# CREATE CHATBOT IN PYTHON

Au952721104011 -k.Mohamed Mansoor

**Phase-4 Submission**

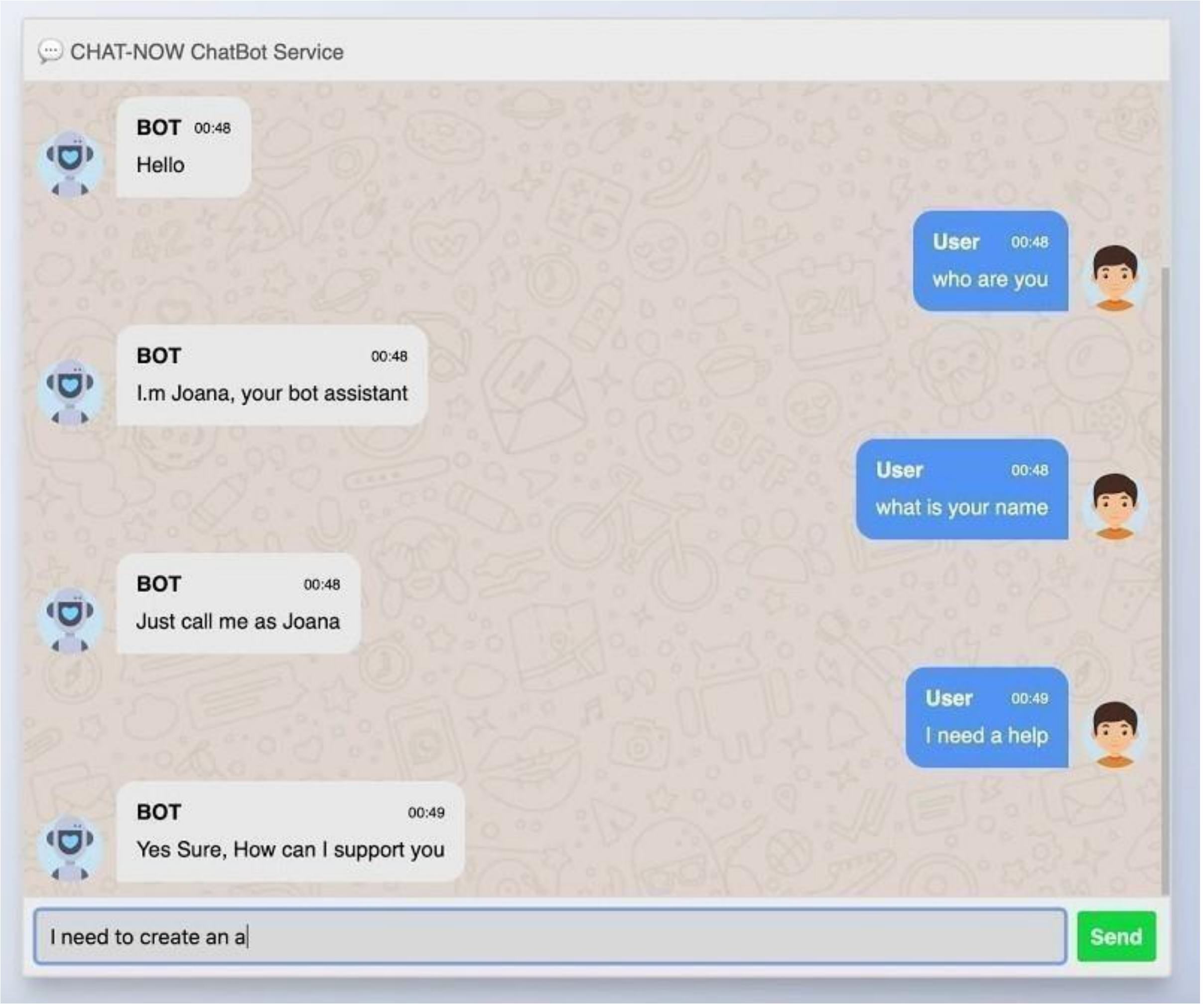
**Project Title:create chatbot in python Phase 4: Development part – 2.**

***Topic*** **-** continue building the project by performing different activities like feature engineering, model training, evaluation

## Introduction:

### Automated Feature Engineering Basics

In this notebook, we will walk through applying automated feature engineering to the Home Credit Default Risk dataset using the featuretools library. Featuretools is an open-source Python package for automatically creating new features from multiple tables of structured, related data. It is ideal tool for problems such as the Home Credit Default Risk competition where there are several related tables that need to be combined into a single dataframe for training (and one for testing).




