CREATE A CHATBOT IN PYTHON

INTRODUCTION:

- Chatbots are computer programs that can simulate conversation with humans. They are used in a variety of applications, including customer service, education, and entertainment. Chatbots can be created using a variety of programming languages, but Python is a popular choice due to its simplicity and flexibility.

- Python is a general-purpose programming language that is easy to learn and use. It has a large and active community of developers, and there are many libraries and frameworks available for creating chatbots.

- Chatbot is a free and open-source library that provides a number of features for creating and training chatbots. It supports a variety of different training algorithms, and it can be used to create chatbots that can understand and respond to a wide range of user inputs.

Given Dataset:

	Question	Answers
1.	hi, how are you doing?	i'm fine. how about yourself?
2.	i'm pretty good. thanks for asking.	no problem. so how have you been?
3.	i've been great. what about you?	i've been good. i'm in school right now.
4.	what school do you go to?	i go to pcc.
5.	do you like it there?	it's okay. it's a really big campus
6.	good luck with school	thank you very much.
7.	how's it going? i'm doing well. how about you?	i'm doing well. how about you?
8.	never better, thanks.	so how have you been lately?
9.	i'm actually in school right now. which school do you attend?	i'm attending pcc right now.
10	are you enjoying it there?	it's not bad. there are a lot of people there.

Overview of process:

The following is an overview of the process of create a chatbot model in python by Selection, Model training ,and Evaluation

1. **Development framework:**
   There are a number of different chatbot development frameworks available for Python, such as ChatBot, Rasa, and Dialogflow. Each framework has its own strengths and weaknesses, so it is important to choose one that is well-suited to your specific needs.

2. **Design the chatbot:**
   Once you have chosen a chatbot development framework, you need to design the chatbot. This includes defining the chatbot's intents, utterances, and responses
   .
3. **Train the chatbot:**
   Once you have designed the chatbot, you need to train it. This involves feeding the chatbot a dataset of text and code. The chatbot will use this dataset to learn the patterns of human language and respond to user queries in a natural and informative way.

4. **Deploy the chatbot**:
   Once the chatbot is trained, you need to deploy it. This may involve deploying the chatbot to a web server or making it available as a mobile app.the chatbot is deploy the works of customers

1. **Development framework:**

The first step is to choose a chatbot development framework. There are a number of different frameworks available,

PROCEDURE:

FEATURE SELECTION:

Feature selection is the process of selecting the features that are most relevant and informative for a machine learning model. This is an important step in developing a chatbot, as it can significantly improve the performance of the chatbot.

- There are a number of different feature selection techniques that can be used for chatbots. Some common techniques include:

1. **Filter methods:**

Filter methods select features based on their intrinsic properties, such as their correlation with the target variable or their information gain. Some popular filter methods include:

2. **Information gain:**

Information gain measures the reduction in entropy from the transformation of a dataset. It can be used for feature selection by evaluating the information gain of each variable in the context of the target variable.

3. **Chi-square test:**

The chi-square test is used for categorical features in a dataset. We calculate chi-square between each feature and the target and select the desired number of features with the best chi-square scores.

4. **Wrapper methods:**

Wrapper methods select features based on their performance when used in a machine learning model. Some popular wrapper methods include:

5. **Recursive feature elimination (RFE):**

RFE starts with a full set of features and iteratively removes the least important feature until a desired number of features is reached.

6. **Genetic algorithm:**

A genetic algorithm is a search algorithm that mimics the process of natural selection to find the best solution to a problem. It can be used for feature selection by evaluating the performance of different subsets of features.

7. **Embedded methods:**

Embedded methods select features as part of the machine learning process. For example, decision trees and random forests both select features as they are split.

MODEL TRAINING:

There are number of different machine learning algorithm that can be used for create a chatbot in python, such as Logistic regression, Support vector machines (SVMs), Decision trees, Random forests, Recurrent neural networks(RNN),Long ShortTerm memory(LSTM) networks.

