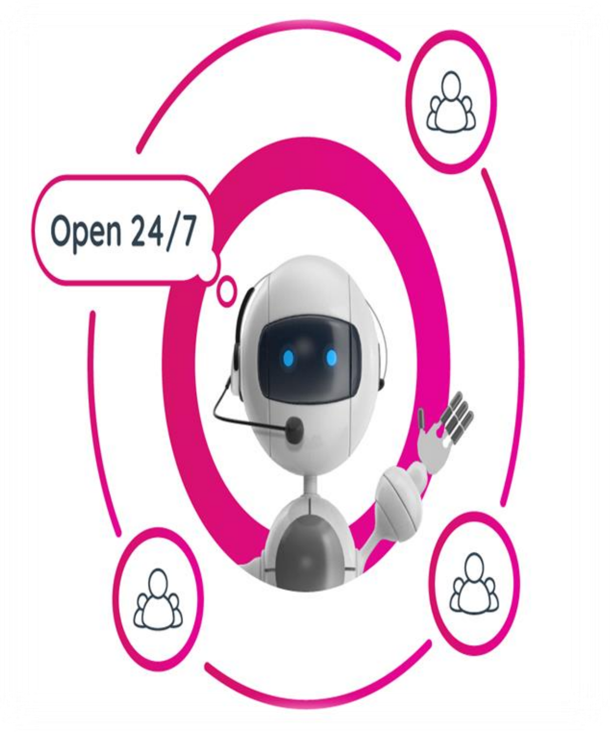


ChatBot

AMIT KUMAR

| <u>Data Science with Python Internship Project</u> |
(Cioncent pvt.)

05/05/2023

Phone no. 7991156683,7277399505

Abstract

In light of the quick growth of technology, the notion of the chat bot, and the time and work it may save. There are many specialized frameworks available to carry out chatbot programming. By depending on artificial intelligence, the chat has expanded to include more technological disciplines and has combined machine learning with it. To make the process of getting information about students' inquiries about admissions, registration, and the institution itself easier, we will design a chatbot for the university's admission and registration.

A lot of individuals are utilizing smart assistant devices like Google Home or Alexa from Amazon. The chatbot was evaluated across a variety of platforms, including Google Home and Android's Assistant, to highlight the key distinctions between the spoken and text-based ways of communication. The purpose of these tests was to determine the utility of incorporating technologies related to the current interest in chatbots and conversational interfaces. demonstrating that chatbots can be used in a particular industry to improve accessibility, reiterating that they are more than simply a passing trend and have a practical application.

Table of content

Objective

As we develop the ChatBot in training phase a PIZZA ORDER chatbot who help end-user to order the pizza and talk politely as just like human. So, that thing will help a lot to users for save their time as well as any complexity to understand the procedure like, how to order, where we insert your data and all. In main project we done with a dataset where we have story , questions and answer is their, we just do python command for analysis the way of expressing the view.

However, the main goal of a chatbot project is to create a computer program that can communicate with humans in natural language and provide useful information or perform tasks.

Customer support: Providing 24/7 customer support and assistance to customers, answering frequently asked questions, and resolving issues.
Sales and Marketing: Engaging with potential customers, promoting products and services, and generating leads.
Personalized recommendations: Using data and machine learning to provide personalized recommendations to users based on their preferences and behavior.
Information retrieval: Providing instant access to information such as news, weather, and traffic updates.
Task automation: Automating repetitive tasks such as booking appointments, ordering food, and making reservations.
Overall, the objective of a chatbot project is to enhance the customer experience, increase efficiency, and save time and resources for businesses.

1.Introduction

1.1. Introduction:

Chatbot that uses text messaging or voice chat is a computer that simulates a conversation with a human user. Artificial intelligence (AI) and natural language processing (NLP) technologies are used to create chatbots that can understand human speech and respond in a way that is relevant and helpful to the user. Chatbots can be used on various platforms such as websites, mobile applications, messaging services and voice assistants. It has grown in popularity in recent years due to its ability to engage customers, automate processes, and deliver personalized experiences.

1.2. Problem Statement:

At the start of each academic semester, registration opens for those wishing to join the university in various disciplines, and telephone calls for admission and registration abound. This leads to an increase in the loads and work for the employees of the Deanship of Admission and Registration as a result of the constant pressure of those wishing to register and their families by flocking to the Deanship, so the employees are not able to answer the phone calls and social media. This often leads to many students who wish to register to be ignored.

1.3. Scope:

People who wish to enroll Palestine Polytechnic University
Admission and registration staff

Methodology

The methodology of building a chatbot using Python can be summarized in the following steps:

1. Define the use case and gather requirements: Determine the purpose of the chatbot and the tasks it should perform. Identify the target audience and the channels through which the chatbot will be deployed. As in my case I used pickle library to load data.

2. Collect data: Gather relevant data and information to train the chatbot. This may include customer conversations, product information, and FAQs.

3. Preprocessing data: Prepare the data by cleaning and formatting it to make it usable for machine learning algorithms. The data was already clear just to preprocess and use the tokenized the data and then combine to get required output.

4. Train the chatbot: Train the chatbot using the preprocessed data and machine learning algorithms. The chatbot should be trained on a diverse range of data to improve its accuracy and responsiveness.

5. Test and refine the chatbot: Test the chatbot using sample conversations and refine it based on user feedback. The chatbot should be continuously improved to enhance its performance and accuracy.

Overall, building a chatbot using Python involves a combination of programming skills, natural language processing knowledge, and machine learning expertise.

CODE

```
#!/usr/bin/env python
# coding: utf-8

# In[11]:


import pickle
import numpy as np


# In[12]:


with open("train_qa.txt","rb") as fp:
    train_data = pickle.load(fp)


# In[13]:


train_data
```

```
# In[14]:


with open("test_qa.txt","rb") as fp:
    test_data = pickle.load(fp)


# In[15]:


test_data


# In[16]:


type(test_data)


# In[17]:
```

```
type(train_data)

# In[18]:

len(test_data)

# In[19]:

len(train_data)

# In[20]:

train_data[0]

# In[21]:
```

```
' '.join(train_data[0][0])

# In[22]:

' '.join(train_data[0][1])

# In[23]:

train_data[0][2]

# In[24]:

#set up vacabulary
vocab = set()
# In[25]:
all_data = test_data+ train_data
```

```
# In[26]:

type(all_data)

# In[27]:all_data

# In[28]:for story, question, answer in all_data:

    vocab = vocab.union(set(story))

    vocab = vocab.union(set(question))

# In[29]:vocab.add('yes')

vocab.add('no')

# In[30]:vocab

# In[31]:len(vocab)

# In[32]:vocab_len = len(vocab)+1

# In[33]:all_data

# In[34]:for data in all_data:

    print(data)

    print("\n")

# In[35]:for data in all_data:

    print(data[0])

    print("\n")

# In[36]:vocab_len = len(vocab)+1

# In[37]:max_story_len = max([len(data[0]) for data in all_data])

max_story_len

# In[38]:max_ques_len = max([len(data[1]) for data in all_data])

max_ques_len
```

```
# In[39]:vocab

# In[2]:pip install tensorflow

# In[1]:from tensorflow import keras

# In[5]:from tensorflow.keras.preprocessing.sequence import pad_sequences

from keras.preprocessing.text import Tokenizer

# In[9]:tokenizer = Tokenizer(filters = [])

# In[40]:tokenizer.fit_on_texts(vocab)

# In[41]: tokenizer.word_index

# In[44]: train_story_text = []

train_question_text = []

train_answer= []

for story, question, answer in train_data:

    train_story_text.append(story)

    train_question_text.append(question)

# In[45]:train_story_seq = tokenizer.texts_to_sequences(train_story_text)

# In[46]:len(train_story_seq)

# In[48]:len(train_story_text)

# In[50]:train_story_text

# In[51]: train_story_seq

# In[56]: def vectorize_stories(data, word_index = tokenizer.word_index,

                     max_story_len = max_story_len, max_ques_len =
max_ques_len):

    X = [] #STORies
    Xq = []#queri/questiom
    Y = []#correct answer
    for story, query, answer in data:
        x = [word_index[word.lower()] for word in story]
        xq = [word_index[word.lower()] for word in query]
        y = np.zeros(len(word_index)+1)
        y[word_index[answer]] = 1


        X.append(x)
        Xq.append(xq)
        Y.append(y)
    return(pad_sequences(X, maxlen = max_story_len),
          pad_sequences(Xq, maxlen = max_ques_len),
          np.array(Y))
# In[57]:inputs_train, queries_train, answer_train = 
vectorize_stories(train_data)
# In[58]:inputs_test, queries_test, answer_test = vectorize_stories(test_data)
# In[59]:inputs_train
# In[60]:queries_test
# In[61]:answer_test
# In[62]:tokenizer.word_index['yes']
# In[63]:tokenizer.word_index['no']
# In[72]:from tensorflow import keras
```