## Feature Engineering

The objective of feature engineering is to create new features (alos called explantory variables or predictors) to represent as much information from an entire dataset in one table. Typically, this process is done by hand using pandas operations such as `groupby`, `agg`, or `merge` and can be very tedious. Moreover, manual feature engineering is limited both by human time constraints and imagination: we simply cannot conceive of every possible feature that will be useful. (For an example of using manual feature engineering, check out part one and part two applied to this competition). The importance of creating the proper features cannot be overstated because a machine learning model can only learn from the data we give to it. Extracting as much information as possible from the available datasets is crucial to creating an effective solution.

Automated feature engineering aims to help the data scientist with the problem of feature creation by automatically building hundreds or thousands of new features from a dataset. Featuretools - the only library for automated feature engineering at the moment - will not replace the data scientist, but it will allow her to focus on more valuable parts of the machine learning pipeline, such as delivering robust models into production.

Here we will touch on the concepts of automated feature engineering with featuretools and show how to implement it for the Home Credit Default Risk competition. We will stick to the basics so we can get the ideas down and then build upon this foundation in later work when we customize featuretools. We will work with a subset of the data because this is a computationally intensive job that is outside the capabilities of the Kaggle kernels. I took the work done in this notebook and ran the methods on the entire dataset with the results available here. At the end of this notebook, we'll look at the features themselves, as well as the results of modeling with different combinations of hand designed and automatically built features.

If you are new to this competition, I suggest checking out this post to get started. For a good take on why features are so important, here's a blog post by one of the developers of Featuretools.

In [1]:

```
# Uncomment and run if kernel does not already have featuretools
# !pip install featuretools
```

In [2]:

```
# pandas and numpy for data manipulation import
pandas as pd import numpy as np

# featuretools for automated feature engineering import featuretools as ft

# matplotlit and seaborn for visualizations import
matplotlib.pyplot as plt plt.rcParams['font.size'] = 22 import
seaborn as sns

# Suppress warnings from pandas import warnings
warnings.filterwarnings('ignore')
```

linkcode

# Problem

The Home Credit Default Risk competition is a supervised classification machine learning task. The objective is to use historical financial and socioeconomic data to predict whether or not an applicant will be able to repay a loan. This is a standard supervised classification task:

- **Supervised**: The labels are included in the training data and the goal is to train a model to learn to predict the labels from the features
- **Classification**: The label is a binary variable, 0 (will repay loan on time), 1 (will have difficulty repaying loan)

## Dataset

The data is provided by Home Credit, a service dedicated to provided lines of credit (loans) to the unbanked population.

