CREATE A CHATBOT IN PYTHON

INTRODUCTION:

- Chatbots are computer programs that can simulate conversation with humans. They are used in a variety of applications, including customer service, education, and entertainment. Chatbots can be created using a variety of programming languages, but Python is a popular choice due to its simplicity and flexibility.

- Python is a general-purpose programming language that is easy to learn and use. It has a large and active community of developers, and there are many libraries and frameworks available for creating chatbots.

- Chatbot is a free and open-source library that provides a number of features for creating and training chatbots. It supports a variety of different training algorithms, and it can be used to create chatbots that can understand and respond to a wide range of user inputs.

GIVEN DATA SET:

	Question	Answers
1.	hi, how are you doing?	i'm fine. how about yourself?
2.	i'm pretty good. thanks for asking.	no problem. so how have you been?
3.	i've been great. what about you?	i've been good. i'm in school right now.
4.	what school do you go to?	i go to pcc.
5.	do you like it there?	it's okay. it's a really big campus
6.	good luck with school	thank you very much.
7.	how's it going? i'm doing well. how about you?	i'm doing well. how about you?
8.	never better, thanks.	so how have you been lately?
9.	i'm actually in school right now. which school do you attend?	i'm attending pcc right now.
10.	are you enjoying it there?	it's not bad. there are a lot of people there.

NECESSARY STEPS TO FOLLOW:

1.Import Libraries:

Start by importing the necessary libraries in an dataset

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
```

2.Load the Dataset:

Load the dataset into your chatbot library. Most chatbot libraries provide a way to load a dataset into the chatbot. For example, the Chatbot library provides a Chatbot Corpus class that can be used to load a dataset into a chatbot.

Program:

```
 chatbot = ChatBot('My Chatbot')
chatbot.train(df['conversation'].values)
```

3.Exploratory Data Analysis(EDA):

Perform EDA to understand data better. This includes checking for missing values, exploring the data’s statistics, and visualising it to identify patterns

Program:

```
#Check for missing values
Print(df.isnuli().sum())
#Explore statistics
Print(df.(describe())
#Visualise the data (eg.,histogram,scatter plots,etc)
```

Importance of Loading and Preprocessing the Datasets:

- To improve the accuracy of the chatbot. The more data the chatbot is trained on, the better it will be able to understand and respond to user input.

- To make the chatbot more informative. By Preprocessing the dataset to extract features, such as keywords and entities, the chatbot can be made to generate more informative and relevant responses.

- To make the chatbot more engaging. By Preprocessing the dataset to remove noise and irrelevant information, the chatbot can be made to generate more engaging and conversational responses

Challenges involved in Loading and Preprocessing the chatbots Datasets:

- **Data quality:** Chatbot datasets can be noisy and contain errors. This can make it difficult to load and preprocess the data effectively.

- **Data size**: Chatbot datasets can be very large, which can make it time-consuming and computationally expensive to load and preprocess the data.

- **Data format**: Chatbot datasets can be in a variety of different formats, which can make it difficult to load and preprocess the data consistently.

- **Data privacy**: Chatbot datasets may contain sensitive personal information, which must be protected during loading and preprocessing.

EXAMPLES OF CHALLENGES:

- **Handling duplicate data:** Chatbot datasets may contain duplicate data, which can lead to inaccurate results. It is important to identify and remove duplicate data before preprocessing the dataset.

- **Handling missing values:** Chatbot datasets may contain missing values, which can also lead to inaccurate results. It is important to handle missing values in a consistent manner. For example, you may choose to replace missing values with the mean or median value of the feature.

- **Handling outliers:** Chatbot datasets may contain outliers, which are data points that are significantly different from the rest of the data. Outliers can skew the results of the preprocessing and training process. It is important to identify and handle outliers in a consistent manner. For example, you may choose to remove outliers from the dataset or to cap the values of the outliers.

- **Protecting user privacy:** Chatbot datasets may contain sensitive personal information, such as names, addresses, and phone numbers. It is important to protect this information during loading and preprocessing. For example, you may choose to anonymize the data or to encrypt the data.

How to Overcome the Challenges involved in Loading and Preprocessing the chatbots Datasets:

- **Use a data quality tool:** There are a number of data quality tools available that can help you to identify and remove errors in your dataset.

- **Use a data sampling technique:** If your dataset is very large, you may want to use a data sampling technique to reduce the size of the dataset before preprocessing.

- **Use a consistent data format:** Convert your dataset to a consistent data format, such as CSV or JSON. This will make it easier to load and preprocess the data.

- **Protect user privacy:** Take steps to protect any sensitive personal information in your dataset. For example, you may want to anonymize the data or to encrypt the data

<u>LOADING THE DATASET:</u>

It is a crucial step in developing a chatbot that can understand and generate meaningful responses. The dataset serves as the training data for the chatbot's natural language processing (NLP) model. Here's a brief overview of the process:

1. Data Collection:

The first step is to gather a dataset of conversational data. This dataset can be in the form of text, with each example containing a pair of a user message and a chatbot's response. Depending on the specific use case, you can obtain data from various sources, such as customer support chat logs, social media conversations, or even create your own dataset.

2. Data Preprocessing:

- **Cleaning**: Raw conversational data may contain noise or irrelevant information. You need to clean the data by removing HTML tags, special characters, or any other unwanted elements.
- **Tokenization:** Split sentences into words or subword units (tokens) to make it easier for the model to process.

3.Formatting the Dataset:

The dataset should be organized in a format suitable for training. Typically, it's structured as a list of conversation pairs, where each pair consists of an input message and the corresponding chatbot response.

4.Splitting the Dataset:

Divide the dataset into training and testing subsets. The training set is used to teach the chatbot how to respond, while the testing set helps evaluate its performance.

5. Vectorization:

Convert words or tokens into numerical representations (vectors) using techniques like Word Embeddings (e.g., Word2Vec, GloVe) or subword embeddings (e.g., FastText).

6.Model Training:

Train an NLP model for your chatbot using libraries like TensorFlow, PyTorch, or pre-built models such as GPT-3 or GPT-4. During training, the model learns to understand and generate text responses based on the dataset.

Program:

