RAG FOR WELLBEING INSIGHTS

CREATING DATASET

```
!pip install google-generativeai
```

```
Requirement already satisfied: google-generativeai in /usr/local/lib/python3.11/dist-packages (0.8.4)
Requirement already satisfied: google-ai-generativelanguage==0.6.15 in /usr/local/lib/python3.11/dist-packages (from google-generati
Requirement already satisfied: google-api-core in /usr/local/lib/python3.11/dist-packages (from google-generativeai) (2.24.2)
Requirement already satisfied: google-api-python-client in /usr/local/lib/python3.11/dist-packages (from google-generativeai) (2.164
Requirement already satisfied: google-auth>=2.15.0 in /usr/local/lib/python3.11/dist-packages (from google-generativeai) (2.38.0)
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from google-generativeai) (5.29.4)
Requirement already satisfied: pydantic in /usr/local/lib/python3.11/dist-packages (from google-generativeai) (2.11.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from google-generativeai) (4.67.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from google-generativeai) (4.13.1)
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.3 in /usr/local/lib/python3.11/dist-packages (from google-ai-generativelar
Requirement already satisfied: googleapis-common-protos<2.0.0,>=1.56.2 in /usr/local/lib/python3.11/dist-packages (from google-api-c
Requirement already satisfied: requests<3.0.0,>=2.18.0 in /usr/local/lib/python3.11/dist-packages (from google-api-core->google-gene
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from google-auth>=2.15.0->google-g
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from google-auth>=2.15.0->google-ge
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.11/dist-packages (from google-auth>=2.15.0->google-generative
Requirement already satisfied: httplib2<1.dev0,>=0.19.0 in /usr/local/lib/python3.11/dist-packages (from google-api-python-client->g
Requirement already satisfied: google-auth-httplib2<1.0.0,>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from google-api-python
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.11/dist-packages (from google-api-python-client->goog
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic->google-generativeai
Requirement already satisfied: pydantic-core==2.33.1 in /usr/local/lib/python3.11/dist-packages (from pydantic->google-generativeai
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic->google-generative
Requirement already satisfied: grpcio<2.0dev,>=1.33.2 in /usr/local/lib/python3.11/dist-packages (from google-api-core[grpc]!=2.0.*,
Requirement already satisfied: grpcio-status<2.0.dev0,>=1.33.2 in /usr/local/lib/python3.11/dist-packages (from google-api-core[grpc
Requirement already satisfied: pyparsing!=3.0.0,!=3.0.1,!=3.0.2,!=3.0.3,<4,>=2.4.2 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1->google-a
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.18.0->go
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.18.0->google-api-cor
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.18.0->google-a
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.18.0->google-a
```

```
import os
import time
import json
import pandas as pd
import google.generativeai as genai

# Configure Gemini API
genai.configure(api_key="AIzaSyAHX6Zl-x5iNQQnGnWtjLxYJ6VTdkq0Zfo")
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 40,
    "max_output_tokens": 8192,
    "response_mime_type": "text/plain",
}
gemini_model = genai.GenerativeModel(model_name="gemini-2.0-flash", generation_config=generation_config)

# Paths
csv_path = "final_anxiety.csv"
json_path = "anxiety.json"

# Check if JSON file exists and load existing data
if os.path.exists(json_path):
    with open(json_path, "r") as f:
        dataset = json.load(f)
else:
    dataset = []

# Read CSV
df = pd.read_csv(csv_path)

# Process first 10 records for testing
# df_sample = df.head(10)
df_sample = df[107:400]

# Change this line to process the entire dataset
# df_sample = df

def generate_insight(text, issue):
    # Construct prompt
    prompt = f"Analyze the following mental health issue: {issue}\nText: {text}\nProvide wellbeing insights based on the Ryff Scale

```

```
    # Generate response
    response = gemini_model.generate_content([prompt])
    return response.text.strip()

def create_json_dataset():
    for index, row in df_sample.iterrows():
        text = row.get('text', '')
        issue = row.get('mental_health_issue', '')

        # Skip if entry already exists in JSON
        # if any(entry['text'] == text and entry['mental_health_issue'] == issue for entry in dataset):
        #    print(f"Skipping record {index + 1}: Already present in JSON")
        #    continue

        try:
            # Generate wellbeing insight
            insight = generate_insight(text, issue)

            print("---------------------------------\n")
            print(f"Processed record {index + 1}:")
            print(f"Text: {text}")
            print(f"Mental Issue: {issue}")
            print(f"Wellbeing Insight: {insight}")
            print("---------------------------------\n")

            # Append to dataset
            dataset.append({
                "text": text,
                "mental_issue": issue,
                "wellbeing_insight": insight
            })

            # Save to JSON
            with open(json_path, "w") as f:
                json.dump(dataset, f, indent=4)

            print(f"✅ Record {index + 1} appended to JSON")

        except Exception as e:
            print(f"❌ Error on record {index + 1}: {e}")

        # Wait for 5 seconds to handle rate limits
        print(f"⏳ Waiting for 10 seconds before processing the next record...")
        time.sleep(10)

create_json_dataset()
```

```
---------------------------------

Processed record 108:
Text: Anxiety I feel anxiety whenever I have to take decision in that moment lots of thoughts are running in my head and form and
Mental Issue: anxiety
Wellbeing Insight: This person's anxiety manifests as decision paralysis, physical symptoms, and social withdrawal.

**Ryff Scale Insights:**

*   **Autonomy:** Practice independent decisions, starting small. Trust your judgment.

*   **Environmental Mastery:** Create a calming, organized space. Break down tasks into manageable steps.

*   **Personal Growth:** Explore new hobbies. Challenge negative thoughts.

*   **Positive Relations:** Reconnect with supportive people. Set boundaries.

*   **Purpose in Life:** Identify values & align actions. Volunteer.

*   **Self-Acceptance:** Practice self-compassion. Acknowledge strengths.Therapy can help navigate these issues.
---------------------------------

✅ Record 108 appended to JSON
⏳ Waiting for 10 seconds before processing the next record...
---------------------------------

Processed record 109:
Text: Treatment resistant GAD Hi everyone, I'm 28 and have been dealing with persistent, severe anxiety symptoms since childhood.

I've tried a wide range of treatments, including SSRIs, SNRIs, mood stabilizers, antipsychotics, and other meds, but most either

I’ve spent years undergoing medical tests (MRIs, blood work, specialist visits) to rule out physical causes, but no definitive an

I’m looking into nardil as a next step and am curious if anyone here has had success with it for anxiety. Also open to any advice
Mental Issue: anxiety
Wellbeing Insight: This individual faces significant anxiety challenges, potentially treatment-resistant GAD.

**Ryff Scale Insights:**
```

```
*   **Autonomy:** Explore assertiveness training. Practice making small, independent decisions daily.

*   **Environmental Mastery:** Break down large tasks into smaller, manageable steps. Focus on achievable daily goals.

*   **Personal Growth:** Engage in activities that foster learning, like online courses or skill-based hobbies.

*   **Positive Relations:** Prioritize meaningful connections. Join support groups for anxiety or related conditions.

*   **Purpose in Life:** Identify core values and align daily actions with them. Volunteering can provide a sense of purpose.

*   **Self-Acceptance:** Practice self-compassion. Acknowledge strengths and accept imperfections. Engage in mindfulness to incre
----------------------------------

✅ Record 109 appended to JSON
⏳ Waiting for 10 seconds before processing the next record...
----------------------------------

Processed record 110:
```

COMBINING THE DATASETS

```python
import os
import json
from glob import glob

# Paths
json_folder = "./"  # Folder containing JSON files
output_path = "combined_mental_health_dataset.json"

# Collect all JSON file paths
json_files = glob(os.path.join(json_folder, "*.json"))

# Final dataset list
combined_dataset = []

# Read and combine JSON files
for file in json_files:
    with open(file, "r") as f:
        data = json.load(f)
        if isinstance(data, list):  # Ensure data is a list
            combined_dataset.extend(data)
        else:
            print(f"⚠️ Skipping {file} as it does not contain a valid JSON list.")

# Remove duplicate entries based on text and mental_issue
unique_dataset = []
seen_entries = set()

for entry in combined_dataset:
    identifier = (entry.get("text", ""), entry.get("mental_issue", ""))
    if identifier not in seen_entries:
        seen_entries.add(identifier)
        unique_dataset.append(entry)

# Save combined JSON
with open(output_path, "w") as f:
    json.dump(unique_dataset, f, indent=4)

print(f"✅ Combined JSON file created: {output_path}")
```

```
✅ Combined JSON file created: combined_mental_health_dataset.json
```

Converting it into records with instruction, input output format

```python
import json
import re

def clean_text(text):
    # Keep only English letters, numbers, basic punctuation, and whitespace
    cleaned = re.sub(r"[^a-zA-Z0-9\s.,?!]", "", text)
    cleaned = re.sub(r"\s+", " ", cleaned).strip()  # Normalize whitespace
    return cleaned

# Load the original records
with open('combined_mental_health_dataset.json', 'r') as f:
    records = json.load(f)

instruction_data = []
for idx, record in enumerate(records, start=1):
```

```
    cleaned_text = clean_text(record["text"])

    new_record = {
        "instruction": f"Provide wellbeing insight for the below text with {record['mental_issue']}.",
        "input": cleaned_text,
        "output": record["wellbeing_insight"]
    }
    instruction_data.append(new_record)
    print(f"Converted record #{idx}")

# Save the transformed data
with open('instruction_data.json', 'w') as f:
    json.dump(instruction_data, f, indent=4)
```

```
Converted record #1
Converted record #2
Converted record #3
Converted record #4
Converted record #5
Converted record #6
Converted record #7
Converted record #8
Converted record #9
Converted record #10
Converted record #11
Converted record #12
Converted record #13
Converted record #14
Converted record #15
Converted record #16
Converted record #17
Converted record #18
Converted record #19
Converted record #20
Converted record #21
Converted record #22
Converted record #23
Converted record #24
Converted record #25
Converted record #26
Converted record #27
Converted record #28
Converted record #29
Converted record #30
Converted record #31
Converted record #32
Converted record #33
Converted record #34
Converted record #35
Converted record #36
Converted record #37
Converted record #38
Converted record #39
Converted record #40
Converted record #41
Converted record #42
Converted record #43
Converted record #44
Converted record #45
Converted record #46
Converted record #47
Converted record #48
Converted record #49
Converted record #50
Converted record #51
Converted record #52
Converted record #53
Converted record #54
Converted record #55
Converted record #56
Converted record #57
Converted record #58
```

```
import json

# Load the transformed instructions
with open('instruction_data.json', 'r') as f:
    data = json.load(f)

print(len(data))
print("Example entry : ", data[50])
```

```
2381
Example entry :  {'instruction': 'Provide wellbeing insight for the below text with anxiety.', 'input': 'Anyone here have highfuncti
```

PACKAGES

```
!pip install faiss-cpu
```

```
Collecting faiss-cpu
  Downloading faiss_cpu-1.11.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (4.8 kB)
Requirement already satisfied: numpy<3.0,>=1.25.0 in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (2.0.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (24.2)
Downloading faiss_cpu-1.11.0-cp311-cp311-manylinux_2_28_x86_64.whl (31.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 31.3/31.3 MB 62.5 MB/s eta 0:00:00
Installing collected packages: faiss-cpu
Successfully installed faiss-cpu-1.11.0
```

```
!pip show sentence-transformers
```

```
Name: sentence-transformers
Version: 3.4.1
Summary: State-of-the-Art Text Embeddings
Home-page: https://www.SBERT.net
Author:
Author-email: Nils Reimers <info@nils-reimers.de>, Tom Aarsen <tom.aarsen@huggingface.co>
License: Apache 2.0
Location: /usr/local/lib/python3.11/dist-packages
Requires: huggingface-hub, Pillow, scikit-learn, scipy, torch, tqdm, transformers
Required-by:
```

```
!pip install faiss-cpu
!pip install huggingface-hub
!pip install Pillow
!pip install scikit-learn
!pip install sentence-transformers --no-deps
```

```
Requirement already satisfied: faiss-cpu in /usr/local/lib/python3.11/dist-packages (1.11.0)
Requirement already satisfied: numpy<3.0,>=1.25.0 in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (2.0.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (24.2)
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.11/dist-packages (0.30.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (2025.3.2)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (4.67.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (4.13.2
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (2.4.(
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (2025
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (11.2.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: sentence-transformers in /usr/local/lib/python3.11/dist-packages (3.4.1)
```

IMPORT PACKAGES

```
import json
import re
import numpy as np
import faiss
from sentence_transformers import SentenceTransformer
from transformers import pipeline
```

INSTRUCTION DATA AND EMBEDDING MODEL

```
# Load instruction data
with open("instruction_data.json", "r") as file:
    data = json.load(file)

documents = [f"{item['instruction']} {item['input']}" for item in data]
outputs = [item["output"] for item in data]

# Load embedding model (CPU)
embedding_model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2", device="cpu")

```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100%                                                  349/349 [00:00<00:00, 4.42kB/s]
config_sentence_transformers.json: 100%                             116/116 [00:00<00:00, 2.10kB/s]
README.md: 100%                                                     10.5k/10.5k [00:00<00:00, 293kB/s]
sentence_bert_config.json: 100%                                     53.0/53.0 [00:00<00:00, 1.34kB/s]
config.json: 100%                                                   612/612 [00:00<00:00, 30.8kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better p
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back
model.safetensors: 100%                                             90.9M/90.9M [00:00<00:00, 136MB/s]
tokenizer_config.json: 100%                                         350/350 [00:00<00:00, 29.8kB/s]
vocab.txt: 100%                                                     232k/232k [00:00<00:00, 4.75MB/s]
tokenizer.json: 100%                                                466k/466k [00:00<00:00, 10.8MB/s]
special_tokens_map.json: 100%                                       112/112 [00:00<00:00, 8.34kB/s]
config.json: 100%                                                   190/190 [00:00<00:00, 16.9kB/s]
```

FAISS SEARCHING

```
# Build FAISS index
embeddings = np.array([embedding_model.encode(doc) for doc in documents]).astype("float32")
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(embeddings)
```

