Revolutionizing customer support with an intelligent chatbot for automated assistance

Source code for chat bot model:

#Program:

```
# Google Colab Customer Support Chatbot with CSV Support and Example
  Questions
# Copy and run this entire notebook in Google Colab

import csv
import re
import random
import pandas as pd
from difflib import get_close_matches
from IPython.display import HTML, display, clear_output
from google.colab import output
import ipywidgets as widgets

class CustomerSupportChatbot:
    def __init__(self, csv_path=None):
        """Initialize the chatbot with dataset from CSV file or use sample
  data"""
        self.dataset = []

        if csv_path:
            try:
                # Load dataset from CSV using pandas
                df = pd.read_csv(csv_path)
                print(f"Loaded {len(df)} entries from CSV dataset.")
```

```
            # Convert DataFrame to list of dictionaries
            # Assuming CSV has columns: customer_query, intent, response,
etc.
            self.dataset = df.to_dict('records')
        except FileNotFoundError:
            print(f"CSV file not found. Using built-in sample data.")
            self.dataset = self.get_sample_data()
        except Exception as e:
            print(f"Error loading CSV: {e}. Using built-in sample data.")
            self.dataset = self.get_sample_data()
    else:
        print("No dataset provided. Using built-in sample data.")
        self.dataset = self.get_sample_data()

    # Create a mapping of queries to their intents for faster lookup
    self.query_intent_map = {}
    for item in self.dataset:
        if 'customer_query' in item and 'intent' in item:
            query = str(item['customer_query']).lower()
            self.query_intent_map[query] = item['intent']

    # Group responses by intent
    self.intent_responses = {}
    for item in self.dataset:
        if 'intent' in item and 'response' in item:
            intent = item['intent']
            response = item['response']
            if intent not in self.intent_responses:
                self.intent_responses[intent] = []
            self.intent_responses[intent].append(response)

    # Default responses for unknown intents
    self.default_responses = [
```

```
        "I'm not sure I understand. Could you please rephrase your question?",
        "I don't have enough information to help with that. Could you provide more details?",
        "I'm still learning and don't have an answer for that yet. Would you like to speak with a human agent?",
        "I'm sorry, but I don't understand. Can you try asking in a different way?"
    ]

    # Track conversation context
    self.conversation_history = []
    self.current_intent = None

    # Get a list of example questions for the UI
    self.example_questions = self.get_example_questions()

def get_sample_data(self):
    """Return sample data in case the dataset file is not found"""
    return [
        {
            "customer_query": "How do I reset my password?",
            "intent": "password_reset",
            "response": "You can reset your password by clicking on the 'Forgot Password' link on the login page. We'll send you an email with instructions to create a new password.",
            "follow_up_questions": "I didn't receive the reset email,How long does it take to get the reset link?",
            "sentiment": "neutral",
            "priority": "medium"
        },
        {
            "customer_query": "Your website is not loading properly on my phone",
            "intent": "technical_issue",
```

"response": "I'm sorry to hear that. Could you tell me which type of phone and browser you're using? Also, try clearing your cache or using incognito mode to see if that resolves the issue.",

"follow_up_questions": "I cleared my cache but it's still not working,How do I clear my cache?",

"sentiment": "negative",

"priority": "high"

},

{

"customer_query": "When will my order #45789 be delivered?",

"intent": "order_status",

"response": "I'd be happy to check that for you. Order #45789 is currently in transit and scheduled for delivery tomorrow. You can track your package in real-time through your account dashboard.",

"follow_up_questions": "Can I change the delivery address?,What if I'm not home for the delivery?",

"sentiment": "neutral",

"priority": "medium"

},

{

"customer_query": "I want to cancel my subscription",

"intent": "cancel_subscription",

"response": "I can help you cancel your subscription. To proceed, please confirm your account email address and the reason for cancellation. This helps us improve our services.",

"follow_up_questions": "Will I get a refund?,Can I pause instead of cancel?",

"sentiment": "negative",

"priority": "high"

},

{

"customer_query": "Do you offer student discounts?",

"intent": "discount_inquiry",

"response": "Yes, we offer a 15% discount for students with valid ID. You can verify your student status through our website by clicking on

