1.Upload Dataset

```
# Step 1: Import required libraries
from google.colab import files
import pandas as pd

# Step 2: Upload the file
uploaded = files.upload()

# Step 3: Load the file into a pandas DataFrame (assuming it's a C
import io
df = pd.read_csv(io.BytesIO(uploaded[next(iter(uploaded))]))

# Step 4: Display the first few rows
df.head()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving student-dataset.csv to student-dataset.csv
```

	school;sex;age;address;famsize;Pstatus;Medu;Fedu;Mjob;Fjob
0	
1	
2	
3	
4	

2.Load the Dataset

```
# Step 1: Import necessary library
import pandas as pd

# Step 2: Load the dataset from a local path
# Replace 'your_dataset.csv' with your actual filename or path
# Assuming the dataset was uploaded in the previous cell, use the
df = pd.read_csv(io.BytesIO(uploaded[next(iter(uploaded))]))

# Step 3: Display basic information about the dataset
print("✅ Dataset Loaded Successfully!")
print(f"Shape of the dataset: {df.shape}")
print("Column names:", df.columns.tolist())
print("\n🔍 Preview of the dataset:")
print(df.head())
```

```
✅ Dataset Loaded Successfully!
Shape of the dataset: (395, 1)
Column names: ['school;sex;age;address;famsize;Pstatus;Medu;Fe

🔍 Preview of the dataset:
  school;sex;age;address;famsize;Pstatus;Medu;Fedu;Mjob;Fjob;r
0  GP;"F";18;"U";"GT3";"A";4;4;"at_home";"teacher...
1  GP;"F";17;"U";"GT3";"T";1;1;"at_home";"other";...
```

File "<ipython-input-2-9a7000ef27b •••

Please explain the error:

```
  File "<ipython-input-2-9a7000ef27
    import random
                ^
IndentationError: expected an inder
```

Hypothesis

The error is an `IndentationError` and it happens because the `import random` statement on line 18 is not indented correctly. In Python, indentation is crucial for defining code blocks, especially within conditional statements like `if`, `elif`, and `else`. The `elif` statement on line 17 expects an indented block of code to execute if the condition is True. However, the next line (`import random`) is not indented, which violates Python's syntax rules and leads to the `IndentationError`.

Suggested Changes

```
import random

# Simulated trained intents and res
intents = {
    "greeting": ["Hello! How can I
    "hours": ["Our business hours a
    "reset_password": ["You can res
    "shipping": ["Yes, we do offer
    "fallback": ["I'm sorry, I didr
}

# Basic keyword-based intent predic
def predict_intent(user_input):
    user_input = user_input.lower()
    if "hello" in user_input or "hi
        return "greeting"
    elif "hours" in user_input or '
        return "hours" # This line
    elif "password" in user_input c
        return "reset_password"
    elif "shipping" in user_input c
```

```
2  GP;"F";15;"U";"LE3";"T";1;1;"at_home";"other";...
3  GP;"F";15;"U";"GT3";"T";4;2;"health";"services...
4  GP;"F";16;"U";"GT3";"T";3;3;"other";"other";"h...
```

3.Data Exploration

```
# Step 1: Import required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io # Import io module

# Step 2: Load the dataset
# Assuming the dataset was uploaded in the previous cells, use the
df = pd.read_csv(io.BytesIO(uploaded[next(iter(uploaded))]))  # Us

# Step 3: Preview the data
print("🔍 Dataset Preview:")
print(df.head())
```

```
ataset Preview:
ıool;sex;age;address;famsize;Pstatus;Medu;Fedu;Mjob;Fjob;reason
';"F";18;"U";"GT3";"A";4;4;"at_home";"teacher...
';"F";17;"U";"GT3";"T";1;1;"at_home";"other";...
';"F";15;"U";"LE3";"T";1;1;"at_home";"other";...
';"F";15;"U";"GT3";"T";4;2;"health";"services...
';"F";16;"U";"GT3";"T";3;3;"other";"other";"h...
```

4.Check for Missing Values and Duplicates

```
# Step 1: Import the required library
import pandas as pd
import io

# Step 2: Load your dataset - Using the uploaded file instead of '
```

5.Visualize a Few Features

```
# Step 1: Import required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io

# Step 2: Load the dataset
# Instead of reading from 'your_dataset.csv', use the uploaded dat
# df = pd.read_csv('your_dataset.csv')  #
```

