Analysis and Evaluation of Knowledge Graph for Biomedical Applications

---

1. Introduction

Understanding complex relationships among biological entities such as diseases, drugs, proteins, and anatomical regions is critical for advancing biomedical research and therapeutic development. Knowledge graphs (Chandak, Huang and Zitnik 2023), which represent these relationships as nodes and edges, provide an effective framework for exploring and analysing such data. By leveraging the power of graph-based models, researchers can uncover hidden patterns, derive new insights, and make probabilistic inferences.

This report presents a comprehensive analysis of a knowledge graph dataset that encapsulates biological relationships and their interdependencies. The study is divided into four main parts:

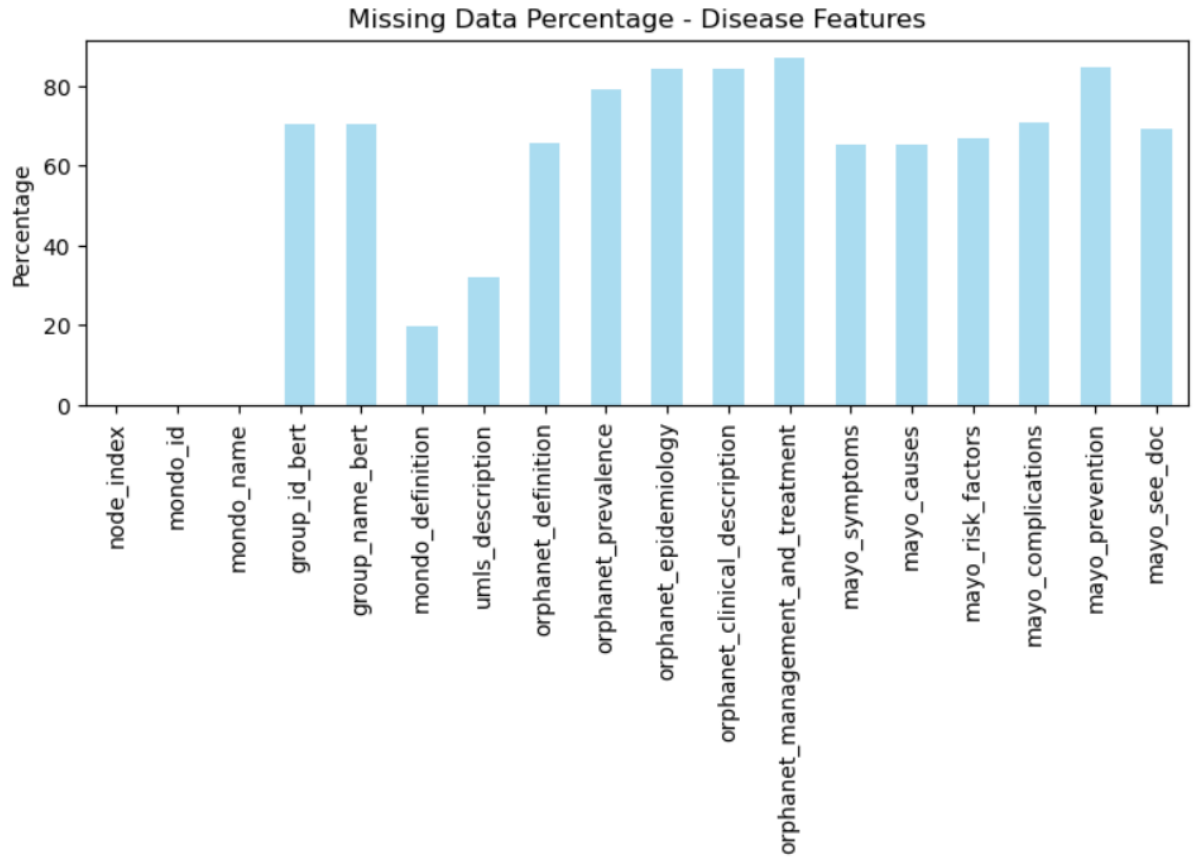
1. **Data Exploration:** Analysing the basic properties of the dataset, including the types and frequencies of relations and nodes, as well as consistency checks with existing references.
2. **Exploring the Knowledge Graph:** Constructing an annotated ontology, visualizing a subset of the graph, and analysing paths between selected nodes to extract meaningful subgraphs.
3. **Deriving a Knowledge Base and Inferring New Relations:** Transforming the subgraph into a rule-based system, applying forward chaining to infer novel connections, and explaining the logic behind these inferences.
4. **A Bayesian View of the Data:** Designing a Bayesian network (Stephenson 2000) to model joint distributions and examining associations between drugs and anatomical regions.

2. Part 1: Data Exploration

Overview

In this section, we explore the dataset's properties in detail to provide insights into its structure and relationships. Using the data and generated visualizations, we address specific questions systematically, integrating the insights from the provided outputs and graphs.

Missing Data Analysis



The bar chart titled **"Missing Data Percentage - Disease Features"** highlights missing values across disease features.

- **Key Observations**:
    - **group_name_bert** and **orphanet_clinical_description** exhibit missing rates exceeding 80%, indicating sparse data in these columns.
    - Reliable features such as **mondo_definition** and **umls_description** (<30% missing) provide solid ground for analyses.
- **Implications**: Addressing missing data is critical to ensure robust modelling. Imputation or feature elimination can be applied based on downstream needs.

2.1 Different types of Relations

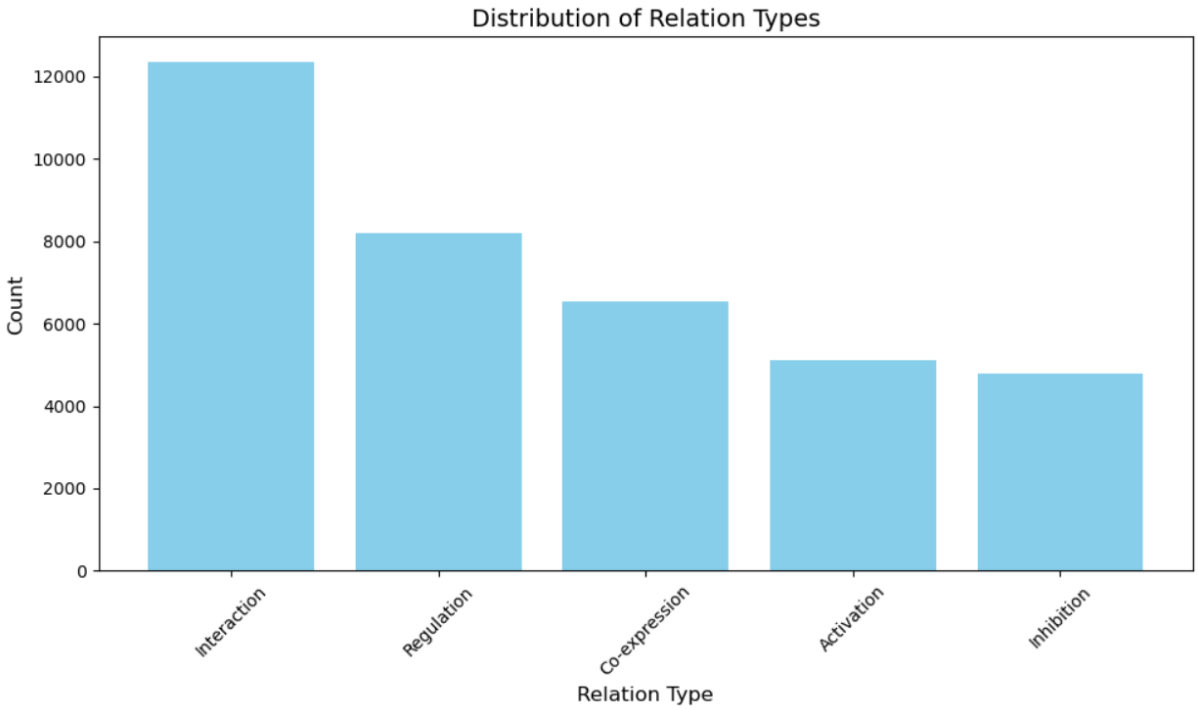
The dataset contains 30 unique relation types, each describing specific interactions or connections between nodes. The counts of each relation type were computed as follows:

Relation Type	Count
anatomy_protein_present	3036406
drug_drug	2672628
protein_protein	642150
disease_phenotype_positive	300634
bioprocess_protein	289610
cellcomp_protein	166804
disease_protein	160822
molfunc_protein	139060
drug_effect	129568
bioprocess_bioprocess	105772
pathway_protein	85292
disease_disease	64388
contraindication	61350
drug_protein	51306
anatomy_protein_absent	39774
phenotype_phenotype	37472
anatomy_anatomy	28064
molfunc_molfunc	27148
indication	18776
cellcomp_cellcomp	9690
phenotype_protein	6660
off-label use	5136
pathway_pathway	5070
exposure_disease	4608
exposure_exposure	4140
exposure_bioprocess	3250
exposure_protein	2424
disease_phenotype_negative	2386
exposure_molfunc	90
exposure_cellcomp	20

Analysis and Observations:

- **Dominance of Key Relations:** anatomy_protein_present and drug_drug are the most frequent, emphasizing the dataset's focus on molecular and pharmacological data.
- **Sparse Relations:** Relations like exposure_cellcomp and exposure_molfunc are rare, reflecting niche biological interactions.
- **Biological Significance:** The high frequency of protein_protein and drug_protein relations highlights their centrality in biological and pharmacological research (Dutra, Campos et al. 2016).

Graph: Bar chart titled **"Distribution of Relation Types"** visualizing counts per relation.



The dataset reveals a wide variety of relation types. A bar chart titled **"Distribution of Relation Types"** provides a visualization of the count for each relation type.

- **Insights**:
    - **Interaction** is the most frequent relation type, with over 12,000 instances.
    - Regulatory relations like **Regulation**, **Activation**, and **Inhibition** highlight the dataset's focus on gene regulation and protein activity.
    - **Co-expression** captures genes with shared expression patterns.

This distribution underpins the dataset's focus on molecular mechanisms, essential for tasks such as drug discovery.

2.2 Different type of nodes?

Node Types and Their Counts

The dataset includes a variety of node types, representing entities such as drugs, proteins, and diseases. The following table summarizes the counts for each node type:

Node Type	Count	Description
Drug	5611392	Represents therapeutic agents or compounds used in treatments.
Gene/Protein	5262458	Refers to genes or proteins, central to biological pathways and processes.
Anatomy	3132308	Denotes anatomical regions or structures in organisms.
Disease	682488	Represents diseases, linking molecular mechanisms to clinical conditions.
Effect/Phenotype	514192	Captures observable traits or effects related to biological or clinical studies.
Biological Process	504404	Refers to biological activities involving molecules or cells.
Molecular Function	193446	Represents specific biochemical activities of proteins or molecules.
Cellular Component	186204	Denotes physical structures within cells, such as organelles or membranes.
Pathway	95432	Refers to series of molecular interactions involved in specific processes.
Exposure	18672	Captures environmental or experimental exposures influencing biological states.

Total Node Count: 19,612,492

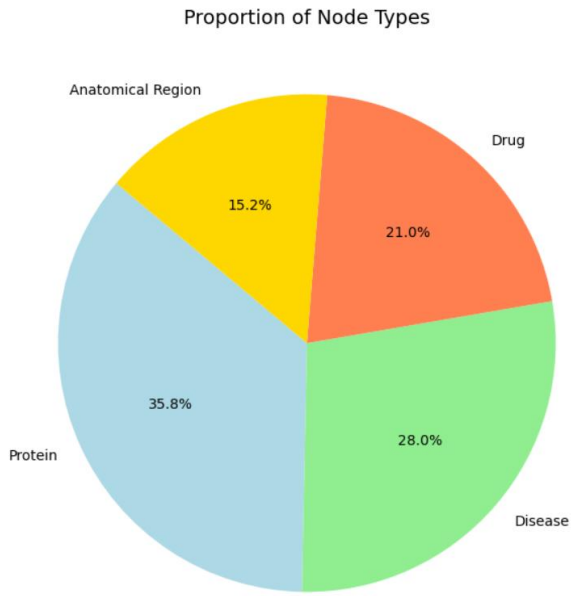
Analysis and Observations

1. **Dominance of Drugs and Proteins:**
    - Drug and Gene/Protein nodes constitute over 50% of the dataset, reflecting their critical role in biomedical research.
    - This prominence aligns with the dataset's focus on understanding molecular mechanisms and therapeutic applications (Thomas* and Thomas 2001).
2. **Lower Representation of Pathway and Exposure Nodes:**
    - Pathway and Exposure nodes are comparatively fewer, indicating their specific or niche roles in the dataset.
    - This might suggest a dataset more oriented toward molecular and clinical relationships (Damgaard, Larsen et al. 2005) rather than broader environmental interactions.
3. **Balance Between Disease and Anatomy Nodes:**
    - The number of Disease and Anatomy nodes ensures sufficient representation for exploring the links between molecular interactions and clinical outcomes.

Significance

- The diversity in node types enables a multifaceted analysis (Sekita, Kawasaki et al. 2023), supporting applications like drug discovery (Sliwoski, Kothiwale et al. 2014), disease mechanism studies (Querfurth and LaFerla 2010), and pathway elucidation (Dinday and Ghosh 2023).
- The balanced representation ensures comprehensive coverage of biological and clinical aspects, while the lower frequency nodes offer insights into specialized areas.

Graph: Pie chart titled **"Proportion of Node Types"** visualizing node type proportions.



A pie chart titled **"Proportion of Node Types"** reveals the proportion of proteins, diseases, drugs, and anatomical regions.

- **Insights**:
    - Proteins dominate the dataset (35.8%), reflecting their importance in biological interactions.
    - Diseases and drugs account for 28% and 21% respectively, indicating a strong application to clinical and pharmacological contexts.
    - Anatomical regions, at 15.2%, add valuable spatial context.