'Student Discount' and following the verification process.",
"follow_up_questions": "What verification do you need?,Does the discount apply to all products?",
"sentiment": "positive",
"priority": "low"
},
{
"customer_query": "I received a damaged product",
"intent": "product_issue",
"response": "I'm very sorry to hear that. We'll make this right. Could you please provide your order number and attach a photo of the damaged item? We'll arrange for a replacement or refund right away.",
"follow_up_questions": "Do I need to return the damaged item?,How long will the replacement take?",
"sentiment": "negative",
"priority": "high"
},
{
"customer_query": "How do I update my billing information?",
"intent": "account_management",
"response": "You can update your billing information by logging into your account, going to 'Settings', then selecting 'Payment Methods'. From there, you can edit existing information or add new payment methods.",
"follow_up_questions": "Can I have multiple payment methods?,Is my payment information secure?",
"sentiment": "neutral",
"priority": "medium"
},
{
"customer_query": "Do you ship internationally?",
"intent": "shipping_inquiry",
"response": "Yes, we ship to over 50 countries worldwide. International shipping typically takes 7-14 business days and costs vary by destination. You can see specific shipping rates for your country during

```
checkout before finalizing your purchase.",
            "follow_up_questions": "Do I have to pay customs fees?,Can I
track international orders?",
            "sentiment": "neutral",
            "priority": "low"
        }
    ]

 def get_example_questions(self):
    """Get a list of example questions from the dataset for the UI"""
    examples = []

    # Extract unique customer queries for each intent type
    intent_queries = {}
    for item in self.dataset:
        if 'customer_query' in item and 'intent' in item:
            intent = item['intent']
            query = item['customer_query']

            # Only collect one example per intent type
            if intent not in intent_queries:
                intent_queries[intent] = query
                examples.append(query)

    # Limit to 5 examples (or fewer if there are less intents)
    return examples[:5]

 def find_intent(self, user_input):
    """Find the intent that best matches the user input"""
    user_input = user_input.lower()

    # Check for order number pattern (e.g., #12345)
    order_match = re.search(r'order\s+(?:#|number\s+)?(\d+)', user_input)
```

```
    if order_match:
        return "order_status"

    # Check for direct matches first
    if user_input in self.query_intent_map:
        return self.query_intent_map[user_input]

    # Check for keyword matches
    intent_keywords = {
        "password_reset": ["password", "reset", "forgot", "can't login", "can't sign in"],
        "technical_issue": ["bug", "glitch", "not working", "broken", "error", "crash"],
        "order_status": ["order", "delivery", "package", "shipping", "arrive", "track"],
        "cancel_subscription": ["cancel", "subscription", "stop service", "end plan"],
        "discount_inquiry": ["discount", "coupon", "promo", "code", "offer", "deal"],
        "product_issue": ["damaged", "defective", "broken", "not as described", "wrong item"],
        "billing_issue": ["charge", "bill", "payment", "invoice", "refund", "charged twice"],
        "account_management": ["account", "profile", "settings", "update", "change email"],
        "escalation": ["manager", "supervisor", "human", "real person", "agent"],
        "shipping_inquiry": ["ship", "shipping", "delivery", "international", "domestic"]
    }

    for intent, keywords in intent_keywords.items():
        if any(keyword in user_input for keyword in keywords):
            return intent
```

```
    # Use fuzzy matching as a fallback
    if self.query_intent_map:
        matches = get_close_matches(user_input,
self.query_intent_map.keys(), n=1, cutoff=0.6)
        if matches:
            return self.query_intent_map[matches[0]]

    # Default intent if no match found
    return "unknown"

 def get_response(self, intent):
    """Get a response based on the identified intent"""
    if intent == "unknown":
        return random.choice(self.default_responses)

    if intent in self.intent_responses and self.intent_responses[intent]:
        return random.choice(self.intent_responses[intent])

    # Fallback if we have the intent but no responses
    return "I understand you're asking about " + intent.replace("_", " ") + ".
Let me connect you with someone who can help with that."

 def process_input(self, user_input):
    """Process user input and return a response"""
    # Store the input in conversation history
    self.conversation_history.append({"role": "user", "message":
user_input})

    # Find the intent
    intent = self.find_intent(user_input)
    self.current_intent = intent

    # Get a response
    response = self.get_response(intent)
```

```
        # Store the response in conversation history
        self.conversation_history.append({"role": "bot", "message": response})

        return response

    def get_follow_up_questions(self):
        """Return relevant follow-up questions based on current intent"""
        if self.current_intent == "unknown":
            return []

        follow_ups = []
        for item in self.dataset:
            if 'intent' in item and item['intent'] == self.current_intent and
    'follow_up_questions' in item:
                # Handle both list and comma-separated string formats
                if isinstance(item['follow_up_questions'], list):
                    follow_ups.extend(item['follow_up_questions'])
                elif isinstance(item['follow_up_questions'], str):
                    follow_ups.extend(item['follow_up_questions'].split(','))

        # Return unique follow-up questions (up to 3)
        unique_follow_ups = list(set(follow_ups))
        return unique_follow_ups[:3]

# Function to create sample CSV for testing
def create_sample_csv(filename="sample_customer_support_data.csv"):
    """Create a sample CSV file for testing"""
    data = [
        {
            "customer_query": "How do I reset my password?",
            "intent": "password_reset",
            "response": "You can reset your password by clicking on the 'Forgot
```

