CREATE A CHATBOT IN PYTHON

NAME: ARUN KRISHNA J
REG NO: 61772221T302

NM USER ID: aut2221t302

PHASE 4 Submission Document

TABLE OF CONTENTS:

Introduction:

Chatbots have become an essential component of modern communication, improving user experiences on websites, social networking platforms, and customer service systems. To construct an effective chatbot, we need to start with high-quality data and a thorough grasp of the dataset. This explains the critical procedures for importing, cleaning, and analysing data as the foundation for the chatbot project.

Problem Statement:

Customers expect excellent service when using your app or website. They may lose interest in the app if they are unable to find an answer to a query they have. To avoid losing consumers and harming your bottom line, you must provide the best service possible while establishing a website or application.

Creating a Chatbot in Python: Data Preparation and Analysis

NLP(Natural Language Process):

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.

Import necessary libraries

```
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
```

Install the Transformers Library:

The Transformers library provides access to a wide range of pre-trained language models, including GPT-3.

Command: pip install transformers

Then import the Transormer library:

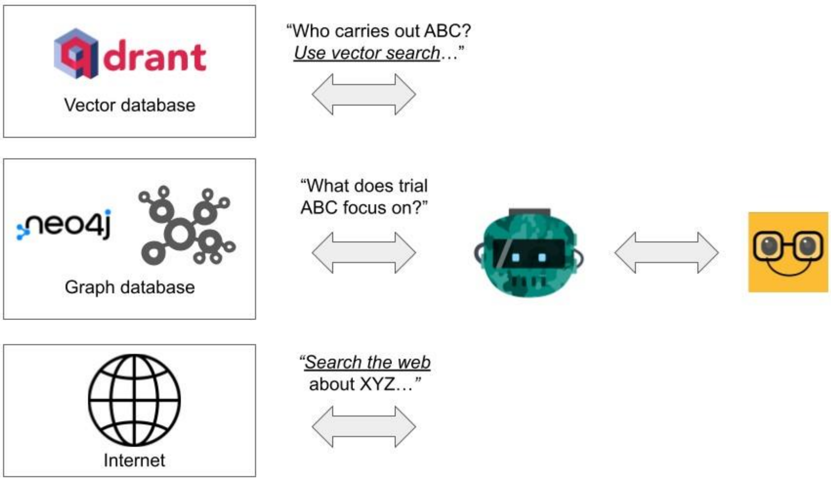
```
from transformers import GPT2LMHeadModel, GPT2Tokenizer, AdamW
```

Install Flask for Web App Development:

Flask is a lightweight web framework for building web applications.

Command: pip install Flask

Import the Dataset:



To build a chatbot, we first need a dataset.We can either collect conversational data or obtain a dataset from sources like Twitter, Reddit, or customer support interactions. Assuming that we have a CSV file with our dataset, we can import it using Python and Pandas:

```
df=pd.read_csv('/kaggle/input/simple-dialogs-for-
chatbot/dialogs.txt',sep='\t',names=['question','answer'])
print(f'Dataframe size: {len(df)}')
df.head()
```

```
Dataframe size: 3725

                              question  \
0               hi, how are you doing?
1        i'm fine. how about yourself?
2  i'm pretty good. thanks for asking.
3    no problem. so how have you been?
4     i've been great. what about you?

                                     answer
0             i'm fine. how about yourself?
1       i'm pretty good. thanks for asking.
2         no problem. so how have you been?
3          i've been great. what about you?
4  i've been good. i'm in school right now.
```

Data Cleaning:

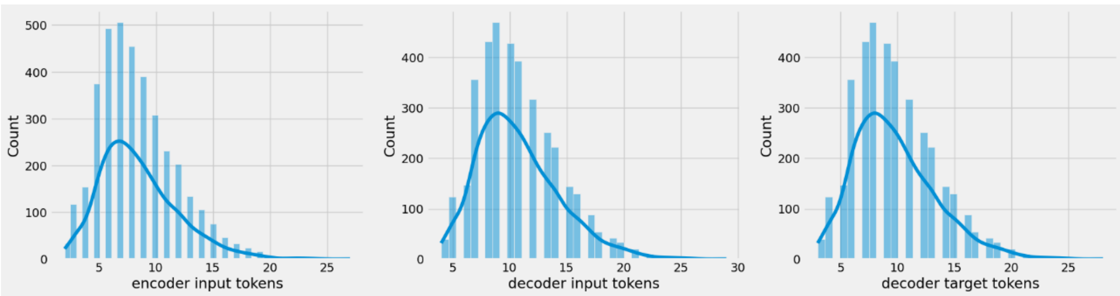

```
def clean_text(text):
    text=re.sub('-',' ',text.lower())
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
    text=re.sub('[3]',' 3 ',text)
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    text=re.sub('[6]',' 6 ',text)
    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)
    text=re.sub('[\']',' \' ',text)
    text=re.sub('[\"]',' \" ',text)
    text=re.sub('\t',' ',text)
    return text
```

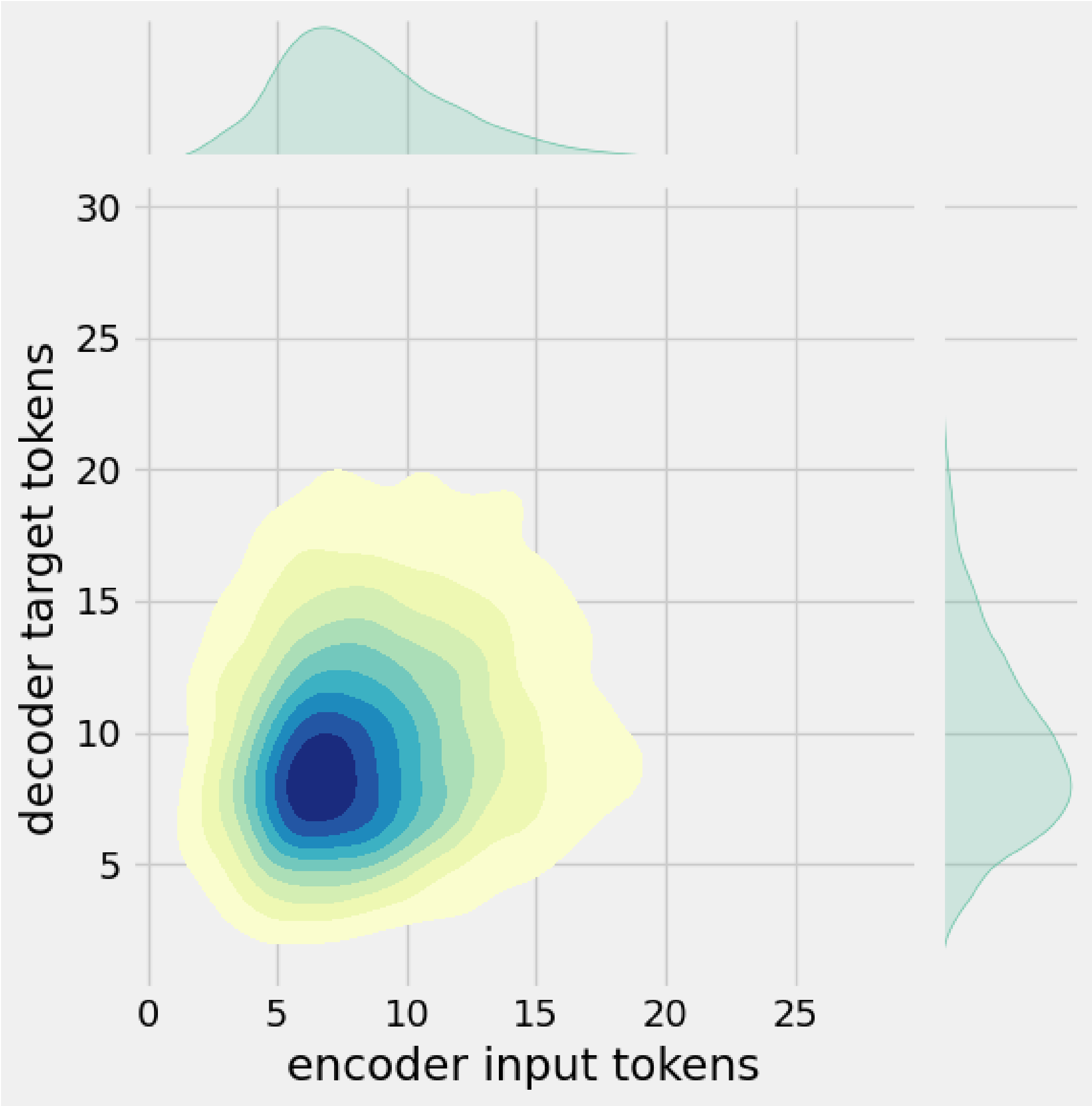
```
df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'

df.head(10)
```

```
                                         question  \
0                        hi, how are you doing?
1                 i'm fine. how about yourself?
2           i'm pretty good. thanks for asking.
3             no problem. so how have you been?
4              i've been great. what about you?
5  i've been good. i'm in school right now.
6                      what school do you go to?
7                                   i go to pcc.
8                          do you like it there?
9          it's okay. it's a really big campus.

                                           answer  \
0                 i'm fine. how about yourself?
1           i'm pretty good. thanks for asking.
2             no problem. so how have you been?
3              i've been great. what about you?
4  i've been good. i'm in school right now.
5                      what school do you go to?
6                                   i go to pcc.
7                          do you like it there?
8          it's okay. it's a really big campus.
9                        good luck with school.

                                           encoder_inputs  \
0                           hi ,  how are you doing ?
1                   i ' m fine .  how about yourself ?
2             i ' m pretty good .  thanks for asking .
3                 no problem .  so how have you been ?
4                i ' ve been great .  what about you ?
5  i ' ve been good .  i ' m in school right now .
6                           what school do you go to ?
7                                        i go to pcc .
8                                do you like it there ?
9      it ' s okay .  it ' s a really big campus .

                                          decoder_targets  \
0          i ' m fine .  how about yourself ?  <end>
1    i ' m pretty good .  thanks for asking .  <end>
2        no problem .  so how have you been ?  <end>
3       i ' ve been great .  what about you ?  <end>
4  i ' ve been good .  i ' m in school right now ...
5                  what school do you go to ?  <end>
```

```
6                                  i go to pcc .  <end>
7                         do you like it there ?  <end>
8  it ' s okay .  it ' s a really big campus .  <...
9                        good luck with school .  <end>

                                            decoder_inputs
0  <start> i ' m fine .  how about yourself ?  <end>
1  <start> i ' m pretty good .  thanks for asking...
2  <start> no problem .  so how have you been ?  ...
3  <start> i ' ve been great .  what about you ? ...
4  <start> i ' ve been good .  i ' m in school ri...
5          <start> what school do you go to ?  <end>
6                          <start> i go to pcc .  <end>
7                <start> do you like it there ?  <end>
8  <start> it ' s okay .  it ' s a really big cam...
9               <start> good luck with school .  <end>
```

```python
df['encoder input tokens']=df['encoder_inputs'].apply(lambda x:len(x.split()))
df['decoder input tokens']=df['decoder_inputs'].apply(lambda x:len(x.split()))
df['decoder target tokens']=df['decoder_targets'].apply(lambda x:len(x.split()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=3,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['encoder input tokens'],data=df,kde=True,ax=ax[0])
sns.histplot(x=df['decoder input tokens'],data=df,kde=True,ax=ax[1])
sns.histplot(x=df['decoder target tokens'],data=df,kde=True,ax=ax[2])
sns.jointplot(x='encoder input tokens',y='decoder target
tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')
plt.show()
```



