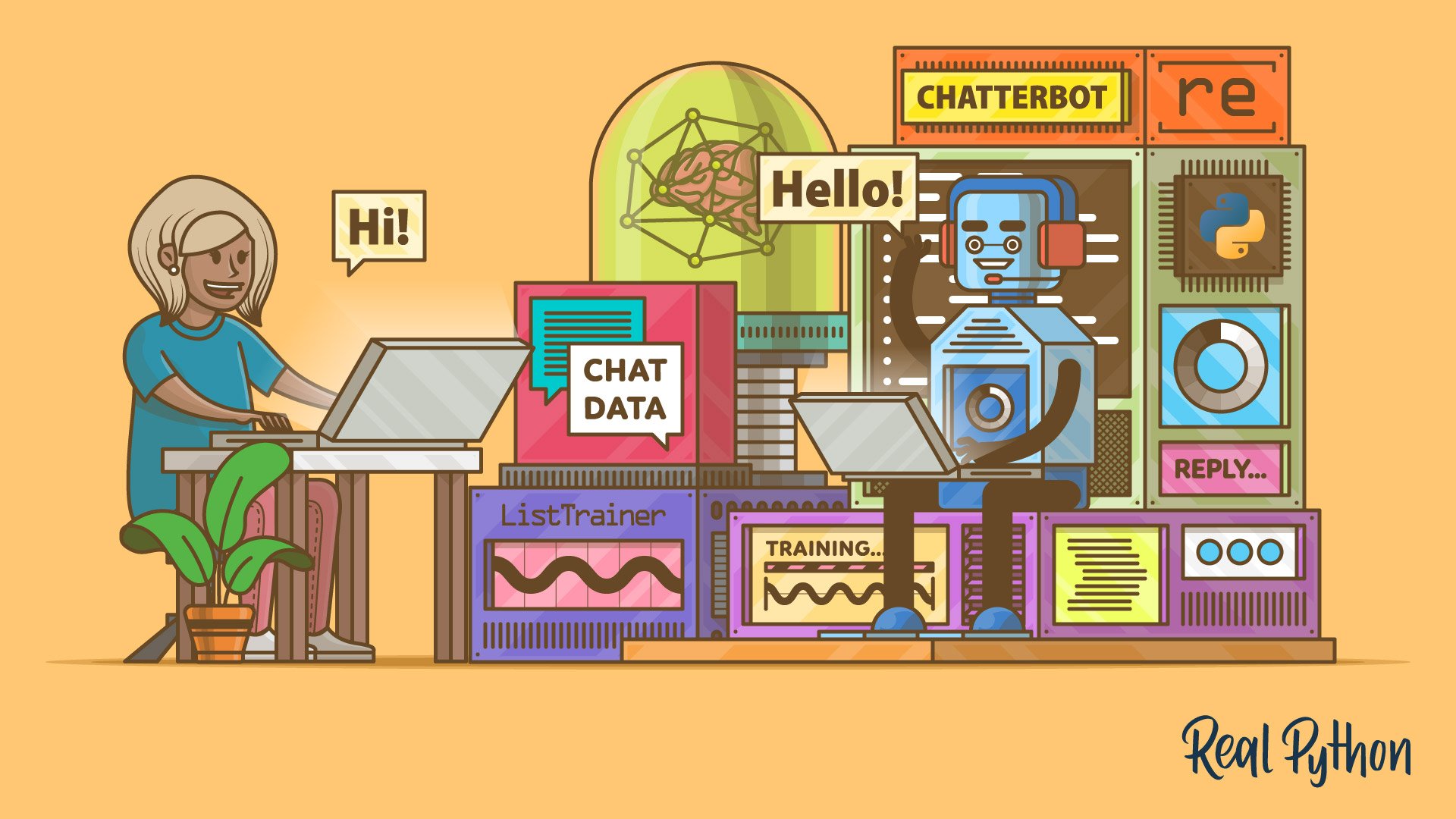
**PHASE 3 DOCUMENT SUBMISSION**

**Project Tittle:** CREATE A CHATBOT IN PYTHON

**PHASE 3:** Development Part 1

**Topic**:  Building Your Project By Loading And Preprocessing The Dataset.



***INTRODUCTION:***

**Rule-Based Chatbot:**

A rule-based chatbot operates on predefined rules and patterns. It matches user input against a set of rules and generates a response accordingly. Here's a step-by-step guide to building a simple rule-based chatbot in Python:

***Define Responses:***

Create a set of predefined responses for different user inputs. These responses will guide the chatbot's behavior.