Password' link on the login page. We'll send you an email with instructions to create a new password.",

"follow_up_questions": "I didn't receive the reset email,How long does it take to get the reset link?",

"sentiment": "neutral",

"priority": "medium"

},

{

"customer_query": "Your website is not loading properly on my phone",

"intent": "technical_issue",

"response": "I'm sorry to hear that. Could you tell me which type of phone and browser you're using? Also, try clearing your cache or using incognito mode to see if that resolves the issue.",

"follow_up_questions": "I cleared my cache but it's still not working,How do I clear my cache?",

"sentiment": "negative",

"priority": "high"

},

{

"customer_query": "When will my order #45789 be delivered?",

"intent": "order_status",

"response": "I'd be happy to check that for you. Order #45789 is currently in transit and scheduled for delivery tomorrow. You can track your package in real-time through your account dashboard.",

"follow_up_questions": "Can I change the delivery address?,What if I'm not home for the delivery?",

"sentiment": "neutral",

"priority": "medium"

},

{

"customer_query": "I want to cancel my subscription",

"intent": "cancel_subscription",

"response": "I can help you cancel your subscription. To proceed, please confirm your account email address and the reason for cancellation.

This helps us improve our services.",
"follow_up_questions": "Will I get a refund?,Can I pause instead of cancel?",
"sentiment": "negative",
"priority": "high"
},
{
"customer_query": "Do you offer student discounts?",
"intent": "discount_inquiry",
"response": "Yes, we offer a 15% discount for students with valid ID. You can verify your student status through our website by clicking on 'Student Discount' and following the verification process.",
"follow_up_questions": "What verification do you need?,Does the discount apply to all products?",
"sentiment": "positive",
"priority": "low"
},
{
"customer_query": "I received a damaged product",
"intent": "product_issue",
"response": "I'm very sorry to hear that. We'll make this right. Could you please provide your order number and attach a photo of the damaged item? We'll arrange for a replacement or refund right away.",
"follow_up_questions": "Do I need to return the damaged item?,How long will the replacement take?",
"sentiment": "negative",
"priority": "high"
},
{
"customer_query": "How do I update my billing information?",
"intent": "account_management",
"response": "You can update your billing information by logging into your account, going to 'Settings', then selecting 'Payment Methods'. From there, you can edit existing information or add new payment methods.",
"follow_up_questions": "Can I have multiple payment methods?,Is

```
    my payment information secure?",
            "sentiment": "neutral",
            "priority": "medium"
        },
        {
            "customer_query": "Do you ship internationally?",
            "intent": "shipping_inquiry",
            "response": "Yes, we ship to over 50 countries worldwide.
    International shipping typically takes 7-14 business days and costs vary by
    destination. You can see specific shipping rates for your country during
    checkout before finalizing your purchase.",
            "follow_up_questions": "Do I have to pay customs fees?,Can I track
    international orders?",
            "sentiment": "neutral",
            "priority": "low"
        }
    ]

    df = pd.DataFrame(data)
    df.to_csv(filename, index=False)
    print(f"Sample CSV created: {filename}")
