CREATE A CHATBOT IN PYTHON

BATCH MEMBER

ZENITA DROSTON 963321104065

Phase 3 submission document

Project Title: Create a Chatbot in python

Phase 3: Development part 1

Topic: Start Creating a Chatbot in python by loading and preprocessing the dataset.

Create a Chatbot in python

Introduction:

Creating a chatbot in Python involves several steps and considerations. Here's an introduction to the process, summarized in three key points:

1. Understanding the Basics: To create a chatbot in Python, it's essential to have a solid understanding of natural language processing (NLP) concepts and techniques. NLP involves processing and understanding human language, allowing the chatbot to interact with users effectively. Key concepts include tokenization, part-of-speech tagging, named entity recognition, and sentiment analysis. Python provides a range of libraries, such as NLTK (Natural Language Toolkit) and spaCy, which can be used to implement these NLP functionalities.

2. Choosing a Framework: There are various frameworks available in Python that simplify the process of building a chatbot. One popular option is the ChatterBot library, which provides a straightforward way to implement conversational agents. ChatterBot allows you to train a chatbot using a large dataset of conversations and provides functionalities for generating appropriate responses. Other options like Rasa and Dialogflow also offer rich features for building chatbots with advanced capabilities like natural language understanding, intent recognition, and context management.

3. Implementing the Chatbot: Once you have chosen the framework, it's time to implement the chatbot. This involves defining the chatbot's responses based on the user's input, handling various conversational scenarios, and integrating the chatbot into your desired platform or interface. You can train the chatbot using existing datasets or create your own dataset.

Given Data Set:

	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.
5	i've been good. i'm in school right now.	what school do you go to?
6	what school do you go to?	i go to pcc.
7	i go to pcc.	do you like it there?
8	do you like it there?	it's okay. it's a really big campus.
9	it's okay. it's a really big campus.	good luck with school.

Necessary Steps 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
```

2.Load the dataset:

Load your dataset into a Pandas DataFrame. You can typically find Create a Chatbot datasets in CSV format, but you can adapt this code to other formats as needed.

Program:

```
data = pd.read_csv(r"/kaggle/input/deepnlp/Sheet_1.csv",encoding="latin1")
data.drop(["Unnamed: 3","Unnamed: 4","Unnamed: 5",
        "Unnamed: 6","Unnamed: 7",],axis = 1,inplace =True)
data = pd.concat([data["class"],data["response_text"]],axis = 1)

data.dropna(axis=0,inplace =True)
data.head(10)
```

3.Exploratory data analysis:

Exploratory Data Analysis (EDA) is a crucial step in data analysis that involves the initial investigation of a dataset to understand its structure, identify patterns, and uncover any anomalies or insights. By employing various statistical techniques and visualization tools, EDA helps to summarize the main characteristics of the data, detect outliers, and gain a preliminary understanding of the relationships between variables. It assists in making informed decisions about data preprocessing, feature selection, and model building. EDA plays a vital role in uncovering hidden patterns and trends that can ultimately guide further data exploration and decision-making processes.

Creating a chatbot in Python involves leveraging Natural Language Processing (NLP) techniques and libraries such as NLTK or spaCy. The process typically includes defining the chatbot's behavior, designing an intuitive user interface, and implementing an algorithm that understands and generates responses. Initial steps may involve data preprocessing, such as tokenization, stemming, and stop-word removal. Next, suitable machine learning algorithms, such as sequence-to-sequence models or rule-based approaches, can be employed to train the chatbot on a corpus of

data. The final step involves deploying the chatbot and continuously improving its performance based on user feedback and incremental updates.

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()
```

4.Feature Engineering:

"Feature Engineering in creating a chatbot in Python involves transforming raw data into meaningful features to improve the performance and accuracy of the chatbot's understanding and response generation capabilities."

Program:

```
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)
```

5.Split the data:

To create a chatbot in Python, utilize libraries like NLTK or Chatterbot for natural language processing and build a conversational interface using frameworks like Flask or Django.

