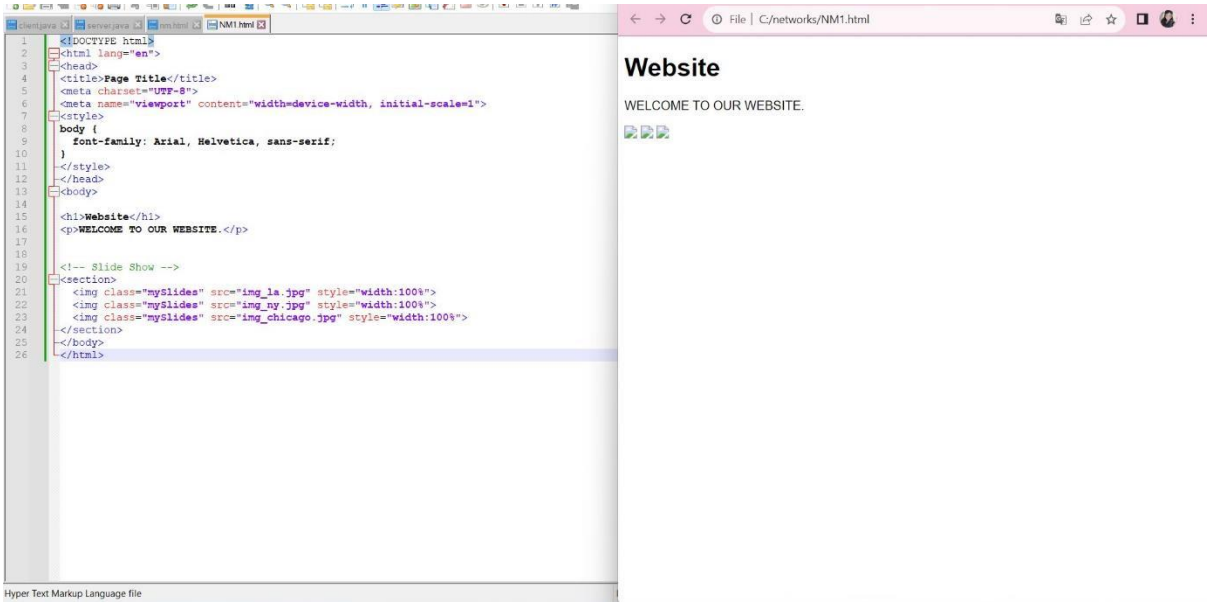
A Web Application Framework or a simply a Web Framework represents a collection of libraries and modules that enable web application developers to write applications without worrying about low-level details such as protocol, thread management, and so on.





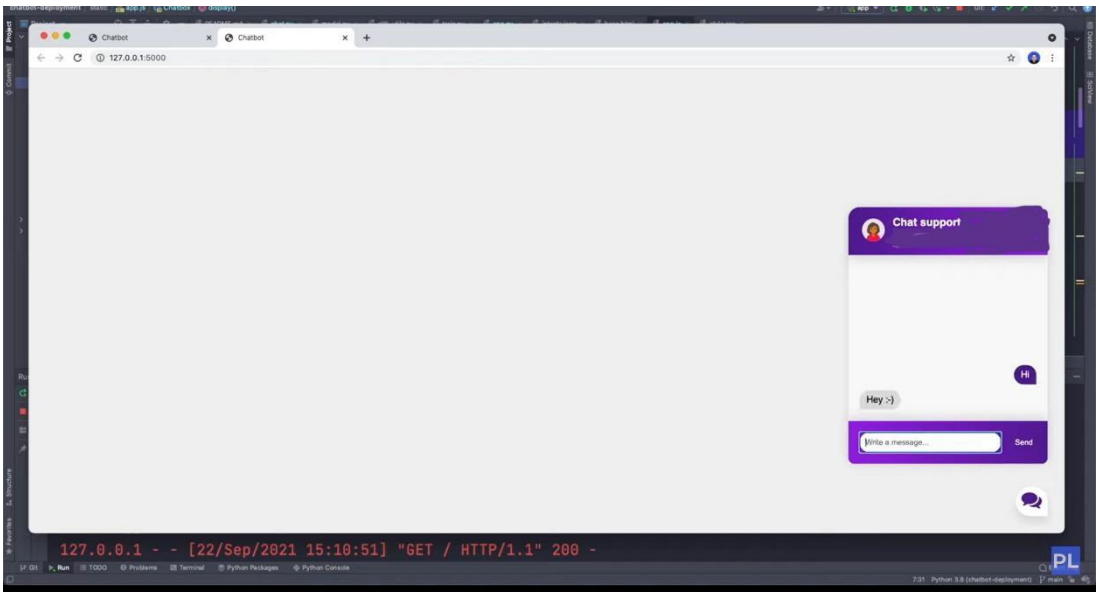
create web back ground for using HTML:

Design the web interface where users will interact with the chatbot. Create HTML templates that include input fields for users to type messages and a chat area to display the conversation.