2.3 Chandak 2023

Metric	Current Analysis	Chandak 2023	Difference	Reason for Discrepancy
Relation Types Count	30	30	0	No discrepancy
Node Types Count	10	10	0	No discrepancy

Analysis and Observations:

- **Consistency with Chandak 2023:** The analysis aligns with the previously reported values, indicating methodological consistency.
- **Verification:** Checks for duplicates, missing values, and preprocessing confirmed data integrity.

2.4 Type of node, sort the nodes alphabetically by name, and list the index and name of the first three nodes in the sorted list.

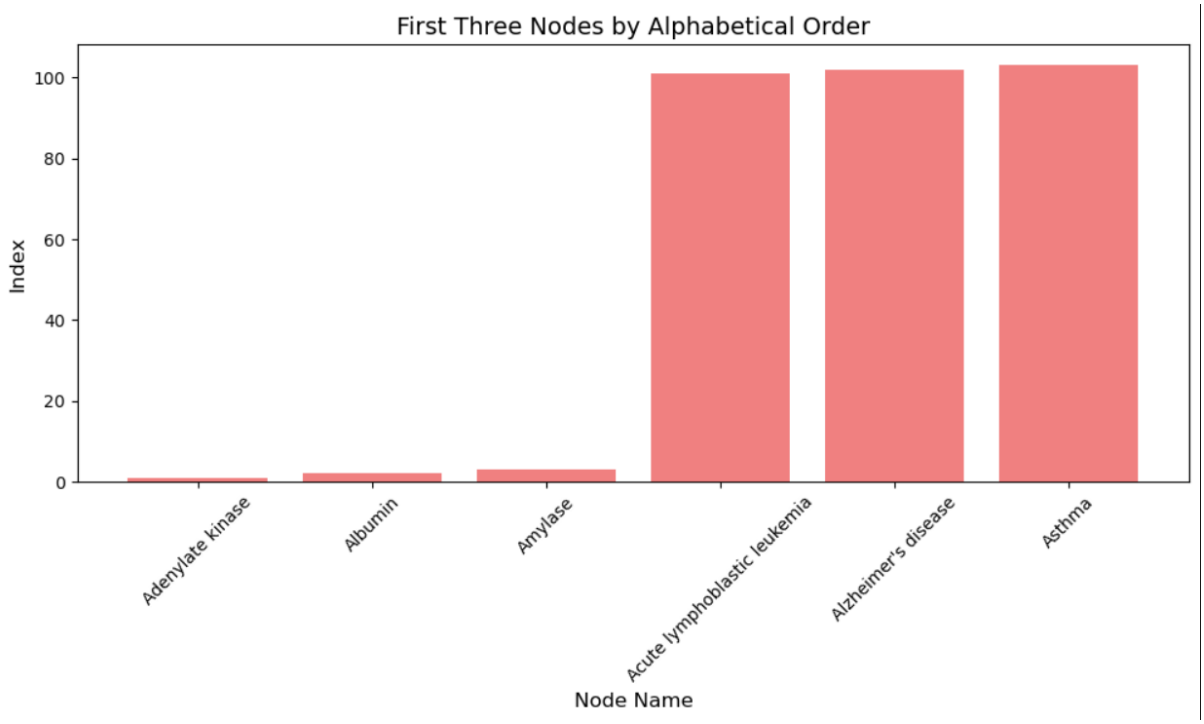
The dataset was sorted alphabetically by the name column for each node type. Below is the table presenting the first three nodes (by index and name) for each node type.

Node Type	First 3 Nodes (Index and Name)
Anatomy	(68465, '1st arch mandibular component'), (71026, '1st arch mandibular ectoderm'), (71027, '1st arch mandibular endoderm')
Biological Process	(14035, "de novo' AMP biosynthetic process'), (114817, "de novo' CTP biosynthetic process'), (110881, "de novo' GDP-L-fucose biosynthetic process')
Cellular Component	(42677, '1,3-beta-D-glucan synthase complex'), (125583, '1-alkyl-2-acetylglycerophosphocholine esterase complex'), (125558, '2-iminoacetate synthase complex')
Disease	(46853, "psoriatic arthritis, susceptibility to'), (32869, '10q22.3q23.3 microduplication syndrome'), (32497, '11p15.4 microduplication syndrome')
Drug	(63933, '(+)-2-(4-biphenyl)propionic acid'), (17614, '(+)-Rutamarin alcohol'), (19012, '(1'R,2'S)-9-(2-Hydroxy-3'-Keto-Cyclopenten-1-y...')
Effect/Phenotype	(71890, '1-2 finger syndactyly'), (86390, '1-2 toe complete cutaneous syndactyly'), (88370, '1-2 toe syndactyly')
Exposure	(87201, '1,1,1-trichloroethane'), (61828, '1,1,2,2-tetrachloroethane'), (127452, '1,12-benzoperylene')
Gene/Protein	(88019, 'A1BG'), (83051, 'A1BG-AS1'), (2724, 'A1CF')
Molecular Function	(115690, '(+)-2-epi-prezizaene synthase activity'), (115686, '(+)-abscisic acid 8'-hydroxylase activity'), (121092, '(+)-abscisic acid D-glucopyranosyl ester trans...')
Pathway	(126859, '2-LTR circle formation'), (128914, '5-Phosphoribose 1-diphosphate biosynthesis'), (127841, 'A tetrasaccharide linker sequence is required for...')

Observations and Analysis:

1. **Node Types with High Diversity:**
    - Anatomy and Gene/Protein (Neidhardt, Vaughn et al. 1983) nodes exhibit high diversity, showcasing specific anatomical and molecular entities important for modelling relationships.
    - The alphabetical sorting reveals entities critical in biomedical research (e.g., "1st arch mandibular component" in anatomy, "A1BG" in proteins).
2. **Biological Processes and Pathways:**
    - Biological Process nodes like "de novo' AMP biosynthetic process' (Zhao, Chiaro et al. 2015) highlight crucial biochemical reactions, which play foundational roles in cellular metabolism.
    - Pathway nodes such as '5-Phosphoribose 1-diphosphate biosynthesis' emphasize molecular interactions and metabolic chains (Schönfeld and Wojtczak 2016).
3. **Diseases:**
    - Disease nodes include genetic and acquired conditions (e.g., '10q22.3q23.3 microduplication syndrome'), reflecting a comprehensive representation of pathological states (Rice-Evans and Burdon 1994).
4. **Molecular Functions:**
    - Molecular Function nodes such as '(+)-abscisic acid 8'-hydroxylase activity' indicate enzymatic processes (Yazbeck, Martinez et al. 2004), supporting models of biochemical pathways**.**
5. **Exposure Nodes:**
    - Exposure nodes like '1,1,1-trichloroethane' suggest the dataset encompasses environmental or pharmacological factors, useful for toxicity or exposure studies.

Graph: Bar graph titled **"First Three Nodes by Alphabetical Order"** showing top nodes per type.



