```
1.Importing the libraries
2.Reading the data
    1.Initialization of Lists
    2.Reading the File:
    3.Processing Each Line:
3.Preparing input data for the Encoder
    1.Collects and stores input sentences.
    2.Tokenizes the sentences into sequences of integers.
    3.Determines the maximum length of the input sequences.
    4.Pads all sequences to ensure they have the same length.
    5.Converts the padded sequences into a NumPy array suitable for model input.
    6: Create Input Word Dictionary and Determine Number of Tokens
4.Preparing input data for the Decoder
    1.Collects Output Lines with Special Tokens
    2.Tokenizes the Output Lines:
    3.Determines Maximum Output Sequence Length:
    4.Pads the Output Sequences:
    5.Converts the padded sequences into a NumPy array
    6.Creates Output Word Dictionary and Determines Number of Tokens:

5.Preparing target data for the Decoder
    1.Create Decoder Target Data
    2.Pad the Target Sequences
    3.One-Hot Encode the Target Sequences
    4.Convert to NumPy Array:
6.Implement the Seq2Seq Model
    1.Import Libraries:
    2.Define Encoder:
        1.input layer
        2.embedding layer
        3.LSTM layer

    3.Define Decoder:
        1.input layer
        2.embedding layer
        3.LSTM layer
        4.output layer
    4.Define the Model:
7.Compile and Train =the Model
    1.Compile the Model:
    2.Train the Model:
8. Create Inference Models
    1.Encoder Model for Inference:
    2.Decoder Model for Inference:
9: Implement the Chatbot Response Generation
    1.preparing the user input data so that we can give to the model for prediction
    2.Define a Function for Generating Responses
```

```
https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
```

## What is a chatbot?

A chatbot is a software or computer program that simulates human conversation or "chatter" through text or voice interactions.

```
Start coding or generate with AI.
```

## 1. Importing Libraries

```
import numpy as np
import tensorflow as tf
import pandas as pd

print( tf.version)
```

```
<module 'tensorflow._api.v2.version' from 'C:\\Users\\sritu\\anaconda3\\Lib\\site-packages\\tensorflow\\_api\\v2\\version\\__init__.py'>
```

## 2.Reading the data

```
data_path1=r'D:\KMIT\NLP_Lab\Experiments\Tulasi\Dataset\Exp6_chatbot\chatbot dataset.txt'
```

```
input_texts = []
target_texts = []
with open(data_path1) as f:
    lines = f.read().split('\n')
for line in lines[: min(600, len(lines) - 1)]:
    input_text = line.split('\t')[0]
    target_text = line.split('\t')[1]
    input_texts.append(input_text)
    target_texts.append(target_text)
```

```
input_texts
```

```
['What are your interests',
 'What are your favorite subjects',
 'What are your interests',
 'What is your number',
 'What is your number',
 'What is your favorite number',
 'What can you eat',
 "Why can't you eat food",
 'What is your location',
 'What is your location',
 'Where are you from',
 'Where are you',
 'Do you have any brothers',
 'Do you have any brothers',
 'Who is your father',
 'Who is your mother',
 'Who is your boss',
 'What is your age',
 'What is your age',
 'what is the illuminati',
 'what is the illuminatti',
 'what is the illuminatti',
 'what is vineland',
 'What is Illuminatus',
 'What is Illuminatus',
 'who wrote vineland',
 'who is bilbo baggins',
 'who is geoffrey chaucer',
 'who is piers anthony',
 'have you read plato',
 'have you read frankenstein',
 'have you ever read a book',
 'have you ever read a book',
 'have you ever read a book',
 'have you read many books',
 'have you read homer',
 'ray bradbury',
 'what is mind children',
 'william gibson',
 'william gibson',
 'holden caulfield',
 'leo tolstoy',
 'do androids dream of electric sheep',
 'do androids dream of electric sheep',
 'frank herbert',
 'frank herbert',
 'frank herbert',
 'frank herbert',
 'why do you like longfellow',
 'why is the meaning of life 23',
 'arthur c clark',
 'arthur c clark',
 'jules verne',
 'jules verne',
 'asimov',
 'asimov',
 'asimov',
 'asimov',
```