```
  import tensorflow as tf

  from tensorflow.keras.preprocessing.text import Tokenizer

  from tensorflow.keras.preprocessing.sequence

   import pad_sequences

# Sample data (replace with your dataset)
conversations = [
    ("Hello", "Hi there!"),
    ("How are you?", "I'm doing well, thank you."),
    # Add more conversation pairs...

# Extract user messages and chatbot responses
user_messages = [pair[0] for pair in conversations]
chatbot_responses = [pair[1] for pair in conversations]

# Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(user_messages + chatbot_responses)

# Convert text to sequences of integers
```

```
user_messages_sequences =
tokenizer.texts_to_sequences(user_messages)
chatbot_responses_sequences =
tokenizer.texts_to_sequences(chatbot_responses)

# Pad sequences to ensure consistent input length
user_messages_padded =
pad_sequences(user_messages_sequences, padding='post')
chatbot_responses_padded =
pad_sequences(chatbot_responses_sequences, padding='post')
```

PREPROCESSING OF DATASETS:

This may involve cleaning the data, removing stop words, and converting the data to a format that is compatible with your chatbot library.

- Data Visualization
- Text Cleaning
- Normalization
- Tokenization

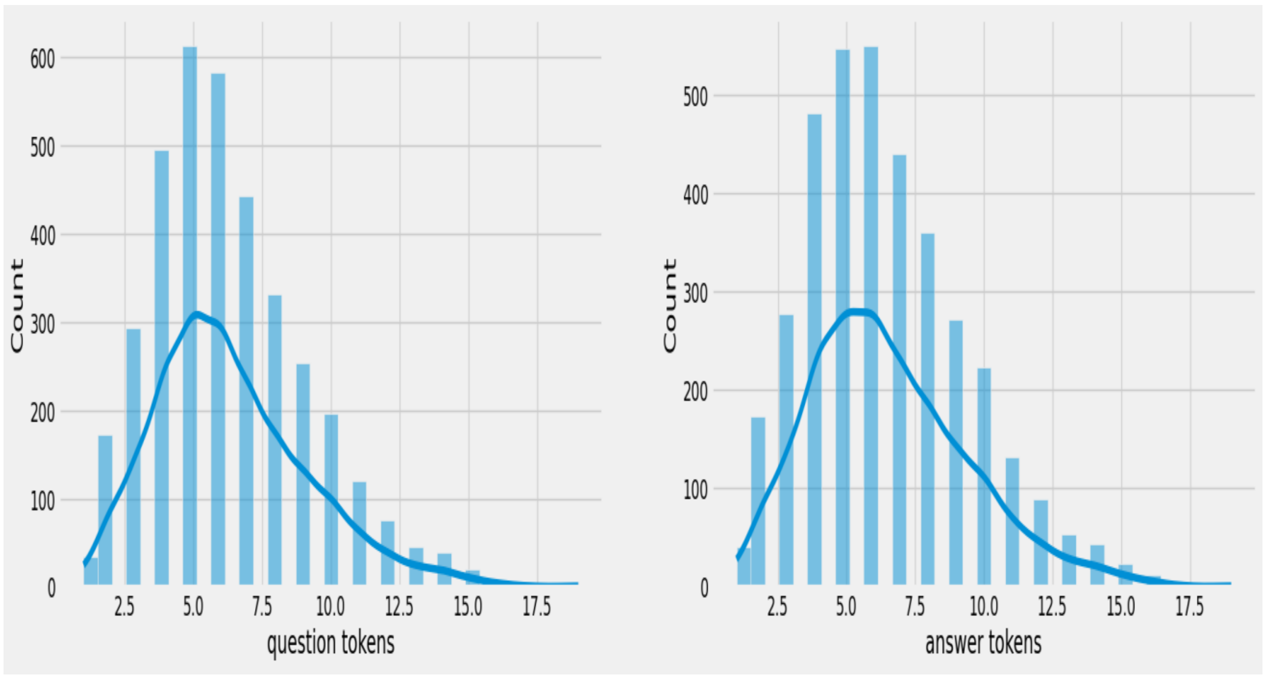
Data Visualization:

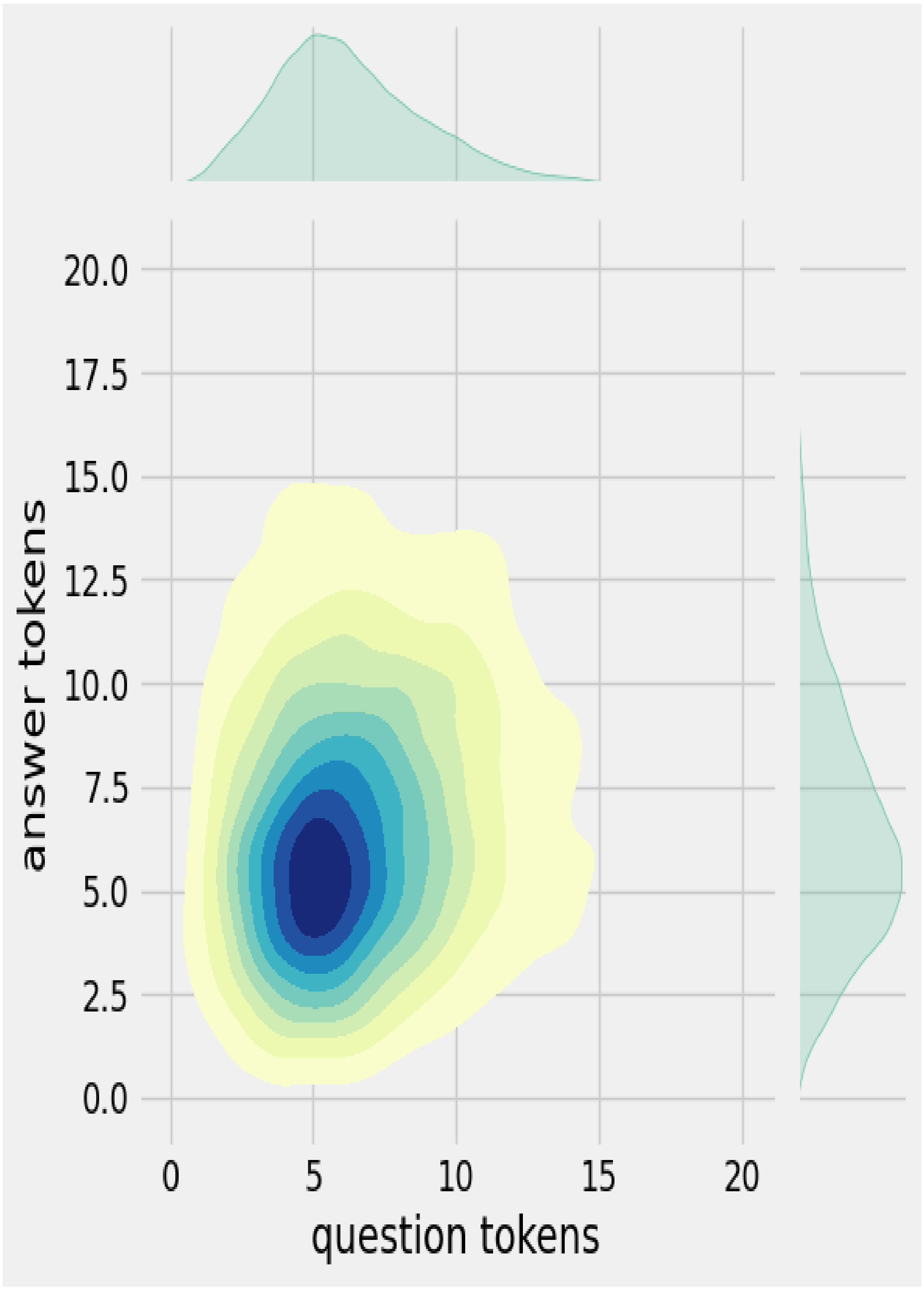
Data visualization is the process of converting data into a visual format, such as a chart, graph, or map, to make it easier to understand and interpret. Data visualization can be used to identify trends and patterns in data, to compare different datasets, and to communicate data to others in a clear and concise way.

Program:

```
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()
```

Plots:





20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
answer tokens
0
5
10
15
20
question tokens

Text Cleaning:

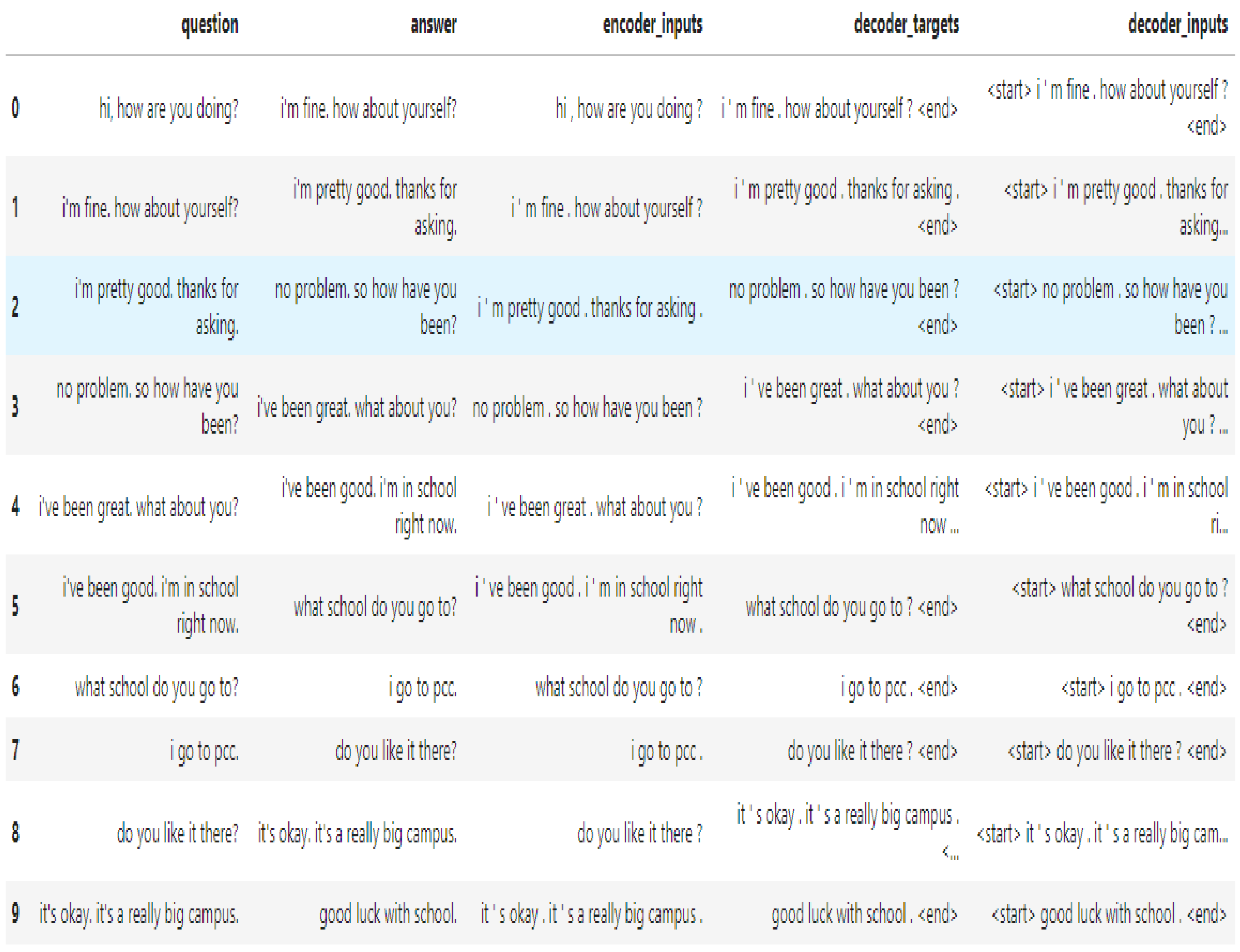
Text cleaning is the process of preparing text data for further processing, such as machine learning or natural language processing (NLP). It involves removing noise and irrelevant information from the text, and converting the text to a format that is compatible with the downstream application.

Program:

```
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...
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>

Normalization:

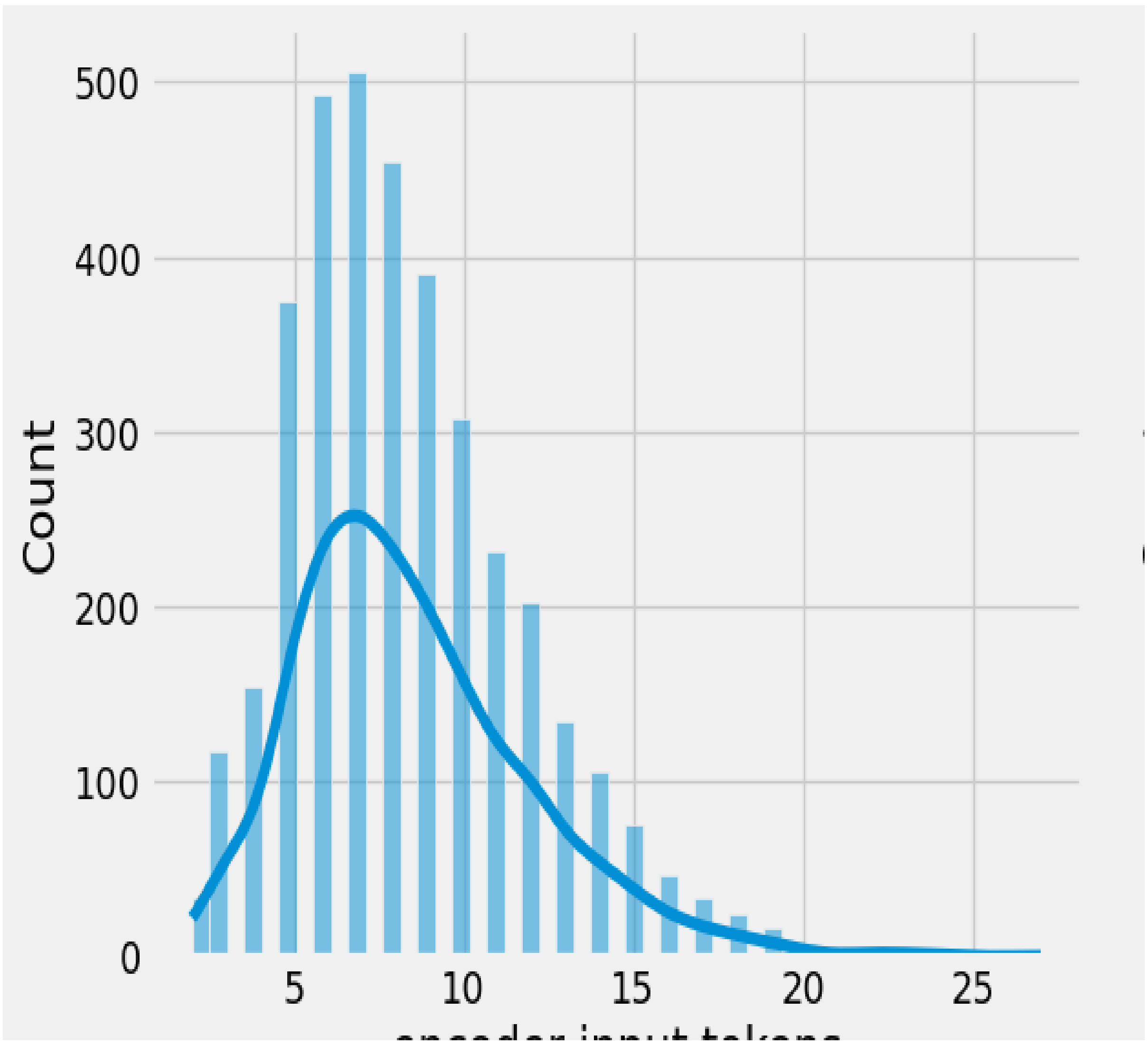
Normalization in chatbot in Python is the process of converting text to a consistent format so that it can be more easily understood and processed by the chatbot. This may involve converting the text to lowercase, removing punctuation, and stemming or lemmatizing the words.

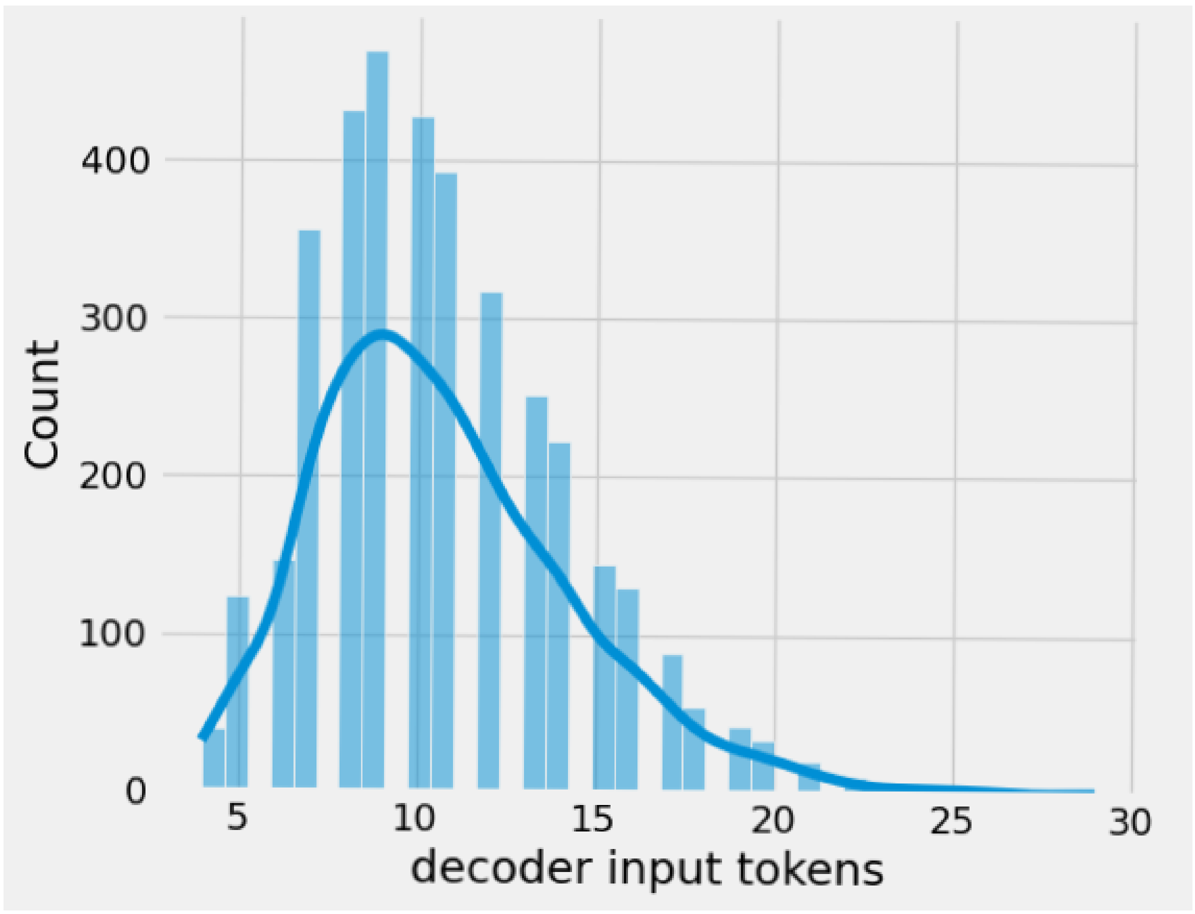
it is an important step in chatbot development because it helps to improve the accuracy of the chatbot. By converting the text to a consistent format, the chatbot can more easily match the user input to its training data and generate a more relevant response.

