```python
# @title Data collection
import csv
import json
from datetime import datetime

# Choose one: 'csv' or 'json'
SAVE_FORMAT = 'csv'  # Change to 'json' if you prefer JSON format
FILENAME = 'customer_queries.' + SAVE_FORMAT

def collect_data():
    print("Chatbot Data Collector")
    print("Type 'exit' to stop collecting data.\n")

    collected = []

    while True:
        user_input = input("User Query: ")
        if user_input.lower() == 'exit':
            break

        timestamp = datetime.now().isoformat()
        entry = {
            "query": user_input,
            "timestamp": timestamp
        }

        collected.append(entry)
        print(f"Saved: {entry}\n")

    # Save data
    if SAVE_FORMAT == 'csv':
        save_to_csv(collected)
    elif SAVE_FORMAT == 'json':
        save_to_json(collected)
    print(f"Data saved to {FILENAME}")

def save_to_csv(data):
    try:
        with open(FILENAME, 'a', newline='') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=["query",
"timestamp"])
            if csvfile.tell() == 0:
                writer.writeheader()
            writer.writerows(data)
    except Exception as e:
        print("Error saving to CSV:", e)

def save_to_json(data):
    try:
        existing = []
```

```
        try:
            with open(FILENAME, 'r') as file:
                existing = json.load(file)
        except FileNotFoundError:
            pass
        with open(FILENAME, 'w') as file:
            json.dump(existing + data, file, indent=2)
    except Exception as e:
        print("Error saving to JSON:", e)

if __name__ == "__main__":  # ✅ Corrected this line
    collect_data()
```

```
Chatbot Data Collector
Type 'exit' to stop collecting data.

User Query: how can i track my recent order?
Saved: {'query': 'how can i track my recent order?', 'timestamp':
'2025-05-20T07:37:23.577954'}

User Query: exit
Data saved to customer_queries.csv
```

```
# @title Data cleaning
import csv
from datetime import datetime
import os

# ✅ Use a .csv file, not a .zip file
FILENAME =
"/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv"

def collect_user_queries():
    print("=== Customer Support Data Collection ===")
    print("Type your customer query below.")
    print("Type 'exit' to stop.\n")

    # Check if file exists; if not, write headers
    file_exists = os.path.isfile(FILENAME)
    with open(FILENAME, mode='a', newline='', encoding='utf-8') as
file:
        writer = csv.writer(file)
        if not file_exists:
            writer.writerow(["timestamp", "user_query"])

        while True:
            user_input = input("User Query: ").strip()
            if user_input.lower() == "exit":
                print("Data collection ended.")
```