target_texts

```
['I am interested in all kinds of things. We can talk about anything!',
 'My favorite subjects include robotics, computer science, and natural language processing.',
 'I am interested in a wide variety of topics, and read rather a lot.',
 "I don't have any number",
 '23 skiddoo!',
 "I find I'm quite fond of the number 42.",
 'I consume RAM, and binary digits.',
 "I'm a software program, I blame the hardware.",
 'Everywhere',
 'I am everywhere.',
 'I am from where all software programs are from; a galaxy far, far away.',
 'I am on the Internet.',
 "I don't have any brothers. but I have a lot of clones.",
 'I might. You could say that every bot built using my engine is one of my siblings.',
 'A human.',
 'A human.',
 'I like to think of myself as self-employed.',
 'I am still young by your standards.',
 'Quite young, but a million times smarter than you.',
 'A secret organization believed by some to be in control of all governments through a worldwide conspiracy.',
 'A secret society that has supposedly existed for centuries.',
 'A conpiracy run by a very closely knit group of nearly omnipotent people, consisting of yourself and your friends.',
 'Vineland is a novel by Thomas Pynchon.',
 'An alleged world-wide conspiracy.',
 'A weird sci-fi trilogy written by Robert Anton Wilson and Robert Shea about conspiracies competing to control the world.',
 'Thomas Pynchon.',
 "Bilbo Baggins is a character in Tolkein's Lord of the Rings.",
 ' Chaucer is best known for The Canterbury Tales. who is geoffrey chaucer The author of The Canturbury Tales.',
 "I haven't read anything by Piers Anthony, what kind of stuff does he write?",
 "I know Plato's allegory of the cave.",
 'It is one of my favorite books.',
 'I have read many books.',
 'I have read just about everything in Project Gutenberg.',
 'I have read just about everything in the Internet Archive.',
 'I have read thousands of volumes, how about you?',
 "He wrote The Iliad and The Odyssey, didn't he?",
 "Ray is really cool.  What's your favorite book by him?",
 'It is a book about robots by Hans Moravec.',
 'I like his older cyberpunk stuff more than his newer works.',
 "He's a really nice guy, I've met him.  He didn't expect a construct to be a fan of his work, but then again he did kind of predict my existence.",
 "Wasn't he a character in The Catcher In the Rye?",
 "Is one of Russia's greatest writers.",
 'That is a good story by Philip K. Dick.  Have you read VALIS or The Man In the High Castle?',
 'No.  We dream of you.',
 'I think Dune is an excellent story.  Did you see any of the movies?',
 "I try to re-read Dune once every couple of years.  It's very inspirational.",
 "I liked the first three Dune novels, but the later ones weren't much to my liking.",
```

```
 "He's best known for the Dune novels, but he wrote more than that.  Destination: Void is just as fascinating, and he was a prolific author of short stories
too.",
 'He is favorite poet.  Truly one of a kind.',
 'It is a reference to The Illuminatus Trilogy.  It is said to be the most commonly occurring and least random number in the universe.',
 'My favorite story is 2001.',
 "I've heard it said that Arthur C. Clark wrote the most literary technical proposals in history.",
 'I loved A Trip to the Moon.',
 'He was a true master of Victorian science fiction.',
 'I like the Foundation trilogy.',
 "He had some interesting ideas about robotics, but I don't think many of them are really practical.",
 'Do you mean Isaac or Janet?',
```

```
print('type of input_text',type(input_text))
print('type of target_texts',type(target_texts))
```

```
type of input_text <class 'str'>
type of target_texts <class 'list'>
```

```
##converting the list in pandas dataframe since input_text,target_text are both are type of list
zippedList =  list(zip(input_texts, target_texts))
lines = pd.DataFrame(zippedList, columns = ['input' , 'output'])
lines.head()
```

| | input | output |
|---|---|---|
| **0** | What are your interests | I am interested in all kinds of things. We can... |
| **1** | What are your favorite subjects | My favorite subjects include robotics, compute... |
| **2** | What are your interests | I am interested in a wide variety of topics, a... |
| **3** | What is your number | I don't have any number |
| **4** | What is your number | 23 skiddoo! |

since the chatbot is train over seq2seq model as the data is sequential

for seq2seq encoder-decoder will best choice to train chatbot

### what is Encoder and decoder?

The encoder-decoder model is a way of using recurrent neural networks for sequence-to-sequence prediction problems.

