Problem Definition:

The challenge is to create a chatbot in Python that provides exceptional customer service, answering user queries on a website or application. The objective is to deliver high-quality support to users, ensuring a positive user experience and customer satisfaction.

Design Thinking:

Functionality: Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.
User Interface:
Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.
Natural Language Processing (NLP): Implement NLP techniques to understand and process user input in a conversational manner.
Responses: Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.

Integration:
Decide how the chatbot will be integrated with the website or app.
Testing and Improvement: Continuously test and refine the chatbot's performance based on user interactions.

Designing and implementing a chatbox can be a complex task, but I can provide you with a high-level algorithm to get you started. Keep in mind that the implementation details may vary depending on the platform and technologies you're using. Here's a basic algorithm:

1. **Define the Requirements:**
   - Determine the purpose of your chatbox (e.g., customer support, virtual assistant, chatbot, etc.).
   - Specify the features you want, such as text input, messages, user authentication, etc.

2. **Choose a Platform:**
   - Decide whether you're building a web-based chatbox, mobile app, or integrating it into an existing platform.

3. **Select a Technology Stack:**
   - Choose the programming languages, frameworks, and libraries you'll use (e.g., Python, JavaScript, Node.js, React, etc.).

4. **User Interface (UI) Design:**
   - Create a user-friendly and visually appealing chatbox interface with message bubbles, input field, and any other elements you need.

5. **Backend Development:**
   - Develop the backend server to handle user interactions, process messages, and store chat history if necessary.
   - Implement natural language processing (NLP) if you want the chatbox to understand and respond to user messages effectively.

6. **Frontend Development:**
   - Build the frontend components for the chatbox, including the input field and message display area.
   - Implement real-time updates to display messages as they come in.

7. **User Authentication (Optional):**
   - If required, add user authentication and authorization to secure the chatbox.

8. **Database Integration (Optional):**
   - Set up a database to store chat histories and user data.

9. **Testing:**
   - Thoroughly test the chatbox for usability and functionality.
   - Test it with real users to gather feedback for improvements.

10. **Deployment:**
    - Deploy your chatbox on a server or hosting platform of your choice.

11. **Maintenance and Updates:**
    - Regularly update and maintain your chatbox to fix bugs, add features, and improve its performance.

12. **Scalability (Optional):**
    - Plan for scalability, especially if you expect a large user base. Consider load balancing and cloud-based solutions.

13. **Monitoring and Analytics (Optional):**
    - Implement monitoring tools and analytics to track user interactions and improve the chatbox's responses over time.

14. **Security:**
    - Ensure the chatbox is secure and protect user data. Implement encryption and follow security best practices.

15. **Documentation:**
    - Document your code and chatbox usage to make it easier for others to understand and work with.

This algorithm provides a general guideline for designing and implementing a chatbox. The specific steps and technologies you use may vary based on your project's requirements and constraints.

Create a chatbot in python

Project title: Create a chatbot in python

Phase 3: Development Part 1

Topic: *Start building the create a chatbot in python model by loading and pre-processing the dataset*

CREATE A CHATBOT IN PYTHON

Introduction:

Creating a chatbot in Python is an exciting and useful project that can be a valuable addition to various applications, websites, or even standalone conversational agents. A chatbot is a computer program designed to simulate human conversation and interact with users. In this introduction, I'll outline the essential steps and considerations to get you started on your journey to creating a chatbot in Python.

Step 1: Define the Purpose and Scope

Before you begin coding, it's crucial to determine the purpose and scope of your chatbot. Ask yourself what problem it will solve or what tasks it will assist with. Understanding the goals of your chatbot will help you define its functionalities and capabilities.

Step 2: Choose a Framework or Library

Python offers several libraries and frameworks that can simplify chatbot development.

Necessary step to follow:

1.Import Libraries:

Start by importing the necessary libraries.