```
print(embeddings[0])
```

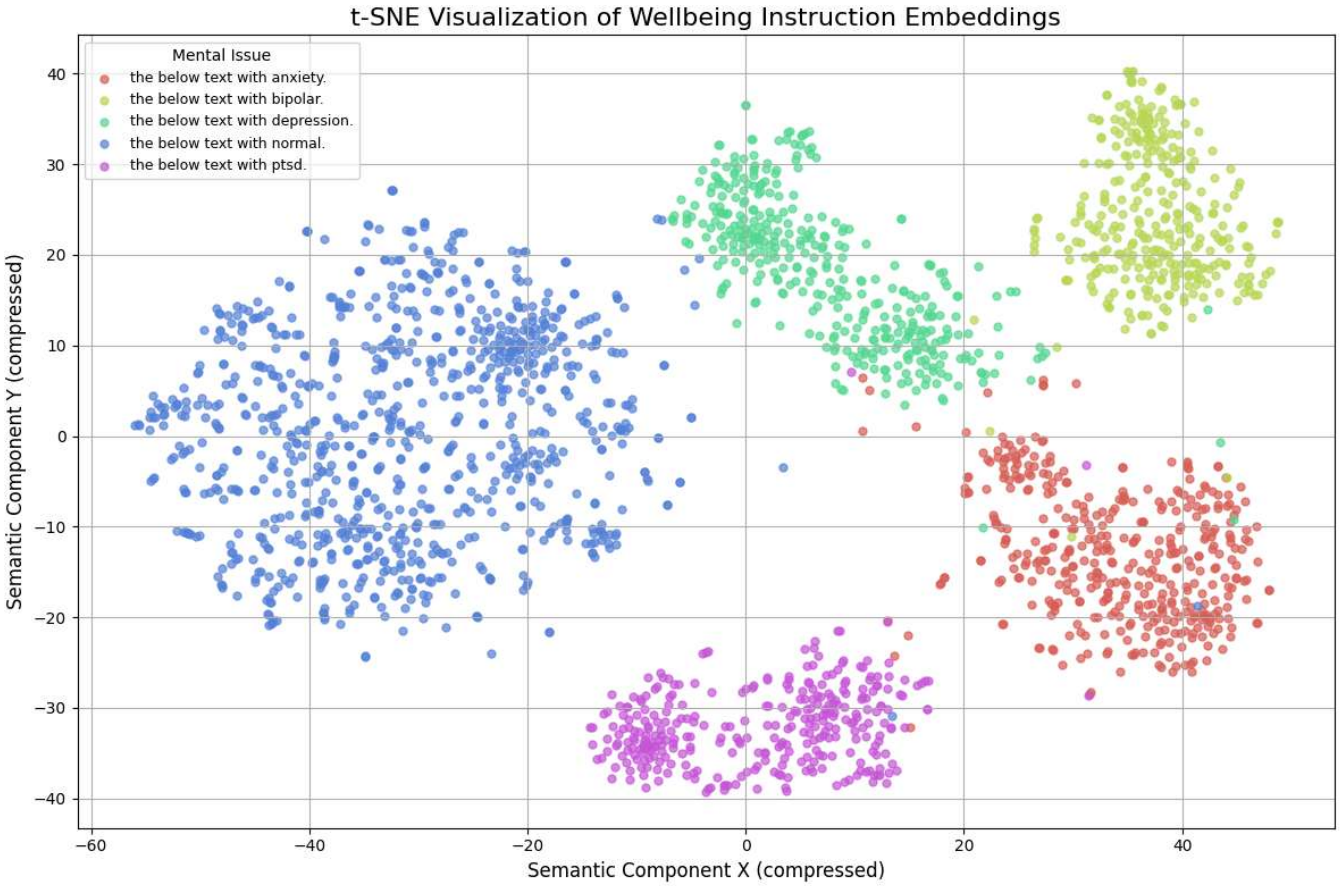
```
 3.82294087e-03 -3.55414972e-02  3.64467464e-02 -4.93446253e-02
 1.54761821e-02  1.59747787e-02 -2.82240342e-02  1.99781340e-02
-1.45216789e-02 -4.36281860e-02 -4.88159284e-02 -2.06167158e-03
 5.38420565e-02  1.14040241e-01  1.47353504e-02 -3.45445350e-02
-7.48330206e-02  9.51653253e-03 -1.73184983e-02  4.36034612e-02
 2.58803014e-02 -5.97075035e-04 -1.05251372e-02 -5.33167832e-02
 1.25680519e-02 -1.43018784e-02 -1.06770717e-01  1.44992638e-02
```

```
 1.54681597e-02 -3.30300210e-03  5.46972174e-03 -8.39293450e-02
 6.66664168e-02  1.18339896e-01 -1.76115893e-02 -4.63223308e-02
 8.41972008e-02  4.74589132e-02 -2.29016133e-02 -3.13805863e-02
-2.80073471e-02 -1.57955079e-03 -9.38227866e-03 -3.34682665e-03
-2.05889400e-02 -4.75831330e-02 -2.56755184e-02  5.37940376e-02
-1.08207902e-02 -8.97366703e-02 -5.61051480e-02 -4.97705899e-02
 1.07073307e-01 -7.65510574e-02 -4.25121188e-02  2.19034813e-02
-1.23145422e-02 -3.07583418e-02 -3.91158722e-02  4.37857322e-02
 7.28315338e-02 -4.28529270e-02  2.60282345e-02 -6.85357004e-02
 7.73008317e-02  2.70340014e-02  1.22347586e-01  1.75248161e-02
 5.60270846e-02 -3.39703821e-02 -1.06468890e-02  8.86213779e-02
-3.81235257e-02  9.27360579e-02 -1.21102929e-02 -3.20660546e-02
 6.92659765e-02  2.38941256e-02 -1.18919782e-01  5.59744425e-02]
```

GRAPHICAL VIEW OF THE EMBEDDINGS AND FAISS INDEX

t-sne

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from collections import defaultdict
import seaborn as sns
import re

# Extract labels (mental issues) from instruction
labels = []
for item in data:
    match = re.search(r'for (.+)', item["instruction"], re.IGNORECASE)
    issue = match.group(1).strip().lower() if match else "unknown"
    labels.append(issue)

# Map labels to colors
unique_labels = sorted(set(labels))
label_to_color = {label: i for i, label in enumerate(unique_labels)}
colors = [label_to_color[label] for label in labels]

# Run t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
reduced = tsne.fit_transform(embeddings)

# Plot using seaborn for better aesthetics
plt.figure(figsize=(12, 8))
palette = sns.color_palette("hls", len(unique_labels))

for i, label in enumerate(unique_labels):
    idxs = [j for j, l in enumerate(labels) if l == label]
    plt.scatter(reduced[idxs, 0], reduced[idxs, 1], s=25, alpha=0.7, label=label, color=palette[i])

plt.title("t-SNE Visualization of Wellbeing Instruction Embeddings", fontsize=16)
plt.xlabel("Semantic Component X (compressed)", fontsize=12)
plt.ylabel("Semantic Component Y (compressed)", fontsize=12)
plt.legend(title="Mental Issue", fontsize=9, title_fontsize=10)
plt.grid(True)
plt.tight_layout()
plt.show()

```

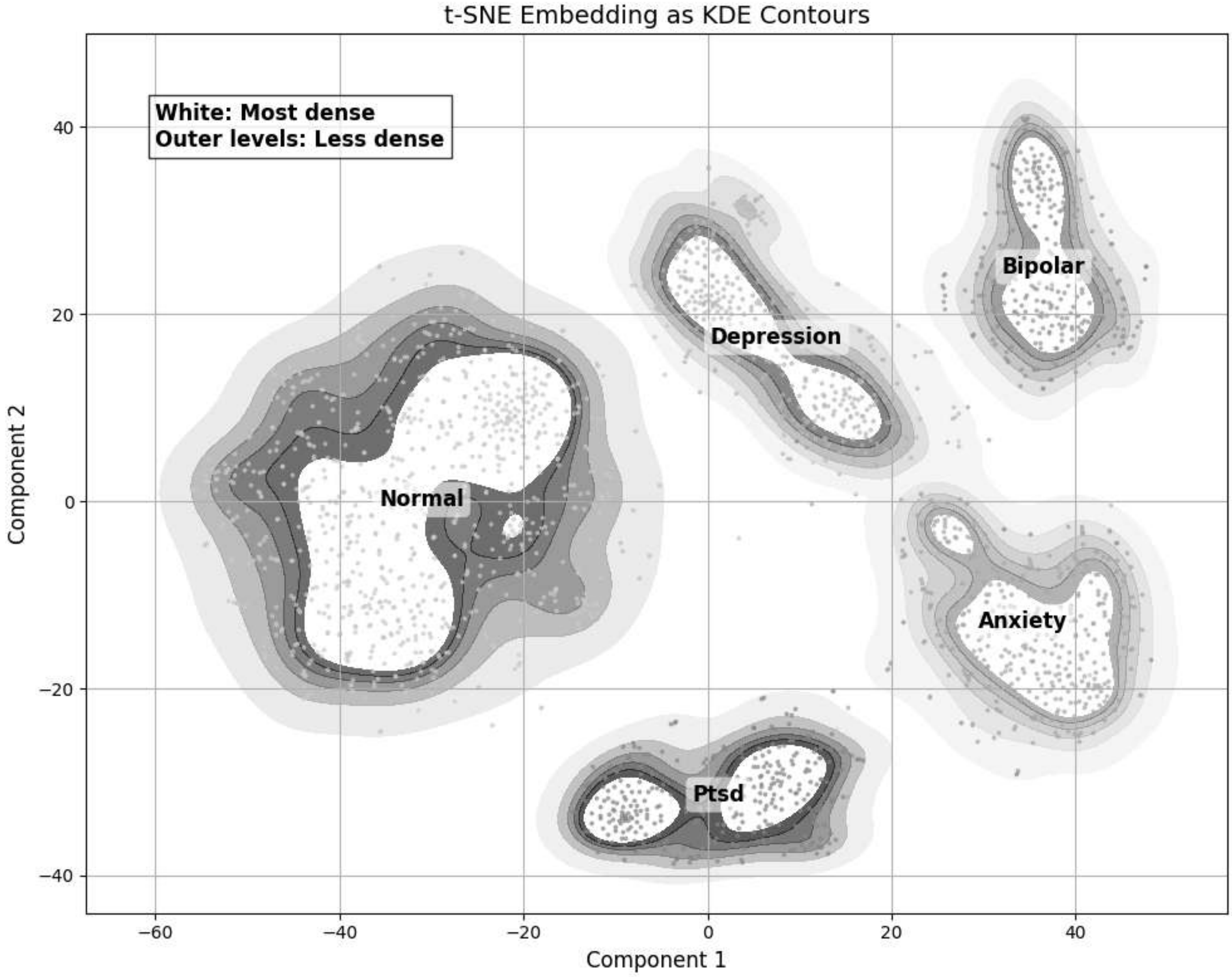


CONTOURS

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import re

# --- 1. Extract labels using keyword matching ---
labels = []
for item in data:
    instruction = item["instruction"].lower()
    if "normal" in instruction:
        labels.append("NORMAL")
    elif "anxiety" in instruction:
        labels.append("ANXIETY")
    elif "depression" in instruction:
        labels.append("DEPRESSION")
    elif "bipolar" in instruction:
        labels.append("BIPOLAR")
    elif "ptsd" in instruction:
        labels.append("PTSD")
    else:
        labels.append("unknown")

# --- 2. Label mapping ---
unique_labels = sorted(set(labels))
label_to_idx = {label: i for i, label in enumerate(unique_labels)}

# --- 3. Define marker and light color for each class ---
label_markers = {
    "NORMAL": ".",
    "DEPRESSION": ".",
    "ANXIETY": ".",
    "BIPOLAR": ".",
    "PTSD": ".",
    "unknown": "."
}

label_colors = {
```

```
    "NORMAL": "#CCCCCC",
    "DEPRESSION": "#BBBBBB",
    "ANXIETY": "#AAAAAA",
    "BIPOLAR": "#999999",
    "PTSD": "#888888",
    "unknown": "#DDDDDD"
}

# --- 4. Run t-SNE (assuming embeddings is defined) ---
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
reduced = tsne.fit_transform(embeddings)

# --- 5. Plot ---
plt.figure(figsize=(10, 8))

for label, idx in label_to_idx.items():
    pts = reduced[[i for i, L in enumerate(labels) if L == label]]

    # KDE with filled contours
    sns.kdeplot(
        x=pts[:, 0], y=pts[:, 1],
        thresh=0.1,
        cmap="Greys",
        fill=True,
        alpha=0.3 + 0.1 * idx,
        linewidths=1.2,
        #levels=[0.05, 0.15, 0.3, 0.45, 0.6]
        levels = np.linspace(0.05, 0.6, 6)  # 6 levels from 0.05 to 0.6

    )

    # Overlay light symbols
    plt.scatter(
        pts[:, 0], pts[:, 1],
        marker=label_markers.get(label, "o"),
        color=label_colors.get(label, "#CCCCCC"),
        s=12,
        alpha=0.6
    )

    # Label at the cluster center
    center_x, center_y = np.mean(pts[:, 0]), np.mean(pts[:, 1])
    plt.text(center_x, center_y, label.capitalize(),
             fontsize=12, weight='bold', ha='center', va='center',
             bbox=dict(facecolor='white', alpha=0.6, edgecolor='none', boxstyle='round,pad=0.3'))