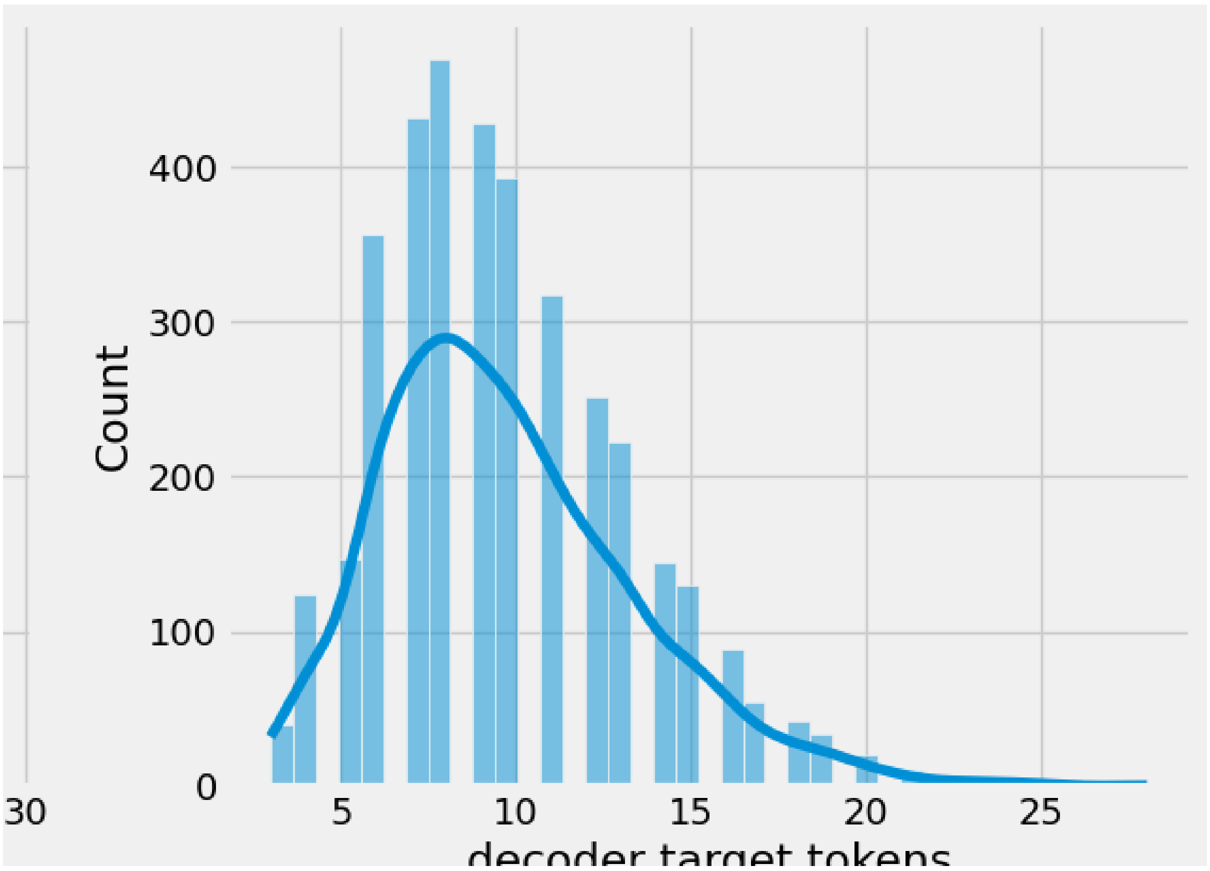
Program:

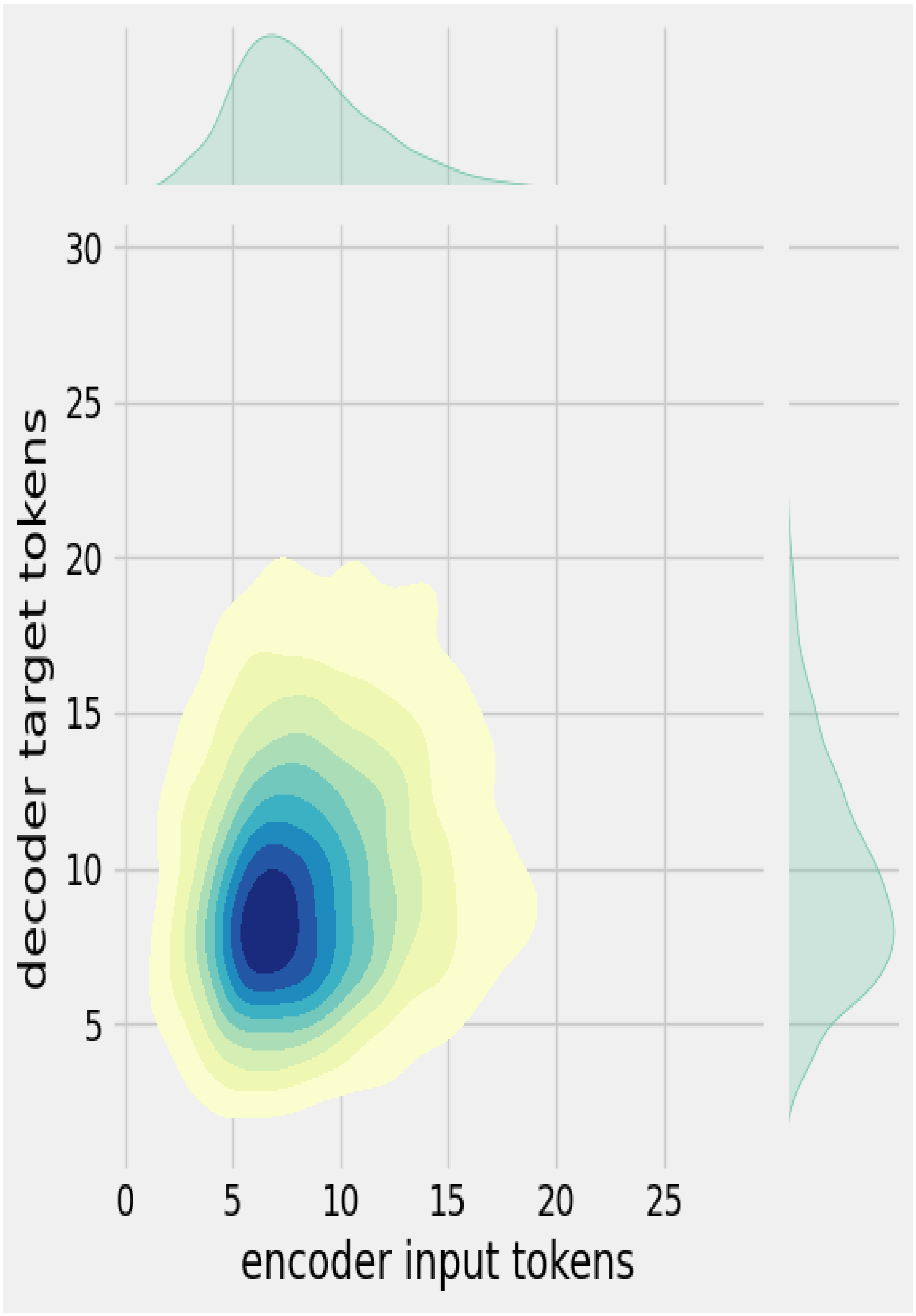
```
df['encoder input tokens']=df['encoder_inputs'].
   apply(lambda    x:len(x.split()))

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()
```