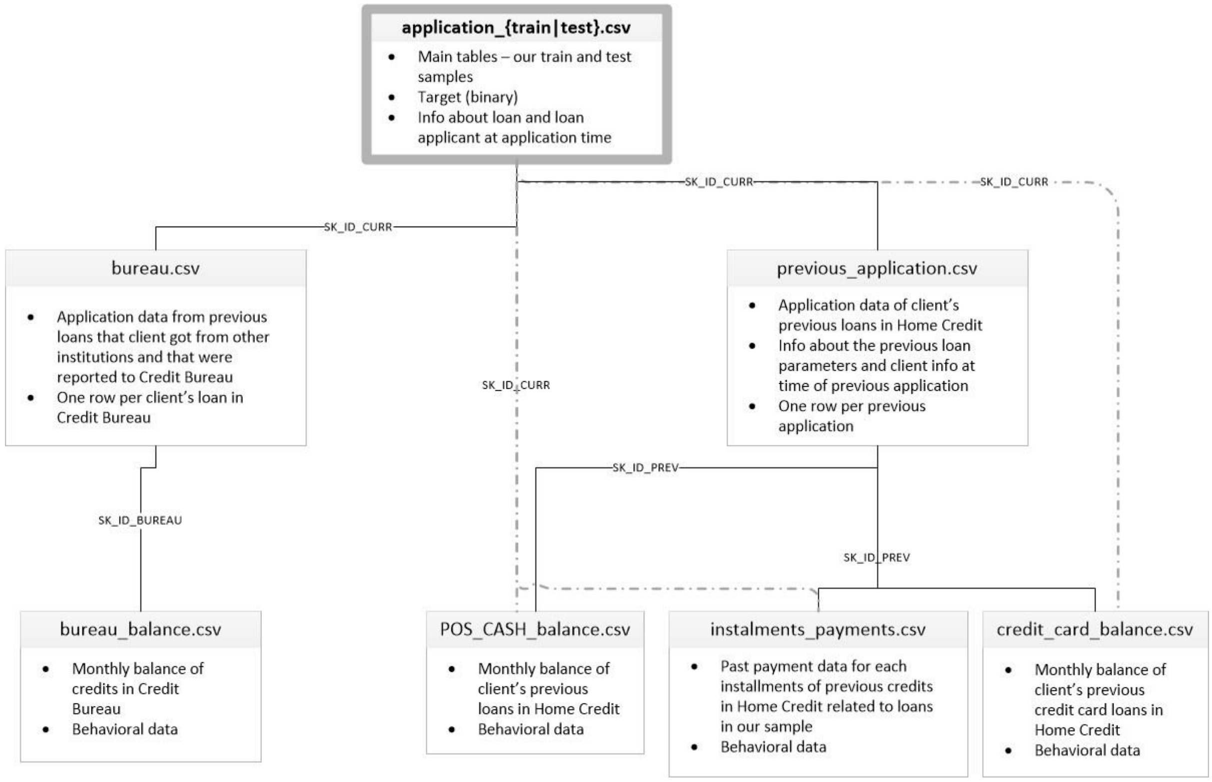
There are 7 different data files:

- **application_train/application_test**: the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the `SK_ID_CURR`. The training application data comes with the `TARGET` with indicating 0: the loan was repaid and 1: the loan was not repaid.
- **bureau**: data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau and is identified by the `SK_ID_BUREAU`, Each loan in the application data can have multiple previous credits.
- **bureau_balance**: monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application**: previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data

can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.

- **POS_CASH_BALANCE**: monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- **credit_card_balance**: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment**: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

The diagram below (provided by Home Credit) shows how the tables are related. This will be very useful when we need to define relationships in featuretools.




# Training Model

Now, we will create the training data in which we will provide the input and the output.

- Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn't understand text so we will convert text into numbers

In [9]:

```
# create our training data training = []
# create an empty array for our output output_empty = [0] * len(classes)
# training set, bag of words for each sentence for doc in documents:
  # initialize our bag of words     bag = []
  # list of tokenized words     pattern_words = doc[0]
  # convert pattern_words in lower case
  pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern
_words]
```

*# create bag of words array,if word match found in current pattern then put 1 otherwise 0.[row * colm(263)]* for w **in** words: bag.append(1) if w **in** pattern_words else bag.append(0)

*# in output array 0 value for each tag ang 1 value for matched tag.[row * colm(8)]*
output_row = list(output_empty) output_row[classes.index(doc[1])] = 1

training.append([bag, output_row]) *# shuffle training and turn into np.array* random.shuffle(training) training = np.array(training)
*# create train and test. X - patterns(words), Y - intents(tags)* train_x = list(training[:,0])
train_y = list(training[:,1]) print("Training data created")

Training data created

In [10]:

linkcode
from tensorflow.python.framework import ops ops.reset_default_graph()

# Build the model

We have our training data ready, now we will build a deep neural network that has 3 layers. We use the Keras sequential API for this. After training the model for 200 epochs, we achieved 100% accuracy on our model. Let us save the model as 'chatbot_model.h5'.