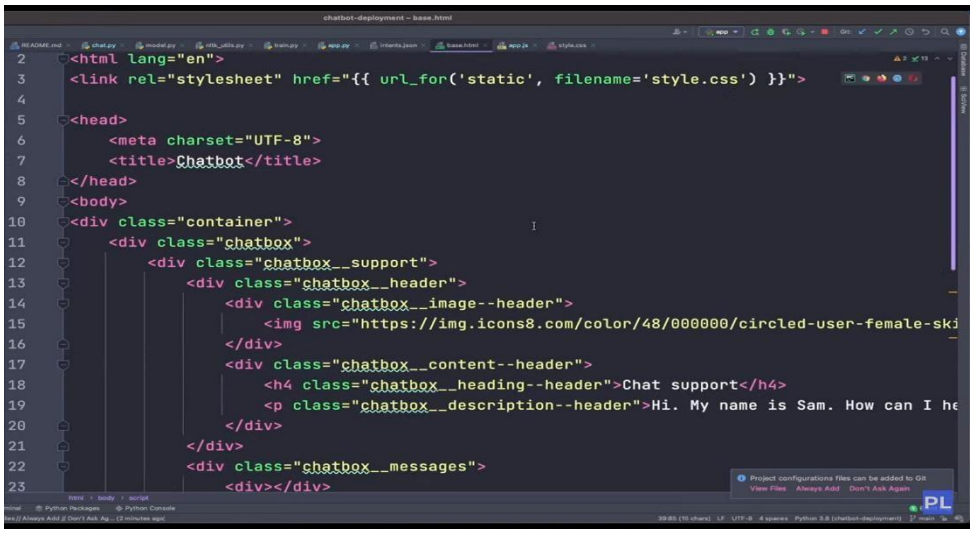
Integration of chatbot in website using API:

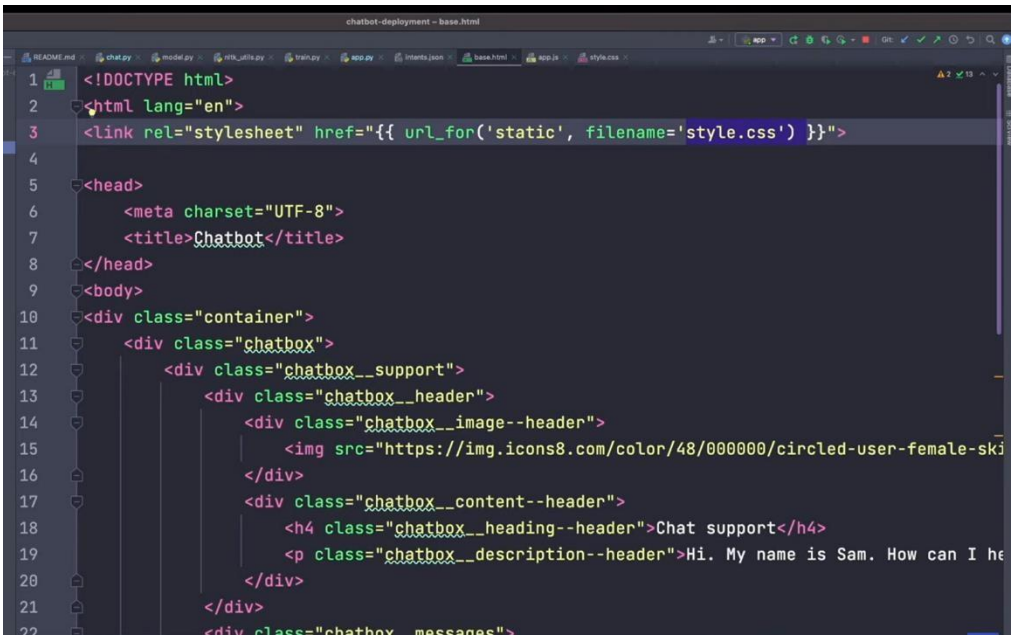
This might involve using an external chatbot service, such as Dialogflow, or a custom chatbot you've developed. Just need an endpoint or function that accepts user messages and returns chatbot responses.