6.Identify Target and Features

```
        return "shipping"
    else:
        return "fallback"

# Get user input
user_input = input("You: ")

# Predict intent
predicted_intent = predict_intent(u

# Get a random response from the cc
if predicted_intent in intents:
    bot_response = random.choice(ir
    print("Bot:", bot_response)
else:
    print("Bot:", "I'm sorry, I dic
```

```
# ... previous code ...

# Step 5: Identify Feature Columns
# Assuming 'user_query' or 'text' is the main input feature
# Check if either 'user_query' or 'text' or the first column name
# If none of these exist, consider using another relevant column a
# Instead of using index 1 directly, check if it exists first
feature_columns = ['user_query'] if 'user_query' in df.columns els
print(f"🧩 Feature Column(s): {feature_columns}")

# Step 6: Extract Features (X) and Target (y)
# Check if feature_columns is not empty before accessing it
X = df[feature_columns[0]] if feature_columns else pd.Series([])
y = df[target_column]       # labels

# ... rest of the code ...
```

```
🧩 Feature Column(s): ['school;sex;age;address;famsize;Pstatu
```

7.Convert Categorical Columns to Numerical

```
# Step 1: Import required libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Step 2: Instead of loading from file, use the existing DataFrame
# df = pd.read_csv('your_dataset.csv')  # Remove this line
```

8.Encode Multiple Categorical Columns

```
# Suppose you also have other categorical features (like 'user_typ
categorical_cols = ['intent', 'user_type']  # Update with your col

# Apply Label Encoding to each
label_encoders = {}
for col in categorical_cols:
    # Check if the column exists in the DataFrame before encoding
    if col in df.columns:
        le = LabelEncoder()
        df[col + '_encoded'] = le.fit_transform(df[col])
        label_encoders[col] = le
    else:
        print(f"⚠️ Column '{col}' not found in DataFrame. Skippin
```

```
⚠️ Column 'intent' not found in DataFrame. Skipping encoding.
⚠️ Column 'user_type' not found in DataFrame. Skipping encodi
```

9.One-Hot Encoding

```
# Step 1: Import required libraries
import pandas as pd

# Step 2: Instead of loading from file, use the existing DataFrame
```

```
# df = pd.read_csv('your_dataset.csv')  # Replace with your datase
# Use the existing DataFrame 'df'

# Step 3: Check if 'intent' column exists, and if not, create it
# --- Added this code block ---
if 'intent' not in df.columns:
    # If 'intent' doesn't exist, assume the first column contains
    # (You might need to adjust based on your data structure)
    if len(df.columns) > 0:
        df.rename(columns={df.columns[0]: 'intent'}, inplace=True)
    else:
        raise KeyError("The DataFrame is empty or doesn't have any
# --- End of added code block ---

# Step 3: Apply One-Hot Encoding to the 'intent' column
df_encoded = pd.get_dummies(df, columns=['intent'])

# Step 4: Preview the result
print("✅ One-Hot Encoded DataFrame:")
print(df_encoded.head())
```

```
   intent_MS;"M";20;"U";"LE3";"A";2;2;"services";"services
0                                                  False
1                                                  False
2                                                  False
3                                                  False
4                                                  False

   intent_MS;"M";21;"R";"GT3";"T";1;1;"other";"other";"cour
0                                                  False
1                                                  False
2                                                  False
3                                                  False
4                                                  False

[5 rows x 395 columns]
```

10. Feature Scaling

```
# Step 1: Import required libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Step 2: Use the existing DataFrame (df) instead of loading from
# df = pd.read_csv('your_dataset.csv')  # This line is removed
# The DataFrame 'df' from previous cells is used

# Step 3: Select numerical features for scaling
# Exclude text fields like 'user_query' or categorical columns unl
# Assuming 'intent_encoded' is a numerical column if 'intent' was
numerical_cols = df.select_dtypes(include=['number']).columns.toli
# Remove 'intent' if it's not encoded or is the original categoric
if 'intent' in numerical_cols and 'intent_encoded' not in numerica
    numerical_cols.remove('intent')

# Optional: Manually specify numerical columns if needed
# numerical_cols = ['feature1', 'feature2', 'intent_encoded']

# Step 4: Initialize the scaler
scaler = StandardScaler()

# Step 5: Fit and transform the numerical columns
df_scaled = df.copy()
if numerical_cols:  # Check if there are any numerical columns to
    df_scaled[numerical_cols] = scaler.fit_transform(df[numerical_
else:
    print("⚠️ No numerical columns found for scaling.")

# Step 6: Preview the scaled dataset
print("✅ Scaled Feature Preview:")
if numerical_cols:
    print(df_scaled[numerical_cols].head())
else:
    print("No scaling performed.")
```

```
⚠️ No numerical columns found for scaling.
✅ Scaled Feature Preview:
No scaling performed.
```