Program:

```
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()
```

6.Feature Scaling:

Feature scaling is a technique used to normalize the range of features in a chatbot's dataset in Python.

Program:

```
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)
```

Importance of loading and preprocessing dataset:

Loading and preprocessing the dataset is of utmost importance in creating a chatbot in Python. The dataset serves as the foundation for training the chatbot and enabling it to generate meaningful responses. Properly loading the dataset ensures that all the necessary information is available for the chatbot to learn from. Additionally, preprocessing the dataset involves cleaning, formatting, and transforming the data into a suitable format for training. This step is crucial as it enhances the quality and relevance of the dataset, allowing the chatbot to better understand and respond to user queries. By paying attention to loading and preprocessing the dataset, we can significantly improve the accuracy and effectiveness of the chatbot, providing a more satisfying user experience.

Challenges involved in loading and preprocessing create a chatbot dataset:

The challenges involved in creating a chatbot in Python, specifically related to loading and preprocessing the dataset. Here are some common challenges you might encounter:

1. Data Gathering: One of the first challenges is to gather a suitable dataset for training your chatbot. You need to ensure that the dataset is relevant, diverse, and covers a wide range of possible user inputs and responses.

2. Data Formatting: Once you have the dataset, you need to format it in a way that is suitable for training your chatbot. This may involve converting the data into the appropriate file format (such as JSON or CSV) and structuring it in a way that the chatbot can understand.

3. Data Cleaning: The quality of your dataset is crucial. You may need to clean the dataset to remove any irrelevant or duplicate entries, correct spelling or grammatical errors, and handle missing or inconsistent data.

4. Data Preprocessing: Preprocessing the dataset is essential to transform the raw data into a suitable format for training the chatbot. This could involve tasks such as tokenization, stemming, lemmatization, or removing stop words. Preprocessing helps in reducing noise and improving the efficiency of the chatbot.

5. Training Data Size: The size of the dataset can also pose a challenge. Having a smaller dataset may lead to overfitting, where the chatbot performs well on the training data but struggles with real-world inputs.

How to overcome the challenges involved in loading and preprocessing create a Chatbot dataset:

To overcome the challenges of loading and preprocessing a chat bot dataset, you can follow these steps:

1. Data Collection: Begin by gathering a diverse and representative dataset for training your chat bot. You can collect data from various sources, such as customer interactions, chat logs, or publicly available datasets.

2. Data Cleaning: Clean the dataset by removing any irrelevant or noisy data. This may include removing duplicates, correcting spelling errors, or removing irrelevant information.

3. Tokenization: Convert the text data into tokens, which are smaller units of text such as words or characters. This step helps in breaking down the text into manageable chunks for further processing.

4. Text Normalization: Normalize the text by converting it to a standard format. This includes tasks like converting all text to lowercase, removing punctuation, or handling special characters.

5. Stopword Removal: Remove common words that do not add much meaning to the text, such as articles, prepositions, and pronouns. This helps in reducing the noise in the dataset.

6. Lemmatization or Stemming: Reduce words to their root form using techniques like lemmatization or stemming. This helps in reducing the vocabulary size and improving the model's performance.

7. Handling Out-of-Vocabulary Words: Identify and handle words that are not present in the training vocabulary. You can either replace them with a special token or use techniques like subword encoding to handle such cases.

Program:

DATA PREPROCESSING:

In [1]:

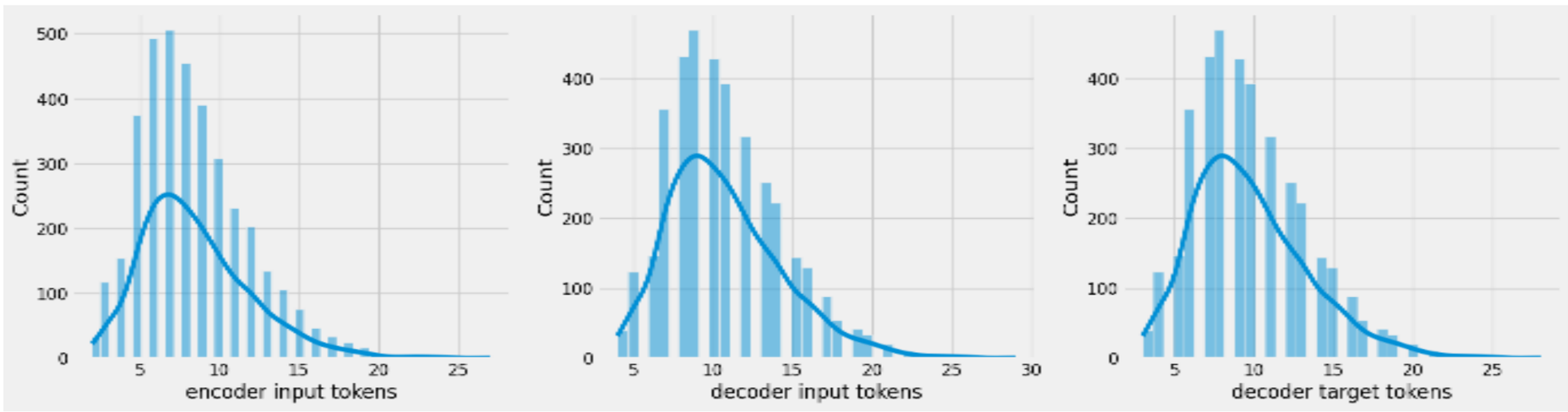
```
from tensorflow.keras.layers import TextVectorization
import re,string
from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization
#data preprocessing
df=pd.read_csv('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt',sep='\t',names=['question','answer'])
print(f'Dataframe size: {len(df)}')
df.head()
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()
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)
```

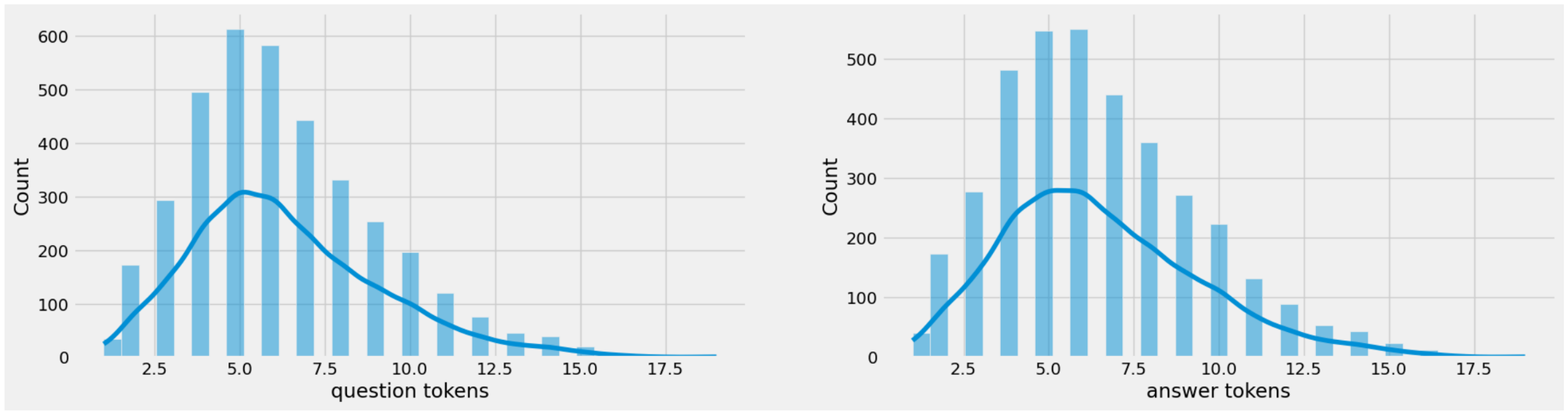
```
    text=re.sub('[3]',' 3',text)
    text=re.sub('[4]',' 4',text)
    text=re.sub('[5]',' 5',text)
    return text
df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'
df.head(10)
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,}
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)
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>')
```