# Add an annotation or text explaining the color scale
plt.text(-60, 40, 'White: Most dense\nOuter levels: Less dense',
         horizontalalignment='left', verticalalignment='center',
         fontsize=12, color='black', weight='bold', bbox=dict(facecolor='white', alpha=0.8))


plt.title("t-SNE Embedding as KDE Contours", fontsize=14)
plt.xlabel("Component 1", fontsize=12)
plt.ylabel("Component 2", fontsize=12)
plt.grid(True)

plt.tight_layout()
plt.show()

```

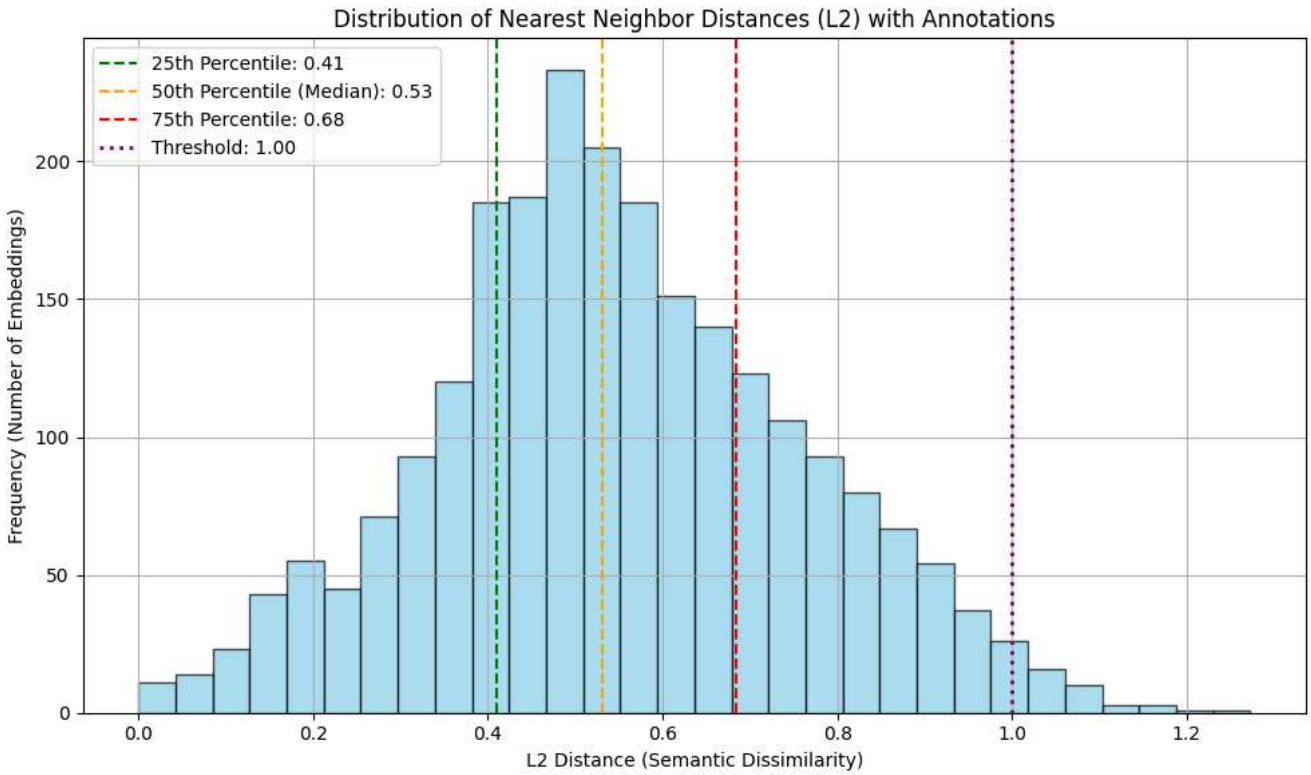
```
/usr/local/lib/python3.11/dist-packages/seaborn/distributions.py:1176: UserWarning: linewidths is ignored by contourf
  cset = contour_func(
/usr/local/lib/python3.11/dist-packages/seaborn/distributions.py:1176: UserWarning: linewidths is ignored by contourf
  cset = contour_func(
/usr/local/lib/python3.11/dist-packages/seaborn/distributions.py:1176: UserWarning: linewidths is ignored by contourf
  cset = contour_func(
/usr/local/lib/python3.11/dist-packages/seaborn/distributions.py:1176: UserWarning: linewidths is ignored by contourf
  cset = contour_func(
/usr/local/lib/python3.11/dist-packages/seaborn/distributions.py:1176: UserWarning: linewidths is ignored by contourf
  cset = contour_func(
```



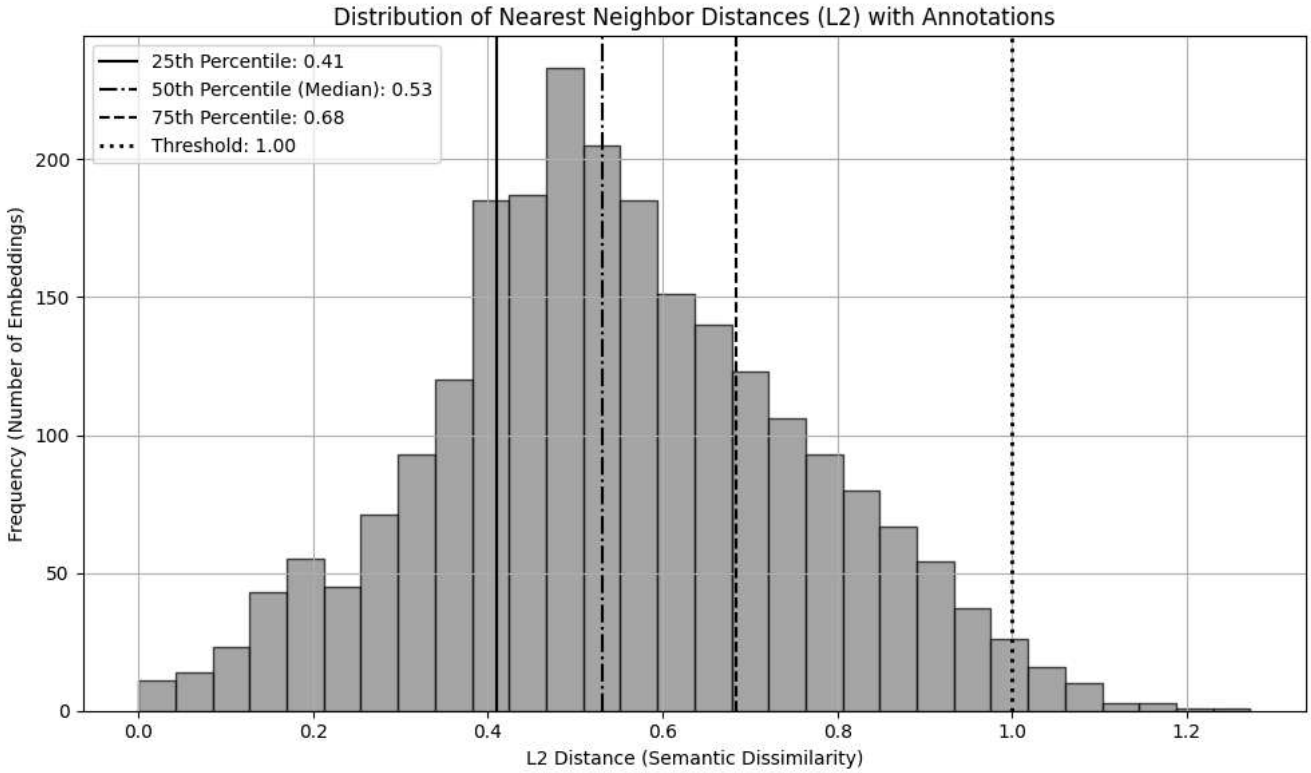
Nearest Neighbour Distance Distribution

```
import numpy as np
import matplotlib.pyplot as plt

# Search for nearest 2 neighbors (self + nearest other)
D, _ = index.search(embeddings, 2)
nearest_distances = D[:, 1]  # skip the distance to self

# Compute histogram
plt.figure(figsize=(10, 6))
counts, bins, patches = plt.hist(nearest_distances, bins=30, color='skyblue', edgecolor='black', alpha=0.7)

# Calculate percentiles
p25 = np.percentile(nearest_distances, 25)
p50 = np.percentile(nearest_distances, 50)
p75 = np.percentile(nearest_distances, 75)

# Plot percentile lines
plt.axvline(p25, color='green', linestyle='..', label=f'25th Percentile: {p25:.2f}')
plt.axvline(p50, color='orange', linestyle='--', label=f'50th Percentile (Median): {p50:.2f}')
plt.axvline(p75, color='red', linestyle='--', label=f'75th Percentile: {p75:.2f}')

# Optional threshold (you can adjust this value based on domain knowledge)
threshold = 1.0
plt.axvline(threshold, color='purple', linestyle=':', linewidth=2, label=f'Threshold: {threshold:.2f}')

# Plot settings
```

```
plt.title("Distribution of Nearest Neighbor Distances (L2) with Annotations")
plt.xlabel("L2 Distance (Semantic Dissimilarity)")
plt.ylabel("Frequency (Number of Embeddings)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

# Search for nearest 2 neighbors (self + nearest other)
D, _ = index.search(embeddings, 2)
nearest_distances = D[:, 1]  # skip the distance to self

# Compute histogram
plt.figure(figsize=(10, 6))
counts, bins, patches = plt.hist(nearest_distances, bins=30, color='grey', edgecolor='black', alpha=0.7)

# Calculate percentiles
p25 = np.percentile(nearest_distances, 25)
p50 = np.percentile(nearest_distances, 50)
p75 = np.percentile(nearest_distances, 75)

# Plot percentile lines
# Changed linestyle from '..' to ':'
plt.axvline(p25, color='black', linestyle='-', label=f'25th Percentile: {p25:.2f}')
plt.axvline(p50, color='black', linestyle='-.', label=f'50th Percentile (Median): {p50:.2f}')
plt.axvline(p75, color='black', linestyle='--', label=f'75th Percentile: {p75:.2f}')

# Optional threshold (you can adjust this value based on domain knowledge)
threshold = 1.0
plt.axvline(threshold, color='black', linestyle=':', linewidth=2, label=f'Threshold: {threshold:.2f}')

# Plot settings
plt.title("Distribution of Nearest Neighbor Distances (L2) with Annotations")
plt.xlabel("L2 Distance (Semantic Dissimilarity)")
plt.ylabel("Frequency (Number of Embeddings)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



LOADING THE SUMMARIZER (no need - works faster without it)

```
import torch

# Load summarization model (facebook/bart-large-cnn)
summarizer = pipeline("summarization", model="facebook/bart-large-cnn", device=0 if torch.cuda.is_available() else -1)

print("✅ Models and FAISS index are ready!")
```

```
config.json: 100%  1.58k/1.58k [00:00<00:00, 72.8kB/s]
model.safetensors: 100%  1.63G/1.63G [00:21<00:00, 120MB/s]
generation_config.json: 100%  363/363 [00:00<00:00, 18.3kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 4.67MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 5.19MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 16.6MB/s]
Device set to use cpu
✅ Models and FAISS index are ready!
```

```
# Define Ryff parameters
ryff_params = [
    "Autonomy", "Environmental Mastery", "Personal Growth",
    "Positive Relations", "Purpose in Life", "Self-Acceptance"
]
```

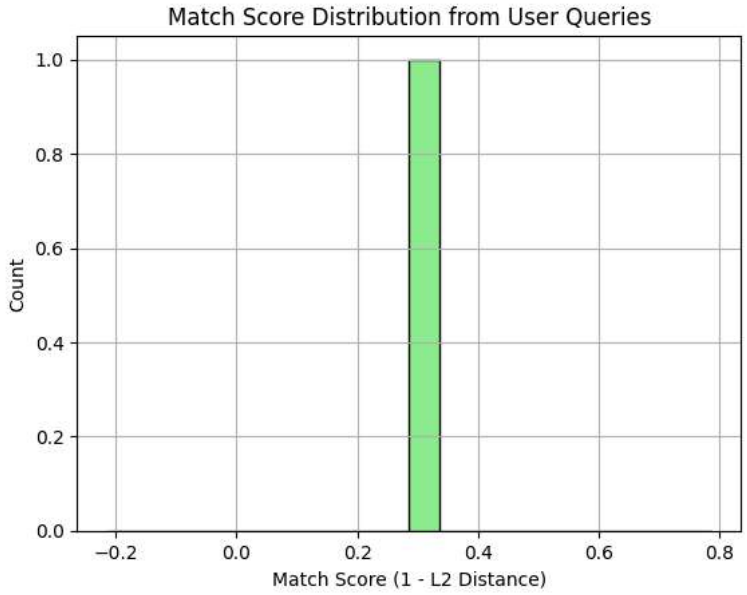
```
# --- User Inputs ---
text_input = input("📝 Enter your situation or thoughts: ").strip()
mental_issue = input("💭 Enter mental issue (e.g., anxiety, depression): ").strip()

ryff_input = input("🎯 (Optional) Enter up to 3 Ryff parameters (comma-separated): ").strip()
selected_ryff = [p.strip() for p in ryff_input.split(",") if p.strip() in ryff_params][:3]

```

```
📝 Enter your situation or thoughts: I am depressed. I dont know what to do. GOD save me
💭 Enter mental issue (e.g., anxiety, depression): anxiety
🎯 (Optional) Enter up to 3 Ryff parameters (comma-separated):
```

```
# --- Step 1: Summarize the user input ---
def safe_summarize(text):
    try:
        summary = summarizer(text, max_length=1024, min_length=2, do_sample=False)[0]["summary_text"]
        return summary
    except Exception:
```

```
        return text[:150] + "..."  # Fallback: trim text

# summarized_text = safe_summarize(text_input)
# No need for summarizer
summarized_text = text_input

# --- Step 2: Retrieve similar wellbeing insight ---
query = f"{summarized_text} {mental_issue}"
query_embedding = np.array([embedding_model.encode(query)]).astype("float32")

D, I = index.search(query_embedding, 1)  # Search the index
retrieved_output = outputs[I[0][0]]

# --- Store the match score ---
match_score = 1 - D[0][0]  # Calculate match score

# Initialize an empty list to store match scores (if needed for multiple queries)
match_scores = []
match_scores.append(match_score) #Append the match score to the list

# --- Step 3: Optional Ryff filtering ---
def filter_output_by_ryff(output_text, selected_ryff):
    if not selected_ryff:
        return output_text
    filtered = []
    for sentence in re.split(r'(?<=[.!?])\s+', output_text):
        for param in selected_ryff:
            if param.lower() in sentence.lower():
                filtered.append(sentence)
                break
    return " ".join(filtered) if filtered else "⚠️ No insights found for selected Ryff parameters."

final_insight = filter_output_by_ryff(retrieved_output, selected_ryff)

# --- Step 4: Show combined result ---
print(f"Text:\n{text_input}")
print("\n💡 Final Wellbeing Output:\n")
print(f"📘 Summary of your situation:\n{summarized_text}")
print("\n🧠 Wellbeing Insight:")
print(final_insight)
print(f"\n🔗 Match Score: {match_score:.4f} (higher is better)")

# --- Plotting the histogram (this is where the error was) ---
plt.hist(match_scores, bins=20, color='lightgreen', edgecolor='black')
plt.title("Match Score Distribution from User Queries")
plt.xlabel("Match Score (1 - L2 Distance)")
plt.ylabel("Count")
plt.grid(True)
plt.show()
```

```
Text:
Hey, I know it’s been weeks since we last talked, but... today was hard. Really hard. I don’t even know why I’m writing this, maybe

💡 Final Wellbeing Output:

📘 Summary of your situation:
Hey, I know it’s been weeks since we last talked, but... today was hard. Really hard. I don’t even know why I’m writing this, maybe

🧠 Wellbeing Insight:
This person likely struggles with PTSD, evidenced by intrusive memories, triggers, intense emotions, and thoughts of self-harm. Here

*   **Autonomy:** Identify personal values and make small, independent choices daily to feel more in control.

*   **Environmental Mastery:** Focus on achievable tasks, breaking larger goals down. Build supportive routines.

*   **Personal Growth:** Explore new hobbies or skills, even in small increments, to foster a sense of development.

*   **Positive Relations:** Connect with understanding individuals. Limit exposure to unsupportive people.

*   **Purpose in Life:** Explore personal values and passions to find what resonates, even if small.

*   **Self-Acceptance:** Practice self-compassion. Acknowledge strengths and limitations without judgment.

🔗 Match Score: 0.2857 (higher is better)
```