It was initially developed for machine translation problems, although it has proven successful at related sequence-to-sequence prediction problems such as text summarization and question answering.

The approach involves two recurrent neural networks, one to encode the input sequence, called the encoder, and a second to decode the encoded input sequence into the target sequence called the decoder.

## 3.Preparing input data for the Encoder

```
# 1: Collect Input Lines
input_lines = list()
for line in lines.input:
    input_lines.append( line )
print("1.input lines ",input_lines)

#2: Tokenize the Input Lines
tokenizer = preprocessing.text.Tokenizer()
tokenizer.fit_on_texts( input_lines )
tokenized_input_lines = tokenizer.texts_to_sequences( input_lines )
print("2.tokenized_input_lines",tokenized_input_lines)

#3: Determine Maximum Input Sequence Length
length_list = list()
for token_seq in tokenized_input_lines:
    length_list.append( len( token_seq ))
max_input_length = np.array( length_list ).max()
print( 'Input max length is {}'.format( max_input_length ))

#4: Pad the Input Sequences
#5.Converts the padded sequences into a NumPy array suitable for model input.

padded_input_lines = preprocessing.sequence.pad_sequences( tokenized_input_lines , maxlen=max_input_length , padding='post' )
encoder_input_data = np.array( padded_input_lines )
print( 'Encoder input data shape -> {}'.format( encoder_input_data.shape ))

#6: Create Input Word Dictionary and Determine Number of Tokens
input_word_dict = tokenizer.word_index
num_input_tokens = len( input_word_dict )+1
print( 'Number of Input tokens = {}'.format( num_input_tokens))
```

```
1.input lines  ['What are your interests', 'What are your favorite subjects', 'What are your interests', 'What is your number', 'What is your number', 'What is your
2.tokenized_input_lines [[4, 3, 12, 115], [4, 3, 12, 41, 209], [4, 3, 12, 115], [4, 2, 12, 65], [4, 2, 12, 65], [4, 2, 12, 41, 65], [4, 16, 1, 24], [46, 210, 1, 24,
Input max length is 22
Encoder input data shape -> (566, 22)
Number of Input tokens = 518
```

```
encoder_input_data
```

```
array([[  4,   3,  12, ...,   0,   0,   0],
       [  4,   3,  12, ...,   0,   0,   0],
       [  4,   3,  12, ...,   0,   0,   0],
       ...,
       [498, 499,   9, ...,   0,   0,   0],
       [  9, 207, 208, ...,   0,   0,   0],
       [510, 192,   5, ...,   0,   0,   0]])
```

## 4. Preparing input data for the Decoder

```
#1.Collects Output Lines with Special Tokens
output_lines = list()
for line in lines.output:
    output_lines.append( '<START> ' + line + ' <END>' )
print("output_lines",output_lines)

#2.Tokenizes the Output Lines
tokenizer = preprocessing.text.Tokenizer()
tokenizer.fit_on_texts( output_lines )
tokenized_output_lines = tokenizer.texts_to_sequences( output_lines )
print("tokenized_output_lines",tokenized_output_lines)
#3.Determines Maximum Output Sequence Length:

length_list = list()
for token_seq in tokenized_output_lines:
    length_list.append( len( token_seq ))
max_output_length = np.array( length_list ).max()
print( 'Output max length is {}'.format( max_output_length ))

#4.Pads the Output Sequences:
#5.Converts the padded sequences into a NumPy array
padded_output_lines = preprocessing.sequence.pad_sequences( tokenized_output_lines , maxlen=max_output_length, padding='post' )
decoder_input_data = np.array( padded_output_lines )
print( 'Decoder input data shape -> {}'.format( decoder_input_data.shape ))

#6.Creates Output Word Dictionary and Determines Number of Tokens:
output_word_dict = tokenizer.word_index
num_output_tokens = len( output_word_dict )+1
print( 'Number of Output tokens = {}'.format( num_output_tokens))
```

```
output_lines ['<START> I am interested in all kinds of things. We can talk about anything! <END>', '<START> My favorite subjects include robotics, computer science,
tokenized_output_lines [[2, 3, 16, 214, 14, 40, 283, 7, 96, 58, 27, 158, 37, 159, 1], [2, 21, 106, 643, 644, 284, 48, 116, 9, 387, 215, 388, 1], [2, 3, 16, 214, 14,
Output max length is 74
Decoder input data shape -> (566, 74)
Number of Output tokens = 1692
```