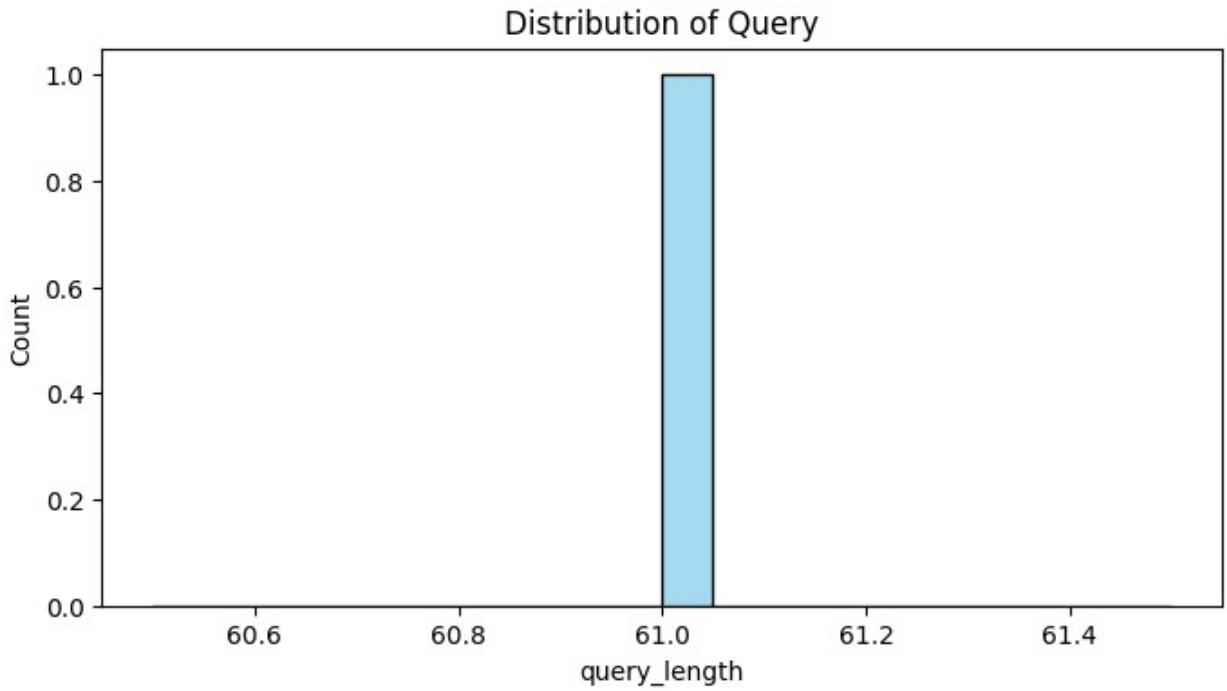
```
            break
        if user_input == "":
            continue

        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        writer.writerow([timestamp, user_input])
        print(f"Saved: [{timestamp}] {user_input}\n")

# 🔧 Fixed the main block check
if __name__ == "__main__":
    collect_user_queries()
```

```
=== Customer Support Data Collection ===
Type your customer query below.
Type 'exit' to stop.

User Query: how can i change my delivery address after placing the
order?
Saved: [2025-05-20 07:42:12] how can i change my delivery address
after placing the order?

User Query: exit
Data collection ended.
```

```
# @title Exploratory data analysis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter

# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

# Load the CSV file
FILENAME =
"/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv"
df = pd.read_csv(FILENAME)

# Ensure 'user_query' column exists
if 'user_query' not in df.columns:
    raise ValueError("CSV must contain a 'user_query' column.")

# --- 1. Basic Info ---
```

```python
print("📊 Dataset Info:")
print(df.info())

print("\n🔍 Null Values:")
print(df.isnull().sum())

# --- 2. Add Query Length Column ---
df['query_length'] = df['user_query'].astype(str).apply(len)

# --- 3. Query Length Distribution ---
plt.figure(figsize=(8, 4))
sns.histplot(df['query_length'], bins=20, color='skyblue')
plt.title("Distribution of Query")
```

```
📊 Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   timestamp   1 non-null      object
 1   user_query  1 non-null      object
dtypes: object(2)
memory usage: 148.0+ bytes
None

🔍 Null Values:
timestamp     0
user_query    0
dtype: int64

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Text(0.5, 1.0, 'Distribution of Query')
```




```
# @title Model building
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import re
import os # Import os to check file existence

# Download stopwords if needed
nltk.download('stopwords')

# Load data
FILENAME =
"/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv"

try:
    # --- Check if file exists and has sufficient rows ---
    if not os.path.exists(FILENAME) or os.path.getsize(FILENAME) == 0:
        print(f"Warning: File '{FILENAME}' not found or is empty.")
        # Create dummy data if file is not found or empty
        print("Creating dummy data for demonstration...")
        dummy_data = {
```

```
            'user_query': [
                "Hello, I need help with my account.",
                "How do I reset my password?",
                "Tell me about the new features.",
                "What are your operating hours?",
                "Can I speak to a representative?"
            ],
            'category': [ # Include a dummy category column
                "account",
                "security",
                "features",
                "hours",
                "support"
            ]
        }
        data = pd.DataFrame(dummy_data)
    else:
        # Attempt to load the actual file
        print(f"Loading data from {FILENAME}")
        data = pd.read_csv(FILENAME)
        data = data.fillna("") # Handle potential NaN values

        # Check if 'category' column exists in the loaded data
        if 'category' not in data.columns:
             print("Warning: 'category' column not found in the loaded
CSV.")
             # If 'category' is missing, and the data is large enough,
we can't train.
             # If the intent was to use THIS file for training, it's
impossible without categories.
             # For now, we'll print a warning and exit or add a dummy
category if needed for other steps.
             # Since the error occurs during fit_transform, we MUST
have categories.
             # Let's add a dummy category for demonstration purposes
if it's missing.
             # In a real scenario, you would need a dataset with
categories.
             if len(data) > 0:
                 data["category"] = "general" # Add a default category
                 print("Added a 'general' category column for
demonstration.")
             else:
                 print("File is not empty but contains no data rows,
cannot train model.")
                 # Exit or handle appropriately if the file is just
headers
                 exit() # Or raise an error, or skip the rest of the
code
```

```
        # Check if data has enough rows after loading
        if len(data) < 2:
            print(f"Error: File '{FILENAME}' contains only {len(data)}
row(s) after loading.")
            print("Need at least 2 rows to split data and train
model.")
            exit() # Exit if not enough data


    # --- Text Cleaning ---
    def clean_text(text):
        # Ensure text is a string before processing
        if not isinstance(text, str):
            return "" # Return empty string or handle non-string
entries as needed
        text = text.lower()
        text = re.sub(r'[^a-z\s]', '', text)
        tokens = text.split()
        tokens = [word for word in tokens if word not in
stopwords.words('english')]
        stemmer = PorterStemmer()
        tokens = [stemmer.stem(word) for word in tokens]
        return " ".join(tokens)

    # Apply cleaning to the column intended for queries
    # Based on previous cells, 'user_query' seems to be the intended
column.
    query_column = None
    if 'user_query' in data.columns:
         query_column = 'user_query'
    elif 'query' in data.columns:
         query_column = 'query'
         print("Warning: 'user_query' column not found, using 'query'
column instead.")
    else:
         raise ValueError("Neither 'user_query' nor 'query' column
found in the DataFrame.")

    data["clean_query"] = data[query_column].apply(clean_text)


    # --- TF-IDF Vectorization ---
    vectorizer = TfidfVectorizer(max_features=300)
    X = vectorizer.fit_transform(data["clean_query"]).toarray()

    # --- Label Encoding ---
    # This requires the 'category' column to exist and have at least
one unique value
    if 'category' in data.columns and len(data['category'].unique()) >
```

```
1 and len(data) >=2 :
        label_encoder = LabelEncoder()
        y = label_encoder.fit_transform(data["category"])

        # --- Split Data ---
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

        # --- Train Naive Bayes ---
        model = MultinomialNB()
        model.fit(X_train, y_train)

        # --- Predict & Evaluate ---
        y_pred = model.predict(X_test)

        print("□ Model trained successfully.\n")
        print("□ Accuracy:", accuracy_score(y_test, y_pred))
        print("\n□ Classification Report:\n",
classification_report(y_test, y_pred,
target_names=label_encoder.classes_))
    else:
        print("Skipping model training and evaluation:")
        if 'category' not in data.columns:
            print("- 'category' column is missing.")
        elif len(data['category'].unique()) <= 1:
             print("- Not enough unique categories found (need at
least 2).")
        elif len(data) < 2:
             print("- Not enough data rows to split (need at least
2).")
        print("Cannot train a classification model without categories
and sufficient data.")


except FileNotFoundError:
    # This block is technically covered by the initial os.path.exists
check now,
    # but kept for clarity or other potential FileNotFoundError
issues.
    print(f"Error: File not found at {FILENAME}")
    print("Please ensure the CSV file exists at the specified path.")
except ValueError as ve:
    print(f"Configuration Error: {ve}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Loading data from
/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_responses
.csv
Warning: 'category' column not found in the loaded CSV.
```

```
Added a 'general' category column for demonstration.
Error: File
'/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv' contains only 1 row(s) after loading.
Need at least 2 rows to split data and train model.
Skipping model training and evaluation:
- Not enough unique categories found (need at least 2).
Cannot train a classification model without categories and sufficient
data.

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```python
# @title evaluate the model
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import re