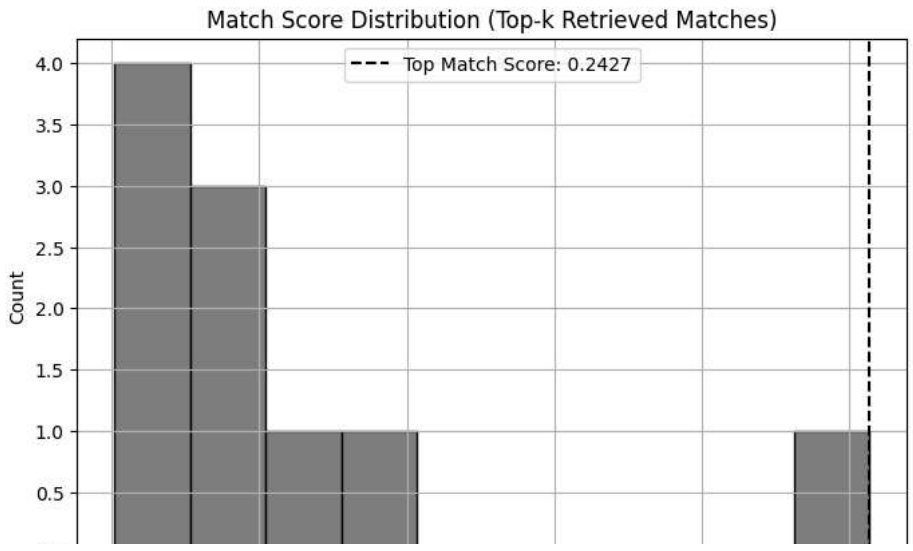
TESTING WITH USER INPUT

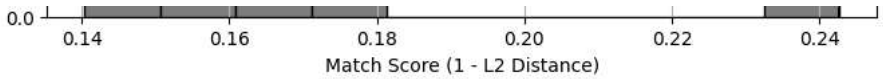
```python
import numpy as np
import re
import matplotlib.pyplot as plt

# --- Step 1: Summarize the user input ---
# No need for summarizer
summarized_text = text_input

# --- Step 2: Retrieve similar wellbeing insights (top-k) ---
query = f"{summarized_text} {mental_issue}"
query_embedding = np.array([embedding_model.encode(query)]).astype("float32")

k = 10  # Number of top matches to retrieve
D, I = index.search(query_embedding, k)  # Search top-k

# --- Step 3: Convert L2 distances to match scores ---
match_scores = 1 - D[0]  # Array of match scores (higher = more similar)

# --- Step 4: Optional Ryff filtering ---
def filter_output_by_ryff(output_text, selected_ryff):
    if not selected_ryff:
        return output_text
    filtered = []
    for sentence in re.split(r'(?<=[.!?])\s+', output_text):
        for param in selected_ryff:
            if param.lower() in sentence.lower():
                filtered.append(sentence)
                break
    return " ".join(filtered) if filtered else "⚠️ No insights found for selected Ryff parameters."
```

```
30
31 # --- Step 5: Display summary and top match ---
32 print(f"Text:\n{text_input}")
33 print("\n💡 Final Wellbeing Output:\n")
34 print(f"📘 Summary of your situation:\n{summarized_text}")
35
36 # --- Show all top-k retrieved outputs and scores ---
37 for i, idx in enumerate(I[0]):
38     insight = outputs[idx]
39     filtered_insight = filter_output_by_ryff(insight, selected_ryff)
40     print(f"\n🔹 Top-{i+1} Match Score: {match_scores[i]:.4f}")
41     print(f"🧠 Wellbeing Insight:\n{filtered_insight}")
42
43 # --- Step 6: Plot histogram of match scores ---
44 plt.figure(figsize=(8, 5))
45 plt.hist(match_scores, bins=10, color='grey', edgecolor='black')
46 plt.axvline(match_scores[0], color='black', linestyle='--', label=f"Top Match Score: {match_scores[0]:.4f}")
47 plt.title("Match Score Distribution (Top-k Retrieved Matches)")
48 plt.xlabel("Match Score (1 - L2 Distance)")
49 plt.ylabel("Count")
50 plt.grid(True)
51 plt.legend()
52 plt.show()
53
```

```
Text:
I am depressed. I dont know what to do. GOD save me

💡 Final Wellbeing Output:

📘 Summary of your situation:
I am depressed. I dont know what to do. GOD save me

🔹 Top-1 Match Score: 0.2427
🧠 Wellbeing Insight:
The text indicates significant depression: hopelessness, anhedonia, social isolation, anxiety, self-loathing, and suicidal ideation

**Ryff Scale Insights:**

*   **Autonomy:** Identify small, achievable decisions you *can* control. Choose what to watch/listen to.
*   **Env. Mastery:** Start with one tiny task: tidy a drawer. Success breeds confidence.
*   **Personal Growth:** Explore free online courses; something new, no pressure to succeed.
*   **Positive Relations:** Reconnect with a distant family member. Small interactions matter.
*   **Purpose in Life:** Volunteer (online if needed) to help others. Shifting focus can help.
*   **Self-Acceptance:** Practice daily affirmations. "I am worthy of kindness" can be powerful.

Prioritize seeking immediate professional help, given the mention of suicidal thoughts.

🔹 Top-2 Match Score: 0.1781
🧠 Wellbeing Insight:
This person describes severe anxiety impacting daily life. Using Ryff's model:

*   **Autonomy:** Practice assertive communication online; small steps to voice opinions.
*   **Env. Mastery:** Create a calming, predictable routine. Break tasks into manageable steps.
*   **Personal Growth:** Explore new hobbies/skills online; celebrate small victories.
*   **Pos. Relations:** Join online anxiety support groups; connect with understanding others.
*   **Purpose in Life:** Identify values/passions. Focus on helping others online.
*   **Self-Acceptance:** Practice self-compassion through journaling or meditation; acknowledge strengths.

Seeking professional help (therapy, medication) is vital.

🔹 Top-3 Match Score: 0.1661
🧠 Wellbeing Insight:
This text describes significant anxiety symptoms: breathlessness, panic ("mini heart attacks"), nausea, physical paralysis, and trem

**Wellbeing Insights (Ryff Scale):**

*   **Autonomy:** Practice small independent choices daily.

*   **Env. Mastery:** Break down tasks into smaller, manageable steps. Seek safe spaces.

*   **Personal Growth:** Journal feelings, learn coping skills (breathing exercises).

*   **Pos. Relations:** Talk to trusted adults (family, school counselor).

*   **Purpose in Life:** Find enjoyable hobbies, even small ones.

*   **Self-Acceptance:** Acknowledge anxiety is real, not a personal failing. Be kind to yourself. Seek professional help.

🔹 Top-4 Match Score: 0.1568
🧠 Wellbeing Insight:
The individual is experiencing a mental health crisis, possibly related to recent "news," leading to depression and feelings of beir

**Ryff Scale Insights:**

*   **Autonomy:** Assert needs in treatment; practice independent decisions within the unit.
*   **Env. Mastery:** Focus on manageable tasks; create a calming space within their environment.
*   **Personal Growth:** Embrace therapy as a learning experience; journal reflections.
*   **Pos. Relations:** Connect with supportive staff/patients; engage in group activities.
*   **Purpose in Life:** Reconnect with values; set small, achievable goals for recovery.
*   **Self-Acceptance:** Practice self-compassion; acknowledge struggles without judgment.

🔹 Top-5 Match Score: 0.1530
🧠 Wellbeing Insight:
The text suggests a severe mental health crisis. **Immediate action is needed:** contact a crisis hotline (988 in the US/Canada) or

**Wellbeing Insights (Ryff Scale):**

*   **Autonomy:** Reclaim control by making small, independent choices (e.g., what to eat, music).
*   **Env. Mastery:** Simplify your surroundings. One manageable task: tidy a drawer.
*   **Personal Growth:** Remind yourself of past resilience. Note one small thing you learned today.
*   **Positive Relations:** Reach out to a trusted friend/family member. Even a brief text helps.
*   **Purpose in Life:** Focus on immediate goals: getting through the next hour, then the next day.
*   **Self-Acceptance:** Acknowledge your pain without judgment. "I'm struggling, and that's okay right now."

🔹 Top-6 Match Score: 0.1514
🧠 Wellbeing Insight:
This text suggests symptoms aligning with anxiety, particularly generalized anxiety disorder (GAD), presenting as "Morning Anxiety.

**Wellbeing Insights (Ryff Scale):**

*   **Autonomy:** Challenge anxious thoughts. Make independent decisions daily.

*   **Environmental Mastery:** Create a calm morning routine (meditation, nature).
```

```
*   **Personal Growth:** Learn coping mechanisms for anxiety (mindfulness, CBT).

*   **Positive Relations:** Connect with loved ones. Share your feelings.

*   **Purpose in Life:** Reflect on values and goals. Find meaning in daily activities.

*   **Self-Acceptance:** Acknowledge and validate your feelings. Practice self-compassion.

🔹 Top-7 Match Score: 0.1467
🧠 Wellbeing Insight:
Okay, here's an analysis of the text regarding anxiety, followed by Ryff Scale-based wellbeing advice:

**Analysis:**

The individual expresses pervasive anxiety, unrelated to specific circumstances. It's a constant state causing distress, compounded

**Wellbeing Insights (Ryff Scale):**

*   **Autonomy:** Challenge anxious thoughts. Make small decisions to feel in control (e.g., meal choices).
*   **Env. Mastery:** Identify manageable tasks and complete them. Organize one small space.
*   **Personal Growth:** Learn coping skills via free online resources (meditation apps, anxiety blogs).
*   **Pos. Relations:** Reach out to trusted friends/family for support. Share feelings.
*   **Purpose in Life:** Find free volunteer options that align with your values.
*   **Self-Acceptance:** Practice self-compassion. Acknowledge your struggles without judgment.

🔹 Top-8 Match Score: 0.1453
🧠 Wellbeing Insight:
The text describes acute anxiety, bordering on panic, with a history of suicidal ideation.

**Ryff Scale Insights:**

*   **Autonomy:** Practice assertive communication; politely decline extra work shifts.
*   **Environmental Mastery:** Plan your work weekend in advance to feel more in control.
*   **Personal Growth:** Acknowledge progress in reducing work and seeking help.
*   **Positive Relations:** Reach out to a trusted friend or family member for support/hug.
*   **Purpose in Life:** Reconnect with values and remind yourself why reduced workload is beneficial.
*   **Self-Acceptance:** Practice self-compassion; accept anxiety as a temporary feeling.

🔹 Top-9 Match Score: 0.1408
🧠 Wellbeing Insight:
Okay, let's analyze the text and provide Ryff-based wellbeing insights.

The text indicates a diagnosis of Generalized Anxiety Disorder (GAD). The individual (f15) expresses mixed feelings: concern about p

**Ryff Scale & Practical Advice:**

*   **Autonomy:** Practice independent decision-making (small choices).
*   **Environmental Mastery:** Set achievable daily goals; organize your space.
*   **Personal Growth:** Learn a new skill; reflect on personal strengths.
*   **Positive Relations:** Spend time with supportive friends/family; express gratitude.
*   **Purpose in Life:** Identify values and small ways to act accordingly.
*   **Self-Acceptance:** Practice self-compassion; challenge negative self-talk.

🔹 Top-10 Match Score: 0.1404
🧠 Wellbeing Insight:
Based on the text, the individual is experiencing severe distress and suicidal ideation, strongly suggesting depression.

**Ryff Wellbeing Insights:**

*   **Autonomy:** Practice independent decision-making, even small choices.
*   **Env. Mastery:** Identify manageable tasks to regain control (e.g., cleaning).
*   **Personal Growth:** Explore new hobbies or skills to reignite interest.
*   **Positive Relations:** Seek professional support and connect with trusted friends.
*   **Purpose in Life:** Define values & goals, focusing on small, achievable steps.
*   **Self-Acceptance:** Practice self-compassion and challenge negative self-talk.

**Immediate Action:** Contact a crisis hotline or mental health professional.
```



