```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os

# Base project directory on Google Drive
BASE_DIR = "/content/drive/MyDrive/SkinAI_Project"

# Data dirs
DATA_DIR       = f"{BASE_DIR}/data"
RAW_DIR        = f"{DATA_DIR}/raw"
PROCESSED_DIR  = f"{DATA_DIR}/processed"
SYNTHETIC_DIR  = f"{DATA_DIR}/synthetic"

MODELS_DIR     = f"{BASE_DIR}/models"
NOTEBOOKS_DIR  = f"{BASE_DIR}/notebooks"

print("BASE_DIR      :", BASE_DIR)
print("SYNTHETIC_DIR :", SYNTHETIC_DIR)

print("\nSynthetic dir contents:")
print(os.listdir(SYNTHETIC_DIR))
```

```
BASE_DIR      : /content/drive/MyDrive/SkinAI_Project
SYNTHETIC_DIR : /content/drive/MyDrive/SkinAI_Project/data/synthetic

Synthetic dir contents:
['clinical_cases_10000.csv', 'patient_profiles_10000.csv', 'treatment_records_10000.csv', 'disease_master.csv', 'symptom_list.cs
```

```
import pandas as pd
import os

disease_master_path = os.path.join(SYNTHETIC_DIR, "disease_master.csv")
symptom_list_path   = os.path.join(SYNTHETIC_DIR, "symptom_list.csv")
edges_path          = os.path.join(SYNTHETIC_DIR, "disease_symptom_edges.csv")

print("disease_master_path:", disease_master_path, "| exists:", os.path.exists(disease_master_path))
print("symptom_list_path  :", symptom_list_path,   "| exists:", os.path.exists(symptom_list_path))
print("edges_path         :", edges_path,          "| exists:", os.path.exists(edges_path))

df_disease_master = pd.read_csv(disease_master_path)
df_symptom_list   = pd.read_csv(symptom_list_path)
df_edges          = pd.read_csv(edges_path)

print("\n=== disease_master ===")
print(df_disease_master.shape)
print(df_disease_master.columns.tolist())
display(df_disease_master.head())

print("\n=== symptom_list ===")
print(df_symptom_list.shape)
print(df_symptom_list.columns.tolist())
display(df_symptom_list.head())

print("\n=== disease_symptom_edges ===")
print(df_edges.shape)
print(df_edges.columns.tolist())
display(df_edges.head())
```

```
disease_master_path: /content/drive/MyDrive/SkinAI_Project/data/synthetic/disease_master.csv | exists: True
symptom_list_path  : /content/drive/MyDrive/SkinAI_Project/data/synthetic/symptom_list.csv | exists: True
edges_path         : /content/drive/MyDrive/SkinAI_Project/data/synthetic/disease_symptom_edges.csv | exists: True

=== disease_master ===
(29, 3)
['disease_name', 'category', 'description']
```

	disease_name	category	description
0	Acne vulgaris	inflammatory	Chronic inflammatory disease with comedones, p...
1	Atopic dermatitis (Eczema)	inflammatory	Chronic relapsing inflammatory skin disease wi...
2	Psoriasis vulgaris	inflammatory	Chronic immune-mediated disease with well-dema...
3	Seborrheic dermatitis	inflammatory	Erythematous, scaly eruption in seborrheic are...
4	Contact dermatitis (Irritant)	inflammatory	Inflammatory reaction from direct damage by ir...

```
=== symptom_list ===
(83, 2)
['symptom_id', 'symptom_name']
```

	symptom_id	symptom_name
0	1	Wickham striae
1	2	angioedema
2	3	around hair follicles
3	4	band-like pain
4	5	blackheads

```
=== disease_symptom_edges ===
(89, 3)
['disease_name', 'symptom_id', 'symptom_name']
```

	disease_name	symptom_id	symptom_name
0	Acne vulgaris	49	pimples
1	Acne vulgaris	5	blackheads
2	Acne vulgaris	82	whiteheads
3	Acne vulgaris	46	oily skin
4	Atopic dermatitis (Eczema)	34	itching

```
import networkx as nx

G = nx.Graph()

# --- Add disease nodes ---
for _, row in df_disease_master.iterrows():
    dname = str(row["disease_name"]).strip()
    if not dname:
        continue
    G.add_node(
        f"disease:{dname}",
        node_type="disease",
        disease_name=dname,
        category=str(row.get("category", "")).strip(),
        description=str(row.get("description", "")).strip()
    )

# --- Add symptom nodes ---
# We'll use symptom_id as unique id if it exists, else symptom_name
symptom_id_col   = "symptom_id" if "symptom_id" in df_symptom_list.columns else None
symptom_name_col = "symptom_name" if "symptom_name" in df_symptom_list.columns else None

for _, row in df_symptom_list.iterrows():
    if symptom_id_col:
        sid = row[symptom_id_col]
```

```
        else:
            sid = str(row[symptom_name_col]).strip()

        sname = str(row[symptom_name_col]).strip() if symptom_name_col else str(sid)

        if not sname:
            continue

        G.add_node(
            f"symptom:{sid}",
            node_type="symptom",
            symptom_id=sid,
            symptom_name=sname
        )

# --- Add edges from disease_symptom_edges ---
edge_count = 0
for _, row in df_edges.iterrows():
    dname = str(row["disease_name"]).strip()
    sid   = row["symptom_id"] if "symptom_id" in row else None

    # Edge only if both ends exist
    d_node = f"disease:{dname}"
    s_node = f"symptom:{sid}"

    if d_node in G.nodes and s_node in G.nodes:
        G.add_edge(d_node, s_node)
        edge_count += 1

print(f"\nGraph built with {G.number_of_nodes()} nodes and {G.number_of_edges()} edges.")