***Input Processing:***

Take user input and preprocess it, such as converting it to lowercase for easier handling and understanding.

***Rule Matching:***

Match the preprocessed user input against predefined rules and patterns to determine an appropriate response.

***Response Generation:***

*Based on the matched rule, generate a response to the user.*

***Interactive Loop:***

Set up a loop where the chatbot continually interacts with the user, taking their input and providing responses until the conversation is ended***.***

**Create a Simple Dataset:**

dataset = [

("Hello", "Hi there! How can I help you?"),

("How are you?", "I'm just a computer program, but I'm here to assist you."),

("What's your name?", "I'm a chatbot."),

("Goodbye", "Goodbye! Have a great day!"),

# Add more data as needed

]

**Dataset Preparation:**

# Sample dataset (question: answer)

dataset = {

"What is your name?": "My name is Chatbot.",

"How are you?": "I'm doing fine, thank you!",

"What do you do?": "I am a chatbot and I'm here to assist you.",

"Who created you?": "I was created by a team of developers.",

"Good morning": "Good morning! How can I assist you today?",

"Goodbye": "Goodbye! Have a great day!",

# Add more question-answer pairs as needed

}

**Preprocessing Functions:**

import re

import string

def preprocess\_text(text):

# Convert to lowercase

text = text.lower()

# Remove punctuation

text = text.translate(str.maketrans("", "", string.punctuation))

return text

def prepare\_dataset(dataset):

prepared\_dataset = {}

for question, answer in dataset.items():

# Preprocess question and answer

preprocessed\_question = preprocess\_text(question)

preprocessed\_answer = preprocess\_text(answer)

prepared\_dataset[preprocessed\_question] = preprocessed\_answer

return prepared\_dataset

**Chatbot Program:**

import nltk

nltk.download('punkt')

class SimpleChatbot:

def \_\_init\_\_(self):

self.responses = {

"hello": "Hello! How can I assist you today?",

"goodbye": "Goodbye! Have a great day!",

"default": "I'm sorry, I don't understand. Could you please rephrase or ask a different question?"

}

def respond(self, user\_input):

# Convert user input to lowercase for case-insensitive matching

user\_input = user\_input.lower()

# Check if the user input is a predefined response or use the default response

return self.responses.get(user\_input, self.responses["default"])

# Create an instance of the chatbot

chatbot = SimpleChatbot()

# Example conversation

print("User: Hello")

print("Chatbot:", chatbot.respond("hello"))

print("User: How are you?")

print("Chatbot:", chatbot.respond("How are you?"))

print("User: Goodbye!")

print("Chatbot:", chatbot.respond("Goodbye!"))

**Output:**

User: Hello

Chatbot: Hello! How can I assist you today?

User: How are you?

Chatbot: I'm sorry, I don't understand. Could you please rephrase or ask a different question?

User: Goodbye!

Chatbot: Goodbye! Have a great day!

**Data Preprocessing:**

1. Sentence Segmentation
2. Normalization
3. Tokenization

**Segmentation:**

Formatting data to be in a question answer format

In [1]:

#reading data

data=open('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt','r').read()

In [2]:

#paried list of question and corresponding answer

QA\_list=[QA.split('\t') for QA in data.split('\n')]

print(QA\_list[:5])

**Normalization:**

To reduce its randomness, bringing it closer to a predefined “standard”

In [3]:

def remove\_diacritic(text):

return ''.join(char for char in unicodedata.normalize('NFD',text)

if unicodedata.category(char) !='Mn')

In [4]:

def preprocessing(text):

#Case folding and removing extra whitespaces

text=remove\_diacritic(text.lower().strip())

#Ensuring punctuation marks to be treated as tokens

text=re.sub(r"([?.!,¿])", r" \1 ", text)

#Removing redundant spaces

text= re.sub(r'[" "]+', " ", text)

#Removing non alphabetic characters

text=re.sub(r"[^a-zA-Z?.!,¿]+", " ", text)

text=text.strip()

