**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:

# 