print(f"Edges added from edges CSV: {edge_count}")
```

```
Graph built with 112 nodes and 89 edges.
Edges added from edges CSV: 89
```

```
# Map disease_name -> node_id
DISEASE_NODES = {
    data["disease_name"]: node_id
    for node_id, data in G.nodes(data=True)
    if data.get("node_type") == "disease"
}

# Map symptom_id -> symptom_name (from df_symptom_list)
SYMPTOM_ID_TO_NAME = {}
if "symptom_id" in df_symptom_list.columns:
    for _, row in df_symptom_list.iterrows():
        SYMPTOM_ID_TO_NAME[row["symptom_id"]] = str(row.get("symptom_name", "")).strip()

print("Example disease nodes:", list(DISEASE_NODES.keys())[:10])
print("Example symptom ids:", list(SYMPTOM_ID_TO_NAME.items())[:10])
```

```
Example disease nodes: ['Acne vulgaris', 'Atopic dermatitis (Eczema)', 'Psoriasis vulgaris', 'Seborrheic dermatitis', 'Contact d
Example symptom ids: [(1, 'Wickham striae'), (2, 'angioedema'), (3, 'around hair follicles'), (4, 'band-like pain'), (5, 'blackh
```

```
def get_symptoms_for_disease(disease_name: str, top_n: int = 10):
    """
    Return list of (symptom_name, symptom_id) connected to the given disease in G.
    """
    dname_norm = disease_name.strip()
    d_node = DISEASE_NODES.get(dname_norm)

    if d_node is None:
        print(f"[get_symptoms_for_disease] Disease '{disease_name}' not found in graph.")
        return []

    neighbors = list(G.neighbors(d_node))

    # Extract symptom info
    symptom_list = []
    for n in neighbors:
        data = G.nodes[n]
        if data.get("node_type") != "symptom":
            continue
```

```
            sid = data.get("symptom_id")
            sname = data.get("symptom_name") or SYMPTOM_ID_TO_NAME.get(sid, "")
            symptom_list.append((sname, sid))

        # For now, all have equal weight; you could later add frequency weighting if needed.
        # Just sort by name for stable results.
        symptom_list = sorted(symptom_list, key=lambda x: x[0])

        if top_n is not None:
            symptom_list = symptom_list[:top_n]

        return symptom_list
```

```
def get_related_diseases(disease_name: str, top_n: int = 5):
    """
    Find diseases related to the given one based on shared symptoms.
    If no shared-symptom diseases are found, fall back to diseases with same category.
    Returns list of dicts: {"disease_name", "shared_symptom_count", "category", "reason"}
    """
    dname_norm = disease_name.strip()
    d_node = DISEASE_NODES.get(dname_norm)

    if d_node is None:
        print(f"[get_related_diseases] Disease '{disease_name}' not found in graph.")
        return []

    # Symptoms of the given disease
    neighbors = list(G.neighbors(d_node))
    disease_symptoms = {
        G.nodes[n].get("symptom_id") for n in neighbors
        if G.nodes[n].get("node_type") == "symptom"
    }

    related = []

    # Check all other disease nodes for shared symptoms
    for other_name, other_node in DISEASE_NODES.items():
        if other_name == dname_norm:
            continue
        neighbors_other = list(G.neighbors(other_node))
        other_symptoms = {
            G.nodes[n].get("symptom_id") for n in neighbors_other
            if G.nodes[n].get("node_type") == "symptom"
        }
        shared = disease_symptoms.intersection(other_symptoms)
        if len(shared) > 0:
            cat = G.nodes[other_node].get("category", "")
            related.append({
                "disease_name": other_name,
                "shared_symptom_count": len(shared),
                "category": cat,
                "reason": "shared_symptoms"
            })

    # If we found some by shared symptoms, sort and return
    if related:
        related_sorted = sorted(
            related,
            key=lambda x: x["shared_symptom_count"],
            reverse=True
        )
        if top_n is not None:
            related_sorted = related_sorted[:top_n]
        return related_sorted

    # Fallback: use same category from df_disease_master
    print("[get_related_diseases] No shared-symptom diseases found. Using same-category fallback.")

