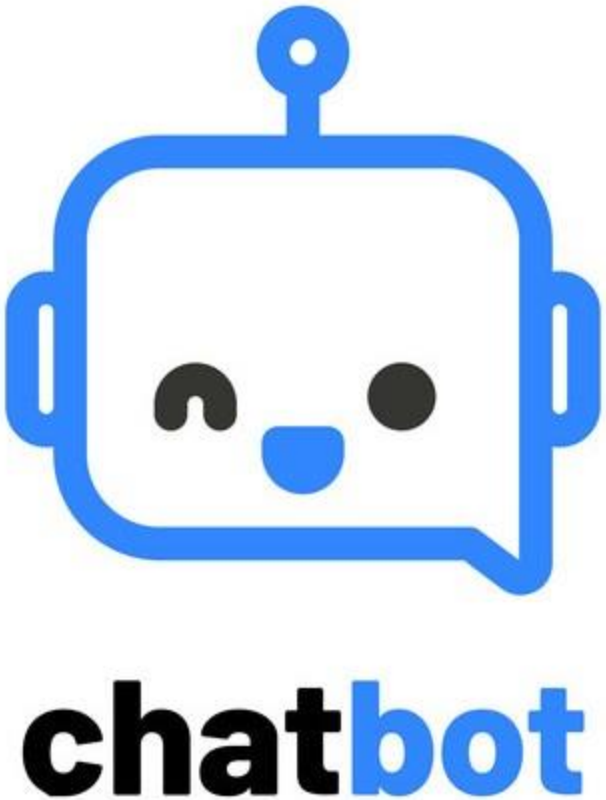
CHATBOT



Presented by :

Sanika Sarang

ABSTRACT:

Chatbot can be described as software that can chat with people using artificial intelligence. These software are used to perform tasks such as quickly responding to users, informing them, helping to purchase products and providing better service to customers. At the most basic level, a chatbot is a computer program that simulates and processes human conversation (either written or spoken), allowing humans to interact with digital devices as if they were communicating with a real person. Chatbots can be as simple as rudimentary programs that answer a simple query with a single-line response, or as sophisticated as digital assistants that learn and evolve to deliver increasing levels of personalization as they gather and process information.

OBJECTIVE:

A chatbot is often described as one of the most advanced and promising expressions of interaction between humans and machines. These digital assistants streamline interactions between people and services, enhancing customer experience. At the same time, they offer companies new opportunities to streamline the customer’s engagement process for efficiency that can reduce traditional support costs.

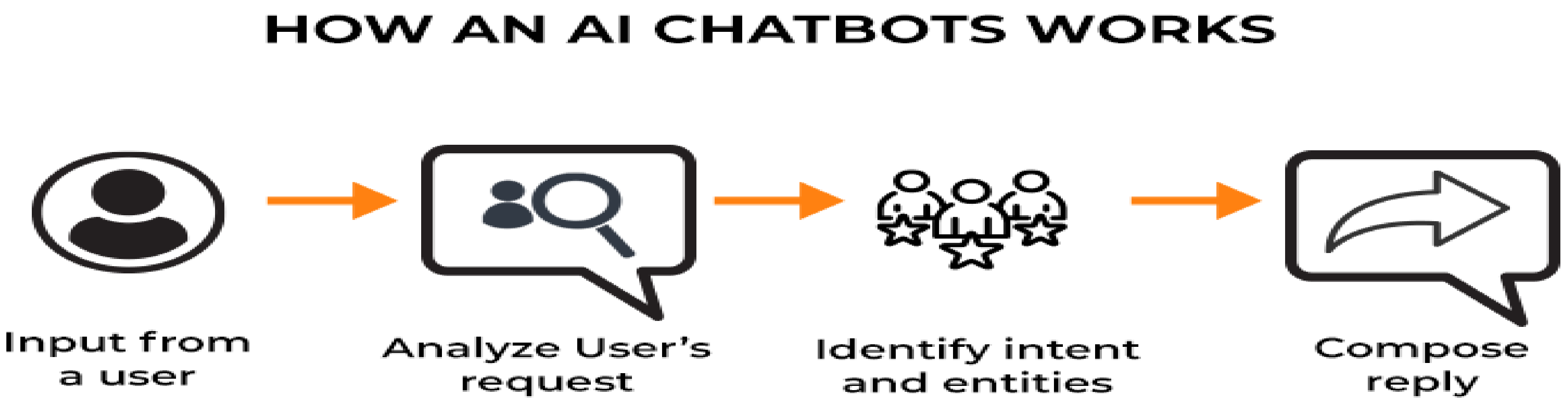
For companies looking to improve their customer experiences, the addition of chatbots to answer simple questions can improve satisfaction, streamline the customer journey, and provide customer-centric support.

INTRODUCTION:

AI chat bots are based on machine learning and natural language processing (NLP). Data power them, and they use it to answer user questions freely. AI bots can learn independently, and they get better with every conversation they have with users. Chatbots are trained to act upon the inputs provided by consumers or they can be driven by rules. They rely on a machine's ability to interpret human language (spoken or typed) and are trained to respond to interactions. The more data fed to the chatbot, the more human-like the response.

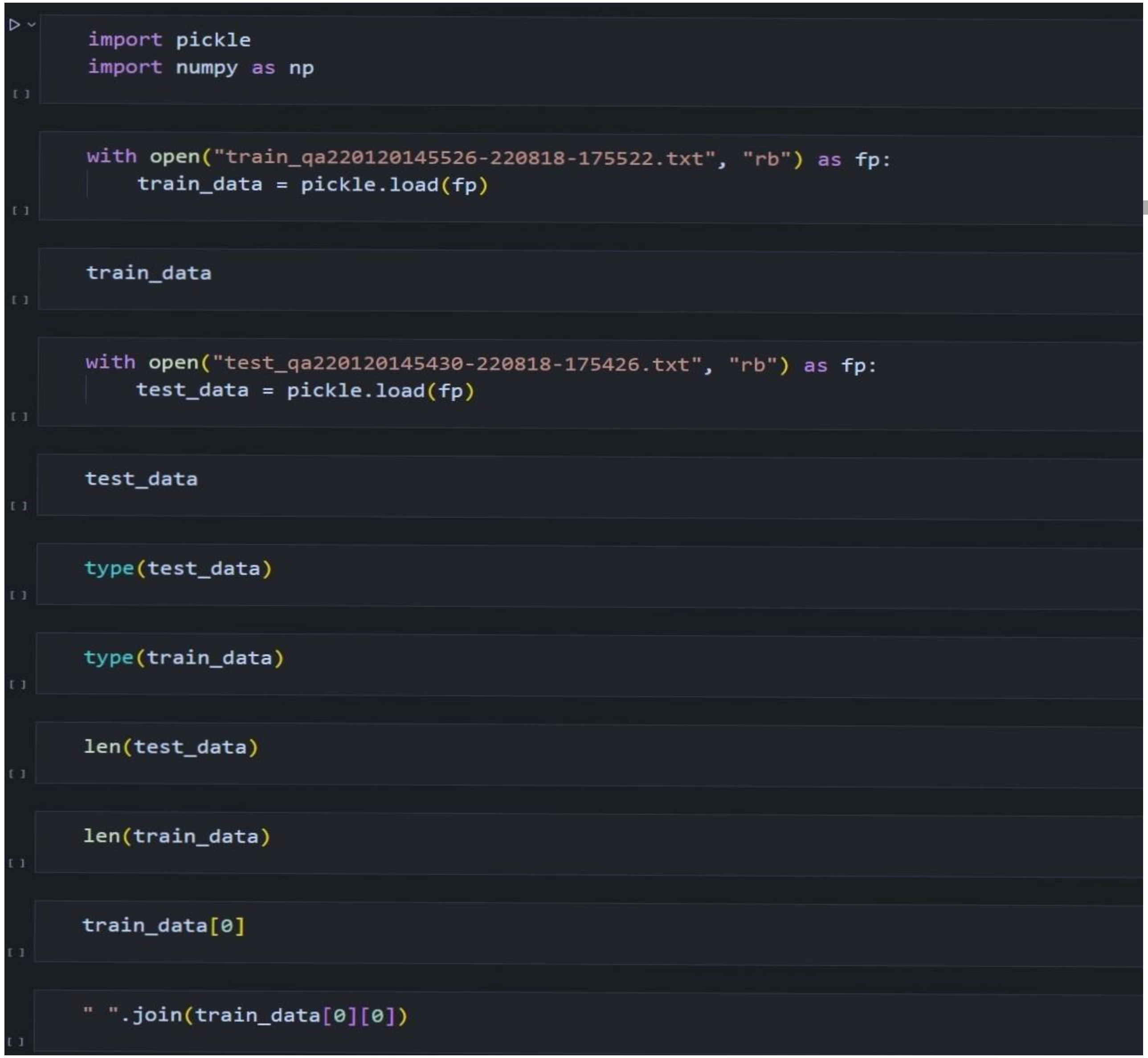
Brands use bots to automate their business processes, speed up customer service, and lower support costs.

METHODOLOGY:

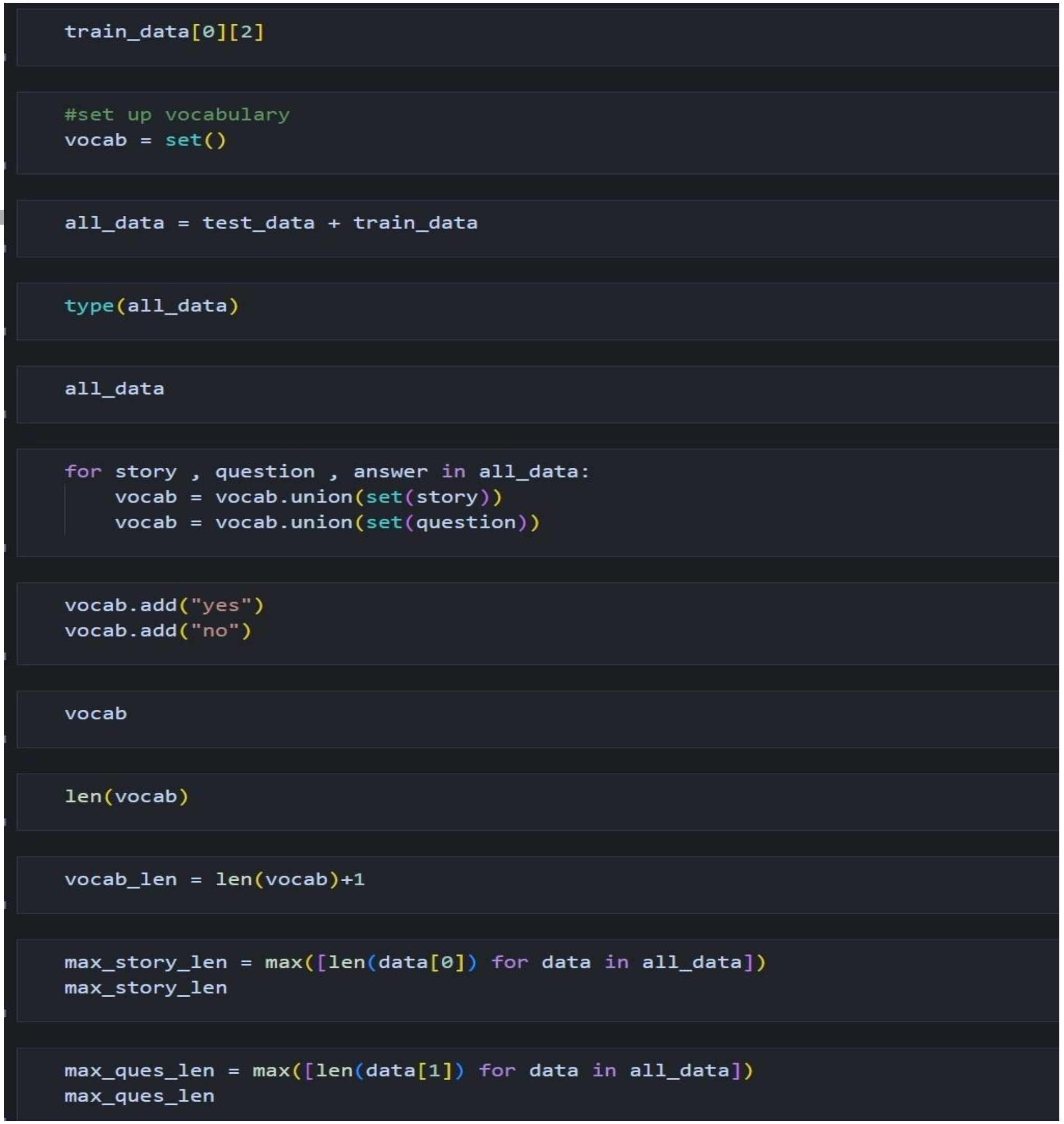


This approach uses a machine learning engine to train itself to deliver an optimal response to a customer query. It learns based on past inquiries and evolves as inputs are analyzed. A large amount of data is needed to train the system, and machine learning of the chatbot application is done in a black box with no insight into what is learned.

CODING:

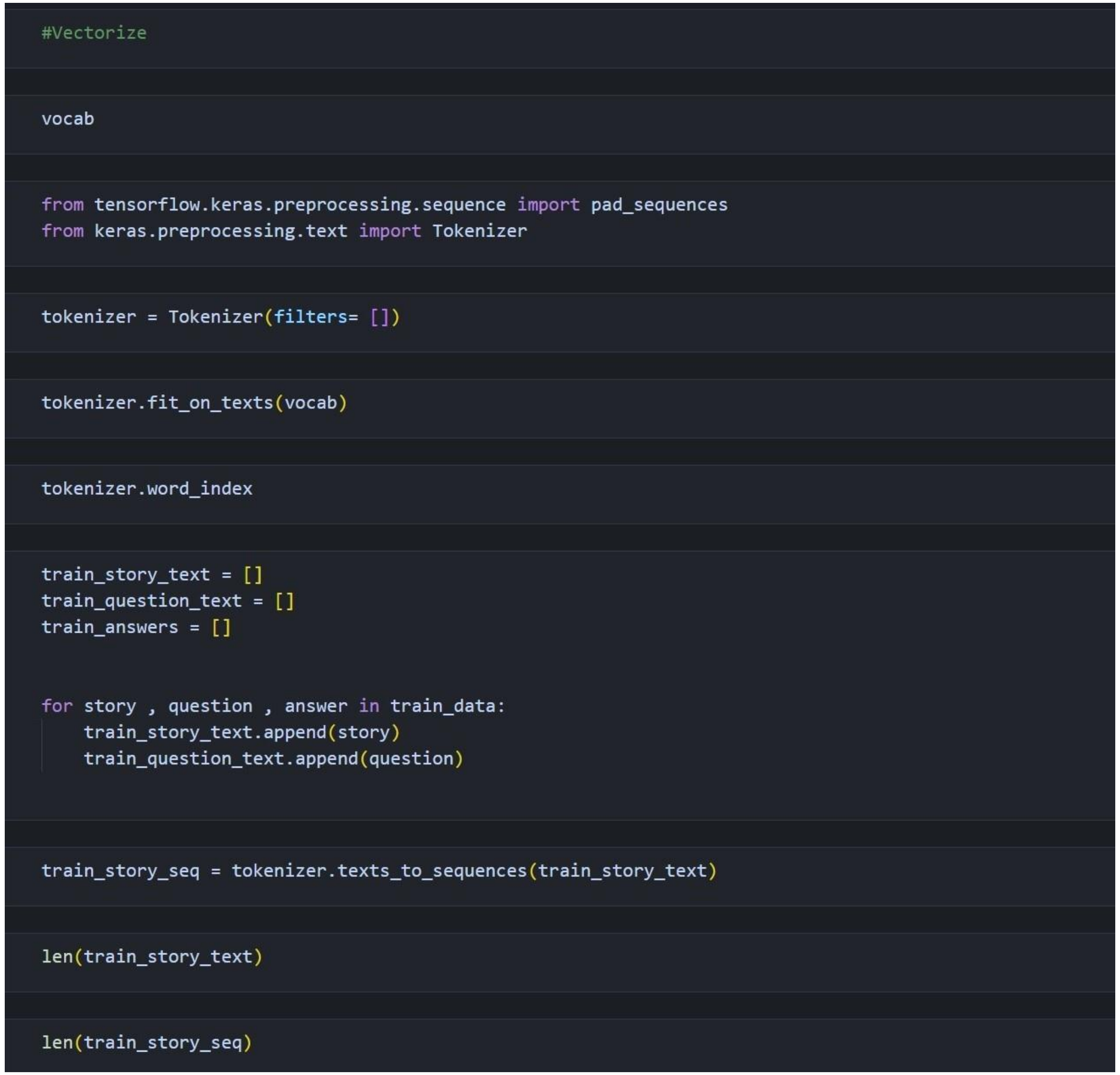


```
import pickle
import numpy as np

with open("train_qa220120145526-220818-175522.txt", "rb") as fp:
    train_data = pickle.load(fp)

train_data

with open("test_qa220120145430-220818-175426.txt", "rb") as fp:
    test_data = pickle.load(fp)

test_data

type(test_data)

type(train_data)

len(test_data)

len(train_data)

train_data[0]

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



```
train_data[0][2]

#set up vocabulary
vocab = set()

all_data = test_data + train_data

type(all_data)

all_data

for story , question , answer in all_data:
    vocab = vocab.union(set(story))
    vocab = vocab.union(set(question))

vocab.add("yes")
vocab.add("no")

vocab

len(vocab)

vocab_len = len(vocab)+1

max_story_len = max([len(data[0]) for data in all_data])
max_story_len

max_ques_len = max([len(data[1]) for data in all_data])
max_ques_len
```

CODING:



```
#Vectorize
```

```
vocab
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
```

```
tokenizer = Tokenizer(filters= [])
```

```
tokenizer.fit_on_texts(vocab)
```

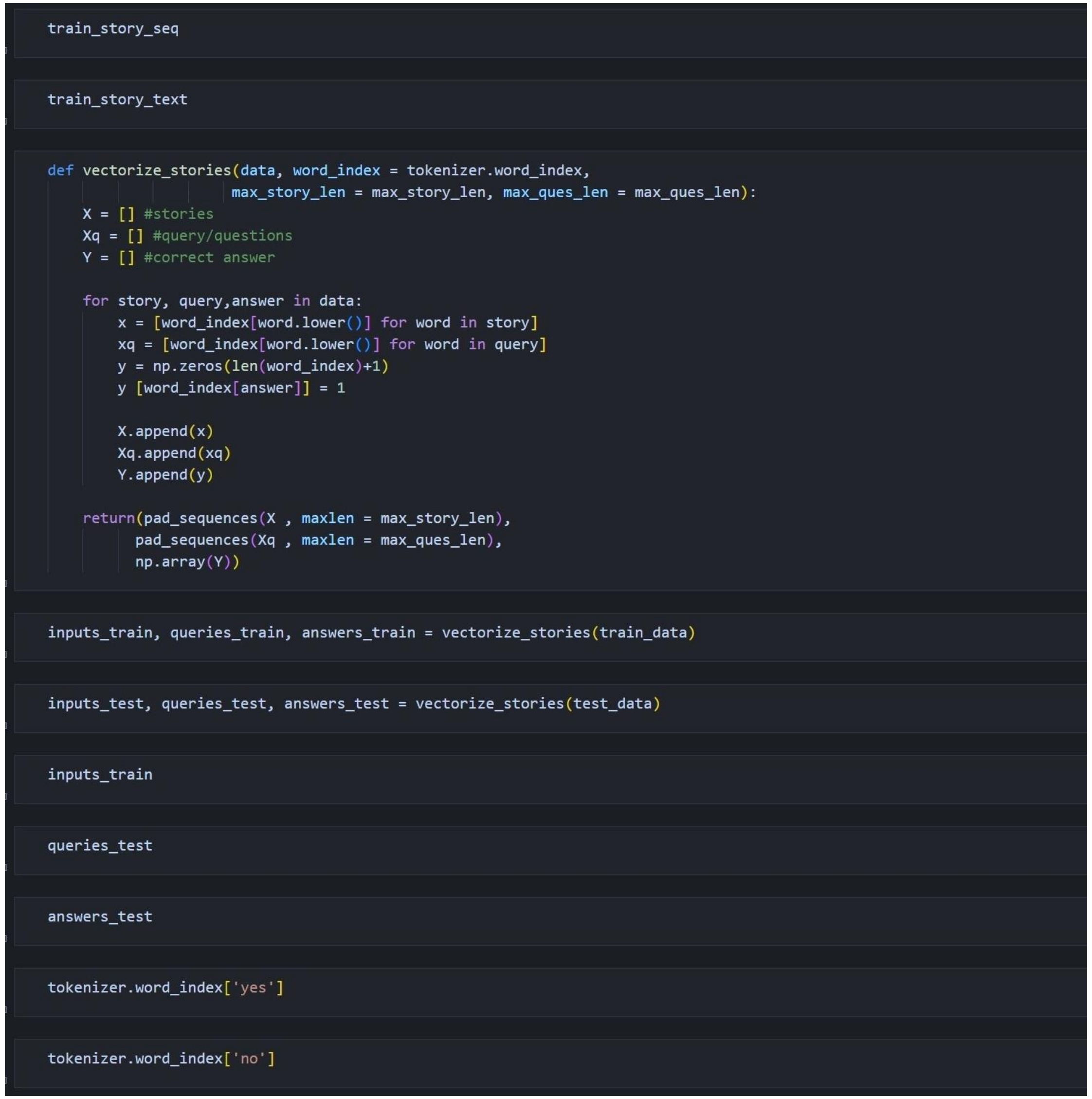
```
tokenizer.word_index
```

```
train_story_text = []
train_question_text = []
train_answers = []


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

```
train_story_seq = tokenizer.texts_to_sequences(train_story_text)
```

```
len(train_story_text)
```

```
len(train_story_seq)
```



```
train_story_seq
```

```
train_story_text
```

```
def vectorize_stories(data, word_index = tokenizer.word_index,
                     max_story_len = max_story_len, max_ques_len = max_ques_len):
    X = [] #stories
    Xq = [] #query/questions
    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))
```

```
inputs_train, queries_train, answers_train = vectorize_stories(train_data)
```

```
inputs_test, queries_test, answers_test = vectorize_stories(test_data)
```

```
inputs_train
```

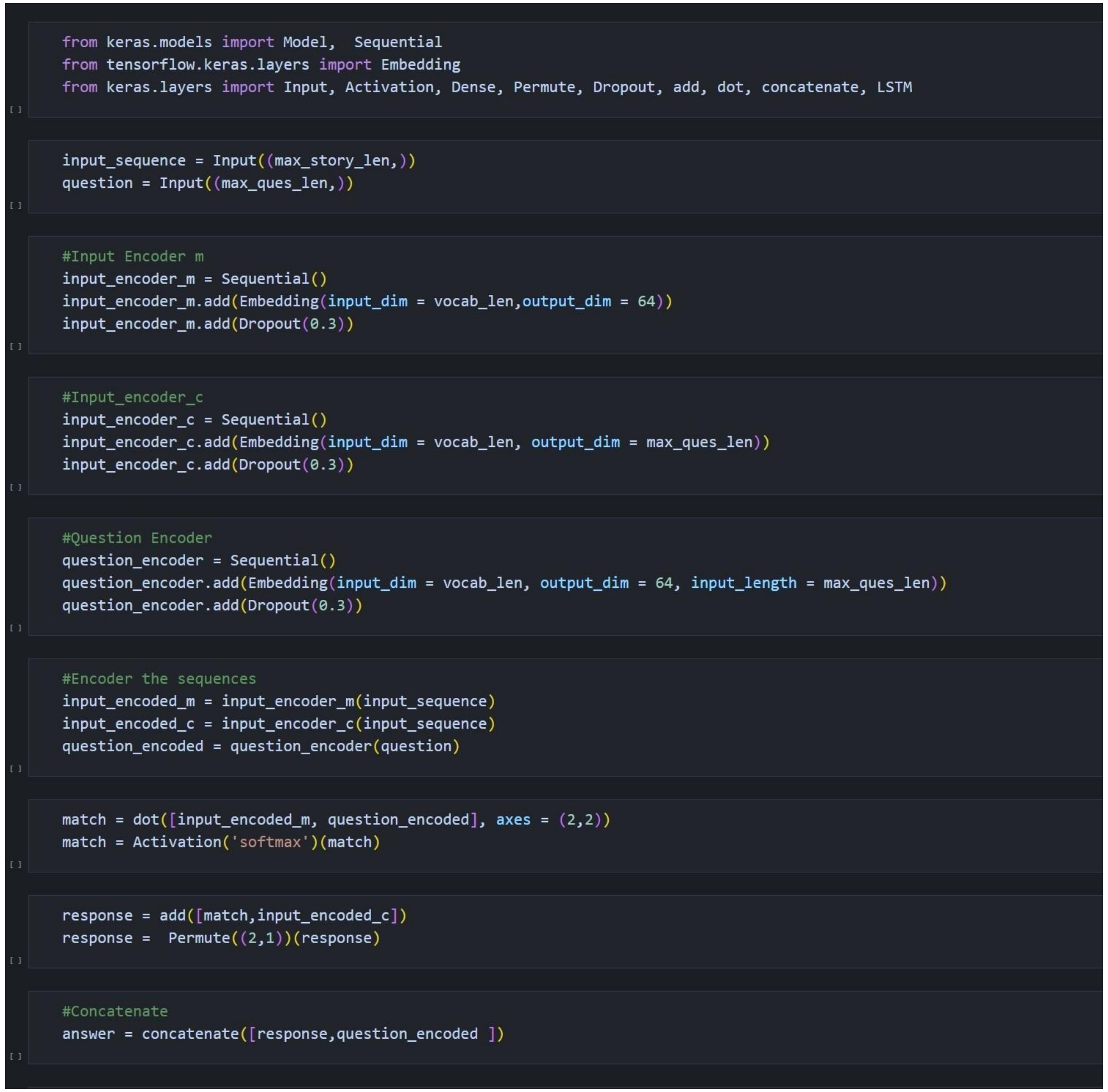
```
queries_test
```

```
answers_test
```

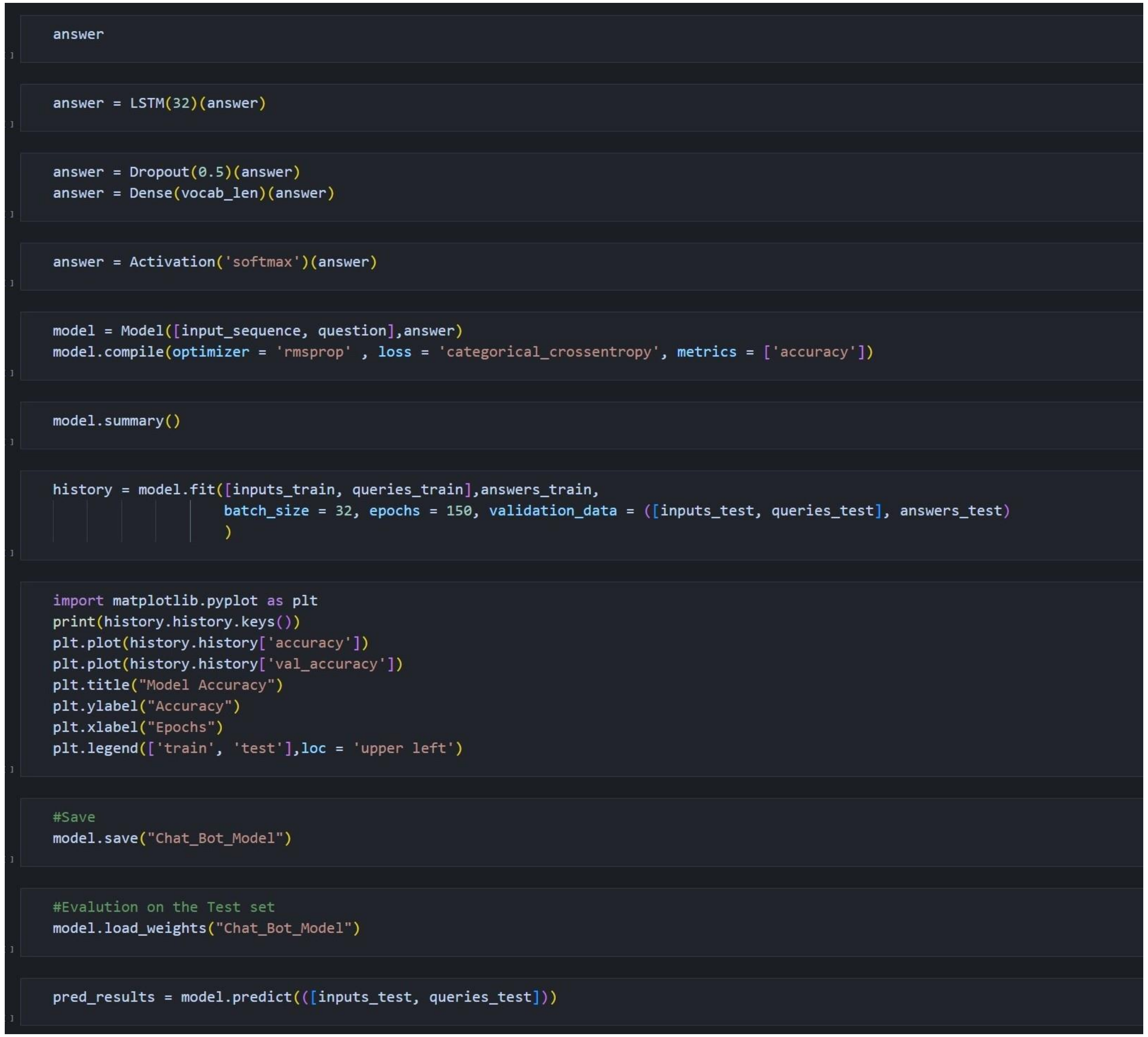
```
tokenizer.word_index['yes']
```

```
tokenizer.word_index['no']
```

CODING:

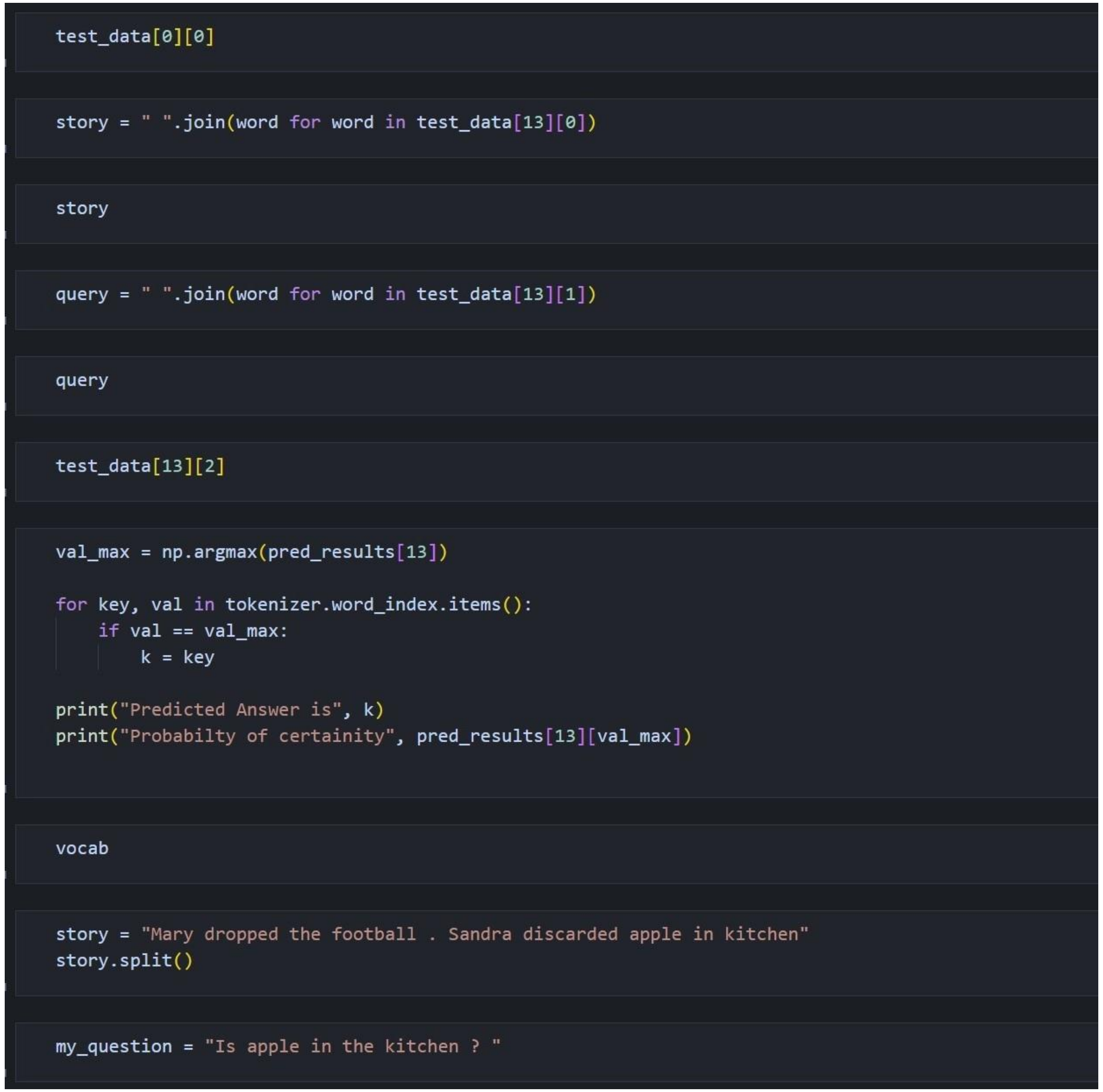


```
from keras.models import Model,  Sequential
from tensorflow.keras.layers import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout, add, dot, concatenate, LSTM

input_sequence = Input((max_story_len,))
question = Input((max_ques_len,))

#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))

#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))

#Question Encoder
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim = vocab_len, output_dim = 64, input_length = max_ques_len))
question_encoder.add(Dropout(0.3))

#Encoder the sequences
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)

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

response = add([match,input_encoded_c])
response =  Permute((2,1))(response)

#Concatenate
answer = concatenate([response,question_encoded ])
```



```
answer

answer = LSTM(32)(answer)

answer = Dropout(0.5)(answer)
answer = Dense(vocab_len)(answer)

answer = Activation('softmax')(answer)

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

model.summary()

history = model.fit([inputs_train, queries_train],answers_train,
                    batch_size = 32, epochs = 150, validation_data = ([inputs_test, queries_test], answers_test)
                    )

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.legend(['train', 'test'],loc = 'upper left')

#Save
model.save("Chat_Bot_Model")

#Evalution on the Test set
model.load_weights("Chat_Bot_Model")

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

CODING:



```
test_data[0][0]
```

```
story = " ".join(word for word in test_data[13][0])
```

```
story
```

```
query = " ".join(word for word in test_data[13][1])
```

```
query
```

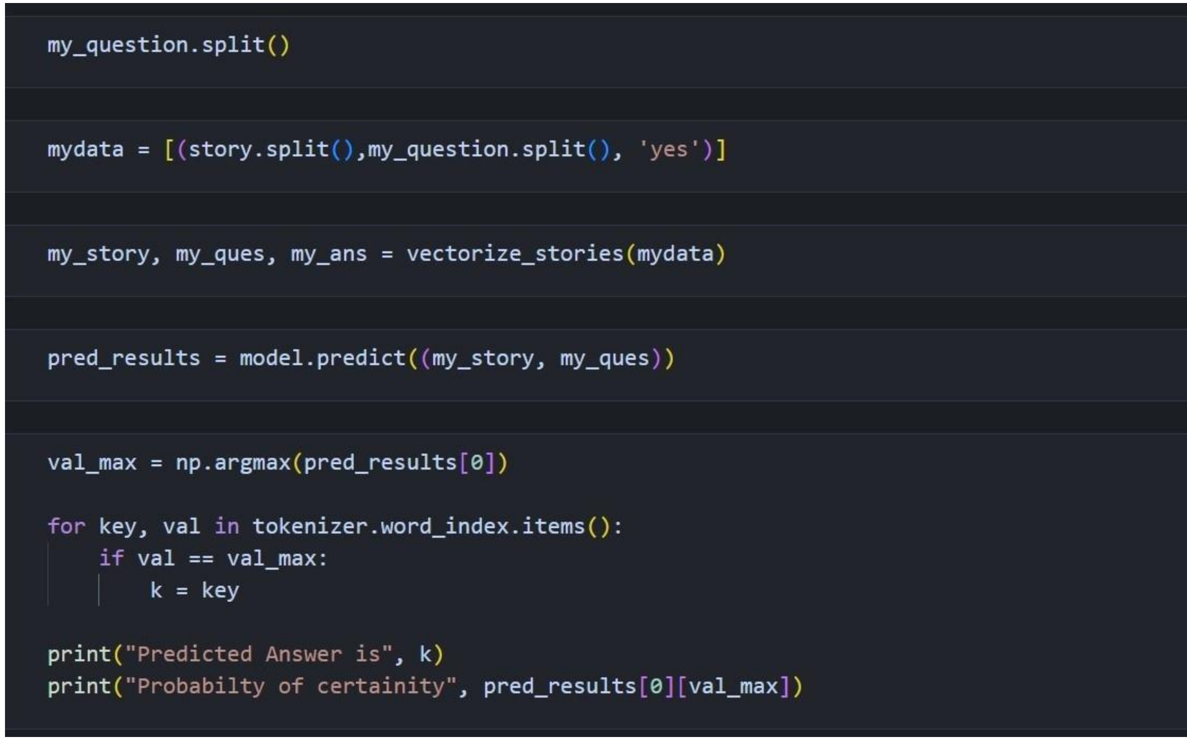
```
test_data[13][2]
```

```
val_max = np.argmax(pred_results[13])

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

print("Predicted Answer is", k)
print("Probabilty of certainity", pred_results[13][val_max])
```

```
vocab
```

```
story = "Mary dropped the football . Sandra discarded apple in kitchen"
story.split()
```

```
my_question = "Is apple in the kitchen ? "
```



```
my_question.split()
```

```
mydata = [(story.split(),my_question.split(), 'yes')]
```

```
my_story, my_ques, my_ans = vectorize_stories(mydata)
```

```
pred_results = model.predict(([my_story, my_ques]))
```

```
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("Probabilty of certainity", pred_results[0][val_max])
```

CONCLUSION:

Hence, we can conclude that in this project we have made use of different concepts of AI and created a Chatbot using Python.

Chatbot with simulated intelligence can deliver a highly personalized conversational experience to the user while minimizing the workload of healthcare provider's team. This blog is encompassing various aspects of healthcare Chatbot development. It also explains how a thoughtfully developed Chatbot can be a right fit for all delivering the value to the community.

This project took us through various phases of learning and project development, also we have gained a lot of new technical and non-technical knowledge.

THANK YOU!