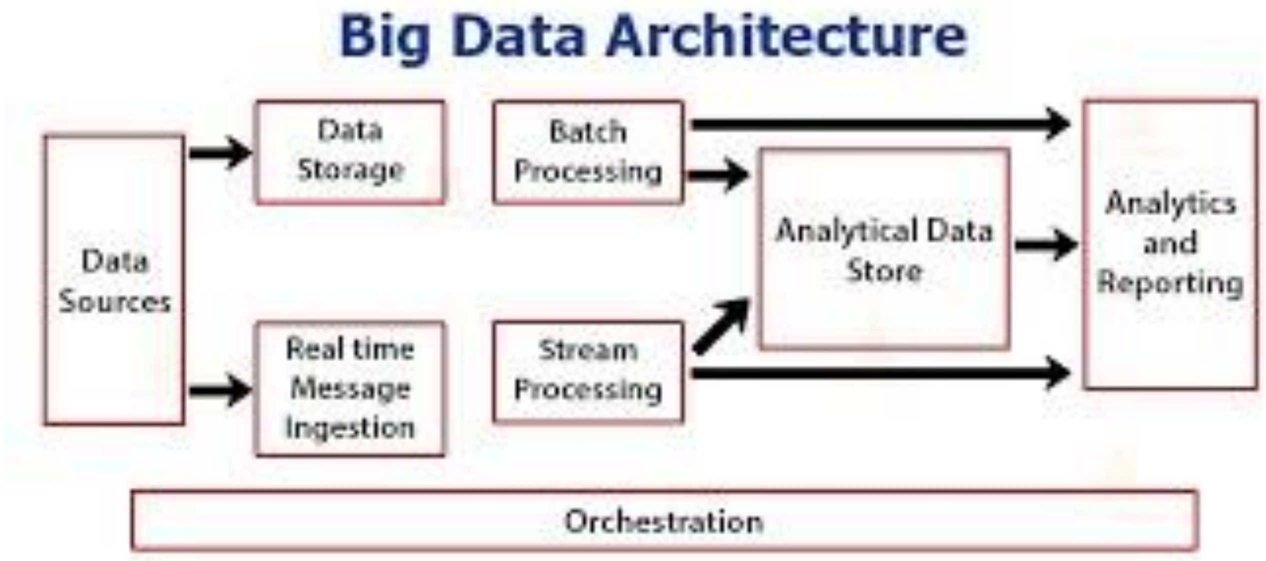



```
print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()==df['encoder input tokens']]['encoder_inputs'].values.tolist())}")
print(f"Max encoder input length: {df['encoder input tokens'].max()}")
print(f"Max decoder input length: {df['decoder input tokens'].max()}")
print(f"Max decoder target length: {df['decoder target tokens'].max()}")

df.drop(columns=['question','answer','encoder input tokens','decoder input tokens','decoder target tokens'],axis=1,inplace=True)
params={
    "vocab_size":2500,
    "max_sequence_length":30,
    "learning_rate":0.008,
    "batch_size":149,
    "lstm_cells":256,
    "embedding_dim":256,
    "buffer_size":10000
}
learning_rate=params['learning_rate']
batch_size=params['batch_size']
embedding_dim=params['embedding_dim']
lstm_cells=params['lstm_cells']
```

```
vocab_size=params['vocab_size']
buffer_size=params['buffer_size']
max_sequence_length=params['max_sequence_length']
df.head(10)
```

```
After preprocessing: for example ,  if your birth date is january  1  2  ,   1  9
8  7  ,  write  0  1  /  1  2  /  8  7  .
Max encoder input length: 27
Max decoder input length: 29
Max decoder target length: 28

                                     encoder_inputs  \
0                         hi ,  how are you doing ?
1                i ' m fine .  how about yourself ?
2          i ' m pretty good .  thanks for asking .
3              no problem .  so how have you been ?
4            i ' ve been great .  what about you ?
5  i ' ve been good .  i ' m in school right now .
6                        what school do you go to ?
7                                     i go to pcc .
8                            do you like it there ?
9      it ' s okay .  it ' s a really big campus .

                                     decoder_targets  \
0          i ' m fine .  how about yourself ?  <end>
1    i ' m pretty good .  thanks for asking .  <end>
2        no problem .  so how have you been ?  <end>
3       i ' ve been great .  what about you ?  <end>
4  i ' ve been good .  i ' m in school right now ...
5                  what school do you go to ?  <end>
6                               i go to pcc .  <end>
7                      do you like it there ?  <end>
8  it ' s okay .  it ' s a really big campus .  <...
9                     good luck with school .  <end>

                                      decoder_inputs
0  <start> i ' m fine .  how about yourself ?  <end>
1  <start> i ' m pretty good .  thanks for asking...
2  <start> no problem .  so how have you been ?  ...
3  <start> i ' ve been great .  what about you ? ...
4  <start> i ' ve been good .  i ' m in school ri...
5          <start> what school do you go to ?  <end>
6                       <start> i go to pcc .  <end>
7              <start> do you like it there ?  <end>
8  <start> it ' s okay .  it ' s a really big cam...
9             <start> good luck with school .  <end>
```

Data Analysis:



Understanding your dataset through data analysis is a crucial step in creating a chatbot. It provides insights and informs your chatbot's design.

Basic text preprocessing includes:

- Converting the entire text into **uppercase** or **lowercase**, so that the algorithm does not treat the same words in different cases as different
- **Tokenization**: Tokenization is just the process of transforming standard text strings into a list of tokens, or words that we truly desire.
- Sentence tokenizer can be used to find a list of sentences, whereas Word tokenizer can find a list of words in strings.

The NLTK data package includes a pre-trained Punkt tokenizer for English.

- Removing **Noise** i.e everything that isn't in a standard number or letter.
- Removing the **Stop words**. Sometimes, some extremely common words which would appear to be of little value in

helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words

- **Stemming**: The process of reducing inflected (or sometimes derived) words to their stem, base, or root form — typically a written word form — is known as stemming. For example, if we stemmed the phrases "Stems", "Stemming", "Stemmed", and "and Stemtization", the result would be a single word "stem".

- **Lemmatization**: A slight variant of stemming is lemmatization. The major difference between these is, that, stemming can often create non-existent words, whereas lemmas are actual words.

- So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma.

Tokenization

```
vectorize_layer=TextVectorization(
    max_tokens=vocab_size,
    standardize=None,
    output_mode='int',
    output_sequence_length=max_sequence_length
)
vectorize_layer.adapt(df['encoder_inputs']+' '+df['decoder_targets']+' <start>
<end>')
vocab_size=len(vectorize_layer.get_vocabulary())
print(f'Vocab size: {len(vectorize_layer.get_vocabulary())}')
print(f'{vectorize_layer.get_vocabulary()[:12]}')
```

```
Vocab size: 2443
['', '[UNK]', '<end>', '.', '<start>', "'", 'i', '?', 'you', ',', 'the', 'to']
```

```
def sequences2ids(sequence):
    return vectorize_layer(sequence)
```

```python
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)
```

```python
print(f'Encoder input: {x[0][:12]} ...')
print(f'Decoder input: {yd[0][:12]} ...')    # shifted by one time step of the
target as input to 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] ...
```

```python
data=tf.data.Dataset.from_tensor_slices((x,yd,y))
data=data.shuffle(buffer_size)

train_data=data.take(int(.9*len(data)))
train_data=train_data.cache()
train_data=train_data.shuffle(buffer_size)
train_data=train_data.batch(batch_size)
train_data=train_data.prefetch(tf.data.AUTOTUNE)
train_data_iterator=train_data.as_numpy_iterator()

val_data=data.skip(int(.9*len(data))).take(int(.1*len(data)))
val_data=val_data.batch(batch_size)
val_data=val_data.prefetch(tf.data.AUTOTUNE)

_=train_data_iterator.next()
print(f'Number of train batches: {len(train_data)}')
print(f'Number of training data: {len(train_data)*batch_size}')
print(f'Number of validation batches: {len(val_data)}')
print(f'Number of validation data: {len(val_data)*batch_size}')
```

```
print(f'Encoder Input shape (with batches): {_[0].shape}')
print(f'Decoder Input shape (with batches): {_[1].shape}')
print(f'Target Output shape (with batches): {_[2].shape}')
```

```
Number of train batches: 23
Number of training data: 3427
Number of validation batches: 3
Number of validation data: 447
Encoder Input shape (with batches): (149, 30)
Decoder Input shape (with batches): (149, 30)
Target Output shape (with batches): (149, 30)
```

Code and Dataset:

Code:

```
import pandas as pd

# Read the CSV file into a Pandas DataFrame
csv_file = 'sampled.csv'
df = pd.read_csv(csv_file, delimiter='\t')
user =0
chatbot = 1
while True:
    user_input0 = input("Enter: ").strip()
    user_input = user_input0.lower()
    if(df.columns[user] == user_input):
        print("Robo: ",df.columns[chatbot])
        continue
    if user_input.lower() == "bye":
        print("Robo: Welcome")
        break  # Exit the loop if the user enters "bye"


    # Search for the user input in the DataFrame
    response = df[df.iloc[:,user] == user_input].iloc[:,chatbot].values
    if len(response) > 0:
        print("Robo:", response[0])
    else:
        print("Robo: I can't hear properly")
```