11.Train-Test Split

```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split

# Step 2: Use the existing DataFrame (df) instead of loading from
# df = pd.read_csv('your_dataset.csv')  # Remove this line, use ex

# Step 3: Define features (X) and target (y)
# Assuming 'user_query' is the feature and 'intent' is the target
# Adjust column names if needed based on your DataFrame
X = df['intent']  # Replace 'user_query' or 'text' with your featu
y = df['intent']  # Replace 'intent' or 'intent_encoded' with your

# Step 4: Perform train-test split
# Check if stratification is possible
if len(y.unique()) > 1 and y.value_counts().min() >= 2:
    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=0.2,       # 20% for testing
        random_state=42,     # for reproducibility
        stratify=y           # preserve label distribution
    )
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split

# Step 2: Use the existing DataFrame (df) instead of loading from
# df = pd.read_csv('your_dataset.csv')  # Remove this line, use ex

# Step 3: Define features (X) and target (y)
# Assuming 'user_query' is the feature and 'intent' is the target
# Adjust column names if needed based on your DataFrame
X = df['intent']  # Replace 'user_query' or 'text' with your featu
y = df['intent']  # Replace 'intent' or 'intent_encoded' with your

# Step 4: Perform train-test split
# Check if stratification is possible
if len(y.unique()) > 1 and y.value_counts().min() >= 2:
    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=0.2,       # 20% for testing
        random_state=42,     # for reproducibility
        stratify=y           # preserve label distribution
    )
else:
    # If stratification is not possible, remove 'stratify' argumer
    print("⚠️ Stratification is not possible. Performing a regula
    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=0.2,       # 20% for testing
        random_state=42      # for reproducibility
    )
```

```
⚠️ Stratification is not possible. Performing a regular split
```

12.Model Building

```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import io

# Step 2: Load the dataset - Use the in-memory dataframe `df` inst
# df = pd.read_csv('your_dataset.csv')  # Replace with your actual
```

13. Evaluation Code

```
from sklearn.metrics import accuracy_score, precision_score, recal
from difflib import SequenceMatcher

# Simulated chatbot response function (to be replaced with your ac
def chatbot_response(input_text):
    # Example dummy response (replace with your chatbot)
    responses = {
        "What are your business hours?": "Our business hours are 9
        "How can I reset my password?": "You can reset your passwo
        "Do you offer international shipping?": "Yes, we ship inte
    }
    return responses.get(input_text, "I'm sorry, I didn't understa

# Sample test data
test_data = [
    {"input": "What are your business hours?", "expected": "Our bu
    {"input": "How can I reset my password?", "expected": "You car
    {"input": "Do you offer international shipping?", "expected":
]

# Evaluation metrics
def evaluate_chatbot(test_data):
    similarities = []
    for item in test_data:
        user_input = item["input"]
        expected_output = item["expected"]
        actual_output = chatbot_response(user_input)
        similarity = SequenceMatcher(None, expected_output, actual
        similarities.append(similarity)
        print(f"\nUser Input: {user_input}")
        print(f"Expected Response: {expected_output}")
        print(f"Actual Response:   {actual_output}")
        print(f"Similarity Score:  {similarity:.2f}")

    average_similarity = sum(similarities) / len(similarities)
    print(f"\nAverage Similarity: {average_similarity:.2f}")
    return average_similarity

# Run evaluation
evaluate_chatbot(test_data)
```

```
User Input: What are your business hours?
Expected Response: Our business hours are 9 AM to 5 PM, Monday
Actual Response:   Our business hours are 9 AM to 5 PM, Monday
Similarity Score:  1.00

User Input: How can I reset my password?
Expected Response: You can reset your password using the 'Forg
Actual Response:   You can reset your password using the 'Forg
Similarity Score:  1.00

User Input: Do you offer international shipping?
Expected Response: Yes, we ship internationally. Delivery time
Actual Response:   Yes, we ship internationally. Delivery time
Similarity Score:  1.00

Average Similarity: 1.00
1.0
```