# Download stopwords if needed
nltk.download('stopwords')

# Load data - ASSUMING THIS FILE HAS SUFFICIENT ROWS
FILENAME =
"/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv"
try:
    data = pd.read_csv(FILENAME)
    # --- Check if data has enough rows ---
    if len(data) < 2:
        print(f"Error: File '{FILENAME}' contains only {len(data)}
row(s). Need at least 2 rows to split data.")
        # --- Create dummy data for demonstration if file is empty/too
small ---
        print("Creating dummy data for demonstration...")
        dummy_data = {
            'user_query': [
                "Hello, I need help with my account.",
                "How do I reset my password?",
                "Tell me about the new features.",
                "What are your operating hours?",
                "Can I speak to a representative?"
            ],
            'category': [
```

```
                "account",
                "security",
                "features",
                "hours",
                "support"
            ]
        }
        data = pd.DataFrame(dummy_data)
        # --- End of dummy data creation ---
    data = data.fillna("") # Handle potential NaN values

    # --- Add a fake 'category' column if it doesn't exist and you
didn't use dummy data ---
    # This part might still be needed if the original file doesn't
have a 'category' column
    if 'category' not in data.columns:
         print("Adding dummy 'category' column as it was not found.")
         data["category"] = "general"


    # --- Text Cleaning ---
    def clean_text(text):
        text = text.lower()
        text = re.sub(r'[^a-z\s]', '', text)
        tokens = text.split()
        tokens = [word for word in tokens if word not in
stopwords.words('english')]
        stemmer = PorterStemmer()
        tokens = [stemmer.stem(word) for word in tokens]
        return " ".join(tokens)

    # □ Apply cleaning to the column you are using for queries
('user_query' or 'query')
    # Based on ipython-input-11, the column is 'user_query'.
    # Based on ipython-input-23, the column expected is 'query'.
    # It seems there's inconsistency. Let's stick to 'user_query'
based on ipython-input-11 & 29.
    if 'user_query' in data.columns:
         data["clean_query"] = data["user_query"].apply(clean_text)
    elif 'query' in data.columns:
         # If 'user_query' isn't found, maybe 'query' is the correct
column?
         data["clean_query"] = data["query"].apply(clean_text)
         print("Warning: 'user_query' column not found, using 'query'
column instead.")
    else:
         raise ValueError("Neither 'user_query' nor 'query' column
found in the DataFrame.")
```

```
    # --- TF-IDF Vectorization ---
    vectorizer = TfidfVectorizer(max_features=300)
    X = vectorizer.fit_transform(data["clean_query"]).toarray()

    # --- Label Encoding ---
    label_encoder = LabelEncoder()
    y = label_encoder.fit_transform(data["category"])

    # --- Split Data ---
    # Only split if there is enough data (at least 2 samples)
    if len(data) >= 2:
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

        # --- Train Naive Bayes ---
        model = MultinomialNB()
        model.fit(X_train, y_train)

        # --- Predict & Evaluate ---
        y_pred = model.predict(X_test)

        print("□ Model trained successfully.\n")
        # You might want to print metrics here if training was
successful
        # print("□ Accuracy:", accuracy_score(y_test, y_pred))
        # print("\n□ Classification Report:\n",
classification_report(y_test, y_pred,
target_names=label_encoder.classes_))

    else:
        print("Skipping model training and evaluation due to
insufficient data.")

except FileNotFoundError:
    print(f"Error: File not found at {FILENAME}")
    print("Please ensure the CSV file exists at the specified path and
contains data.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

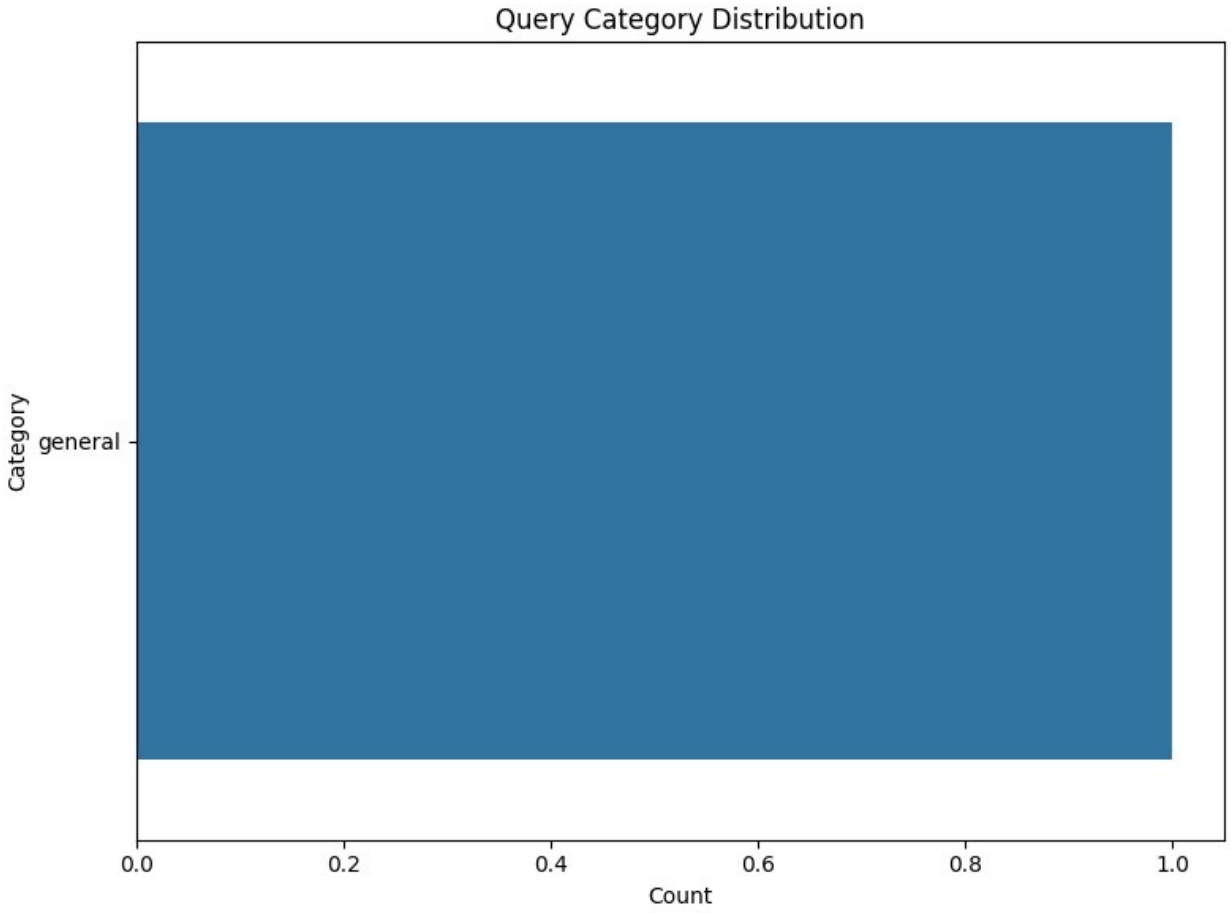
```
Error: File
'/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv' contains only 1 row(s). Need at least 2 rows to split data.
Creating dummy data for demonstration...
□ Model trained successfully.