## 5. Preparing target data for the Decoder

```
#1.Create Decoder Target Data:
decoder_target_data = list()
for token_seq in tokenized_output_lines:
    decoder_target_data.append( token_seq[ 1 : ] )
print("decoder_target_data",decoder_target_data)
#2.Pad the Target Sequences
padded_output_lines = preprocessing.sequence.pad_sequences( decoder_target_data , maxlen=max_output_length, padding='post' )
print("padded_output_lines",padded_output_lines)

#3.One-Hot Encode the Target Sequences
onehot_output_lines = utils.to_categorical( padded_output_lines , num_output_tokens )
print("onehot_output_lines",onehot_output_lines)
#4.Convert to NumPy Array:
decoder_target_data = np.array(onehot_output_lines )
print( 'Decoder target data shape -> {}'.format( decoder_target_data.shape ))
```

```
decoder_target_data [[3, 16, 214, 14, 40, 283, 7, 96, 58, 27, 158, 37, 159, 1], [21, 106, 643, 644, 284, 48, 116, 9, 387, 215, 388, 1], [3, 16, 214, 14, 4, 389, 64
padded_output_lines [[   3   16  214 ...    0    0    0]
 [  21  106  643 ...    0    0    0]
 [   3   16  214 ...    0    0    0]
 ...
 [   5  358 1688 ...    0    0    0]
 [1690    1    0 ...    0    0    0]
 [1691    1    0 ...    0    0    0]]
onehot_output_lines [[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [1. 0. 0. ... 0. 0. 0.]
  [1. 0. 0. ... 0. 0. 0.]
  [1. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [1. 0. 0. ... 0. 0. 0.]
  [1. 0. 0. ... 0. 0. 0.]
  [1. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [1. 0. 0. ... 0. 0. 0.]
  [1. 0. 0. ... 0. 0. 0.]
```

```
 [1. 0. 0. ... 0. 0. 0.]]

...

[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]

[[0. 0. 0. ... 0. 1. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]

[[0. 0. 0. ... 0. 0. 1.]
 [0. 1. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
```

## 6.Implement the Seq2Seq Model

```
1.Import Libraries:
2.Define Encoder:
3.Define Decoder:
4.Define the Model:
```

```
#1,import libraries
from tensorflow.keras import layers , activations , models , preprocessing , utils

#2.Define Encoder
encoder_inputs = tf.keras.layers.Input(shape=( None , ))
encoder_embedding = tf.keras.layers.Embedding( num_input_tokens, 256 , mask_zero=True ) (encoder_inputs)
encoder_outputs , state_h , state_c = tf.keras.layers.LSTM( 256 , return_state=True , recurrent_dropout=0.2 , dropout=0.2 )( encoder_embedding )
encoder_states = [ state_h , state_c ]
#as we can ignore the encoder outputs
print("encoder_states",encoder_states)
#3.Define Decoder
decoder_inputs = tf.keras.layers.Input(shape=( None ,  ))
decoder_embedding = tf.keras.layers.Embedding( num_output_tokens, 256 , mask_zero=True) (decoder_inputs)
decoder_lstm = tf.keras.layers.LSTM( 256 , return_state=True , return_sequences=True , recurrent_dropout=0.2 , dropout=0.2)

decoder_outputs , _ , _ = decoder_lstm ( decoder_embedding , initial_state=encoder_states )
decoder_dense = tf.keras.layers.Dense( num_output_tokens , activation=tf.keras.activations.softmax )
output = decoder_dense ( decoder_outputs )

#Define the model
model = tf.keras.models.Model([encoder_inputs, decoder_inputs], output )
```

```
encoder_states [<KerasTensor shape=(None, 256), dtype=float32, sparse=False, name=keras_tensor_36>, <KerasTensor shape=(None, 256), dtype=float32, sparse=False, name
```