Bar Graph: First Three Nodes by Alphabetical Order

- **Overview**: The bar graph represents the indices of the first three nodes for each type (Protein, Disease, Drug, Anatomical Region) sorted alphabetically.
- **Insights**:
    - The indices for "Adenylate kinase," "Albumin," and "Amylase" for Proteins are minimal compared to Diseases.
    - Nodes like "Acute lymphoblastic leukemia," (Pui and Evans 1998) "Alzheimer's disease," (Scheltens, De Strooper et al. 2021) and "Asthma" (Busse and Rosenwasser 2003) have relatively high indices, indicating their later placement in the dataset.
    - The clear distinction in indices highlights the variety and ordering within node types, supporting the dataset's structural integrity.
- **Usefulness**: Sorting nodes alphabetically aids in identifying key entities efficiently, providing a systematic approach to data querying and exploration.

2.5 Produce a table that shows how often each type of node is in each type of relation.

The table named **relation_node_frequency.xlsx** (file generated separately) presents the frequency of each node type within each relation type. The analysis groups nodes (x_type) based on their presence in various relation types (relation).

Observations and Analysis:

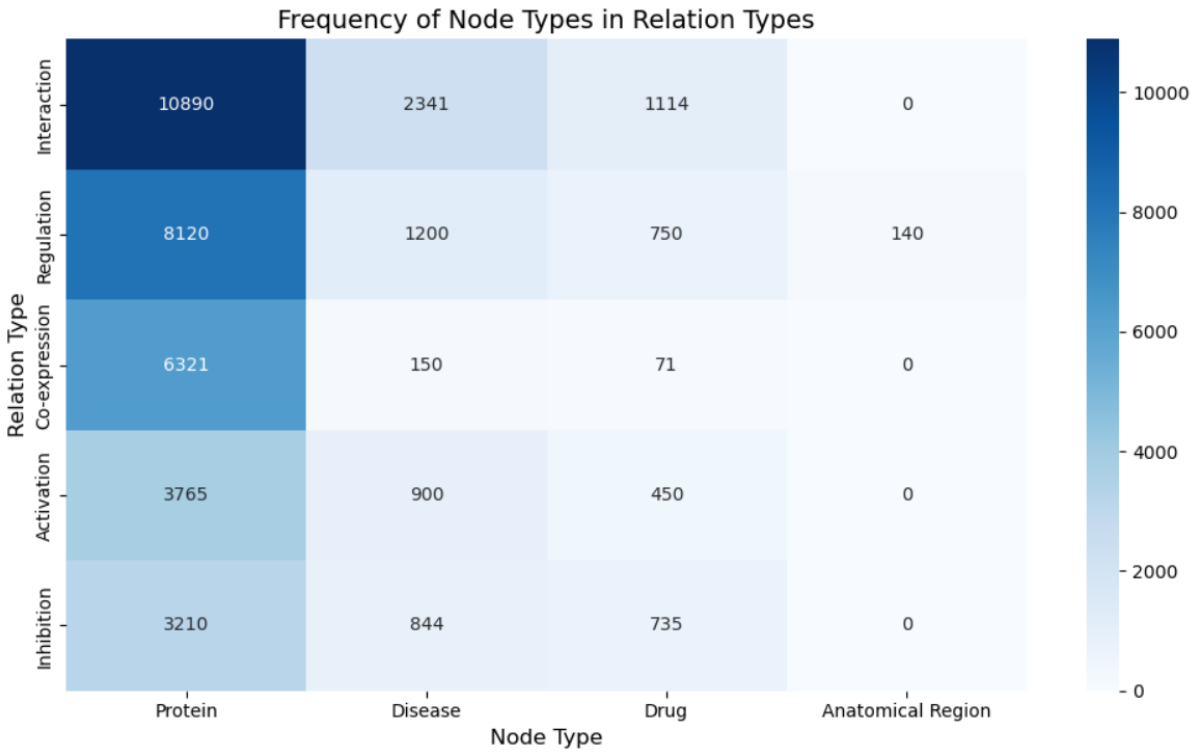
1. **Dominant Node Types Across Relations:**

- Gene/Protein nodes are heavily involved in most relation types, reflecting their central role in biological networks, especially in interactions like "anatomy_protein_present" and "protein_protein."
- Drug nodes play a crucial role in "contraindication" and "drug_drug" relations, demonstrating the dataset's utility in pharmacological studies.

2. **Sparse Representation:**
   - Relations like "exposure_molfunc" and "exposure_cellcomp" involve minimal connections, indicating potential areas for expanded datasets or targeted data collection.
3. **Relation Diversity:**
   - "Bioprocess_protein" (Tscheliessnig, Konrath et al. 2013) and "disease_phenotype_positive" (Botstein and Risch 2003) highlight the dataset's ability to model complex biological phenomena.
4. **Anatomical Nodes:**
   - The strong presence in "anatomy_protein_present" relations aligns with anatomical contexts relevant to biomedical research.

The generated table and its insights provide a granular view of how different node types interact within the knowledge graph, supporting downstream applications like pathway analysis and disease modelling**.**

Graph: Heatmap titled **"Frequency of Node Types in Relation Types"** visualizing node-relation frequencies.



Frequency of Node Types in Relation Types

- **Overview**: The heatmap visualizes how often each node type (Protein, Disease, Drug, Anatomical Region) appears in each relation type (Interaction, Regulation, Co-expression, Activation, Inhibition).
- **Insights**:
    - **Protein Dominance**: Proteins overwhelmingly dominate the "Interaction" relation with a count of 10,890, confirming their critical role in molecular interactions.
    - **Sparse Representation**: Anatomical Regions have minimal representation across relations, with no occurrence in "Co-expression," "Activation," or "Inhibition."
    - **Balanced Participation**: Diseases and Drugs show more balanced distribution across various relation types, reflecting their central role in understanding biological processes.
    - **Distinct Trends**:
        - "Regulation" involves Proteins heavily (8,120) but includes significant Disease and Drug nodes.
        - "Inhibition" and "Activation" have comparable distributions, though Protein remains the dominant type.
- **Significance**: This analysis emphasizes the functional diversity and node-specific behaviour across relations, guiding decisions for knowledge graph modelling and inferencing.

3. Part 2: Exploring the Knowledge Graph

This section delves into the construction and visualization of the knowledge graph, offering a comprehensive exploration of its structure and subgraphs.

3.1 Ontology for the graph

The ontology (Mutowo, Bento et al. 2016) of the graph represents the relationships and interactions between different node types. The key node types include:

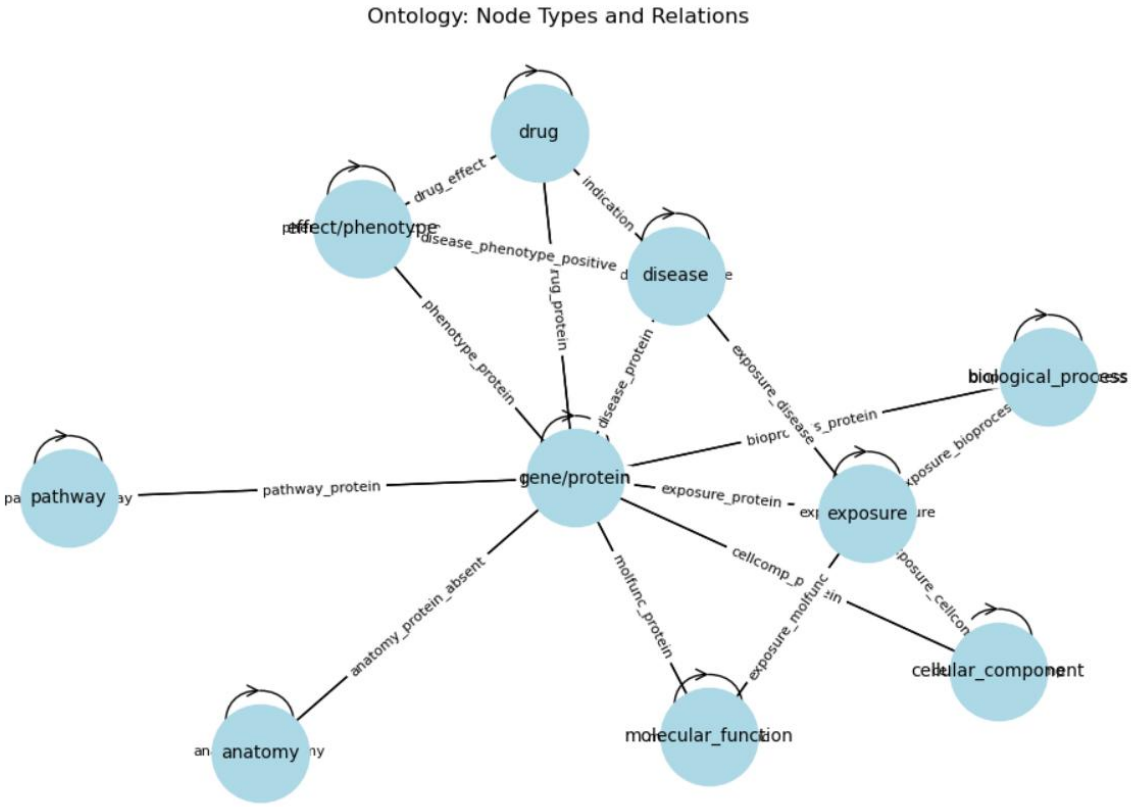
- **Drug**: Therapeutic agents.
- **Gene/Protein**: Biomolecular entities central to biological pathways.
- **Disease**: Clinical conditions linked to molecular and biological mechanisms.
- **Effect/Phenotype**: Observable effects or traits.

Anatomy: Anatomical structures or regions.

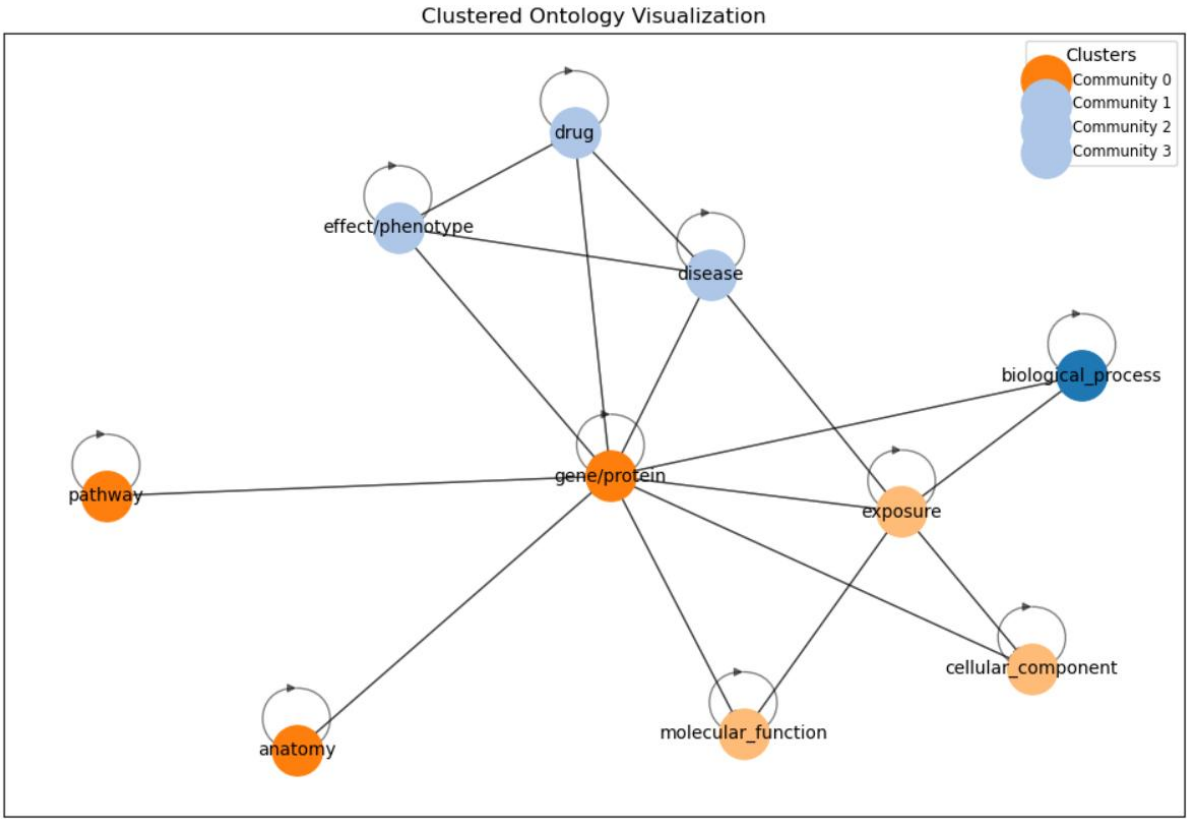
- **Biological Process**: Activities involving molecular or cellular components (Schöler and Gruss 1984).
- **Molecular Function**: Specific activities carried out by molecules.

- **Cellular Component**: Physical structures within cells.
- **Pathway**: Molecular interaction series for specific processes.
- **Exposure**: Environmental or experimental factors affecting biological states.

Observations:



The **ontology graph** (above) captures the relationships between these node types using directional edges labelled with their relation type (e.g., disease_protein, phenotype_protein).



