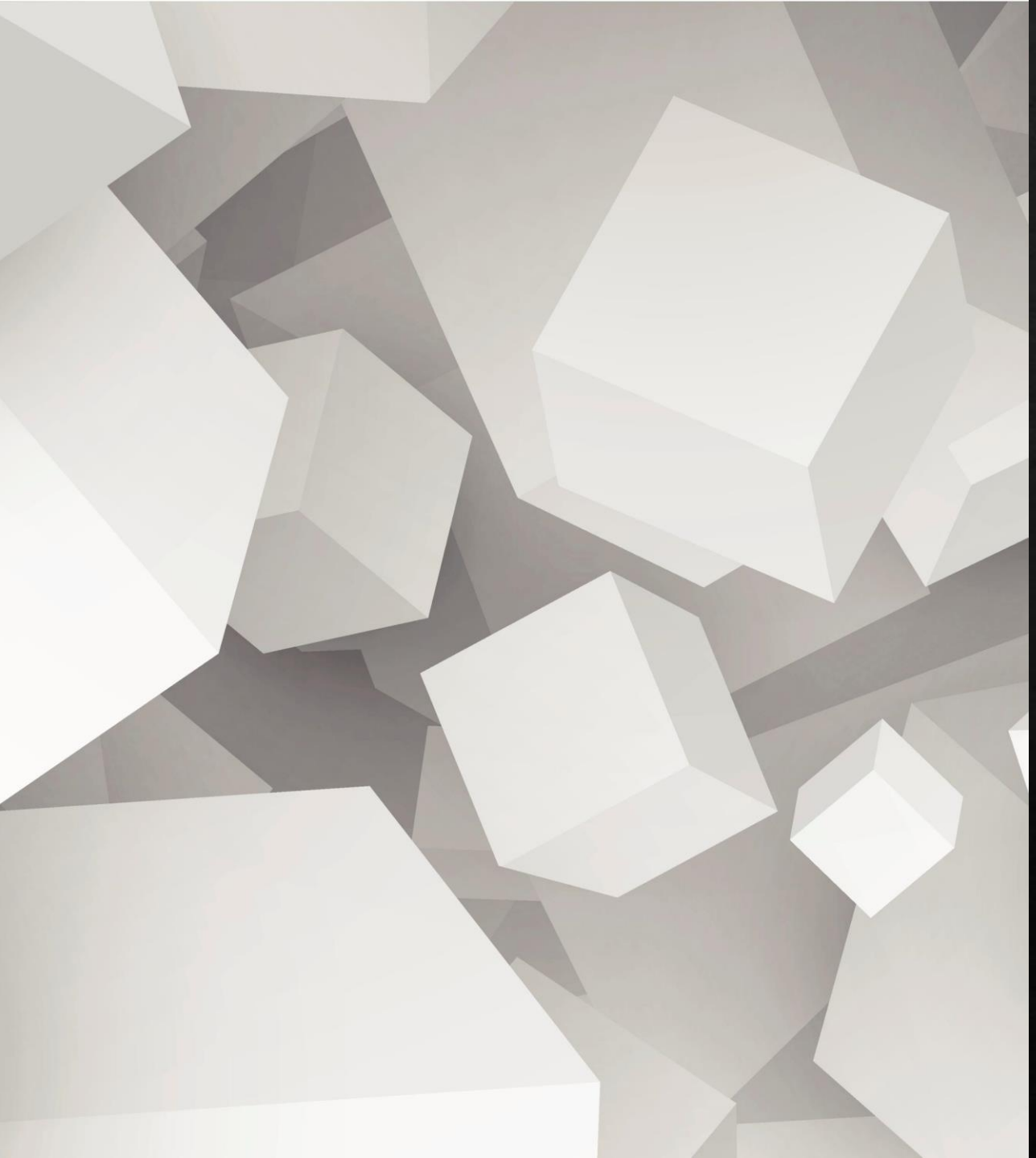
ARTIFICIAL INTELLIGENT

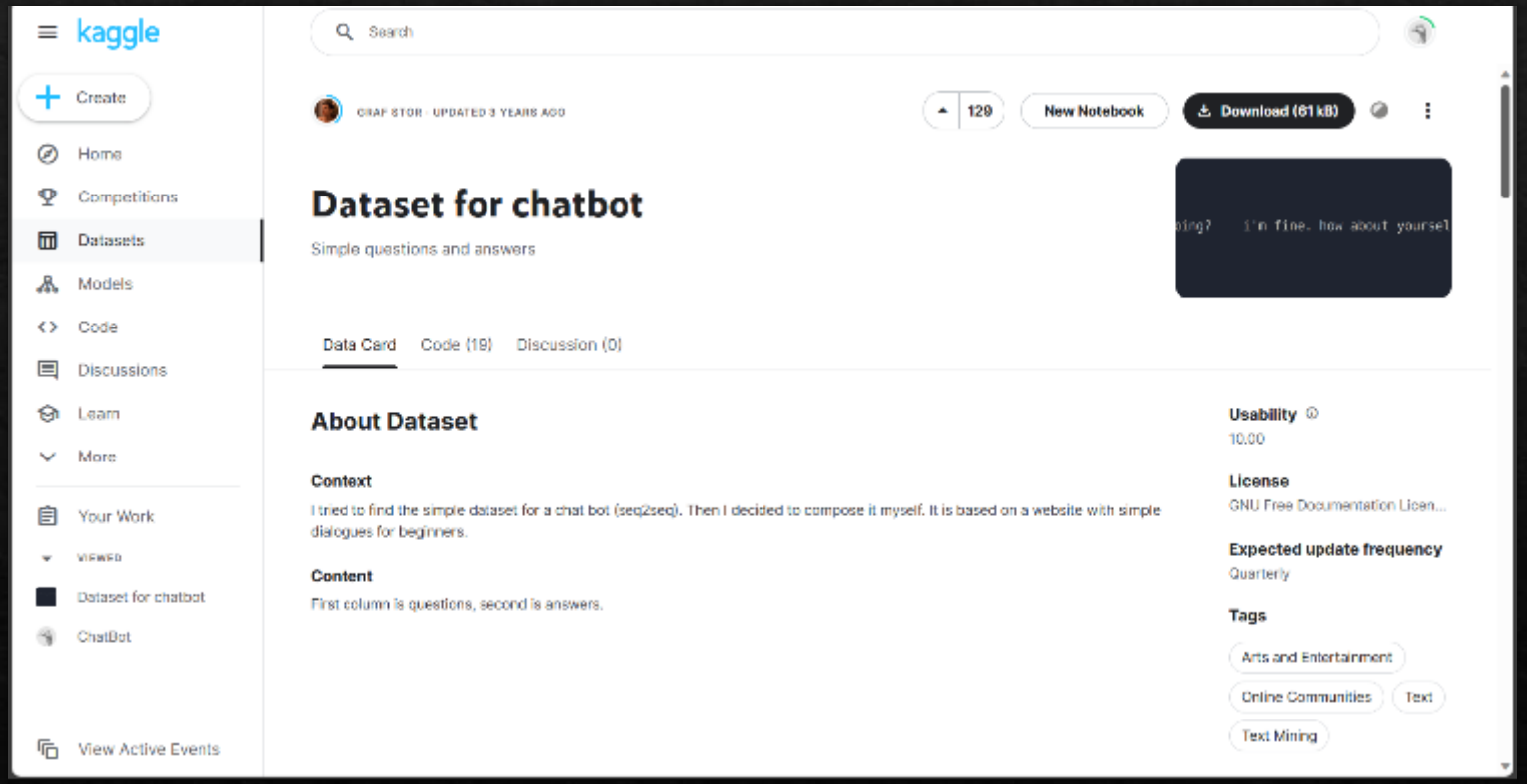
PHASE 3



INTRODUCTION:

- Chatbot often powered by sophisticated language models like GPT, have gained popularity in natural language processing tasks. These models can generate human-like responses based on input text and are trained on vast amounts of diverse textual data.
- To effectively train such models, a crucial step is the preprocessing of the dataset. In this Python script, we outline a systematic approach to prepare a dataset for training a chatbot.

DATASET DOWNLOAD FROM KAGGLE:



Data Preprocessing:

Data preprocessing is a crucial step in the data mining and data analysis process that involves transforming raw data into a format that can be understood and analyzed by computers and machine learning algorithms.