    return filename

# Create a Colab-friendly interface for the chatbot
def create_colab_interface():
    """Create an interactive interface for Google Colab"""
    # First create a chatbot instance
    print("Initializing chatbot...")

    # Create a sample CSV file for testing if needed
    sample_csv = create_sample_csv()

    # Initialize chatbot with the sample CSV
    chatbot = CustomerSupportChatbot(sample_csv)
```

```
# CSS for styling
display(HTML("""
<style>
    .chat-message {
        padding: 8px 12px;
        border-radius: 15px;
        margin: 5px 0;
        max-width: 80%;
        word-wrap: break-word;
    }
    .user-message {
        background-color: #e5e5ea;
        margin-left: auto;
        margin-right: 10px;
        text-align: right;
    }
    .bot-message {
        background-color: #0084ff;
        color: white;
        margin-right: auto;
        margin-left: 10px;
    }
    .example-btn {
        margin: 5px;
        padding: 5px 10px;
        background-color: #f0f0f0;
        border: 1px solid #ddd;
        border-radius: 15px;
        cursor: pointer;
        display: inline-block;
    }
    .example-btn:hover {
```

```
        background-color: #e0e0e0;
    }
    .follow-up-btn {
        margin: 3px;
        padding: 3px 8px;
        background-color: #e8f4ff;
        border: 1px solid #cce7ff;
        border-radius: 12px;
        cursor: pointer;
        display: inline-block;
        font-size: 12px;
    }
    .follow-up-btn:hover {
        background-color: #d4eaff;
    }
    .chat-container {
        margin-top: 10px;
        border: 1px solid #ddd;
        border-radius: 10px;
        padding: 10px;
        background-color: #f9f9f9;
    }
    .separator {
        border-top: 1px solid #ddd;
        margin: 10px 0;
    }
</style>
"""))

# Store conversation
conversation_output = widgets.Output()

# Create input widget
```

```
user_input = widgets.Text(
    value='',
    placeholder='Type your message here...',
    description='You:',
    layout=widgets.Layout(width='80%')
)

# Create send button
send_button = widgets.Button(
    description='Send',
    button_style='primary',
    tooltip='Send message',
    icon='paper-plane'
)

# Create clear button
clear_button = widgets.Button(
    description='Clear Chat',
    button_style='danger',
    tooltip='Clear the conversation',
    icon='trash'
)

# Upload widget for custom dataset
upload_button = widgets.FileUpload(
    accept='.csv',
    multiple=False,
    description='Upload CSV:',
    layout=widgets.Layout(width='300px')
)

# Function to upload custom dataset
def on_upload_change(change):
```

```
    if not change.new:
        return

    uploaded_file = next(iter(change.new.values()))
    filename = uploaded_file['name']
    content = uploaded_file['content']

    # Save uploaded content to a temporary file
    with open(filename, 'wb') as f:
        f.write(content)

    # Reinitialize chatbot with new dataset
    nonlocal chatbot
    chatbot = CustomerSupportChatbot(filename)

    # Update example questions widget
    update_example_questions_widget(chatbot.example_questions)

    with conversation_output:
        clear_output()
        display(HTML('<div class="chat-message bot-message">Hi there!
How can I help you today?</div>'))

upload_button.observe(on_upload_change, names='value')

# Function to handle send button click
def on_send_button_clicked(b):
    handle_input(user_input.value)
    user_input.value = ''

# Function to handle Enter key press
def on_enter_pressed(widget):
    if widget.value.strip():
```