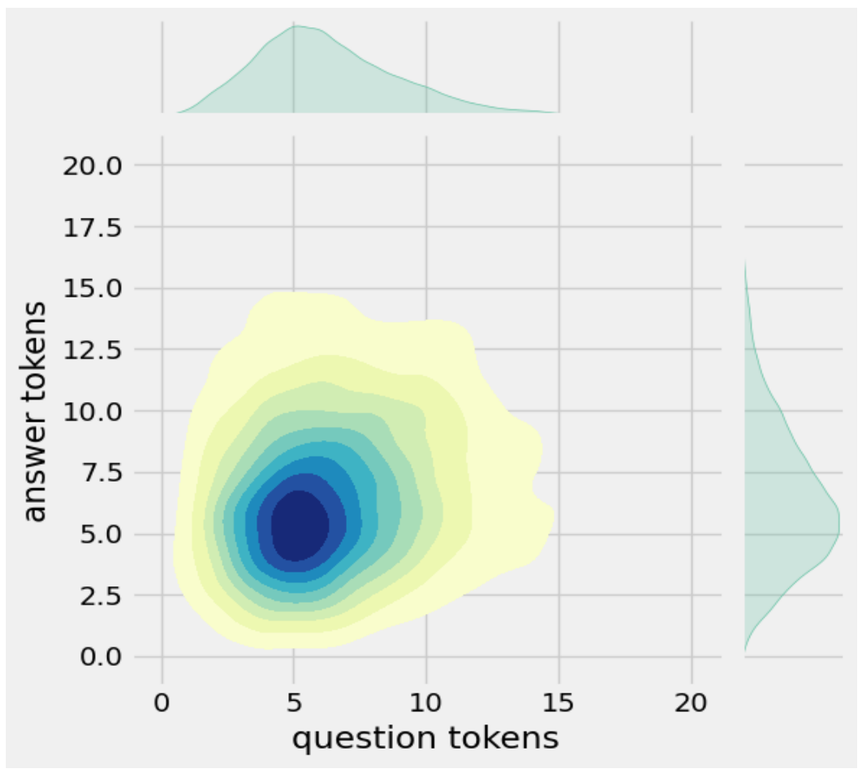
```
vocab_size=len(vectorize_layer.get_vocabulary())
print(f'Vocab size: {len(vectorize_layer.get_vocabulary())}')
print(f'{vectorize_layer.get_vocabulary()[:12]}')
def sequences2ids(sequence):
    return vectorize_layer(sequence)
def ids2sequences(ids):
    decode=''
    if type(ids)==int:
        ids=[ids]
    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}')
```

Out [1]:

```
hi, how are you doing?  i'm fine. how about yourself?
i'm fine. how about yourself?  i'm pretty good. thanks for asking.
i'm pretty good. thanks for asking.  no problem. so how have you been?
no problem. so how have you been?  i've been great. what about you?
i've been great. what about you?  i've been good. i'm in school right now.
i've been good. i'm in school right now.  what school do you go to?
what school do you go to?  i go to pcc.
i go to pcc.  do you like it there?
do you like it there?  it's okay. it's a really big campus.
```







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
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)

BUILD MODELS:

In [2]:

```
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()
```

```
        )
encoder=Encoder(lstm_cells,embedding_dim,vocab_size,name='encoder')
encoder.call(_[0])
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()
        )
decoder=Decoder(lstm_cells,embedding_dim,vocab_size,name='decoder')
decoder(_[1][:1],encoder(_[0][:1]))
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']
)
    callbacks=[
        tf.keras.callbacks.TensorBoard(log_dir='logs'),
        tf.keras.callbacks.ModelCheckpoint('ckpt',verbose=1,save_best_only=True)
    ]
)
```