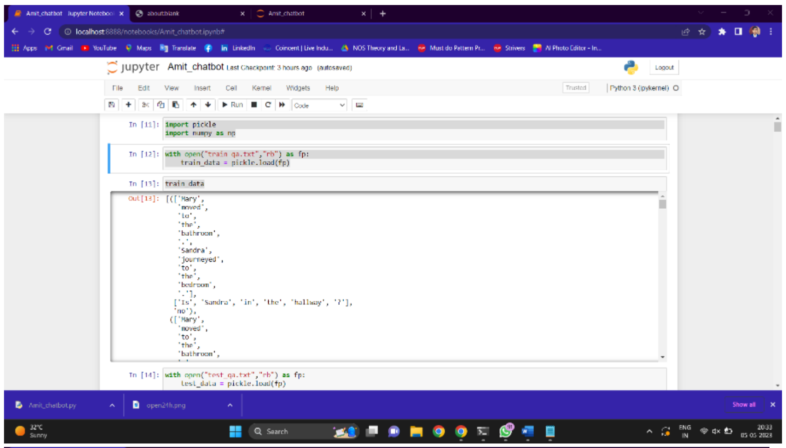
```
from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Embedding

from tensorflow.keras.layers import Input, Activation, Dense, Permute,
Dropout, add, dot, concatenate, LSTM

# In[74]:

input_sequence = Input((max_story_len,))

question = Input((max_ques_len,))

# In[77]:

#Input_Encoder_m

input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim = vocab_len, output_dim= 64))

input_encoder_m.add(Dropout(0.3))

# In[81]:

#input_encoder_c

input_encoder_c = Sequential()

input_encoder_c.add(Embedding(input_dim = vocab_len, output_dim =
max_ques_len))

input_encoder_c.add(Dropout(0,3))

# In[87]:

#Question encoder

question_encoder = Sequential()

question_encoder.add(Embedding(input_dim = vocab_len, output_dim = 64,
input_length = max_ques_len))

question_encoder_c.add(Dropout(0,3))
```

```
# In[88]:

#encode the Sequences

input_encoded_m = input_encoder_m(input_sequence)

input_encoded_c = input_encoder_c(input_sequence)

question_encoded = question_encoder(question)

# In[89]:

match = dot([input_encoded_m,question_encoded],axes = (2,2))

match = Activation('softmax')(match)

# In[90]:

response = add([match, input_encoded_c])

response = Permute((2,1))(response)

# In[91]:

#Concatenate

answer = concatenate([response,question_encoded ])

# In[92]:

answer

# In[93]:

answer = LSTM(32)(answer)

# In[94]:

answer = Dropout(0.5)(answer)

answer = Dense(vocab_len)(answer)

# In[95]:

answer = Activation('softmax')(answer)
```

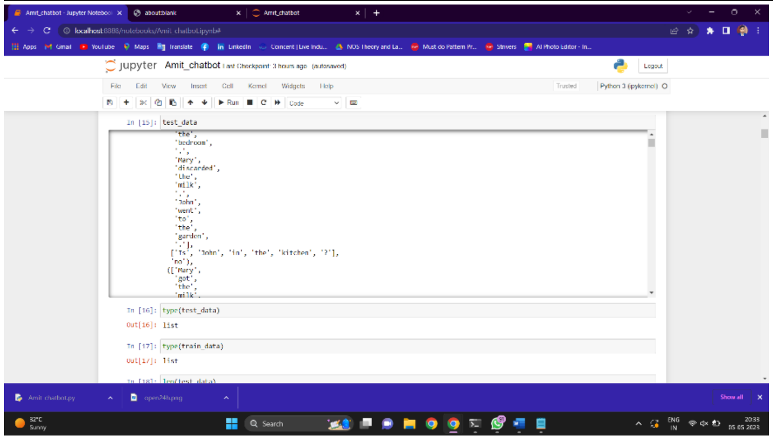
```
# In[96]:

model = Model([input_sequence, question],answer)

model.compile(optimizer = 'rmsprop',loss=
'categorical_crossentropy',metrics= ['accuracy'])

# In[97]:

model.summary()

# In[99]:

history = model.fit([inputs_train, queries_train], answer_train,

                    batch_size = 32, epochs = 20,

                    validation_data = ([inputs_test, queries_test],

                                       answer_test))

# In[106]:

import matplotlib.pyplot as plt

print(history.history.keys())

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title("Model Accuracy")

plt.ylabel("Accuracy")

plt.xlabel("epochs")

#plt.legends(['train','test'], loc = 'upper left')

# In[107]:

#savethis model

model.save("Chatbot_model_amit")
```

```
# In[108]:

#evaluation on the test set

model.load_weights("Chatbot_model_amit")

# In[109]:

pred_results = model.predict(([inputs_test, queries_test]))

# In[110]:

test_data[0][0]

# In[113]:

story = ''.join(word for word in test_data[0][0])

story

query = ''.join(word for word in test_data[0][1])

query

test_data[0][2]

val_max = np.argmax(pred_results[0])


for key, val in tokenizer.word_index.items():

    if val ==val_max:

        k = key

print("predicted Answer is ", k)

print("Probablility of cetunity", pred_results[0][val_max])

# In[121]:vocab
```

