

CREATE A CHATBOT USING PYTHON

TEAM MEMBER

311121205057 - SWETHA S

Project : Create a chatbot using python

Phase 3: Development Part 1

In this part you will begin building your project by loading and preprocessing the dataset.

Start building the chatbot by preparing the environment and implementing basic user interactions. Install required libraries, like transformers for GPT-3 integration and flask for web app development.

TABLE OF CONTENT

- ☐ Introduction
- ☐ Import Libraries
- ☐ Loading the dataset
- ☐ Preprocessing the Data
- ☐ Setting up Environment
- ☐ Basic User Interaction
- ☐ Conclusion

INTRODUCTION

Chatbots have emerged as powerful tools in the world of technology, transforming the way we interact with businesses, services, and even each other. They are computer programs designed to simulate human conversation, providing real-time responses to text or voice inputs. Whether for customer support, information retrieval, or entertainment, chatbots have become an integral part of many online platforms and applications.

Creating a chatbot is an exciting venture that combines elements of artificial intelligence, natural language processing, and user experience design. The primary goal of a chatbot is to engage users in meaningful, efficient, and often personalized conversations, ultimately providing value, assistance, or entertainment.

To load libraries for working with datasets in Python, you typically use a combination of popular libraries, such as NumPy, Pandas, and Matplotlib. Here's a brief description of how to load these libraries and what they are commonly used for:

NumPy (Numerical Python):

Description: NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, as well as a variety of high-level mathematical functions to operate on these arrays.

Importing NumPy:

```
python
```

```
import numpy as np
```

Pandas:

Description: Pandas is a powerful data manipulation library that provides data structures for efficiently storing and analyzing data, particularly in the form of DataFrames. DataFrames are two-dimensional, tabular data structures with labeled axes, which makes it easy to perform data cleaning, exploration, and analysis.

Importing Pandas:

```
python
```

```
import pandas as pd
```

Matplotlib:

Description: Matplotlib is a popular data visualization library in Python. It provides a wide range of functions for creating static, animated, or interactive plots and charts. It is especially useful for visualizing the data contained in Pandas DataFrames.

Importing Matplotlib:

```
python
```

```
import matplotlib.pyplot as plt
```

Here's an example of how you can load these libraries together for working with datasets:

```
python
```

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

Once you've imported these libraries, you can use them to read, manipulate, analyze, and visualize datasets. NumPy and Pandas are particularly essential for data manipulation, while Matplotlib is useful for data visualization. Depending on the specific dataset and task, you might also need to import additional libraries for machine learning, statistical analysis, or domain-specific tasks.

```
`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
```

LOAD THE DATASET

"Loading the dataset" refers to the process of importing a specific collection of data into a software application or program for further analysis, manipulation, or use. This dataset can include various types of information such as text, numbers, images, or any other structured or unstructured data. Loading the dataset typically involves reading data from a file, a database, or an external source, and making it accessible within the software for tasks like data analysis, machine learning, or statistical processing. This step is crucial for any data-driven project or analysis, as it sets the foundation for deriving insights or making informed decisions based on the available information.

```
file_name = 'dialogs.txt'

# Read the uploaded CSV file into a DataFrame
df = pd.read_csv(file_name, sep='\t', names=['question', 'answer'])
print(f'Dataframe size: {len(df)}')
df.head()
```

 Dataframe size: 3725

	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.

PREPROCESSING

Dataset preprocessing is a crucial step in data analysis, machine learning, and data mining. It involves a series of operations and transformations performed on the raw data to make it suitable for further analysis or modeling. The primary objectives of dataset preprocessing are to improve data quality, address issues, and ensure that the data is in a format that is compatible with the algorithms or methods you intend to apply. Some common preprocessing tasks include:

Data Cleaning: This step involves identifying and handling missing values, outliers, and errors in the dataset. It may include techniques like imputation for missing data and removing or correcting outliers.

Data Transformation: Data often needs to be transformed to meet the requirements of specific algorithms or to make it more interpretable. Common transformations include normalization, standardization, and encoding categorical variables.

Feature Selection: Sometimes, datasets contain many irrelevant or redundant features. Feature selection techniques are used to identify and retain only the most relevant features, which can help improve model performance and reduce computational complexity.

Feature Engineering: Creating new features from the existing ones to provide more informative inputs to a model. This may involve mathematical operations or domain-specific knowledge to extract meaningful information.

Data Splitting: Datasets are typically divided into training, validation, and test sets to train, tune, and evaluate machine learning models. This splitting ensures that the model's performance is assessed on unseen data.

