SINGAPORE UNIVERSITY OF SOCIAL SCIENCES

ICT233 Data Programming

Tutor-Marked Assignment

July 2025 Presentation

TUTOR-MARKED ASSIGNMENT (TMA)

This assignment is worth 18% of the final mark for ICT233, Data Programming.

The cut-off date for this assignment is **Thursday, 09 October 2025, 2355 hours**.

Note to Students:

You are to include the following particulars in your submission: Course Code, Title of the TMA, SUSS PI No., Your Name, and Submission Date.

1. Please provide answers into the given template ICT233_TMA_JULY25.ipynb between the following tags:

############### Your code starts here ###############

################ Your code ends here ################

2. Please do not modify the given code in the template, do not add/delete/split any code cell.

3. Please print out your analysis and insights.

3. If there is any discrepancy between the questions displayed in the template and those in the TMA paper (in pdf format), please refer to the TMA paper (in pdf format) as the source of truth.

Dataset Information

Important Note: Please use the dataset located in the folder provided with the notebook. There is no need to download it from the URL, as the dataset may change over time and affect your results. Additionally, the provided dataset includes minor modifications from the original source for the purposes of this notebook.

The dataset is DiaKG - https://tianchi.aliyun.com/dataset/88836 consists of JSON files representing a **Chinese Diabetes Knowledge Graph** containing structured medical text data with detailed entity recognition and relationship extraction. Each JSON file contains the following hierarchical structure:

Structure Overview

- **Document-based organization**: Each JSON file represents a medical document with a unique doc_id
- **Hierarchical text structure**: Documents → Paragraphs → Sentences
- **Rich annotation layers**: Entity recognition and relationship extraction

Key Components

1. **Document Level:**
    - doc_id: Unique identifier for each document
    - paragraphs: Array of paragraph objects
2. **Paragraph Level:**
    - paragraph_id: Sequential identifier within the document
    - paragraph: Full text content of the paragraph
    - sentences: Array of sentence objects with detailed annotations
3. **Sentence Level:**
    - sentence_id: Sequential identifier within the paragraph
    - sentence: Full sentence text
    - start_idx & end_idx: Character position indices within the paragraph
    - entities: Array of named entities found in the sentence
    - relations: Array of relationships between entities
4. **Entity Annotations:**
    - entity_id: Unique identifier (e.g., 'T0', 'T1')
    - entity: The actual text span of the entity
    - entity_type: Medical category (Disease, Drug, Test_items, Test_Value, Class, etc.)
    - start_idx & end_idx: Character positions within the sentence
5. **Relationship Annotations:**
    - relation_type: Type of relationship (e.g., 'Drug_Disease', 'Test_items_Disease', 'Class_Disease')
    - relation_id: Unique identifier (e.g., 'R0', 'R1')
    - head_entity_id & tail_entity_id: Links to related entities

You will need to explore and understand the structure and content of this dataset to perform meaningful analysis.

Question 1 (42 marks)

Objectives:

- Understand dataset with data scientist mindset
- Understand and design computation logic and routines in Python
- Assess use of Python only and Python data structures to perform extract, load, and transformation operations
- Assess use of Pandas dataframe to perform extract, load, transformation and calculation operations
- Structure code in appropriate methods (functions), looping and conditions
- Design methods to perform extract, load, transformation and calculation operations on dataset
- Conduct visualization in an appropriate way

(a) Data Loading and Entity Extraction

Task: Load all JSON files from the data/diakg/ directory and create a comprehensive DataFrame called entities_df that flattens the hierarchical structure to extract all entity information.

Requirements:

1. **File Loading**: Successfully load all JSON files from the data/diakg/ directory using appropriate Python methods
2. **Data Extraction**: Extract entity information from the nested JSON structure, ensuring all levels (document → paragraph → sentence → entities) are properly traversed
3. **DataFrame Creation**: Create a DataFrame called entities_df with exactly these columns:
   - doc_id: Document identifier
   - paragraph_id: Paragraph identifier within the document
   - sentence_id: Sentence identifier within the paragraph
   - entity_id: Entity identifier (e.g., 'T0', 'T1')
   - entity: The actual text of the entity
   - entity_type: The type/category of the entity
   - start_idx: Start character position within the sentence
   - end_idx: End character position within the sentence

(6 marks)

(b) Entity Type Analysis and Statistics
Task: Analyze the distribution and characteristics of different entity types in the dataset by creating a comprehensive statistical summary.

