AI Project:

Building a chat box

OVERVIEW:

Creating a chatbot that uses Natural Language Processing (NLP),that includes data collection, data preprocessing, model training, and deployment. Below, I outline a Python code framework that covers these aspects. This example does not include the specifics of model training or the details of each step but provides a structured approach you can customize based on your needs. We'll use popular libraries such as pandas for data handling, nltk and scikit-learn for NLP and model training, and pickle for saving models.

AI Code and Steps:

Step 1: Data Collection code

```
import pandas as pd

def load_data(file_path):
"""
Load conversational data from the specified file path.
This function assumes data is in a CSV format.
"""
data = pd.read_csv(file_path)
return data

def generate_data_report(data):
"""
Generate a report detailing the volume and initial observations of the collected data.
"""
report = {
"total_conversations": len(data),
"columns": data.columns.tolist(),
}
return report

data = load_data("path_to_your_data.csv")
```

```
report = generate_data_report(data)
print(report)
```

Step 2 Data Preprocessing

```
from nltk.corpus
import stopwords
from nltk.stem
import PorterStemmer
from nltk.tokenize
import word_tokenize
from sklearn.feature_extraction.text
import TfidfVectorizer
import nltk

nltk.download('punkt')
nltk.download('stopwords')

def preprocess_data(data):
"""
Preprocess conversational data for NLP tasks.
"""
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()

data['processed'] = data['text'].apply(lambda x: ' '.join([stemmer.stem(word) for word
in word_tokenize(x.lower()) if word not in stop_words and word.isalpha()]))

vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(data['processed'])
return vectors, vectorizer

vectors, vectorizer = preprocess_data(data)
```

Step 3 Model Training

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(vectors, data['label'],
test_size=0.2, random_state=42)
```

```
model = LogisticRegression()
model.fit(X_train, y_train)

print(f"Model accuracy: {model.score(X_test, y_test)}")
```

Step 4: Saving Models and Files

```
import pickle

with open('chatbot_model.pkl', 'wb') as file:
pickle.dump(model, file)

with open('vectorizer.pkl', 'wb') as file:
pickle.dump(vectorizer, file)
```

Step 5: Inference

```
def load_model_and_vectorizer(model_path, vectorizer_path):
with open(model_path, 'rb') as file:
model = pickle.load(file)
with open(vectorizer_path, 'rb') as file:
vectorizer = pickle.load(file)
return model, vectorizer

def predict(model, vectorizer, text):
processed = preprocess_data(pd.DataFrame([{"text": text}]))[0]
prediction = model.predict(processed)
return prediction

model, vectorizer = load_model_and_vectorizer('chatbot_model.pkl', 'vectorizer.pkl')
print(predict(model, vectorizer, "Your input text here"))
```

This code outlines a basic structure to build, save, and use an NLP-based chatbot.

Web Interface Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Chatbox</title>
<style>
p {
font-family: Arial, Helvetica, sans-serif;
font-size:xx-large;
color: aliceblue;

}
body {
font-family: Arial, sans-serif;
margin: 0;
padding: 0;
background-color: #0c0b0b;
}
img {
display: block;
margin:auto;

}
.container {
max-width: 600px;
margin: auto;
padding: 20px;
background-color: #fff;
border-radius: 10px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.message-container {
margin-bottom: 20px;
}
.user-message {
background-color: #e0f7fa;
border-radius: 10px;
```

```
padding: 10px;
margin-right: auto;
margin-left: 20px;
max-width: 70%;
}
.bot-message {
background-color: #dcedc8;
border-radius: 10px;
padding: 10px;
margin-left: auto;
margin-right: 20px;
max-width: 70%;
}
input[type="text"] {
width: calc(100% - 20px);
padding: 10px;
border: 1px solid #ddd;
border-radius: 5px;
}
input[type="submit"] {
background-color: #4caf50;
color: white;
border: none;
padding: 10px 20px;
text-align: center;
text-decoration: none;
display: inline-block;
font-size: 16px;
border-radius: 5px;
cursor: pointer;
}
.color1 {
background: linear-gradient(-45deg, #07a0f3, #041628, #73b4bb, #d6e1e7);
margin-bottom: 135px;
}
</style>
<script>
const form = document.getElementById("message-form");
const messageContainer = document.getElementById("message-container");

form.addEventListener("submit", async (e) => {
e.preventDefault();
```

```
const userInput = document.getElementById("user-input").value;
const response = await fetch("/generate_response", {
method: "POST",
headers: {
"Content-Type": "application/x-www-form-urlencoded"
},
body: `user_input=${userInput}`
});
const responseData = await response.json();
const botMessage = `<div class="bot-message">${responseData.response}</div>`;
messageContainer.innerHTML += botMessage;
document.getElementById("user-input").value = "";
});
</script>
</head>
<body class="color1">
<div>
<div style="margin-top: 50px;"><img src="chatbox.png" height="350px"
width="350px"></div>
<p style="text-align: center;">How can I help you today?</p>
<div class="container">
<div class="message-container" id="message-container">
<div class="bot-message">Welcome to the Chatbox!</div>
<!-- <div class ="bot-message">How can I help you today?</div> -->
</div>
<form id="message-form">
<input type="text" id="user-input" placeholder="Message Chatbox...">
<input type="submit" value="Send">
</form>
</div>
</div>
</body>
</html>
```