Out [2]:

```
(<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],
<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,
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, ...,
```

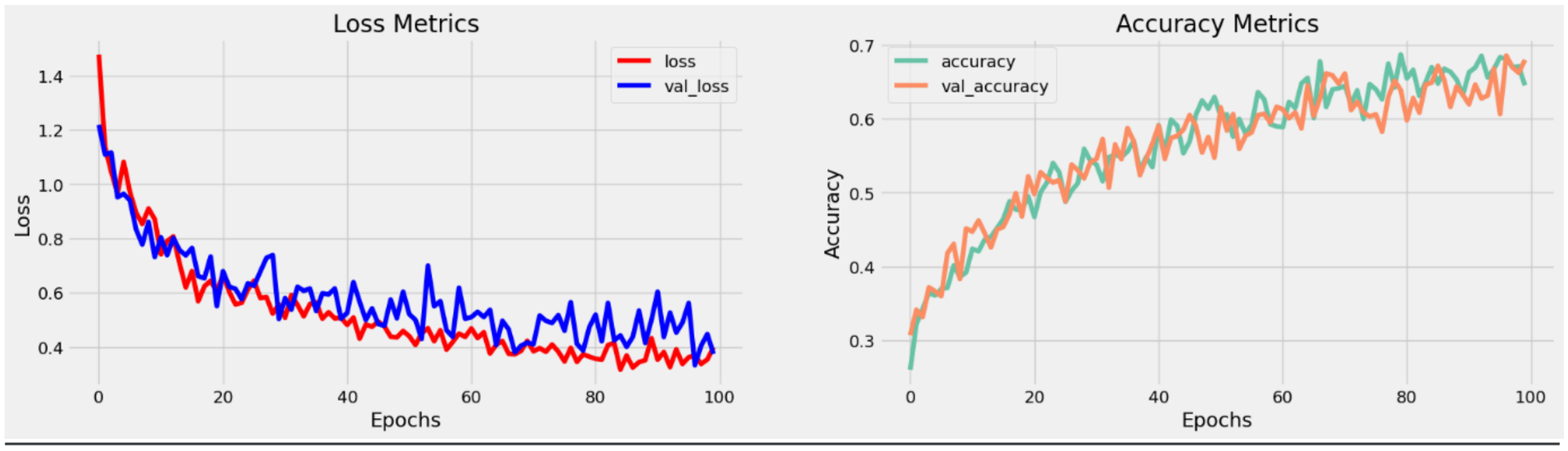
```
        1.91874733e-05, 9.72440175e-05, 7.64339056e-05],
       [9.69291723e-05, 2.74417835e-05, 1.37613132e-03, ...,
        3.60095728e-05, 1.55378671e-04, 1.83973272e-04],
       ...,
Epoch 1/100
```

VISUALIZE METRICS:

In [3]:

```
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
ax[0].plot(history.history['loss'],label='loss',c='red')
ax[0].plot(history.history['val_loss'],label='val_loss',c = 'blue')
ax[0].set_xlabel('Epochs')
ax[1].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[1].set_ylabel('Accuracy')
ax[0].set_title('Loss Metrics')
ax[1].set_title('Accuracy Metrics')
ax[1].plot(history.history['accuracy'],label='accuracy')
ax[1].plot(history.history['val_accuracy'],label='val_accuracy')
ax[0].legend()
ax[1].legend()
plt.show()
```

Out [3]:



Some common preprocessing tasks include:

1. **Data cleaning:** This involves handling missing values, outliers, and inconsistent data to ensure high data quality.

2. **Data integration:** Combining data from multiple sources into a single dataset, ensuring consistency and compatibility.

3. **Data transformation:** Modifying the format or structure of the data, such as scaling numerical features or encoding categorical variables.

4. Feature selection: Identifying and selecting the most relevant features from the dataset to improve model performance and reduce dimensionality.

5. Feature engineering: Creating new features from existing ones to capture valuable information and improve model performance.