Requirements:

1. **Entity Type Identification**: Identify all unique entity types present in the entities_df DataFrame
2. **Statistical Analysis**: For each entity type, calculate:
   - One example entity (any representative entity of that type)
   - Total count of entities for that type
   - Average character length of entity values (the actual entity text) for that type (rounded to 2 decimal places)
3. **DataFrame Creation**: Create a new DataFrame called entity_stats_df with exactly these columns:
   - entity_type: The type/category of entity
   - example_entity: One example entity of that type
   - entity_count: Total number of entities of that type
   - avg_char_length: Average character length (rounded to 2 decimal places)
4. **Sorting**: Sort the DataFrame by entity_count in descending order (highest count first)

(6 marks)

(c) Relation Extraction and Analysis
Task: Extract all relationship information from the DiaKG dataset and create a DataFrame that captures the connections between entities across the medical knowledge graph.

Requirements:

1. **Relation Extraction**: Extract all relation information from the nested JSON structure, ensuring proper traversal through all levels (document → paragraph → sentence → relations)
2. **DataFrame Creation**: Create a DataFrame called relations_df with exactly these columns:
   - doc_id: Document identifier
   - paragraph_id: Paragraph identifier within the document
   - sentence_id: Sentence identifier within the paragraph
   - sentence: The actual text content of the sentence
   - relation_id: Relation identifier (e.g., 'R0', 'R1')
   - relation_type: Type of relationship (e.g., 'Drug_Disease', 'Test_items_Disease')
   - head_entity_id: Entity identifier for the head/source entity
   - tail_entity_id: Entity identifier for the tail/target entity

(3 marks)

(d) Sentence Distance Analysis

Task: Analyze the distance between entity pairs within sentences by calculating how many sub-sentences separate head and tail entities in each relation.

Note: In this context, a **sub-sentence** refers to text segments created when splitting a sentence by the Chinese period character '。'.

Requirements:

1. **Distance Calculation**: Calculate sentence distance between head and tail entities, where:
    - Distance = 0: Both head and tail entities are in the same sub-sentence (when split by Chinese period character '。')
    - Distance = 1: Entities are in adjacent sub-sentences
    - Distance = n: Entities are separated by n sub-sentences
2. **DataFrame Enhancement**: Add a new column called sentence distance to relations_df containing the calculated distances
3. **Analysis**: Identify and display all relations with the highest sentence_distance value

(6 marks)

(e) Paragraph Sentence Count Analysis with Entity Distribution

Task: Analyze the distribution of sentences across paragraphs in the DiaKG dataset by counting sentences with entities and without entities separately for each paragraph.

Note: This refers to actual sentences as they appear in the dataset structure (not sub-sentences created by splitting on ' ' as in Question 1.4).

Requirements:

1. **Sentence Classification**: For each paragraph in the dataset, classify sentences into two categories:
    - Sentences **with entities**: Sentences that contain at least one entity.
    - Sentences **without entities**: Sentences that contain no entities.
2. **DataFrame Creation**: Create a DataFrame called paragraph_sentence_counts with exactly these columns:
    - doc_id: Document identifier
    - paragraph_id: Paragraph identifier within the document
    - sentences_with_entities: Count of sentences containing at least one entity
    - sentences_without_entities: Count of sentences containing no entities
    - total_sentences: Total number of sentences in that paragraph
3. **DataFrame Sorting**: Sort the DataFrame by sentences_with_entities in descending order (highest count first)
4. **Statistical Summary**: Calculate and display the following sentence distribution statistics (rounded to 2 decimal places):
    - Average sentences with entities per paragraph
    - Average sentences without entities per paragraph

    - Average total sentences per paragraph
5. **Visualization**: Create a box plot using matplotlib/seaborn to visualize the distribution:
    - X-axis: Two categories ('With Entities', 'Without Entities')
    - Y-axis: Sentence counts per paragraph
    - Show the distribution of sentence counts across all paragraphs for both categories
6. **Analysis and Insight**: Based on the box plot visualization and statistical analysis, provide one clear insight about the distribution patterns between sentences with entities versus sentences without entities across paragraphs

(7 marks)

(f) Duplicate Entity Removal and Dataset Cleaning

Task: Identify and remove duplicate entities within the same sentence that have identical properties but are not referenced by any relations, creating a cleaned version of the entities dataset.