In [11]: *# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons*
*# equal to number of intents to predict output intent with softmax* model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu')) model.add(Dropout(0.5))
model.add(Dense(64, activation='relu')) model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax')) print("First layer:",model.layers[0].get_weights()[0])

```
First layer: [[ 0.08108504 -0.06599443 -0.10388638 ... -0.01234975  0.0
2568085
   0.00633688]
 [-0.02540757 -0.0221673  -0.0489299  ...  0.10772091  0.00711305
   0.03869867]
 [-0.06639696 -0.05009066 -0.03959011 ... -0.0571945  -0.11444904   -0.06228179]
 ...
 [ 0.02686372  0.0873628   0.12299983 ... -0.07360662  0.05407895
  -0.01691054]
 [-0.08417445 -0.10581411 -0.07542053 ... -0.06181952 -0.12180413
  -0.08388676]
 [-0.07259022  0.11421812 -0.04386763 ...  0.00979565  0.05784626    0.09121044]]
```

In [12]: *# Compile model. Stochastic gradient descent with Nesterov accelerated gradi ent gives good results for this model*
*# sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)*
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=[' accuracy'])

In [13]:

*#fitting and saving the model*
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_si ze=5, verbose=1)
model.save('chatbot_model.h5', hist)

print("model created") Epoch 1/200

81/81 [==============================] - 1s 2ms/step - loss: 3.6136 - a ccuracy: 0.0543
Epoch 2/200
81/81 [==============================] - 0s 2ms/step - loss: 3.4736 - a ccuracy: 0.1259
Epoch 3/200
81/81 [==============================] - 0s 2ms/step - loss: 3.2848 - a ccuracy: 0.1753
Epoch 4/200
81/81 [==============================] - 0s 2ms/step - loss: 3.0604 - a ccuracy: 0.2346
Epoch 5/200
81/81 [==============================] - 0s 2ms/step - loss: 2.8305 - a ccuracy: 0.2716
Epoch 6/200
81/81 [==============================] - 0s 2ms/step - loss: 2.5375 - a ccuracy: 0.3432
Epoch 7/200
81/81 [==============================] - 0s 2ms/step - loss: 2.3111 - a ccuracy: 0.4025
Epoch 8/200
81/81 [==============================] - 0s 2ms/step - loss: 2.1470 - a ccuracy: 0.4568
Epoch 9/200
81/81 [==============================] - 0s 2ms/step - loss: 1.9539 - a ccuracy: 0.4864
Epoch 10/200
81/81 [==============================] - 0s 2ms/step - loss: 1.7000 - a ccuracy: 0.6025
Epoch 11/200
81/81 [==============================] - 0s 2ms/step - loss: 1.5961 - a ccuracy: 0.6148
Epoch 12/200
81/81 [==============================] - 0s 2ms/step - loss: 1.4055 - a ccuracy: 0.6593
Epoch 13/200
81/81 [==============================] - 0s 2ms/step - loss: 1.3002 - a ccuracy: 0.6963
Epoch 14/200
81/81 [==============================] - 0s 2ms/step - loss: 1.1978 - a ccuracy: 0.6963
Epoch 15/200
81/81 [==============================] - 0s 2ms/step - loss: 1.0640 - a ccuracy: 0.7407
Epoch 16/200
81/81 [==============================] - 0s 2ms/step - loss: 1.0210 - a ccuracy: 0.7506
Epoch 17/200
81/81 [==============================] - 0s 2ms/step - loss: 0.9202 - a ccuracy: 0.7679
Epoch 18/200
81/81 [==============================] - 0s 2ms/step - loss: 0.8287 - a ccuracy: 0.8099