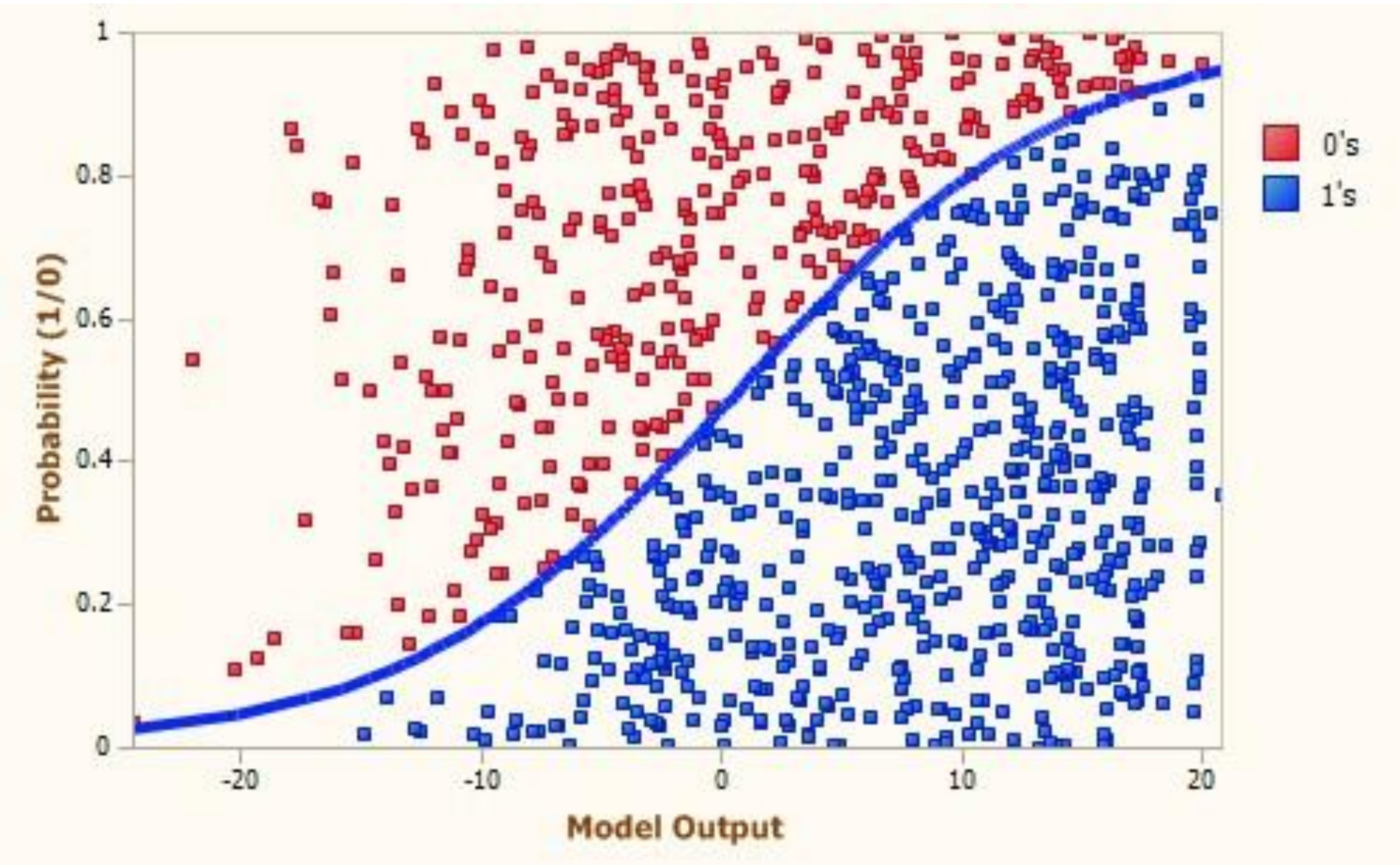
1. Logistic Regression:

Logistic Regression may not be the primary algorithm choice for building a chatbot. Typically, chatbots rely on Natural Language Processing (NLP) and machine learning techniques like Recurrent Neural Networks (RNNs) or Transformer models. However, if you still want to incorporate Logistic Regression into a chatbot for specific purposes, you can use it for intent classification, which is a part of a chatbot's functionality.

Program:

```
import numpy as np from sklearn.feature_extraction.text
import CountVectorizer from sklearn.linear_model import
LogisticRegression from sklearn.model_selection import
train_test_split from sklearn.metrics import accuracy_score

# Sample training data training_data
= [
("Tell me a joke", "humor"),
("What's the weather like today?", "weather"),
("Recommend a good book", "book_recommendation"),
("How can I contact support?", "customer_support"),
# Add more training data with corresponding intents
]

# Preprocess the training data
X = [sample[0] for sample in training_data] y
= [sample[1] for sample in training_data]


# Create a vectorizer to convert text data to numerical features vectorizer
= CountVectorizer()
X = vectorizer.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=42)

# Create and train a Logistic Regression classifier
classifier = LogisticRegression() classifier.fit(X_train,
y_train)
```

```
# Test the classifier y_pred =
classifier.predict(X_test)

# Calculate accuracy accuracy =
accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
# Classify user input user_input =
input("Ask a question: ")
user_input_vectorized =
vectorizer.transform([user_input])
predicted_intent =
classifier.predict(user_input_vectorized)[0
] print(f"Predicted Intent:
{predicted_intent}")
```