Definition: Duplicate entities are entities that exist in the same sentence and have:

- Same entity text value
- Same entity_type
- Same start_idx position
- Same end_idx position
- But different entity_id values

Requirements:

1. **Duplicate Detection**: Identify all groups of duplicate entities within the same sentence (same doc_id, paragraph_id, sentence_id)
2. **Relation Dependency Analysis**: For each group of duplicates, determine which entities are referenced by relations in relations_df (used as head_entity_id or tail_entity_id)
3. **Safe Removal Strategy**: For each duplicate group:
    - Keep all entities that are referenced by relations
    - If no entities in the group are referenced by relations, keep only the first entity (by entity_id alphabetical order)
    - If some entities are referenced by relations, remove only the unreferenced duplicates
4. **DataFrame Creation**: Create a cleaned DataFrame called entities_cleaned_df with the same structure as entities_df but with duplicate entities removed

(5 marks)

(g) Nested Entity Detection and Analysis

ask: Identify entities that are nested within other entities (i.e., one entity's text span is completely contained within another entity's text span in the same sentence) and analyze these nested patterns.

Definition: An entity A is considered "nested within" entity B if:

- Both entities are in the same sentence (same `doc_id`, `paragraph_id`, `sentence_id`)
- Entity A's start position is greater than or equal to entity B's start position
- Entity A's end position is less than or equal to entity B's end position
- Entity A and entity B are different entities (different `entity_id`).

Note: Use `entities_cleaned_df`, which contains the non-duplicate entities cleaned in Question 1(f).

Requirements:

1. **Nested Entity Detection**: Identify all pairs of entities where one entity is nested within another
2. **DataFrame Creation**: Create a DataFrame called `nested_entities_df` with exactly these columns:
    - `doc_id`: Document identifier
    - `paragraph_id`: Paragraph identifier
    - `sentence_id`: Sentence identifier
    - `outer_entity_id`: Entity ID of the containing (outer) entity
    - `outer_entity`: Text of the outer entity
    - `outer_entity_type`: Type of the outer entity
    - `outer_entity_start_idx`: Start character position of the outer entity
    - `outer_entity_end_idx`: End character position of the outer entity
    - `inner_entity_id`: Entity ID of the nested (inner) entity
    - `inner_entity`: Text of the inner entity
    - `inner_entity_type`: Type of the inner entity
    - `inner_entity_start_idx`: Start character position of the inner entity
    - `inner_entity_end_idx`: End character position of the inner entity
3. **Pattern Analysis**: Create a summary DataFrame called `nesting_patterns_df` that shows:
    - `outer_type`: Type of the outer entity
    - `inner_type`: Type of the inner entity
    - `pattern_count`: Number of unique nesting patterns (remove duplicates by outer_type, inner_type, outer_entity, inner_entity before counting)
    - `example_outer_entity`: One example of the outer entity for this pattern
    - `example_inner_entity`: One example of the inner entity for this pattern
4. **Sorting**: Sort `nesting_patterns_df` by `pattern_count` in descending order
5. **Visualization**: Create a horizontal bar plot showing all nesting patterns (outer_type → inner_type combinations)
6. **Analysis and Insight**: Based on the visualization, provide one clear insight about the nesting patterns observed in the medical text

(9 marks)

Question 2 (41 marks)

Objectives:

- Understand dataset with data scientist mindset
- Design computation logic and routines in Python
- Conduct visualization in an appropriate way
- Assess use of Pandas dataframe to perform extract, load, transformation and calculation operations
- Design methods to perform extract, load, transformation and calculation operations on dataset
- Assess the design and use of database ORM / SQLite methods to perform extract, load, transformation and calculation operations

(a) NetworkX Relation Type Visualization

Task: Create a NetworkX - https://networkx.org/documentation/stable/tutorial.html graph to visualize all relation types in the DiaKG dataset by representing each relation type as connections between entity types.

Note: This question focuses on visualizing unique relation types (e.g., 'Drug_Disease', 'Class_Disease'), not individual relation instances. Each relation type should appear only once in the graph as an edge between corresponding entity type nodes.

Requirements:

1. **Graph Construction**: Create a **directed** graph where:
   - Each unique entity type becomes a node (e.g., 'Disease', 'Drug', 'Class')
   - Each relation type creates directed edges between corresponding entity type nodes
   - For example: relation_type = 'Class_Disease' creates an edge from 'Class' node to 'Disease' node
2. **Graph Visualization**: Create a clear visualization showing:
   - All entity type nodes with appropriate labels
   - Directed edges representing relation types
   - Node positions that make the graph readable
3. **Analysis and Insight**: Based on the graph visualization, provide one clear insight about the relationship patterns between different entity types in the medical knowledge graph

(5 marks)

(b) Entity Type Comparison and Analysis

Task: Compare the unique entity types found in entities_cleaned_df with the entity types parsed from relation types in Question 2.1 to identify any discrepancies and analyze the completeness of the graph representation.

Requirements:

1. **Comparison Analysis**: Create a comparison summary that includes:
    - Entity types that exist in the data but not in the graph (missing from graph) in Question 2.1
    - Entity types that exist in the graph but not in the data (missing from data)
    - Entity types that exist in both (intersection)
2. **Relation Type Analysis for Data-Only Entity Types**: Write a function called find_entity_relations() that takes an entity type as parameter and returns a list of relation types where this entity type appears as either head or tail entities in relations_df.
3. **Entity Type Correction**: For each discrepancy found in part 2, correct the entity types in entities_cleaned_df to be consistent with the relation types they appear in. Update entities_cleaned_df by replacing inconsistent entity types with the correct entity types derived from the relation instances they participate in. Return the unique entity IDs whose type has been corrected.
4. **Analysis and Insight**: Based on the entity type comparison and correction process, provide one clear insight about which entity types do not have any connections (relations) to other nodes and the implications for knowledge graph completeness.

(6 marks)

(c) Complete Knowledge Graph Visualization

Task: Create a comprehensive NetworkX visualization that represents the complete knowledge graph structure using all entities from entities_cleaned_df and all relations from relations_df.

Requirements:

1. **Graph Construction**: Create a directed graph nx.DiGraph where:
    - Each unique entity (identified by entity text, not entity ID) becomes a node
    - Each relation creates a directed edge between corresponding entity nodes
    - Since an entity can be annotated with different types depending on its context, assign a node attribute for each type, with its value equal to the number of times the entity appears with that type
        - For example, Disease = 2 indicates that the entity text was tagged as Disease twice.
2. **Visualization**:
    - A spring layout visualization, coloring each node based on the type in which it most frequently appears
    - Color each edge based on its edge type (relation type)
    - A legend showing the color mapping for both entity types and edge types