Output:

```
Enter:  i'm fine. how about yourself?
Robo: i'm pretty good. thanks for asking.
Enter:  i'm pretty good. thanks for asking.
Robo: no problem. so how have you been?
Enter: ↑↓ for history. Search history with c-↑/c-↓
```

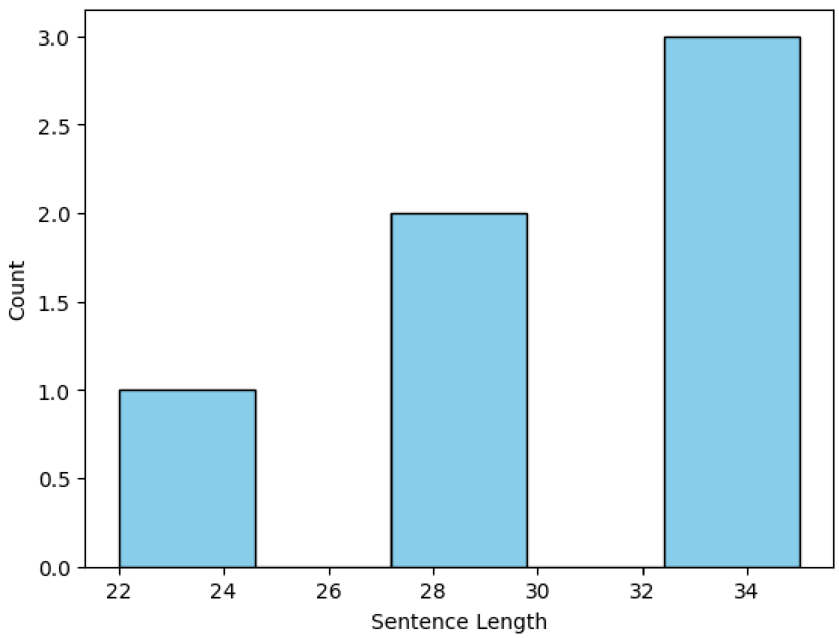
Dataset Link:

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

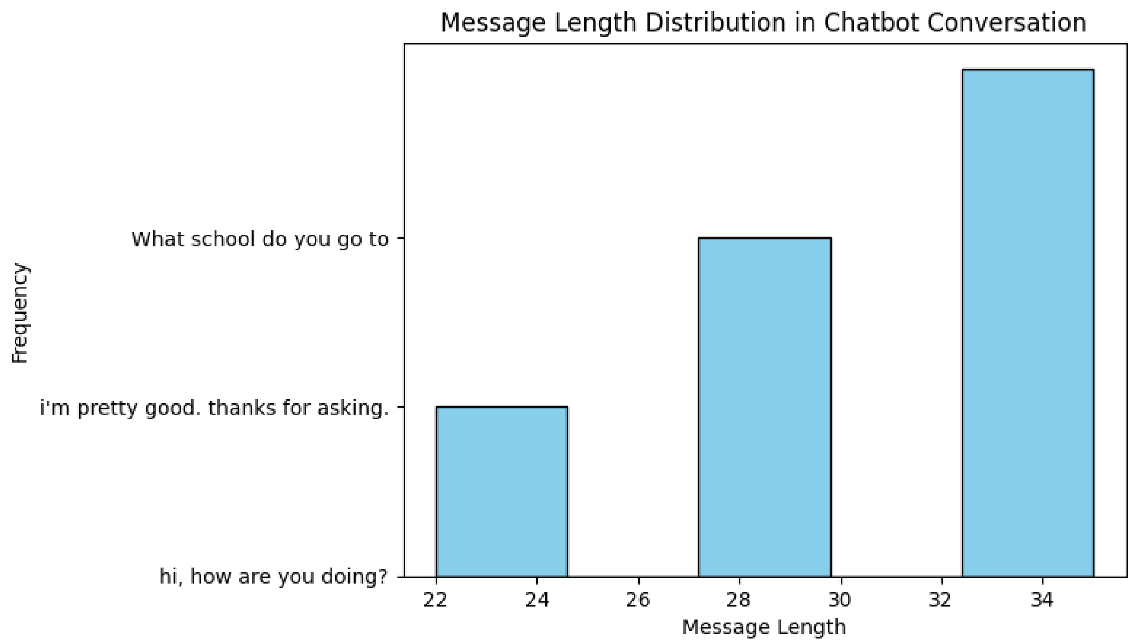
Data Visualization:

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

** Using **Matplotlib**

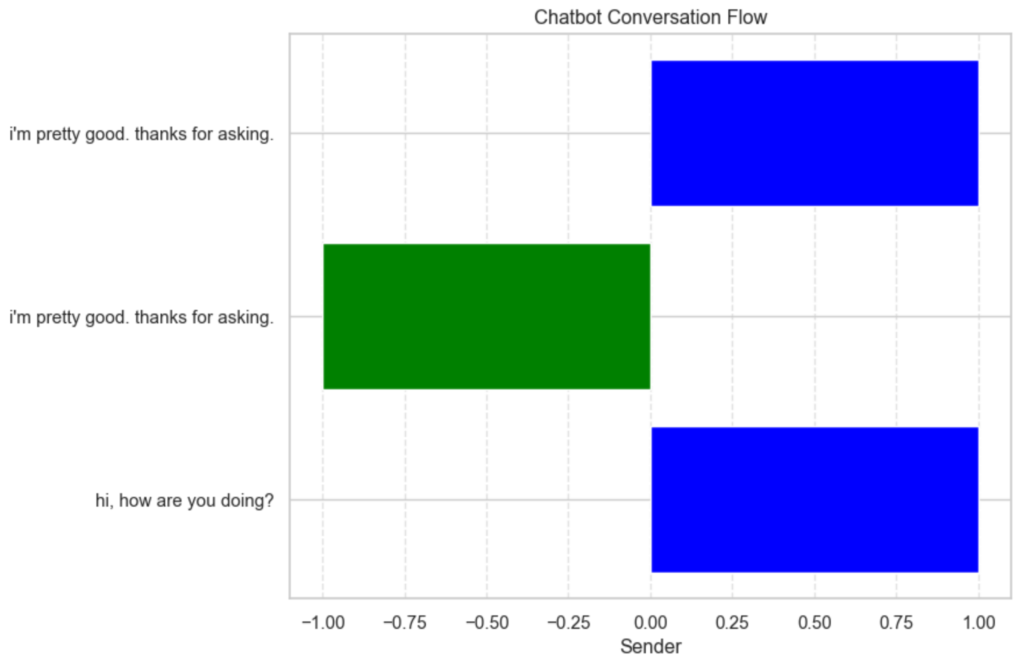


```python
# Customize the plot
plt.yticks(range(len(senders)), messages)
plt.xlabel("Message Length")
plt.ylabel("Frequency")
plt.title("Message Length Distribution in Chatbot Conversation")

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



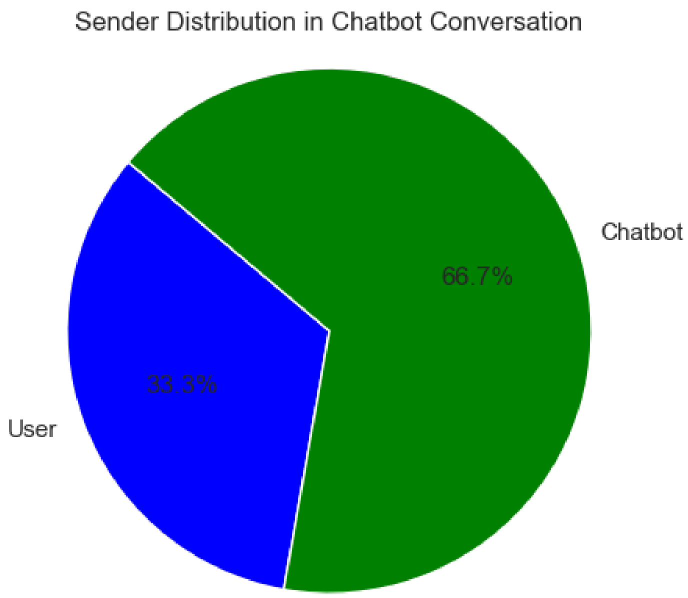
Bar chat with Matplotlib:

```python
# Create a bar chart
plt.figure(figsize=(8, 6))
plt.barh(range(len(senders)), [1 if sender == "User" else -1 for sender in senders], color=['blue' if sender == "User" else 'green' for sender in senders])
plt.yticks(range(len(senders)), messages)
plt.xlabel("Sender")
plt.title("Chatbot Conversation Flow")
plt.grid(axis='x', linestyle='--', alpha=0.6)
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()
```

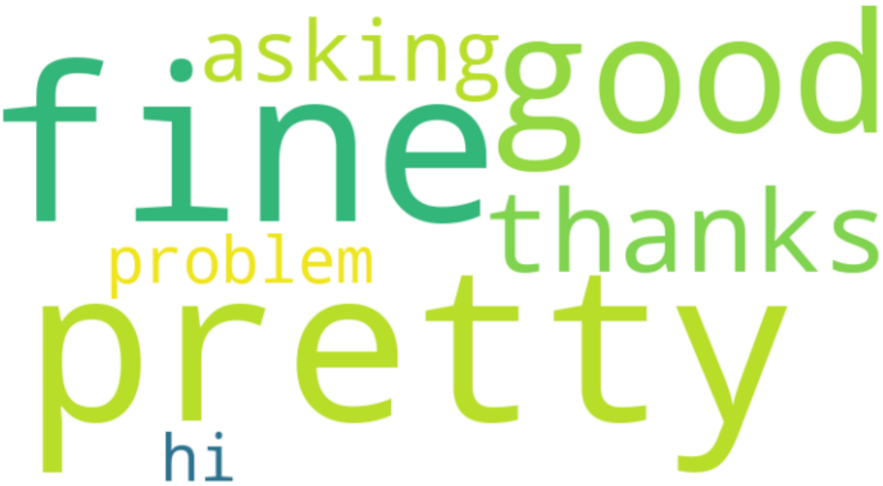


** Using **Word Cloud with WordCloud Library**

from wordcloud import WordCloud

