Code:-

```
import pandas as pd
from tkinter import Tk
from tkinter.filedialog import askopenfilename
from pyxdameraulevenshtein import damerau_levenshtein_distance
from transformers import BertTokenizer, BertForSequenceClassification
import torch
import spacy
from sklearn.model_selection import train_test_split

# 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)
model.eval()

# Load SpaCy for entity extraction
nlp = spacy.load("en_core_web_sm")

# Global variables to store the filtered and split datasets
filtered_data = None
train_data = None
test_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 split the dataset into training and testing (80/20 split)
def split_data(data):
   return train_test_split(data, test_size=0.2, random_state=42)

# Function to find the best match using Damerau-Levenshtein distance
```

```python
def damerau_levenshtein_match(query, choices):
   best_match = None
   best_distance = float('inf')

   for choice in choices:
       distance = damerau_levenshtein_distance(query, choice)
       if distance < best_distance:
           best_distance = distance
           best_match = choice

   return best_match, best_distance

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

   project_choices = data['Project'].dropna().unique().tolist()
   discipline_choices = data['DISCIPLINE'].dropna().unique().tolist()
   dashboard_choices = data['DASHBOARD NAME'].dropna().unique().tolist()

   best_project, project_distance = damerau_levenshtein_match(project_input,
project_choices)
   best_discipline, discipline_distance =
damerau_levenshtein_match(discipline_input, discipline_choices)
   best_dashboard, dashboard_distance = damerau_levenshtein_match(dashboard_input,
dashboard_choices)

   print(f"Best match for 'Project': {best_project} (Distance:
{project_distance})")
   print(f"Best match for 'DISCIPLINE': {best_discipline} (Distance:
{discipline_distance})")
   print(f"Best match for 'DASHBOARD NAME': {best_dashboard} (Distance:
{dashboard_distance})")

   filtered = data[
       (data['Project'] == best_project) &
       (data['DISCIPLINE'] == best_discipline) &
       (data['DASHBOARD NAME'] == best_dashboard)
   ]

   if filtered.empty:
       print("No records found with the specified filters.")
```

```python
        return None
    return filtered

# 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()

    intent_labels = {0: "information_request", 1: "comparison", 2: "other"}

    return intent_labels[predicted_label]

# Function to extract entities and keywords from text using spaCy
def extract_entities(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    keywords = [token.lemma_ for token in doc if not token.is_stop and not
token.is_punct]

    return entities, keywords

# 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}")

    # Extract entities and keywords from the user query
    user_entities, user_keywords = extract_entities(query)
    print(f"Extracted Entities from user query: {user_entities}")
    print(f"Extracted Keywords from user query: {user_keywords}")

    # Get all the values from 'daxformula_Processed' column
    processed_formulas = filtered_data['daxformula_Processed'].dropna().tolist()

    # Perform Damerau-Levenshtein distance matching as before
    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]

        # Extract entities and keywords from the matched row
        dataset_entities, dataset_keywords =
extract_entities(matched_row.iloc[0]['daxformula_Processed'])
        print(f"\nBest Match: {best_match} (Distance: {distance})")
        print(f"Corresponding daxformula: {matched_row.iloc[0]['daxformula']}")
        print(f"Extracted Entities from dataset: {dataset_entities}")
        print(f"Extracted Keywords from dataset: {dataset_keywords}")

        # Now perform entity matching
        user_entity_texts = set(ent[0] for ent in user_entities)
        dataset_entity_texts = set(ent[0] for ent in dataset_entities)

        # Match based on common entities
        common_entities = user_entity_texts.intersection(dataset_entity_texts)
        if common_entities:
            print(f"Matched entities between query and dataset: {common_entities}")
        else:
            print("No common entities found between query and dataset.")
    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: Split the dataset into training and testing sets
        global train_data, test_data
        train_data, test_data = split_data(file)
        print(f"Data split into Training (80%) and Testing (20%)", flush=True)

        # Step 2: Filter the dataset for interaction
        global filtered_data
        filtered_data = filter_data(train_data)

        if filtered_data is not None:
            # Step 3: Query matching and interaction
            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)
                    # Provide immediate feedback after processing
                    print("\nQuery processed. You can ask again or change filters.",
flush=True)

                elif user_input == 'change filters':
                    filtered_data = filter_data(train_data)
                    print("\nFilters updated. You can ask a query now.", flush=True)

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

                else:
                    print("I didn't understand that. Please try again.", flush=True)
    else:
        print("No file uploaded. Please try again.", flush=True)

# Start the chatbot session
chatbot_session()
```

Documentation: Initial Chatbot Using BERT for Intent Classification and Damerau-Levenshtein Matching

1. **Objective and Overview**
   - This chatbot assists users by interpreting natural language queries, filtering relevant data, and matching their query against preprocessed information in a dataset. The model combines BERT for classifying user intent, SpaCy for entity and keyword extraction, and Damerau-Levenshtein distance for accurate text matching, making it versatile in responding to user requests for information retrieval.