3. **Insight**: Based on the visualization, provide one noticeable observation

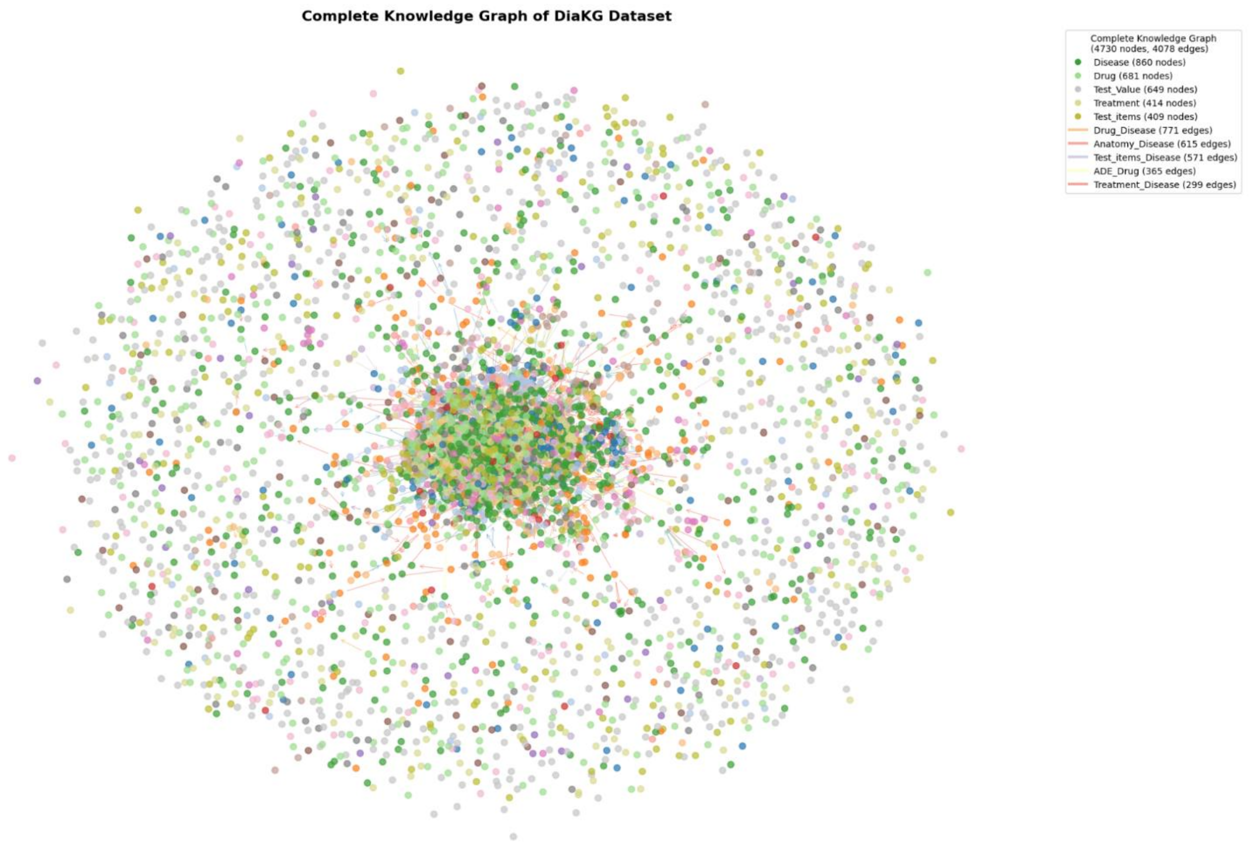

Figure 1: Sample Visualization for Q2(c)

(8 marks)

(d) Largest Cluster Visualization and Analysis

Task: Extract and visualize the largest connected component (cluster) from the complete knowledge graph to focus on the most densely connected part of the medical knowledge network.

Requirements:

1. **Subgraph Extraction**: Create a subgraph that includes only the nodes and edges from the largest connected component using nx.connected_components and complete_graph.subgraph
    - Convert to undirected graph to find connected components
2. **Subgraph Visualization**:
    - Node coloring based on the most frequent entity type for each node (same approach as Question 2.c)
    - Color each edge based on its edge type (relation type)
    - A legend showing the color mapping for both entity types and edge types