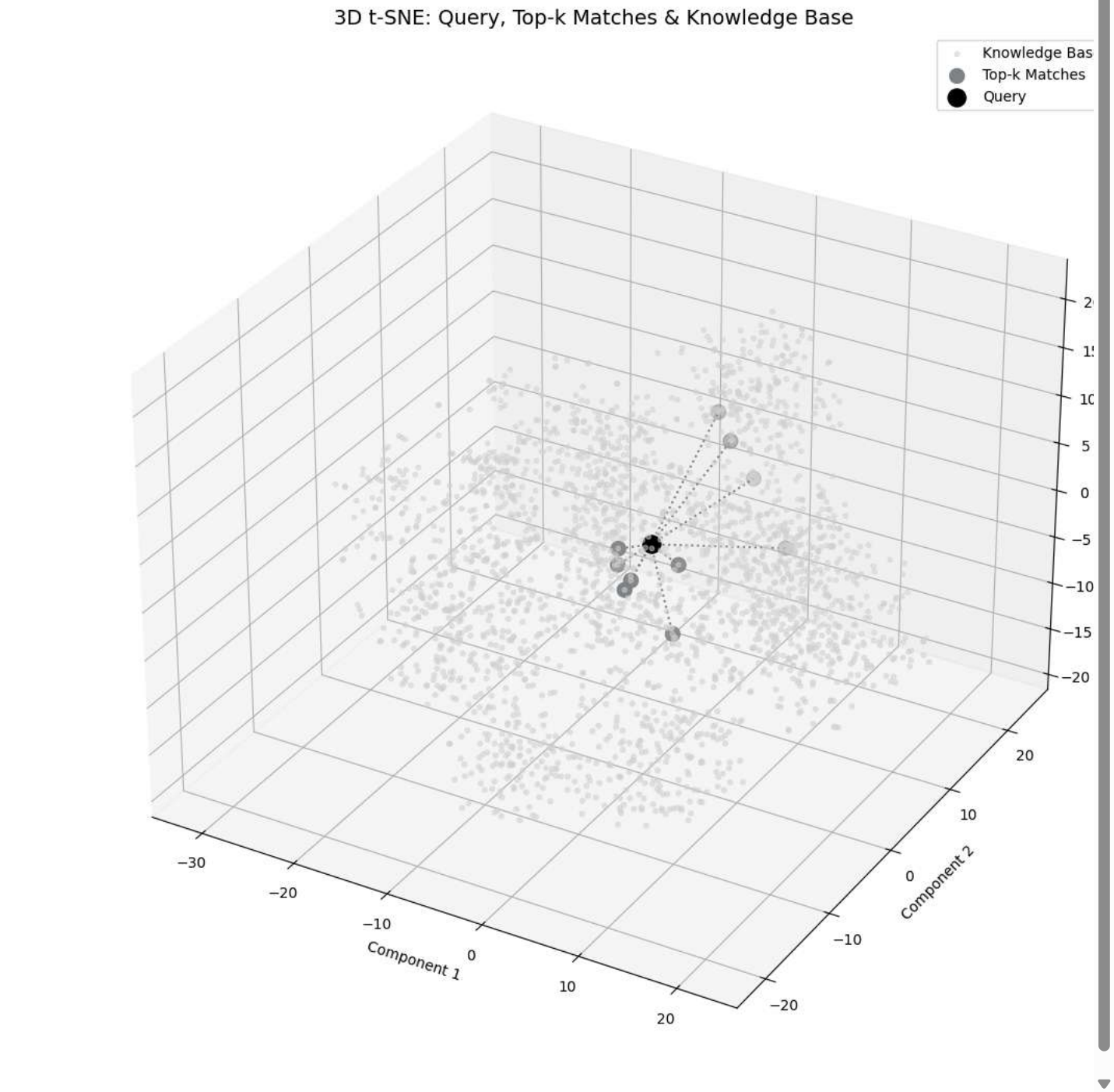

0.0
0.14
0.16
0.18
0.20
0.22
0.24
Match Score (1 - L2 Distance)

#

GRAPH FOR TOP K MATCHES

```
# Get indices of all knowledge base docs
all_indices = list(range(len(embeddings)))

# Extract embeddings
all_embeddings = np.array(embeddings)
query_embedding = query_embedding.reshape(1, -1)  # Ensure correct shape

# Combine query, top-k matches, and all other embeddings
combined_embeddings = np.vstack([query_embedding, all_embeddings])

# Run 3D t-SNE with lower perplexity (since total size is small + 1)
reducer = TSNE(n_components=3, random_state=42, perplexity=30, n_iter=1000)
reduced_all = reducer.fit_transform(combined_embeddings)

# Split the reduced embeddings
query_3d = reduced_all[0]
all_3d = reduced_all[1:]

# Prepare top-k reduced embeddings using their relative positions
# Define topk_indices using the previously defined 'I' variable
topk_indices = I[0]  # Extract indices from Faiss search result 'I'
topk_indices_set = set(topk_indices) # Update to use topk_indices
topk_3d = np.array([all_3d[i] for i in topk_indices])

# Plot
fig = plt.figure(figsize=(13, 10))
ax = fig.add_subplot(111, projection='3d')

# Plot all knowledge base embeddings in background
ax.scatter(all_3d[:, 0], all_3d[:, 1], all_3d[:, 2], c='lightgray', s=10, alpha=0.5, label='Knowledge Base')

# Plot the top-k matches
ax.scatter(topk_3d[:, 0], topk_3d[:, 1], topk_3d[:, 2], c='#818589', s=100, label='Top-k Matches')

# Plot the query
ax.scatter(query_3d[0], query_3d[1], query_3d[2], c='black', s=150, label='Query')

# Draw lines from query to each top-k match
for point in topk_3d:
    ax.plot(
        [query_3d[0], point[0]],
        [query_3d[1], point[1]],
        [query_3d[2], point[2]],
        c='gray',
        linestyle='dotted'
    )

# Labels and legend
ax.set_title("3D t-SNE: Query, Top-k Matches & Knowledge Base", fontsize=14)
ax.set_xlabel("Component 1")
ax.set_ylabel("Component 2")
ax.set_zlabel("Component 3")
ax.legend()
# plt.tight_layout()
plt.subplots_adjust(left=0, right=1, bottom=0, top=1)
plt.show()
```

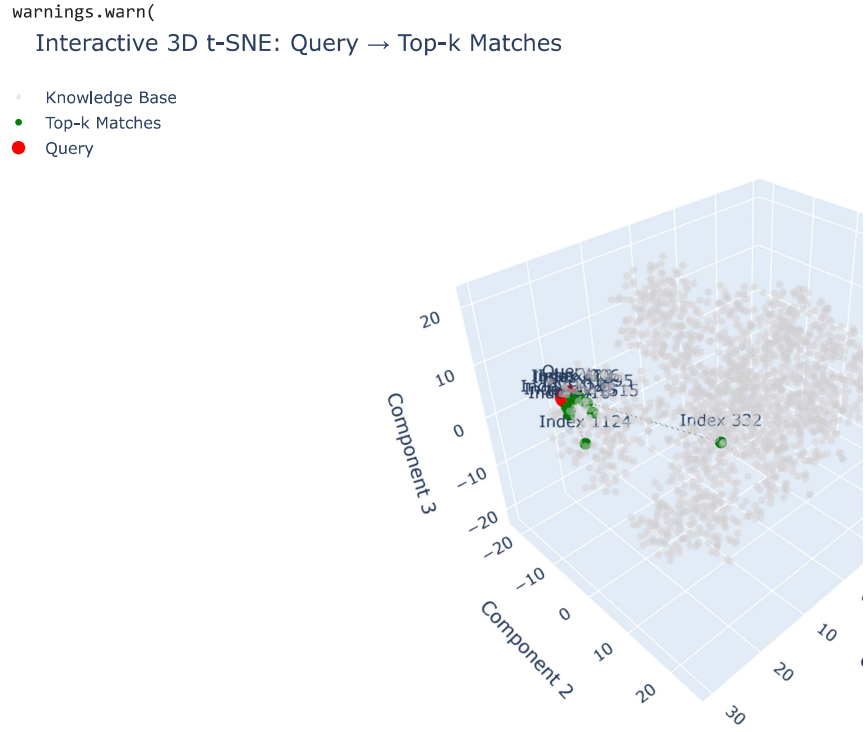
```
/usr/local/lib/python3.11/dist-packages/sklearn/manifold/_t_sne.py:1164: FutureWarning: 'n_iter' was renamed to 'max_iter' in ver
  warnings.warn(
```




```
import plotly.graph_objs as go
from sklearn.manifold import TSNE
import numpy as np

# Combine query and knowledge base embeddings
query_embedding = query_embedding.reshape(1, -1)
combined_embeddings = np.vstack([query_embedding, embeddings])

# Run t-SNE to reduce to 3D
reducer = TSNE(n_components=3, random_state=42, perplexity=30, n_iter=1000)
reduced_all = reducer.fit_transform(combined_embeddings)

# Separate reduced points
query_3d = reduced_all[0]
all_3d = reduced_all[1:]
topk_indices = I[0]
topk_3d = np.array([all_3d[i] for i in topk_indices])

# Prepare trace for knowledge base
kb_trace = go.Scatter3d(
    x=all_3d[:, 0], y=all_3d[:, 1], z=all_3d[:, 2],
    mode='markers',
```

```
    name='Knowledge Base',
    marker=dict(size=3, color='lightgray', opacity=0.5),
    hoverinfo='none'
)

# Prepare trace for top-k matches with annotations
topk_trace = go.Scatter3d(
    x=topk_3d[:, 0], y=topk_3d[:, 1], z=topk_3d[:, 2],
    mode='markers+text',
    name='Top-k Matches',
    marker=dict(size=5, color='green'),
    text=[f"Index {idx}" for idx in topk_indices],
    textposition='top center',
    hoverinfo='text',
    hovertext=[f"Match Index: {idx}" for idx in topk_indices]
)

# Prepare trace for query point with tooltip
query_trace = go.Scatter3d(
    x=[query_3d[0]], y=[query_3d[1]], z=[query_3d[2]],
    mode='markers+text',
    name='Query',
    marker=dict(size=10, color='red'),
    text=["Query"],
    textposition="top center",
    hoverinfo='text',
    hovertext=[f"Query: {query}"] # Changed query_text to query
)

# Prepare dotted lines from query to top-k
lines = []
for point in topk_3d:
    lines.append(go.Scatter3d(
        x=[query_3d[0], point[0]],
        y=[query_3d[1], point[1]],
        z=[query_3d[2], point[2]],
        mode='lines',
        line=dict(color='gray', dash='dot'),
        showlegend=False,
        hoverinfo='none'
    ))

# Combine all traces
fig = go.Figure(data=[kb_trace, topk_trace, query_trace] + lines)

# Layout settings
fig.update_layout(
    title="Interactive 3D t-SNE: Query → Top-k Matches",
    scene=dict(
        xaxis_title='Component 1',
        yaxis_title='Component 2',
        zaxis_title='Component 3'
    ),
    legend=dict(x=0, y=1),
    margin=dict(l=0, r=0, b=0, t=40)
)

# Show the interactive plot
fig.show()

```

```
/usr/local/lib/python3.11/dist-packages/sklearn/manifold/_t_sne.py:1164: FutureWarning: 'n_iter' was renamed to 'max_iter' in versic
  warnings.warn(
```



STREAMLIT APPLICATION FOR NAIVE TEST

RUNTIME GPU

PACKAGES

```
!pip install faiss-cpu
!pip install huggingface-hub
!pip install Pillow
!pip install scikit-learn
!pip install sentence-transformers --no-deps
!pip install Streamlit
!pip install pyngrok
!pip install google-generativeai
```

```
Requirement already satisfied: faiss-cpu in /usr/local/lib/python3.11/dist-packages (1.10.0)
Requirement already satisfied: numpy<3.0,>=1.25.0 in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (2.0.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (24.2)
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.11/dist-packages (0.30.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (2025.3.2)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (4.67.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (4.13
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hu
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (20
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (11.1.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: sentence-transformers in /usr/local/lib/python3.11/dist-packages (3.4.1)
Requirement already satisfied: Streamlit in /usr/local/lib/python3.11/dist-packages (1.44.1)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (5.5.0)
Requirement already satisfied: blinker<2,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (1.9.0)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (5.5.2)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (8.1.8)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (2.0.2)
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (24.2)
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (2.2.2)
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (11.1.0)
Requirement already satisfied: protobuf<6,>=3.20 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (5.29.4)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (18.1.0)
```

```
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (2.32.3)
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (9.1.2)
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.4.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (4.13.1)
Requirement already satisfied: watchdog<7,>=2.1.5 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (6.0.0)
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (3.1.44)
Requirement already satisfied: pydeck<1,>=0.8.0b4 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (0.9.1)
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (6.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->Streamlit) (3.1.6)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->Streamlit) (4.23.
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->Streamlit) (1.33
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-packages (from gitpython!=3.1.19,<4,>=3.0.7->Str
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->Streamli
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->Streamlit) (2025.2
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->Streamlit) (2025
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->Strea
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->Streamlit) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->Streamlit)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->Streamlit)
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.11/dist-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.1
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->altair<6,>=4.0->Streamlit
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->St
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=
```

SAVING THE EMBEDDING MODEL

```python
from sentence_transformers import SentenceTransformer
import pickle

# Load model once
embedding_model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2", device="cuda") # change to cpu when in cpu runtime

# Save it to a pickle file remove gpu when in cpu
with open("rag_embedding_gpu.pkl", "wb") as f:
    pickle.dump(embedding_model, f)

print("✅ Saved embedding_model to pkl")

```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100%                                                  349/349 [00:00<00:00, 13.2kB/s]
config_sentence_transformers.json: 100%                             116/116 [00:00<00:00, 3.44kB/s]
README.md: 100%                                                     10.5k/10.5k [00:00<00:00, 345kB/s]
sentence_bert_config.json: 100%                                     53.0/53.0 [00:00<00:00, 1.42kB/s]
config.json: 100%                                                   612/612 [00:00<00:00, 19.2kB/s]
model.safetensors: 100%                                             90.9M/90.9M [00:00<00:00, 220MB/s]
tokenizer_config.json: 100%                                         350/350 [00:00<00:00, 29.5kB/s]
vocab.txt: 100%                                                     232k/232k [00:00<00:00, 12.1MB/s]
tokenizer.json: 100%                                                466k/466k [00:00<00:00, 24.6MB/s]
special_tokens_map.json: 100%                                       112/112 [00:00<00:00, 12.7kB/s]
config.json: 100%                                                   190/190 [00:00<00:00, 15.5kB/s]
✅ Saved embedding_model to pkl
```

Store the documents, output and faiss index

```python
import json
import numpy as np
import faiss
from sentence_transformers import SentenceTransformer
import pickle

# Load embedding model from pickle
with open("rag_embedding_gpu.pkl", "rb") as f:
    embedding_model = pickle.load(f)
    print("✅ Loaded embedding_model from pkl")

```

```
# Load updated instruction data
with open("instruction_data.json", "r") as file:
    data = json.load(file)

documents = [f"{item['instruction']} {item['input']}" for item in data]
outputs = [item["output"] for item in data]
print(f"📦 Total Records: {len(documents)}")

# Build FAISS index
embeddings = np.array([embedding_model.encode(doc) for doc in documents]).astype("float32")
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(embeddings)
print(f"📦 FAISS Index rebuilt with {index.ntotal} vectors")

# Save to .pkl
with open("global_store_gpu.pkl", "wb") as f:
    pickle.dump({"documents": documents, "outputs": outputs, "index": index, "embeddings": embeddings}, f)
print("💾 Saved updated global_store_gpu.pkl")
```