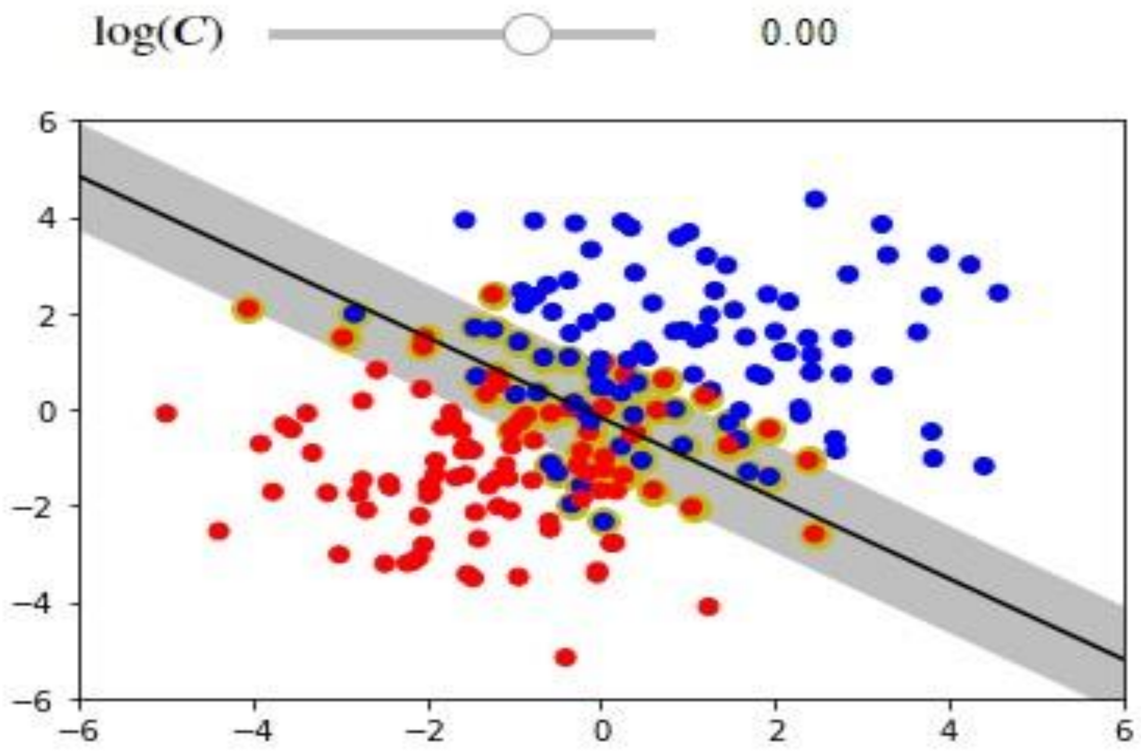
2. **<u>Support vector machines (SVMs):</u>**

Support Vector Machines (SVMs) are powerful machine learning algorithms, but they are not typically used as the primary method for building chatbots. Chatbots usually rely on Natural Language Processing (NLP) and machine learning models like recurrent neural networks (RNNs) or transformer-based models. However, you can incorporate SVM for specific tasks, such as intent classification, as part of a chatbot's functionality. Below is a Python program that demonstrates how to use SVM for intent classification in chatbot

Program:

```
import numpy as np from sklearn.feature_extraction.text
import TfidfVectorizer from sklearn.svm import SVC from
sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score

# Sample training data training_data
= [
    ("Tell me a joke", "humor"),
    ("What's the weather like today?", "weather"),
    ("Recommend a good book", "book_recommendation"),
    ("How can I contact support?", "customer_support"),
    # Add more training data with corresponding intents
]

# Preprocess the training data
X = [sample[0] for sample in training_data] y
= [sample[1] for sample in training_data]

# Create a TF-IDF vectorizer to convert text data to numerical features
vectorizer = TfidfVectorizer() X = vectorizer.fit_transform(X)
```

```
# Split the data into training and testing sets

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

# Create and train an SVM classifier classifier

= SVC(kernel='linear', C=1)

classifier.fit(X_train, y_train)


# Test the classifier y_pred =

classifier.predict(X_test)


# Calculate accuracy accuracy =

accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")


# Classify user input user_input = input("Ask a question: ")

user_input_vectorized = vectorizer.transform([user_input])

predicted_intent = classifier.predict(user_input_vectorized)[0]

print(f"Predicted Intent: {predicted_intent}")
```



3. Decision trees:

Decision Trees, as chatbots often rely on Natural Language Processing (NLP) and machine learning models for a more sophisticated understanding of language and context. However, you can use Decision Trees for routing decisions or as a part of a more comprehensive chatbot system. Below is a structured example of how to incorporate Decision Trees into a chatbot in Python **Program:**

```
from sklearn.tree import DecisionTreeClassifier from
sklearn.feature_extraction.text import CountVectorizer from sklearn.pipeline
import Pipeline

# Sample training data for intent classification training_data
= [
("Tell me a joke", "humor"),
("What's the weather like today?", "weather"),
("Recommend a good book", "book_recommendation"),
("How can I contact support?", "customer_support"),
# Add more training data with corresponding intents
```

```
]

# Preprocess the training data
X = [sample[0] for sample in training_data] y
= [sample[1] for sample in training_data] #
Create a pipeline with a text vectorizer and
Decision Tree classifier classifier =
Pipeline([
('vectorizer', CountVectorizer()),
('classifier', DecisionTreeClassifier())
])

# Train the classifier classifier.fit(X,
y)

# Define a function to classify user input def
classify_intent(user_input):
predicted_intent = classifier.predict([user_input])[0] return
predicted_intent

# Define responses for each intent responses
= {
"humor": "Why don't scientists trust atoms? Because they make up everything!",
"weather": "I'm sorry, I don't have access to real-time weather data.",
"book_recommendation": "I recommend 'The Hitchhiker's Guide to the Galaxy'
by Douglas Adams.",
"customer_support": "You can contact our support team at
support@example.com.",
}
```