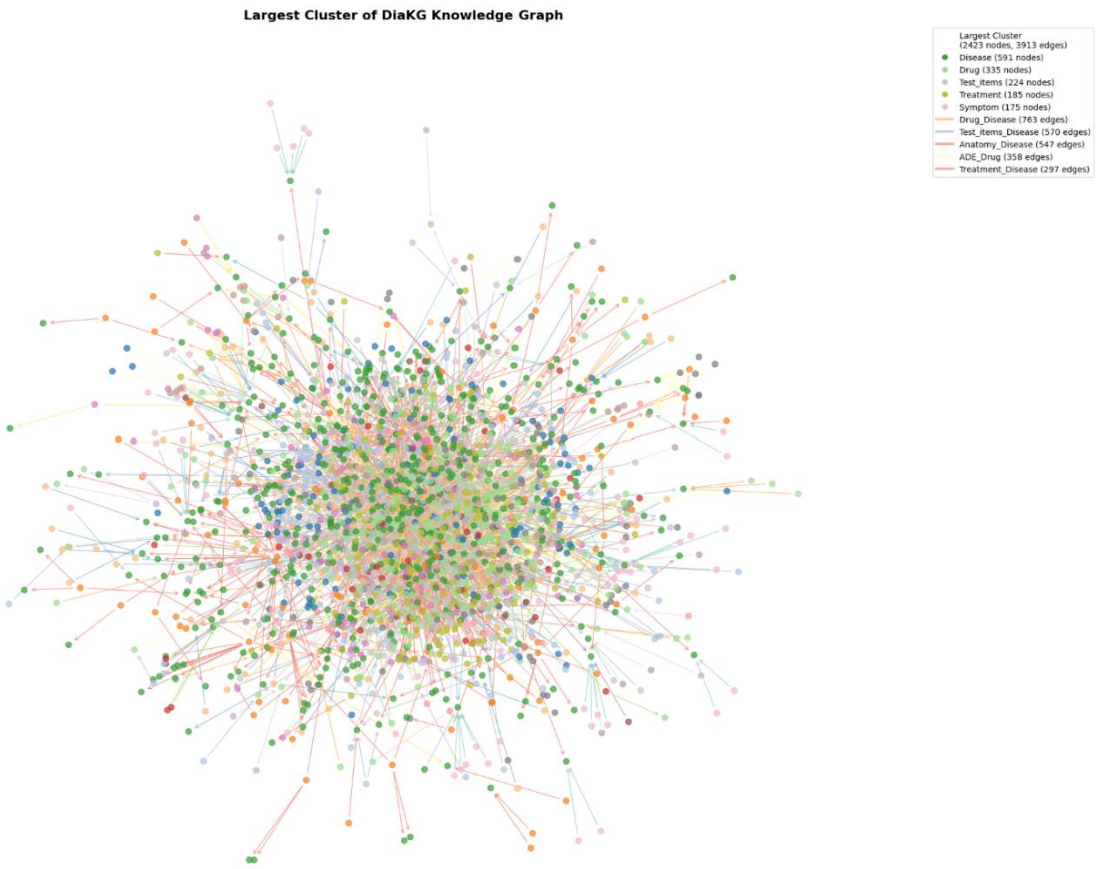

Figure 2: Sample visualization for Q2(d)

(2 marks)

(e) Basic Graph Statistics using NetworkX

Task: Compute and analyze fundamental graph statistics to understand the structural properties of the DiaKG knowledge graph.

Requirements:

1. **Degree Statistics**:

    - Calculate basic degree statistics including average degree, maximum degree, and minimum degree for both in-degree and out-degree of the largest connected component extracted from Question 2.d using `complete_graph.in_degree` and `complete_graph.out_degree`
    - In the context of the largest connected component, explain the following two situations:
        A. When the minimum in-degree is zero
        B. When the minimum out-degree is zero
2. **Centrality Measures**:

    - Compute degree centrality, betweenness centrality, and closeness centrality for the largest connected component using NetworkX methods:
        - `nx.degree_centrality()`
        - `nx.betweenness_centrality()`
        - `nx.closeness_centrality()`
    - Draw a clear observation specifically in the context of a diabetes knowledge graph

(5 marks)

(f) Google PageRank Algorithm Implementation and Visualization

Task: Implement the Google PageRank algorithm from scratch and use the PageRank scores to visualize the largest connected component with node sizes proportional to their importance.

Requirements:

1. **PageRank Implementation**: Implement one iteration of the PageRank algorithm using the power iteration method with the following formula:

    - $PR(A) = (1-d)/N + d * \Sigma(PR(T)/C(T))$
        - Where $d = 0.85$ (damping factor), N = number of nodes, T = nodes linking to A, C(T) = out-degree of T
        - Use `graph.predecessors(A)` to get all nodes linking to A
        - Replace `C(T)`with `N` if `C(T) = 0` to handle nodes with zero out-degree by distributing their PageRank equally to all nodes

2. **Visualization**:

    - Reuses the visualization code from Question 2.d
    - Set node sizes proportional to PageRank scores (multiply by 100000 for visibility)

3. **Analysis**:

    - Provide one insight about which types of entities have high PageRank scores and why this makes sense in the context of a diabetes knowledge graph

(4 marks)

(g) Laplacian Matrix for Node Positioning

Task: Use the Laplacian matrix (https://en.wikipedia.org/wiki/Laplacian_matrix) of the largest connected component to compute node positions for visualization based on the graph's structural properties.