```
✅ Loaded embedding_model from pkl
📦 Total Records: 2381
📦 FAISS Index rebuilt with 2381 vectors
💾 Saved updated global_store_gpu.pkl
```

App code

```
%%writefile app.py

import streamlit as st
import json
import re
import pickle
import numpy as np
import faiss
from sentence_transformers import SentenceTransformer
from transformers import pipeline
import google.generativeai as genai

import plotly.graph_objs as go
from sklearn.manifold import TSNE
import plotly.offline as pyo
import os

# Configure Gemini API for wellbeing insights
genai.configure(api_key="AIzaSyCaN_5ie8UhhnCg-T1iNhSKs20D9QQpYzw")
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 40,
    "max_output_tokens": 8192,
    "response_mime_type": "text/plain",
}
gemini_model = genai.GenerativeModel(
    model_name="gemini-2.0-flash",
    generation_config=generation_config,
)

# Load embedding model from pickle
with open("rag_embedding_gpu.pkl", "rb") as f:
    embedding_model = pickle.load(f)
    st.success("✅ Loaded embedding_model from pkl")

# Load global store at the start of the app
with open("global_store_gpu.pkl", "rb") as f:
    store = pickle.load(f)

documents = store["documents"]
outputs = store["outputs"]
index = store["index"]
embeddings = store["embeddings"]

st.success(f"🔁 Loaded {len(documents)} documents from global_store.pkl")

# Append new input and response to instruction_data.json
def append_to_json_file(instruction, input_text, output, filename="instruction_data.json"):
    progress = st.progress(0, text="Updating...")

    progress.progress(10, text="📄 Appending new record to JSON...")
    new_record = {
        "instruction": instruction,
```

```
        "input": input_text,
        "output": output
    }

    with open(filename, "r+") as file:
        content = json.load(file)
        content.append(new_record)
        file.seek(0)
        json.dump(content, file, indent=4)

    st.success(f"New record appended to {filename}")

    progress.progress(30, text="📥 Loading updated instruction_data.json...")
    # Load instruction data
    with open("instruction_data.json", "r") as file:
        data = json.load(file)

    # Rebuild documents and outputs
    documents = [f"{item['instruction']} {item['input']}" for item in data]
    outputs = [item["output"] for item in data]
    st.info(f"📄 Total Records in instruction_data.json: {len(documents)}")

    # Build FAISS index
    progress.progress(50, text="🔍 Encoding documents with embedding model...")
    embeddings = np.array([embedding_model.encode(doc) for doc in documents]).astype("float32")
    progress.progress(70, text="📦 Building FAISS Index...")
    index = faiss.IndexFlatL2(embeddings.shape[1])
    index.add(embeddings)
    st.success(f"📦 FAISS Index rebuilt with {index.ntotal} vectors")

    # Save to global_store.pkl
    progress.progress(90, text="💾 Updating global_store.pkl...")
    with open("global_store_gpu.pkl", "wb") as f:
        pickle.dump({
            "documents": documents,
            "outputs": outputs,
            "index": index,
            "embeddings":embeddings
        }, f)

    progress.progress(100, text="✅ All steps completed!")
    st.success("🎉 Updated and saved global store!")

def format_hover_text(text, idx, max_len=200, line_width=50):
    truncated = text[:max_len]
    # Insert <br> every `line_width` characters for better readability
    wrapped = '<br>'.join([truncated[i:i+line_width] for i in range(0, len(truncated), line_width)])
    return f"Index: {idx}<br>Text:<br>{wrapped}"

def format_hover_query(text, label="Text", max_len=200, line_width=50):
    truncated = text[:max_len]
    wrapped = '<br>'.join([truncated[i:i+line_width] for i in range(0, len(truncated), line_width)])
    return f"{label}:<br>{wrapped}"

# Ryff filtering ---
def filter_output_by_ryff(output_text, selected_ryff):
    if not selected_ryff:
        return output_text
    filtered = []
    for sentence in re.split(r'(?<=[.!?])\s+', output_text):
        for param in selected_ryff:
            if param.lower() in sentence.lower():
                filtered.append(sentence)
                break
    return " ".join(filtered) if filtered else "⚠️ No insights found for selected Ryff parameters."

# Streamlit App UI
st.title("🧠 Wellbeing Insight Generator")

text_input = st.text_area("📝 Describe your situation or thoughts")
mental_issue = st.text_input("💭 Mental health issue (e.g., anxiety, depression)")

ryff_params = [
    "Autonomy", "Environmental Mastery", "Personal Growth",
    "Positive Relations", "Purpose in Life", "Self-Acceptance"
]
selected_ryff = st.multiselect("🎯 Optional: Select up to 3 Ryff parameters", ryff_params, max_selections=3)

if st.button("🔍 Get Wellbeing Insight"):
    if not text_input or not mental_issue:
        st.warning("Please provide both text input and mental issue.")
    else:
```

```
        # Summarize input (no need to summarize works faster and better match)
        summarized = text_input

        # Create query
        query = f"{summarized} {mental_issue}"
        query_embedding = np.array([embedding_model.encode(query)]).astype("float32")

        # Retrieve from FAISS
        D, I = index.search(query_embedding, 5)
        match_score = 1 - D[0][0]
        matched_instruction_input = documents[I[0][0]]
        retrieved_output = outputs[I[0][0]]

        # Store all the 5 queries
        all_queries = {
        i: documents[i].split('.', 1)[1].strip() if '.' in documents[i] else documents[i]
        for i in I[0]
        }

        # Filter by Ryff parameters
        final_output = filter_output_by_ryff(retrieved_output, selected_ryff)
        matched_output = final_output

        if selected_ryff == []:
            selected_ryff = ryff_params

        # USING GEMINI 2.0 FLASH LLM THROUGH API TO REFINE IT BASED ON OBTAINED MATCH
        # Display matched document and its wellbeing insight

        st.subheader("\n🔍 Best Matched Record from RAG Store:")
        st.info(f"📥 Input: {matched_instruction_input}")
        st.info(f"📄 Output: {matched_output}")

        num_lines = len(selected_ryff)
        # Use Gemini to generate a fresh wellbeing insight
        prompt = f"""You are a part of RAG System. Based on the retrieved text : {matched_output} make small refinements on the ret
        """

        # Call Gemini model
        response = gemini_model.generate_content(prompt)
        generated_output = response.text.strip()
        final_output = generated_output

        # Output
        st.subheader("\n📘 Summary of your situation:")
        st.write(summarized)

        st.subheader("\n🧠 Wellbeing Insight:")
        st.write(final_output)

        st.subheader("🔗 Match Score:")
        st.write(f"{1 - D[0][0]:.4f} (higher is better)")

        # ========== Interactive Graph Section ==========
        st.subheader("\n📊 Generating interactive match visualization...")

        # Combine query and knowledge base embeddings
        query_embedding = query_embedding.reshape(1, -1)
        # embeddings = np.array([embedding_model.encode(doc) for doc in documents]).astype("float32")
        combined_embeddings = np.vstack([query_embedding, embeddings])

        # Run t-SNE to reduce to 3D
        reducer = TSNE(n_components=3, random_state=42, perplexity=30, max_iter=1000)
        reduced_all = reducer.fit_transform(combined_embeddings)

        # Separate reduced points
        query_3d = reduced_all[0]
        all_3d = reduced_all[1:]
        topk_indices = I[0]
        topk_3d = np.array([all_3d[i] for i in topk_indices])

        # Prepare trace for knowledge base
        kb_trace = go.Scatter3d(
            x=all_3d[:, 0], y=all_3d[:, 1], z=all_3d[:, 2],
            mode='markers',
            name='Knowledge Base',
            marker=dict(size=3, color='lightgray', opacity=0.5),
            hoverinfo='none'
        )


        # Prepare trace for top-k matches with annotations
```

```
        topk_trace = go.Scatter3d(
            x=topk_3d[:, 0], y=topk_3d[:, 1], z=topk_3d[:, 2],
            mode='markers+text',
            name='Top-k Matches',
            marker=dict(size=5, color='green'),
            text=[f"Index {idx}" for idx in topk_indices],
            textposition='top center',
            hoverinfo='text',
            hovertext=[
                format_hover_text(all_queries.get(idx, 'N/A'), idx)
                for idx in topk_indices
            ]
        )

        # Prepare trace for query point with tooltip
        query_trace = go.Scatter3d(
            x=[query_3d[0]], y=[query_3d[1]], z=[query_3d[2]],
            mode='markers+text',
            name='Query',
            marker=dict(size=10, color='red'),
            text=["Query"],
            textposition="top center",
            hoverinfo='text',
            hovertext=[format_hover_text(query, len(documents))]
        )
        # Prepare dotted lines from query to top-k
        lines = []
        for point in topk_3d:
            lines.append(go.Scatter3d(
                x=[query_3d[0], point[0]],
                y=[query_3d[1], point[1]],
                z=[query_3d[2], point[2]],
                mode='lines',
                line=dict(color='gray', dash='dot'),
                showlegend=False,
                hoverinfo='none'
            ))

        # Combine all traces
        fig = go.Figure(data=[kb_trace, topk_trace, query_trace] + lines)

        # Layout settings
        fig.update_layout(
            title="Interactive 3D t-SNE: Query → Top-k Matches",
            scene=dict(
                xaxis_title='Component 1',
                yaxis_title='Component 2',
                zaxis_title='Component 3'
            ),
            legend=dict(x=0, y=1),
            margin=dict(l=0, r=0, b=0, t=40)
        )

        # Show the interactive plot
        # fig.show()
        st.plotly_chart(fig, use_container_width=True)

        to_append_instruction = "Provide wellbeing insight for the below text with " + mental_issue + "."

        # Save to instruction_data.json
        append_to_json_file(to_append_instruction, summarized, final_output)
        st.success("✅ Your insight was saved to instruction_data.json")
```

```
Writing app.py
```

```
# Import ngrok
from pyngrok import ngrok

# Set your authtoken
ngrok.set_auth_token("2ohUKqk37HcGbvwN0s8Y1E2WNxE_39z1gVF3bYq9vFSEm7Wzq") # Replace YOUR_AUTHTOKEN with your actual authtoken

# Kill any existing ngrok processes
ngrok.kill()

# Start Streamlit with nohup
!nohup streamlit run app.py &

# Create a public URL with ngrok to access the app
public_url = ngrok.connect(addr='8501')
print(f"Public URL: {public_url}")
```

```
nohup: appending output to 'nohup.out'
Public URL: NgrokTunnel: "https://30d9-34-16-190-182.ngrok-free.app" -> "http://localhost:8501"
```

```
ngrok.kill()
```

Test in terminal

FULL RAG (RETRIEVAL + LLM + KNOWLEDGE BASE UPDATION)

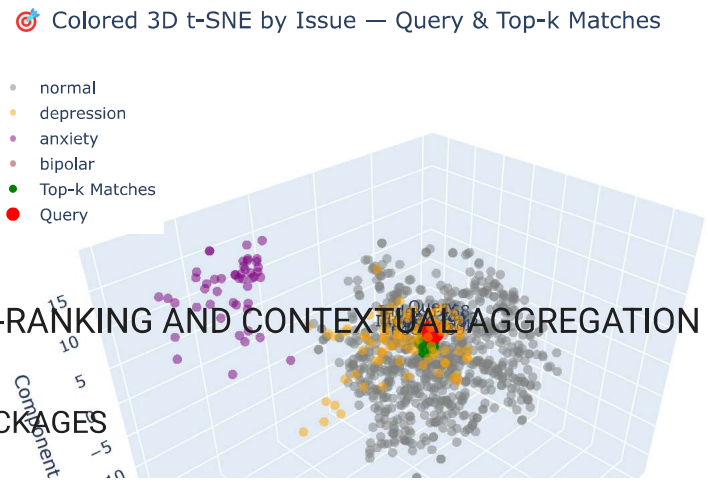
```
import json
import re
import pickle
import numpy as np
import faiss
from sentence_transformers import SentenceTransformer
import google.generativeai as genai
import plotly.graph_objs as go
from sklearn.manifold import TSNE
import plotly.offline as pyo
import os

# Configure Gemini API
genai.configure(api_key="AIzaSyCTdjPtjHrSYcU-_hLhSBAvnB0P9dKeDkc")
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 40,
    "max_output_tokens": 8192,
    "response_mime_type": "text/plain",
}
gemini_model = genai.GenerativeModel(
    model_name="gemini-2.0-flash",
    generation_config=generation_config,
)

# Load embedding model
with open("rag_embedding_gpu.pkl", "rb") as f:
    embedding_model = pickle.load(f)

# Load global store
with open("global_store_gpu.pkl", "rb") as f:
    store = pickle.load(f)

documents = store["documents"]
outputs = store["outputs"]
index = store["index"]
embeddings = store["embeddings"]

# Assuming you want to color the graph points by mental health issue extracted from the text
# You will need to create the 'metadatas' list containing the mental health issue for each record in 'documents'
metadatas = []
for document in documents:
    match = re.search(r"with (.+)\.", document)  # Adjust regex if needed
    issue = match.group(1).strip().lower() if match else "unknown"  # Default to 'unknown'
    metadatas.append({"issue": issue})

print(metadatas[0])

# Ryff scale filter
def filter_output_by_ryff(output_text, selected_ryff):
    if not selected_ryff:
        return output_text
    filtered = []
    for sentence in re.split(r'(?<=[.!?])\s+', output_text):
        for param in selected_ryff:
            if param.lower() in sentence.lower():
                filtered.append(sentence)
                break
    return " ".join(filtered) if filtered else "⚠️ No insights found for selected Ryff parameters."

def format_hover_text(text, idx, max_len=200, line_width=50):
    truncated = text[:max_len]
    # Insert <br> every `line_width` characters for better readability
    wrapped = '<br>'.join([truncated[i:i+line_width] for i in range(0, len(truncated), line_width)])
    return f"Index: {idx}<br>Text:<br>{wrapped}"

def format_hover_query(text, label="Text", max_len=200, line_width=50):
    truncated = text[:max_len]
    wrapped = '<br>'.join([truncated[i:i+line_width] for i in range(0, len(truncated), line_width)])
```

```
    return f"{label}:<br>{wrapped}"

