



A black rectangular box with a thin green border is positioned in the upper right quadrant of the image. Inside this box, the words "ARTIFICIAL" and "INTELLIGENT" are stacked vertically in a large, serif, white font. Below this box, the word "PHASE" is followed by the number "3" in a larger, bold, serif, white font.

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:

The screenshot shows a Kaggle dataset page titled "Dataset for chatbot". The left sidebar includes links for Create, Home, Competitions, Datasets (selected), Models, Code, Discussions, Learn, and More. Under "Your Work", there are entries for "Dataset for chatbot" and "ChatBot". At the bottom of the sidebar are "View Active Events" and "View My Profile". The main content area features a search bar, a user profile for GRAF STOR (updated 3 years ago), and a "Download (61 kB)" button. The dataset title is "Dataset for chatbot" with a subtitle "Simple questions and answers". Below the title are Data Card, Code (19), and Discussion (0) buttons. The "About Dataset" section contains "Context" (describing it as a seq2seq dataset for a chat bot), "Content" (describing the structure as questions and answers), and "Usability" (rating 10.00). It also lists License (GNU Free Documentation License), Expected update frequency (Quarterly), and Tags (Arts and Entertainment, Online Communities, Text, Text Mining).

Dataset for chatbot

Simple questions and answers

Data Card    Code (19)    Discussion (0)

About Dataset

Context

I tried to find the simple dataset for a chat bot (seq2seq). Then I decided to compose it myself. It is based on a website with simple dialogues for beginners.

Content

First column is questions, second is answers.

Usability 10.00

License

GNU Free Documentation License

Expected update frequency

Quarterly

Tags

Arts and Entertainment

Online Communities

Text

Text Mining

# 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.

# PAIRIED LIST OF QUESTIONS AND CORRESPONDING ANSWER:

In [3]:

```
#paried list of question and correspoding 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 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 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 asking.", '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 now."]
```

# 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]:

# 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'\s+', ' ', 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.

# TOKENIZING:

```
[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 to 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] ...  
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]:

# 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
```

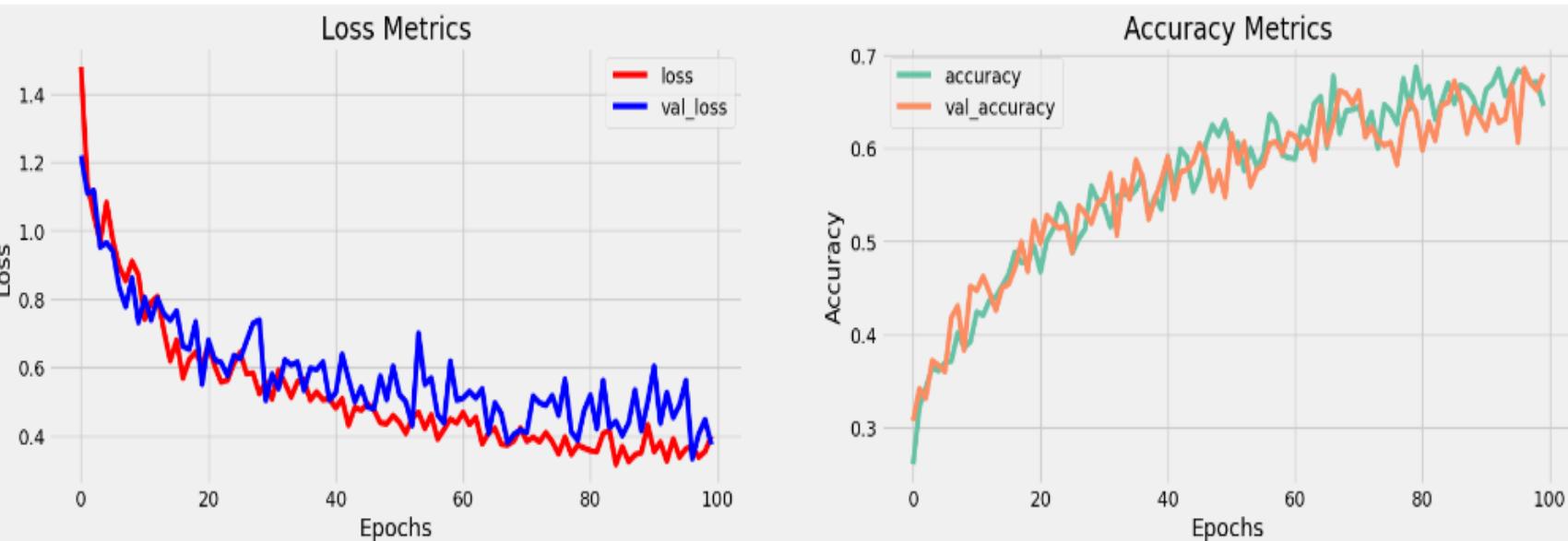
# TENSORFLOW DATA:

```
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:



# 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('[\d]','0',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>'
```