```
# Generate a word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud for Chatbot Conversation")
plt.show()
```

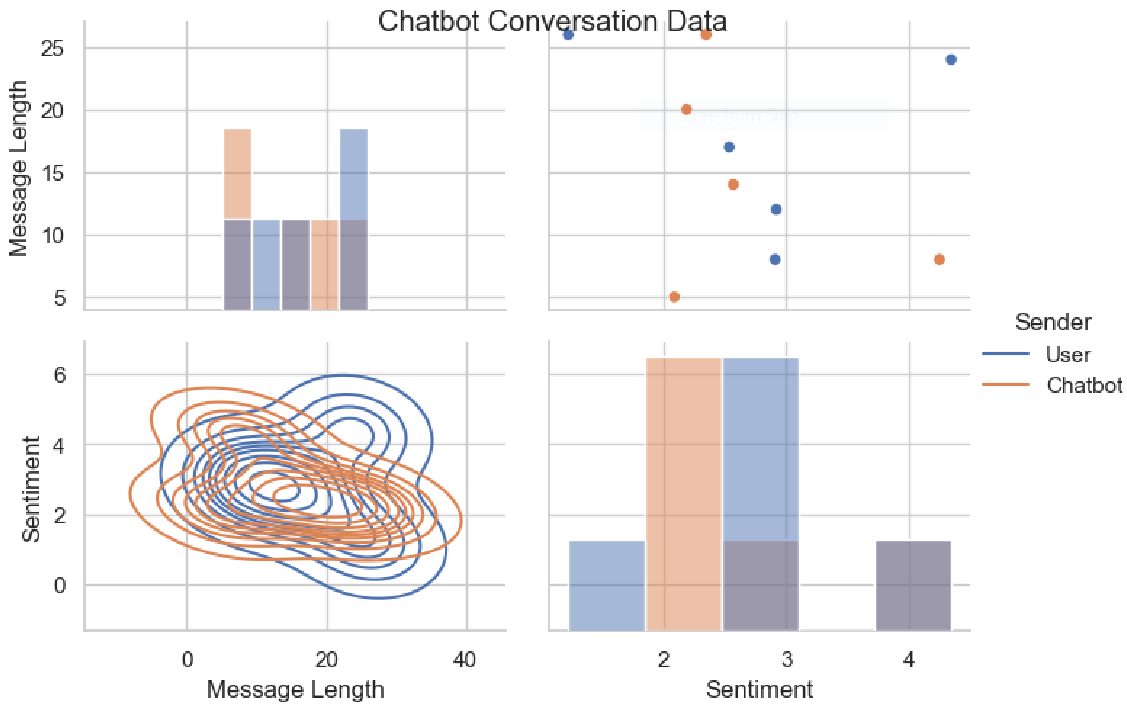
Word Cloud for Chatbot Conversation



** Using **SEABORN**

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
# Create a pairplot with creative data representation
sns.set(style="whitegrid")
g = sns.PairGrid(df, hue="Sender", aspect=1.4)
g.map_upper(sns.scatterplot)
g.map_diag(sns.histplot)
g.map_lower(sns.kdeplot, cmap="Blues")

# Customize the plot
g.add_legend()
g.fig.suptitle("Chatbot Conversation Data")

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



Model Development:

Build Encoder

```
class Encoder(tf.keras.models.Model):
    def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> None:
        super().__init__(*args,**kwargs)
        self.units=units
        self.vocab_size=vocab_size
        self.embedding_dim=embedding_dim
        self.embedding=Embedding(
            vocab_size,
            embedding_dim,
            name='encoder_embedding',
            mask_zero=True,
            embeddings_initializer=tf.keras.initializers.GlorotNormal()
        )
        self.normalize=LayerNormalization()
        self.lstm=LSTM(
            units,
            dropout=.4,
            return_state=True,
            return_sequences=True,
            name='encoder_lstm',
            kernel_initializer=tf.keras.initializers.GlorotNormal()
        )
```

```
    def call(self,encoder_inputs):
        self.inputs=encoder_inputs
        x=self.embedding(encoder_inputs)
        x=self.normalize(x)
        x=Dropout(.4)(x)
        encoder_outputs,encoder_state_h,encoder_state_c=self.lstm(x)
        self.outputs=[encoder_state_h,encoder_state_c]
        return encoder_state_h,encoder_state_c

encoder=Encoder(lstm_cells,embedding_dim,vocab_size,name='encoder')
encoder.call(_[0])
```

```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.16966951, -0.10419625, -0.12700348, ..., -0.12251794,
          0.10568858,  0.14841646],
        [ 0.08443093,  0.08849293, -0.09065959, ..., -0.00959182,
          0.10152507, -0.12077457],
        [ 0.03628462, -0.02653611, -0.11506603, ..., -0.14669597,
          0.10292757,  0.13625325],
        ...,
        [-0.14210635, -0.12942064, -0.03288083, ...,  0.0568463 ,
         -0.02598592, -0.22455114],
        [ 0.20819993,  0.01196991, -0.09635217, ..., -0.18782297,
          0.10233591,  0.20114912],
        [ 0.1164271 , -0.07769038, -0.06414707, ..., -0.06539135,
         -0.05518465,  0.25142196]], dtype=float32)>,
 <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.34589   , -0.30134732, -0.43572   , ..., -0.3102559 ,
          0.34630865,  0.2613009 ],
        [ 0.14154069,  0.17045322, -0.17749965, ..., -0.02712595,
          0.17292541, -0.2922624 ],
        [ 0.07106856, -0.0739173 , -0.3641197 , ..., -0.3794833 ,
          0.36470377,  0.23766585],
        ...,
        [-0.2582597 , -0.25323495, -0.06649272, ...,  0.16527973,
         -0.04292646, -0.58768904],
        [ 0.43155715,  0.03135502, -0.33463806, ..., -0.47625306,
          0.33486888,  0.35035062],
        [ 0.23173636, -0.20141824, -0.22034441, ..., -0.16035017,
         -0.17478186,  0.48899865]], dtype=float32)>)
```

Build Encoder## Build Decoder

```
class Decoder(tf.keras.models.Model):
    def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> None:
        super().__init__(*args,**kwargs)
        self.units=units
        self.embedding_dim=embedding_dim
        self.vocab_size=vocab_size
        self.embedding=Embedding(
```

```
            vocab_size,
            embedding_dim,
            name='decoder_embedding',
            mask_zero=True,
            embeddings_initializer=tf.keras.initializers.HeNormal()
        )
        self.normalize=LayerNormalization()
        self.lstm=LSTM(
            units,
            dropout=.4,
            return_state=True,
            return_sequences=True,
            name='decoder_lstm',
            kernel_initializer=tf.keras.initializers.HeNormal()
        )
        self.fc=Dense(
            vocab_size,
            activation='softmax',
            name='decoder_dense',
            kernel_initializer=tf.keras.initializers.HeNormal()
        )

    def call(self,decoder_inputs,encoder_states):
        x=self.embedding(decoder_inputs)
        x=self.normalize(x)
        x=Dropout(.4)(x)

x,decoder_state_h,decoder_state_c=self.lstm(x,initial_state=encoder_states)
        x=self.normalize(x)
        x=Dropout(.4)(x)
        return self.fc(x)

decoder=Decoder(lstm_cells,embedding_dim,vocab_size,name='decoder')
decoder(_[1][:1],encoder(_[0][:1]))

<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[3.4059247e-04, 5.7348556e-05, 2.1294907e-05, ...,
         7.2067953e-05, 1.5453645e-03, 2.3599296e-04],
        [1.4662130e-03, 8.0250365e-06, 5.4062020e-05, ...,
         1.9187471e-05, 9.7244098e-05, 7.6433855e-05],
        [9.6929165e-05, 2.7441782e-05, 1.3761305e-03, ...,
         3.6009602e-05, 1.5537882e-04, 1.8397317e-04],
        ...,
        [1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ...,
         1.9552530e-04, 1.7106640e-05, 1.0252406e-04],
        [1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ...,
         1.9552530e-04, 1.7106640e-05, 1.0252406e-04],
        [1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ...,
         1.9552530e-04, 1.7106640e-05, 1.0252406e-04]]], dtype=float32)>
```

Build 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])
```

```
<tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy=
array([[[3.40592262e-04, 5.73484940e-05, 2.12948853e-05, ...,
         7.20679745e-05, 1.54536311e-03, 2.35993255e-04],
        [1.46621116e-03, 8.02504110e-06, 5.40619949e-05, ...,
         1.91874733e-05, 9.72440175e-05, 7.64339056e-05],
        [9.69291723e-05, 2.74417835e-05, 1.37613132e-03, ...,
         3.60095728e-05, 1.55378671e-04, 1.83973272e-04],
        ...,
        [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,
         1.95525470e-04, 1.71066222e-05, 1.02524005e-04],
        [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,
         1.95525470e-04, 1.71066222e-05, 1.02524005e-04],
        [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,
         1.95525470e-04, 1.71066222e-05, 1.02524005e-04]],

       [[9.24730921e-05, 3.46553512e-04, 2.07866033e-05, ...,
         3.65934626e-04, 7.63039337e-04, 5.52638434e-04],
        [8.46863186e-05, 3.65541164e-05, 2.54740953e-05, ...,
         7.12379551e-05, 3.62201303e-04, 4.16714087e-04],
        [2.30146630e-04, 3.91469621e-06, 2.72463716e-04, ...,
         9.26126595e-05, 1.03836363e-04, 1.40792166e-04],
        ...,
        [6.84961735e-04, 9.07644513e-04, 2.86691647e-04, ...,
         3.87946144e-04, 6.09236558e-05, 1.12995331e-05],
        [6.84961735e-04, 9.07644513e-04, 2.86691647e-04, ...,
         3.87946144e-04, 6.09236558e-05, 1.12995331e-05],
        [6.84961735e-04, 9.07644513e-04, 2.86691647e-04, ...,
         3.87946144e-04, 6.09236558e-05, 1.12995322e-05]],

       [[1.19036995e-03, 8.10516722e-05, 2.42324077e-05, ...,
         4.99442758e-05, 6.67208573e-04, 9.55566764e-04],
        [1.53046989e-04, 9.76863957e-05, 4.96972689e-06, ...,
         3.24743196e-05, 2.12563842e-04, 1.18708890e-03],
        [9.40205529e-04, 1.80782794e-04, 7.26205144e-06, ...,
         1.96355060e-04, 8.16940737e-05, 1.38416886e-03],
        ...,
        [3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ...,
         2.35450850e-03, 3.25187625e-06, 9.46984728e-05],
        [3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ...,
         2.35450850e-03, 3.25187625e-06, 9.46984728e-05],
        [3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ...,
         2.35450850e-03, 3.25187625e-06, 9.46984728e-05]],

       ...,
```

```
[[9.03617911e-05, 1.57651404e-04, 1.02747028e-04, ...,
  2.20922651e-04, 3.61504179e-04, 2.32456136e-03],
 [1.55469708e-04, 1.53608169e-04, 1.14945491e-04, ...,
  1.88878359e-04, 5.11967926e-04, 5.13108505e-04],
 [8.27641197e-05, 2.83437112e-05, 6.29429938e-04, ...,
  2.15980137e-04, 3.02832137e-04, 1.77760507e-04],
 ...,
 [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,
  4.06600971e-04, 7.58682154e-06, 6.05909081e-05],
 [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,
  4.06600971e-04, 7.58682154e-06, 6.05909081e-05],
 [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,
  4.06600971e-04, 7.58682154e-06, 6.05909081e-05]],

[[3.99837241e-04, 2.36026899e-05, 6.89777007e-05, ...,
  5.94239136e-05, 4.32556757e-04, 4.60232928e-04],
 [3.88111075e-04, 8.31133584e-05, 1.11861555e-04, ...,
  3.03280340e-05, 2.54765386e-04, 2.82170397e-04],
 [2.12516752e-03, 7.19837190e-05, 1.88700986e-04, ...,
  1.86366087e-04, 7.02239413e-05, 2.54370330e-04],
 ...,
 [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,
  2.64523784e-04, 4.05454011e-05, 1.55662783e-04],
 [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,
  2.64523784e-04, 4.05454011e-05, 1.55662783e-04],
 [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,
  2.64523784e-04, 4.05454011e-05, 1.55662783e-04]],

[[3.24600202e-04, 9.31067043e-05, 4.60048941e-05, ...,
  6.66230699e-05, 5.76460850e-04, 1.52416309e-04],
 [7.51478728e-05, 7.63997741e-05, 2.09082973e-05, ...,
  2.55555002e-04, 2.28998848e-04, 4.37303359e-04],
 [1.03114333e-04, 1.55743372e-04, 9.97955431e-06, ...,
  1.12485175e-03, 4.80950950e-03, 6.83143327e-04],
 ...,
 [5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ...,
  3.07609705e-04, 6.09844255e-06, 8.61325825e-05],
 [5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ...,
  3.07609705e-04, 6.09844255e-06, 8.61325825e-05],
 [5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ...,
  3.07609705e-04, 6.09844255e-06, 8.61325825e-05]]], dtype=float32)>
```