Program:

```
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

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()
```

Output:

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

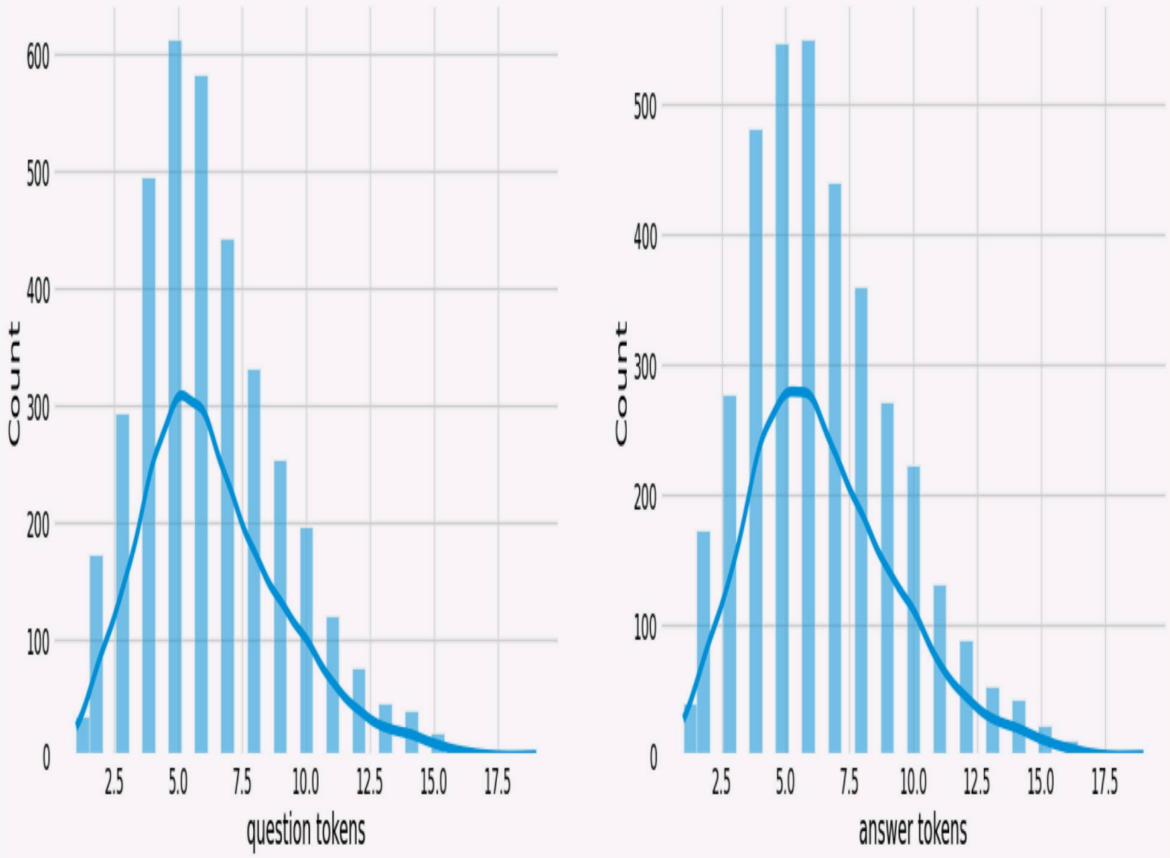
Data Preprocessing:

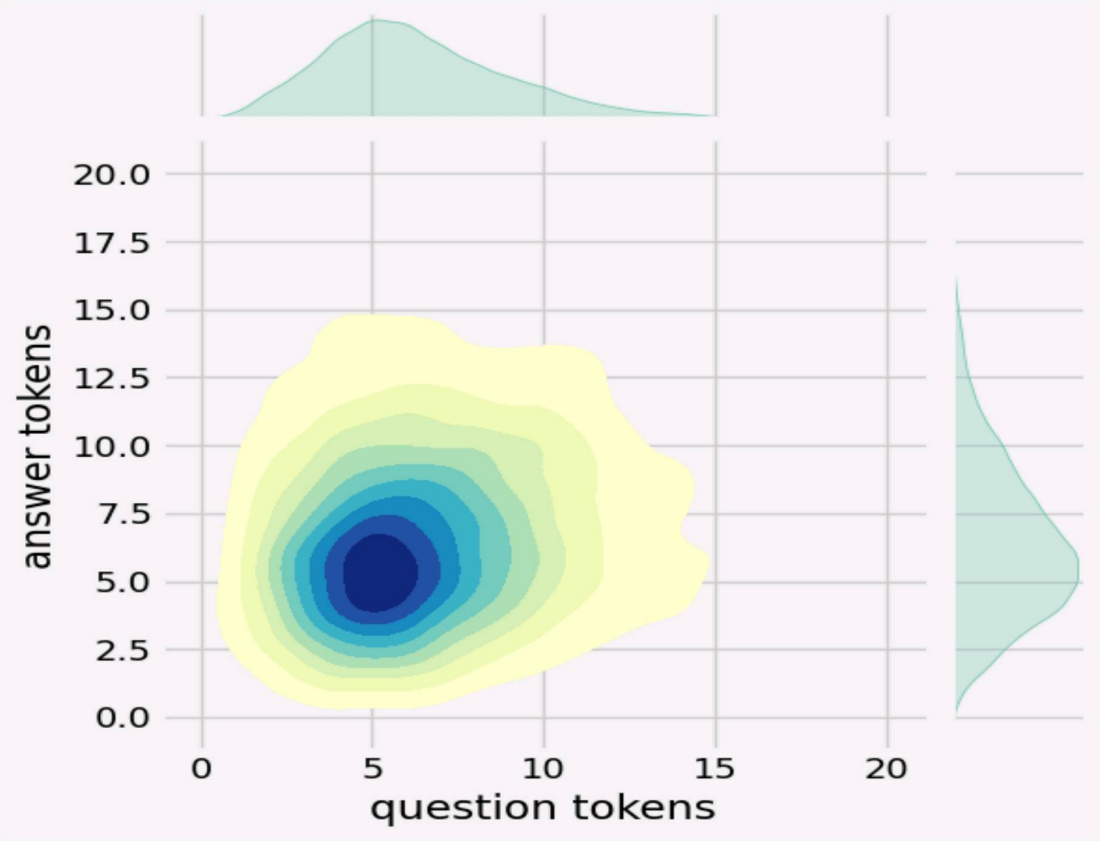
Data Visualization:

```
df['question tokens']=df['question'].apply(lambda x:len(x.split()))

df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))

plt.style.use('fivethirtyeight')

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

sns.set_palette('Set2')

sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])

sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=ax[1])

sns.jointplot(x='question tokens',y='answer tokens',data=df,kind='kde',fill=True,
cmap='YlGnBu')
```

```
plt.show()
```

output:





Text 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)
```

Output:

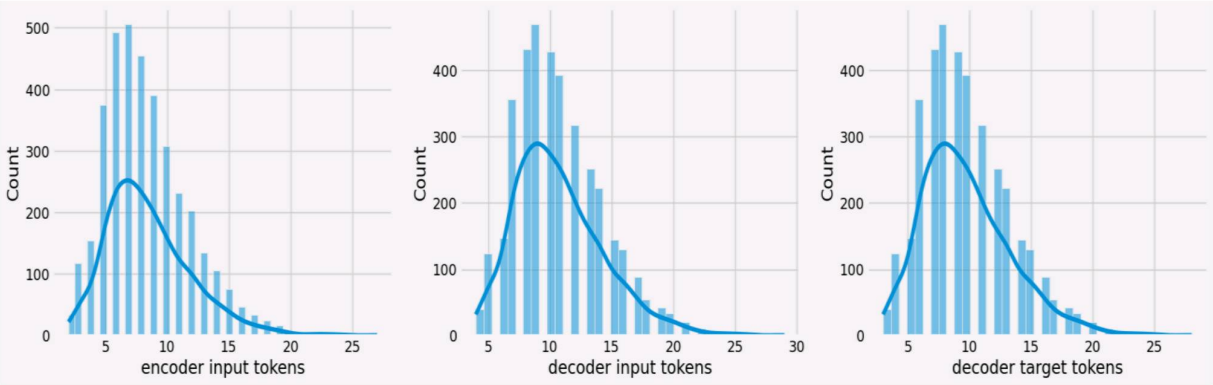
question	answer	encoder_inputs	decoder_targets	decoder_inputs	
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i'm pretty good. thanks for asking.	no problem. so how have you been?	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem. so how have you been?	i've been great. what about you?	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> i ' ve been great . what about you ? ...
4	i've been great. what about you?	i've been good. i'm in school right now.	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...

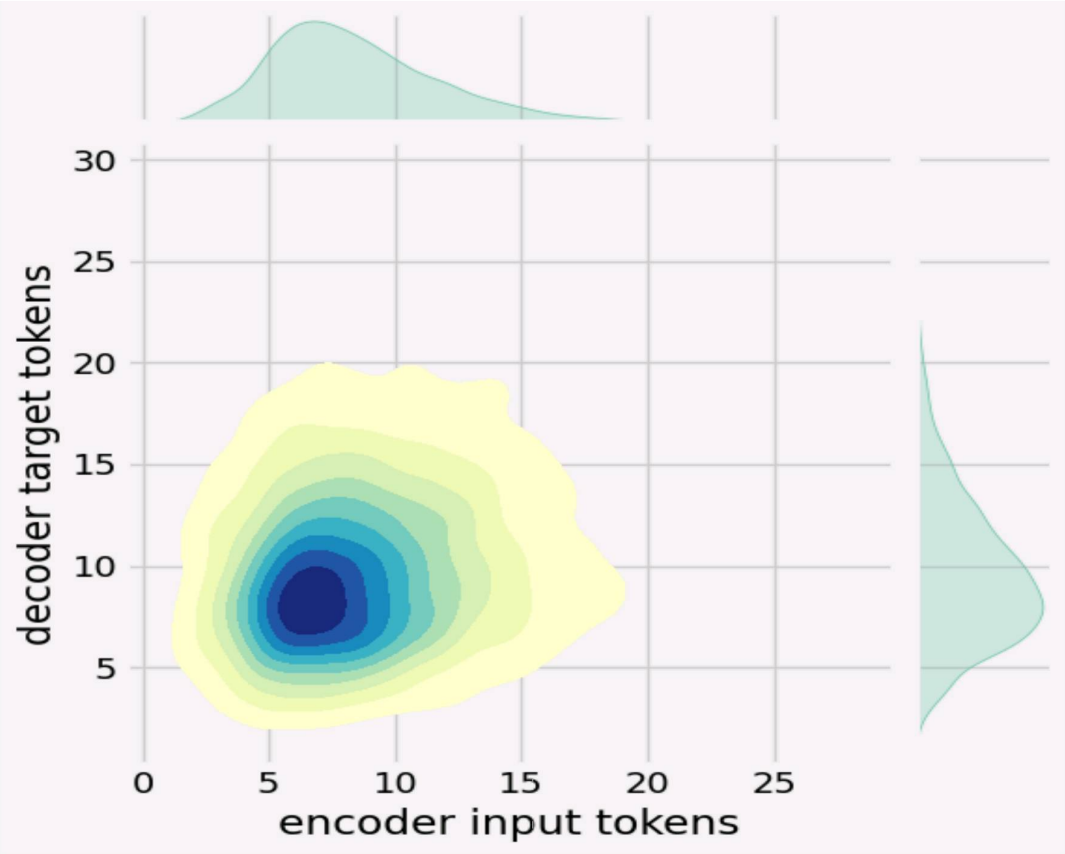
question	answer	encoder_inputs	decoder_targets	decoder_inputs	
5	i've been good. i'm in school right now.	what school do you go to?	i ' ve been good . i ' m in school right now .	what school do you go to ? <end>	<start> what school do you go to ? <end>
6	what school do you go to?	i go to pcc.	what school do you go to ?	i go to pcc . <end>	<start> i go to pcc . <end>
7	i go to pcc.	do you like it there?	i go to pcc .	do you like it there ? <end>	<start> do you like it there ? <end>
8	do you like it there?	it's okay. it's a really big campus.	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...
9	it's okay. it's a really big campus.	good luck with school.	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

Input:

```
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='kd
e',fill=True,cmap='YlGnBu')
plt.show()
```

Output:





Conclusion:

In conclusion, chatbots have become a pivotal part of modern technology and have made a significant impact on various industries and applications. Here are the key takeaways:

Enhanced Customer Engagement: Chatbots have revolutionized customer service, providing immediate responses and 24/7 availability. They improve customer engagement by answering queries, resolving issues, and guiding users through processes.

Efficiency and Automation: Chatbots automate repetitive and time-consuming tasks, reducing the need for human intervention. This leads to increased operational efficiency and cost savings for businesses.

Personalization: Advanced chatbots leverage AI and machine learning to offer personalized recommendations and responses, enhancing user experiences and driving customer satisfaction.

Scalability: Chatbots can handle a large number of user interactions simultaneously, making them ideal for businesses with varying customer loads.

Data Insights: Chatbots collect and analyze valuable user data, enabling businesses to gain insights into customer preferences, behavior, and pain points, which can inform strategic decisions.

Create a chatbot in Python

Project title: Create a chatbot in Python

Phase 4: Development part 2
Topic: start building the create a chatbot in Python model by loding and pre-processing the dataset.

CREATE A CHATBOT IN PYTHON

PROGRAM:

```
import numpy as np
 import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
 from keras.layers import Dense
import json
 import re
import string
from sklearn.feature_extraction.text import TfidfVectorizer
import unicodedata
from sklearn.model_selection import train_test_split
```

In [2]:

```
question  =[]
answer = []
with open("../input/sjaj-data/Sjaj-Data.txt",'r') as f :
    for line in f :
        line  =  line.split('\t')
        question.append(line[0])
        answer.append(line[1])
print(len(question) == len(answer))
```

```
True
```

In [3]:

```
question[:5]
```

Out[3]:

```
['holidays are important for everyone not just the businessman',
 'The nail to the right of the front door brought back the sweet
memories of the bird to the woman',
 'radar speed guns dont need any maintenance',
 'radar speed guns can be maintained if they are checked and fixed
regularly',

 'a banker needs to be a good architect']
```

In [4]:

```
answer[:5]
```

Out[4]:

```
['true\n', 'true\n', 'false\n', 'true\n', 'false\n']
```

In [5]:

```
answer = [ i.replace("\n","") for i in answer]
```

In [6]:

```
answer[:5]
```

Out[6]:

```
['true', 'true', 'false', 'true', 'false']
```

In [7]:

```
data = pd.DataFrame({"question" : question ,"answer":answer})
data.head()
```

Out[7]:

	question	answer

0	holidays are important for everyone not just t...	true
1	The nail to the right of the front door brough...	true
2	radar speed guns dont need any maintenance	false
3	radar speed guns can be maintained if they are...	true
4	a banker needs to be a good architect	false

In [8]:

```
def unicode_to_ascii(s):
    return ''.join(c for c in unicodedata.normalize('NFD', s)
      if unicodedata.category(c) != 'Mn')
```

In [9]:

```
def clean_text(text):
    text = unicode_to_ascii(text.lower().strip())
    text = re.sub(r"i'm", "i am", text)
    text = re.sub(r"\r", "", text)
    text = re.sub(r"he's", "he is", text)
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"it's", "it is", text)
    text = re.sub(r"that's", "that is", text)
    text = re.sub(r"what's", "that is", text)
    text = re.sub(r"where's", "where is", text)
    text = re.sub(r"how's", "how is", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'ve", " have", text)
```

```
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "cannot", text)
    text = re.sub(r"n't", " not", text)
    text = re.sub(r"n'", "ng", text)
    text = re.sub(r"'bout", "about", text)
    text = re.sub(r"'til", "until", text)
    text = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", text)
#     text = re.sub(r"[-()\"#/@:<>{}`+=~|.!?]", "", text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = re.sub("(\\W)"," ",text)
    text = re.sub('\S*\d\S*\s*','', text)
    text =  "<sos> " +  text + " <eos>"
    return text
```

In [10]:

```
data["question"][0]
```

Out[10]:

```
'holidays are important for everyone not just the businessman'
```

In [11]:

```
data["question"] = data.question.apply(clean_text)
```

In [12]:

```
data["question"][0]
```

Out[12]:

```
'<sos> holidays are important for everyone not just the businessman
<eos>'
```

In [13]:

```
data["answer"] = data.answer.apply(clean_text)
```

In [14]:

```
question  = data.question.values.tolist()
answer =  data.answer.values.tolist()
```

In [15]:

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

    tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,

padding='post')

    return tensor, lang_tokenizer
```

In [16]:

```
input_tensor , inp_lang  =  tokenize(question)
```

In [17]:

```
target_tensor , targ_lang  =  tokenize(answer)
```

In [18]:

```
#len(inp_question) ==  len(inp_answer)
```

In [19]:

```
def remove_tags(sentence):
    return sentence.split("<start>")[-1].split("<end>")[0]
```

In [20]:

```
max_length_targ, max_length_inp = target_tensor.shape[1],
input_tensor.shape[1]
```

In [21]:

```
# Creating training and validation sets using an 80-20 split
input_tensor_train, input_tensor_val, target_tensor_train,
target_tensor_val = train_test_split(input_tensor, target_tensor,
test_size=0.2)
```

In [22]:

```
#print(len(train_inp) , len(val_inp) , len(train_target) ,
len(val_target))
```

In [23]:

```
BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 19
steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
embedding_dim = 256
units = 1024
vocab_inp_size = len(inp_lang.word_index)+1
vocab_tar_size = len(targ_lang.word_index)+1

dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train,
target_tensor_train)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)

example_input_batch, example_target_batch = next(iter(dataset))
example_input_batch.shape, example_target_batch.shape
```

Out[23]:

```
(TensorShape([19, 22]), TensorShape([19, 3]))
```

In [24]:

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
```

```
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size,
embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units,
                                       return_sequences=True,
                                       return_state=True,

recurrent_initializer='glorot_uniform')

    def call(self, x,hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```

In [25]:

```
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)

# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch,
sample_hidden)
print ('Encoder output shape: (batch size, sequence length, units)
{}'.format(sample_output.shape))
print ('Encoder Hidden state shape: (batch size, units)
{}'.format(sample_hidden.shape))
```

```
Encoder output shape: (batch size, sequence length, units) (19, 22,
1024)
Encoder Hidden state shape: (batch size, units) (19, 1024)
```

In [26]:

```
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
```

```
        # query hidden state shape == (batch_size, hidden size)
        # query_with_time_axis shape == (batch_size, 1, hidden size)
        # values shape == (batch_size, max_len, hidden size)
        # we are doing this to broadcast addition along the time axis
to calculate the score
        query_with_time_axis = tf.expand_dims(query, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying score to
self.V
        # the shape of the tensor before applying self.V is
(batch_size, max_length, units)
        score = self.V(tf.nn.tanh(
            self.W1(query_with_time_axis) + self.W2(values)))

        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

In [27]:

```
attention_layer = BahdanauAttention(10)
attention_result, attention_weights = attention_layer(sample_hidden,
sample_output)

print("Attention result shape: (batch size, units)
{}".format(attention_result.shape))
print("Attention weights shape: (batch_size, sequence_length, 1)
{}".format(attention_weights.shape))
```

```
Attention result shape: (batch size, units) (19, 1024)
Attention weights shape: (batch_size, sequence_length, 1) (19, 22, 1)
```

In [28]:

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
```

```
        self.embedding = tf.keras.layers.Embedding(vocab_size,
embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_units,
                                       return_sequences=True,
                                       return_state=True,

recurrent_initializer='glorot_uniform')
        self.fc = tf.keras.layers.Dense(vocab_size)

        # used for attention
        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)
        context_vector, attention_weights = self.attention(hidden,
enc_output)

        # x shape after passing through embedding == (batch_size, 1,
embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim
+ hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # output shape == (batch_size, vocab)
        x = self.fc(output)

        return x, state, attention_weights
```