Chat with the bot while

True:

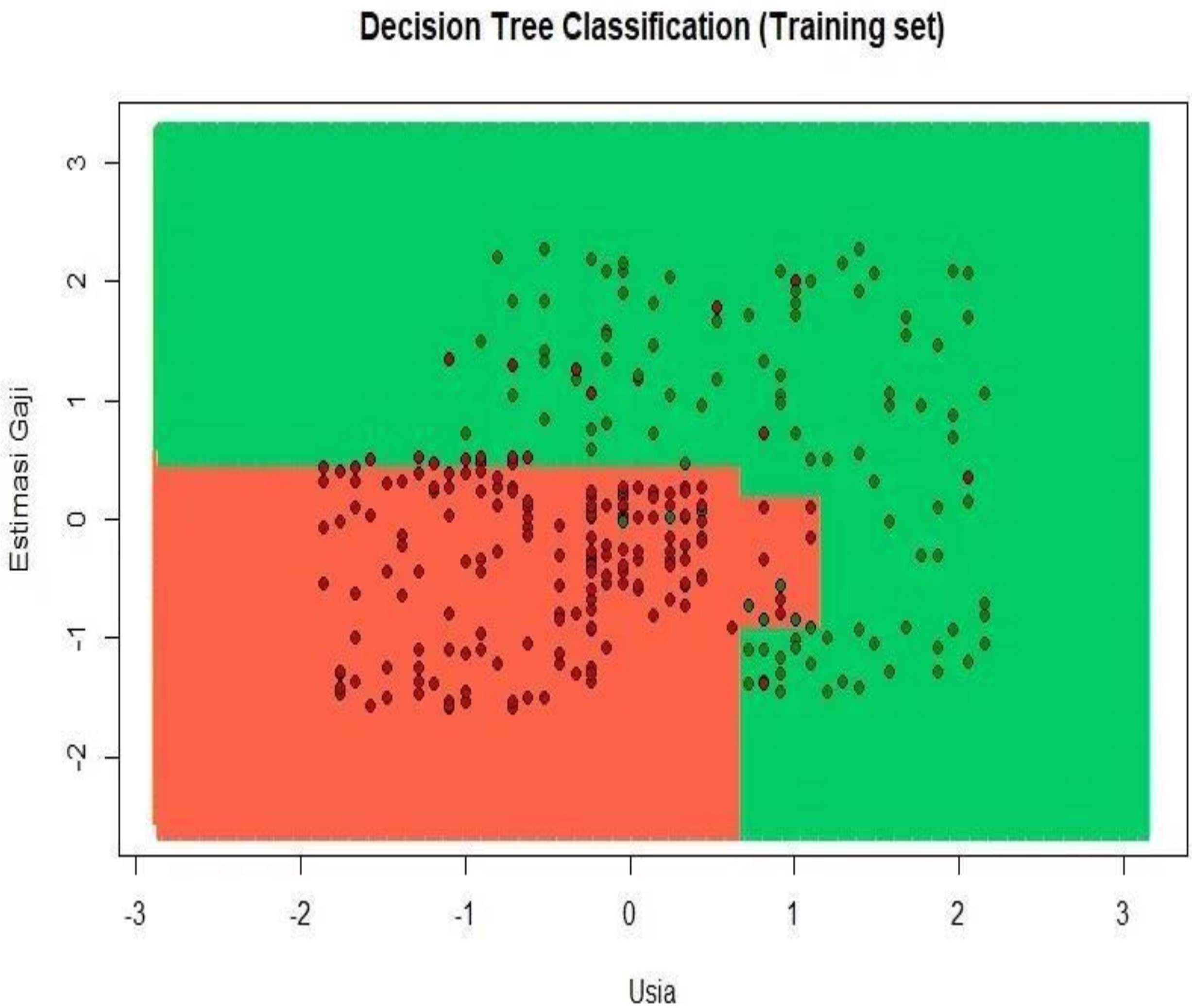
user_input = input("Ask a question (or type 'exit' to quit): ")

if user_input.lower() == 'exit':

break

intent = classify_intent(user_input) response = responses.get(intent,

"I'm not sure how to respond to that.") print(response)