6. Data normalization: Scaling the numerical data to a standardized range, such as using methods like min-max scaling or z-score normalization.

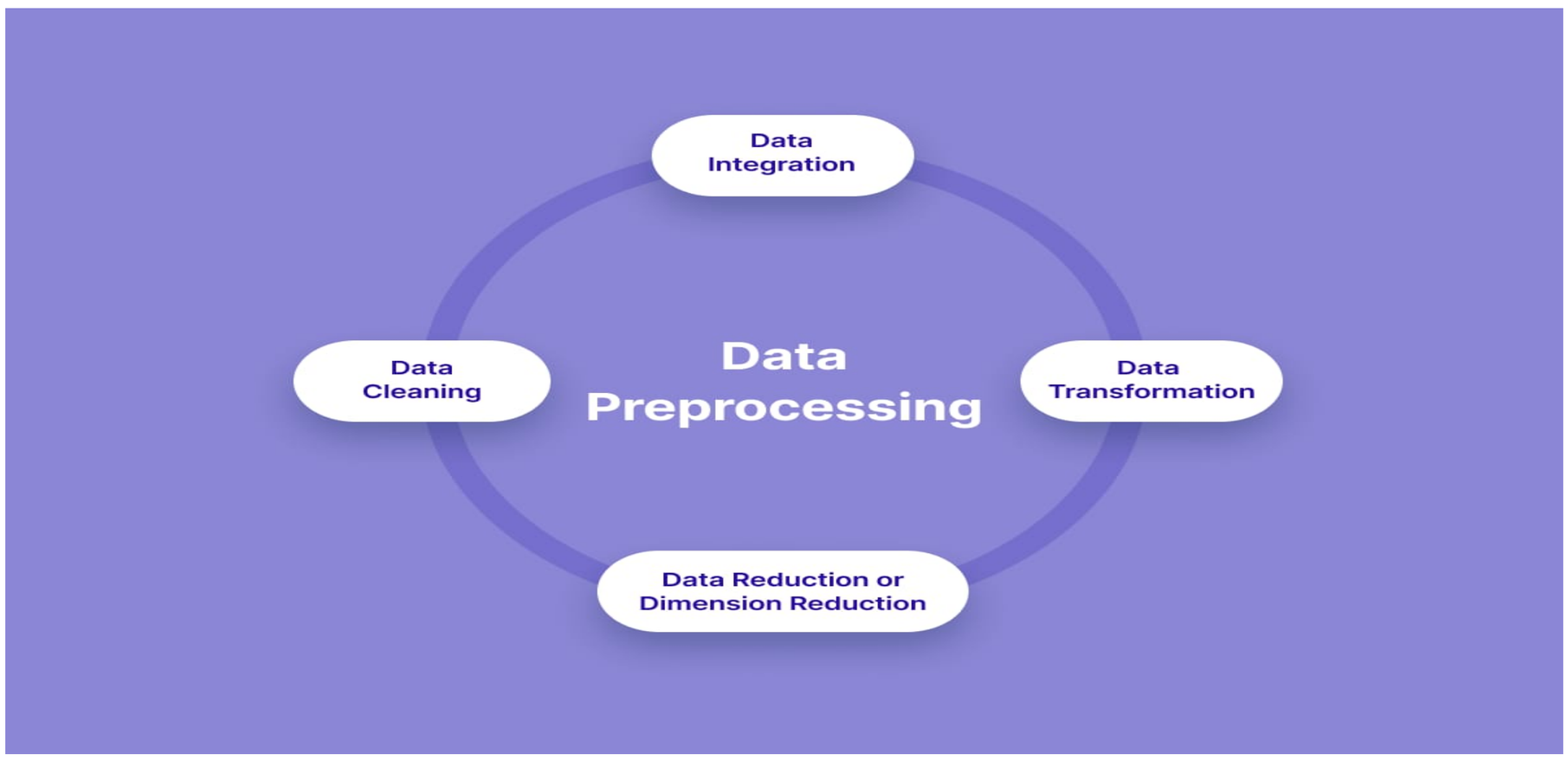
7. Handling imbalanced data: Addressing the issue of imbalanced class distribution by oversampling, under sampling, or using techniques like SMOTE (Synthetic Minority Over-sampling Technique).