2. **Core Components and Functionalities**

   - File Upload and Data Preparation
     - File Upload: `upload_file` allows users to select a CSV or Excel file to be used as the chatbot's dataset. The chatbot reads and loads the data, which is subsequently used to generate responses based on user queries.
     - Data Splitting: To facilitate training and testing, the chatbot splits the dataset into training (80%) and testing (20%) sets using `split_data`.

   - Data Filtering Based on User Input

- Data Filtering: The `filter_data` function enables users to filter the dataset based on specific columns like `Project`, `DISCIPLINE`, and `DASHBOARD NAME`. Damerau-Levenshtein distance is used to find the closest match to user input, allowing for slight spelling variations in entries.
- Match Display: The function displays the best match for each column, along with the respective Damerau-Levenshtein distance, ensuring that the chatbot can refine the dataset to only relevant entries.

- Intent Classification with BERT
    - BERT Model for Intent Classification: BERT is used to classify the user's intent into one of three categories: `information_request`, `comparison`, or `other`. The model tokenizes and processes the user query to determine the intent, which helps the chatbot understand the purpose of the query.
    - Classifier Output: The `classify_intent` function returns the intent category, guiding the chatbot to respond appropriately based on the detected user intent.

- Entity Extraction with SpaCy
    - Entity and Keyword Extraction: Using SpaCy, the `extract_entities` function identifies named entities (e.g., organizations, dates, locations) and keywords in the user's query. This feature enables the chatbot to focus on important terms that are relevant to the query.
    - Keyword Filtering: Keywords are extracted using lemmatization and stop word removal, providing a clean list of essential terms for further matching and interaction.

- Query Matching with Damerau-Levenshtein Distance
    - Best Match Calculation: The chatbot identifies the best match for a user query from the `daxformula_Processed` column in the filtered dataset using `damerau_levenshtein_match`. This distance measure allows for slight spelling and typographical errors, improving the chatbot's resilience to user input variation.
    - Detailed Match Information: If a match is found, the chatbot displays detailed information, including the `daxformula` value and matched entities and keywords, enabling the user to validate the response accuracy.

3. **Workflow and Interaction Flow**

- Step 1: Dataset Upload and Splitting
    - The user uploads a dataset file, which is loaded and split into training and testing sets. The chatbot then proceeds to interact with the user based on this dataset.

- Step 2: Dataset Filtering
    - The user filters the dataset by specifying project details, discipline, and dashboard name. The chatbot refines the dataset based on these inputs, allowing for an initial context-based interaction.

- Step 3: Intent Classification and Entity Extraction
    - When a query is submitted, the chatbot classifies the intent using BERT. It also extracts entities and keywords from both the query and the dataset entry, facilitating a more informed response.

- Step 4: Query Matching and Entity Comparison
  - The chatbot attempts to match the query against preprocessed formulae using Damerau-Levenshtein distance. If a match is found, it provides detailed information, including matched entities and keywords. If not, it informs the user that no suitable match was found.

- Step 5: User Interaction Options
  - The chatbot provides options for the user to refine filters, ask another query, or exit the session. This structure makes the interaction flexible and iterative.

4. **Key Benefits and Features**

- Hybrid Approach to Matching: The combination of BERT intent classification, entity extraction, and Damerau-Levenshtein distance provides a balanced model for understanding and accurately responding to user queries.
- Robust Error Handling in Matching: Damerau-Levenshtein distance allows for minor variations in spelling or formatting, making the chatbot tolerant to common typographical errors.
- Detailed Entity Comparison: Entity extraction from both the query and the dataset ensures that the chatbot can confirm the accuracy of matches by identifying common terms and entities.

5. **Limitations and Future Improvements**

- Limited Intent Categories: The current setup only includes three intent categories. Expanding this to include more refined intents could enhance response accuracy.
- Static Filters for Initial Dataset: Currently, filters are limited to specific columns (Project, DISCIPLINE, DASHBOARD NAME). Expanding to a dynamic filtering system would provide a more versatile user experience.
- Additional Contextual Understanding: Integrating additional contextual NLP features, like sentiment analysis or context-based entity prioritization, could improve the chatbot's ability to understand complex user queries.