Evaluation of Model:

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)
    ]
)
```

Output:

```
Epoch 1/100
23/23 [==============================] - ETA: 0s - loss: 1.6590 - accuracy:
0.2180
Epoch 1: val_loss improved from inf to 1.21875, saving model to ckpt
23/23 [==============================] - 68s 3s/step - loss: 1.6515 - accuracy:
0.2198 - val_loss: 1.2187 - val_accuracy: 0.3072
Epoch 2/100
23/23 [==============================] - ETA: 0s - loss: 1.2327 - accuracy:
0.3087
Epoch 2: val_loss improved from 1.21875 to 1.10877, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 1.2287 - accuracy:
0.3092 - val_loss: 1.1088 - val_accuracy: 0.3415
Epoch 3/100
23/23 [==============================] - ETA: 0s - loss: 1.1008 - accuracy:
0.3368
Epoch 3: val_loss did not improve from 1.10877
23/23 [==============================] - 22s 973ms/step - loss: 1.0984 -
accuracy: 0.3370 - val_loss: 1.1161 - val_accuracy: 0.3315
Epoch 4/100
23/23 [==============================] - ETA: 0s - loss: 1.0209 - accuracy:
0.3536
Epoch 4: val_loss improved from 1.10877 to 0.95189, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 1.0186 - accuracy:
0.3540 - val_loss: 0.9519 - val_accuracy: 0.3718
Epoch 5/100
23/23 [==============================] - ETA: 0s - loss: 0.9622 - accuracy:
0.3673
Epoch 5: val_loss did not improve from 0.95189
23/23 [==============================] - 23s 979ms/step - loss: 0.9672 -
accuracy: 0.3670 - val_loss: 0.9642 - val_accuracy: 0.3666
Epoch 6/100
23/23 [==============================] - ETA: 0s - loss: 0.9159 - accuracy:
```

```
0.3801
Epoch 6: val_loss improved from 0.95189 to 0.94015, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.9182 - accuracy:
0.3796 - val_loss: 0.9401 - val_accuracy: 0.3598
Epoch 7/100
23/23 [==============================] - ETA: 0s - loss: 0.8737 - accuracy:
0.3908
Epoch 7: val_loss improved from 0.94015 to 0.83293, saving model to ckpt
23/23 [==============================] - 52s 2s/step - loss: 0.8746 - accuracy:
0.3900 - val_loss: 0.8329 - val_accuracy: 0.4180
Epoch 8/100
23/23 [==============================] - ETA: 0s - loss: 0.8389 - accuracy:
0.4013
Epoch 8: val_loss improved from 0.83293 to 0.77748, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.8395 - accuracy:
0.4013 - val_loss: 0.7775 - val_accuracy: 0.4305
Epoch 9/100
23/23 [==============================] - ETA: 0s - loss: 0.8148 - accuracy:
0.4094
Epoch 9: val_loss did not improve from 0.77748
23/23 [==============================] - 23s 983ms/step - loss: 0.8187 -
accuracy: 0.4084 - val_loss: 0.8608 - val_accuracy: 0.3830
Epoch 10/100
23/23 [==============================] - ETA: 0s - loss: 0.7889 - accuracy:
0.4200
Epoch 10: val_loss improved from 0.77748 to 0.73131, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.7923 - accuracy:
0.4188 - val_loss: 0.7313 - val_accuracy: 0.4515
Epoch 11/100
23/23 [==============================] - ETA: 0s - loss: 0.7624 - accuracy:
0.4284
Epoch 11: val_loss did not improve from 0.73131
23/23 [==============================] - 22s 965ms/step - loss: 0.7615 -
accuracy: 0.4282 - val_loss: 0.8036 - val_accuracy: 0.4472
Epoch 12/100
23/23 [==============================] - ETA: 0s - loss: 0.7433 - accuracy:
0.4361
Epoch 12: val_loss did not improve from 0.73131
23/23 [==============================] - 23s 984ms/step - loss: 0.7452 -
accuracy: 0.4354 - val_loss: 0.7384 - val_accuracy: 0.4623
Epoch 13/100
23/23 [==============================] - ETA: 0s - loss: 0.7246 - accuracy:
0.4493
Epoch 13: val_loss did not improve from 0.73131
23/23 [==============================] - 23s 988ms/step - loss: 0.7281 -
accuracy: 0.4488 - val_loss: 0.8017 - val_accuracy: 0.4449
Epoch 14/100
23/23 [==============================] - ETA: 0s - loss: 0.7080 - accuracy:
0.4513
Epoch 14: val_loss did not improve from 0.73131
23/23 [==============================] - 23s 995ms/step - loss: 0.7080 -
accuracy: 0.4509 - val_loss: 0.7568 - val_accuracy: 0.4259
```

```
Epoch 15/100
23/23 [==============================] - ETA: 0s - loss: 0.6853 - accuracy:
0.4620
Epoch 15: val_loss did not improve from 0.73131
23/23 [==============================] - 22s 974ms/step - loss: 0.6826 -
accuracy: 0.4616 - val_loss: 0.7376 - val_accuracy: 0.4502
Epoch 16/100
23/23 [==============================] - ETA: 0s - loss: 0.6731 - accuracy:
0.4673
Epoch 16: val_loss did not improve from 0.73131
23/23 [==============================] - 23s 983ms/step - loss: 0.6733 -
accuracy: 0.4672 - val_loss: 0.7646 - val_accuracy: 0.4538
Epoch 17/100
23/23 [==============================] - ETA: 0s - loss: 0.6576 - accuracy:
0.4732
Epoch 17: val_loss improved from 0.73131 to 0.66131, saving model to ckpt
23/23 [==============================] - 52s 2s/step - loss: 0.6539 - accuracy:
0.4738 - val_loss: 0.6613 - val_accuracy: 0.4714
Epoch 18/100
23/23 [==============================] - ETA: 0s - loss: 0.6468 - accuracy:
0.4807
Epoch 18: val_loss improved from 0.66131 to 0.65303, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.6458 - accuracy:
0.4805 - val_loss: 0.6530 - val_accuracy: 0.4993
Epoch 19/100
23/23 [==============================] - ETA: 0s - loss: 0.6353 - accuracy:
0.4881
Epoch 19: val_loss did not improve from 0.65303
23/23 [==============================] - 23s 994ms/step - loss: 0.6357 -
accuracy: 0.4876 - val_loss: 0.7331 - val_accuracy: 0.4677
Epoch 20/100
23/23 [==============================] - ETA: 0s - loss: 0.6194 - accuracy:
0.4968
Epoch 20: val_loss improved from 0.65303 to 0.55054, saving model to ckpt
23/23 [==============================] - 54s 2s/step - loss: 0.6188 - accuracy:
0.4967 - val_loss: 0.5505 - val_accuracy: 0.5221
Epoch 21/100
23/23 [==============================] - ETA: 0s - loss: 0.6160 - accuracy:
0.4978
Epoch 21: val_loss did not improve from 0.55054
23/23 [==============================] - 23s 987ms/step - loss: 0.6182 -
accuracy: 0.4965 - val_loss: 0.6790 - val_accuracy: 0.4979
Epoch 22/100
23/23 [==============================] - ETA: 0s - loss: 0.6011 - accuracy:
0.5052
Epoch 22: val_loss did not improve from 0.55054
23/23 [==============================] - 23s 996ms/step - loss: 0.6011 -
accuracy: 0.5051 - val_loss: 0.6221 - val_accuracy: 0.5277
Epoch 23/100
23/23 [==============================] - ETA: 0s - loss: 0.5950 - accuracy:
0.5079
Epoch 23: val_loss did not improve from 0.55054
```