8. Handling categorical data: Converting categorical variables into numerical representations through techniques like one-hot encoding or label encoding.

9. Removing duplicates: Identifying and removing duplicate records to ensure data integrity and avoid bias in analysis.

10. Data discretization: Grouping continuous data into intervals or categories to simplify analysis and improve interpretability.

These are just a few examples of common data preprocessing tasks. The specific tasks may vary depending on the nature of the dataset and the goals of the analysis or model development.



SAVE MODEL:

In [4]:

```
model.load_weights('ckpt')
model.save('models',save_format='tf')
```

linkcode

```
for idx,i in enumerate(model.layers):
    print('Encoder layers:' if idx==0 else 'Decoder layers:')
```

```
for j in i.layers:
    print(j)
print('----------')
```

Out [4]:

```
Encoder layers:
<keras.layers.core.embedding.Embedding object at 0x782084b9d190>
>
----------
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>
---------
```

CREATE INFERENCE MODEL:

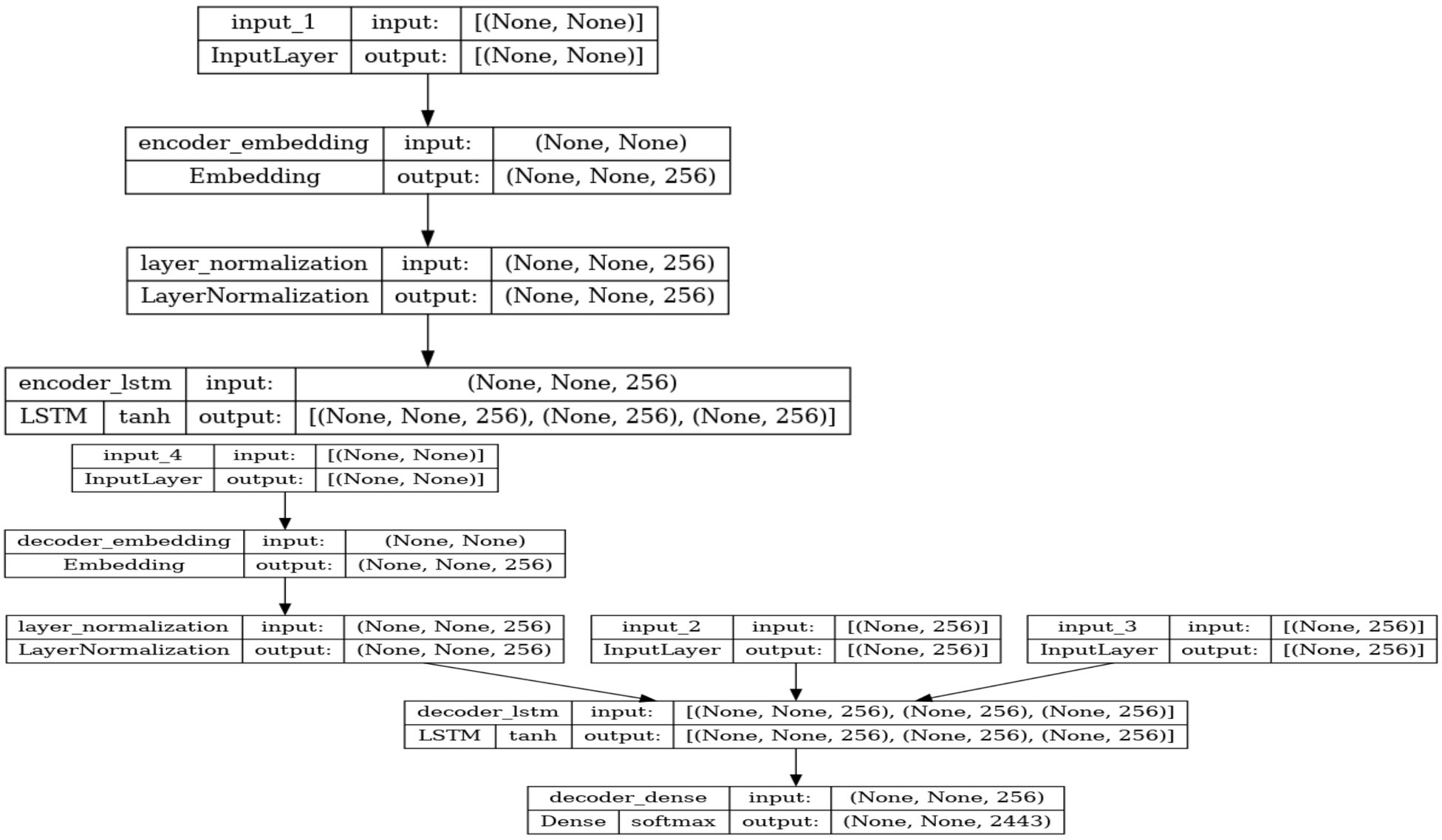
In [5]:

```
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,encoder_st
ate_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_stat
e_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
chatbot.summary()
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,show_layer_activations=True)
tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',show_shapes=True,show_layer_activations=True)
```