# CLI
if __name__ == "__main__":
    print("\n🧠 Welcome to the Wellbeing Insight Generator (CLI Version)\n")

    text_input = input("📝 Describe your situation or thoughts:\n> ").strip()
    mental_issue = input("💭 Mental health issue (e.g., anxiety, depression):\n> ").strip()

    ryff_params = [
        "Autonomy", "Environmental Mastery", "Personal Growth",
        "Positive Relations", "Purpose in Life", "Self-Acceptance"
    ]
    print("\n🎯 Optional: Select up to 3 Ryff parameters (comma-separated):")
    print(", ".join(ryff_params))
    selected_input = input("> ").strip()
    selected_ryff = [r.strip() for r in selected_input.split(",") if r.strip() in ryff_params]

    if not text_input or not mental_issue:
        print("⚠️ Please provide both text input and mental issue.")
        exit()

    summarized = text_input
    query = f"{summarized} {mental_issue}"
    query_embedding = np.array([embedding_model.encode(query)]).astype("float32")

    # FAISS search
    D, I = index.search(query_embedding, 5)
    match_score = 1 - D[0][0]
    matched_instruction_input = documents[I[0][0]]
    retrieved_output = outputs[I[0][0]]

    # Store all the 5 queries
    all_queries = {
    i: documents[i].split('.', 1)[1].strip() if '.' in documents[i] else documents[i]
    for i in I[0]
    }

    # Filter by Ryff parameters
    final_output = filter_output_by_ryff(retrieved_output, selected_ryff)
    matched_output = final_output

    if selected_ryff == []:
        selected_ryff = ryff_params

    num_lines = len(selected_ryff)

    # Gemini refinement
    prompt = f"""You are a part of RAG System. Based on the retrieved text: {matched_output}, make small refinements on the retriev
    response = gemini_model.generate_content(prompt)
    generated_output = response.text.strip()
    final_output = generated_output

    # Output
    print("\n🔍 Top Matched RAG Entry:")
    print(f"📥 Input: {matched_instruction_input}")
    print(f"📄 Output: {matched_output}")
    print("\n📘 Summary:")
    print(summarized)
    print("\n🧠 Wellbeing Insight:")
    print(final_output)
    print(f"\n🔗 Match Score: {match_score:.4f}")

    # ========== Interactive Graph Section (with issue coloring) ==========

    print("\n📊 Generating interactive match visualization...")

    # Combine query and knowledge base embeddings
    query_embedding = query_embedding.reshape(1, -1)
    # embeddings = np.array([embedding_model.encode(doc) for doc in documents]).astype("float32")
    combined_embeddings = np.vstack([query_embedding, embeddings])

    # Run t-SNE to reduce to 3D
    reducer = TSNE(n_components=3, random_state=42, perplexity=30, n_iter=1000)
    reduced_all = reducer.fit_transform(combined_embeddings)

    query_3d = reduced_all[0]
    all_3d = reduced_all[1:]
    topk_indices = I[0]
    topk_3d = np.array([all_3d[i] for i in topk_indices])

    # Define issue colors
```

```
issue_colors = {
    "normal": "grey",
    "depression": "orange",
    "anxiety": "purple",
    "bipolar": "brown",
    "PTSD": "cyan"
}

# Group points by issue for colored scatter plot
issue_traces = []
for issue, color in issue_colors.items():
    indices = [i for i, meta in enumerate(metadatas) if meta["issue"] == issue]
    if not indices:
        continue
    points = np.array([all_3d[i] for i in indices])
    trace = go.Scatter3d(
        x=points[:, 0],
        y=points[:, 1],
        z=points[:, 2],
        mode='markers',
        name=f"{issue}",
        marker=dict(size=4, color=color, opacity=0.5),
        # text=[f"{issue} | {all_queries.get(i, 'N/A')}" for i in indices],
        hoverinfo='none'
    )
    issue_traces.append(trace)

# Color gradient for top-k matches
colorscale = ['#00FF00', '#33FF66', '#66FF99', '#99FFCC', '#CCFFF2']
topk_colors = colorscale[:len(topk_indices)]

topk_trace = go.Scatter3d(
    x=topk_3d[:, 0],
    y=topk_3d[:, 1],
    z=topk_3d[:, 2],
    mode='markers+text',
    name='Top-k Matches',
    marker=dict(size=6, color='green'),
    text=[f"Index {idx}" for idx in topk_indices],
    textposition='top center',
    hoverinfo='text',
    hovertext=[
        format_hover_text(all_queries.get(idx, 'N/A'), idx)
        for idx in topk_indices
    ]
)

query_trace = go.Scatter3d(
    x=[query_3d[0]],
    y=[query_3d[1]],
    z=[query_3d[2]],
    mode='markers+text',
    name='Query',
    marker=dict(size=10, color='red'),
    text=["Query"],
    textposition="top center",
    hoverinfo='text',
    hovertext=[format_hover_query(query, "Query")]
)

# Dotted lines from query to top-k
lines = []
for point in topk_3d:
    lines.append(go.Scatter3d(
        x=[query_3d[0], point[0]],
        y=[query_3d[1], point[1]],
        z=[query_3d[2], point[2]],
        mode='lines',
        line=dict(color='gray', width=2, dash='dot'),
        showlegend=False,
        hoverinfo='none'
    ))

# Combine all traces
fig = go.Figure(data=issue_traces + [topk_trace, query_trace] + lines)

fig.update_layout(
    title="🎯 Colored 3D t-SNE by Issue — Query & Top-k Matches",
    scene=dict(
        xaxis_title='Component 1',
        yaxis_title='Component 2',
        zaxis_title='Component 3',
```

```
            bgcolor='white'
        ),
        legend=dict(x=0.02, y=0.98),
        margin=dict(l=0, r=0, b=0, t=40)
    )

    fig.show()
```

```
{'issue': 'anxiety. everyday hyperventilation does anyone deal with daily like mini hyperventilation episodes? i feel im constantly

🧠 Welcome to the Wellbeing Insight Generator (CLI Version)

📝 Describe your situation or thoughts:
> hello world
💭 Mental health issue (e.g., anxiety, depression):
> normal

🎯 Optional: Select up to 3 Ryff parameters (comma-separated):
Autonomy, Environmental Mastery, Personal Growth, Positive Relations, Purpose in Life, Self-Acceptance
>

🔍 Top Matched RAG Entry:
📥 Input: Provide wellbeing insight for the below text with normal. what is normal? i am normal but i am not sure what you think is
📄 Output: The text suggests anxiety about conforming to undefined social norms.

**Wellbeing Insights (Ryff Scale):**

*   **Autonomy:** Define *your* values, not others'.
*   **Env. Mastery:** Focus on achievable goals.
*   **Personal Growth:** Learn a new skill to boost confidence.
*   **Pos. Relations:** Connect with accepting friends.
*   **Purpose:** Find meaning in small acts of kindness.
*   **Self-Acceptance:** Recognize strengths & accept imperfections.

📘 Summary:
hello world

🧠 Wellbeing Insight:
Given the anxiety about conforming to undefined social norms, consider these wellbeing insights:

*   **Autonomy:** Prioritize defining your own values over adhering to perceived social expectations.
*   **Env. Mastery:** Set realistic, personal goals to counteract feelings of being overwhelmed by external pressures.
*   **Personal Growth:** Enhance confidence by learning skills unrelated to social approval.
*   **Pos. Relations:** Cultivate connections with individuals who value you for who you are.
*   **Purpose:** Seek meaning in acts of kindness and self-compassion, independent of social validation.
*   **Self-Acceptance:** Acknowledge your strengths and accept imperfections, understanding that social norms are often flawed.

🔗 Match Score: 0.1586

📊 Generating interactive match visualization...
/usr/local/lib/python3.11/dist-packages/sklearn/manifold/_t_sne.py:1164: FutureWarning:

'n_iter' was renamed to 'max_iter' in version 1.5 and will be removed in 1.7.
```



RE-RANKING AND CONTEXTUAL AGGREGATION IN RAG

PACKAGES

```
!pip install faiss-cpu
!pip install huggingface-hub
!pip install Pillow
!pip install scikit-learn
!pip install sentence-transformers --no-deps
!pip install Streamlit
!pip install pyngrok
!pip install google-generativeai
```

```
Collecting faiss-cpu
  Downloading faiss_cpu-1.10.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (4.4 kB)
```

```
Requirement already satisfied: numpy<3.0,>=1.25.0 in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (2.0.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (24.2)
Downloading faiss_cpu-1.10.0-cp311-cp311-manylinux_2_28_x86_64.whl (30.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 30.7/30.7 MB 42.4 MB/s eta 0:00:00
Installing collected packages: faiss-cpu
Successfully installed faiss-cpu-1.10.0
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.11/dist-packages (0.30.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (2025.3.2)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (4.67.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub) (4.13
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hu
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (20
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (11.1.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: sentence-transformers in /usr/local/lib/python3.11/dist-packages (3.4.1)
Collecting Streamlit
  Downloading streamlit-1.44.1-py3-none-any.whl.metadata (8.9 kB)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (5.5.0)
Requirement already satisfied: blinker<2,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (1.9.0)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (5.5.2)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (8.1.8)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (2.0.2)
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (24.2)
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (2.2.2)
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (11.1.0)
Requirement already satisfied: protobuf<6,>=3.20 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (5.29.4)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (18.1.0)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (2.32.3)
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (9.1.2)
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.4.0 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (4.13.1)
Collecting watchdog<7,>=2.1.5 (from Streamlit)
  Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl.metadata (44 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 3.4 MB/s eta 0:00:00
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (3.1.44)
Collecting pydeck<1,>=0.8.0b4 (from Streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.11/dist-packages (from Streamlit) (6.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->Streamlit) (3.1.6)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->Streamlit) (4.23.
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->Streamlit) (1.33
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-packages (from gitpython!=3.1.19,<4,>=3.0.7->Str
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->Streamli
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->Streamlit) (2025.2
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->Streamlit) (2025
```

GETTING THE CROSS ENCODER

```
from sentence_transformers import CrossEncoder
import pickle

# Load the cross-encoder model (use device='cuda' if using GPU runtime)
cross_encoder_model = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2', device='cuda')

# Save it to a pickle file
with open("cross_encoder_gpu.pkl", "wb") as f:
    pickle.dump(cross_encoder_model, f)

print("✅ Saved cross_encoder_model to pkl")

```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100%                                                   794/794 [00:00<00:00, 80.0kB/s]
model.safetensors: 100%                                           90.9M/90.9M [00:00<00:00, 282MB/s]
tokenizer_config.json: 100%                                       1.33k/1.33k [00:00<00:00, 111kB/s]
vocab.txt: 100%                                                   232k/232k [00:00<00:00, 535kB/s]
tokenizer.json: 100%                                              711k/711k [00:00<00:00, 11.4MB/s]
special_tokens_map.json: 100%                                     132/132 [00:00<00:00, 12.9kB/s]
✅ Saved cross_encoder_model to pkl
```

TERMINAL TEST

UPDATED WITH COLORS FOR ISSUES AND HOVERTEXT WRAPPING

```python
import json
import re
import pickle
import numpy as np
import faiss
import sys # For exiting on critical errors
import time

# Core ML/AI Libraries
from sentence_transformers import SentenceTransformer, CrossEncoder
# Keep transformers import even if pipeline not used directly now
from transformers import pipeline
import google.generativeai as genai

# Visualization (Optional for Colab, might open new tab or render statically)
import plotly.graph_objs as go
from sklearn.manifold import TSNE
# import plotly.offline as pyo # Not typically needed in Colab for fig.show()
import os
from collections import defaultdict # Make sure this import is at the top of your script

# --- Configuration ---
# IMPORTANT: Replace with your actual API key securely
GEMINI_API_KEY = "AIzaSyCTdjPtjHrSYcU-_hLhSBAvnB0P9dKeDkc" # <<<--- ### REPLACE WITH YOUR KEY ### (Using placeholder for safety)
if GEMINI_API_KEY == "YOUR_GEMINI_API_KEY" or len(GEMINI_API_KEY) < 30: # Basic check
    print("🚨 ERROR: Please replace 'YOUR_GEMINI_API_KEY' with your actual Gemini API key in the code.")
    sys.exit(1) # Exit if key not set

try:
    genai.configure(api_key=GEMINI_API_KEY)
    print("✅ Gemini API Configured.")
except Exception as e:
    print(f"🚨 ERROR: Failed to configure Gemini API: {e}")
    sys.exit(1)

# --- Constants ---
EMBEDDING_MODEL_PKL = "rag_embedding_gpu.pkl"
GLOBAL_STORE_PKL = "global_store_gpu.pkl"
INSTRUCTION_DATA_JSON = "instruction_data.json"
# Load CrossEncoder FROM THE PICKLE FILE (as per previous steps)
CROSS_ENCODER_PKL = 'cross_encoder_gpu.pkl'
INITIAL_RETRIEVAL_K = 10 # Retrieve more candidates initially from FAISS
RE_RANKED_TOP_N = 5 # Use top N re-ranked results for LLM context
ISSUE_KEYWORDS = ["normal", "anxiety", "bipolar", "ptsd", "depression"]

# --- Color Mapping for Plot --- ADDED
issue_colors = {
    "normal": "#1f77b4",     # Strong blue
    "depression": "#ff7f0e", # Vivid orange
    "anxiety": "#9467bd",    # Medium purple
    "bipolar": "#8c564b",    # Muted brown
    "ptsd": "#17becf",       # Teal / Cyan
    "unknown": "#7f7f7f"     # Neutral gray
}