#Indicating the start and end of each sentence

text='<start> ' + text + ' <end>'

return text

## ****Tokenization:****

## **In[5]**

## def tokenize(lang):

## lang\_tokenizer = tf.keras.preprocessing.text.Tokenizer(

## filters='')

## 

## #build vocabulary on unique words

## lang\_tokenizer.fit\_on\_texts(lang)

## 

## return lang\_tokenize

**Word Embedding:**

Representing words in form of real-valued vetors

In[6]

def vectorization(lang\_tokenizer,lang):

#word embedding for training the neural network

tensor = lang\_tokenizer.texts\_to\_sequences(lang)

tensor = tf.keras.preprocessing.sequence.pad\_sequences(tensor,

padding='post')

return tensor

**Creating Chatbot:**

For training and testing the model

In[7]

def load\_Dataset(data,size=None):

if(size!=None):

y,X=data[:size]

else:

y,X=data

X\_tokenizer=tokenize(X)

y\_tokenizer=tokenize(y)

X\_tensor=vectorization(X\_tokenizer,X)

y\_tensor=vectorization(y\_tokenizer,y)

return X\_tensor,X\_tokenizer, y\_tensor, y\_tokenizer

**Tensorflow Dataset:**

In[8]

BUFFER\_SIZE = len(X\_train)

BATCH\_SIZE = 64

steps\_per\_epoch = len(X\_train)//BATCH\_SIZE

embedding\_dim = 256

units = 1024

vocab\_inp\_size = len(X\_tokenizer.word\_index)+1

vocab\_tar\_size = len(y\_tokenizer.word\_index)+1

dataset = tf.data.Dataset.from\_tensor\_slices((X\_train, y\_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

Output:

(TensorShape([64, 24]), TensorShape([64, 24]))

# **Import Libraries**

In[1]

import tensorflow as tf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from tensorflow.keras.layers import TextVectorization

import re,string

from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization

In[2]

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

# **Data Preprocessing**

## Data Visualization:

In[3]

## 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))

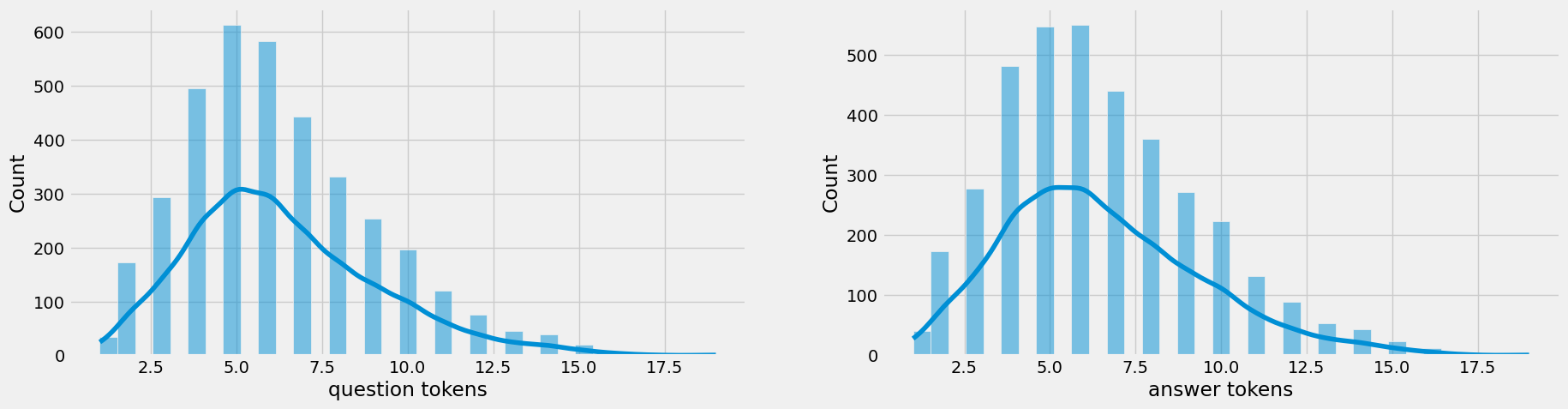
## sns.set\_palette('Set2')

## 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()



## 

## Text Cleaning

In[4]