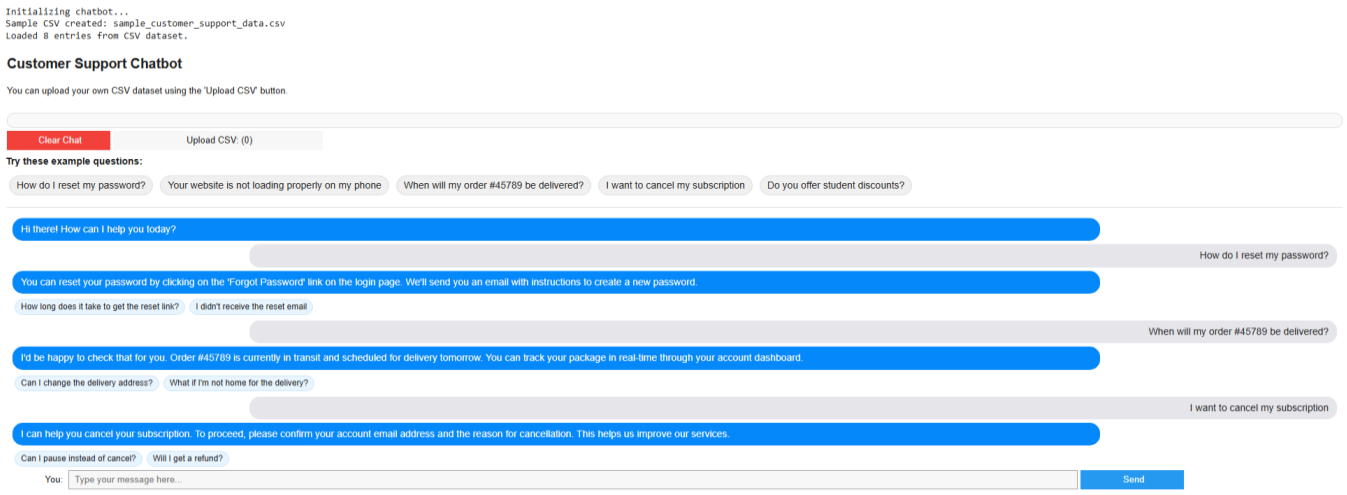
```
        handle_input(widget.value)
        widget.value = ''

# Function to handle user input
def handle_input(input_text):
    if not input_text.strip():
        return

    # Display user message
    with conversation_output:
        display(HTML(f'<div class="chat-message user-message">{input_text}</div>'))

    # Get chatbot response
    response = chatbot.process_input(input_text)

    # Display bot response
    with conversation_output:
        display(HTML(f'<div class="chat-message bot-message">{response}</div>'))

        # Display follow-up suggestions
        follow_ups = chatbot.get_follow_up_questions()
        if follow_ups:
            follow_ups_html = '<div style="margin-left: 10px; margin-top: 5px;">'
            for question in follow_ups:
                follow_ups_html += f'<div class="follow-up-btn" onclick="suggestQuestion(\'{question}\')">{question}</div>'
            follow_ups_html += '</div>'
            display(HTML(follow_ups_html))

            # Add JavaScript to make follow-up buttons work
            display(HTML("""
```

```
            <script>
            function suggestQuestion(question) {
                // Find the input element and set its value
                var inputElements = document.getElementsByTagName('input');
                for (var i = 0; i < inputElements.length; i++) {
                    if (inputElements[i].placeholder === 'Type your message
here...') {
                        inputElements[i].value = question;
                        // Trigger the enter key event
                        var event = new KeyboardEvent('keydown', {
                            key: 'Enter',
                            code: 'Enter',
                            keyCode: 13,
                            which: 13,
                            bubbles: true
                        });
                        inputElements[i].dispatchEvent(event);
                        break;
                    }
                }
            }
            </script>
            """))

 # Function to clear conversation
 def on_clear_button_clicked(b):
     with conversation_output:
         clear_output()
         display(HTML('<div class="chat-message bot-message">Hi there!
How can I help you today?</div>'))

 # Connect event handlers
 send_button.on_click(on_send_button_clicked)
```

```
user_input.on_submit(on_enter_pressed)
clear_button.on_click(on_clear_button_clicked)

# Create example questions widget
example_questions_output = widgets.Output()

def update_example_questions_widget(examples):
    with example_questions_output:
        clear_output()
        html = '<div><p><strong>Try these example
questions:</strong></p>'
        for question in examples:
            html += f'<div class="example-btn"
onclick="setExampleQuestion(\'{question}\')">{question}</div>'
        html += '</div>'
        display(HTML(html))

        # Add JavaScript to make example buttons work
        display(HTML("""
        <script>
        function setExampleQuestion(question) {
            // Find the input element and set its value
            var inputElements = document.getElementsByTagName('input');
            for (var i = 0; i < inputElements.length; i++) {
                if (inputElements[i].placeholder === 'Type your message
here...') {
                    inputElements[i].value = question;
                    // Focus on the input
                    inputElements[i].focus();
                    break;
                }
            }
        }
        </script>
```

```
        """))

    # Initialize example questions widget
    update_example_questions_widget(chatbot.example_questions)

    # Display initial message
    with conversation_output:
        display(HTML('<div class="chat-message bot-message">Hi there! How can I help you today?</div>'))

    # Arrange widgets
    input_area = widgets.HBox([user_input, send_button])
    controls = widgets.HBox([clear_button, upload_button])

    # Display the interface
    display(widgets.HTML("<h2>Customer Support Chatbot</h2>"))
    display(widgets.HTML("<p>You can upload your own CSV dataset using the 'Upload CSV' button.</p>"))
    display(widgets.HTML('<div class="chat-container">'))
    display(controls)
    display(example_questions_output)
    display(widgets.HTML('<div class="separator"></div>'))
    display(conversation_output)
    display(input_area)
    display(widgets.HTML('</div>'))

# Run the interface when the notebook is executed
create_colab_interface()
```