Out [5]:

```
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
_________________________________________________________________
```



Time to Chat:

In [6]:

```
def print_conversation(texts):
    for text in texts:
        print(f'You: {text}')
        print(f'Bot: {chatbot(text)}')
        print('==========================')
print_conversation([
    'hi',
    'do yo know me?',
    'what is your name?',
    'you are bot?',
    'hi, how are you doing?',
    "i'm pretty good. thanks for asking.",
    "Don't ever be in a hurry",
    "'I'm gonna put some dirt in your eye '",
    "'You're trash '",
    "'I've read all your research on nano-technology '",
    "'You want forgiveness? Get religion'",
    "'While you're using the bathroom, i'll order some food.'",
    "'Wow! that's terrible.'",
    "'We'll be here forever.'",
    "'I need something that's reliable.'",
    "'A speeding car ran a red light, killing the girl.'",
    "'Tomorrow we'll have rice and fish for lunch.'",
    "'I like this restaurant because they give you free bread.'"])
```

Out [6]:

You: hi
Bot: i have to go to the bathroom.
==========
You: do yo know me?
Bot: yes, it's too close to the other.
==========
You: what is your name?
Bot: i have to walk the house.
==========
You: you are bot?
Bot: no, i have. all my life.
==========
You: hi, how are you doing?
Bot: i'm going to be a teacher.
==========
You: i'm pretty good. thanks for asking.

Conclusion:

Developing a chatbot in Python for Phase 3 project submission is an excellent choice. Python offers a wide range of libraries and frameworks that make it easy to implement natural language processing and machine learning algorithms, which are essential for building an effective chatbot.

To create a chatbot in Python, you will need to follow a systematic approach. Firstly, you need to define the scope and purpose of your chatbot. This includes understanding the target audience, the type of interactions it should handle, and the specific functionality it should offer.

Next, you will need to design the conversational flow and user interface. This involves creating a dialog management system that can handle user inputs, generate appropriate responses, and maintain context throughout the conversation. Python libraries such as NLTK or spaCy can be used for natural language processing and understanding.

To train your chatbot, you will need a dataset of labeled conversations. This can be obtained through manual labeling or by using pre-existing chat logs. Machine learning algorithms, such as deep learning models like recurrent neural networks (RNNs) or transformer models like GPT-2, can be used for training your chatbot to generate accurate and contextually relevant responses.

Once your chatbot is trained, you can deploy it on various platforms such as web applications, messaging services, or even custom hardware devices. Python frameworks like Flask or Django can be used for web development, while libraries like PyTorch or TensorFlow can be used for model deployment.