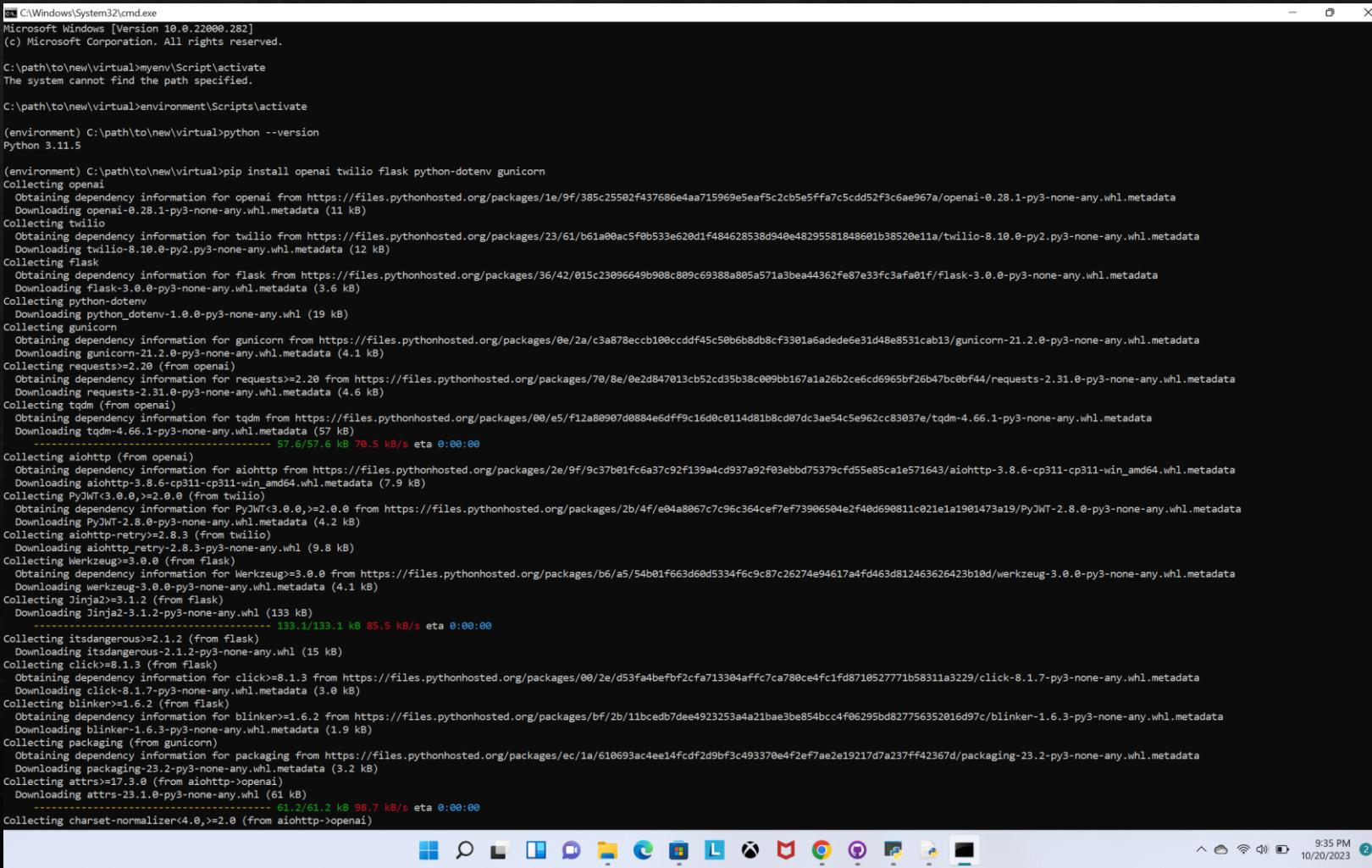
# OUTPUT:

```
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 :



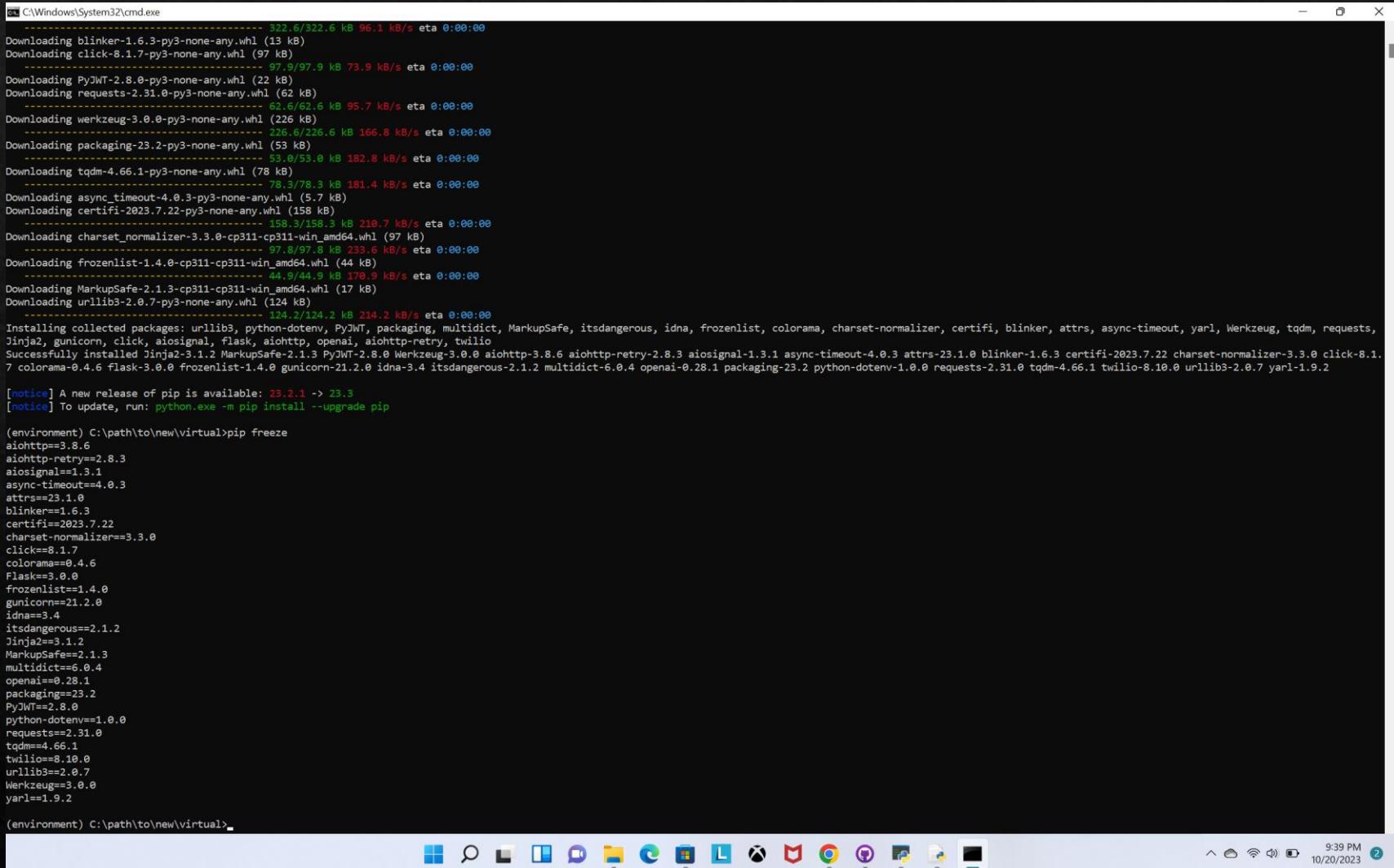
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.282]
(c) Microsoft Corporation. All rights reserved.

C:\path\to\new\virtual>myenv\Script\activate
The system cannot find the path specified.

C:\path\to\new\virtual>python --version
Python 3.11.5

(environment) C:\path\to\new\virtual>pip install openai twilio flask python-dotenv gunicorn
Collecting openai
  Obtaining dependency information for openai from https://files.pythonhosted.org/packages/1e/9f/385c25502f437686e4aa715969e5eaf5c2cb5e5ffa7c5cdd52f3c6ae967a/openai-0.28.1-py3-none-any.whl.metadata
    Downloading openai-0.28.1-py3-none-any.whl.metadata (11 kB)
Collecting twilio
  Obtaining dependency information for twilio from https://files.pythonhosted.org/packages/23/61/b61a00ac5f0b533e620d1f484628538d940e48295581848601b38520e11a/twilio-8.10.0-py2.py3-none-any.whl.metadata
    Downloading twilio-8.10.0-py2.py3-none-any.whl.metadata (12 kB)
Collecting flask
  Obtaining dependency information for flask from https://files.pythonhosted.org/packages/36/42/015c23096649b908c809c69388a805a571a3bea44362fe87e3fc3afa01f/flask-3.0.0-py3-none-any.whl.metadata
    Downloading flask-3.0.0-py3-none-any.whl.metadata (3.6 kB)
Collecting python-dotenv
  Downloading python_dotenv-1.0.0-py3-none-any.whl (19 kB)
Collecting gunicorn