```
23/23 [==============================] - 23s 987ms/step - loss: 0.5934 -
accuracy: 0.5081 - val_loss: 0.6142 - val_accuracy: 0.5198
Epoch 24/100
23/23 [==============================] - ETA: 0s - loss: 0.5810 - accuracy:
0.5160
Epoch 24: val_loss did not improve from 0.55054
23/23 [==============================] - 22s 971ms/step - loss: 0.5803 -
accuracy: 0.5170 - val_loss: 0.5759 - val_accuracy: 0.5137
Epoch 25/100
23/23 [==============================] - ETA: 0s - loss: 0.5716 - accuracy:
0.5227
Epoch 25: val_loss did not improve from 0.55054
23/23 [==============================] - 23s 986ms/step - loss: 0.5733 -
accuracy: 0.5229 - val_loss: 0.6344 - val_accuracy: 0.5169
Epoch 26/100
23/23 [==============================] - ETA: 0s - loss: 0.5676 - accuracy:
0.5225
Epoch 26: val_loss did not improve from 0.55054
23/23 [==============================] - 22s 963ms/step - loss: 0.5708 -
accuracy: 0.5210 - val_loss: 0.6254 - val_accuracy: 0.4882
Epoch 27/100
23/23 [==============================] - ETA: 0s - loss: 0.5616 - accuracy:
0.5291
Epoch 27: val_loss did not improve from 0.55054
23/23 [==============================] - 23s 988ms/step - loss: 0.5624 -
accuracy: 0.5280 - val_loss: 0.6774 - val_accuracy: 0.5379
Epoch 28/100
23/23 [==============================] - ETA: 0s - loss: 0.5531 - accuracy:
0.5318
Epoch 28: val_loss did not improve from 0.55054
23/23 [==============================] - 22s 949ms/step - loss: 0.5543 -
accuracy: 0.5310 - val_loss: 0.7284 - val_accuracy: 0.5302
Epoch 29/100
23/23 [==============================] - ETA: 0s - loss: 0.5398 - accuracy:
0.5389
Epoch 29: val_loss did not improve from 0.55054
23/23 [==============================] - 23s 1s/step - loss: 0.5391 - accuracy:
0.5398 - val_loss: 0.7385 - val_accuracy: 0.5193
Epoch 30/100
23/23 [==============================] - ETA: 0s - loss: 0.5375 - accuracy:
0.5416
Epoch 30: val_loss improved from 0.55054 to 0.50346, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.5384 - accuracy:
0.5417 - val_loss: 0.5035 - val_accuracy: 0.5411
Epoch 31/100
23/23 [==============================] - ETA: 0s - loss: 0.5270 - accuracy:
0.5481
Epoch 31: val_loss did not improve from 0.50346
23/23 [==============================] - 22s 958ms/step - loss: 0.5262 -
accuracy: 0.5477 - val_loss: 0.5805 - val_accuracy: 0.5457
Epoch 32/100
23/23 [==============================] - ETA: 0s - loss: 0.5304 - accuracy:
```

```
0.5447
Epoch 32: val_loss did not improve from 0.50346
23/23 [==============================] - 22s 963ms/step - loss: 0.5329 -
accuracy: 0.5435 - val_loss: 0.5374 - val_accuracy: 0.5725
Epoch 33/100
23/23 [==============================] - ETA: 0s - loss: 0.5196 - accuracy:
0.5520
Epoch 33: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 975ms/step - loss: 0.5211 -
accuracy: 0.5518 - val_loss: 0.6217 - val_accuracy: 0.5066
Epoch 34/100
23/23 [==============================] - ETA: 0s - loss: 0.5129 - accuracy:
0.5558
Epoch 34: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 1000ms/step - loss: 0.5129 -
accuracy: 0.5556 - val_loss: 0.6070 - val_accuracy: 0.5653
Epoch 35/100
23/23 [==============================] - ETA: 0s - loss: 0.5059 - accuracy:
0.5620
Epoch 35: val_loss did not improve from 0.50346
23/23 [==============================] - 22s 966ms/step - loss: 0.5081 -
accuracy: 0.5614 - val_loss: 0.6153 - val_accuracy: 0.5452
Epoch 36/100
23/23 [==============================] - ETA: 0s - loss: 0.5037 - accuracy:
0.5619
Epoch 36: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 980ms/step - loss: 0.5063 -
accuracy: 0.5617 - val_loss: 0.5328 - val_accuracy: 0.5873
Epoch 37/100
23/23 [==============================] - ETA: 0s - loss: 0.4977 - accuracy:
0.5682
Epoch 37: val_loss did not improve from 0.50346
23/23 [==============================] - 22s 969ms/step - loss: 0.4980 -
accuracy: 0.5682 - val_loss: 0.5976 - val_accuracy: 0.5693
Epoch 38/100
23/23 [==============================] - ETA: 0s - loss: 0.4939 - accuracy:
0.5704
Epoch 38: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 993ms/step - loss: 0.4953 -
accuracy: 0.5687 - val_loss: 0.5937 - val_accuracy: 0.5236
Epoch 39/100
23/23 [==============================] - ETA: 0s - loss: 0.4860 - accuracy:
0.5758
Epoch 39: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 986ms/step - loss: 0.4868 -
accuracy: 0.5746 - val_loss: 0.6155 - val_accuracy: 0.5457
Epoch 40/100
23/23 [==============================] - ETA: 0s - loss: 0.4809 - accuracy:
0.5778
Epoch 40: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 1s/step - loss: 0.4821 - accuracy:
0.5760 - val_loss: 0.5046 - val_accuracy: 0.5662
```

```
Epoch 41/100
23/23 [==============================] - ETA: 0s - loss: 0.4781 - accuracy:
0.5817
Epoch 41: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 990ms/step - loss: 0.4782 -
accuracy: 0.5821 - val_loss: 0.5256 - val_accuracy: 0.5907
Epoch 42/100
23/23 [==============================] - ETA: 0s - loss: 0.4713 - accuracy:
0.5836
Epoch 42: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 982ms/step - loss: 0.4729 -
accuracy: 0.5824 - val_loss: 0.6387 - val_accuracy: 0.5456
Epoch 43/100
23/23 [==============================] - ETA: 0s - loss: 0.4641 - accuracy:
0.5904
Epoch 43: val_loss did not improve from 0.50346
23/23 [==============================] - 23s 1s/step - loss: 0.4627 - accuracy:
0.5908 - val_loss: 0.5668 - val_accuracy: 0.5741
Epoch 44/100
23/23 [==============================] - ETA: 0s - loss: 0.4608 - accuracy:
0.5921
Epoch 44: val_loss improved from 0.50346 to 0.49920, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.4618 - accuracy:
0.5920 - val_loss: 0.4992 - val_accuracy: 0.5768
Epoch 45/100
23/23 [==============================] - ETA: 0s - loss: 0.4592 - accuracy:
0.5902
Epoch 45: val_loss did not improve from 0.49920
23/23 [==============================] - 22s 970ms/step - loss: 0.4599 -
accuracy: 0.5887 - val_loss: 0.5423 - val_accuracy: 0.5854
Epoch 46/100
23/23 [==============================] - ETA: 0s - loss: 0.4535 - accuracy:
0.5978
Epoch 46: val_loss improved from 0.49920 to 0.48429, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.4552 - accuracy:
0.5966 - val_loss: 0.4843 - val_accuracy: 0.6049
Epoch 47/100
23/23 [==============================] - ETA: 0s - loss: 0.4528 - accuracy:
0.5987
Epoch 47: val_loss improved from 0.48429 to 0.47868, saving model to ckpt
23/23 [==============================] - 54s 2s/step - loss: 0.4537 - accuracy:
0.5990 - val_loss: 0.4787 - val_accuracy: 0.5906
Epoch 48/100
23/23 [==============================] - ETA: 0s - loss: 0.4441 - accuracy:
0.6016
Epoch 48: val_loss did not improve from 0.47868
23/23 [==============================] - 23s 982ms/step - loss: 0.4439 -
accuracy: 0.6025 - val_loss: 0.5746 - val_accuracy: 0.5542
Epoch 49/100
23/23 [==============================] - ETA: 0s - loss: 0.4436 - accuracy:
0.6041
Epoch 49: val_loss did not improve from 0.47868
```

```
23/23 [==============================] - 22s 951ms/step - loss: 0.4432 -
accuracy: 0.6045 - val_loss: 0.5058 - val_accuracy: 0.5753
Epoch 50/100
23/23 [==============================] - ETA: 0s - loss: 0.4435 - accuracy:
0.6033
Epoch 50: val_loss did not improve from 0.47868
23/23 [==============================] - 22s 949ms/step - loss: 0.4441 -
accuracy: 0.6043 - val_loss: 0.6037 - val_accuracy: 0.5473
Epoch 51/100
23/23 [==============================] - ETA: 0s - loss: 0.4382 - accuracy:
0.6069
Epoch 51: val_loss did not improve from 0.47868
23/23 [==============================] - 22s 957ms/step - loss: 0.4383 -
accuracy: 0.6067 - val_loss: 0.5206 - val_accuracy: 0.6154
Epoch 52/100
23/23 [==============================] - ETA: 0s - loss: 0.4293 - accuracy:
0.6125
Epoch 52: val_loss did not improve from 0.47868
23/23 [==============================] - 23s 971ms/step - loss: 0.4284 -
accuracy: 0.6123 - val_loss: 0.4997 - val_accuracy: 0.5840
Epoch 53/100
23/23 [==============================] - ETA: 0s - loss: 0.4309 - accuracy:
0.6109
Epoch 53: val_loss improved from 0.47868 to 0.42987, saving model to ckpt
23/23 [==============================] - 52s 2s/step - loss: 0.4317 - accuracy:
0.6094 - val_loss: 0.4299 - val_accuracy: 0.6062
Epoch 54/100
23/23 [==============================] - ETA: 0s - loss: 0.4292 - accuracy:
0.6120
Epoch 54: val_loss did not improve from 0.42987
23/23 [==============================] - 22s 980ms/step - loss: 0.4309 -
accuracy: 0.6115 - val_loss: 0.6996 - val_accuracy: 0.5592
Epoch 55/100
23/23 [==============================] - ETA: 0s - loss: 0.4225 - accuracy:
0.6115
Epoch 55: val_loss did not improve from 0.42987
23/23 [==============================] - 22s 976ms/step - loss: 0.4224 -
accuracy: 0.6102 - val_loss: 0.5500 - val_accuracy: 0.5769
Epoch 56/100
23/23 [==============================] - ETA: 0s - loss: 0.4220 - accuracy:
0.6180
Epoch 56: val_loss did not improve from 0.42987
23/23 [==============================] - 23s 995ms/step - loss: 0.4236 -
accuracy: 0.6169 - val_loss: 0.5689 - val_accuracy: 0.5817
Epoch 57/100
23/23 [==============================] - ETA: 0s - loss: 0.4173 - accuracy:
0.6210
Epoch 57: val_loss did not improve from 0.42987
23/23 [==============================] - 22s 976ms/step - loss: 0.4161 -
accuracy: 0.6217 - val_loss: 0.4614 - val_accuracy: 0.6048
Epoch 58/100
23/23 [==============================] - ETA: 0s - loss: 0.4183 - accuracy:
```