Data Balancing: In classification tasks, imbalanced datasets can lead to biased models. Balancing techniques, such as oversampling or undersampling, are used to address this issue.

Text Preprocessing: For natural language processing tasks, text data may undergo tokenization, stemming, and removing stop words to prepare it for analysis or modeling.

Dimensionality Reduction: Large datasets with many features can be computationally expensive and may lead to overfitting. Dimensionality reduction techniques like PCA (Principal Component Analysis) are used to reduce the number of features while retaining as much information as possible.

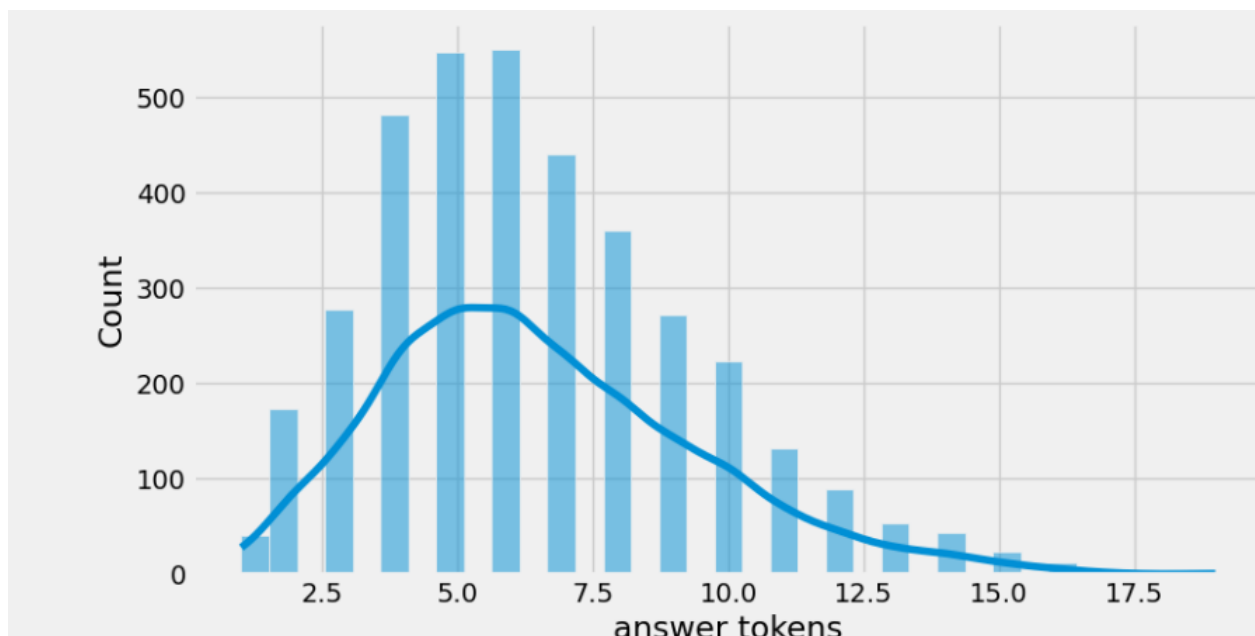
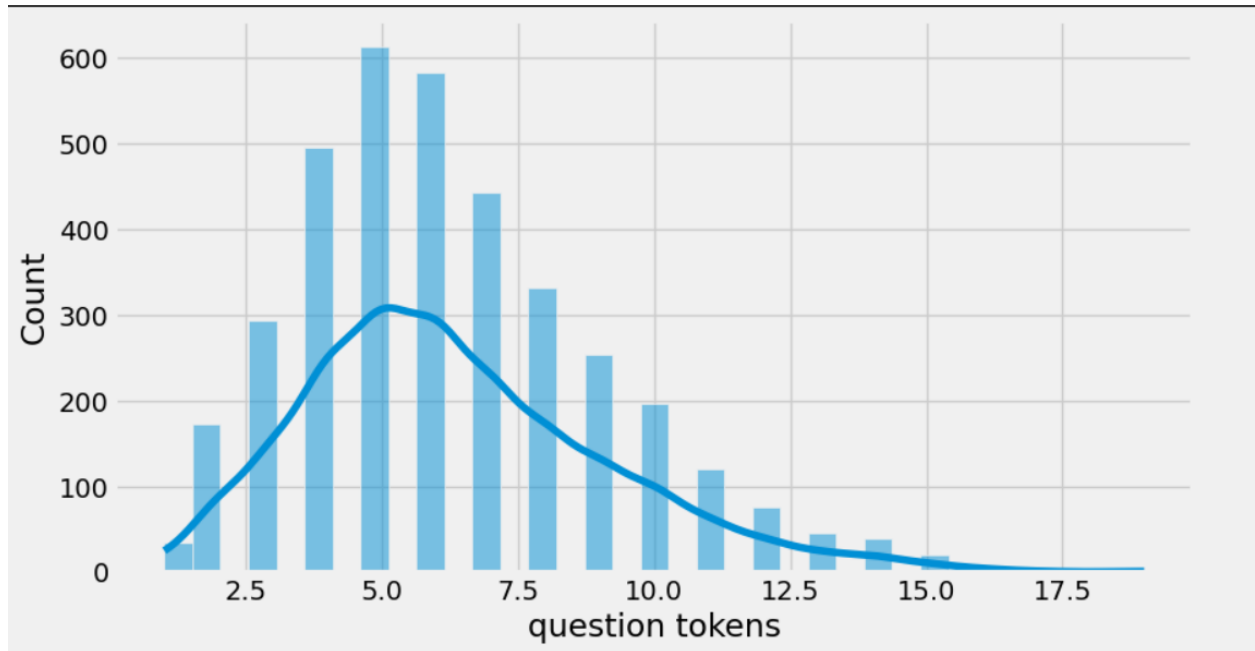
Data Scaling: Scaling numeric features to a common range (e.g., between 0 and 1) helps algorithms that are sensitive to the scale of the data, such as k-means clustering or support vector machines.