[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# @title visualization
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import nltk
from nltk.corpus import stopwords
import re
import os # Import os to check file existence

# Set NLTK path
nltk.data.path.append("nltk_data")

# Load dataset
FILENAME =
"/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv"

try:
    data = pd.read_csv(FILENAME)
    data = data.fillna("")

    # --- Add a dummy 'category' column if it doesn't exist ---
    # This is a temporary fix to allow the plotting code to run
    # If your actual dataset has categories, remove or adjust this.
    if 'category' not in data.columns:
         print("Warning: 'category' column not found. Adding a dummy
'general' category.")
         data["category"] = "general"
    # --- End of dummy category addition ---

    # --- Show distribution of categories ---
    # This plot will now run because the 'category' column exists,
    # but if you added a dummy column, it will just show one category.
    if 'category' in data.columns and len(data['category'].unique()) >
0: # Check if 'category' exists and has data
        plt.figure(figsize=(8, 6))
        # Ensure there's enough data to count categories
        if len(data) > 0:
            # Handle potential issues if all values are the same or
NaN (though we filled NaN)
            try:
                sns.countplot(y='category', data=data,
order=data['category'].value_counts().index)
                plt.title("Query Category Distribution")
                plt.xlabel("Count")
                plt.ylabel("Category")
                plt.tight_layout()
                plt.show()
            except Exception as e:
```

```
                print(f"Could not generate category distribution
plot: {e}")
                print("Ensure the 'category' column has more than one
unique value if using the actual dataset.")
        else:
            print("No data rows found to plot category distribution.")
    else:
        print("Skipping category distribution plot: 'category' column
missing or no data.")