```
0.6198
Epoch 58: val_loss did not improve from 0.42987
23/23 [==============================] - 23s 1s/step - loss: 0.4183 - accuracy:
0.6201 - val_loss: 0.4372 - val_accuracy: 0.6067
Epoch 59/100
23/23 [==============================] - ETA: 0s - loss: 0.4120 - accuracy:
0.6251
Epoch 59: val_loss did not improve from 0.42987
23/23 [==============================] - 23s 994ms/step - loss: 0.4136 -
accuracy: 0.6237 - val_loss: 0.6183 - val_accuracy: 0.5948
Epoch 60/100
23/23 [==============================] - ETA: 0s - loss: 0.4090 - accuracy:
0.6239
Epoch 60: val_loss did not improve from 0.42987
23/23 [==============================] - 23s 980ms/step - loss: 0.4101 -
accuracy: 0.6225 - val_loss: 0.5042 - val_accuracy: 0.6161
Epoch 61/100
23/23 [==============================] - ETA: 0s - loss: 0.4051 - accuracy:
0.6314
Epoch 61: val_loss did not improve from 0.42987
23/23 [==============================] - 23s 1s/step - loss: 0.4077 - accuracy:
0.6296 - val_loss: 0.5100 - val_accuracy: 0.6128
Epoch 62/100
23/23 [==============================] - ETA: 0s - loss: 0.4016 - accuracy:
0.6326
Epoch 62: val_loss did not improve from 0.42987
23/23 [==============================] - 24s 1s/step - loss: 0.4029 - accuracy:
0.6322 - val_loss: 0.5295 - val_accuracy: 0.6005
Epoch 63/100
23/23 [==============================] - ETA: 0s - loss: 0.4049 - accuracy:
0.6323
Epoch 63: val_loss did not improve from 0.42987
23/23 [==============================] - 23s 981ms/step - loss: 0.4069 -
accuracy: 0.6316 - val_loss: 0.5103 - val_accuracy: 0.6088
Epoch 64/100
23/23 [==============================] - ETA: 0s - loss: 0.3951 - accuracy:
0.6335
Epoch 64: val_loss did not improve from 0.42987
23/23 [==============================] - 22s 981ms/step - loss: 0.3943 -
accuracy: 0.6341 - val_loss: 0.5366 - val_accuracy: 0.5869
Epoch 65/100
23/23 [==============================] - ETA: 0s - loss: 0.3967 - accuracy:
0.6344
Epoch 65: val_loss improved from 0.42987 to 0.40702, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.3972 - accuracy:
0.6352 - val_loss: 0.4070 - val_accuracy: 0.6452
Epoch 66/100
23/23 [==============================] - ETA: 0s - loss: 0.3942 - accuracy:
0.6351
Epoch 66: val_loss did not improve from 0.40702
23/23 [==============================] - 22s 961ms/step - loss: 0.3954 -
accuracy: 0.6337 - val_loss: 0.4963 - val_accuracy: 0.6039
```

```
Epoch 67/100
23/23 [==============================] - ETA: 0s - loss: 0.3884 - accuracy:
0.6409
Epoch 67: val_loss did not improve from 0.40702
23/23 [==============================] - 22s 951ms/step - loss: 0.3879 -
accuracy: 0.6424 - val_loss: 0.4651 - val_accuracy: 0.6276
Epoch 68/100
23/23 [==============================] - ETA: 0s - loss: 0.3876 - accuracy:
0.6398
Epoch 68: val_loss improved from 0.40702 to 0.38016, saving model to ckpt
23/23 [==============================] - 52s 2s/step - loss: 0.3870 - accuracy:
0.6388 - val_loss: 0.3802 - val_accuracy: 0.6614
Epoch 69/100
23/23 [==============================] - ETA: 0s - loss: 0.3897 - accuracy:
0.6394
Epoch 69: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 961ms/step - loss: 0.3895 -
accuracy: 0.6395 - val_loss: 0.4046 - val_accuracy: 0.6587
Epoch 70/100
23/23 [==============================] - ETA: 0s - loss: 0.3855 - accuracy:
0.6433
Epoch 70: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 967ms/step - loss: 0.3870 -
accuracy: 0.6432 - val_loss: 0.4162 - val_accuracy: 0.6475
Epoch 71/100
23/23 [==============================] - ETA: 0s - loss: 0.3828 - accuracy:
0.6422
Epoch 71: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 986ms/step - loss: 0.3828 -
accuracy: 0.6423 - val_loss: 0.4099 - val_accuracy: 0.6612
Epoch 72/100
23/23 [==============================] - ETA: 0s - loss: 0.3825 - accuracy:
0.6460
Epoch 72: val_loss did not improve from 0.38016
23/23 [==============================] - 24s 1s/step - loss: 0.3831 - accuracy:
0.6449 - val_loss: 0.5160 - val_accuracy: 0.6117
Epoch 73/100
23/23 [==============================] - ETA: 0s - loss: 0.3795 - accuracy:
0.6451
Epoch 73: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 1s/step - loss: 0.3797 - accuracy:
0.6448 - val_loss: 0.4963 - val_accuracy: 0.6231
Epoch 74/100
23/23 [==============================] - ETA: 0s - loss: 0.3769 - accuracy:
0.6479
Epoch 74: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 975ms/step - loss: 0.3783 -
accuracy: 0.6459 - val_loss: 0.4888 - val_accuracy: 0.6084
Epoch 75/100
23/23 [==============================] - ETA: 0s - loss: 0.3719 - accuracy:
0.6541
Epoch 75: val_loss did not improve from 0.38016
```

```
23/23 [==============================] - 22s 971ms/step - loss: 0.3724 -
accuracy: 0.6538 - val_loss: 0.5175 - val_accuracy: 0.6032
Epoch 76/100
23/23 [==============================] - ETA: 0s - loss: 0.3697 - accuracy:
0.6555
Epoch 76: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 1s/step - loss: 0.3687 - accuracy:
0.6548 - val_loss: 0.4598 - val_accuracy: 0.6059
Epoch 77/100
23/23 [==============================] - ETA: 0s - loss: 0.3702 - accuracy:
0.6552
Epoch 77: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 954ms/step - loss: 0.3713 -
accuracy: 0.6540 - val_loss: 0.5650 - val_accuracy: 0.5824
Epoch 78/100
23/23 [==============================] - ETA: 0s - loss: 0.3685 - accuracy:
0.6548
Epoch 78: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 982ms/step - loss: 0.3675 -
accuracy: 0.6557 - val_loss: 0.4115 - val_accuracy: 0.6292
Epoch 79/100
23/23 [==============================] - ETA: 0s - loss: 0.3659 - accuracy:
0.6584
Epoch 79: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 970ms/step - loss: 0.3662 -
accuracy: 0.6577 - val_loss: 0.3868 - val_accuracy: 0.6516
Epoch 80/100
23/23 [==============================] - ETA: 0s - loss: 0.3626 - accuracy:
0.6628
Epoch 80: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 994ms/step - loss: 0.3627 -
accuracy: 0.6638 - val_loss: 0.4733 - val_accuracy: 0.6388
Epoch 81/100
23/23 [==============================] - ETA: 0s - loss: 0.3623 - accuracy:
0.6578
Epoch 81: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 970ms/step - loss: 0.3621 -
accuracy: 0.6577 - val_loss: 0.5189 - val_accuracy: 0.5979
Epoch 82/100
23/23 [==============================] - ETA: 0s - loss: 0.3603 - accuracy:
0.6612
Epoch 82: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 982ms/step - loss: 0.3600 -
accuracy: 0.6614 - val_loss: 0.4210 - val_accuracy: 0.6280
Epoch 83/100
23/23 [==============================] - ETA: 0s - loss: 0.3608 - accuracy:
0.6604
Epoch 83: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 1s/step - loss: 0.3627 - accuracy:
0.6592 - val_loss: 0.5621 - val_accuracy: 0.6082
Epoch 84/100
23/23 [==============================] - ETA: 0s - loss: 0.3605 - accuracy:
```

```
0.6640
Epoch 84: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 998ms/step - loss: 0.3628 -
accuracy: 0.6634 - val_loss: 0.4241 - val_accuracy: 0.6462
Epoch 85/100
23/23 [==============================] - ETA: 0s - loss: 0.3498 - accuracy:
0.6713
Epoch 85: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 976ms/step - loss: 0.3484 -
accuracy: 0.6713 - val_loss: 0.4425 - val_accuracy: 0.6489
Epoch 86/100
23/23 [==============================] - ETA: 0s - loss: 0.3537 - accuracy:
0.6663
Epoch 86: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 1s/step - loss: 0.3543 - accuracy:
0.6656 - val_loss: 0.4006 - val_accuracy: 0.6716
Epoch 87/100
23/23 [==============================] - ETA: 0s - loss: 0.3503 - accuracy:
0.6698
Epoch 87: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 987ms/step - loss: 0.3493 -
accuracy: 0.6697 - val_loss: 0.4375 - val_accuracy: 0.6527
Epoch 88/100
23/23 [==============================] - ETA: 0s - loss: 0.3497 - accuracy:
0.6714
Epoch 88: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 986ms/step - loss: 0.3495 -
accuracy: 0.6710 - val_loss: 0.5339 - val_accuracy: 0.6160
Epoch 89/100
23/23 [==============================] - ETA: 0s - loss: 0.3500 - accuracy:
0.6671
Epoch 89: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 970ms/step - loss: 0.3501 -
accuracy: 0.6666 - val_loss: 0.4148 - val_accuracy: 0.6438
Epoch 90/100
23/23 [==============================] - ETA: 0s - loss: 0.3494 - accuracy:
0.6661
Epoch 90: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 995ms/step - loss: 0.3529 -
accuracy: 0.6647 - val_loss: 0.4992 - val_accuracy: 0.6324
Epoch 91/100
23/23 [==============================] - ETA: 0s - loss: 0.3479 - accuracy:
0.6718
Epoch 91: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 986ms/step - loss: 0.3482 -
accuracy: 0.6715 - val_loss: 0.6037 - val_accuracy: 0.6195
Epoch 92/100
23/23 [==============================] - ETA: 0s - loss: 0.3436 - accuracy:
0.6767
Epoch 92: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 964ms/step - loss: 0.3452 -
accuracy: 0.6764 - val_loss: 0.4368 - val_accuracy: 0.6462
```