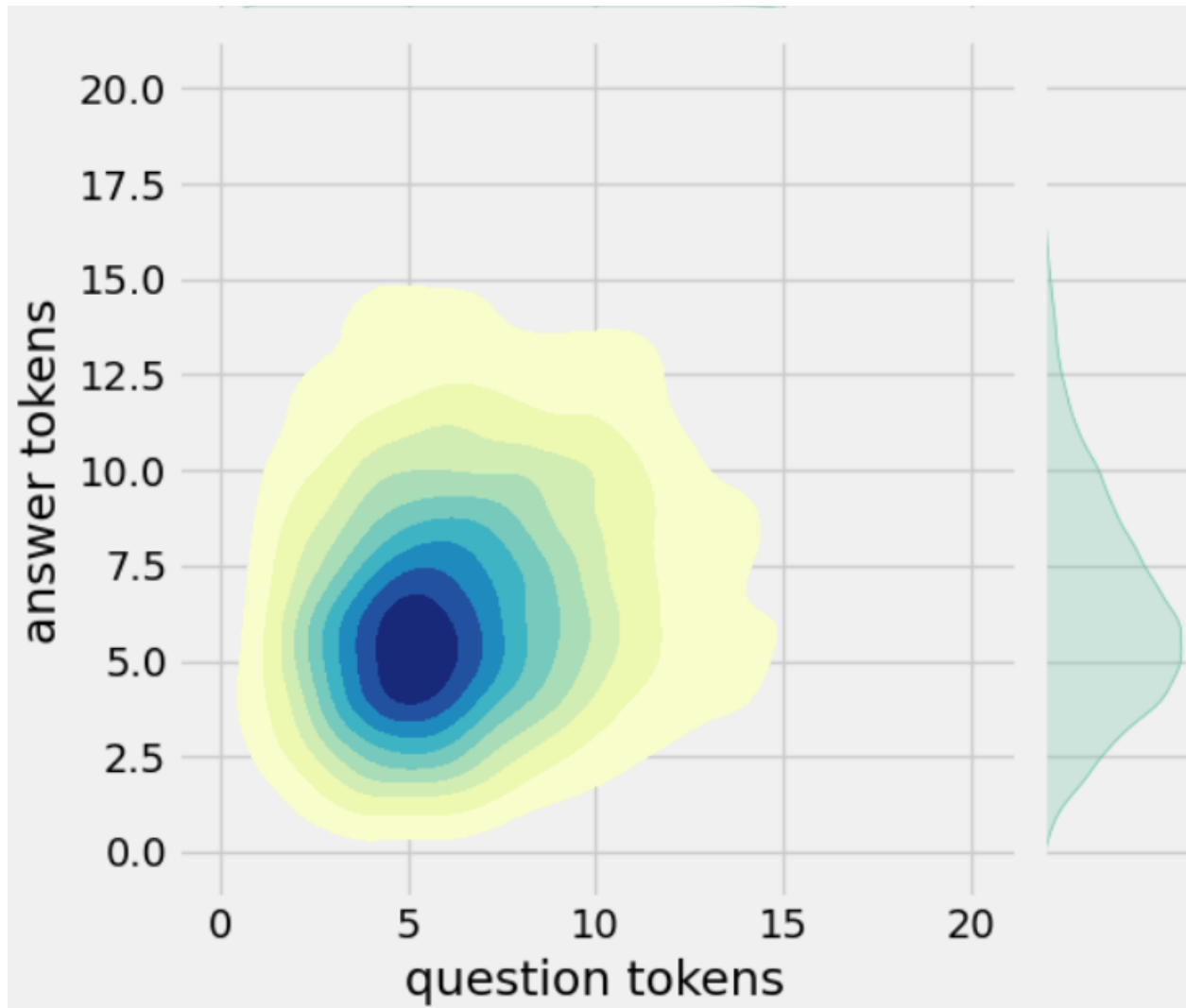
Handling Time Series Data: For time series data, additional preprocessing might include resampling, lag creation, and handling seasonality and trends.