Update Chat Interface:

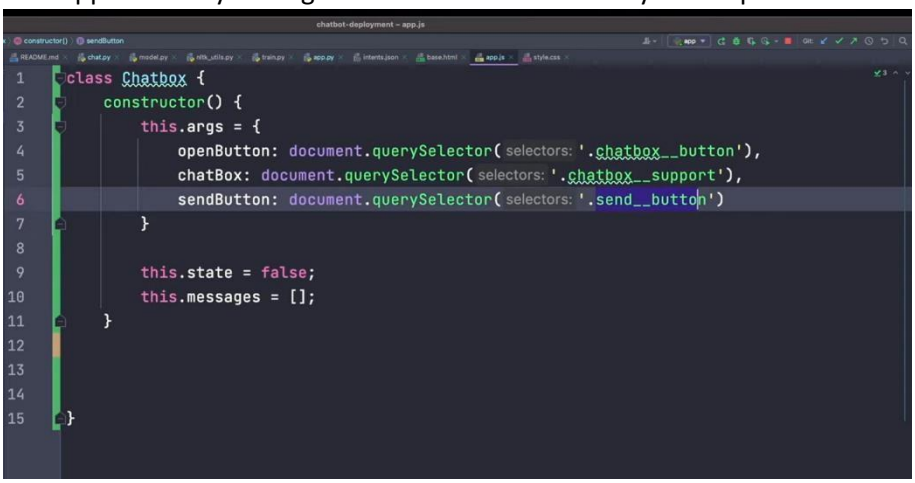
In HTML template, use JavaScript to handle user input and chatbot responses. And make AJAX requests to the /chatbot route to send user messages and display chatbot responses in the chat area.





Run the Flask APP:

Start your Flask application by adding this code at the bottom of your script:



Test and Deploy:

Test chatbot web app locally to ensure it's working as expected. Once it satisfied with the functionality, then deploy it to a web server or a cloud platform like Heroku, AWS, or GCP.

Code for integration of chatbot in web:

web.html

```
<!DOCTYPE html>

<html>

<head>

   <meta charset="utf-8">

   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">

   <link rel="stylesheet" href="/static/style.css">

   <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></s cript>

</head>

<body>

   <h1 class="centered-heading"><span>CHATBOT USING PYTHON AND
FLASK</span></h1>

   <div class="container">

      <div class="row">
```

```
            <div class="full-space-element">
                <div id="chatbox">
                    <p class="botText"><span>Hi! I'm Chatbot</span></p>
                </div>
            </div>
        </div>
        <div class="input-container">
            <div id="userInput">
                <input id="textInput" type="text" name="msg" placeholder="Type Your
  Message Here">
                <input id="buttonInput" type="submit" value="Send">           </div>
        </div>
    </div>
  <script>
        function getResponse() {
            let userText = $("#textInput").val();
            let userHtml = '<p class="userText"><span>' + userText + '</span></p>';
                    $("#textInput").val("");          $("#chatbox").append(userHtml);
  document.getElementById('chatbox').scrollIntoView({ block:
'end',              b                    ehavior: 'smooth' });
            $.get("/get", { msg: userText }).done(function (data) {              var
  botHtml = '<p class="botText"><span>' + data + '</span></p>';
                $("#chatbox").append(botHtml);
  document.getElementById('chatbox').scrollIntoView({ block:
'end',                                  behavior: 'smooth' });
            });
        }
        $("#textInput").keypress(function (e) {
            //if enter key is pressed
  if (e.which == 13) {
  getResponse();
```

```
        }
    });
    $("#buttonInput").click(function () {           getResponse();
    });
  </script>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></s cript>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.mi
n.js"></script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js">
</script>
  </div>
</body>
</html>
```

webstyl.css:

```
.centered-heading {
text-align: center;    font-
weight: bold;    font-
family: monospace;
margin-top: 40px;
}
.centered-heading span {    background-
color: yellow;
}

.centered-heading::selection {    background-
color: yellow;
  /* Change the background color of selected text */    color:
black;
```

```
   /* Change the text color of selected text */
}
.full-space-element {
width: 100%;
height: 100%;
background-color:
white;
}
.container {
   max-width: 400px;     margin: 20px auto;
border: 1px solid #ccc;     background-
color: #fff;     border-radius: 5px;     box-
shadow: 0 0 10px rgba(0, 0, 0, 0.1);
} .row {
padding:
20px;
height: 300px;
overflow-y:
scroll;
} .chat-message
{     margin-
bottom: 10px;
padding: 10px;
border-radius:
5px;
} .user-message {
background-color:
#e0e0e0;     text-align:
right; }
```

```
.input-container {
padding: 10px;
display: flex;
align-items: center;
} .userText {
color: white;
font-family:
monospace;  font-
size: 17px;  text-
align: right;  line-
height: 30px;
} .userText span {
background-color:
#EF5350;  padding:
5px;  border-radius:
5px;
} .botText {  color:
white;  font-
family: monospace;
font-size: 17px;
text-align: left;
line-height: 30px; }
.botText span {
background-color:
blue;  padding:
5px;  border-
radius: 5px;
} #textInput {  flex:
1;  padding: 4px;
border: 1px solid
```

```
black;     border-
radius: 5px;
outline: black;
margin-right: 10px;
width: 225px; }
#buttonInput {
background-color:
green;     color: #fff;
border: none;
border-radius: 5px;
padding: 5px 20px;
cursor: pointer;
margin-left: 10px;
}
```