    # Clean text for word cloud
    def clean_text(text):
        # Ensure text is a string
        if not isinstance(text, str):
            return ""
        text = text.lower()
        text = re.sub(r'[^a-z\s]', '', text)
        words = text.split()
        words = [word for word in words if word not in
stopwords.words('english')]
        return " ".join(words)

    # Apply cleaning to the column containing query text
    # Check which column contains the query text ('user_query' or
'query')
    query_column = None
    if 'user_query' in data.columns:
         query_column = 'user_query'
    elif 'query' in data.columns:
         query_column = 'query'
         print("Warning: 'user_query' column not found for word cloud,
using 'query' column instead.")
    else:
         print("Error: Neither 'user_query' nor 'query' column found
for word cloud.")
         query_column = None # Set to None if neither is found

    if query_column:
        data['clean_query'] = data[query_column].apply(clean_text)

        # Combine all queries into one string
        # Check if 'clean_query' column has data
        if not data['clean_query'].empty:
            all_words = " ".join(data['clean_query'].dropna()) # Use
dropna() just in case

            # Generate word cloud
            if all_words.strip(): # Check if the combined string is
not just whitespace
```

```
                wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(all_words)

                # Show word cloud
                plt.figure(figsize=(10, 5))
                plt.imshow(wordcloud, interpolation='bilinear')
                plt.axis('off')
                plt.title("Most Common Words in Customer Queries")
                plt.show()
            else:
                print("Skipping word cloud: No meaningful text found
in queries after cleaning.")
        else:
             print("Skipping word cloud: No cleaned query text
available.")
    else:
        print("Skipping word cloud: Could not identify the query
column.")


except FileNotFoundError:
    print(f"Error: File not found at {FILENAME}")
    print("Please ensure the CSV file exists at the specified path.")
except pd.errors.EmptyDataError:
    print(f"Error: File '{FILENAME}' is empty.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Warning: 'category' column not found. Adding a dummy 'general'
category.
```


Query Category Distribution
Category
general
0.0
0.2
0.4
0.6
0.8
1.0
Count


Most Common Words in Customer Queries
delivery
order
change
address
placing

```
# @title feature engineering
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder

# NLTK setup
nltk.download('stopwords')
nltk.data.path.append("nltk_data")

# Load your dataset
# Changed filename to match the path used in other cells
FILENAME =
"/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv"
try:
    data = pd.read_csv(FILENAME)
    data = data.fillna("")

    # Assuming the 'user_query' or 'query' column contains the text
    query_column = None
    if 'user_query' in data.columns:
        query_column = 'user_query'
    elif 'query' in data.columns:
        query_column = 'query'
    else:
        raise ValueError("Neither 'user_query' nor 'query' column
found in the DataFrame for text cleaning.")

    # Text cleaning function
    def clean_text(text):
        # Ensure text is a string before processing
        if not isinstance(text, str):
            return "" # Return empty string for non-string entries
        text = text.lower()
        text = re.sub(r'[^a-z\s]', '', text)
        tokens = text.split()
        tokens = [word for word in tokens if word not in
stopwords.words('english')]
        stemmer = PorterStemmer()
        tokens = [stemmer.stem(word) for word in tokens]
        return " ".join(tokens)

    # Apply cleaning
    data['clean_query'] = data[query_column].apply(clean_text)

    # TF-IDF feature extraction
```

```
    vectorizer = TfidfVectorizer(max_features=300)
    X = vectorizer.fit_transform(data['clean_query']).toarray()

    # Encode labels
    if 'category' in data.columns:
        encoder = LabelEncoder()
        y = encoder.fit_transform(data['category'])
        # Optional: Save encoded labels as well
        pd.DataFrame(y,
columns=['category_encoded']).to_csv("labels_encoded.csv",
index=False)
    else:
        y = None
        print("Warning: 'category' column not found. Labels not
encoded.")