14.Make Predictions from New Input

```
import random
import random

# Simulated trained intents and responses
intents = {
    "greeting": ["Hello! How can I assist you today?", "Hi there!
    "hours": ["Our business hours are 9 AM to 5 PM, Monday to Fric
    "reset_password": ["You can reset your password using the 'For
    "shipping": ["Yes, we do offer international shipping. Deliver
}

# Simulated trained intents and responses
intents = {
    "greeting": ["Hello! How can I assist you today?", "Hi there!
    "hours": ["Our business hours are 9 AM to 5 PM, Monday to Fric
    "reset_password": ["You can reset your password using the 'For
    "shipping": ["Yes, we do offer international shipping. Deliver
}
```

15.Convert to DataFrame and Encode

```
import pandas as pd

# Sample chatbot training data
data = {
    "patterns": [
        "Hi", "Hello", "Hey",
        "What are your hours?", "When do you open?",
        "How can I reset my password?", "I forgot my password",
        "Do you ship internationally?", "Is overseas delivery avai
    ],
    "intent": [
        "greeting", "greeting", "greeting",
        "hours", "hours",
        "reset_password", "reset_password",
        "shipping", "shipping"
```

```
    ]
}

# Convert to DataFrame
df = pd.DataFrame(data)
print("DataFrame:\n", df)
```

```
DataFrame:
                            patterns          intent
0                                 Hi        greeting
1                              Hello        greeting
2                                Hey        greeting
3             What are your hours?           hours
4                When do you open?           hours
5     How can I reset my password?  reset_password
6             I forgot my password  reset_password
7     Do you ship internationally?        shipping
8  Is overseas delivery available?        shipping
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer

# Encode the intent labels
le = LabelEncoder()
df['intent_encoded'] = le.fit_transform(df['intent'])

# Convert user input text to numerical features
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['patterns']).toarray()

# Labels
y = df['intent_encoded']

print("\nEncoded Intents:\n", df[['intent', 'intent_encoded']])
print("\nFeature Names (Vocabulary):\n", vectorizer.get_feature_na
print("\nVectorized Input Features:\n", X)
```

```
Encoded Intents:
            intent  intent_encoded
0        greeting               0
1        greeting               0
2        greeting               0
3           hours               1
4           hours               1
5  reset_password               2
6  reset_password               2
7        shipping               3
8        shipping               3

Feature Names (Vocabulary):
 ['are' 'available' 'can' 'delivery' 'do' 'forgot' 'hello' 'he
 'hours' 'how' 'internationally' 'is' 'my' 'open' 'overseas' '
 'reset' 'ship' 'what' 'when' 'you' 'your']

Vectorized Input Features:
 [[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1]
```

```
       [0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0]
       [0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0]
       [0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0]
       [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0]
       [0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]]
```

```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(X, y)

# Predict new input
new_input = ["How do I change my password?"]
new_vector = vectorizer.transform(new_input).toarray()
predicted = model.predict(new_vector)

# Decode predicted intent
predicted_intent = le.inverse_transform(predicted)
print("\nPredicted intent:", predicted_intent[0])
```

```
Predicted intent: reset_password
```

16.Predict the Final Grade

```
import pandas as pd

# Sample chatbot performance data
data = {
    "accuracy": [0.85, 0.90, 0.80, 0.75, 0.95],
    "response_time": [1.2, 0.8, 1.5, 1.8, 0.5],  # seconds
    "queries_resolved": [90, 95, 80, 70, 98],    # out of 100
    "user_satisfaction": [4.5, 4.8, 4.2, 3.9, 4.9],  # out of 5
    "final_grade": [88, 92, 85, 78, 95]          # target to predi
}

df = pd.DataFrame(data)
print("Chatbot Performance Data:\n", df)
```

```
Chatbot Performance Data:
    accuracy  response_time  queries_resolved  user_satisfacti
0      0.85            1.2                90               4.
1      0.90            0.8                95               4.
2      0.80            1.5                80               4.
3      0.75            1.8                70               3.
4      0.95            0.5                98               4.
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Features and target
X = df.drop(columns=["final_grade"])
y = df["final_grade"]
```

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)
print("\nPredicted Grade(s):", y_pred)
print("Actual Grade(s):", y_test.values)
print("MSE:", mean_squared_error(y_test, y_pred))
```

```
Predicted Grade(s): [92.52844837]
Actual Grade(s): [92]
MSE: 0.27925768396293
```

```
# New chatbot performance input (as an example)
new_chatbot_metrics = pd.DataFrame([{
    "accuracy": 0.88,
    "response_time": 1.0,
    "queries_resolved": 93,
    "user_satisfaction": 4.6
}])

predicted_grade = model.predict(new_chatbot_metrics)
print("\nPredicted Final Grade for New Input:", predicted_grade[0])
```