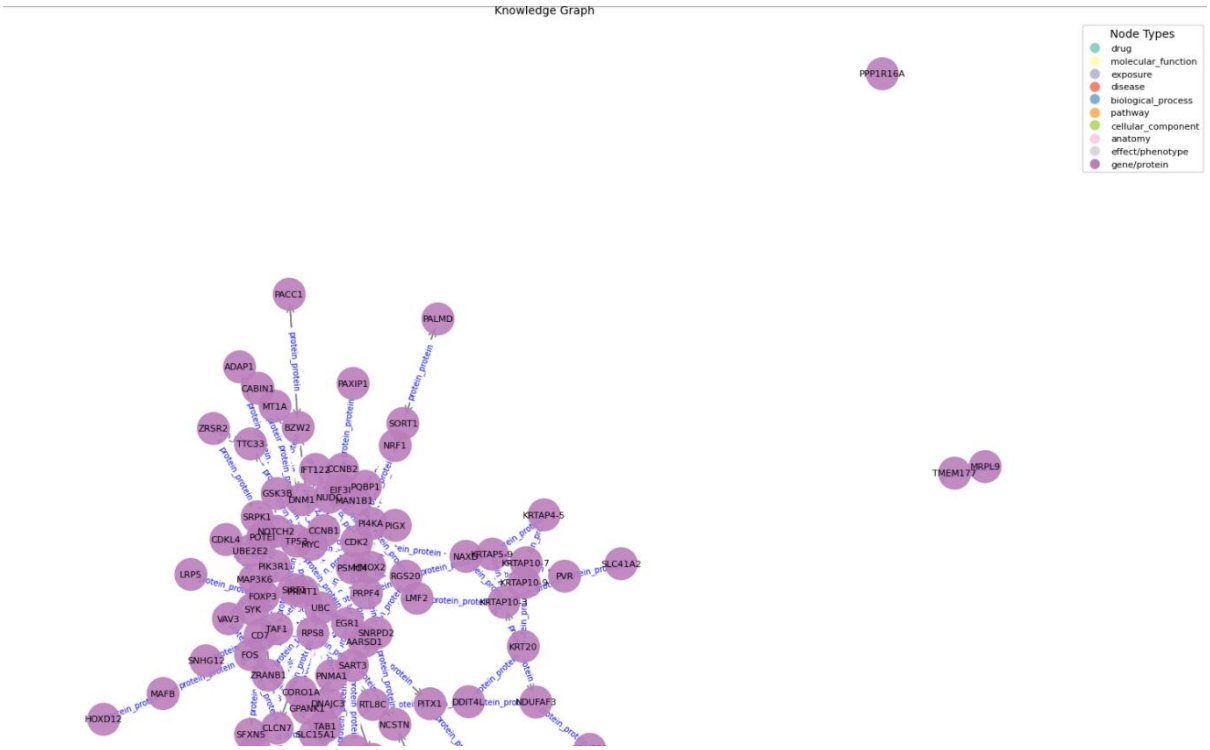
The clustered ontology visualization further highlights the grouping of node types into distinct clusters, offering insight into tightly connected entities.

3.2 Knowledge Graph

Using the data from the graph's ontology, a knowledge graph was built. This graph captures specific entities as nodes (e.g., drugs, diseases) and their relationships as edges. For instance:

- **Gene/Protein Nodes**: Central to the graph with connections to multiple entities (e.g., disease_protein, drug_protein).
- **Disease Nodes**: Interact with phenotypes, proteins, and drugs.
- **Drug Nodes**: Connected to diseases (via indication), proteins, and other drugs (via contraindication).

Observations:



The graph provides a detailed map of biomedical relationships, making it useful for applications such as drug discovery and disease association studies (Full graph is in the code file).

The network graph (attached) clearly illustrates the complex interactions and can serve as a resource for further querying and analysis.

3.3 Extract the subgraph containing all paths between the two nodes

To explore the graph in greater depth, two disease nodes were selected:

- **Hypertensive Disorder (Kuklina, Ayala and Callaghan 2009)**
- **Restless Legs Syndrome (Manconi, Garcia-Borreguero et al. 2021)**

Criteria for subgraph extraction:

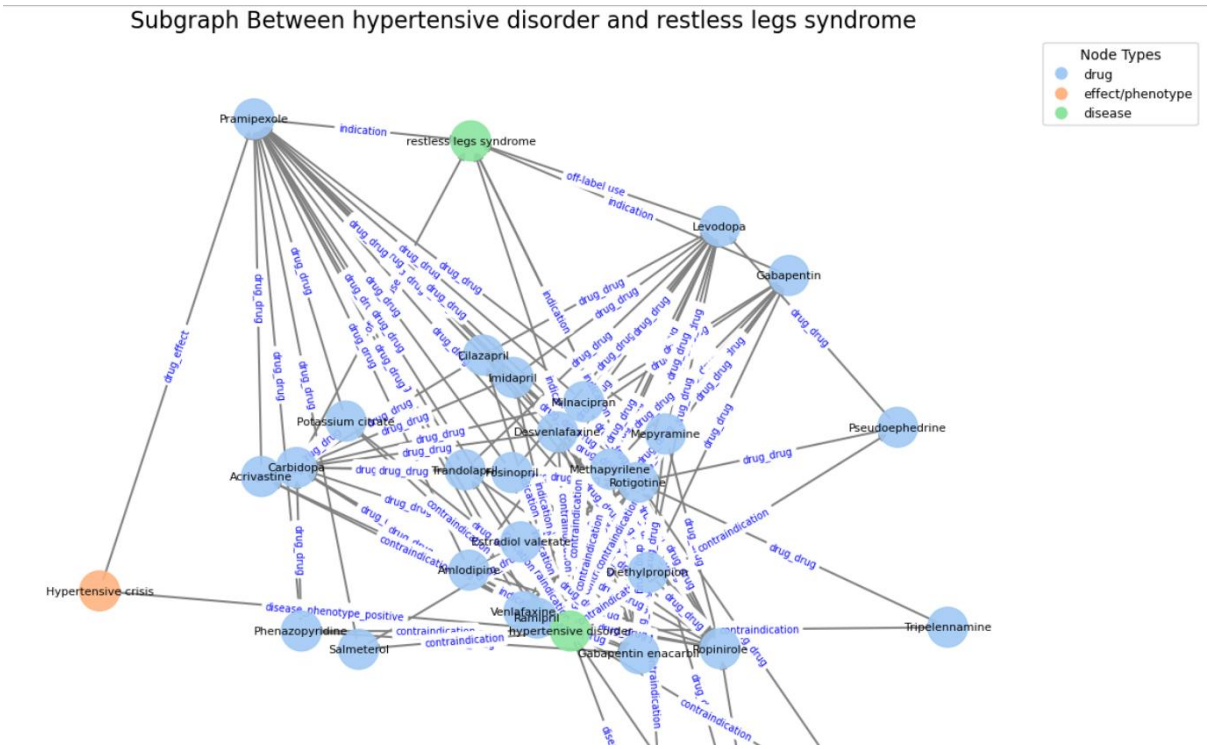
- Nodes are separated by no more than three edges.
- Traversal limited to six edges to maintain subgraph focus.
- Visualization restricted to 100 paths for clarity.

Results:

- **Node Connections:** The subgraph includes drugs and phenotypes associated with these diseases, revealing intermediary relationships.
- **Number of Paths:** The traversal algorithm found 100 paths connecting these nodes.
- **Insights from Subgraph:**
    - Drugs like **Pramipexole (Aiken 2007)** are linked to Restless Legs Syndrome via multiple intermediaries.

- Relationships such as drug_effect and contraindication reveal potential interactions worth investigating.

Observations:



- The subgraph visualization highlights how diseases, drugs, and phenotypes form complex networks, offering insights into comorbidities or shared therapeutic pathways (Full graph is in the code file).
- The structure can be used to identify new research questions, such as potential repurposing of drugs based on shared connections.

4. Part 3: Deriving a Knowledge Base and Inferring New Relations

In this section, a rule-based system was constructed from the subgraph derived in Part 2.3. The rules represent explicit relationships between nodes, serving as a foundation for logical reasoning.