The specific preprocessing steps you need to perform depend on the nature of your data, the goals of your analysis, and the algorithms or models you plan to use. Effective preprocessing can significantly impact the quality and effectiveness of the final analysis or machine learning model.

```
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()
```





```
#Data_cleansing
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)
```

	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>

```

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)

```

SETTING UP ENVIRONMENT

Preparing the environment when building a chatbot involves setting up your development environment to ensure that you have all the tools and dependencies needed for your project. Here's a brief overview of the steps typically involved in preparing the environment for a chatbot development project:

Install Python: If you don't already have Python installed, download and install the latest version of Python from the official website (<https://www.python.org/>). Make sure to add Python to your system's PATH during installation.

Use a Virtual Environment: It's a good practice to create a virtual environment for your chatbot project. Virtual environments help you manage project-specific dependencies and avoid conflicts with other Python projects.

Activate the virtual environment:

On Windows: myenv\Scripts\activate
 On macOS and Linux: source myenv/bin/activate

Install Required Libraries: Depending on your chatbot's requirements, you may need to install various Python libraries. You can use pip, Python's package manager, to install these libraries.

Version Control: Consider using version control software like Git to manage your project's source code. Initialize a Git repository in your project's directory, and start tracking your code changes.

IDE or Text Editor: Choose an integrated development environment (IDE) or text editor that you're comfortable with for writing and debugging your code. Popular choices for Python development include Visual Studio Code, PyCharm, and Sublime Text.

Set Up an API or Service Keys: If your chatbot project involves integrating with external services or APIs, obtain the necessary API keys or credentials and set them up securely in your project.

Database Setup (if applicable): If your chatbot requires a database to store data or user interactions, set up the database system you intend to use (e.g., SQLite, PostgreSQL, MySQL) and configure your project to interact with it.

Create a Project Directory Structure: Organize your project files and directories in a structured manner. For example, you might have separate directories for code, data, and templates (if building a web-based chatbot).

Testing and Debugging Tools: Set up testing frameworks and debugging tools, such as Python's built-in unittest or third-party libraries like pytest. This will help you ensure the functionality and reliability of your chatbot.

Documentation: It's a good practice to create and maintain documentation for your project. Document your code, how to run the chatbot, and any important project-specific information.

Dependency Management: Use a dependency management tool like pip and create a requirements.txt file to list all the project dependencies. This makes it easy to recreate the environment on another machine.

Once you've completed these steps, your development environment should be ready for building your chatbot. Remember to follow best practices for software development, such as version control, testing, and documentation, to ensure a smooth and organized development process.

BASIC USER INTERACTION

```
import unicodedata
import re
```

```
# Modify the path to your dataset
data = open('dialogs.txt', 'r').read()
```

```
# Split the data into question and answer pairs
QA_list = [QA.split("\t") for QA in data.split("\n")]
```

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

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

```
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

# Preprocess your questions and answers
preprocessed_questions = [preprocessing(sen) for sen in questions]
preprocessed_answers = [preprocessing(sen) for sen in answers]

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

