Code:-

```
import pandas as pd
from tkinter import Tk
from tkinter.filedialog import askopenfilename
from pyxdameraulevenshtein import damerau_levenshtein_distance  # For
Damerau-Levenshtein distance
from transformers import BertTokenizer, BertForSequenceClassification
import torch

# Load pre-trained BERT model and tokenizer for intent classification
model_name = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=3)  #
Assume 3 intent categories
model.eval()

# Global variable to store the filtered dataset
filtered_data = None

# Function to upload the dataset
def upload_file():
   Tk().withdraw()
   file_path = askopenfilename(title="Select CSV/Excel File",
                               filetypes=[("CSV Files", "*.csv"), ("Excel Files",
"*.xlsx")])
   if file_path:
       if file_path.endswith('.csv'):
           data = pd.read_csv(file_path)
       elif file_path.endswith('.xlsx'):
           data = pd.read_excel(file_path)
       return data
   else:
       return None

# Function to filter the dataset based on 'Project', 'DISCIPLINE', and 'DASHBOARD
NAME'
def filter_data(data):
   project = input("Enter Project Name: ").strip()
   discipline = input("Enter Discipline: ").strip()
   dashboard_name = input("Enter Dashboard Name: ").strip()

   # Apply the filters to the data
   filtered = data[
       (data['Project'] == project) &
       (data['DISCIPLINE'] == discipline) &
```

```
        (data['DASHBOARD NAME'] == dashboard_name)
    ]

    if filtered.empty:
        print("No records found with the specified filters.")
        return None
    return filtered

# Function to find the best match using Damerau-Levenshtein distance
def damerau_levenshtein_match(query, choices):
    best_match = None
    best_distance = float('inf')

    for choice in choices:
        distance = damerau_levenshtein_distance(query, choice)  # Calculate
Damerau-Levenshtein distance
        if distance < best_distance:
            best_distance = distance
            best_match = choice

    return best_match, best_distance

# Function to classify user intent using BERT
def classify_intent(query):
    inputs = tokenizer(query, return_tensors='pt', padding=True, truncation=True,
max_length=128)
    outputs = model(inputs)
    logits = outputs.logits
    predicted_label = torch.argmax(logits, dim=1).item()

    # Assume the following labels for the intent classification
    intent_labels = {0: "information_request", 1: "comparison", 2: "other"}

    return intent_labels[predicted_label]

# Function to match the user's query with 'daxformula_Processed' and return
'daxformula'
def match_query(filtered_data):
    query = input("Enter your query to match with daxformula_Processed: ").strip()

    # Classify the user's query intent using BERT
    intent = classify_intent(query)
    print(f"Classified user query intent: {intent}")

    # Get all the values from 'daxformula_Processed' column
```

```
    processed_formulas = filtered_data['daxformula_Processed'].dropna().tolist()

    # Use Damerau-Levenshtein distance to find the closest match
    best_match, distance = damerau_levenshtein_match(query, processed_formulas)

    if best_match is not None:
        # Get the corresponding row and 'daxformula' for the best match
        matched_row = filtered_data[filtered_data['daxformula_Processed'] ==
best_match]
        print(f"\nBest Match: {best_match} (Distance: {distance})")
        print(f"Corresponding daxformula: {matched_row.iloc[0]['daxformula']}")
    else:
        print("\nNo match found for the query.")

# Chatbot interaction flow
def chatbot_session():
    print("Welcome to the Machine Learning Chatbot!")
    file = upload_file()

    if file is not None:
        print("File uploaded successfully.")

        # Step 1: Filter the dataset
        global filtered_data
        filtered_data = filter_data(file)

        if filtered_data is not None:
            # Step 2: Prompt user for a query and search in 'daxformula_Processed'
            while True:
                user_input = input("\nYou can ask 'Make a query', 'Change filters',
or 'Quit': ").lower()

                if user_input == 'make a query':
                    match_query(filtered_data)

                elif user_input == 'change filters':
                    filtered_data = filter_data(file)

                elif user_input == 'quit':
                    print("Exiting the chatbot. Goodbye!")
                    break

                else:
                    print("I didn't understand that. Please try again.")
    else:
```

```
        print("No file uploaded. Please try again.")
chatbot_session()
```

Documentation: Machine Learning-Based Chatbot Using BERT for Intent Classification and Damerau-Levenshtein Matching

1. **Objective and Overview**
   - The chatbot is designed to assist users by processing and understanding natural language queries. Using machine learning and text-matching techniques, the chatbot responds accurately based on information in a user-uploaded dataset. The primary components are:
     - Intent Classification: Using a pre-trained BERT model to determine the purpose of a user's query.
     - Query Matching: Employing Damerau-Levenshtein distance to find the best matches for user queries in a specified column.
     - Dynamic Interaction: Allowing users to filter data by specific fields and re-submit queries.