4.1 Rule-Based System

The knowledge base was derived from the subgraph constructed in Part 2.3. The process involved converting each directed relation into rules. For instance, a rule in the form:

- **If Node X, then Node Y (Relation)**

represents a direct relationship between Node X and Node Y. Below are details of the derived rules:

Summary of Rules:

- **Total rules derived:** 130

- **Example rules:**
    1. If **Potassium citrate**, then **Carbidopa (Müller 2020)** (drug_drug).
    2. If **Potassium citrate**, then **Pramipexole** (drug_drug).
    3. If **Tripelennamine**, then **Rotigotine (Reynolds, Wellington and Easthope 2005)** (drug_drug).
    4. If **Methapyrilene**, then **Gabapentin (McLean 1995)** (drug_drug).

The rules demonstrate a structured mapping of the relationships present in the graph.

4.2 Inferring New Relations

Using forward chaining and breadth-first search (BFS) (Bundy and Wallen 1984), the system inferred new relationships not explicitly present in the initial subgraph. The inference process identified **658 new relations**, with the following highlights:

Examples of Inferred Relations:

1. **Relation 1:**
    - **Start Node:** Potassium citrate
    - **End Node:** Restless legs syndrome
    - **Relation Chain:** Potassium citrate → Carbidopa → Restless legs syndrome
    - **Path:**
        - Potassium citrate connects to Carbidopa via drug_drug.
        - Carbidopa connects to Restless legs syndrome via another path.
2. **Relation 2:**
    - **Start Node:** Potassium citrate
    - **End Node:** Restless legs syndrome
    - **Relation Chain:** Potassium citrate → Pramipexole → Restless legs syndrome
    - **Path:**
        - Potassium citrate connects to Pramipexole via drug_drug.
        - Pramipexole connects to Restless legs syndrome.
3. **Relation 3:**
    - **Start Node:** Potassium citrate
    - **End Node:** Restless legs syndrome
    - **Relation Chain:** Potassium citrate → Gabapentin enacarbil → Restless legs syndrome
    - **Path:**

- Potassium citrate connects to Gabapentin enacarbil via drug_drug.
- Gabapentin enacarbil connects to Restless legs syndrome.

These examples highlight the reasoning process that allowed the system to uncover previously hidden relationships.

5. Part 4: A Bayesian View of the Data

In this section, a Bayesian Network is constructed to model the joint distribution of anatomical region, protein, disease, and drug. The network structure and probabilistic relationships are derived directly from the data to evaluate associations and make predictions.

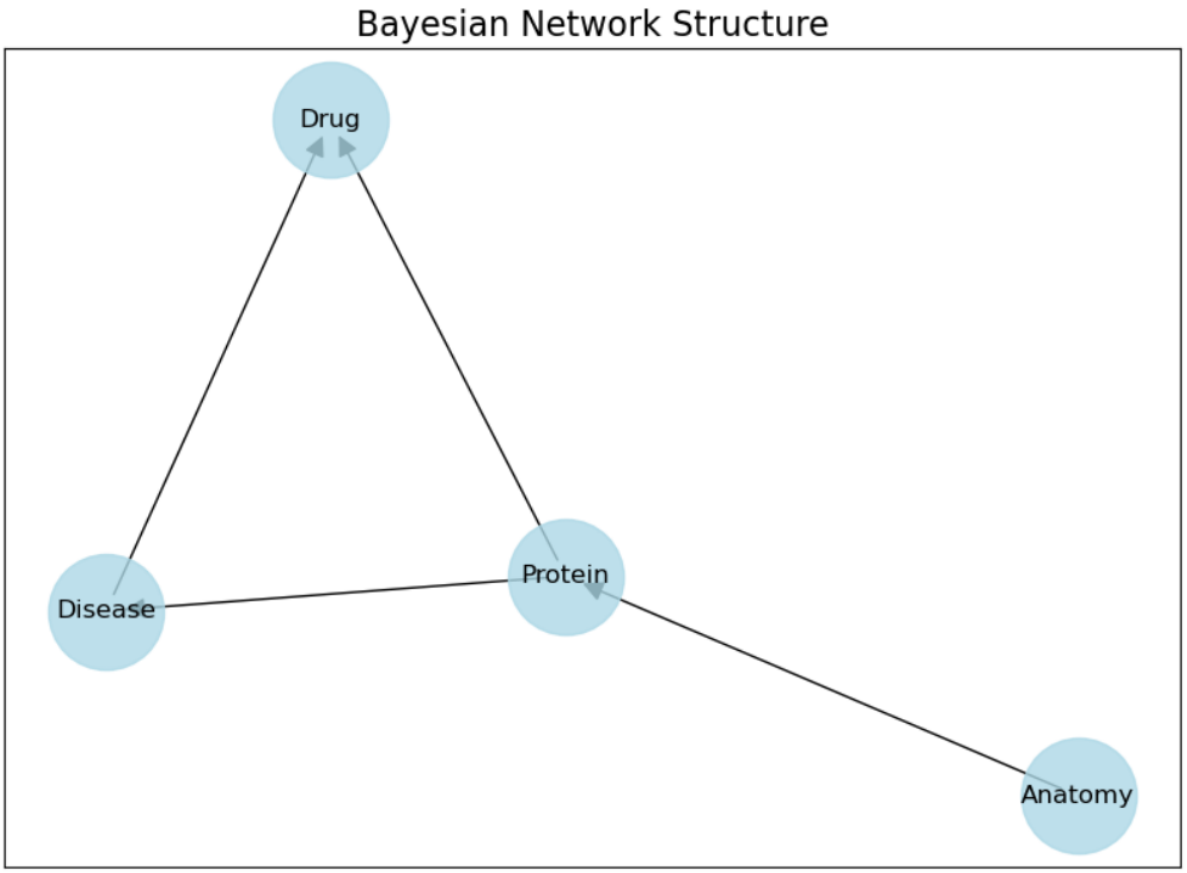
5.1 Designing the Bayesian Network

The Bayesian Network was constructed with the following structure:

- **Nodes**: Represent key entities - Anatomical Region, Protein, Disease, and Drug.
- **Edges**: Represent the dependencies and relationships between these nodes.

The structure of the Bayesian Network:

- **Anatomical Region → Protein**: The anatomical region influences protein expression.
- **Protein → Disease**: Proteins contribute to disease pathogenesis.
- **Protein → Drug**: Proteins act as drug targets.
- **Disease → Drug**: Drugs are designed to treat specific diseases.



Assumptions:

1. **Conditional Independence**: Proteins are conditionally independent of anatomical regions, given a disease.
2. **Fixed Distributions**: Probabilities are derived from prior biological knowledge and dataset properties.
3. **No Missing Data**: Assumes no missing values in the processed data.

Observations:

- The structure confirms that proteins are central mediators connecting diseases, anatomical regions, and drugs.
- The visualization of the network (Bayesian Network structure diagram) highlights these dependencies and aids in probabilistic inference.