```

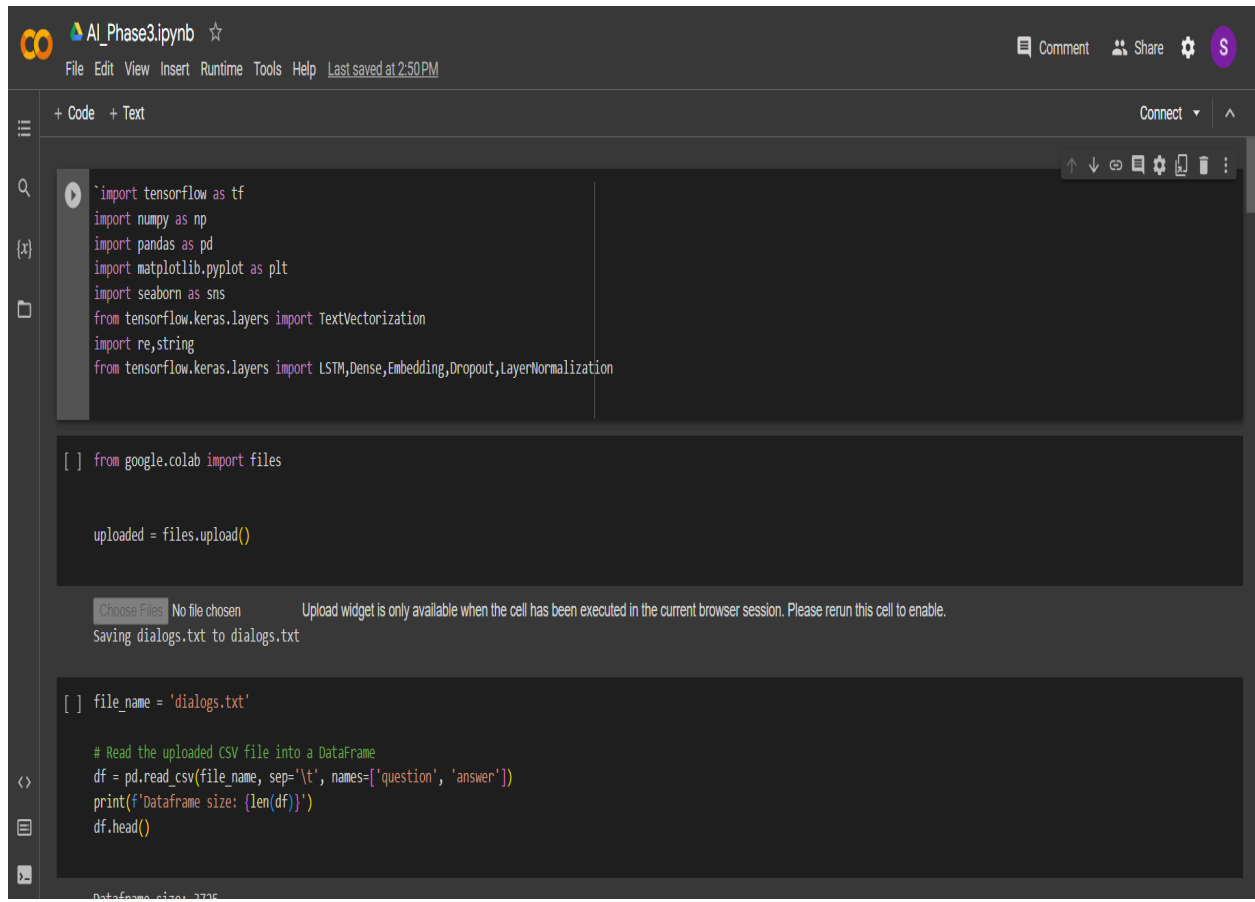
OUTPUT :

```

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

```

SOURCE CODE SNAPSHOTS



The screenshot displays a Jupyter Notebook titled "AI_Phase3.ipynb" with a star icon and a "Last saved at 2:50PM" timestamp. The interface includes a top menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". On the right, there are buttons for "Comment", "Share", a settings gear, and a user profile icon labeled "S". The left sidebar contains icons for a menu, search, variables, and a file explorer. The main area shows three code cells. The first cell contains import statements for tensorflow, numpy, pandas, matplotlib, seaborn, and keras layers. The second cell shows the import of the "files" module from "google.colab" and the execution of "files.upload()". Below this cell is a message: "Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable. Saving dialogs.txt to dialogs.txt". The third cell shows the loading of a CSV file into a DataFrame and printing its size and head. The output of the third cell is partially visible at the bottom: "Dataframe size: 3735".

```
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

[ ] from google.colab import files

uploaded = files.upload()

[ ] file_name = 'dialogs.txt'

# Read the uploaded CSV file into a DataFrame
df = pd.read_csv(file_name, sep='\t', names=['question', 'answer'])
print(f'Dataframe size: {len(df)}')
df.head()
```