#Output:



#Program for measuring performance:

```
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score,
  f1_score, classification_report

# Load chatbot log file
# Your CSV file must have columns: user_input, predicted_intent,
  actual_intent
df = pd.read_csv("/content/sample_customer_support_data.csv")

# Assuming 'intent' is the actual intent, and we'll predict intent using the
  chatbot
# to simulate 'predicted_intent'.
# Here, we're creating a new DataFrame with 'user_input' and 'actual_intent'
# and adding a 'predicted_intent' column using the chatbot's predictions.

# Instead of importing from 'ipython_input_1_e8c08db3296f',
# import the CustomerSupportChatbot directly from the current notebook or
  file.
```

```
# Assuming the CustomerSupportChatbot class is defined in the same
  notebook:
from __main__ import CustomerSupportChatbot # Import from the current
  notebook


chatbot =
  CustomerSupportChatbot("/content/sample_customer_support_data.csv")


# Create a new DataFrame with 'user_input' (customer_query) and
  'actual_intent' (intent)
eval_df = df[['customer_query',
  'intent']].rename(columns={'customer_query': 'user_input', 'intent':
  'actual_intent'})


# Add 'predicted_intent' column using the chatbot
eval_df['predicted_intent'] = eval_df['user_input'].apply(chatbot.find_intent)
  # Assuming find_intent predicts the intent


# Extract predicted and actual labels
y_true = eval_df['actual_intent']
y_pred = eval_df['predicted_intent']


# === Evaluation Metrics ===
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted',
  zero_division=0)
recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)


print("=== Chatbot Performance Metrics ===")
print(f"Accuracy : {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall   : {recall:.4f}")
print(f"F1-score : {f1:.4f}")
```

```
print("\n=== Detailed Classification Report ===")
print(classification_report(y_true, y_pred))
```

#Output:

```
Loaded 8 entries from CSV dataset.
=== Chatbot Performance Metrics ===
Accuracy : 1.0000
Precision: 1.0000
Recall   : 1.0000
F1-score : 1.0000

=== Detailed Classification Report ===
                    precision    recall  f1-score   support

 account_management       1.00      1.00      1.00         1
cancel_subscription       1.00      1.00      1.00         1
   discount_inquiry       1.00      1.00      1.00         1
       order_status       1.00      1.00      1.00         1
     password_reset       1.00      1.00      1.00         1
      product_issue       1.00      1.00      1.00         1
   shipping_inquiry       1.00      1.00      1.00         1
    technical_issue       1.00      1.00      1.00         1

           accuracy                           1.00         8
          macro avg       1.00      1.00      1.00         8
       weighted avg       1.00      1.00      1.00         8
```