```

```
DEFAULT_PLOT_COLOR = "silver" # Fallback color if issue not in map

# --- Model Loading ---
# Load models ONCE at the start
def load_models():
    print("\n--- Loading Models ---")
    embedding_model = None
    cross_encoder = None
    gemini_model = None

    # Load Embedding Model
    try:
        with open(EMBEDDING_MODEL_PKL, "rb") as f:
            embedding_model = pickle.load(f)
        print(f"✅ Loaded embedding_model from {EMBEDDING_MODEL_PKL}")
    except FileNotFoundError:
        print(f"🚨 ERROR: Embedding model file not found: {EMBEDDING_MODEL_PKL}.")
        sys.exit(1)
    except Exception as e:
        print(f"🚨 ERROR: Error loading embedding model: {e}")
        sys.exit(1)

    # Load CrossEncoder Model from Pickle
    try:
        with open(CROSS_ENCODER_PKL, "rb") as f:
            cross_encoder = pickle.load(f)
        print(f"✅ Loaded CrossEncoder model from pickle: {CROSS_ENCODER_PKL}")
        print("   ⚠️ Reminder: Loading models from pickle is less robust than standard methods.")
    except FileNotFoundError:
        print(f"🚨 ERROR: CrossEncoder pickle file not found: {CROSS_ENCODER_PKL}.")
        sys.exit(1)
    except Exception as e:
        print(f"🚨 ERROR: Error loading CrossEncoder model from pickle: {e}")
        sys.exit(1)

    # Configure Gemini Model
    try:
        generation_config = {
            "temperature": 0.8,
            "top_p": 0.95,
            "top_k": 40,
            "max_output_tokens": 8192,
            "response_mime_type": "text/plain",
        }
        gemini_model = genai.GenerativeModel(
            model_name="gemini-1.5-flash",
            generation_config=generation_config,
        )
        print("✅ Configured Gemini model (gemini-1.5-flash)")
    except Exception as e:
        print(f"🚨 ERROR: Failed to configure Gemini model: {e}")
        sys.exit(1)

    print("--- Models Loaded Successfully ---")
    return embedding_model, cross_encoder, gemini_model

embedding_model, cross_encoder, gemini_model = load_models()


# --- Data Loading ---
# Load data store content (can be reloaded if updated)
def load_global_store():
    print(f"\n--- Loading Knowledge Base from {GLOBAL_STORE_PKL} ---")
    try:
        with open(GLOBAL_STORE_PKL, "rb") as f:
            store = pickle.load(f)
        required_keys = {"documents", "outputs", "index", "embeddings"}
        if not required_keys.issubset(store.keys()):
            print(f"🚨 ERROR: Global store file {GLOBAL_STORE_PKL} is missing required keys.")
            return None, None, None, None, None # Indicate failure + metadatas
        documents = store.get("documents", [])
        outputs = store.get("outputs", [])
        index = store.get("index")
        embeddings = store.get("embeddings")
        print(f"🔁 Loaded {len(documents)} documents.")
        if not isinstance(index, faiss.Index):
             print("   ⚠️ Warning: Loaded 'index' might not be a valid Faiss index.")
        elif index.ntotal == 0:
             print("   ⚠️ Warning: Faiss index is empty.")

        # --- Extract Metadata --- ADDED HERE
        metadatas = []
```

```
        print("   -> Extracting metadata (issue) by searching keywords in first sentence...")

        for document in documents:
            issue = "unknown" # Default issue

            if document and isinstance(document, str): # Check if document is a non-empty string
                # Extract first sentence (or whole doc if no period) and lowercase it
                first_period_index = document.find('.')
                if first_period_index != -1:
                    # Extract up to and including the first period
                    first_sentence = document[:first_period_index + 1].lower()
                else:
                    # If no period, search the whole document (lowercased)
                    # Alternatively, you could take the first N characters: document[:100].lower()
                    first_sentence = document.lower()

                # Search for keywords in the defined order
                for keyword in ISSUE_KEYWORDS:
                    # Use simple 'in' check (case-insensitive due to lowercasing first_sentence)
                    if keyword in first_sentence:
                        issue = keyword # Assign the first keyword found
                        break # Stop searching once a keyword is found

            metadatas.append({"issue": issue})

        # --- End Metadata Extraction ---

        return documents, outputs, index, embeddings, metadatas # Return metadatas too

    except FileNotFoundError:
        print(f"🚨 WARNING: Global store file not found: {GLOBAL_STORE_PKL}. KB is empty.")
        return [], [], None, None, [] # Return empty lists/None
    except Exception as e:
        print(f"🚨 ERROR: Error loading global store: {e}")
        return None, None, None, None, None # Indicate failure


# --- Functions ---

# Append new input and response (Updates files directly)
def append_to_json_file(instruction, input_text, output, filename=INSTRUCTION_DATA_JSON, store_filename=GLOBAL_STORE_PKL):
    # (Content of this function remains the same as previous version)
    print("\n--- Updating Knowledge Base ---")
    try:
        print(f"📄 Reading/Appending {filename}...")
        try:
            with open(filename, "r") as file: content = json.load(file)
        except (FileNotFoundError, json.JSONDecodeError):
            print(f"   -> {filename} not found or invalid. Starting fresh."); content = []
        new_record = {"instruction": instruction, "input": input_text, "output": output}
        content.append(new_record)
        with open(filename, "w") as file: json.dump(content, file, indent=4)
        print(f"   -> Record appended to {filename}")

        print("🔄 Rebuilding internal lists...")
        current_documents = [f"{item.get('instruction','')} {item.get('input','')}" for item in content]
        current_outputs = [item.get("output", "") for item in content]
        print(f"   -> Total Records: {len(current_documents)}")

        if not current_documents:
            print("   -> No documents to index."); current_index = None; current_embeddings = None
        else:
            print("🔍 Re-encoding documents...");
            try:
                new_embeddings = np.array([embedding_model.encode(doc) for doc in current_documents]).astype("float32")
            except Exception as e: print(f"   -> 🚨 ERROR: Failed to encode documents: {e}"); return False
            print("📦 Building new FAISS Index...")
            if new_embeddings.shape[0] > 0:
                dimension = new_embeddings.shape[1]; current_index = faiss.IndexFlatL2(dimension)
                current_index.add(new_embeddings); current_embeddings = new_embeddings
                print(f"   -> FAISS Index rebuilt ({current_index.ntotal} vectors, Dim: {dimension})")
            else: print("   -> No embeddings generated, index not rebuilt."); current_index = None; current_embeddings = None

        print(f"💾 Saving updated global store to {store_filename}...")
        updated_store = {"documents": current_documents, "outputs": current_outputs, "index": current_index, "embeddings": curren
        with open(store_filename, "wb") as f: pickle.dump(updated_store, f)

        print("--- Knowledge Base Update Complete! ---"); return True
    except Exception as e: print(f"🚨 ERROR: An error occurred during KB update: {e}"); return False


# --- Hover Text Formatting Functions ---
```

```
220
221 # Original function for TERMINAL output (uses newline \n)
222 def format_terminal_hover_text(text, idx, max_len=200, line_width=70):
223     text = str(text) if text is not None else ""
224     truncated = text[:max_len] + ('...' if len(text) > max_len else '')
225     wrapped_lines = [truncated[i:i+line_width] for i in range(0, len(truncated), line_width)]
226     wrapped = '\n'.join(wrapped_lines)
227     return f"Index: {idx}\nText:\n{wrapped}"
228
229 # NEW function for PLOTLY hover text (uses <br>)
230 def format_plot_hover_text(text, idx, issue="N/A", max_len=200, line_width=50):
231     text = str(text) if text is not None else ""
232     truncated = text[:max_len] + ('...' if len(text) > max_len else '')
233     # Replace potential HTML tags in text to avoid breaking hover layout
234     safe_truncated = truncated.replace('<', '&lt;').replace('>', '&gt;')
235     wrapped_lines = [safe_truncated[i:i+line_width] for i in range(0, len(safe_truncated), line_width)]
236     wrapped = '<br>'.join(wrapped_lines)
237     return f"<b>Index: {idx}</b><br>Issue: {issue}<br>Text:<br>{wrapped}"
238
239 # NEW function for PLOTLY query hover text (uses <br>)
240 def format_plot_hover_query(text, label="Query", max_len=200, line_width=50):
241     text = str(text) if text is not None else ""
242     truncated = text[:max_len] + ('...' if len(text) > max_len else '')
243     safe_truncated = truncated.replace('<', '&lt;').replace('>', '&gt;')
244     wrapped_lines = [safe_truncated[i:i+line_width] for i in range(0, len(safe_truncated), line_width)]
245     wrapped = '<br>'.join(wrapped_lines)
246     return f"<b>{label}:</b><br>{wrapped}"
247 # --- End Hover Text Functions ---
248
249 # Ryff filtering (Ensure input is string) - No changes needed functionally
250 def filter_output_by_ryff(output_text, selected_ryff):
251     # (Content of this function remains the same as previous version)
252     if not selected_ryff: return str(output_text) if output_text is not None else ""
253     output_text = str(output_text) if output_text is not None else ""
254     filtered = []
255     sentences = [s for s in re.split(r'(?<=[.!?])\s+', output_text) if s]
256     for sentence in sentences:
257         for param in selected_ryff:
258             if param.lower() in sentence.lower(): filtered.append(sentence.strip()); break
259     return " ".join(filtered) if filtered else "⚠️ No insights found for selected Ryff parameters."
260
261 # --- Main Interaction Loop ---
262 def main():
263     while True: # Loop for multiple queries
264         # Load the latest data AND METADATA at the start of each iteration
265         documents, outputs, index, embeddings, metadatas = load_global_store()
266
267         # Critical check after loading
268         if documents is None:
269              print("🚨 Exiting due to failure loading knowledge base.")
270              break
271         if not documents:
272              print("⚠️ Knowledge base is currently empty.")
273              # Optionally allow adding data here or just prompt for query anyway
274
275         print("\n==========================================")
276         print("🧠 Advanced Wellbeing Insight Generator")
277         print("==========================================")
278
279         # --- Get User Input ---
280         print("📝 Describe your situation or thoughts:")
281         text_input = input("> ")
282         if not text_input: print("⚠️ Input cannot be empty. Please try again."); continue
283         print("\n💭 Mental health issue (e.g., anxiety, depression):")
284         mental_issue = input("> ")
285         if not mental_issue: print("⚠️ Mental issue cannot be empty. Please try again."); continue
286
287         # Ryff Parameter Selection (Logic remains the same)
288         ryff_params = ["Autonomy", "Environmental Mastery", "Personal Growth", "Positive Relations", "Purpose in Life", "Self-Acc
289         selected_ryff = []
290         print("\n🎯 Optional: Select up to 3 Ryff parameters to focus insight."); print("   Available parameters:")
291         for i, param in enumerate(ryff_params): print(f"     {i+1}. {param}")
292         print("   Enter numbers separated by commas (e.g., 1,3,5) or press Enter for all:"); ryff_selection = input("> ").strip()
293         if ryff_selection:
294             try:
295                 selected_indices = [int(x.strip()) - 1 for x in ryff_selection.split(',') if x.strip()]
296                 if len(selected_indices) > 3: print("   -> Warning: Too many selections. Using the first 3."); selected_indices =
297                 selected_ryff = [ryff_params[i] for i in selected_indices if 0 <= i < len(ryff_params)]
298                 if selected_ryff: print(f"   -> Selected Ryff parameters: {', '.join(selected_ryff)}")
299                 else: print("   -> No valid parameters selected. Will use all."); selected_ryff = ryff_params # Default to all if
300             except (ValueError, IndexError): print("   -> Invalid input. Will use all Ryff parameters."); selected_ryff = ryff_pa
301         else: print("   -> No parameters selected. Will provide insight for all Ryff parameters."); selected_ryff = ryff_params
```

```
    print("============================================")


    # --- Input Validation for KB Index ---
    if index is None or not hasattr(index, 'ntotal') or index.ntotal == 0:
         print("\n🚨 ERROR: Knowledge Base Index is empty or invalid. Cannot perform retrieval.")
         try_again = input("Knowledge base empty. Try another query? (y/n): ").lower()
         if try_again != 'y': break
         else: continue

    # --- RAG Process ---
    # (Steps 1: Query, 2: Retrieval, 3: Re-ranking remain the same)
    print("\n⏳ Processing: Retrieving -> Re-ranking -> Generating...")
    start_rag_time = time.time()
    query = f"Provide wellbeing insight for the below text with {mental_issue}. {text_input}" # Step 1

    # Step 2: Initial Retrieval
    print("   Step 1: Initial Retrieval from Knowledge Base...")
    try:
        query_embedding = np.array([embedding_model.encode(query)]).astype("float32"); k_initial = min(INITIAL_RETRIEVAL_K, i
        distances, initial_indices = index.search(query_embedding, k_initial); initial_indices = initial_indices[0]; distance
        if len(initial_indices) == 0: print("   ⚠️ No documents found in initial retrieval."); continue
        print(f"   -> Retrieved {len(initial_indices)} candidates.")
    except Exception as e: print(f"   -> 🚨 ERROR during initial retrieval: {e}"); continue

    # Step 3: Re-ranking
    print("   Step 2: Re-ranking candidates for relevance...")
    try:
        retrieved_docs_text = [documents[i] for i in initial_indices]; cross_inp = [[query, doc_text] for doc_text in retriev
        cross_scores = cross_encoder.predict(cross_inp); reranked_results = sorted(zip(initial_indices, cross_scores), key=la
        n_final = min(RE_RANKED_TOP_N, len(reranked_results)); reranked_indices = [idx for idx, score in reranked_results[:n_
        if not reranked_indices: print("   -> ⚠️ Re-ranking resulted in zero candidates."); continue
        print(f"   -> Re-ranked and selected top {len(reranked_indices)} candidates.")
    except Exception as e:
        print(f"   -> 🚨 ERROR during re-ranking: {e}"); print("      Falling back to top results from initial retrieval.")
        n_final = min(RE_RANKED_TOP_N, len(initial_indices)); reranked_indices = initial_indices[:n_final]; reranked_scores =

    # Step 4: Prepare Context & Display Matches (Using TERMINAL hover function here)
    print("   Step 3: Preparing context for Generation...")
    combined_context_for_llm = ""; print("\n--- Top Re-ranked Records Used as Context ---")
    context_parts = []
    for i, idx in enumerate(reranked_indices):
         try:
             original_input_text = documents[idx]; original_output_text = outputs[idx]
             filtered_output = filter_output_by_ryff(original_output_text, selected_ryff)
             context_parts.append(f"--- Context Source {i+1} (Score: {reranked_scores[i]:.4f}) ---\n{filtered_output}")
             # Use TERMINAL format for printing details
             print(f"\n### Match {i+1} (Index: {idx}, Score: {reranked_scores[i]:.4f}) ###")
             print(f"  Retrieved Input (Instruction+Situation):\n    {format_terminal_hover_text(original_input_text, idx)}")
             print(f"  Retrieved Output (Filtered for Context):\n    {filtered_output}")
         except IndexError: print(f"   -> Warning: Index {idx} out of bounds. Skipping."); continue
         except Exception as e: print(f"   -> Warning: Error processing context index {idx}: {e}. Skipping."); continue
    combined_context_for_llm = "\n\n".join(context_parts); print("--------------------------------------------")

    # Step 5: Generation (Prompt slightly adjusted for clarity)
```