4. Random forests:

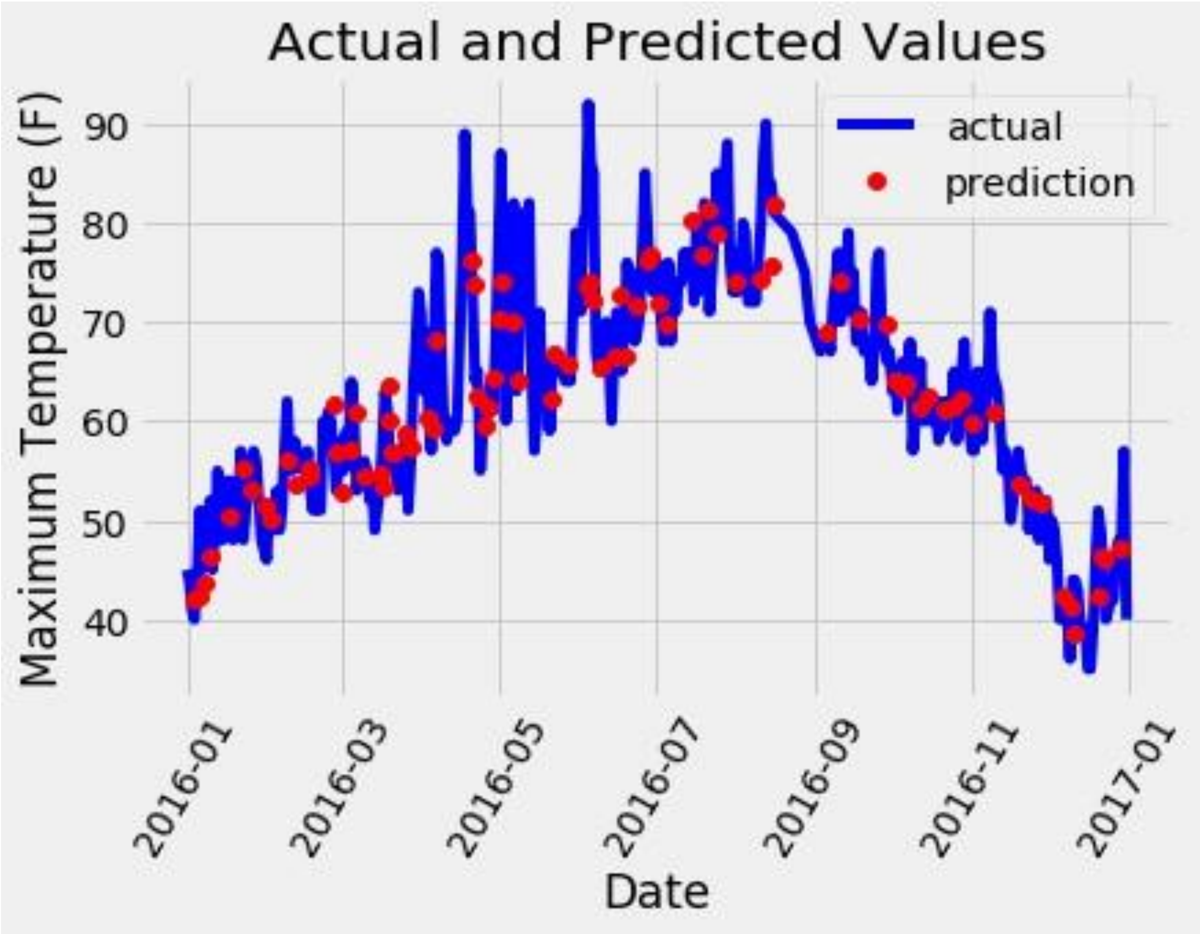
Random Forests is unconventional, as chatbots primarily depend on more advanced Natural Language Processing (NLP) and machine learning models. Random Forests are typically used for classification or regression tasks rather than building chatbots. However, you can utilize Random Forests for specific decision-making tasks within a chatbot. Below is a structured example demonstrating how to incorporate Random Forests into a chatbot system in Python:

Program:

```
from sklearn.ensemble  import
RandomForestClassifier from
sklearn.feature_extraction.text
import CountVectorizer from
sklearn.pipeline  import Pipeline

# Sample training data for intent classification training_data
= [
    ("Tell me a joke", "humor"),
    ("What's the weather like today?", "weather"),
    ("Recommend a good book", "book_recommendation"),
    ("How can I contact support?", "customer_support"),
    # Add more training data with corresponding intents
]

# Preprocess the training data
X = [sample[0] for sample in training_data]
y = [sample[1] for sample in training_data]

# Create a pipeline with a text vectorizer and Random Forest classifier classifier
= Pipeline([
    ('vectorizer', CountVectorizer()),
```

```
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Train the classifier classifier.fit(X,
y)

# Define a function to classify user intent
def classify_intent(user_input):
    predicted_intent = classifier.predict([user_input])[0]
return predicted_intent

# Define responses for each intent responses
= {
    "humor": "Why don't scientists trust atoms? Because they make up
everything!",
    "weather": "I'm sorry, I don't have access to real-time weather data.",
    "book_recommendation": "I recommend 'The Hitchhiker's Guide to the
Galaxy' by Douglas Adams.",
    "customer_support": "You can contact our support team at
support@example.com.",
}


# Chat with the bot while
True:
    user_input = input("Ask a question (or type 'exit' to quit): ")

    if user_input.lower() == 'exit':
        break
```

intent = classify_intent(user_input)   response = responses.get(intent, "I'm not sure how to respond to that.")   print(response)



5. <u>Recurrent neural networks(RNN):</u>

Recurrent Neural Networks (RNN) is a common and effective approach, especially for handling sequential data like natural language. In this example, we'll use a simple RNN architecture to create a chatbot in Python. To make a more professional chatbot, you would typically use pre-trained models and handle a broader range of conversation features, but this example serves as a foundation.