  Obtaining dependency information for gunicorn from https://files.pythonhosted.org/packages/0e/2a/c3a878eccb100ccddf45c50b6b8db8cf3301a6adeda6e31d48e8531cab13/gunicorn-21.2.0-py3-none-any.whl.metadata
    Downloading gunicorn-21.2.0-py3-none-any.whl.metadata (4.1 kB)
Collecting requests>=2.20 (from openai)
  Obtaining dependency information for requests>=2.20 from https://files.pythonhosted.org/packages/70/8e/0e2d847013cb52cd35b38c009bb167a1a26b2ce6cd6965bf26b47bc0bf44/requests-2.31.0-py3-none-any.whl.metadata
    Downloading requests-2.31.0-py3-none-any.whl.metadata (4.6 kB)
Collecting tqdm (from openai)
  Obtaining dependency information for tqdm from https://files.pythonhosted.org/packages/00/e5/f12a80907d0884e6dff9c16d0c0114d81b8cd07dc3ae54c5e962cc83037e/tqdm-4.66.1-py3-none-any.whl.metadata
    Downloading tqdm-4.66.1-py3-none-any.whl.metadata (57 kB)
      57.6/57.6 kB 70.5 kB/s eta 0:00:00
Collecting aiohttp (from openai)
  Obtaining dependency information for aiohttp from https://files.pythonhosted.org/packages/2e/9f/9c37b01fc6a37c92f139a4cd937a92f03ebbd75379cf55e85ca1e571643/aiohttp-3.8.6-cp311-cp311-win_amd64.whl.metadata
    Downloading aiohttp-3.8.6-cp311-cp311-win_amd64.whl.metadata (7.9 kB)
Collecting PyJWT<3.0.0,>=2.0.0 (from twilio)
  Obtaining dependency information for PyJWT<3.0.0,>=2.0.0 from https://files.pythonhosted.org/packages/2b/4f/e04a8067c7c96c364cef7ef73906504e2f40d690811c021e1a1901473a19/PyJWT-2.8.0-py3-none-any.whl.metadata
    Downloading PyJWT-2.8.0-py3-none-any.whl.metadata (4.2 kB)
Collecting aiohttp_retry>=2.8.3 (from twilio)
  Downloading aiohttp_retry-2.8.3-py3-none-any.whl (9.8 kB)