Count
0
100
200
300
400
5
10
15
20
25
30
decoder input tokens


Count
0
100
200
300
400
30
5
10
15
20
25
decoder target tokens



decoder target tokens
30
25
20
15
10
5
0
5
10
15
20
25
encoder input tokens

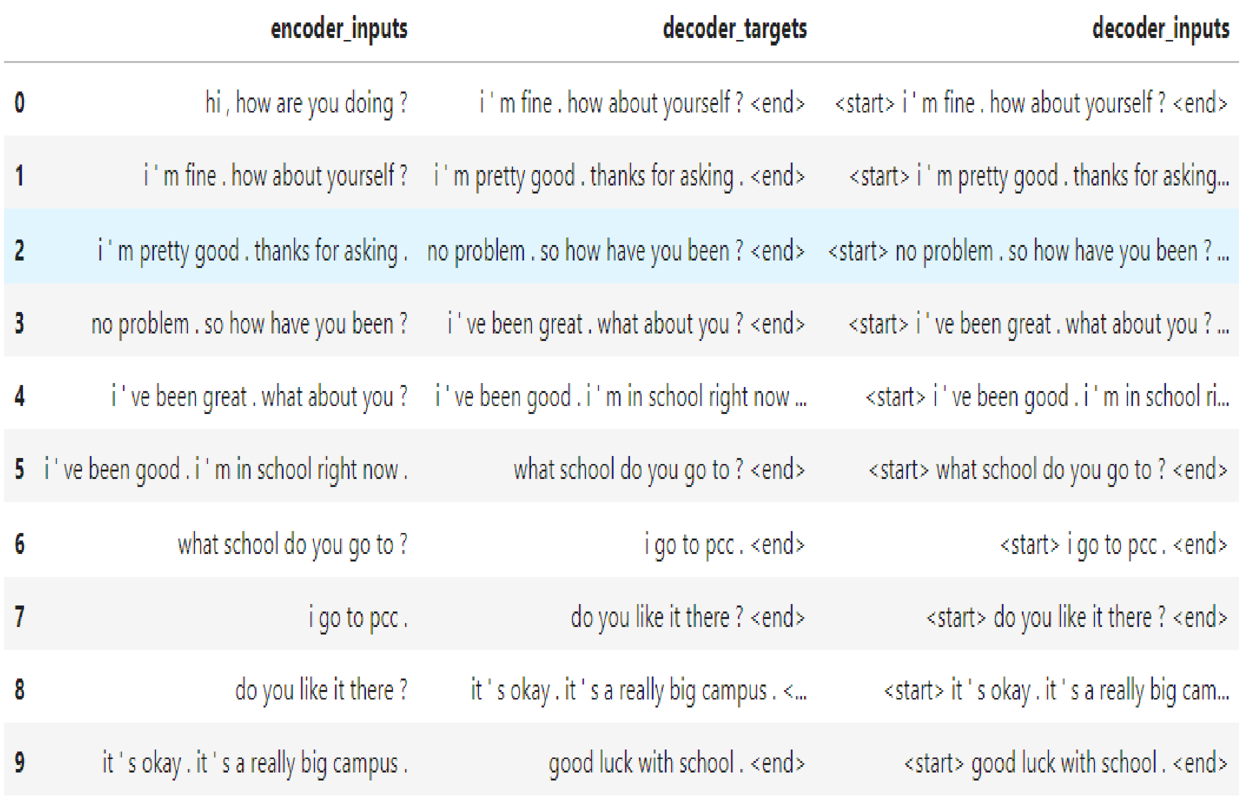
After Processing:

```
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 output:

```
            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
```

Output:



	encoder_inputs	decoder_targets	decoder_inputs
0	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 . <end>	<start> i ' m pretty good . thanks for asking...
2	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 ? <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 ...	<start> i ' ve been good . i ' m in school ri...
5	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 . <end>	<start> i go to pcc . <end>
7	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 . <...	<start> it ' s okay . it ' s a really big cam...
9	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

Tokenization:

Tokenization is the process of splitting text into smaller units, called tokens. Tokens can be words, characters, or subwords. Tokenization is an important step in natural language processing (NLP) tasks, such as machine translation, text summarization, and question answering.

Program:

```
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]}')
```

Output:

```
Vocab size: 2443

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

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

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

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

print(f'Question sentence: hi , how are you ?')

print(f'Question to tokens: {sequences2ids("hi , how are you
?")[:10]}')

print(f'Encoder input shape: {x.shape}')

print(f'Decoder input shape: {yd.shape}')

print(f'Decoder target shape: {y.shape}')
```

Output:

```
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)
```

```
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}')
```

Output:

```
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)
```

CONCLUSION:

Chatbots are a powerful tool that can be used to automate tasks, provide customer service, and educate users. Python is a great language for creating chatbots because it is simple to learn and use, and there are many libraries and frameworks available for chatbot development. developing a chatbot in Python is a dynamic and multifaceted process that combines technical, design, and ethical considerations. It's a field that's continuously evolving, with new tools and techniques emerging regularly. As you embark on your chatbot development journey, remember that success comes not only from mastering the technical aspects but also from empathizing with your users and continuously improving the chatbot's capabilities. Ultimately, a well-designed chatbot has the potential to enhance user experiences, streamline processes, and offer valuable solutions in a variety of domains.