Requirements:

1. **Laplacian Matrix Computation**:
    - Convert the largest connected component to an undirected graph using `to_undirected()`
    - Compute the Laplacian matrix using NetworkX's `normalized_laplacian_matrix()` function
    - Convert the result to a dense matrix using `.todense()`
2. **Eigenvalue Decomposition**:
    - Use NumPy's `linalg.eigh()` to compute eigenvalues and eigenvectors of the Laplacian matrix
    - Sort eigenvalues in ascending order and get corresponding eigenvectors
    - Use the 2nd and 3rd smallest eigenvalues' eigenvectors (Fiedler vectors) as x and y coordinates
3. **Visualization**:
    - Create a position dictionary mapping each node to its (x, y) coordinates
    - Create a visualization using Laplacian-based positions for node placement
    - Uses the same node colors, node sizes, and edge colors as in the visualization in Question 2.6
4. **Analysis**:
    - Provide three insights about how the Laplacian positioning reflects the graph's structure

(11 marks)

Question 3 (17 marks)

Objectives:

- Understand dataset with data scientist mindset
- Design computation logic and routines in Python
- Conduct visualization in an appropriate way
- Assess use of Pandas dataframe to perform extract, load, transformation and calculation operations
- Design methods to perform extract, load, transformation and calculation operations on dataset
- Assess the design and use of database ORM / SQLite methods to perform extract, load, transformation and calculation operations

(a) Entity with Most Entity Types
Task: Identify the entity text that appears with the most number of different entity types in the cleaned dataset. You should use both DataFrame operations and SQLite operations to solve this problem.

Note: Use the cleaned entities dataset (entities_cleaned_df) for this analysis, which has duplicate entities removed and ensures data quality.

Requirements:

1. **DataFrame Analysis**: Using pandas DataFrame operations, find the entity text that has the highest count of unique entity types and display the results with all associated entity types
2. **Database Operations**: Create an SQLite database, populate it with the cleaned entities data, and use SQL queries through SQLite to find the same result

(4 marks)

(b) Anatomy with Most Directly Connected Diseases
Task: Identify the anatomy entity that has the most direct connections to disease entities using the Anatomy_Disease relationship type. You should use both DataFrame operations and SQLite operations to solve this problem.

Requirements:

1. **DataFrame Analysis**: Using pandas DataFrame operations on the relations data, filter for 'Anatomy_Disease' relationship type and find the anatomy entity (head_entity) that connects to the highest number of unique disease entities (tail_entity)
2. **Database Operations**: Create an SQLite database, populate it with the relations data, and use SQL queries through SQLite to find the same result

3. Display the anatomy entity name, the count of connected diseases, and the list of connected diseases for both approaches

(4 marks)

(c) Triplet Pattern Analysis

Task: Analyze the knowledge graph to identify different types of triplet patterns using DataFrame operations. A triplet consists of three entities connected by two relationships, forming paths of length 2 in the knowledge graph.

Requirements:

1. Find all triplets with pattern **(entity > entity > entity)**: where the middle entity is the tail of the first relationship and the head of the second relationship
2. Find all triplets with pattern **(entity > entity < entity)**: where the middle entity is the tail of both relationships
3. Find all triplets with pattern **(entity < entity > entity)**: where the middle entity is the head of both relationships
4. **DataFrame Structure**: Create a results DataFrame for each pattern with the following columns: `['entity_1', 'entity_type_1', 'relation_1', 'entity_2', 'entity_type_2', 'relation_2', 'entity_3', 'entity_type_3']`
5. Explain what each of these patterns represents

Note: Use the `>` symbol to represent the direction from head_entity to tail_entity in a relationship, and `<` to represent the reverse direction. Ensure that the entity types match the relationship type. For example, if the relationship type is Reason-Disease, the head entity must be of type Reason, and the tail entity must be of type Disease.

Output: For each pattern type, display the count of unique triplets and show any random 5 examples with entity names, entity types, and relationship types in the specified DataFrame format.

(9 marks)

----- END OF ASSIGNMENT -----