## df['question'].apply(clean\_text)

## df['decoder\_targets']=df['answer'].apply(clean\_text)+' <end>'

## df['decoder\_inputs']='<start> '+df['answer'].apply(clean\_text)+' <end>'

## 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

In[5]

## df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)

## df['encoder\_inputs']=

## df.head(10)

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')

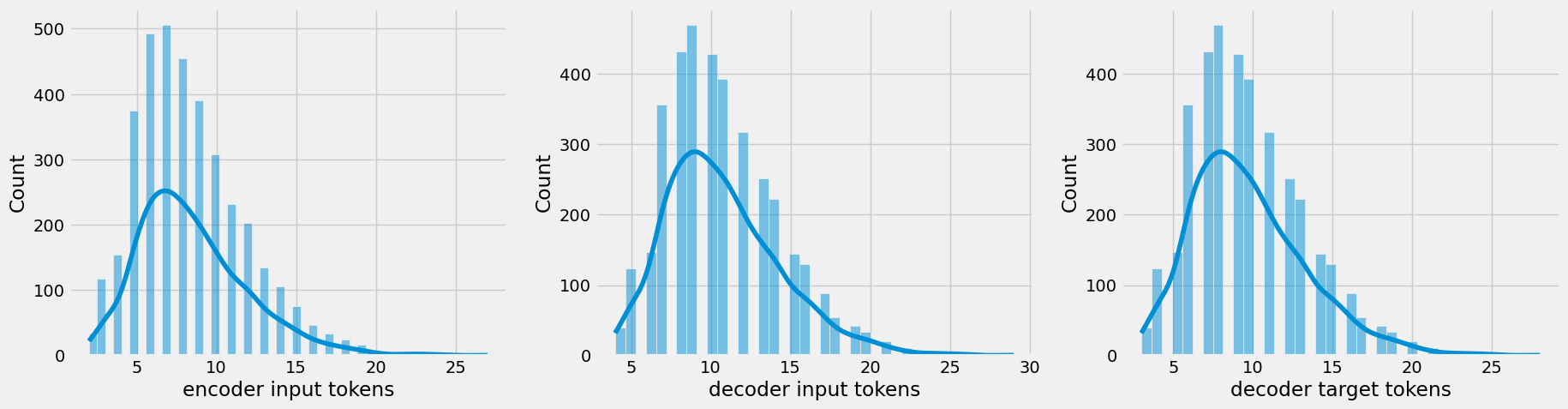
sns.histplot(x=df['encoder input tokens'],data=df,kde=True,ax=ax[0])

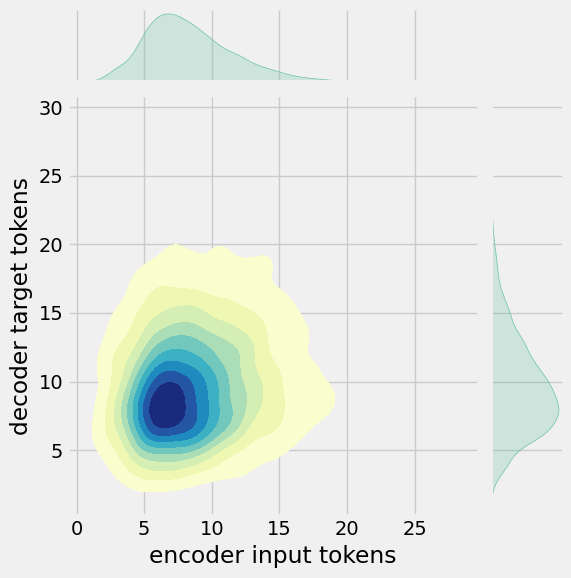
sns.histplot(x=df['decoder input tokens'],data=df,kde=True,ax=ax[1])

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()





**CONCLUSION**

In this project, we have introduced a chatbot that is able to interact with users. This chatbot can answer queries in the textual user input. For this purpose, AIML with program-o has been used. The chatbot can answer only those questions which he has the answer in its AIML dataset. In this project, we have introduced a chatbot that is able to interact with users. This chatbot can answer queries in the textual user input. For this purpose, AIML with program-o has been used. The chatbot can answer only those questions which he has the answer in its AIML dataset.