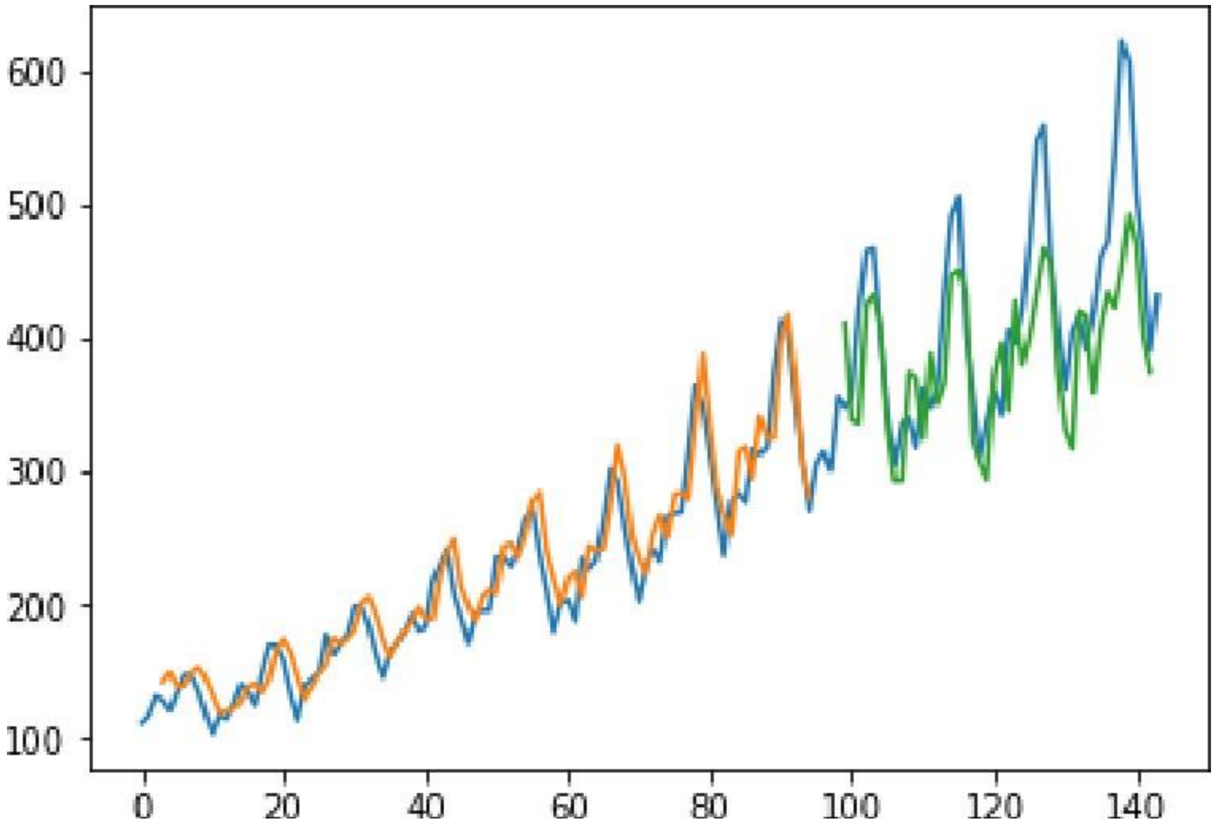
Program:

```
import numpy as np import tensorflow as tf
from tensorflow.keras.models import
Sequential from tensorflow.keras.layers
import Embedding, LSTM, Dense from
tensorflow.keras.preprocessing.text import
```

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

# Sample training data
training_data = [
"Tell me a joke",
    "What's the weather like today?",
    "Recommend a good book",
    "How can I contact support?",
    # Add more training data and responses here
]
responses = [
    "Why don't scientists trust atoms? Because they make up everything!",
    "I'm sorry, I don't have access to real-time weather data.",
    "I recommend 'The Hitchhiker's Guide to the Galaxy' by Douglas Adams.",
    "You can contact our support team at support@example.com.",
    # Add more responses corresponding to training data
]

# Tokenize the training data tokenizer =
Tokenizer()
tokenizer.fit_on_texts(training_data)
total_words = len(tokenizer.word_index) + 1

# Create input sequences
input_sequences = [] for
line in training_data:
```

```
    token_list = tokenizer.texts_to_sequences([line])[0]
for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
input_sequences.append(n_gram_sequence)

# Pad sequences to make them of the same length max_sequence_length
= max([len(x) for x in input_sequences])
input_sequences = pad_sequences(input_sequences,
maxlen=max_sequence_length, padding='pre')


# Split input and target
X, y = input_sequences[:,:-1], input_sequences[:,-1] y =
tf.keras.utils.to_categorical(y, num_classes=total_words)