The Basic Preprocessing:

- Sentence Segmentation
- Normalization
- Tokenization

IMPOTING 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
```

Sentence segmentation:

- Sentence segmentation is the analysis of texts based on sentences. In NLP analysis, we either analyze the text data based on meaningful words which is tokens or we analyze them based on sentences

READING THE DATASET:

```
[ ]:
df=pd.read_csv('C:\Users\mathu\Downloads\archive.zip.txt',sep='\t',names=['question','answer'])
print(f'Dataframe size: {len(df)}')
df.head()
```

OUTPUT:

```
Dataframe size: 3725
```

Out[2]:

	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.

PARIED LIST OF QUESTIONS AND CORRESPONDING ANSWER:

In [3]:

```
#paried list of question and corresponding answer
QA_list=[QA.split('\t') for QA in data.split('\n')]
print(QA_list[:5])
```

```
[['hi, how are you doing?', "i'm fine. how about yourself?"], ["i'm fine. how about yoursel
f?", "i'm pretty good. thanks for asking."], ["i'm pretty good. thanks for asking.", 'no pro
blem. 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 no
w."]]
```

In [4]:

```
questions=[row[0] for row in QA_list]
answers=[row[1] for row in QA_list]
```

In [5]:

```
print(questions[0:5])
print(answers[0:5])
```

```
['hi, how are you doing?', "i'm fine. how about yourself?", "i'm pretty good. thanks for ask
ing.", 'no problem. so how have you been?', "i've been great. what about you?"]
["i'm fine. how about yourself?", "i'm pretty good. thanks for asking.", 'no problem. so how
have you been?', "i've been great. what about you?", "i've been good. i'm in school right no
w."]
```

Normalization:

- Text preprocessing is an important part of Natural Language Processing (NLP), and *normalization* of text is one step of preprocessing.
- The goal of normalizing text is to *group related tokens together*, where tokens are usually the words in the text.
- Depending on the text you are working with and the type of analysis you are doing, you might not need all of the normalization techniques in this post.

In [6]:

```
def remove_diacritic(text):
    return ''.join(char for char in unicodedata.normalize('NFD',text)
                   if unicodedata.category(char) !='Mn')
```

Ensuring punctuation marks to be treated as tokens:

```
#Ensuring punctuation marks to be treated as tokens
text=re.sub(r"([?.!,¿])", r" \1 ", text)
```

Removing redundant spaces:

```
#Removing redundant spaces

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

Removing non alphabetic charters:

```
#Removing non alphabetic characters

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

INDICATING THE STRAT AND END OF EACH SENTENCE:

```
    text=text.strip()

    #Indicating the start and end of each sentence
    text='<start> ' + text + ' <end>'

    return text
```

In [8]:

```
preprocessed_questions=[preprocessing(sen) for sen in questions]
preprocessed_answers=[preprocessing(sen) for sen in answers]

print(preprocessed_questions[0])
print(preprocessed_answers[0])
```

```
<start> hi , how are you doing ? <end>
<start> i m fine . how about yourself ? <end>
```

TOKENIZATION:

Tokenization, when applied to data security, is the process of substituting a sensitive data element with a non-sensitive equivalent, referred to as a token, that has no intrinsic or exploitable meaning or value. The token is a reference (i.e. identifier) that maps back to the sensitive data through a tokenization system.

TOKENIZEING:

[7]:

```
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]}')
```

```
Vocab size: 2443
['', '[UNK]', '<end>', '.', '<start>', "'", 'i', '?', 'you', ',', 'the', 'to']
```

In [8]:

```
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}')
```

```
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)
```

In [9]:

```
print(f'Encoder input: {x[0][:12]} ...')
print(f'Decoder input: {yd[0][:12]} ...')    # shifted by one time step of the target as input t
o decoder is the output of the previous timestep
print(f'Decoder target: {y[0][:12]} ...')
```

```
Encoder input: [1971    9   45   24    8  194    7    0    0    0    0    0] ...
Decoder input: [   4    6    5   38 646    3   45   41 563    7    2    0] ...
Decoder target: [   6    5   38 646    3   45   41 563    7    2    0    0] ...
```

In [10]:

```
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)
```

TOKENIZING:

In [9]:

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

Build Vocabulary on Unique Word:

```
#build vocabulary on unique words
lang_tokenizer.fit_on_texts(lang)


return lang_tokenizer
```

Word Embeddings:

It is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meanings to have a similar representation. They can also approximate meaning. A word vector with 50 values can represent 50 unique features.

Goal of Word Embeddings:

- To reduce dimensionality
- To use a word to predict the words around it
- Interword semantics must be captured

Word Embedding:

In [10]:

```
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
```

LOADING DATASET:

Datasets are loaded from a dataset loading script that downloads and generates the dataset. However, you can also load a dataset from any dataset repository on the Hub without a loading script! Begin by creating a dataset repository and upload your data files. Now you can use the load_dataset () function to load the dataset.

LOADING DATASET:

In [11]:

```
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
```

In [12]:

```
size=30000
data=preprocessed_answers,preprocessed_questions\


X_tensor,X_tokenizer, y_tensor, y_tokenizer=load_Dataset(data,size)
```

In [13]:

```
# Calculate max_length of the target tensors
max_length_y, max_length_X = y_tensor.shape[1], X_tensor.shape[1]
```

SPLITTING DATA:

Splitting Data for Machine Learning Models Courses Splitting facts for system mastering models is an crucial step within the version improvement process. It includes dividing the to be had dataset into separate subsets for education, validation, and trying out the version.

SPLITTING DATA:

In [14]:

```
X_train, X_val, y_train, y_val = train_test_split(X_tensor, y_tensor, test_size=0.2)
```

Show length:

```
# Show length
print(len(X_train), len(y_train), len(X_val), len(y_val))
```

```
2980 2980 745 745
```

TENSORFLOW DATA:

In [15]:

```
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
```

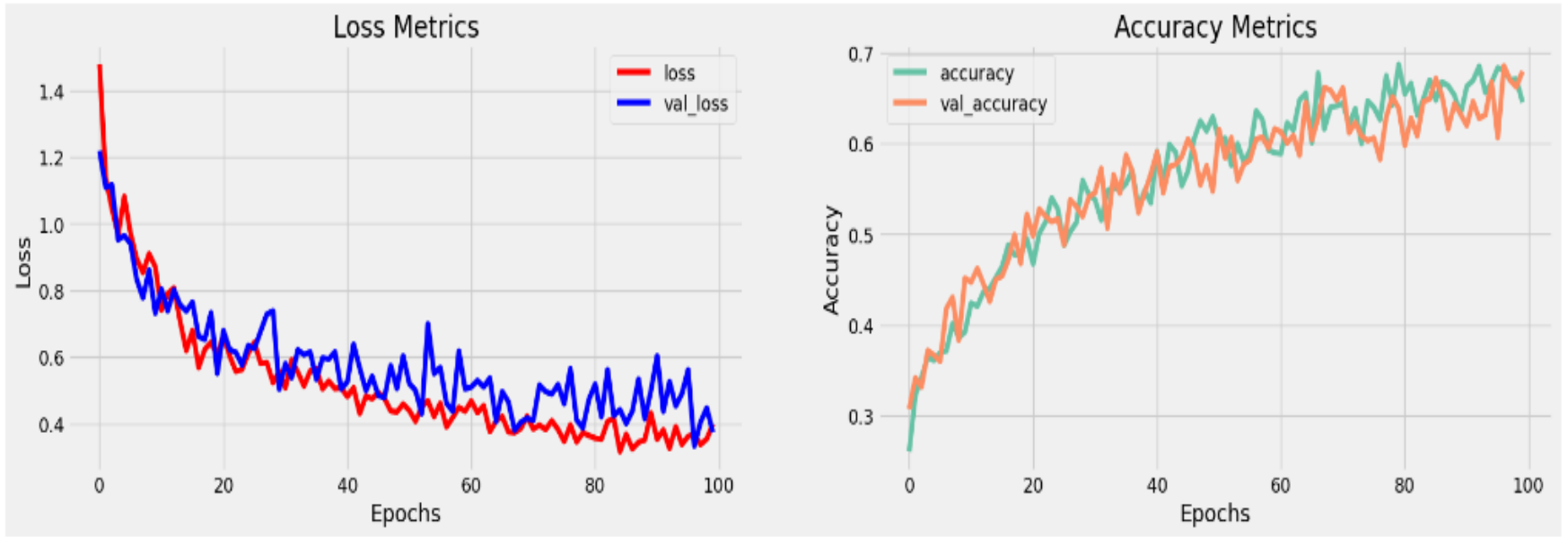
Out[15]:

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

In [16]:

```
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()
```

OUTPUT:


Loss Metrics
loss
val_loss
Loss
Epochs
1.4
1.2
1.0
0.8
0.6
0.4
0
20
40
60
80
100
Accuracy Metrics
accuracy
val_accuracy
Accuracy
Epochs
0.7
0.6
0.5
0.4
0.3

TEXT CLEAN:

[4]:

```
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('[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>'
```

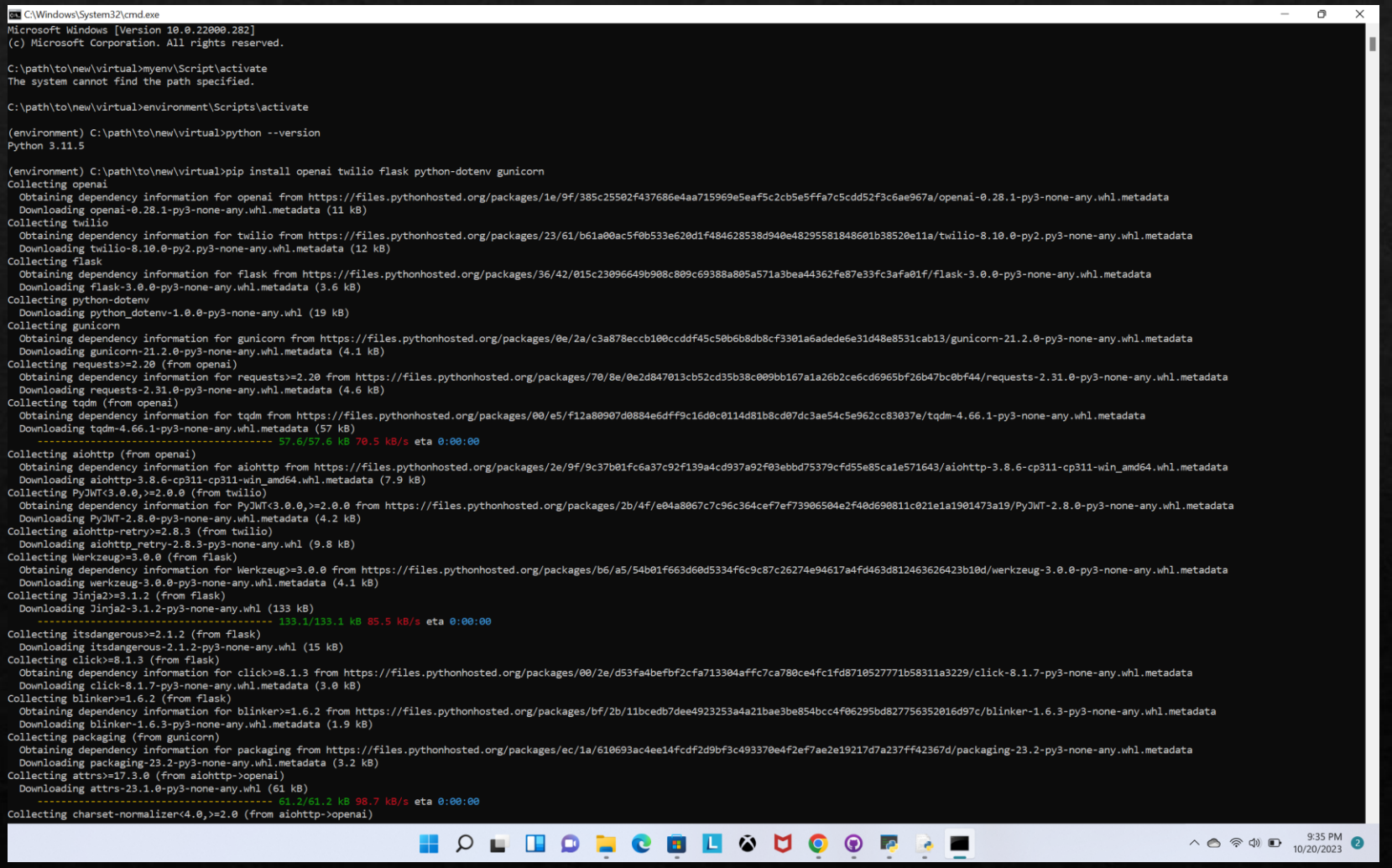
OUTPUT:

```
df.head(10)
```

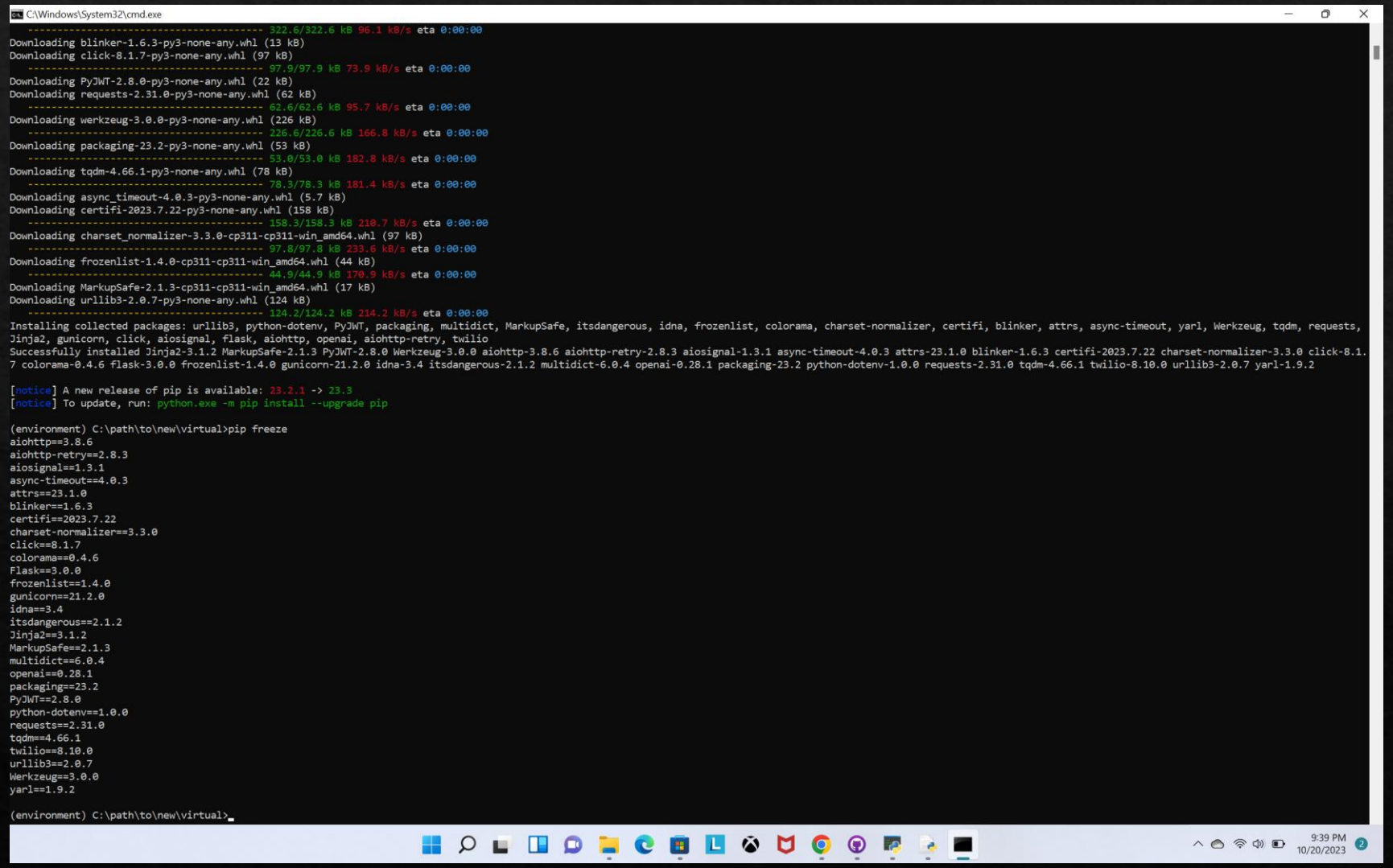
ut[4]:

	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>

CREATING VIRTUAL ENVIRONMENT :



DOWNLOAD THE REQUIRED LIBRARIES (FLASK,OPENAI):



CONCLUTION:

Preprocessing is an essential step in chatbot development. It involves cleaning and normalizing the data, removing irrelevant information, and tokenizing the text. Preprocessing helps optimize inputs and outputs for better results . Once data is collected for training a chatbot, it's important to pre-process it to ensure it's clean and ready for use. And creation of virtual environment is an most important part of building chatbot,after cration install the required libraries like OpenAI,Flask.

hereby,I conclude that those content are not copied and done on my own.