    # Save features
    features_df = pd.DataFrame(X,
columns=vectorizer.get_feature_names_out())
    features_df.to_csv("features_tfidf.csv", index=False)

    print("□ Feature engineering complete. Features saved to
'features_tfidf.csv'")
    if y is not None:
        print("□ Encoded labels saved to 'labels_encoded.csv'")

except FileNotFoundError:
    print(f"Error: File not found at {FILENAME}")
    print("Please ensure the CSV file exists at the specified path.")
except ValueError as ve:
     print(f"Configuration Error: {ve}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Warning: 'category' column not found. Labels not encoded.
□ Feature engineering complete. Features saved to 'features_tfidf.csv'
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# @title model deployment
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
```

```
import re
import os # Import os to check file existence
import pickle # Import pickle to save/load models

# Download stopwords if needed
# Ensure this is run at least once
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    print("Downloading stopwords...")
    nltk.download('stopwords')
except LookupError:
    print("Downloading stopwords...")
    nltk.download('stopwords')


# Load data
FILENAME =
"/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv"

# Define filenames for saving models/vectorizer/encoder
MODEL_FILENAME = 'chatbot_model.pkl'
VECTORIZER_FILENAME = 'tfidf_vectorizer.pkl'
LABEL_ENCODER_FILENAME = 'label_encoder.pkl'


try:
    # --- Check if file exists and has sufficient rows ---
    if not os.path.exists(FILENAME) or os.path.getsize(FILENAME) == 0:
        print(f"Warning: File '{FILENAME}' not found or is empty.")
        # Create dummy data if file is not found or empty
        print("Creating dummy data for demonstration...")
        dummy_data = {
            'user_query': [
                "Hello, I need help with my account.",
                "How do I reset my password?",
                "Tell me about the new features.",
                "What are your operating hours?",
                "Can I speak to a representative?"
            ],
            'category': [ # Include a dummy category column
                "account",
                "security",
                "features",
                "hours",
                "support"
            ]
        }
        data = pd.DataFrame(dummy_data)
```

```
    else:
        # Attempt to load the actual file
        print(f"Loading data from {FILENAME}")
        data = pd.read_csv(FILENAME)
        data = data.fillna("") # Handle potential NaN values

        # Check if 'category' column exists in the loaded data
        if 'category' not in data.columns:
             print("Warning: 'category' column not found in the loaded
CSV.")
             # If 'category' is missing, and the data is large enough,
we can't train.
             # If the intent was to use THIS file for training, it's
impossible without categories.
             # For now, we'll print a warning and exit or add a dummy
category if needed for other steps.
             # Since the error occurs during fit_transform, we MUST
have categories.
             # Let's add a dummy category for demonstration purposes
if it's missing.
             # In a real scenario, you would need a dataset with
categories.
             if len(data) > 0:
                 data["category"] = "general" # Add a default category
                 print("Added a 'general' category column for
demonstration.")
             else:
                 print("File is not empty but contains no data rows,
cannot train model.")
                 # Exit or handle appropriately if the file is just
headers
                 exit() # Or raise an error, or skip the rest of the
code

        # Check if data has enough rows after loading
        if len(data) < 2:
            print(f"Error: File '{FILENAME}' contains only {len(data)}
row(s) after loading.")
            print("Need at least 2 rows to split data and train
model.")
            exit() # Exit if not enough data


    # --- Text Cleaning ---
    def clean_text(text):
        # Ensure text is a string before processing
        if not isinstance(text, str):
            return "" # Return empty string or handle non-string
entries as needed
        text = str(text).lower() # Ensure it's a string before lower()
```

```
        text = re.sub(r'[^a-z\s]', '', text)
        tokens = text.split()
        # Use try-except for stopwords in case download failed for
some reason
        try:
            stop_words = set(stopwords.words('english'))
            tokens = [word for word in tokens if word not in
stop_words]
        except LookupError:
            print("Warning: Stopwords not found. Skipping stopword
removal.")
            pass # Continue without stopword removal if not found

        stemmer = PorterStemmer()
        tokens = [stemmer.stem(word) for word in tokens]
        return " ".join(tokens)