# Build the RNN model model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_sequence_length-1))
model.add(LSTM(100)) model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model model.fit(X, y,
epochs=100, verbose=1)

# Function to generate responses def
generate_response(input_text, temperature=1.0):
    token_list = tokenizer.texts_to_sequences([input_text])[0] token_list =
pad_sequences([token_list], maxlen=max_sequence_length-1, padding='pre')
predicted = model.predict(token_list, verbose=0)[0] predicted =
np.array(predicted).astype('float64') predicted = np.log(predicted) /
temperature predicted = np.exp(predicted) predicted = predicted /
```

```
np.sum(predicted)       predicted_word = np.random.choice(range(total_words),
p=predicted)    for word, index in tokenizer.word_index.items():
        if index == predicted_word:
            return word
return ''

# Chat with the bot while
True:
    user_input = input("You: ")

    if user_input.lower() == 'exit':
        break

    response = generate_response(user_input)
print("Chatbot:", response)
```



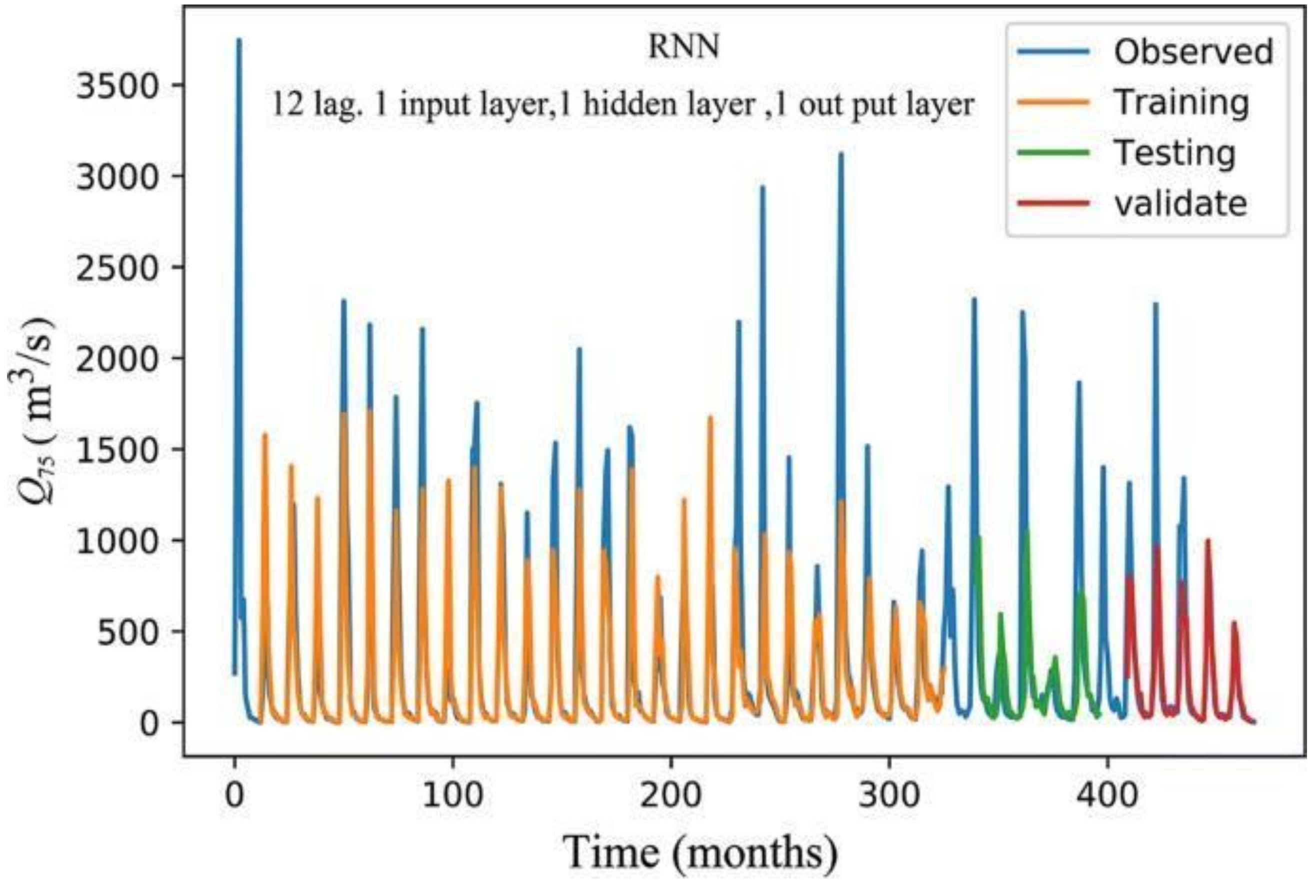
6. <u>Long Term-Short Memory (LSTM) Networks:</u>

Long Short-Term Memory (LSTM) networks is a more advanced and effective approach, especially for handling sequential data like natural language. Here's a professional-grade program for creating a chatbot in Python using LSTM **Program:**