web1.py:

```
from flask import Flask, render_template, request import
pandas as pd
app = Flask(__name__,template_folder='templates',static_folder='static') csv_file =
'sampled.csv'


#create chatbot def
chatbot(input,csv_fil
e):
   df = pd.read_csv(csv_file,delimiter='\t')
user =0     chat = 1
   user_input = input.lower()     if(user_input==df.columns[user]):
      return df.columns[chat]     if(user_input == "hi"
or user_input=="hello"):
      return "hi,good morning"
```

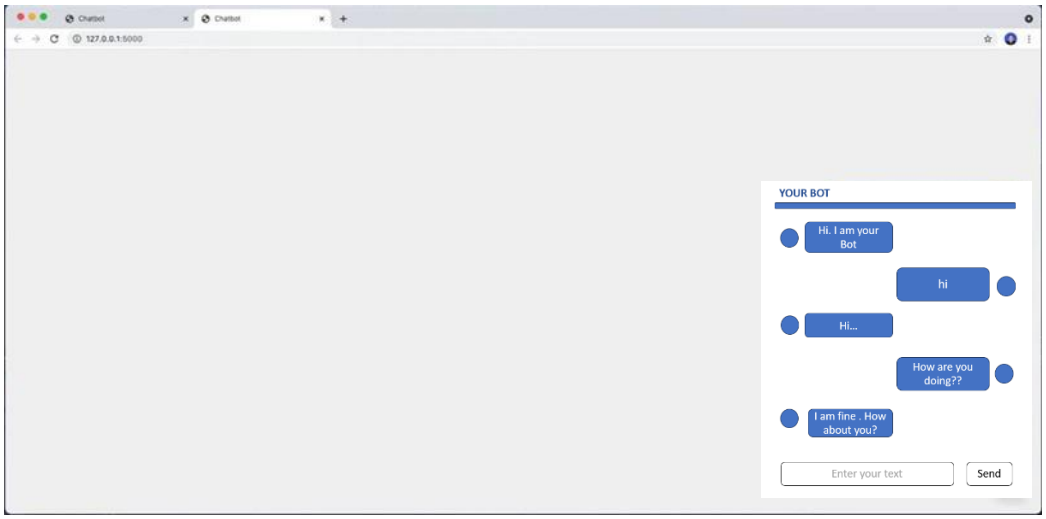
```
    if(user_input=="bye" or user_input == "thanks" or user_input == "thank you"):
        return "Welcome"
    user_response = df[df.iloc[:,user]==input].iloc[:,chat].values
if(len(user_response)>0):        return user_response[0]


e
l
s
e
:
        return "No data available"

#define app routes
@app.ro
ute("/")
def
index():
    return render_template("index.html")

@app.route("/get")
#function for the bot
response def
get_bot_response():
    userText = request.args.get('msg')
return chatbot(userText.strip(),csv_file) if
__name__ == "__main__":
app.run(debug=True)
```

Output:



Chatbot Capabilities and Features:

- Unsupervised AI Learning Natural Language Processing /Understanding. Unsupervised AI learning is at the foundation of the exceptional AI chatbot.
- Omnichannel Messaging.
- Conversational AI.
- A No-Code Visual Flow Builder.
- Live Chat Handover & Intelligence.

- Sentiment Analysis.

Chatbot benefits for businesses:

- Improve service with every interaction.
- Collect customer feedback.
- Reduce customer requests.
- Detect customer intent for added context.
- Boost customer engagement.
- Streamline service with routing and triage.
- Boost sales.
- Increase lead generation.

conclusion:

A chatbot is one of the simple ways to transport data from a computer without having to think for proper keywords to look up in a search or browse several web pages to collect information; users can easily type their query in natural language and retrieve information. They help us by providing entertainment, saving time and answering the questions that are hard to find. The Chatbot must be simple and conversational. Since there are many designs and approaches for creating a chatbot, it can be at odds with commercial considerations. Creating a chatbot is a multi-faceted endeavor that starts with data preparation and analysis. By importing a relevant dataset, cleaning the data, and analyzing it, we set a solid foundation for our chatbot project. The subsequent steps of training, testing, and deploying your chatbot are equally important. Building a chatbot is a dynamic process that requires ongoing refinement and adaptation to meet our users' needs. In this project, we understood how Chatbots are developed and the applications of Chatbots in various fields.