Collecting Werkzeug>=3.0.0 (from flask)
  Obtaining dependency information for Werkzeug>=3.0.0 from https://files.pythonhosted.org/packages/b6/a5/54b01f663d60d5334f6c9c87c26274e94617a4fd463d812463626423b10d/werkzeug-3.0.0-py3-none-any.whl.metadata
    Downloading werkzeug-3.0.0-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
      133.1/133.1 kB 85.5 kB/s eta 0:00:00
Collecting itsdangerous>=2.1.2 (from flask)
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.1.3 (from flask)
  Obtaining dependency information for click>=8.1.3 from https://files.pythonhosted.org/packages/00/2e/d53fa4befbf2cfa713304affc7ca780ce4fc1fd8710527771b58311a3229(click-8.1.7-py3-none-any.whl.metadata
    Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Obtaining dependency information for blinker>=1.6.2 from https://files.pythonhosted.org/packages/bf/2b/11bcdcb7dee4923253a4a21bae3be854bcc4f06295bd827756352016d97c/blinker-1.6.3-py3-none-any.whl.metadata
    Downloading blinker-1.6.3-py3-none-any.whl.metadata (1.9 kB)
Collecting packaging (from gunicorn)
  Obtaining dependency information for packaging from https://files.pythonhosted.org/packages/ec/1a/610693ac4ee14fcfd2d9bf3c493370e4f2ef7ae2e19217d7a237ff42367d/packaging-23.2-py3-none-any.whl.metadata
    Downloading packaging-23.2-py3-none-any.whl.metadata (3.2 kB)
Collecting attrs>=17.3.0 (from aiohttp>openai)
  Downloading attrs-23.1.0-py3-none-any.whl (61 kB)
      61.2/61.2 kB 98.7 kB/s eta 0:00:00
Collecting charset-normalizer<4.0,>=2.0 (from aiohttp>openai)
```