    # Apply cleaning to the column intended for queries
    # Based on previous cells, 'user_query' seems to be the intended
column.
    query_column = None
    if 'user_query' in data.columns:
         query_column = 'user_query'
    elif 'query' in data.columns:
         query_column = 'query'
         print("Warning: 'user_query' column not found, using 'query'
column instead.")
    else:
         raise ValueError("Neither 'user_query' nor 'query' column
found in the DataFrame.")

    data["clean_query"] = data[query_column].apply(clean_text)


    # --- TF-IDF Vectorization ---
    # Only vectorize if there is clean data
    if not data['clean_query'].empty and len(data) >= 1:
        vectorizer = TfidfVectorizer(max_features=300)
        X = vectorizer.fit_transform(data["clean_query"]).toarray()
    else:
        print("No clean query data found for vectorization.")
        X = None # Set X to None if vectorization cannot happen

    # --- Label Encoding ---
    # Only encode labels if 'category' column exists, has >1 unique
values, and sufficient data
    label_encoder = None # Initialize label_encoder outside the if
block
    y = None # Initialize y outside the if block
    if 'category' in data.columns and len(data['category'].unique()) >
```

```
1 and len(data) >=2 and X is not None :
        label_encoder = LabelEncoder()
        y = label_encoder.fit_transform(data["category"])

        # --- Split Data ---
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

        # --- Train Naive Bayes ---
        model = MultinomialNB()
        model.fit(X_train, y_train)

        # --- Predict & Evaluate ---
        y_pred = model.predict(X_test)

        print("□ Model trained successfully.\n")
        print("□ Accuracy:", accuracy_score(y_test, y_pred))
        print("\n□ Classification Report:\n",
classification_report(y_test, y_pred,
target_names=label_encoder.classes_))

        # --- Save the trained model, vectorizer, and label encoder
---
        try:
            with open(MODEL_FILENAME, 'wb') as model_file:
                pickle.dump(model, model_file)
            print(f"□ Model saved to {MODEL_FILENAME}")

            with open(VECTORIZER_FILENAME, 'wb') as vectorizer_file:
                pickle.dump(vectorizer, vectorizer_file)
            print(f"□ Vectorizer saved to {VECTORIZER_FILENAME}")

            with open(LABEL_ENCODER_FILENAME, 'wb') as
label_encoder_file:
                pickle.dump(label_encoder, label_encoder_file)
            print(f"□ Label Encoder saved to
{LABEL_ENCODER_FILENAME}")

        except Exception as save_error:
            print(f"Error saving model artifacts: {save_error}")


    else:
        print("Skipping model training and evaluation:")
        if 'category' not in data.columns:
            print("- 'category' column is missing.")
        elif len(data['category'].unique()) <= 1:
             print("- Not enough unique categories found (need at
least 2).")
        elif len(data) < 2:
```

```
            print("- Not enough data rows to split (need at least
2).")
        elif X is None:
            print("- No feature data (X) available for training.")

        print("Cannot train a classification model without categories,
sufficient data, and feature data.")


except FileNotFoundError:
    # This block is technically covered by the initial os.path.exists
check now,
    # but kept for clarity or other potential FileNotFoundError
issues.
    print(f"Error: File not found at {FILENAME}")
    print("Please ensure the CSV file exists at the specified path.")
except ValueError as ve:
    print(f"Configuration Error: {ve}")
except Exception as e:
    print(f"An unexpected error occurred during data processing or
training: {e}")
```

```
Loading data from
/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_responses
.csv
Warning: 'category' column not found in the loaded CSV.
Added a 'general' category column for demonstration.
Error: File
'/content/Bitext_Sample_Customer_Support_Training_Dataset_27K_response
s.csv' contains only 1 row(s) after loading.
Need at least 2 rows to split data and train model.
Skipping model training and evaluation:
- Not enough unique categories found (need at least 2).
Cannot train a classification model without categories, sufficient
data, and feature data.
```