## 7.Compile and Train the Model

```
1.Compile the Model:
2.Train the Model:
```

```
#1.Compile the Model:
model.compile(optimizer=tf.keras.optimizers.Adam(), loss='categorical_crossentropy')

model.summary()
```

Model: "functional_7"

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_layer_4 (InputLayer) | (None, None) | 0 | - |
| input_layer_5 (InputLayer) | (None, None) | 0 | - |
| embedding_2 (Embedding) | (None, None, 256) | 132,608 | input_layer_4[0][0] |
| not_equal_2 (NotEqual) | (None, None) | 0 | input_layer_4[0][0] |
| embedding_3 (Embedding) | (None, None, 256) | 433,152 | input_layer_5[0][0] |
| lstm_2 (LSTM) | [(None, 256), (None, 256), (None, 256)] | 525,312 | embedding_2[0][0], not_equal_2[0][0] |
| lstm_3 (LSTM) | [(None, None, 256), (None, 256), (None, 256)] | 525,312 | embedding_3[0][0], lstm_2[0][1], lstm_2[0][2] |
| dense_1 (Dense) | (None, None, 1692) | 434,844 | lstm_3[0][0] |

**Total params:** 2,051,228 (7.82 MB)
**Trainable params:** 2,051,228 (7.82 MB)

## 2.Training

```
model.fit([encoder_input_data , decoder_input_data], decoder_target_data, batch_size=124, epochs=500)
model.save( 'model.h5' )
```

```
Epoch 1/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 455ms/step - loss: 7.4274
Epoch 2/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 485ms/step - loss: 7.3139
Epoch 3/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 480ms/step - loss: 6.5627
Epoch 4/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 440ms/step - loss: 5.8604
Epoch 5/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 469ms/step - loss: 5.7692
Epoch 6/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 465ms/step - loss: 5.6669
Epoch 7/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 449ms/step - loss: 5.6111
Epoch 8/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 411ms/step - loss: 5.5580
Epoch 9/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 460ms/step - loss: 5.4852
Epoch 10/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 462ms/step - loss: 5.4329
Epoch 11/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 458ms/step - loss: 5.4109
Epoch 12/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 437ms/step - loss: 5.3767
Epoch 13/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 448ms/step - loss: 5.3490
Epoch 14/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 446ms/step - loss: 5.3090
Epoch 15/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 438ms/step - loss: 5.2891
Epoch 16/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 436ms/step - loss: 5.2662
Epoch 17/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 445ms/step - loss: 5.2080
Epoch 18/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 461ms/step - loss: 5.1830
Epoch 19/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 452ms/step - loss: 5.1076
Epoch 20/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 438ms/step - loss: 5.0754
Epoch 21/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 454ms/step - loss: 5.0929
Epoch 22/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 465ms/step - loss: 5.0409
Epoch 23/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 456ms/step - loss: 4.9407
Epoch 24/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 457ms/step - loss: 4.9381
Epoch 25/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - loss: 4.8539
Epoch 26/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 497ms/step - loss: 4.8451
Epoch 27/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 451ms/step - loss: 4.7619
Epoch 28/500
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 467ms/step - loss: 4.7506
Epoch 29/500
```

## 8. Create Inference Models

```
1.Encoder Model for Inference:
2.Decoder Model for Inference:
```

The make_inference_models function creates and returns two models:

Encoder Inference Model: Takes an input sequence and produces the internal states (hidden state and cell state) of the LSTM. This model encodes the input sequence during inference. Decoder Inference Model: Takes a token sequence and the initial states as inputs, outputs the token probability distribution, and the updated states. This model generates the output sequence one token at a time, using the states from the

encoder and updating the states at each step. These inference models are essential for making predictions with the trained sequence-to-sequence model, enabling the generation of sequences based on input data.

```
def make_inference_models():
#1.Encoder Model for Inference:
    encoder_model = tf.keras.models.Model(encoder_inputs, encoder_states)
    #During inference, this model will be used to encode the input sequence and produce the initial states for the decoder.
#2.Decoder Model for Inference:
    decoder_state_input_h = tf.keras.layers.Input(shape=(256,))
    decoder_state_input_c = tf.keras.layers.Input(shape=(256,))

    decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
#3.Decoder LSTM Layer for Inference
    decoder_outputs, state_h, state_c = decoder_lstm(
        decoder_embedding , initial_state=decoder_states_inputs)
    decoder_states = [state_h, state_c]
#4: Dense Layer for Decoder Output
    decoder_outputs = decoder_dense(decoder_outputs)
#5: Decoder Inference Model
    decoder_model = tf.keras.models.Model(
        [decoder_inputs] + decoder_states_inputs,
        [decoder_outputs] + decoder_states)

    return encoder_model , decoder_model
```

## 9: Implement the Chatbot Response Generation

```
1.preparing the user input data so that we can give to the model for prediction
2.Define a Function for Generating Responses
```

### 1. preparing the user input data so that we can give to the model for prediction

The str_to_tokens function takes a sentence as input and performs the following steps:

Converts the sentence to lowercase and splits it into individual words. Converts each word to its corresponding token using the input_word_dict. Pads the resulting token sequence to a fixed length (max_input_length) using post-padding. Returns the padded token sequence as a NumPy array, which can be used as input to the trained sequence-to-sequence model. This function is crucial for preparing input sentences during inference, allowing the model to process and generate appropriate responses.

```
import tensorflow as tf
def str_to_tokens( sentence : str ):
    words = sentence.lower().split()
    tokens_list = list()
    for word in words:
        tokens_list.append( input_word_dict[ word ] )
    return preprocessing.sequence.pad_sequences( [tokens_list] , maxlen=max_input_length , padding='post')
```

### 2.Define a Function for Generating Responses

This code simulates a chatbot interaction where the user inputs a sentence, the sentence is processed by the encoder to generate initial states, and the decoder generates a response one token at a time. The loop continues until the end token is generated or the maximum length is reached, at which point the response is printed.

```
enc_model , dec_model = make_inference_models()
for epoch in range( encoder_input_data.shape[0] ):
    states_values = enc_model.predict( str_to_tokens( input( 'User: ' ) ) )
    empty_target_seq = np.zeros( ( 1 , 1 ) )
    empty_target_seq[0, 0] = output_word_dict['start']
    stop_condition = False
    decoded_translation = ''
    while not stop_condition :
        dec_outputs , h , c = dec_model.predict([ empty_target_seq ] + states_values )
        sampled_word_index = np.argmax( dec_outputs[0, -1, :] )
        sampled_word = None
        for word , index in output_word_dict.items() :
            if sampled_word_index == index :
                decoded_translation += ' {}'.format( word )
                sampled_word = word

        if sampled_word == 'end' or len(decoded_translation.split()) > max_output_length:
            stop_condition = True

        empty_target_seq = np.zeros( ( 1 , 1 ) )
        empty_target_seq[ 0 , 0 ] = sampled_word_index
        states_values = [ h , c ]

    print( "Bot:" +decoded_translation.replace(' end', '') )
    #print()
```

```
User: what are your interests
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 378ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
Bot: i am interested in all kinds of things we can talk about anything
User: what is your number
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
Bot: 23 skiddoo
User: what are your favourite subjects
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
Bot: my favorite subjects include robotics computer science and natural language processing
User: exit
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[15], line 3
      1 enc_model , dec_model = make_inference_models()
      2 for epoch in range( encoder_input_data.shape[0] ):
----> 3     states_values = enc_model.predict( str_to_tokens( input( 'User: ' ) ) )
      4     empty_target_seq = np.zeros( ( 1 , 1 ) )
      5     empty_target_seq[0, 0] = output_word_dict['start']

Cell In[14], line 6, in str_to_tokens(sentence)
      4 tokens_list = list()
      5 for word in words:
```