2. **Core Components and Functionalities**

   - File Upload and Dataset Selection
     - File Upload (`upload_file`): This function enables users to upload a CSV or Excel file, which acts as the dataset from which the chatbot will retrieve information. The chatbot accepts data in either CSV or Excel format, reading it using `pandas`.
     - Dataset Storage: The uploaded dataset is stored globally for easy access, allowing users to filter it or re-query it dynamically throughout the session.

   - Dataset Filtering Based on User Input
     - Filter Data (`filter_data`): Users can refine the dataset by specifying values for the `Project`, `DISCIPLINE`, and `DASHBOARD NAME` fields. This filtering ensures that the chatbot focuses only on relevant subsets of data, which improves query accuracy and reduces unnecessary processing.
     - Filter Validation: After applying the filters, the chatbot checks if any records match the user's criteria. If no matches are found, the user is informed, allowing them to modify their input.

   - Intent Classification with BERT
     - BERT Model Loading: The chatbot loads a pre-trained BERT model (`bert-base-uncased`) for text classification. The model has been fine-tuned with three labels to represent different user intents:
       - information_request: Queries for specific information.
       - comparison: Queries for comparative analysis.
       - other: Other general or undefined queries.
     - Intent Classification (`classify_intent`): This function uses BERT to classify the intent of the user query. The query is tokenized and passed through the BERT model to identify the intent, which determines the chatbot's response style.
     - Classifier Output: The `classify_intent` function returns a label corresponding to the query intent, enhancing the chatbot's ability to understand the user's request.

- Query Matching Using Damerau-Levenshtein Distance
  - Damerau-Levenshtein Matching (`damerau_levenshtein_match`): To find the closest match for a user's query, this function calculates the Damerau-Levenshtein distance between the query and each entry in the `daxformula_Processed` column. This technique accommodates minor typos and spelling errors, allowing the chatbot to handle queries that may not exactly match the dataset.
  - Match Retrieval and Display: Once a match is found, the chatbot retrieves and displays both the matched value and its associated `daxformula`, providing the user with direct and relevant information.

3. **Workflow and Interaction Flow**

- Step 1: File Upload and Dataset Initialization
  - The chatbot begins by prompting the user to upload a dataset file (CSV or Excel). Once uploaded, the data is stored and used as the basis for filtering and query matching.

- Step 2: Dataset Filtering
  - The chatbot prompts the user to provide filter criteria for `Project`, `DISCIPLINE`, and `DASHBOARD NAME`. These filters allow the user to narrow down the dataset to records that are most relevant to their query.

- Step 3: User Query and Intent Classification
  - The chatbot asks the user to enter a query. It then classifies the intent using the BERT model, enabling the chatbot to understand whether the user is requesting specific information, a comparison, or making a general query. This step ensures that the chatbot can tailor its response based on the user's intent.

- Step 4: Query Matching Using Damerau-Levenshtein Distance
  - The chatbot searches for the closest match to the query within the `daxformula_Processed` column. If a match is found, it displays the corresponding `daxformula`. If no match is found, the user is informed, prompting them to modify the query or adjust the filters.

- Step 5: User Interaction Options
  - After a query is processed, the chatbot offers options to the user: ask another query, change the filters, or quit the session. This looped interaction allows for dynamic, ongoing conversation and improved usability.

4. **Key Features and Advantages**

- Robust Matching with Damerau-Levenshtein Distance: This distance-based matching provides resilience against typographical errors, improving match accuracy even when user input is not perfectly formatted.
- Intent Classification Using a Pre-trained BERT Model: BERT enables the chatbot to interpret user intent with high accuracy, guiding responses to match the purpose of the query.

- User-Centric Filtering: By allowing the user to filter data, the chatbot becomes more adaptable to specific needs, ensuring that queries are answered in a highly relevant context.

5. **Limitations and Possible Enhancements**

- Intent Categories: The current model has only three intent categories. Expanding the intent labels and fine-tuning the model on a broader dataset could improve the chatbot's interpretive range.
- Static Filtering Fields: The filtering options are currently limited to three fields. Allowing more flexible or customizable filtering options would enhance user control.
- Query Context and Sequence Tracking: Adding sequence tracking to follow up on user queries could make the chatbot more conversational and contextually aware.