```
Predicted Final Grade for New Input: 90.68104547623025
```

17. Deployment-Building an Interactive App

```
pip install streamlit
```

```
Collecting streamlit
  Downloading streamlit-1.45.0-py3-none-any.whl.metadata (8.9
Requirement already satisfied: altair<6,>=4.0 in /usr/local/li
Requirement already satisfied: blinker<2,>=1.5.0 in /usr/local
Requirement already satisfied: cachetools<6,>=4.0 in /usr/loca
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/li
Requirement already satisfied: packaging<25,>=20 in /usr/local
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local
Requirement already satisfied: protobuf<7,>=3.20 in /usr/local
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/
Requirement already satisfied: requests<3,>=2.27 in /usr/local
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/loc
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/l
Requirement already satisfied: typing-extensions<5,>=4.4.0 in
Collecting watchdog<7,>=2.1.5 (from streamlit)
  Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 3.2
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 ir
```

```
  Collecting pydeck<1,>=0.8.0b4 (from streamlit)
    Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1
  Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local
  Requirement already satisfied: jinja2 in /usr/local/lib/python
  Requirement already satisfied: jsonschema>=3.0 in /usr/local/l
  Requirement already satisfied: narwhals>=1.14.2 in /usr/local/
  Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/l
  Requirement already satisfied: python-dateutil>=2.8.2 in /usr/
  Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/
  Requirement already satisfied: tzdata>=2022.7 in /usr/local/li
  Requirement already satisfied: charset-normalizer<4,>=2 in /us
  Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/
  Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/loca
  Requirement already satisfied: certifi>=2017.4.17 in /usr/loca
  Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/l
  Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/l
  Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib
  Requirement already satisfied: jsonschema-specifications>=2023
  Requirement already satisfied: referencing>=0.28.4 in /usr/loc
  Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/li
  Requirement already satisfied: six>=1.5 in /usr/local/lib/pyth
  Downloading streamlit-1.45.0-py3-none-any.whl (9.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 51.0 MB
  Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 70.9 MB
  Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl (
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 7.1 M
  Installing collected packages: watchdog, pydeck, streamlit
  Successfully installed pydeck-0.9.1 streamlit-1.45.0 watchdog-
```

```
# app.py

import streamlit as st
import random

# Define intents and responses
intents = {
    "greeting": ["Hello! How can I assist you today?", "Hi there!
    "hours": ["Our business hours are 9 AM to 5 PM, Monday to Fric
    "reset_password": ["You can reset your password using the 'For
    "shipping": ["Yes, we offer international shipping. Delivery t
    "fallback": ["I'm sorry, I didn't understand that. Could you p
}

# Simple intent prediction (keyword-based)
def predict_intent(user_input):
    user_input = user_input.lower()
    if any(greet in user_input for greet in ["hello", "hi", "hey"]
        return "greeting"
    elif "hours" in user_input or "open" in user_input:
        return "hours"
    elif "password" in user_input or "reset" in user_input:
        return "reset_password"
    elif "shipping" in user_input or "international" in user_input
        return "shipping"
    else:
        return "fallback"

# Get chatbot response
def get_response(user_input):
```

```
    intent = predict_intent(user_input)
    return random.choice(intents[intent])

# Streamlit app
st.set_page_config(page_title="Customer Support Chatbot", layout="
st.title("💬 Intelligent Customer Support Chatbot")

user_input = st.text_input("You:", placeholder="Ask me something..

if st.button("Send") and user_input:
    response = get_response(user_input)
    st.markdown(f"**Chatbot:** {response}")
```

```
2025-05-10 07:33:16.442 WARNING streamlit.runtime.scriptrunner
2025-05-10 07:33:16.444 WARNING streamlit.runtime.scriptrunner
2025-05-10 07:33:16.594
  Warning: to view this Streamlit app on a browser, run it wit
  command:

    streamlit run /usr/local/lib/python3.11/dist-packages/cola
2025-05-10 07:33:16.596 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.597 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.599 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.600 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.601 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.602 Session state does not function when r
2025-05-10 07:33:16.603 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.605 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.607 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.608 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.609 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.610 Thread 'MainThread': missing ScriptRur
2025-05-10 07:33:16.611 Thread 'MainThread': missing ScriptRur
```

```
!streamlit run app.py
```

```
Usage: streamlit run [OPTIONS] TARGET [ARGS]...
Try 'streamlit run --help' for help.

Error: Invalid value: File does not exist: app.py
```

Enter a prompt here

0/2000

Gemini can make mistakes, so double-check responses and use code with caution. Learn more