Epoch 19/200
81/81 [==============================] - 0s 2ms/step - loss: 0.7831 - a ccuracy: 0.8198
Epoch 20/200
81/81 [==============================] - 0s 2ms/step - loss: 0.7525 - a ccuracy: 0.8148
Epoch 21/200
81/81 [==============================] - 0s 2ms/step - loss: 0.7355 - a ccuracy: 0.8123 Epoch
22/200
81/81 [==============================] - 0s 2ms/step - loss: 0.6728 - a ccuracy: 0.8272
Epoch 23/200
81/81 [==============================] - 0s 2ms/step - loss: 0.6377 - a ccuracy: 0.8321
Epoch 24/200
81/81 [==============================] - 0s 2ms/step - loss: 0.5440 - a ccuracy: 0.8741
Epoch 25/200

```
81/81 [==============================] - 0s 2ms/step - loss: 0.4673 - a ccuracy: 0.8889
Epoch 26/200
81/81 [==============================] - 0s 2ms/step - loss: 0.5191 - a ccuracy: 0.8469
Epoch 27/200
81/81 [==============================] - 0s 2ms/step - loss: 0.5168 - a ccuracy: 0.8840
Epoch 28/200
81/81 [==============================] - 0s 2ms/step - loss: 0.4686 - a ccuracy: 0.8864
Epoch 29/200
81/81 [==============================] - 0s 2ms/step - loss: 0.4586 - a ccuracy: 0.8790
Epoch 30/200
81/81 [==============================] - 0s 2ms/step - loss: 0.4126 - a ccuracy: 0.8963
Epoch 31/200
81/81 [==============================] - 0s 2ms/step - loss: 0.4247 - a ccuracy: 0.8889
Epoch 32/200
81/81 [==============================] - 0s 2ms/step - loss: 0.4080 - a ccuracy: 0.8840
Epoch 33/200
81/81 [==============================] - 0s 2ms/step - loss: 0.3659 - a ccuracy: 0.8988
Epoch 34/200
81/81 [==============================] - 0s 2ms/step - loss: 0.4184 - a ccuracy: 0.8889
Epoch 35/200
81/81 [==============================] - 0s 2ms/step - loss: 0.3590 - a ccuracy: 0.9062
Epoch 36/200
81/81 [==============================] - 0s 2ms/step - loss: 0.3597 - a ccuracy: 0.9185
Epoch 37/200
81/81 [==============================] - 0s 2ms/step - loss: 0.3258 - a ccuracy: 0.9111
Epoch 38/200
81/81 [==============================] - 0s 2ms/step - loss: 0.3448 - a ccuracy: 0.9111
Epoch 39/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2794 - a ccuracy: 0.9259
Epoch 40/200
81/81 [==============================] - 0s 2ms/step - loss: 0.3334 - a ccuracy: 0.9012
Epoch 41/200
81/81 [==============================] - 0s 2ms/step - loss: 0.3310 - a ccuracy: 0.9037
Epoch 42/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2302 - a ccuracy: 0.9407
Epoch 43/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2965 - a ccuracy: 0.9185
Epoch 44/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2444 - a ccuracy: 0.9333
Epoch 45/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2701 - a ccuracy: 0.9210
Epoch 46/200
81/81 [==============================] - 0s 2ms/step - loss: 0.3027 - a ccuracy: 0.9309
Epoch 47/200
81/81 [==============================] - 0s 3ms/step - loss: 0.2240 - a ccuracy: 0.9531
Epoch 48/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2129 - a ccuracy: 0.9432
Epoch 49/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2348 - a ccuracy: 0.9407
Epoch 50/200
```