In [29]:

```
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)

sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE,
1)),
                                      sample_hidden, sample_output)

print ('Decoder output shape: (batch_size, vocab size)
{}'.format(sample_decoder_output.shape))
```

```
Decoder output shape: (batch_size, vocab size) (19, 5)
```

In [30]:

```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

In [31]:

```
@tf.function
def train_step(inp, targ, enc_hidden):
    loss = 0

    with tf.GradientTape() as tape:
        enc_output, enc_hidden = encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([targ_lang.word_index['<sos>']] *
BATCH_SIZE, 1)

        # Teacher forcing - feeding the target as the next input
        for t in range(1, targ.shape[1]):
            # passing enc_output to the decoder
            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden,
enc_output)

            loss += loss_function(targ[:, t], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(targ[:, t], 1)

    batch_loss = (loss / int(targ.shape[1]))

    variables = encoder.trainable_variables +
decoder.trainable_variables
```

```
    gradients = tape.gradient(loss, variables)

    optimizer.apply_gradients(zip(gradients, variables))

    return batch_loss
```

In [32]:

```
EPOCHS = 40

for epoch in range(1, EPOCHS + 1):
    enc_hidden = encoder.initialize_hidden_state()
    total_loss = 0

    for (batch, (inp, targ)) in
enumerate(dataset.take(steps_per_epoch)):
        batch_loss = train_step(inp, targ, enc_hidden)
        total_loss += batch_loss

    if(epoch % 4 == 0):
        print('Epoch:{:3d} Loss:{:.4f}'.format(epoch,
                                               total_loss /
steps_per_epoch))
```

```
Epoch:  4 Loss:0.9066
Epoch:  8 Loss:0.6143
Epoch: 12 Loss:0.4061
Epoch: 16 Loss:0.3050
Epoch: 20 Loss:0.2286
Epoch: 24 Loss:0.0560
Epoch: 28 Loss:0.2241
Epoch: 32 Loss:0.0407
Epoch: 36 Loss:0.0054
Epoch: 40 Loss:0.0059
```

In [33]:

```
def evaluate(sentence):
    sentence = clean_text(sentence)

    inputs = [inp_lang.word_index[i] for i in sentence.split(' ')]
    inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],

maxlen=max_length_inp,
```

```
padding='post')
    inputs = tf.convert_to_tensor(inputs)

    result = ''

    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)

    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([targ_lang.word_index['<sos>']], 0)

    for t in range(max_length_targ):
        predictions, dec_hidden, attention_weights = decoder(dec_input,

dec_hidden,
                                                             enc_out)

        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))

        predicted_id = tf.argmax(predictions[0]).numpy()

        result += targ_lang.index_word[predicted_id] + ' '

        if targ_lang.index_word[predicted_id] == '<eos>':
            return remove_tags(result), remove_tags(sentence)

        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)

    return remove_tags(result), remove_tags(sentence)
```

In [34]:

```
questions  =[]
answers = []
with open("../input/sjaj-data/Sjaj-Data.txt",'r') as f :
    for line in f :
        line  =  line.split('\t')
        questions.append(line[0])
        answers.append(line[1])
print(len(question) == len(answer))
```

```
True
```

In [35]:

```
def ask(sentence):
    result, sentence = evaluate(sentence)

    print('Question: %s' % (sentence))
    print('Predicted answer: {}'.format(result))
ask(questions[5])
```

```
Question: <sos> khudhair found it very easy to control the personal
feelings storming inside him <eos>
Predicted answer: false <eos>
```

In [36]:

```
ask(questions[23])
```

```
Question: <sos> zaid tariq was saved by sea <eos>
Predicted answer: true <eos>
```

In [37]:

```
print(answers[23])
```

```
true
```

الخال

فُزنا ورب الكعبة

In [ ]: