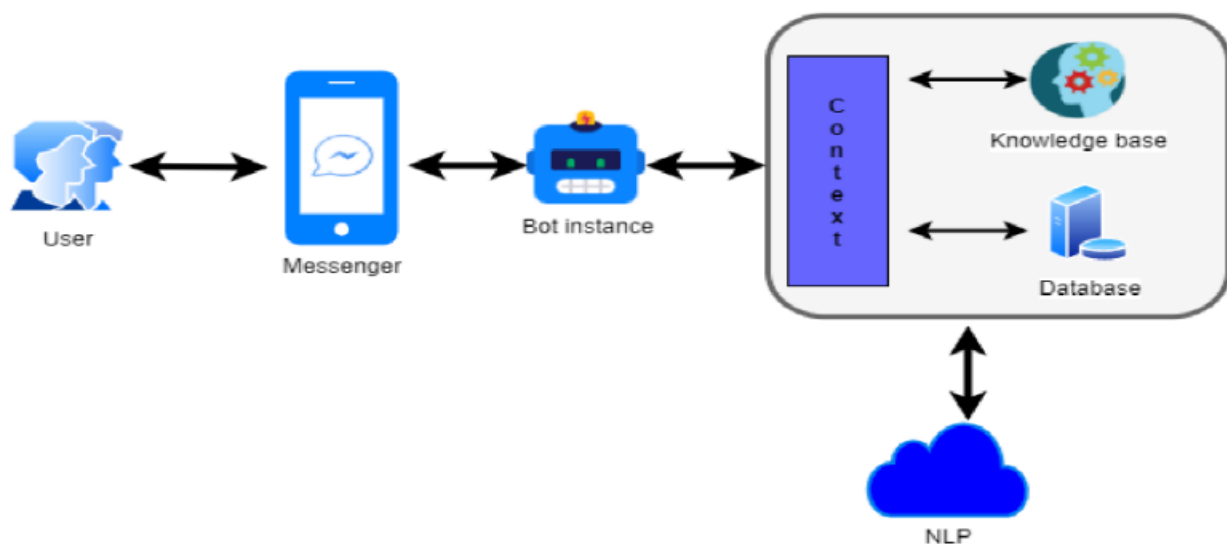




CREATE A CHATBOT IN PYTHON

HAPPYBOT



THARUNA SELVI.R
820421205076

PHASE-5 PROJECT DOCUMENTATION & SUBMISSION

SYNOPSIS

- ❖ Introduction
- ❖ Objectives
- ❖ Design thinking
- ❖ Dataset link
- ❖ intents. json
- ❖ Libraries and Languages
- ❖ Data preparation
- ❖ Data preprocessing
- ❖ Innovation techniques
 - Deep learning
 - Nltk
 - Ensemble learning
- ❖ Feature selection
 - Regression
 - Model training
- ❖ Model evaluation
- ❖ Model comparison
- ❖ Feature Engineering
- ❖ Various features to perform model training
- ❖ Data visualization
- ❖ Flask
- ❖ Source code
- ❖ Output
- ❖ Benefits of chatbot
- ❖ Conclusion

INTRODUCTION:

CHATBOT:

- AI chatbot is a software application that can have a conversation with a human using artificial intelligence.
- Chatbots can make it easy for users to find information by instantaneously responding to questions and requests—through text input ,without the need for human intervention or manual research.
- Modern chatbots increasingly use conversational AI techniques like natural language processing (NLP) to understand the user's questions and automate responses to them.
- Example: ChatGPT, Apple's Siri and Amazon Alexa

OBJECTIVE:

- Create a powerful chatbot in Python by leveraging NLTK for natural language processing, TensorFlow for deep learning, bagging for model ensemble, and Flask for seamless web integration.
- This combination of cutting-edge technologies will enable you to build an intelligent and interactive chatbot that can understand and respond to user input effectively.

DESIGN THINKING:

1.Functionality :

The chatbot's scope encompasses answering common diabetes-related inquiries, offering guidance on lifestyle choices, dietary habits, and exercise routines conducive to diabetes prevention.

2.User Interface :

The chatbot will be integrated website and designed a user-friendly interface for interactions.

3.Natural Language Processing (NLP):

- Implemented NLP techniques to understand and process user input in a conversational manner.
- The code focuses on training a neural network-based chatbot using TensorFlow and Keras for natural language understanding and generation.

4.Responses:

I display responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.

5.Integration:

I display how the chatbot will be integrated with the website .

6.Testing and Improvement:

I tested and improved the chatbot's performance based on user interactions.

DATASET:

<https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot>

A	
1	dialogs
2	hi how are you doing? i'm fine. how about yourself?
3	i'm fine. how about yourself? i'm pretty good. thanks for asking.
4	i'm pretty good. thanks for asking. no problem. so how have you been?
5	no problem. so how have you been? i've been great. what about you?
6	i've been great. what about you? i've been good. i'm in school right now.
7	i've been good. i'm in school right now. what school do you go to?
8	what school do you go to? i go to pcc.
9	i go to pcc. do you like it there?
10	do you like it there? it's okay. it's a really big campus.
11	it's okay. it's a really big campus. good luck with school.
12	good luck with school. thank you very much.
13	how's it going? i'm doing well. how about you?
14	i'm doing well. how about you? never better thanks.
15	never better thanks. so how have you been lately?
16	so how have you been lately? i've actually been pretty good. you?
17	i've actually been pretty good. you? i'm actually in school right now.
18	i'm actually in school right now. which school do you attend?
19	which school do you attend? i'm attending pcc right now.
20	i'm attending pcc right now. are you enjoying it there?
21	are you enjoying it there? it's not bad. there are a lot of people there.
22	it's not bad. there are a lot of people there. good luck with that.
23	good luck with that. thanks.
24	how are you doing today? i'm doing great. what about you?
25	i'm doing great. what about you? i'm absolutely lovely thank you.
26	i'm absolutely lovely thank you. everything's been good with you?
27	everything's been good with you? i haven't been better. how about yourself?
28	i haven't been better. how about yourself? i started school recently.
29	i started school recently. where are you going to school?
30	where are you going to school? i'm going to pcc.
31	i'm going to pcc. how do you like it so far?
32	how do you like it so far? i like it so far. my classes are pretty good right now.
33	i like it so far. my classes are pretty good right now. i wish you luck.
34	it's an ugly day today. i know. i think it may rain.
35	i know. i think it may rain. it's the middle of summer it shouldn't rain today.
36	it's the middle of summer it shouldn't rain today. that would be weird.
37	that would be weird. yeah especially since it's ninety degrees outside.

Intents.json:

SAMPLE:

```
{ "intents": [
  {
    "tag": "news",
    "patterns": [
      "I missed the TV news last night. What was on?",
      "What's the weather going to be like this weekend?",
      "What was the lead story on the news?",
      "A bull chased a man in a supermarket."
    ],
    "responses": [
      "Nothing that would pass as news.",
      "I don't know. Whenever the weather comes on, I switch channels.",
      "Some actress was in court for driving without a license.",
      "Some actor married a woman young enough to be his daughter."
    ]
  }
]
```

LANGUAGE AND LIBRARY

LANGUAGE USED:

- PYTHON

Python is a widely used programming language for various tasks, including web development and artificial intelligence. In this code, Python is used for the overall project development, data preprocessing, model training, and saving/loading the model and related components.

- HTML

It defines the elements and their arrangement on a web page, including headings, paragraphs, images, links, forms, and more. IT used to add interactivity and behavior to web pages.

- JAVASCRIPT

It allows you to respond to user actions, such as clicks, form submissions, and mouse movements and update the page content dynamically.

- CSS

CSS is used to control the presentation and layout of web pages, including fonts, colors, spacing and positioning of elements

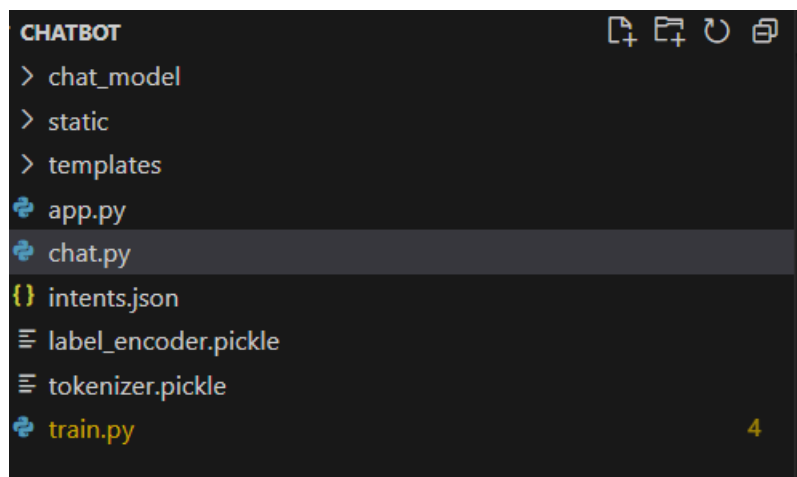
- JSON

JSON is a common format for structuring data in web applications and APIs, making it easy to exchange data between the front end (JavaScript) and back end (Python) components of a web-based chatbot.

LIBRARY AND VERSION USED IN PYTHON:

LIBRARY	VERSION
• tensorflow	2.3.1
• nltk	2.3.1
• colorama	0.4.3
• numpy	1.18.5
• scikit_learn	0.23.2

STRUCTURE OF PROJECT:



DATA PREPARATION:

Data preparation is the process of cleaning and transforming raw data prior to processing and analysis. It is an important step prior to processing and often involves reformatting data, making corrections to data, and combining datasets to enrich data.



with open('intents.json') as file:

```
data = json.load(file)
```

```
training_sentences = []
```

```
training_labels = []
```

```
labels = []
```

```
responses = []
```

```
for intent in data['intents']:
```

```
    for pattern in intent['patterns']:
```

```
        training_sentences.append(pattern)
```

```
        training_labels.append(intent['tag'])
```

```
responses.append(intent['responses'])
```

```
    if intent['tag'] not in labels:
```

```
        labels.append(intent['tag'])
```

```
num_classes = len(labels)
```

In the above coding, The variable “*training_sentences*” holds all the training data (which are the sample messages in each intent category) and the “*training_labels*” variable holds all the target labels correspond to each training data.

Then we use “*LabelEncoder()*” function provided by scikit-learn to convert the target labels into a model understandable form.

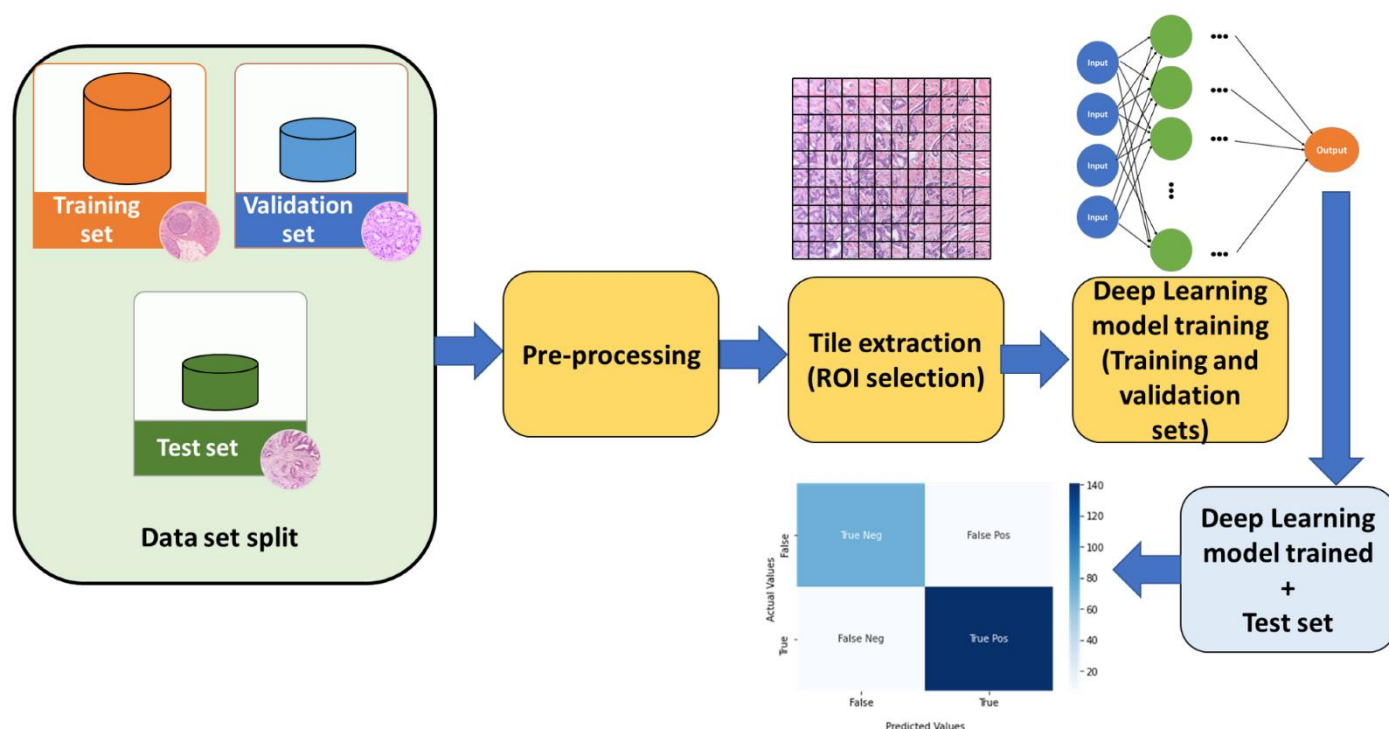
```
lbl_encoder = LabelEncoder()
```

```
lbl_encoder.fit(training_labels)
```

```
training_labels = lbl_encoder.transform(training_labels)
```

DATA PREPROCESSING:

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.



```
vocab_size = 1000
```

```
embedding_dim = 16
```

```
max_len = 20
```

```
oov_token = "<OOV>"
```



```
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)
```

```
tokenizer.fit_on_texts(training_sentences)
```

```
word_index = tokenizer.word_index
```

```
sequences = tokenizer.texts_to_sequences(training_sentences)
```

```
padded_sequences = pad_sequences(sequences, truncating='post', maxlen=max_len)
```

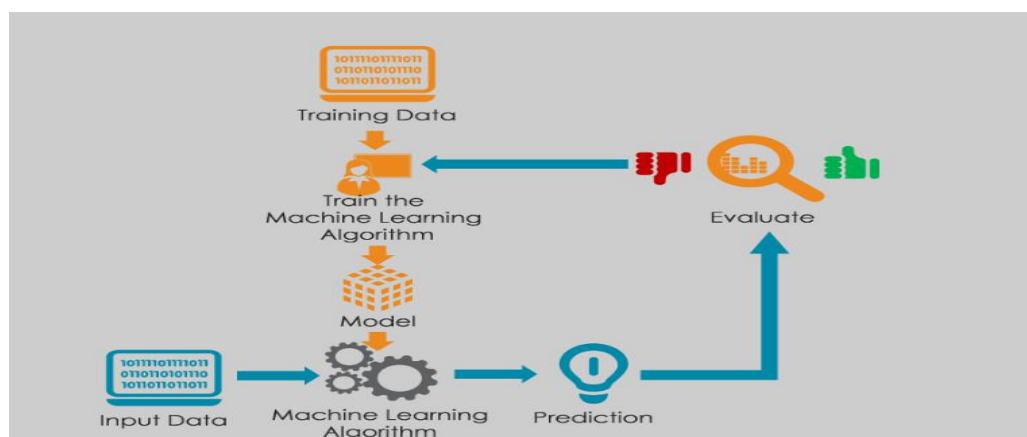
I use this TOKENIZER for the text pre-processing task, by default all punctuations will be removed, turning the texts into space-separated sequences of words, and these sequences are then split into lists of tokens. They will then be indexed or vectorized. We can also add “oov_token” which is a value for “out of token” to deal with out of vocabulary words(tokens) at inference time.

COLAB LINK:

<https://colab.research.google.com/drive/14hpZeycad8evPVR5rzi-gtNvrzdLhgJr?usp=sharing>

INNOVATION TECHNIQUE:

- Innovative techniques in AI refer to novel and creative approaches employed to solve complex problems and improve AI systems' efficiency.
- This includes advancements in machine learning algorithms, neural networks, natural language processing, reinforcement learning, and computer vision.
- These techniques drive AI innovation, enabling the development of smarter, more adaptive, and efficient AI applications for various domains.



1.DEEP LEARNING:

Deep learning is a subset of machine learning that uses neural networks with multiple layers to simulate human brain behavior and learn from large amounts of data. While a single layer can make approximate predictions, additional hidden layers can optimize and refine accuracy

NEURAL NETWORK:

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another

Neural Network architecture for the proposed model and for that we use the “*Sequential*” model class of **Keras**

ReLU ACTIVATOR:

A rectified linear unit (ReLU) is an activation function that introduces the property of non linearity to a deep learning model and solves the vanishing gradients issue.

```
model = Sequential()

model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))

model.add(GlobalAveragePooling1D())

model.add(Dense(16, activation='relu'))

model.add(Dense(16, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',

              optimizer='adam', metrics=['accuracy'])

model.summary()
```

```
PS C:\Users\Withish\Desktop\project\naan mudhalvan\CHATBOT> python train.py
2023-11-01 19:44:48.351689: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 20, 16)	16000
global_average_pooling1d (GlobalAveragePooling1D)	(None, 16)	0
dense (Dense)	(None, 16)	272
dense_1 (Dense)	(None, 16)	272
dense_2 (Dense)	(None, 25)	425

```
=====
Total params: 16969 (66.29 KB)
Trainable params: 16969 (66.29 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
Epoch 1/500 [=====] - 1s 7ms/step - loss: 3.2192 - accuracy: 0.0431
A/A [=====] - 0s 3ms/step - loss: 3.2175 - accuracy: 0.0431
Epoch 2/500 [=====] - 0s 3ms/step - loss: 3.2163 - accuracy: 0.0431
A/A [=====] - 0s 3ms/step - loss: 3.2153 - accuracy: 0.0431
Epoch 3/500 [=====] - 0s 3ms/step - loss: 3.2140 - accuracy: 0.0431
A/A [=====] - 0s 6ms/step - loss: 3.2128 - accuracy: 0.0431
Epoch 4/500 [=====] - 0s 6ms/step - loss: 3.2115 - accuracy: 0.0431
A/A [=====] - 0s 4ms/step - loss: 3.2103 - accuracy: 0.0431
Epoch 5/500 [=====] - 0s 4ms/step - loss: 3.2088 - accuracy: 0.0431
A/A [=====] - 0s 6ms/step - loss: 3.2074 - accuracy: 0.0517
Epoch 6/500 [=====] - 0s 5ms/step - loss: 3.2057 - accuracy: 0.0517
A/A [=====] - 0s 4ms/step - loss: 3.2042 - accuracy: 0.0517
Epoch 7/500 [=====] - 0s 5ms/step - loss: 3.2023 - accuracy: 0.0517
A/A [=====] - 0s 3ms/step - loss: 3.2006 - accuracy: 0.0517
Epoch 8/500 [=====] - 0s 3ms/step - loss: 3.1982 - accuracy: 0.0517
A/A [=====] - 0s 6ms/step - loss: 3.1961 - accuracy: 0.0517
Epoch 9/500 [=====] - 0s 6ms/step - loss: 3.1935 - accuracy: 0.0517
A/A [=====] - 0s 3ms/step - loss: 3.1910 - accuracy: 0.0517
Epoch 10/500 [=====] - 0s 4ms/step - loss: 3.1881 - accuracy: 0.0517
A/A [=====] - 0s 6ms/step - loss: 3.1852 - accuracy: 0.0517
```

call the “*fit*” method with training data and labels.

epochs = 500

history = model.fit(padded_sequences, np.array(training_labels), epochs=epochs)

After running this “train.py”. To save all the required files in order to use it at the inference time. So that we save the trained model, fitted tokenizer object and fitted label encoder object.

model.save("chat_model")

import pickle

with open('tokenizer.pickle', 'wb') as handle:

```
pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

with open('label_encoder.pickle', 'wb') as ecn_file:

```
pickle.dump(lbl_encoder, ecn_file, protocol=pickle.HIGHEST_PROTOCOL)
```

2. NATURAL LANGUAGE(NLTK):

The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP).

The code you provided does not explicitly use the Natural Language Toolkit (NLTK), a popular Python library for natural language processing and text analysis. NLTK is used for tasks like tokenization, stemming, lemmatization, and part-of-speech tagging. To include NLTK in your project, you need to import and use its functions for text preprocessing and analysis. If you need NLTK for specific tasks like text preprocessing or feature extraction, you can use it alongside existing code to enhance the chatbot's functionality.

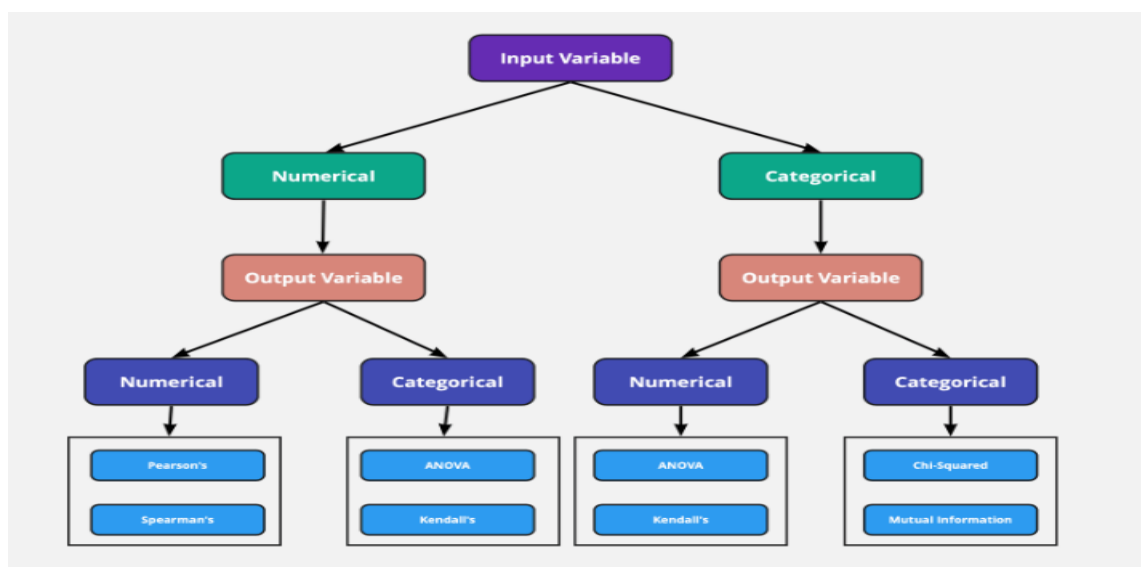
3. ENSEMBLE LEARNING:

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem.

It used in machine learning to improve the performance and robustness of models, including those used in chatbots.

FEATURE SELECTION:

Feature selection is a process that chooses a subset of features from the original features so that the feature space is optimally reduced according to a certain criterion.



```
from sklearn.preprocessing import LabelEncoder

chatbot_data = [

    "i'm fine. how about yourself?", "greeting",

    "i'm doing well. how about you?", "greeting",

    "you like the rain?", "weather",

    "where's your money?", "finance",

    "did you tell her about school?", "school",

    "thank you very much.", "greeting",

    "what do you want to do?", "personal",

    "why do you like it?", "personal",

    text_messages = chatbot_data[:,2]

    labels = chatbot_data[:,1]

    label_encoder = LabelEncoder()

    y = label_encoder.fit_transform(labels)

    df = pd.DataFrame({'text': text_messages, 'label': y})

    X = df['text']

    y = df['label']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    tfidf_vectorizer = TfidfVectorizer()

    X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

    X_test_tfidf = tfidf_vectorizer.transform(X_test)

    model = xgb.XGBClassifier()

    model.fit(X_train_tfidf, y_train)

    y_pred = model.predict(X_test_tfidf)

    accuracy = accuracy_score(y_test, y_pred)
```

```
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:\n", report)
```

In above code, The code uses a chatbot dataset to train an XGBoost classifier for text classification. LabelEncoder is used to encode text messages and labels, which are then split into training and testing sets. TfidfVectorizer converts text data into numerical vectors, and the model's accuracy and classification report are evaluated.

OUTPUT:

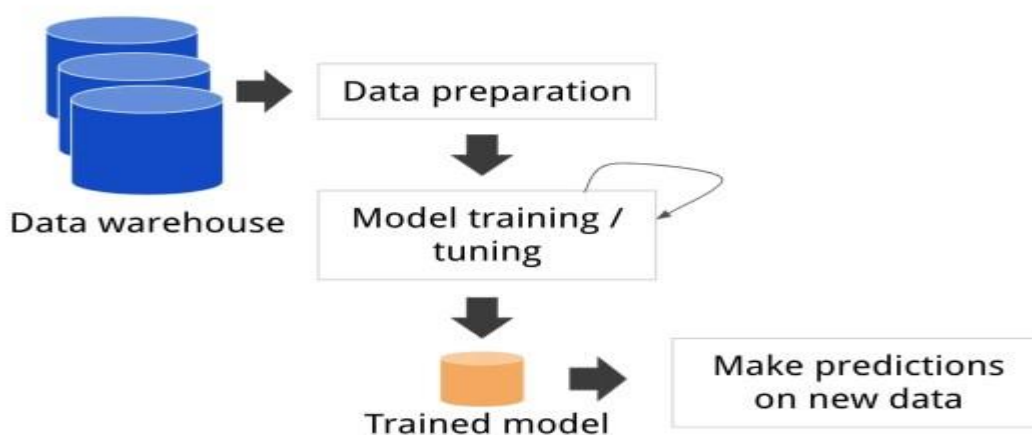
```
Accuracy: 0.00
Classification Report:
      precision    recall  f1-score   support

     0       0.00      0.00      0.00        0.0
     1       0.00      0.00      0.00        2.0

 accuracy
macro avg      0.00      0.00      0.00        2.0
weighted avg    0.00      0.00      0.00        2.0
```

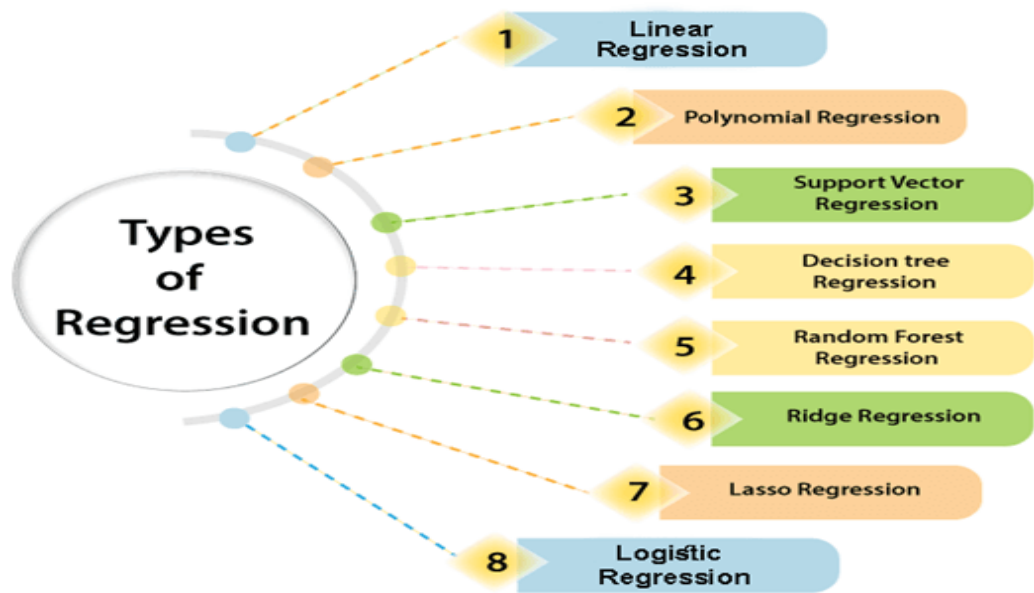
MODEL TRAINING:

Model training refers to the process of allowing a machine learning algorithm to automatically learn patterns based on data. These patterns are statistically learned by observing which signals makes an answer correct or incorrect (supervised learning) or by discovering the inherent patterns in data without being told the correct answers (unsupervised learning).



1.REGRESSION:

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables.



In the below link I include all the coding and output

COLAB LINK:

https://colab.research.google.com/drive/1yuGqIX_CUKzbCymnR9nZxKB-WVlxQGTv?usp=sharing

MODEL EVALUATION:

- Model evaluation is the process that uses some metrics which help us to analyze the performance of the model.
- Evaluating a model plays a vital role so that we can judge the performance of our model. The evaluation also helps to analyze a model's key weaknesses.
- FLOWCHAT:



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

```
plt.figure(figsize=(8, 6))
```

```
plt.scatter(y_test, predictions1, color='blue', label='Actual vs. Predicted')
```

```
plt.plot([0, 5], [0, 5], color='red', linestyle='--', label='Ideal Line')
```

```
plt.title('Actual vs. Predicted Response Time')
```

```
plt.xlabel('Actual Response Time')
```

```
plt.ylabel('Predicted Response Time')
```

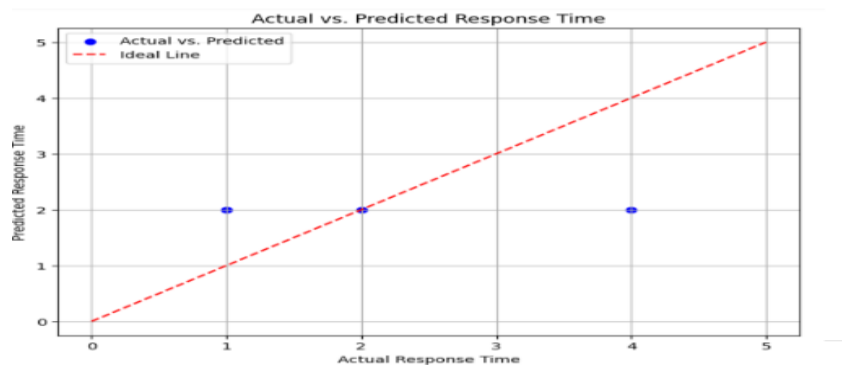
```
plt.legend()
```

```
plt.grid()
```

```
plt.show()
```

This code generates a scatter plot to compare the actual response time values (`y_test`) against the predicted values (`predictions1`). The **blue** points represent the actual versus predicted response times for each data point. The **red** dashed line represents the ideal scenario where actual and predicted values perfectly match.

OUTPUT:



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

```
from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns
```

```
conf_matrix = confusion_matrix(y_test, predictions2)
```

```
plt.figure(figsize=(12, 5))
```

```
plt.subplot(1, 2, 1)
```

```
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
```



```

plt.xlabel("Predicted Labels")

plt.ylabel("Actual Labels")

plt.title("Confusion Matrix")

plt.subplot(1, 2, 2)

plt.scatter(range(len(predictions2)), predictions2, label='Predicted Labels', c='blue',
alpha=0.5)

plt.scatter(range(len(y_test)), y_test, label='Actual Labels', c='green', alpha=0.5)

plt.xlabel("Sample Index")

plt.ylabel("Label")

plt.title("Scatter Plot of SVM Predictions vs. Actual Labels")

plt.legend(loc='upper right')

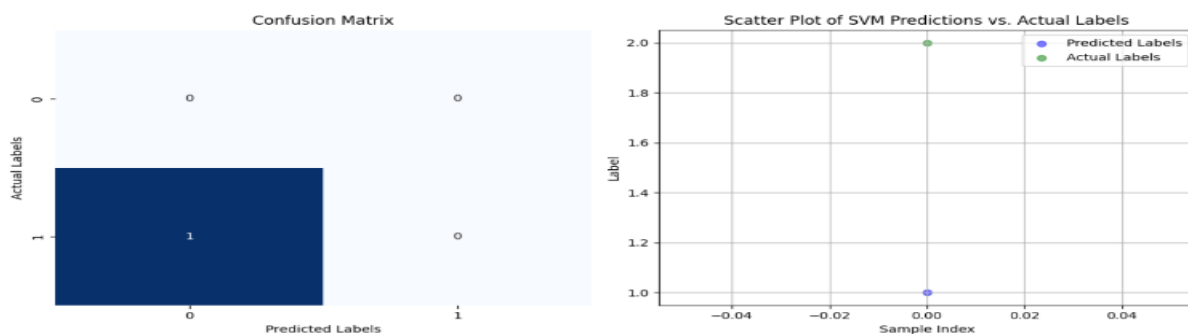
plt.grid(True)

plt.show()

```

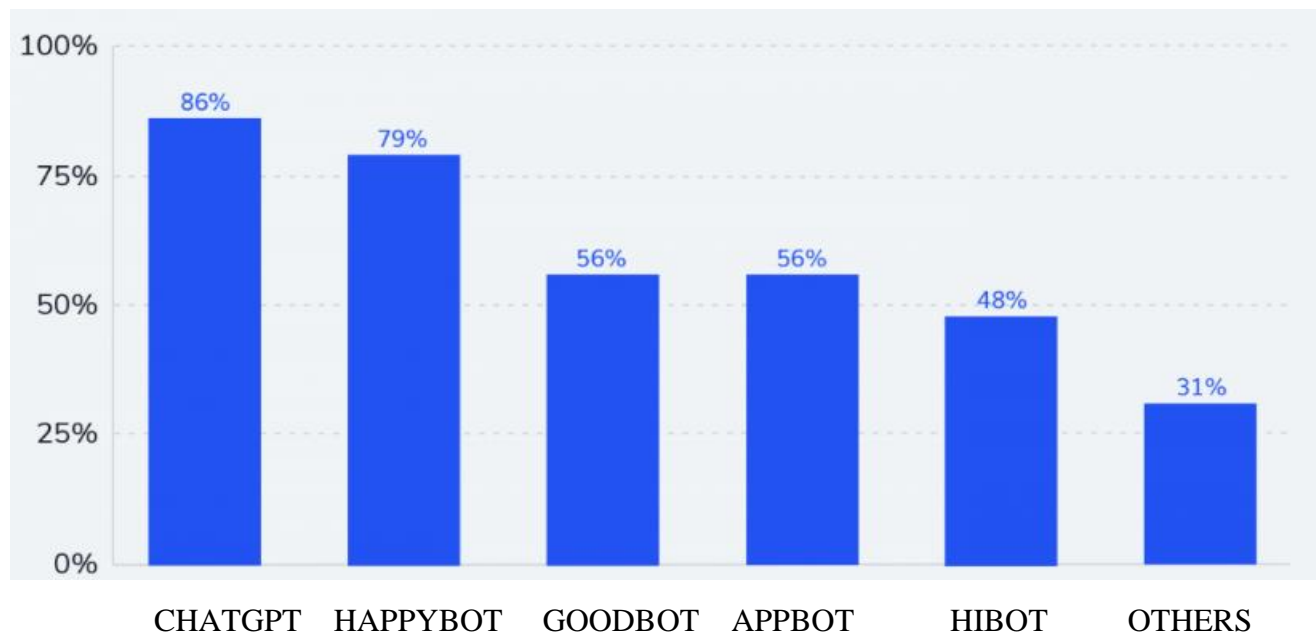
This code generates and visualizes a confusion matrix for evaluating the SVM model's classification performance. It also creates a scatter plot comparing predicted and actual labels for further analysis and understanding of the model's predictions.

OUTPUT:



MODEL COMPARISON:

Model comparison in AI involves evaluating multiple machine learning or deep learning algorithms on a specific task. Metrics like accuracy, precision, are used to assess performance. Comparative analysis helps identify the most effective algorithm for a given problem, aiding in informed decision-making and improved system performance in real-world applications.



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import PolynomialFeatures
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
np.random.seed(0)
X = np.random.rand(100, 1)
```

```

y = 2 * X.squeeze() + 1 + 0.1 * np.random.randn(100)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

models = {

"Linear Regression": LinearRegression(),

"Ridge Regression": Ridge(alpha=1.0),

"Lasso Regression": Lasso(alpha=1.0),

"Polynomial Regression": Pipeline([

("poly_features", PolynomialFeatures(degree=2)),

("linear_regression", LinearRegression())

]),

"SVM": SVR(),

"Random Forest Regression": RandomForestRegressor(n_estimators=100),

"XGBoost Regression": XGBRegressor()

}

mse_values = {}

for model_name, model in models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)

    mse_values[model_name] = mse

mse_df = pd.DataFrame.from_dict(mse_values, orient='index', columns=['MSE'])

mse_df = mse_df.sort_values(by='MSE')

print("MSE Comparison Table:")

print(mse_df)

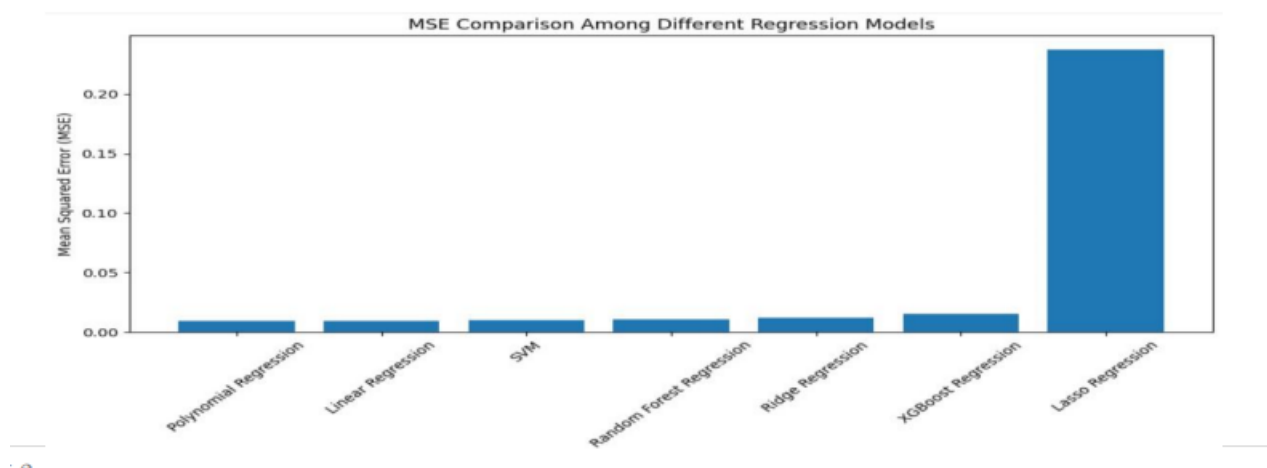
```

OUTPUT:

MSE Comparison Table:	
	MSE
Polynomial Regression	0.009116
Linear Regression	0.009178
SVM	0.009697
Random Forest Regression	0.010862
Ridge Regression	0.011812
XGBoost Regression	0.015321
Lasso Regression	0.237121

```
plt.figure(figsize=(10, 6))  
  
plt.bar(mse_df.index, mse_df['MSE'])  
  
plt.xlabel('Regression Models')  
  
plt.ylabel('Mean Squared Error (MSE)')  
  
plt.title('MSE Comparison Among Different Regression Models')  
  
plt.xticks(rotation=45)  
  
plt.show()
```

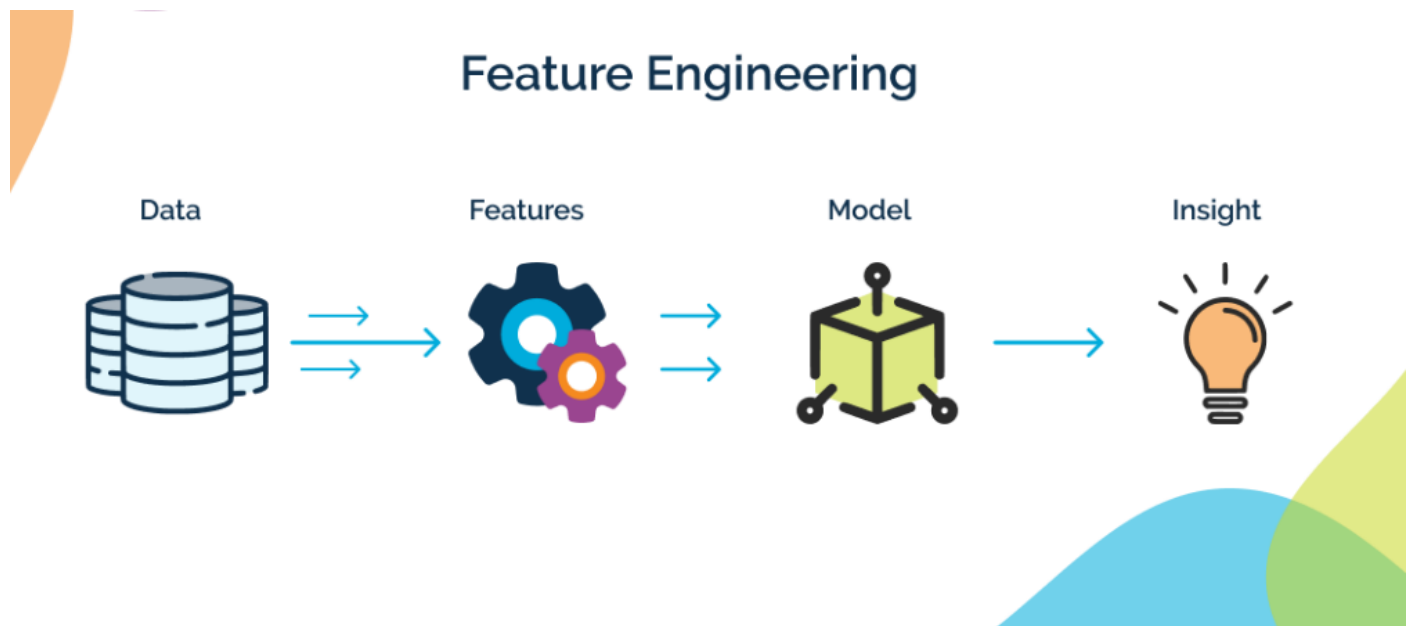
OUTPUT:



FEATURE ENGINEERING:

Feature engineering for chatbots involves transforming input data into structured formats for machine learning algorithms to understand and learn from. This process improves model recognition of patterns, intents, and context, enabling accurate responses. Key

techniques include tokenization, word embeddings, N-grams, and part-of-speech tagging, which help model understanding meaning, context, and grammatical structure of user inputs. "Text length" is an example of Feature engineering.



Tokenization:

Tokenization is the process of breaking down text into smaller units, such as words, subwords, or characters, which is crucial for natural language processing tasks.

Word Embeddings:

Word embeddings, dense vector representations of words, capture semantic relationships, aiding language models in understanding meaning and context. Pre-trained embeddings like Word2Vec, GloVe, or FastText are commonly used.

TF-IDF (Term Frequency-Inverse Document Frequency):

TF-IDF is a numerical statistic that measures the frequency of a word in a document, adjusting for its frequency across multiple documents, aiding in text classification and information retrieval.

N-grams:

N-grams are sequences of N words, characters, or symbols in text, capturing contextual

information beyond individual words, useful for sentiment analysis and language generation tasks.

Part-of-Speech Tagging:

Part-of-speech tagging helps models understand user input grammatical structure by identifying the part of speech (noun, verb, adjective) of each word in a sentence.

ADDING TEXT LENGTH AS A FEATURE

```
import pandas as pd

data = {

'Message': [

    "i'm fine. how about yourself?", "i'm doing well. how about you?", "you like the rain?", "where's your money?", "did you tell her about school?", "thank you very much.", "what do you want to do?", "why do you like it?"]

df = pd.DataFrame(data)

df['TextLength'] = df['Message'].apply(lambda x: len(x.split()))

print(df)}
```

In feature engineering, I use text preprocessing, TD-IDF vectorization, and label encoding for the chatbot dataset.

OUTPUT:

	Message	TextLength
0	i'm fine. how about yourself?	5
1	i'm doing well. how about you?	6
2	you like the rain?	4
3	where's your money?	3
4	did you tell her about school?	6
5	thank you very much.	4
6	what do you want to do?	6
7	why do you like it?	5

```

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.preprocessing import LabelEncoder

data = {

'Message': [

'i'm fine. how about yourself?", "i'm doing well. how about you?", "you like the rain?",

"where's your money?", "did you tell her about school?", "thank you very much.",

"what do you want to do?", "why do you like it?", ],

'Label': ["greeting", "greeting", "weather", "finance", "school", "greeting", "personal",

"personal"]

df = pd.DataFrame(data)

df['Message'] = df['Message'].str.lower()

df['Message'] = df['Message'].str.replace(r'[^w\s]', '')

tfidf_vectorizer = TfidfVectorizer(max_features=100)

X_tfidf = tfidf_vectorizer.fit_transform(df['Message'])

label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(df['Label'])

df_tfidf = pd.DataFrame(X_tfidf.toarray())

df = pd.concat([df_tfidf, pd.DataFrame({'Label': y_encoded})], axis=1)

print(df)

```

In this code, text data is preprocessed, vectorized using TF-IDF, and encoded. The resulting DataFrame combines TF-IDF features and corresponding encoded labels, ready for machine learning model training.

OUTPUT:

```

      0      1      2      3      4      5      6  \
0  0.364907  0.000000  0.000000  0.000000  0.504577  0.000000  0.422875
1  0.353832  0.000000  0.000000  0.489264  0.000000  0.000000  0.410042
2  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
3  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
4  0.331033  0.457738  0.000000  0.000000  0.000000  0.457738  0.000000
5  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
6  0.000000  0.000000  0.680936  0.000000  0.000000  0.000000  0.000000
7  0.000000  0.000000  0.438402  0.000000  0.000000  0.000000  0.000000

      7      8      9  ...      18      19      20      21  \
0  0.422875  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000
1  0.410042  0.000000  0.000000  ...  0.000000  0.000000  0.489264  0.000000
2  0.000000  0.000000  0.487776  ...  0.000000  0.000000  0.000000  0.000000
3  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000
4  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000
5  0.000000  0.000000  0.000000  ...  0.554725  0.000000  0.000000  0.000000
6  0.000000  0.000000  0.000000  ...  0.000000  0.406249  0.000000  0.406249
7  0.000000  0.523104  0.438402  ...  0.000000  0.000000  0.000000  0.000000

      22      23      24      25      26  Label
0  0.000000  0.000000  0.000000  0.000000  0.504577      1
1  0.000000  0.000000  0.244491  0.000000  0.000000      1
2  0.000000  0.000000  0.290840  0.000000  0.000000      4
3  0.57735  0.000000  0.000000  0.57735  0.000000      0
4  0.000000  0.000000  0.228737  0.000000  0.000000      3
5  0.000000  0.000000  0.277202  0.000000  0.000000      1
6  0.000000  0.000000  0.203007  0.000000  0.000000      2
7  0.000000  0.523104  0.261401  0.000000  0.000000      2

[8 rows x 28 columns]
```

VARIOUS FEATURES TO PERFORM MODEL TRAINING:

Chatbot model training uses natural language processing, sentiment analysis, and intent recognition to understand user inputs, extract relevant information, and respond contextually. These features enhance the user's conversational experience by providing Accurate, personalized, and efficient interactions.

```
import pandas as pd
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score
```

```
data = {
```

```
'Message': [
```

```
"i'm fine. how about yourself?", "i'm doing well. how about you?", "you like the rain?", "where's your money?",
```



```
"did you tell her about school?","thank you very much.","what do you want to do?","why do you like it?"),  
'Intent': ["greeting", "greeting", "weather", "finance", "school", "greeting", "personal",  
"personal"]  
  
df = pd.DataFrame(data)  
  
tfidf_vectorizer = TfidfVectorizer()  
  
X = tfidf_vectorizer.fit_transform(df['Message'])  
  
X_train, X_test, y_train, y_test = train_test_split(X, df['Intent'], test_size=0.2,  
random_state=42)  
  
model = RandomForestClassifier(n_estimators=100, random_state=42)  
  
model.fit(X_train, y_train)  
  
y_pred = model.predict(X_test)  
  
accuracy = accuracy_score(y_test, y_pred)  
  
print(f"Accuracy: {accuracy:.2f}")
```

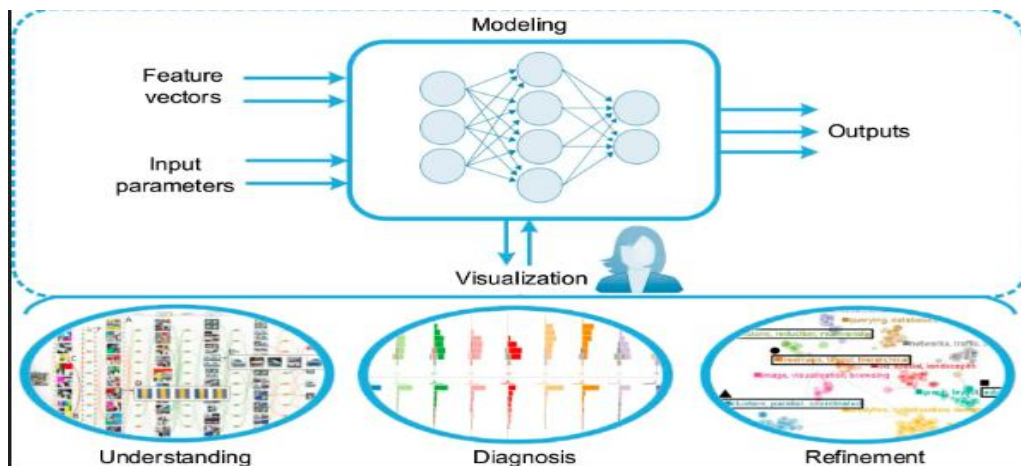
The given code snippet demonstrates text classification using a Random Forest classifier. The code preprocesses text data using TF-IDF vectorization, splits it into training and testing sets, and trains a RandomForestClassifier. It calculates and prints the accuracy score of the model on the test data.

OUTPUT:

Accuracy: 0.50

VISUALIZATION:

- Data visualization is the representation of data through use of common graphics, such as charts, plots, infographics, and even animations.
- These visual displays of information communicate complex data relationships and data-driven insights in a way that is easy to understand.



```
import json

import matplotlib.pyplot as plt

with open('intents.json') as file:

    intents_data = json.load(file)

intent_tags = [intent['tag'] for intent in intents_data['intents']]

intent_counts = [len(intent['patterns']) + len(intent['responses']) for intent in intents_data['intents']]

plt.figure(figsize=(8, 8))

plt.pie(intent_counts, labels=intent_tags, autopct='%1.1f%%', startangle=140)

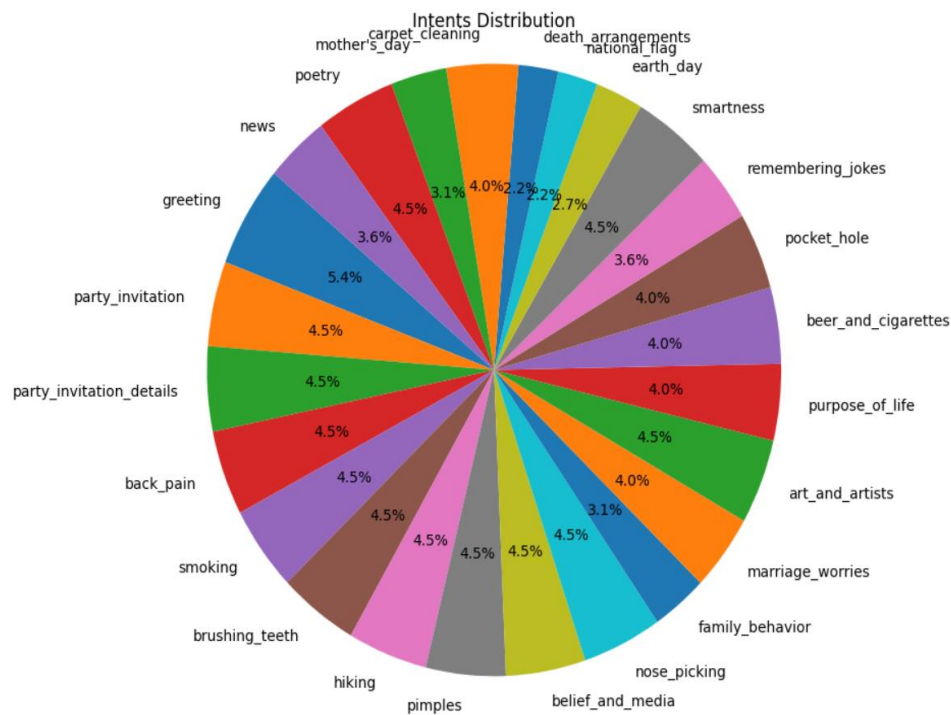
plt.title('Intents Distribution')

plt.axis('equal')

plt.show()
```

This code reads intent data from a JSON file named 'intents.json' and extracts intent tags and their corresponding counts of patterns and responses. It then creates a pie chart where each slice represents

an intent tag, and the size of the slice is proportional to the total count of patterns and responses for that intent. The chart visualizes the distribution of intents in the dataset.



In the below link I include all the coding and output

COLAB LINK:

<https://colab.research.google.com/drive/1VXl1ty-ACTSGOQUB4hYpq8eQWsJppgYs?usp=sharing>

INTERENCE:

It refers to the process of generating responses to user inputs. It is a fundamental aspect of natural language processing (NLP) and machine learning. During inference, the chatbot's underlying model analyzes the input message and predicts an appropriate response based on the patterns and knowledge it has learned from the training data.

while True:

```

print(Fore.LIGHTBLUE_EX + "User: " + Style.RESET_ALL, end="")

inp = input()

if inp.lower() == "quit":

    break

result = model.predict(keras.preprocessing.sequence.pad_sequences(tokenizer.texts_to_sequences([inp]),
                                                                    truncating='post', maxlen=max_len))

tag = lbl_encoder.inverse_transform([np.argmax(result)])

for i in data['intents']:

    if i['tag'] == tag:

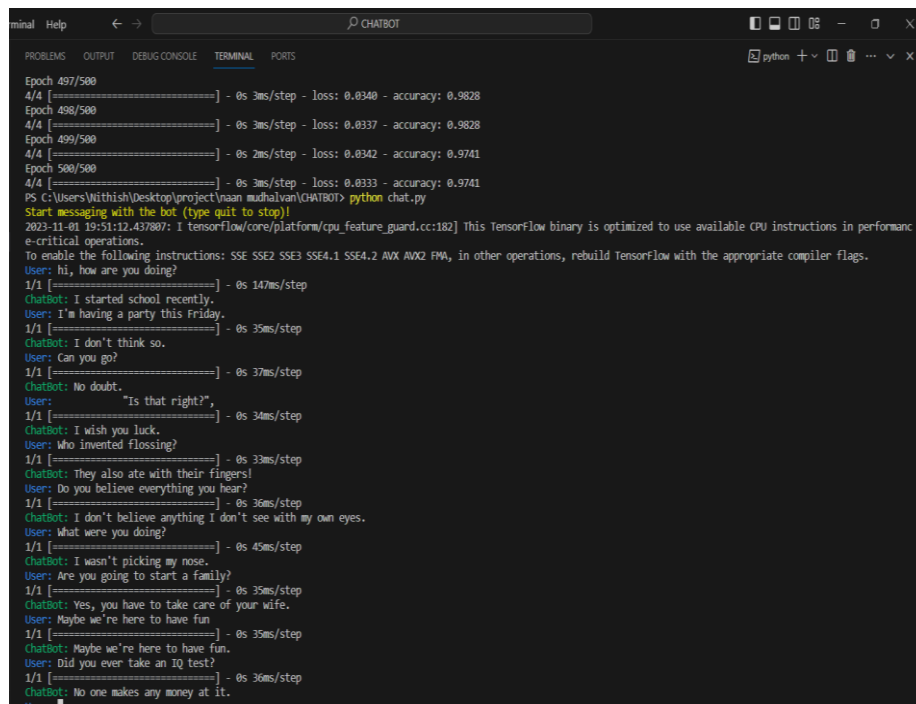
        print(Fore.GREEN + "ChatBot:" + Style.RESET_ALL , np.random.choice(i['responses']))

print(Fore.YELLOW + "Start messaging with the bot (type quit to stop)!" + Style.RESET_ALL)

chat()

```

OUTPUT:



```

Epoch 497/500
4/4 [=====] - 0s 3ms/step - loss: 0.0340 - accuracy: 0.9828
Epoch 498/500
4/4 [=====] - 0s 3ms/step - loss: 0.0337 - accuracy: 0.9828
Epoch 499/500
4/4 [=====] - 0s 2ms/step - loss: 0.0342 - accuracy: 0.9741
Epoch 500/500
4/4 [=====] - 0s 3ms/step - loss: 0.0333 - accuracy: 0.9741
PS C:\Users\Withish\Desktop\project\naan mudhalvan\CHATBOT> python chat.py
Start messaging with the bot (type quit to stop)!
2023-11-01 19:51:12.437807: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performanc
e-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
User: hi, how are you doing?
1/1 [=====] - 0s 147ms/step
ChatBot: I started school recently.
User: I'm having a party this Friday.
1/1 [=====] - 0s 35ms/step
ChatBot: I don't think so.
User: Can you go?
1/1 [=====] - 0s 37ms/step
ChatBot: No doubt.
User: "Is that right?",
1/1 [=====] - 0s 34ms/step
ChatBot: I wish you luck.
User: Who invented flossing?
1/1 [=====] - 0s 33ms/step
ChatBot: They also ate with their fingers!
User: Do you believe everything you hear?
1/1 [=====] - 0s 36ms/step
ChatBot: I don't believe anything I don't see with my own eyes.
User: What were you doing?
1/1 [=====] - 0s 45ms/step
ChatBot: I wasn't picking my nose.
User: Are you going to start a family?
1/1 [=====] - 0s 35ms/step
ChatBot: Yes, you have to take care of your wife.
User: Maybe we're here to have fun
1/1 [=====] - 0s 35ms/step
ChatBot: Maybe we're here to have fun.
User: Did you ever take an IQ test?
1/1 [=====] - 0s 36ms/step
ChatBot: No one makes any money at it.
User:

```

FLASK:

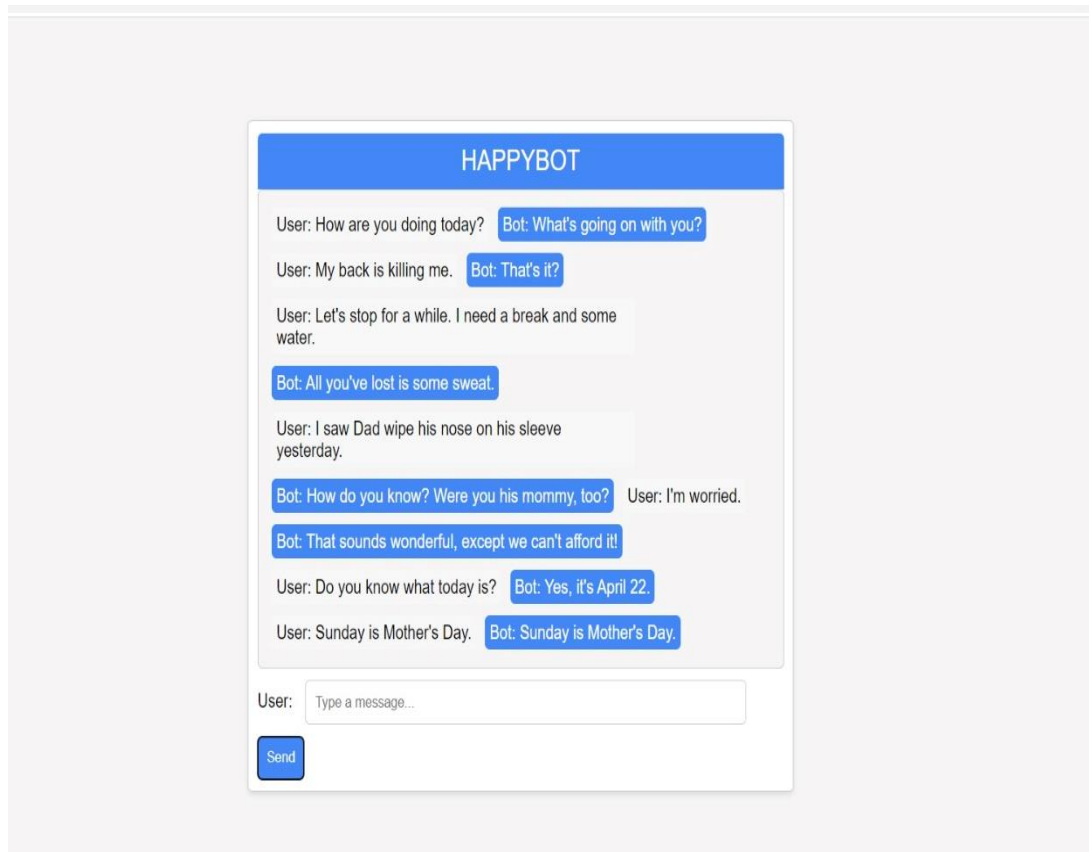
- Flask is a lightweight and versatile web framework for Python.
- It simplifies building web applications and APIs by providing essential tools and libraries.

- Its simplicity and flexibility make it popular for developing web applications and RESTful services in Python.

```
from flask import Flask, render_template, request, jsonify
import json
import numpy as np
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder
import colorama
colorama.init()
from colorama import Fore, Style, Back
import random
import pickle
app = Flask(__name__)
with open("intents.json") as file:
    data = json.load(file)
model = keras.models.load_model('chat_model')
with open('tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)
with open('label_encoder.pickle', 'rb') as enc:
    lbl_encoder = pickle.load(enc)
max_len = 20
@app.route("/")
def home():
    return render_template("index.html")
@app.route("/get_response", methods=["POST"])
def get_response():
    user_input = request.form["user_input"]
    result=
model.predict(keras.preprocessing.sequence.pad_sequences(tokenizer.texts_to_sequences([user_input]),
    truncating='post', maxlen=max_len))
    tag = lbl_encoder.inverse_transform([np.argmax(result)])
    response = ""
```

```
for i in data['intents']:
    if i['tag'] == tag:
        response = np.random.choice(i['responses'])
return jsonify({"response": response})

if __name__ == "__main__":
    app.run(debug=True)
```



FULL SOURCE CODE:

REFER OUR GITHUB LINK

<https://github.com/Shruthi018/HAPPYBOT>

OUTPUT:

WITHOUT FLASK:

train.py

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Epoch 1/500
4/4 [=====] - 1s 7ms/step - loss: 3.2192 - accuracy: 0.0431
Epoch 2/500
4/4 [=====] - 0s 3ms/step - loss: 3.2175 - accuracy: 0.0431
Epoch 3/500
4/4 [=====] - 0s 3ms/step - loss: 3.2163 - accuracy: 0.0431
Epoch 4/500
4/4 [=====] - 0s 3ms/step - loss: 3.2153 - accuracy: 0.0431
Epoch 5/500
4/4 [=====] - 0s 7ms/step - loss: 3.2140 - accuracy: 0.0431
Epoch 6/500
4/4 [=====] - 0s 6ms/step - loss: 3.2128 - accuracy: 0.0431
Epoch 7/500
4/4 [=====] - 0s 6ms/step - loss: 3.2115 - accuracy: 0.0431
Epoch 8/500
4/4 [=====] - 0s 4ms/step - loss: 3.2103 - accuracy: 0.0431
Epoch 9/500
4/4 [=====] - 0s 4ms/step - loss: 3.2088 - accuracy: 0.0431
Epoch 10/500
4/4 [=====] - 0s 6ms/step - loss: 3.2074 - accuracy: 0.0517
Epoch 11/500
4/4 [=====] - 0s 5ms/step - loss: 3.2057 - accuracy: 0.0517
Epoch 12/500
4/4 [=====] - 0s 4ms/step - loss: 3.2042 - accuracy: 0.0517
Epoch 13/500
4/4 [=====] - 0s 5ms/step - loss: 3.2023 - accuracy: 0.0517
Epoch 14/500
4/4 [=====] - 0s 3ms/step - loss: 3.2006 - accuracy: 0.0517
Epoch 15/500
4/4 [=====] - 0s 3ms/step - loss: 3.1982 - accuracy: 0.0517
Epoch 16/500
4/4 [=====] - 0s 6ms/step - loss: 3.1961 - accuracy: 0.0517
Epoch 17/500
4/4 [=====] - 0s 6ms/step - loss: 3.1935 - accuracy: 0.0517
Epoch 18/500
4/4 [=====] - 0s 3ms/step - loss: 3.1910 - accuracy: 0.0517
Epoch 19/500
4/4 [=====] - 0s 4ms/step - loss: 3.1881 - accuracy: 0.0517
Epoch 20/500
```

```
PS C:\Users\Withish\Desktop\project\naan mudhalvan\CHATBOT> python train.py
2023-11-01 19:44:48.351689: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performanc
e-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"

Layer (type)                Output Shape                Param #
=====
embedding (Embedding)       (None, 20, 16)             16000
global_average_pooling1d ( GlobalAveragePooling1D) (None, 16)                  0
dense (Dense)                (None, 16)                  272
dense_1 (Dense)              (None, 16)                  272
dense_2 (Dense)              (None, 25)                  425
=====
Total params: 16969 (66.29 KB)
Trainable params: 16969 (66.29 KB)
Non-trainable params: 0 (0.00 Byte)
```

app.py

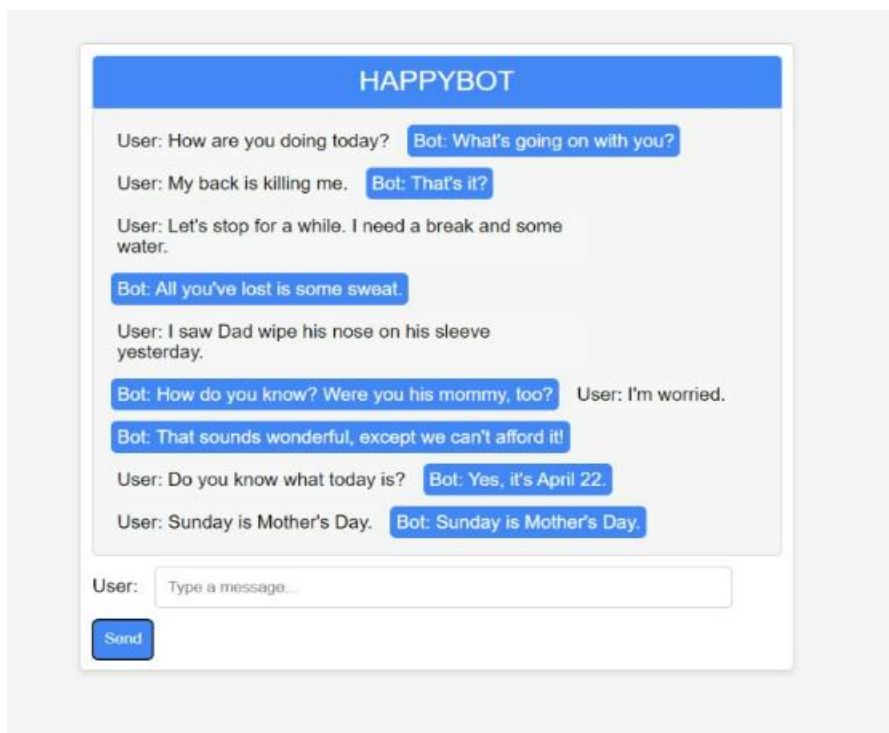
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Epoch 497/500
4/4 [=====] - 0s 3ms/step - loss: 0.0340 - accuracy: 0.9828
Epoch 498/500
4/4 [=====] - 0s 3ms/step - loss: 0.0337 - accuracy: 0.9828
Epoch 499/500
4/4 [=====] - 0s 2ms/step - loss: 0.0342 - accuracy: 0.9741
Epoch 500/500
4/4 [=====] - 0s 3ms/step - loss: 0.0333 - accuracy: 0.9741
PS C:\Users\Withish\Desktop\Project\Naan mudhalvan\CHATBOT> python chat.py
Start messaging with the bot (type quit to stop)!
2023-11-01 19:51:12.437807: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
User: hi, how are you doing?
1/1 [=====] - 0s 147ms/step
ChatBot: I started school recently.
User: I'm having a party this Friday.
1/1 [=====] - 0s 35ms/step
ChatBot: I don't think so.
User: Can you go?
1/1 [=====] - 0s 37ms/step
ChatBot: No doubt.
User: "Is that right?,"
1/1 [=====] - 0s 34ms/step
ChatBot: I wish you luck.
User: Who invented flossing?
1/1 [=====] - 0s 33ms/step
ChatBot: They also ate with their fingers!
User: Do you believe everything you hear?
1/1 [=====] - 0s 36ms/step
ChatBot: I don't believe anything I don't see with my own eyes.
User: What were you doing?
1/1 [=====] - 0s 45ms/step
ChatBot: I wasn't picking my nose.
User: Are you going to start a family?
1/1 [=====] - 0s 35ms/step
ChatBot: Yes, you have to take care of your wife.
User: Maybe we're here to have fun
1/1 [=====] - 0s 35ms/step
ChatBot: Maybe we're here to have fun.
User: Did you ever take an IQ test?
1/1 [=====] - 0s 36ms/step
ChatBot: No one makes any money at it.
User: █
1/1 [=====] - 0s 33ms/step
127.0.0.1 - - [01/Nov/2023 20:03:54] "POST /get_response HTTP/1.1" 200 -
1/1 [=====] - 0s 37ms/step
127.0.0.1 - - [01/Nov/2023 20:05:14] "POST /get_response HTTP/1.1" 200 -
1/1 [=====] - 0s 31ms/step
127.0.0.1 - - [01/Nov/2023 20:06:01] "POST /get_response HTTP/1.1" 200 -
1/1 [=====] - 0s 31ms/step
127.0.0.1 - - [01/Nov/2023 20:07:16] "POST /get_response HTTP/1.1" 200 -
1/1 [=====] - 0s 35ms/step
127.0.0.1 - - [01/Nov/2023 20:08:06] "POST /get_response HTTP/1.1" 200 -
Ln 358, Col 41 (23 selected) Spaces: 4 UTF-8 CRLF {} JSON Go Live

```

WITH FLASK:

app.py



BENEFIT OF CHATBOT:

1. **24/7 Availability:** Chatbots provide round-the-clock customer support, ensuring businesses can engage with users at any time, improving customer satisfaction and loyalty.
2. **Cost-Efficiency:** Automating customer interactions through chatbots reduces operational costs associated with human customer support, allowing businesses to handle a large volume of inquiries without significant manpower.
3. **Instant Responses:** Chatbots offer instantaneous responses to user queries, enhancing user experience by reducing wait times and providing quick solutions to problems.
4. **Data Collection and Analysis:** Chatbots can gather valuable user data, enabling businesses to analyze customer preferences, behaviors, and feedback, leading to data-driven decision-making and personalized marketing strategies.
5. **Scalability:** Chatbots can handle multiple conversations simultaneously, allowing businesses to scale their customer support efforts without a proportional increase in resources, ensuring efficient handling of growing user demands.

CONCLUSION:

Creating a chatbot in Python using a combination of NLTK for natural language processing, TensorFlow for deep learning, bagging for improved model performance, and Flask for web integration provides a powerful and versatile solution for building interactive and intelligent chatbots that can effectively communicate with users and serve a wide range of applications.



