TO CREATE A CHATBOT IN PYTHON

NAME: Abhinand. R

REG NO : 720421104001

INTRODUCTION:

1. *Loading the Dataset*:

- This step refers to obtaining the dataset that you intend to work with. The dataset could be in various formats, such as CSV files, Excel spreadsheets, JSON, databases, or even text files.

- Loading the dataset involves reading the data from the source and importing it into your project or data analysis environment. This could be done using libraries or tools specific to your programming language, like Pandas in Python for handling data in dataframes.

2. *Preprocessing the Dataset*:

- Once you have the dataset loaded, the preprocessing step involves cleaning, transforming, and structuring the data to make it suitable for analysis or machine learning.

- Preprocessing tasks may include:

  - *Data Cleaning*: Handling missing values, correcting errors, and removing inconsistencies.

- *Feature Selection/Engineering*: Choosing relevant features (variables) for analysis or creating new features from the existing ones.

- *Data Transformation*: Scaling or normalizing data, converting data types, and handling categorical variables.

- *Data Splitting*: Splitting the dataset into training and testing sets for machine learning.

- *Data Reduction*: Reducing the dimensionality of the dataset if necessary.

- *Dealing with Outliers*: Identifying and handling outliers in the data.

- *Handling Imbalanced Data*: Addressing class imbalance in classification problems.

- *Encoding*: Converting categorical data into numerical form using techniques like one-hot encoding**.**

3. *Exploratory Data Analysis (EDA)*:

- This step often goes hand in hand with preprocessing. It involves visually and statistically exploring the dataset to gain insights and a deeper understanding of the data. EDA helps identify patterns, trends, and relationships within the data.

4. *Data Visualization*:

   - Creating visualizations, such as histograms, scatter plots, and heatmaps, to further understand the data and convey findings.

5. *Normalization and Standardization*:

   - Ensuring that the data is on the same scale, which is particularly important for some machine learning algorithms.

PROCESS:

STEP 1: building a chatbot using python

```
import numpy as np
import time
import os
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
```

step2:create chatbot instance

```
# checkpoint
checkpoint = "microsoft/DialoGPT-medium"
# download and cache tokenizer
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
# download and cache pre-trained model
model = AutoModelForCausalLM.from_pretrained(checkpoint)
```

STEP 3: training the chatbot

```
# Build a ChatBot class with all necessary modules to make a complet
e conversation
class ChatBot():
    # initialize
    def__init__(self):
        # once chat starts, the history will be stored for chat continuity
        self.chat_history_ids = None
        # make input ids global to use them anywhere within the object
        self.bot_input_ids = None
        # a flag to check whether to end the conversation
        self.end_chat = False
        # greet while starting
        self.welcome()

    def welcome(self):
        print("Initializing ChatBot ...")
        # some time to get user ready
        time.sleep(2)
        print('Type "bye" or "quit" or "exit" to end chat \n')
        # give time to read what has been printed
        time.sleep(3)
        # Greet and introduce
        greeting = np.random.choice([
            "Welcome, I am ChatBot, here for your kind service",
            "Hey, Great day! I am your virtual assistant",
            "Hello, it's my pleasure meeting you",
            "Hi, I am a ChatBot. Let's chat!"
        ])
        print("ChatBot >>  " + greeting)

    def user_input(self):
        # receive input from user
        text = input("User    >> ")
        # end conversation if user wishes so
        if text.lower().strip() in ['bye', 'quit', 'exit']:
            # turn flag on
            self.end_chat=True
            # a closing comment
            print('ChatBot >> See you soon! Bye!')
            time.sleep(1)
            print('\nQuitting ChatBot ...')
        else:
            # continue chat, preprocess input text
        # encode the new user input, add the eos_token and return a te
nsor in Pytorch
        self.new_user_input_ids = tokenizer.encode(text + tokenizer.eos
_token, \
                                            return_tensors='pt')

    def bot_response(self):
        # append the new user input tokens to the chat history
        # if chat has already begun
        if self.chat_history_ids is not None:
            self.bot_input_ids = torch.cat([self.chat_history_ids, self.new_us
er_input_ids], dim=-1)
        else:
            # if first entry, initialize bot_input_ids
            self.bot_input_ids = self.new_user_input_ids

        # define the new chat_history_ids based on the preceding chats
        # generated a response while limiting the total chat history to 100
0 tokens,
        self.chat_history_ids = model.generate(self.bot_input_ids, max_len
gth=1000, \
                                        pad_token_id=tokenizer.eos_token_id)

        # last ouput tokens from bot
        response = tokenizer.decode(self.chat_history_ids[:, self.bot_input
_ids.shape[-1]:][0], \
                            skip_special_tokens=True)
        # in case, bot fails to answer
        if response == "":
            response = self.random_response()
        # print bot response
        print('ChatBot >>  '+ response)

    # in case there is no response from model
    def random_response(self):
        i = -1
        response = tokenizer.decode(self.chat_history_ids[:, self.bot_input
_ids.shape[i]:][0], \
                           skip_special_tokens=True)
        # iterate over history backwards to find the last token
        while response == '':
            i = i-1
            response = tokenizer.decode(self.chat_history_ids[:, self.bot_inp
ut_ids.shape[i]:][0], \
                           skip_special_tokens=True)
        # if it is a question, answer suitably
        if response.strip() == '?':
            reply = np.random.choice(["I don't know",
                                 "I am not sure"])
        # not a question? answer suitably
        else:
            reply = np.random.choice(["Great",
                                  "Fine. What's up?",
                                  "Okay"
                                 ])
        return reply
```

Step 4: chatbot testing

```
# build a ChatBot object
bot = ChatBot()
# start chatting
while True:
    # receive user input
    bot.user_input()
    # check whether to end chat
    if bot.end_chat:
        break
    # output bot response
      bot.bot_response()
```

OUTPUT:

```
Initializing ChatBot ...
Type "bye" or "quit" or "exit" to end chat

ChatBot >> Welcome, I am ChatBot, here for your kind service
ChatBot >> I'm good, how are you?
ChatBot >>  I do.
ChatBot >> I don't really cook.
ChatBot >> I don't really eat food.
ChatBot >> I like that.
ChatBot >>  I like coffee.
ChatBot >> I don't drink coffee.
ChatBot >> I don't drink coffee
ChatBot >> Fine. What's up?
ChatBot >>  Okay
ChatBot >> See you soon! Bye!

Quitting ChatBot ...
```

CONCLUSION:

This article was based on learning how to make a chatbot in Python using the ChatterBot library. Building a chatbot with ChatterBot was not only simple, but also, the results were accurate. With Artificial Intelligence and Machine Learning, in advancement, everything and anything is possible to achieve whether it is creating bots with conversational skills like humans or be it anything else. Check out DataCamp's tutorial on **Deploying Facebook Chatbots with Python**.