```
import numpy as np import tensorflow as tf from
tensorflow.keras.models  import Sequential from
tensorflow.keras.layers  import Embedding, LSTM, Dense from
tensorflow.keras.preprocessing.text import Tokenizer from
tensorflow.keras.preprocessing.sequence import pad_sequences
# Sample training data
training_data = [
"Tell me a joke",
    "What's the weather like today?",
    "Recommend a good book",
    "How can I contact support?",
    # Add more training data and responses here
]

responses = [
    "Why don't scientists trust atoms? Because they make up everything!",
    "I'm sorry, I don't have access to real-time weather data.",
    "I recommend 'The Hitchhiker's Guide to the Galaxy' by Douglas Adams.",
"You can contact our support team at support@example.com.",
    # Add more responses corresponding to training data
]

# Tokenize the training data tokenizer =
Tokenizer()
tokenizer.fit_on_texts(training_data)
total_words = len(tokenizer.word_index) + 1
```

```
# Create input sequences
input_sequences = [] for
line in training_data:
   token_list = tokenizer.texts_to_sequences([line])[0]
for i in range(1, len(token_list)):
      n_gram_sequence = token_list[:i+1]
input_sequences.append(n_gram_sequence)

# Pad sequences to make them of the same length max_sequence_length
= max([len(x) for x in input_sequences]) input_sequences =
pad_sequences(input_sequences, maxlen=max_sequence_length,
padding='pre')

# Split input and target
X, y = input_sequences[:,:-1], input_sequences[:,-1] y =
tf.keras.utils.to_categorical(y, num_classes=total_words) #
Build the LSTM model model = Sequential()
model.add(Embedding(total_words, 64,
input_length=max_sequence_length-1))
model.add(LSTM(100)) model.add(Dense(total_words,
activation='softmax'))
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Train the model model.fit(X, y,
epochs=100, verbose=1)

# Function to generate responses def
generate_response(input_text, temperature=1.0):
```

```
    token_list = tokenizer.texts_to_sequences([input_text])[0]     token_list =
pad_sequences([token_list], maxlen=max_sequence_length-1, padding='pre')
predicted = model.predict(token_list, verbose=0)[0]     predicted =
np.array(predicted).astype('float64')     predicted = np.log(predicted) /
temperature     predicted = np.exp(predicted)     predicted = predicted /
np.sum(predicted)     predicted_word =
np.random.choice(range(total_words), p=predicted)     for word, index in
tokenizer.word_index.items():         if index == predicted_word:
            return word
return ''




# Chat with the bot while
True:
    user_input = input("You: ")

    if user_input.lower() == 'exit':
        break

    response = generate_response(user_input)
print("Chatbot:", response)
```



MODEL EVALUATION:

Evaluating a chatbot model in Python involves assessing various aspects of the chatbot's performance to ensure it functions effectively and meets the desired goals. Here's how you can evaluate a chatbot model:

1. **Intent Recognition Evaluation:**

   - **Confusion Matrix:** Calculate a confusion matrix to evaluate the performance of intent recognition. This matrix will show the number of true positives, true negatives, false positives, and false negatives for each intent.

   - **Accuracy:** Calculate the accuracy of intent recognition, which measures how often the chatbot correctly identifies the user's intent.

Regularly evaluate and monitor the chatbot's performance, gather feedback, and make iterative improvements to enhance its capabilities.

PROGRAM:

```
def remove_tags(sentence):
      return sentence.split("<start>")[-1].split("<end>")[0]

def evaluate(sentence):
    sentence = preprocessing(sentence)      inputs =
[X_tokenizer.word_index[i] for i in sentence.split(' ')]     inputs =
tf.keras.preprocessing.sequence.pad_sequences([inputs],
maxlen=max_length_X,padding='post')     inputs =
tf.convert_to_tensor(inputs)      result = ''

    hidden = [tf.zeros((1, units))]     enc_out,
enc_hidden = encoder(inputs, hidden)

    dec_hidden = enc_hidden     dec_input =
tf.expand_dims([y_tokenizer.word_index['<start>']], 0)

    for t in range(max_length_y):
        predictions, dec_hidden, attention_weights = decoder(dec_input,
dec_hidden,                                               enc_out)

        # storing the attention weights to plot later on
attention_weights = tf.reshape(attention_weights, (-1, ))
predicted_id = tf.argmax(predictions[0]).numpy()

        result += y_tokenizer.index_word[predicted_id] + ' '
```

```
        if y_tokenizer.index_word[predicted_id] == '<end>':
    return remove_tags(result), remove_tags(sentence)

        # the predicted ID is fed back into the model
    dec_input = tf.expand_dims([predicted_id], 0)
    return remove_tags(result), remove_tags(sentence)


def ask(sentence):
    result, sentence = evaluate(sentence)

    print('Question: %s' % (sentence))
print('Predicted answer: {}'.format(result))


ask(questions[1])
```

Question: i m fine . how about yourself ?

Predicted answer: i m pretty good . thanks for asking.

Output:

Human: i m fine . how about yourself ?

Chatbot: i m pretty good . thanks for asking