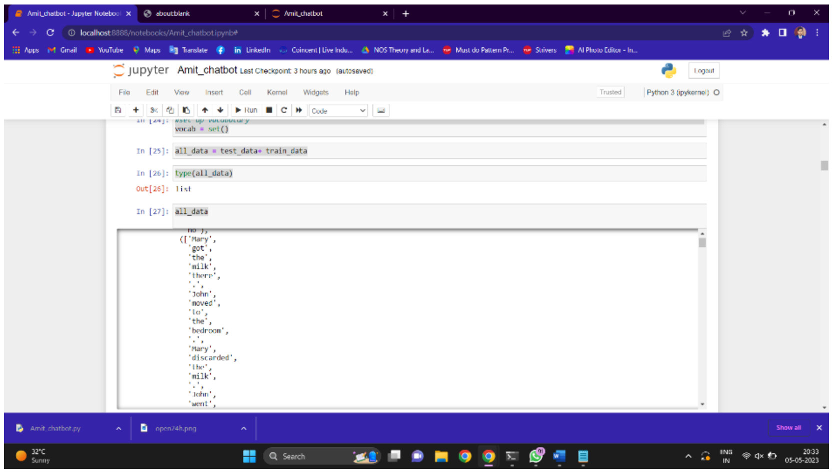
Output:



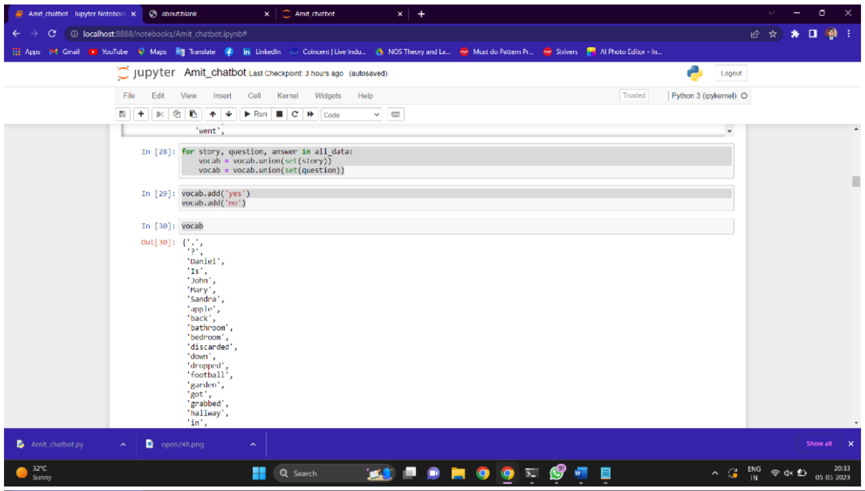
```
In [11]: import pickle
         import numpy as np

In [12]: with open("train_qa.txt","rb") as fp:
             train_data = pickle.load(fp)

In [13]: train_data

Out[13]: [(['Mary',
            'moved',
            'to',
            'the',
            'bathroom',
            '.',
            'Sandra',
            'journeyed',
            'to',
            'the',
            'bedroom',
            '.'],
           ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
           'no'),
          (['Mary',
            'moved',
            'to',
            'the',
            'bathroom',

In [14]: with open("test_qa.txt","rb") as fp:
             test_data = pickle.load(fp)
```



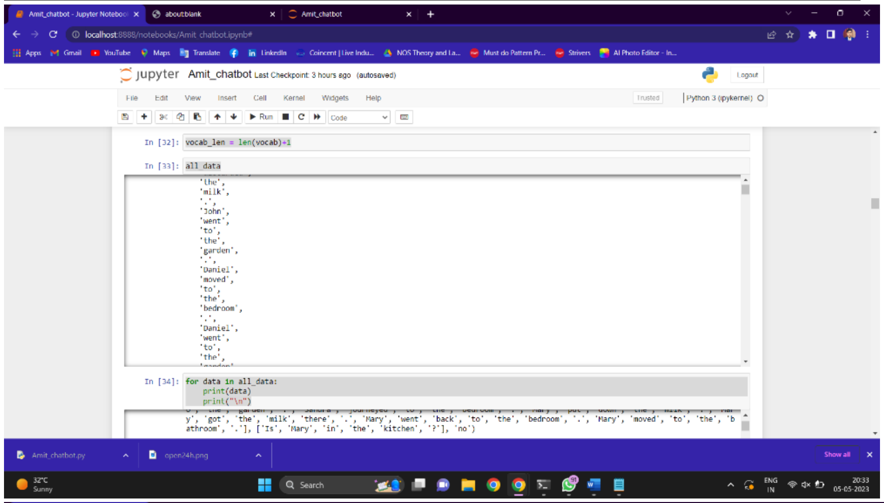
```
In [15]: test_data
            'the',
            'bedroom',
            '.',
            'Mary',
            'discarded',
            'the',
            'milk',
            '.',
            'John',
            'went',
            'to',
            'the',
            'garden',
            '.'],
           ['Is', 'John', 'in', 'the', 'kitchen', '?'],
           'no'),
          (['Mary',
            'got',
            'the',
            'milk',

In [16]: type(test_data)
Out[16]: list

In [17]: type(train_data)
Out[17]: list
```



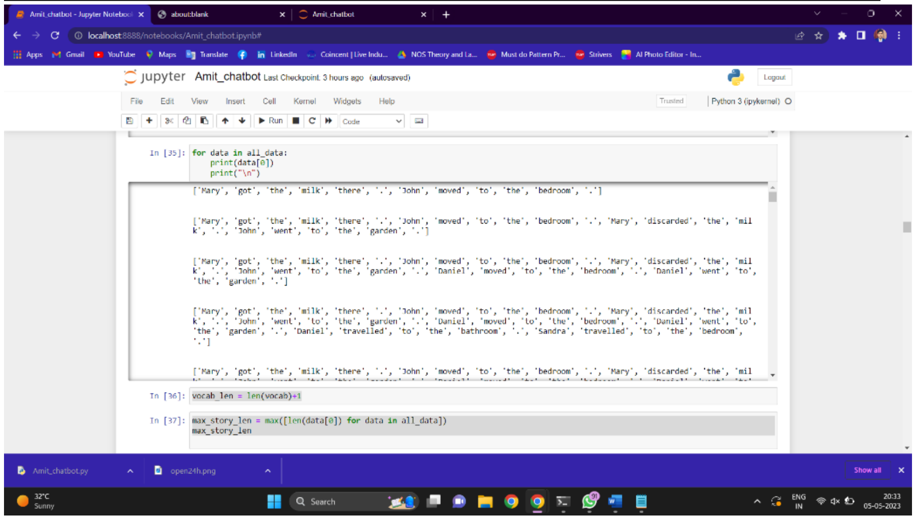
```
         vocab = set()

In [25]: all_data = test_data+ train_data

In [26]: type(all_data)
Out[26]: list

In [27]: all_data
          (['Mary',
            'got',
            'the',
            'milk',
            'there',
            '.',
            'John',
            'moved',
            'to',
            'the',
            'bedroom',
            '.',
            'Mary',
            'discarded',
            'the',
            'milk',
            '.',
            'John',
            'went',
```



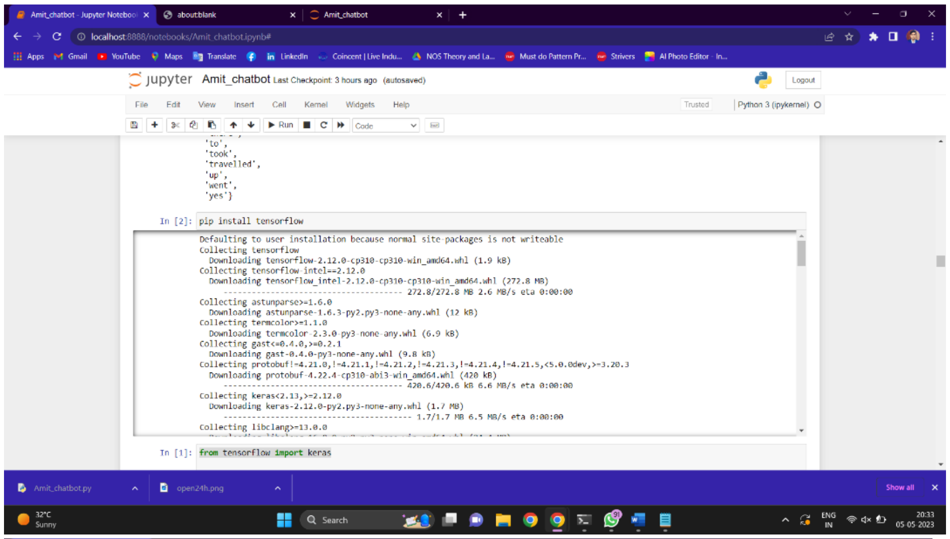
```
 'went',
In [28]: for story, question, answer in all_data:
             vocab = vocab.union(set(story))
             vocab = vocab.union(set(question))

In [29]: vocab.add('yes')
         vocab.add('no')

In [30]: vocab

Out[30]: {'.',
          '?',
          'Daniel',
          'Is',
          'John',
          'Mary',
          'Sandra',
          'apple',
          'back',
          'bathroom',
          'bedroom',
          'discarded',
          'down',
          'dropped',
          'football',
          'garden',
          'got',
          'grabbed',
          'hallway',
          'in',
```



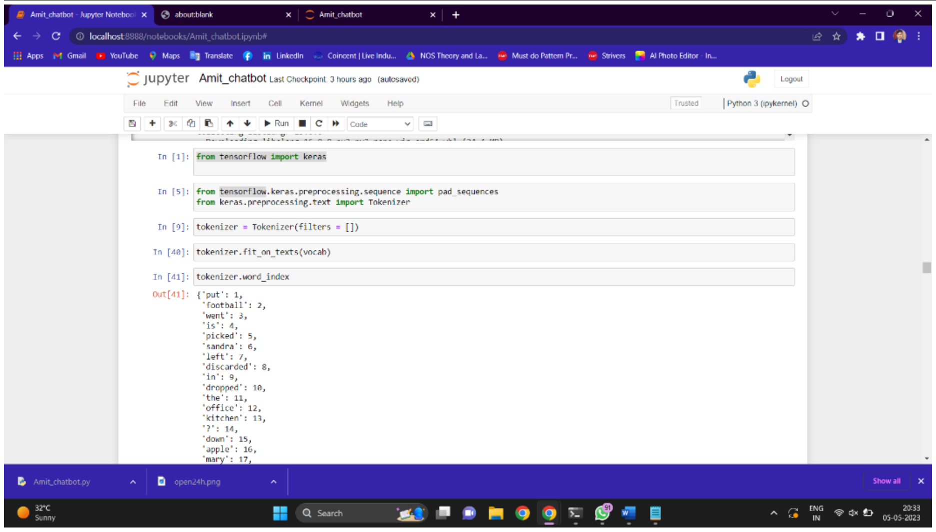
```
In [32]: vocab_len = len(vocab)+1

In [33]: all_data

          'the',
          'milk',
          '.',
          'John',
          'went',
          'to',
          'the',
          'garden',
          '.',
          'Daniel',
          'moved',
          'to',
          'the',
          'bedroom',
          '.',
          'Daniel',
          'went',
          'to',
          'the',

In [34]: for data in all_data:
             print(data)
             print("\n")

         y', 'got', 'the', 'milk', 'there', '.', 'Mary', 'went', 'back', 'to', 'the', 'bedroom', '.', 'Mary', 'moved', 'to', 'the', 'b
         athroom', '.'], ['Is', 'Mary', 'in', 'the', 'kitchen', '?'], 'no')
```



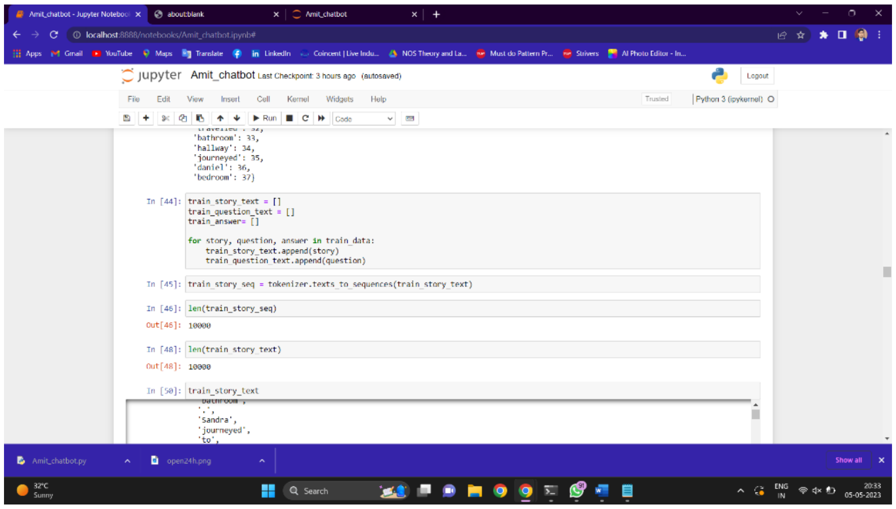
```
In [35]: for data in all_data:
             print(data[0])
             print("\n")

         ['Mary', 'got', 'the', 'milk', 'there', '.', 'John', 'moved', 'to', 'the', 'bedroom', '.']


         ['Mary', 'got', 'the', 'milk', 'there', '.', 'John', 'moved', 'to', 'the', 'bedroom', '.', 'Mary', 'discarded', 'the', 'mil
         k', '.', 'John', 'went', 'to', 'the', 'garden', '.']


         ['Mary', 'got', 'the', 'milk', 'there', '.', 'John', 'moved', 'to', 'the', 'bedroom', '.', 'Mary', 'discarded', 'the', 'mil
         k', '.', 'John', 'went', 'to', 'the', 'garden', '.', 'Daniel', 'moved', 'to', 'the', 'bedroom', '.', 'Daniel', 'went', 'to',
         'the', 'garden', '.']


         ['Mary', 'got', 'the', 'milk', 'there', '.', 'John', 'moved', 'to', 'the', 'bedroom', '.', 'Mary', 'discarded', 'the', 'mil
         k', '.', 'John', 'went', 'to', 'the', 'garden', '.', 'Daniel', 'moved', 'to', 'the', 'bedroom', '.', 'Daniel', 'went', 'to',
         'the', 'garden', '.', 'Daniel', 'travelled', 'to', 'the', 'bathroom', '.', 'Sandra', 'travelled', 'to', 'the', 'bedroom',
         '.']


         ['Mary', 'got', 'the', 'milk', 'there', '.', 'John', 'moved', 'to', 'the', 'bedroom', '.', 'Mary', 'discarded', 'the', 'mil

In [36]: vocab_len = len(vocab)+1

In [37]: max_story_len = max([len(data[0]) for data in all_data])
         max_story_len
```



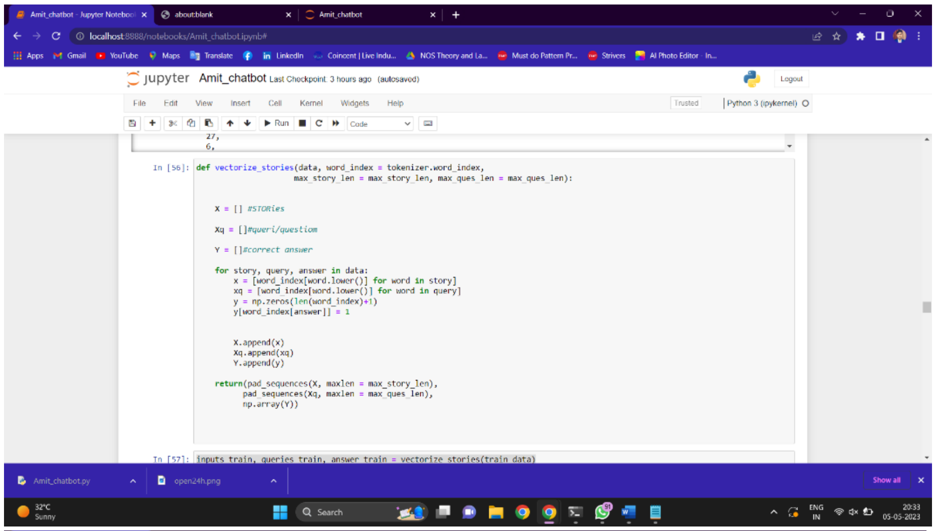
```
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}

In [2]: pip install tensorflow

Defaulting to user installation because normal site-packages is not writeable
Collecting tensorflow
  Downloading tensorflow-2.12.0-cp310-cp310-win_amd64.whl (1.9 kB)
Collecting tensorflow-intel==2.12.0
  Downloading tensorflow_intel-2.12.0-cp310-cp310-win_amd64.whl (272.8 MB)
     ------------------------------------- 272.8/272.8 MB 2.6 MB/s eta 0:00:00
Collecting astunparse>=1.6.0
  Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Collecting termcolor>=1.1.0
  Downloading termcolor-2.3.0-py3-none-any.whl (6.9 kB)
Collecting gast<=0.4.0,>=0.2.1
  Downloading gast-0.4.0-py3-none-any.whl (9.8 kB)
Collecting protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
  Downloading protobuf-4.22.4-cp310-abi3-win_amd64.whl (420 kB)
     ------------------------------------- 420.6/420.6 kB 6.6 MB/s eta 0:00:00
Collecting keras<2.13,>=2.12.0
  Downloading keras-2.12.0-py2.py3-none-any.whl (1.7 MB)
     -------------------------------------- 1.7/1.7 MB 6.5 MB/s eta 0:00:00
Collecting libclang>=13.0.0

In [1]: from tensorflow import keras
```



```
In [1]: from tensorflow import keras

In [5]: from tensorflow.keras.preprocessing.sequence import pad_sequences
        from keras.preprocessing.text import Tokenizer

In [9]: tokenizer = Tokenizer(filters = [])

In [40]: tokenizer.fit_on_texts(vocab)

In [41]: tokenizer.word_index

Out[41]: {'put': 1,
          'football': 2,
          'went': 3,
          'is': 4,
          'picked': 5,
          'sandra': 6,
          'left': 7,
          'discarded': 8,
          'in': 9,
          'dropped': 10,
          'the': 11,
          'office': 12,
          'kitchen': 13,
          '?': 14,
          'down': 15,
          'apple': 16,
          'mary': 17,
```

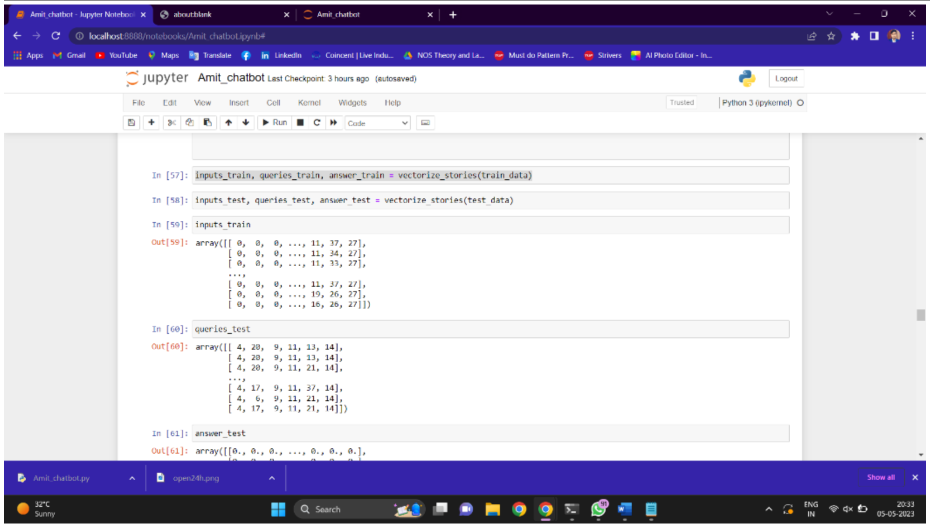


```
          'bathroom': 33,
          'hallway': 34,
          'journeyed': 35,
          'daniel': 36,
          'bedroom': 37}

In [44]: train_story_text = []
         train_question_text = []
         train_answer= []

         for story, question, answer in train_data:
             train_story_text.append(story)
             train_question_text.append(question)

In [45]: train_story_seq = tokenizer.texts_to_sequences(train_story_text)

In [46]: len(train_story_seq)

Out[46]: 10000

In [48]: len(train_story_text)

Out[48]: 10000

In [50]: train_story_text

  '.',
  'Sandra',
  'journeyed',
  'to',
```

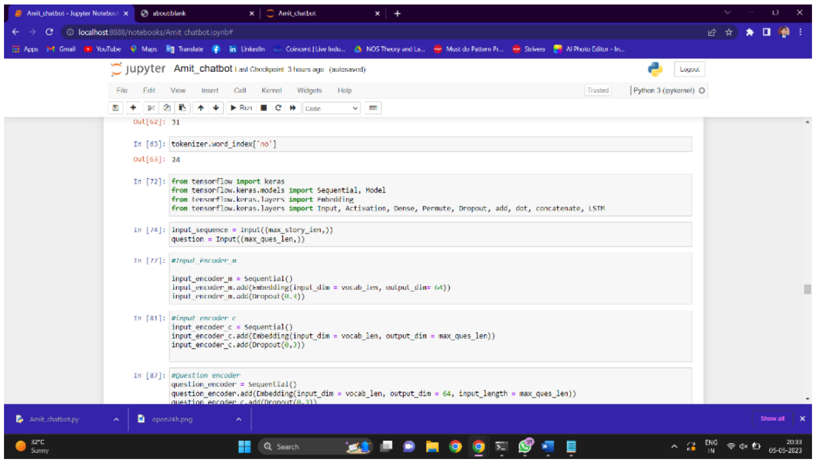


```
       27,
       6,
```

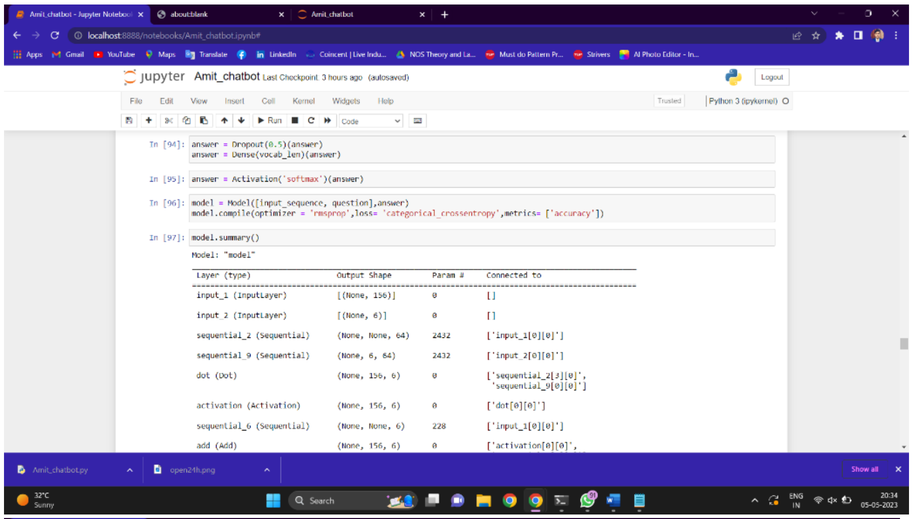
```
In [56]: def vectorize_stories(data, word_index = tokenizer.word_index,
                               max_story_len = max_story_len, max_ques_len = max_ques_len):


             X = [] #STORies

             Xq = []#queri/questiom

             Y = []#correct answer

             for story, query, answer in data:
                 x = [word_index[word.lower()] for word in story]
                 xq = [word_index[word.lower()] for word in query]
                 y = np.zeros(len(word_index)+1)
                 y[word_index[answer]] = 1


                 X.append(x)
                 Xq.append(xq)
                 Y.append(y)

             return(pad_sequences(X, maxlen = max_story_len),
                    pad_sequences(Xq, maxlen = max_ques_len),
                    np.array(Y))
```



```
In [57]: inputs_train, queries_train, answer_train = vectorize_stories(train_data)

In [58]: inputs_test, queries_test, answer_test = vectorize_stories(test_data)

In [59]: inputs_train

Out[59]: array([[ 0,  0,  0, ..., 11, 37, 27],
                [ 0,  0,  0, ..., 11, 34, 27],
                [ 0,  0,  0, ..., 11, 33, 27],
                ...,
                [ 0,  0,  0, ..., 11, 37, 27],
                [ 0,  0,  0, ..., 19, 26, 27],
                [ 0,  0,  0, ..., 16, 26, 27]])

In [60]: queries_test

Out[60]: array([[ 4, 20,  9, 11, 13, 14],
                [ 4, 20,  9, 11, 13, 14],
                [ 4, 20,  9, 11, 21, 14],
                ...,
                [ 4, 17,  9, 11, 37, 14],
                [ 4,  6,  9, 11, 21, 14],
                [ 4, 17,  9, 11, 21, 14]])

In [61]: answer_test

Out[61]: array([[0., 0., 0., ..., 0., 0., 0.],
```

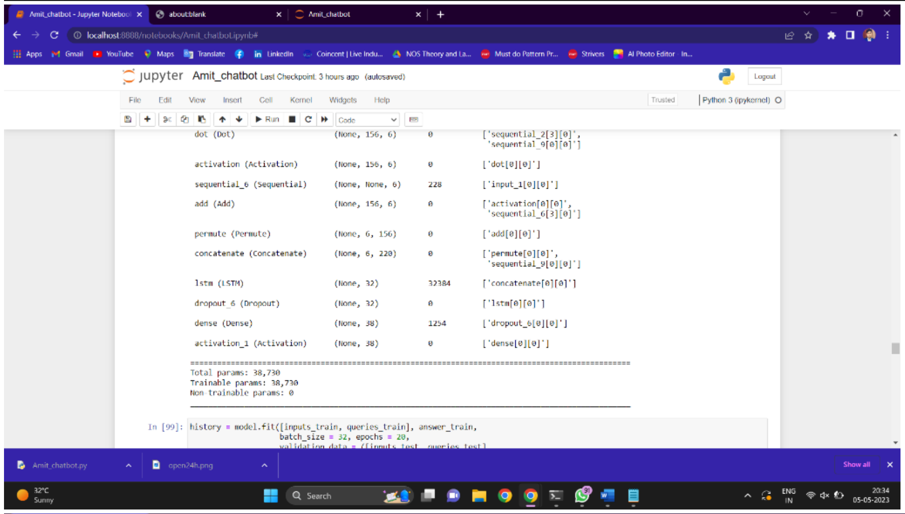


```
Out[62]: 31

In [63]: tokenizer.word_index['no']

Out[63]: 24

In [72]: from tensorflow import keras
         from tensorflow.keras.models import Sequential, Model
         from tensorflow.keras.layers import Embedding
         from tensorflow.keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM

In [74]: input_sequence = Input((max_story_len,))
         question = Input((max_ques_len,))

In [77]: #Input_Encoder_m

         input_encoder_m = Sequential()
         input_encoder_m.add(Embedding(input_dim = vocab_len, output_dim= 64))
         input_encoder_m.add(Dropout(0.3))

In [81]: #input encoder c
         input_encoder_c = Sequential()
         input_encoder_c.add(Embedding(input_dim = vocab_len, output_dim = max_ques_len))
         input_encoder_c.add(Dropout(0,3))

In [87]: #Question encoder
         question_encoder = Sequential()
         question_encoder.add(Embedding(input_dim = vocab_len, output_dim = 64, input_length = max_ques_len))
```



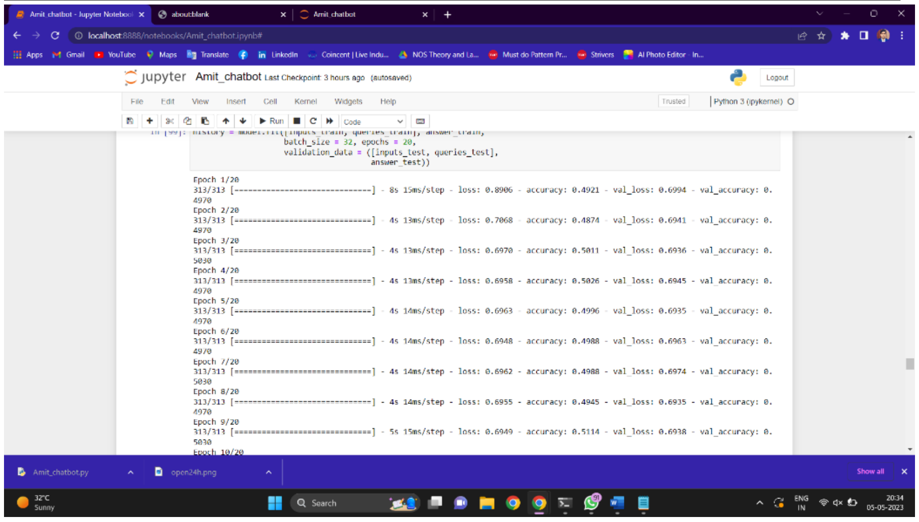
jupyter Amit_chatbot Last Checkpoint: 3 hours ago (autosaved)

```
In [94]: answer = Dropout(0.5)(answer)
         answer = Dense(vocab_len)(answer)

In [95]: answer = Activation('softmax')(answer)

In [96]: model = Model([input_sequence, question],answer)
         model.compile(optimizer = 'rmsprop',loss= 'categorical_crossentropy',metrics= ['accuracy'])

In [97]: model.summary()

         Model: "model"
         ______________________________________________________________________________________
          Layer (type)                   Output Shape         Param #     Connected to
         ======================================================================================
          input_1 (InputLayer)           [(None, 156)]        0           []

          input_2 (InputLayer)           [(None, 6)]          0           []

          sequential_2 (Sequential)      (None, None, 64)     2432        ['input_1[0][0]']

          sequential_9 (Sequential)      (None, 6, 64)        2432        ['input_2[0][0]']

          dot (Dot)                      (None, 156, 6)       0           ['sequential_2[3][0]',
                                                                           'sequential_9[0][0]']

          activation (Activation)        (None, 156, 6)       0           ['dot[0][0]']

          sequential_6 (Sequential)      (None, None, 6)      228         ['input_1[0][0]']

          add (Add)                      (None, 156, 6)       0           ['activation[0][0]',
```