```
Epoch 93/100
23/23 [==============================] - ETA: 0s - loss: 0.3377 - accuracy:
0.6793
Epoch 93: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 984ms/step - loss: 0.3372 -
accuracy: 0.6795 - val_loss: 0.5267 - val_accuracy: 0.6275
Epoch 94/100
23/23 [==============================] - ETA: 0s - loss: 0.3433 - accuracy:
0.6743
Epoch 94: val_loss did not improve from 0.38016
23/23 [==============================] - 22s 964ms/step - loss: 0.3453 -
accuracy: 0.6736 - val_loss: 0.4532 - val_accuracy: 0.6314
Epoch 95/100
23/23 [==============================] - ETA: 0s - loss: 0.3409 - accuracy:
0.6780
Epoch 95: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 987ms/step - loss: 0.3407 -
accuracy: 0.6775 - val_loss: 0.4901 - val_accuracy: 0.6680
Epoch 96/100
23/23 [==============================] - ETA: 0s - loss: 0.3378 - accuracy:
0.6791
Epoch 96: val_loss did not improve from 0.38016
23/23 [==============================] - 23s 991ms/step - loss: 0.3388 -
accuracy: 0.6793 - val_loss: 0.5620 - val_accuracy: 0.6063
Epoch 97/100
23/23 [==============================] - ETA: 0s - loss: 0.3389 - accuracy:
0.6763
Epoch 97: val_loss improved from 0.38016 to 0.33265, saving model to ckpt
23/23 [==============================] - 53s 2s/step - loss: 0.3402 - accuracy:
0.6765 - val_loss: 0.3327 - val_accuracy: 0.6854
Epoch 98/100
23/23 [==============================] - ETA: 0s - loss: 0.3408 - accuracy:
0.6768
Epoch 98: val_loss did not improve from 0.33265
23/23 [==============================] - 22s 974ms/step - loss: 0.3407 -
accuracy: 0.6766 - val_loss: 0.4046 - val_accuracy: 0.6695
Epoch 99/100
23/23 [==============================] - ETA: 0s - loss: 0.3388 - accuracy:
0.6795
Epoch 99: val_loss did not improve from 0.33265
23/23 [==============================] - 23s 985ms/step - loss: 0.3394 -
accuracy: 0.6791 - val_loss: 0.4475 - val_accuracy: 0.6622
Epoch 100/100
23/23 [==============================] - ETA: 0s - loss: 0.3358 - accuracy:
0.6787
Epoch 100: val_loss did not improve from 0.33265
23/23 [==============================] - 22s 968ms/step - loss: 0.3385 -
accuracy: 0.6773 - val_loss: 0.3742 - val_accuracy: 0.6796
```

Final accuracy: 0.6773

Save Model

```
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>
---------------------
```

Create 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()
```

```
Model: "chatbot_encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 encoder_embedding (Embeddin  (None, None, 256)        625408
 g)

 layer_normalization (LayerN  (None, None, 256)        512
 ormalization)

 encoder_lstm (LSTM)         [(None, None, 256),       525312
                              (None, 256),
                              (None, 256)]

=================================================================
Total params: 1,151,232
Trainable params: 1,151,232
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "chatbot_decoder"
__________________________________________________________________________________
________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
=================
 input_4 (InputLayer)           [(None, None)]       0           []

 decoder_embedding (Embedding)  (None, None, 256)    625408
['input_4[0][0]']

 layer_normalization (LayerNorm  (None, None, 256)   512
['decoder_embedding[0][0]']
 alization)

 input_2 (InputLayer)           [(None, 256)]        0           []

 input_3 (InputLayer)           [(None, 256)]        0           []

 decoder_lstm (LSTM)            [(None, None, 256),  525312
['layer_normalization[1][0]',
                                 (None, 256),
'input_2[0][0]',
                                 (None, 256)]
'input_3[0][0]']

 decoder_dense (Dense)          (None, None, 2443)   627851
['decoder_lstm[0][0]']

==================================================================================
=================
Total params: 1,779,083
Trainable params: 1,779,083
Non-trainable params: 0
__________________________________________________________________________________
________________
```

```python
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,
show_layer_activations=True)
```

```python
tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',show_shapes=True,
show_layer_activations=True)
```

Integrating chatbot into web:

Integrating a chatbot into a web app using Flask involves creating a web interface for users to interact with the chatbot.

To do this, set up a Flask application, create routes for handling chat interactions, and integrate a chatbot backend or API for processing and responding to user input.

1. **Create folder Structure:**

```
flask css example
├──templateFiles
│   ├──index.html
│
├──staticFiles
│   ├──main.css
│
├──flask_template.py
```

2. Set up Flask:

Install Flask using pip.
command: pip install flask

```
from flask import Flask, request, render_template

app = Flask(__name)
```

3. Create HTML templates:

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.

4. Define Chatbot routes:

Create Flask routes to handle chat interactions. Typically, it has a route to render the chat interface and another route to handle incoming messages and provide responses.

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

5. Integrate Chatbot Backend or 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.

```
@app.route("/get")
#function for the bot response
def get_bot_response():
    userText = request.args.get('msg')
    return str(englishBot.get_response(userText))
```

6. 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.

7. Run the Flask APP:
Start your Flask application by adding this code at the bottom of your script:

```
if __name__ == '__main__':
    app.run(debug=True)
```

8. 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.

Index.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>
```

Style.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;
}
```

app.py:

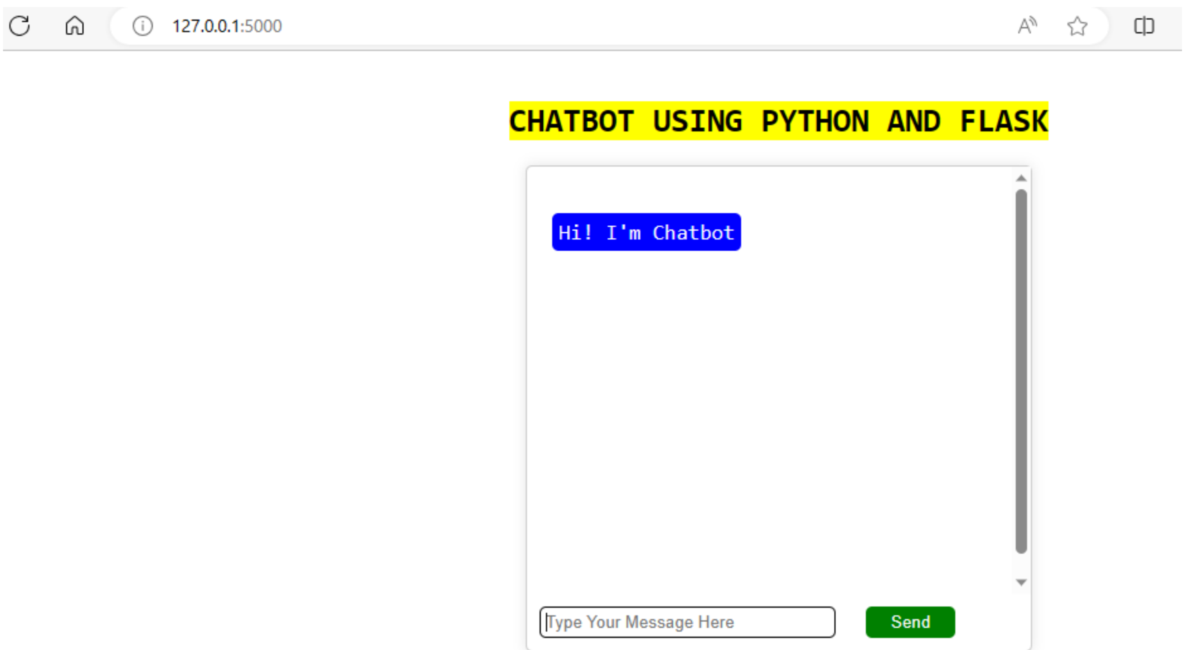
```
from flask import Flask, render_template, request
import pandas as pd
app = Flask(__name__,template_folder='templates',static_folder='static')
csv_file = 'sampled.csv'

#create chatbot
def chatbot(input,csv_file):
    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]

    else:
        return "No data available"

#define app routes
```

```
@app.route("/")
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)
```

Terminal Output:

```
PS C:\Users\ADMIN\Pictures\Screenshots\hello-app>  & 'C:\Users\ADMIN\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\ADMIN\.vscode\extensions\ms-python.python-2023.18.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '62139' '--' 'c:\Users\ADMIN\Pictures\Screenshots\hello-app\app.py'
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 914-844-353
```

Local host Output:



Future Engineering:

1. **Advanced NLP and AI Models**:

Keeping chatbot up-to-date with the latest Natural Language Processing (NLP) models. Explore newer models, like GPT-4 or beyond, to improve conversation quality and context understanding.

2. **Multimodal Capabilities**:

As AI technology progresses, consider adding support for multimodal interactions, such as text, images, and voice. This can make your chatbot more versatile and user-friendly.

3. **Machine Learning for Personalization**:

Implement machine learning algorithms for user personalization. Use historical chat data to offer more tailored responses and recommendations.

4. **Real-time Data Integration**:

Develop mechanisms to integrate real-time data sources, such as IoT devices or live APIs, to provide users with instant, up-to-date information and services.

5. **Continuous Learning**:

Implement reinforcement learning techniques to allow your chatbot to learn from user interactions and adapt its responses over time.

6. **Privacy and Ethics**:

Stay abreast of evolving privacy regulations and ethical considerations in AI development. Ensure your chatbot complies with the latest standards and respects user privacy.

7. **Scalability and High Availability**:

Designing of chatbot to scale seamlessly and maintain high availability, even during high traffic periods.

Feature Extraction:

- Feature extraction is the process of selecting and transforming relevant aspects or characteristics from raw data to create meaningful features that can be used to build machine learning models.

- In the context of text data, like messages in a spam classifier, feature extraction involves converting the text into a format that machine learning algorithms can understand.

Conclusion:

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.