    # Get category of the original disease
    row_ref = df_disease_master[df_disease_master["disease_name"].str.strip() == dname_norm]
    if row_ref.empty or "category" not in row_ref.columns:
        return []

    cat = str(row_ref.iloc[0]["category"]).strip()
```

```
    same_cat = df_disease_master[
        (df_disease_master["category"].astype(str).str.strip() == cat) &
        (df_disease_master["disease_name"].str.strip() != dname_norm)
    ]

    results = []
    for _, row in same_cat.iterrows():
        other_name = str(row["disease_name"]).strip()
        results.append({
            "disease_name": other_name,
            "shared_symptom_count": 0,
            "category": cat,
            "reason": "same_category"
        })

    if top_n is not None:
        results = results[:top_n]

    return results
```

```
def explain_disease_with_kg(disease_name: str, top_symptoms: int = 10, top_related: int = 5):
    """
    Create a structured explanation for a given disease using the knowledge graph:
      - key symptoms
      - related diseases
      - category & description (from disease_master)
    """
    dname_norm = disease_name.strip()

    # Metadata from df_disease_master
    row = df_disease_master[df_disease_master["disease_name"].str.strip() == dname_norm]

    if not row.empty:
        category = str(row.iloc[0].get("category", "")).strip()
        description = str(row.iloc[0].get("description", "")).strip()
    else:
        category = ""
        description = ""

    symptoms = get_symptoms_for_disease(dname_norm, top_n=top_symptoms)
    related_diseases = get_related_diseases(dname_norm, top_n=top_related)

    return {
        "disease_name": dname_norm,
        "category": category,
        "description": description,
        "symptoms": [
            {"symptom_name": sname, "symptom_id": sid}
            for (sname, sid) in symptoms
        ],
        "related_diseases": related_diseases
    }
```

```
def print_kg_explanation(kg_info: dict):
    if not kg_info:
        print("No KG info available.")
        return

    dname = kg_info.get("disease_name", "")
    category = kg_info.get("category", "")
    desc = kg_info.get("description", "")

    print(f"=== Knowledge Graph Explanation for: {dname} ===\n")

    if category:
        print(f"Category: {category}")

    if desc:
        print("\nDescription:")
        print(desc)

    # Symptoms
    print("\nTop associated symptoms:")
    symptoms = kg_info.get("symptoms", [])
```

```
    if not symptoms:
        print(" - (no symptoms found in KG)")
    else:
        for s in symptoms:
            print(f" - {s['symptom_name']} (id={s['symptom_id']})")

    # Related diseases
    print("\nRelated diseases:")
    rel = kg_info.get("related_diseases", [])
    if not rel:
        print(" - (no related diseases found in KG)")
    else:
        for r in rel:
            print(
                f" - {r['disease_name']} "
                f"| shared_symptoms={r['shared_symptom_count']} "
                f"| category={r.get('category', '')} "
                f"| reason={r.get('reason', '')}"
            )
```

```
# Show some disease names to test with
print("Example diseases (first 20):")
for name in df_disease_master["disease_name"].head(20):
    print(" -", name)

# Choose one of them (e.g., 'Acne vulgaris' if it exists)
test_disease = df_disease_master["disease_name"].iloc[0]
print("\nTesting KG explanation for disease:", test_disease)

kg_info = explain_disease_with_kg(test_disease, top_symptoms=10, top_related=5)
print_kg_explanation(kg_info)
```

```
Example diseases (first 20):
 - Acne vulgaris
 - Atopic dermatitis (Eczema)
 - Psoriasis vulgaris
 - Seborrheic dermatitis
 - Contact dermatitis (Irritant)
 - Contact dermatitis (Allergic)
 - Urticaria (Hives)
 - Rosacea
 - Vitiligo
 - Tinea corporis (Ringworm)
 - Tinea pedis (Athlete's foot)
 - Tinea cruris (Jock itch)
 - Tinea capitis (Scalp ringworm)
 - Pityriasis versicolor
 - Impetigo
 - Folliculitis
 - Cellulitis
 - Scabies
 - Herpes zoster (Shingles)
 - Herpes simplex (Cold sores)

Testing KG explanation for disease: Acne vulgaris
[get_related_diseases] No shared-symptom diseases found. Using same-category fallback.
=== Knowledge Graph Explanation for: Acne vulgaris ===

Category: inflammatory

Description:
Chronic inflammatory disease with comedones, papules, and pustules, mainly on face and trunk.

Top associated symptoms:
 - blackheads (id=5)
 - oily skin (id=46)
 - pimples (id=49)
 - whiteheads (id=82)

Related diseases:
 - Atopic dermatitis (Eczema) | shared_symptoms=0 | category=inflammatory | reason=same_category
 - Psoriasis vulgaris | shared_symptoms=0 | category=inflammatory | reason=same_category
 - Seborrheic dermatitis | shared_symptoms=0 | category=inflammatory | reason=same_category
 - Contact dermatitis (Irritant) | shared_symptoms=0 | category=inflammatory | reason=same_category
 - Contact dermatitis (Allergic) | shared_symptoms=0 | category=inflammatory | reason=same_category
```