Dataframe size: 3735



+ Code + Text

[] DataFrame size: 3725

	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.

pre processing

```
[ ] 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)
```

AI_Phase3.ipynb

File Edit View Insert Runtime Tools Help Last saved at 2:50 PM

Comment Share Settings

+ Code + Text

Connect

```
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)
```

	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.	does pcc have a library?	i go to pcc .	does pcc have a library ? <end>	<start> does pcc have a library ? <end>

AI_Phase3.ipynb

File Edit View Insert Runtime Tools Help Last saved at 2:50 PM

+ Code + Text

```

[ ] 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']

[ ] 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'])

```

AI_Phase3.ipynb

File Edit View Insert Runtime Tools Help Last saved at 2:50 PM

+ Code + Text

```

[ ] 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)

▶ 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  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] ...

▶ 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()

```




+ Code + Text



```
print(f'Decoder input: {yd[0][:12]} ...')    # shifted by one time step of the target
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] ...
```

```
[ ] 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}')
```



AI_Phase3.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 2:50 PM

+ Code + Text

[]

```
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)
```



```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# Download NLTK data (if not already downloaded)
nltk.download('punkt')
nltk.download('stopwords')

# Sample user input
user_input = "Please analyze this text and provide me with a summary."

# Tokenize the user input
tokens = word_tokenize(user_input)

# Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

# Perform stemming
stemmer = PorterStemmer()
```



AI_Phase3.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 2:50 PM

+ Code + Text

```
stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
[ ]
# Display the processed tokens
print("Original User Input:", user_input)
print("Tokenized User Input:", tokens)
print("Processed User Input (without stopwords and stemming):", stemmed_tokens)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Original User Input: Please analyze this text and provide me with a summary.
Tokenized User Input: ['Please', 'analyze', 'this', 'text', 'and', 'provide', 'me', 'with', 'a', 'summary', '.']
Processed User Input (without stopwords and stemming): ['pleas', 'analyz', 'text', 'provid', 'summari', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Double-click (or enter) to edit

▼ BASIC USER INTERACTION

```
[ ] import unicodedata
import re

# Modify the path to your dataset
data = open('dialogs.txt', 'r').read()

# Split the data into question and answer pairs
qa_list = [qa.split('\t') for qa in data.split('\n')]
```



AI_Phase3.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 2:50 PM

+ Code + Text

```
[ ] import unicodedata
import re

# Modify the path to your dataset
data = open('dialogs.txt', 'r').read()

# Split the data into question and answer pairs
QA_list = [QA.split('\t') for QA in data.split('\n')]

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

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

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()
```



AI_Phase3.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 2:50 PM

+ Code + Text



{x}



```
text = re.sub(r"[^\w\s]", "", 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

# Preprocess your questions and answers
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>
```

Double-click (or enter) to edit



CONCLUSION

In conclusion, chatbots represent a significant technological advancement with wide-ranging implications across various domains. This essay has explored the development, applications, and impact of chatbots, highlighting their utility in customer service, healthcare, and education, among others.

Chatbots have demonstrated their ability to enhance customer support by providing immediate responses, thereby increasing efficiency and customer satisfaction. They have also shown potential in the healthcare sector by aiding in symptom diagnosis, appointment scheduling, and medication management, which can ultimately lead to improved patient outcomes. In the education sector, chatbots have the potential to personalize learning experiences, offering adaptive tutoring and assistance to students, thereby addressing individual needs.

While chatbots offer numerous advantages, they are not without their challenges. Natural language understanding, the potential for biased responses, and the need for ongoing maintenance and training are all concerns that must be addressed. Furthermore, the ethical implications of chatbots, particularly in sensitive fields like healthcare, must be carefully considered.

As technology continues to advance and as chatbot development becomes more sophisticated, we can anticipate even greater integration into our daily lives and across various industries. Chatbots have the potential to drive efficiency, improve customer experiences, and provide innovative solutions in an increasingly digital world.

In conclusion, chatbots are more than just computer programs simulating conversations; they are powerful tools that have the capacity to reshape the way we interact with technology and each other. Their evolution is an exciting journey with the promise of substantial benefits, but it is essential to navigate this path with awareness of the challenges and ethical considerations that come with it. The future of chatbots holds immense potential, and it will be fascinating to witness how they continue to evolve and impact our lives.