81/81 [==============================] - 0s 2ms/step - loss: 0.2572 - a ccuracy: 0.9358
Epoch 51/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2377 - a ccuracy: 0.9259
Epoch 52/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2324 - a ccuracy: 0.9358
Epoch 53/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2190 - a ccuracy: 0.9407
Epoch 54/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2175 - a ccuracy: 0.9432 Epoch
55/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2259 - a ccuracy: 0.9160
Epoch 56/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2127 - a ccuracy: 0.9481
Epoch 57/200
81/81 [==============================] - 0s 2ms/step - loss: 0.1997 - a ccuracy: 0.9457
Epoch 58/200
81/81 [==============================] - 0s 2ms/step - loss: 0.1975 - a ccuracy: 0.9407
Epoch 59/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2083 - a ccuracy: 0.9333
Epoch 60/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2078 - a ccuracy: 0.9407
Epoch 61/200
81/81 [==============================] - 0s 2ms/step - loss: 0.1838 - a ccuracy: 0.9432
Epoch 62/200
81/81 [==============================] - 0s 2ms/step - loss: 0.1736 - a ccuracy: 0.9506
Epoch 63/200
81/81 [==============================] - 0s 2ms/step - loss: 0.2022 - a ccuracy: 0.9407
Epoch 64/200
81/81 [==============================] - 0s 2ms/step - loss: 0.1883 - a ccuracy: 0.9481

## Evaluation:

**Intelligent ChatBot built with Microsoft's DialoGPT transformer to make conversations with human users!**

*Image by Andy Kelly*

### What is a chatbot?

*A ChatBot is a kind of virtual assistant that can build conversations with human users! A Chatting Robot. Building a chatbot is one of the popular tasks in Natural Language Processing.*

### Are all chatbots the same?

*Chatbots fall under three common categories:*

1. *Rule-based chatbots*
2. *Retrieval-based chatbots*
3. *Intelligent chatbots*

### Rule-based chatbots

*These bots respond to users' inputs based on certain pre-specified rules. For instance, these rules can be defined as if-elif-else statements. While writing rules for these chatbots, it is important to expect all possible user inputs, else the bot may fail to answer properly. Hence, rule-based chatbots do not possess any cognitive skills.*

### Retrieval-based chatbots

*These bots respond to users' inputs by retrieving the most relevant information from the given text document. The most relevant information can be determined by Natural Language Processing with a scoring system such as cosine-similarity-score. Though these bots use NLP to do conversations, they lack cognitive skills to match a real human chatting companion.*

### Intelligent AI chatbots

***These bots respond to users' inputs after understanding the inputs, as humans do. These bots are trained with a Machine Learning Model on a large training dataset of human conversations. These bots are cognitive to match a human in conversing. Amazon's Alexa, Apple's Siri fall under this category. Further, most of these bots can make conversations based on the preceding chat texts.***

### In this Article?

***This article describes building an intelligent AI chatbot based on the famous transformer architecture - Microsoft's DialoGPT. According to Hugging Face's model card, DialoGPT is a State-Of-The-Art large-scale pretrained dialogue response generation model for multiturn conversations. The human evaluation results indicate that the response generated from DialoGPT is comparable to human response quality under a single-turn conversation Turing test. The model is trained on 147M multi-turn dialogue from Reddit discussion thread.***

# Let's Python

***Import necessary libraries and frameworks***

In [1]:

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

# Download Microsoft's DialoGPT model and tokenizer

***The Hugging Face checkpoint for the model and its tokenizer is "microsoft/DialoGPTmedium"***

In [2]:

```
# 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)
```

# A ChatBot class

In [3]:

linkcode

```
# Build a ChatBot class with all necessary modules to make a complete conver sation 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 tens or 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
_user_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 1000 tokens,
      self.chat_history_ids = model.generate(self.bot_input_ids, max_leng th=1000, \
                              pad_token_id=tokenizer.eos_t oken_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_i nput_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
```

**CONCLUSION**

In conclusion, building a chatbot in Python offers a versatile and effective way to automate conversa tions and provide information or services to users. Python's rich libraries and frameworks, like NLTK, spaCy, and TensorFlow, make it a powerful choice for natural language processing. Chatbots can be used for various applications, from customer support to virtual assistants. To create a successful chat bot, you should focus on designing a user-friendly conversational flow, integrating AI and machine le arning techniques for understanding and generating responses, and continually improving its perfor mance through user feedback and iterative development. Python provides the tools and resources n eeded to create intelligent and engaging chatbots.