# DOWNLOAD THE REQUIRED LIBRARIES (FLASK,OPENAI):



```
C:\Windows\System32\cmd.exe
  322.6/322.6 kB 96.1 kB/s eta 0:00:00
  Downloading blinker-1.6.3-py3-none-any.whl (13 kB)
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
    97.9/97.9 kB 73.9 kB/s eta 0:00:00
  Downloading PyJWT-2.8.0-py3-none-any.whl (22 kB)
  Downloading requests-2.31.0-py3-none-any.whl (62 kB)
    62.6/62.6 kB 95.7 kB/s eta 0:00:00
  Downloading werkzeug-3.0.0-py3-none-any.whl (226 kB)
    226.6/226.6 kB 166.8 kB/s eta 0:00:00
  Downloading packaging-23.2-py3-none-any.whl (53 kB)
    53.0/53.0 kB 182.8 kB/s eta 0:00:00
  Downloading tqdm-4.66.1-py3-none-any.whl (78 kB)
    78.3/78.3 kB 181.4 kB/s eta 0:00:00
  Downloading async_timeout-4.0.3-py3-none-any.whl (5.7 kB)
  Downloading certifi-2023.7.22-py3-none-any.whl (158 kB)
    158.3/158.3 kB 210.7 kB/s eta 0:00:00
  Downloading charset_normalizer-3.3.0-cp311-cp311-win_amd64.whl (97 kB)
    97.8/97.8 kB 233.6 kB/s eta 0:00:00
  Downloading frozenlist-1.4.0-cp311-cp311-win_amd64.whl (44 kB)
    44.9/44.9 kB 170.9 kB/s eta 0:00:00
  Downloading MarkupSafe-2.1.3-cp311-cp311-win_amd64.whl (17 kB)
  Downloading urllib3-2.0.7-py3-none-any.whl (124 kB)
    124.2/124.2 kB 214.2 kB/s eta 0:00:00
  Installing collected packages: urllib3, python-dotenv, PyJWT, packaging, multidict, MarkupSafe, itsdangerous, idna, frozenlist, colorama, charset-normalizer, certifi, blinker, attrs, async-timeout, yarl, Werkzeug, tqdm, requests, Jinja2, gunicorn, click, aiosignal, flask, aiowebsocket, openai, aiohttp-retry, twilio
  Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.3 PyJWT-2.8.0 Werkzeug-3.0.0 aiowebsocket-3.8.6 aiosignal-1.3.1 async-timeout-4.0.3 attrs-23.1.0 blinker-1.6.3 certifi-2023.7.22 charset-normalizer-3.3.0 click-8.1.7 colorama-0.4.6 flask-3.0.0 frozenlist-1.4.0 gunicorn-21.2.0 idna-3.4 itsdangerous-2.1.2 multidict-6.0.4 openai-0.28.1 packaging-23.2 python-dotenv-1.0.0 requests-2.31.0 tqdm-4.66.1 twilio-8.10.0 urllib3-2.0.7 yarl-1.9.2
[notice] A new release of pip is available: 23.2.1 -> 23.3
[notice] To update, run: python.exe -m pip install --upgrade pip

(environment) C:\path\to\new\virtual>pip freeze
aiowebsocket==3.8.6
aiowebsocket-retry==2.8.3
aiosignal==1.3.1
async-timeout==4.0.3
attrs==23.1.0
blinker==1.6.3
certifi==2023.7.22
charset-normalizer==3.3.0
click==8.1.7
colorama==0.4.6
Flask==3.0.0
frozenlist==1.4.0
gunicorn==21.2.0
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.3
multidict==6.0.4
openai==0.28.1
packaging==23.2
PyJWT==2.8.0
python-dotenv==1.0.0
requests==2.31.0
tqdm==4.66.1
twilio==8.10.0
urllib3==2.0.7
Werkzeug==3.0.0
yarl==1.9.2

(environment) C:\path\to\new\virtual>
```

# **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.**