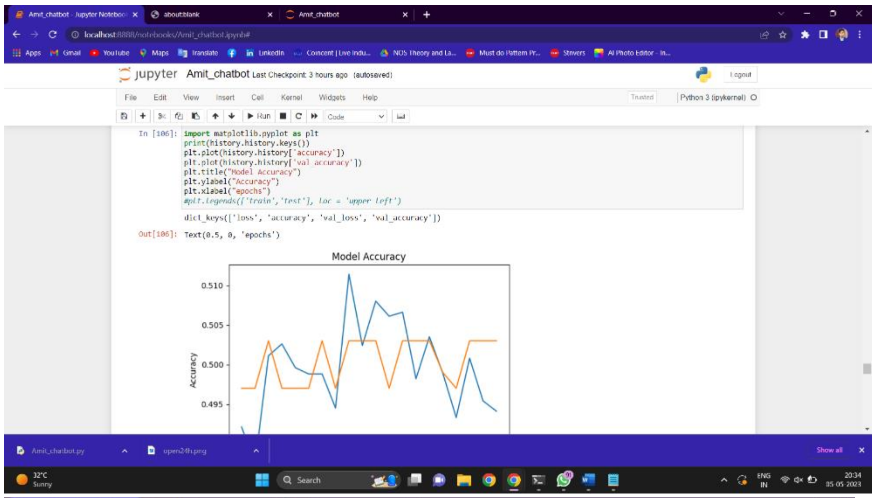
jupyter Amit_chatbot Last Checkpoint: 3 hours ago (autosaved)

```
          dot (Dot)                      (None, 156, 6)       0           ['sequential_2[3][0]',
                                                                           'sequential_9[0][0]']

          activation (Activation)        (None, 156, 6)       0           ['dot[0][0]']

          sequential_6 (Sequential)      (None, None, 6)      228         ['input_1[0][0]']

          add (Add)                      (None, 156, 6)       0           ['activation[0][0]',
                                                                           'sequential_6[3][0]']

          permute (Permute)              (None, 6, 156)       0           ['add[0][0]']

          concatenate (Concatenate)      (None, 6, 220)       0           ['permute[0][0]',
                                                                           'sequential_9[0][0]']

          lstm (LSTM)                    (None, 32)           32384       ['concatenate[0][0]']

          dropout_6 (Dropout)            (None, 32)           0           ['lstm[0][0]']

          dense (Dense)                  (None, 38)           1254        ['dropout_6[0][0]']

          activation_1 (Activation)      (None, 38)           0           ['dense[0][0]']

         ======================================================================================
         Total params: 38,730
         Trainable params: 38,730
         Non-trainable params: 0
         ______________________________________________________________________________________

In [99]: history = model.fit([inputs_train, queries_train], answer_train,
                             batch_size = 32, epochs = 20,
```

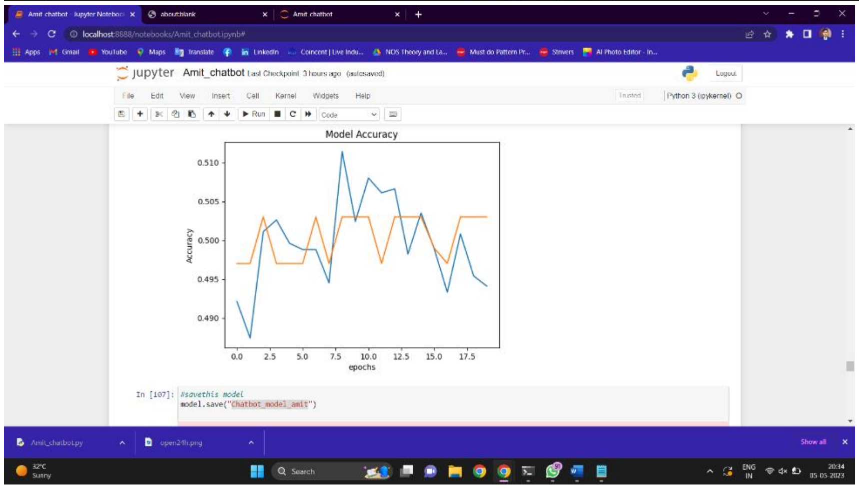


jupyter Amit_chatbot Last Checkpoint: 3 hours ago (autosaved)

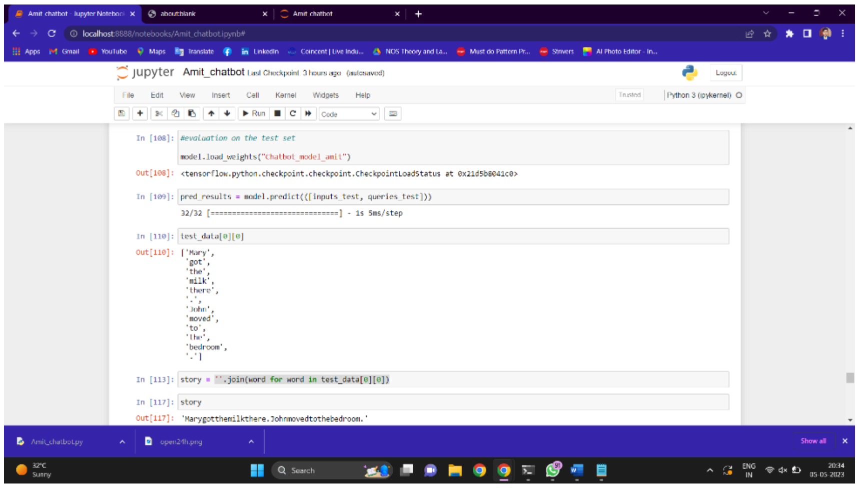
```
In [99]: history = model.fit([inputs_train, queries_train], answer_train,
                             batch_size = 32, epochs = 20,
                             validation_data = ([inputs_test, queries_test],
                                                answer_test))

         Epoch 1/20
         313/313 [==============================] - 8s 15ms/step - loss: 0.8906 - accuracy: 0.4921 - val_loss: 0.6994 - val_accuracy: 0.
         4970
         Epoch 2/20
         313/313 [==============================] - 4s 13ms/step - loss: 0.7068 - accuracy: 0.4874 - val_loss: 0.6941 - val_accuracy: 0.
         4970
         Epoch 3/20
         313/313 [==============================] - 4s 13ms/step - loss: 0.6970 - accuracy: 0.5011 - val_loss: 0.6936 - val_accuracy: 0.
         5030
         Epoch 4/20
         313/313 [==============================] - 4s 13ms/step - loss: 0.6958 - accuracy: 0.5026 - val_loss: 0.6945 - val_accuracy: 0.
         4970
         Epoch 5/20
         313/313 [==============================] - 4s 14ms/step - loss: 0.6963 - accuracy: 0.4996 - val_loss: 0.6935 - val_accuracy: 0.
         4970
         Epoch 6/20
         313/313 [==============================] - 4s 14ms/step - loss: 0.6948 - accuracy: 0.4988 - val_loss: 0.6963 - val_accuracy: 0.
         4970
         Epoch 7/20
         313/313 [==============================] - 4s 14ms/step - loss: 0.6962 - accuracy: 0.4988 - val_loss: 0.6974 - val_accuracy: 0.
         5030
         Epoch 8/20
         313/313 [==============================] - 4s 14ms/step - loss: 0.6955 - accuracy: 0.4945 - val_loss: 0.6935 - val_accuracy: 0.
         4970
         Epoch 9/20
         313/313 [==============================] - 5s 15ms/step - loss: 0.6949 - accuracy: 0.5114 - val_loss: 0.6938 - val_accuracy: 0.
         5030
         Epoch 10/20
```



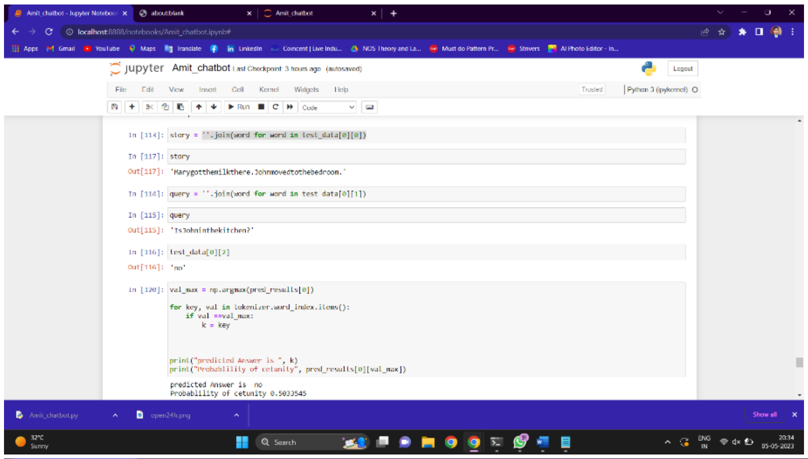
```
In [106]: import matplotlib.pyplot as plt
          print(history.history.keys())
          plt.plot(history.history['accuracy'])
          plt.plot(history.history['val accuracy'])
          plt.title("Model Accuracy")
          plt.ylabel("Accuracy")
          plt.xlabel("epochs")
          #plt.legends(['train','test'], loc = 'upper left')

          dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

Out[106]: Text(0.5, 0, 'epochs')
```



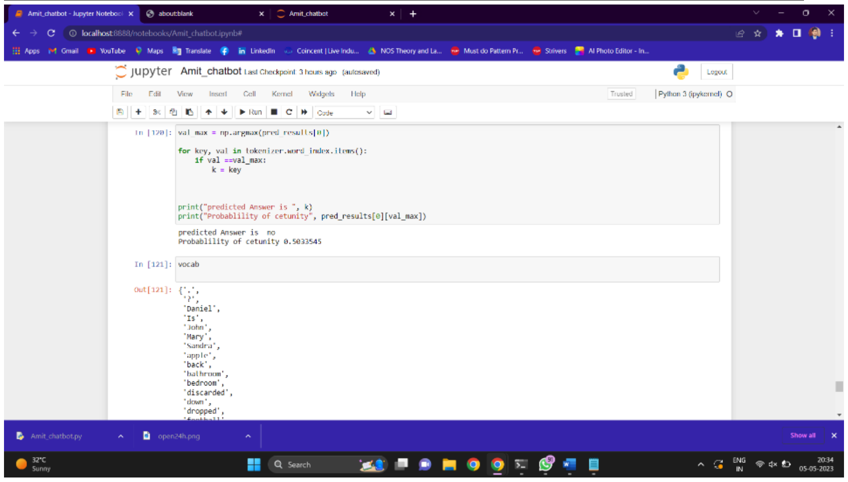
```
In [107]: #savethis model
          model.save("Chatbot_model_amit")
```



```
In [108]: #evaluation on the test set

          model.load_weights("Chatbot_model_amit")

Out[108]: <tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x21d5b0041c0>

In [109]: pred_results = model.predict(([inputs_test, queries_test]))

          32/32 [==============================] - 1s 5ms/step

In [110]: test_data[0][0]

Out[110]: ['Mary',
           'got',
           'the',
           'milk',
           'there',
           '.',
           'John',
           'moved',
           'to',
           'the',
           'bedroom',
           '.']

In [113]: story = ''.join(word for word in test_data[0][0])

In [117]: story

Out[117]: 'Marygotthemilkthere.Johnmovedtothebedroom.'
```



```
In [113]: story = ''.join(word for word in test_data[0][0])

In [117]: story
Out[117]: 'Marygotthemilkthere.Johnmovedtothebedroom.'

In [114]: query = ''.join(word for word in test_data[0][1])

In [115]: query
Out[115]: 'IsJohninthekitchen?'

In [116]: test_data[0][2]
Out[116]: 'no'

In [120]: val_max = np.argmax(pred_results[0])

          for key, val in tokenizer.word_index.items():
              if val ==val_max:
                  k = key



          print("predicted Answer is ", k)
          print("Probablility of cetunity", pred_results[0][val_max])

          predicted Answer is  no
          Probablility of cetunity 0.5033545
```



```
In [120]: val_max = np.argmax(pred_results[0])

          for key, val in tokenizer.word_index.items():
              if val ==val_max:
                  k = key



          print("predicted Answer is ", k)
          print("Probablility of cetunity", pred_results[0][val_max])

          predicted Answer is  no
          Probablility of cetunity 0.5033545

In [121]: vocab

Out[121]: {'.',
           '?',
           'Daniel',
           'Is',
           'John',
           'Mary',
           'Sandra',
           'apple',
           'back',
           'bathroom',
           'bedroom',
           'discarded',
           'down',
           'dropped',
```

Conclusion

Building a chatbot using Python can be a powerful tool for businesses to improve customer engagement, automate tasks, and provide personalized experiences. With the increasing popularity of chatbots, Python provides a robust and versatile programming language that is well-suited for building chatbots.

Using Python, developers can leverage the power of natural language processing libraries like NLTK and spaCy, machine learning frameworks like TensorFlow and PyTorch, and various chatbot development frameworks like Rasa and Chatterbot. This allows developers to build chatbots that can handle a wide range of tasks, including answering FAQs, scheduling appointments, and even making product recommendations.

The process of building a chatbot using Python involves defining the use case and gathering requirements, collecting and preprocessing data, choosing a framework, developing and training the chatbot, testing and refining it, and finally deploying and maintaining it.