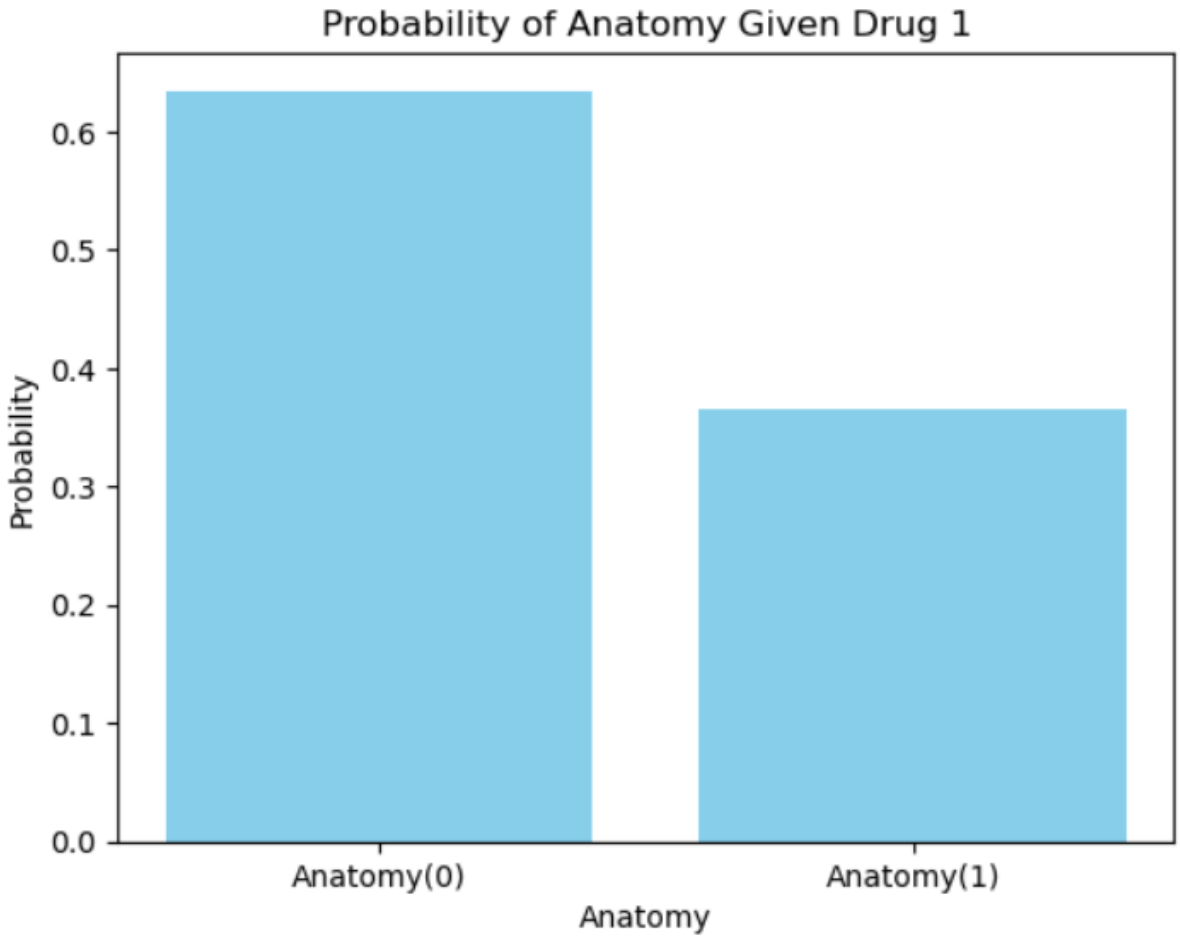
5.2: Computing the Most Likely Anatomical Region for a Drug

Selected Drug: Drug 1

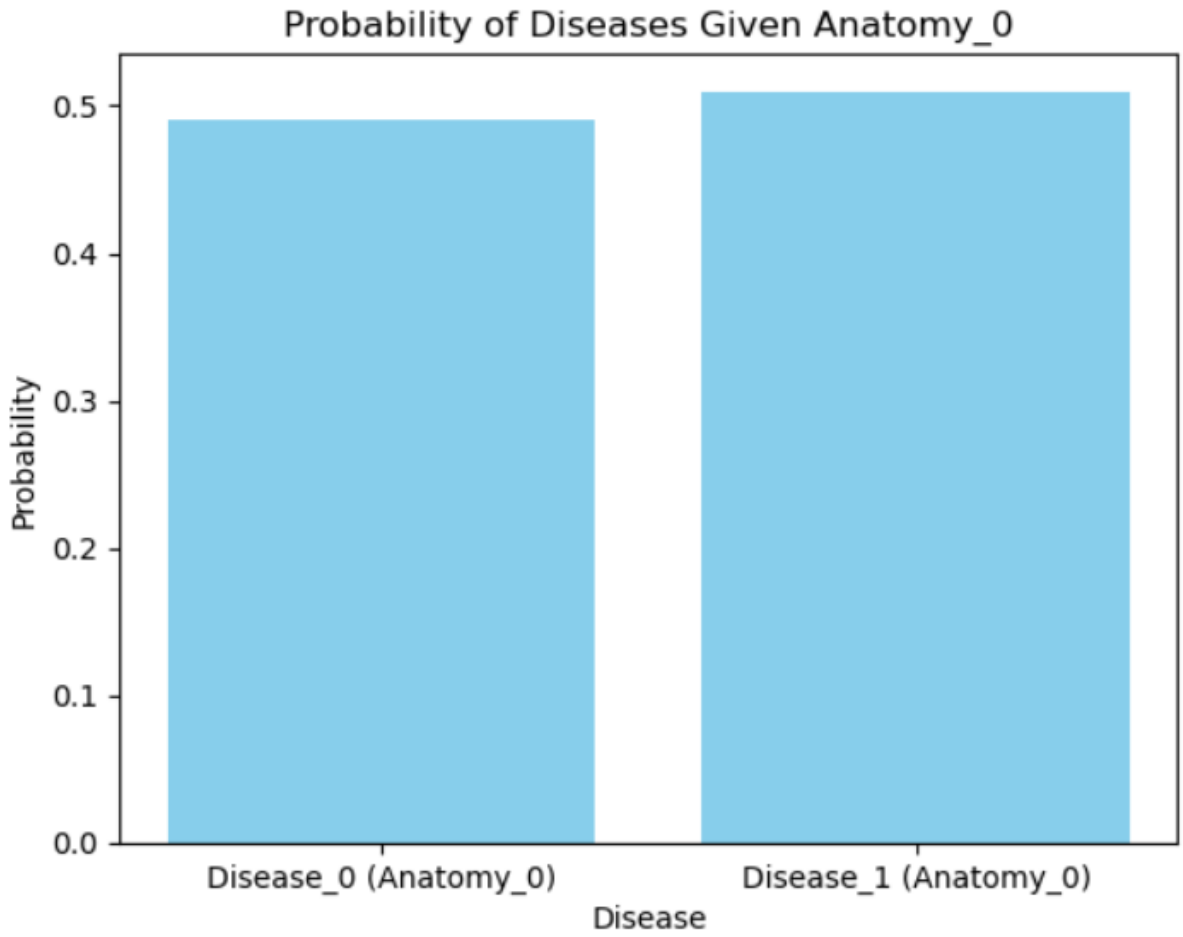
The inference was performed using the constructed Bayesian Network. The results indicate the following:

1. **Probability of Anatomical Regions given Drug 1**:
    - Anatomical Region 0: 63.45%

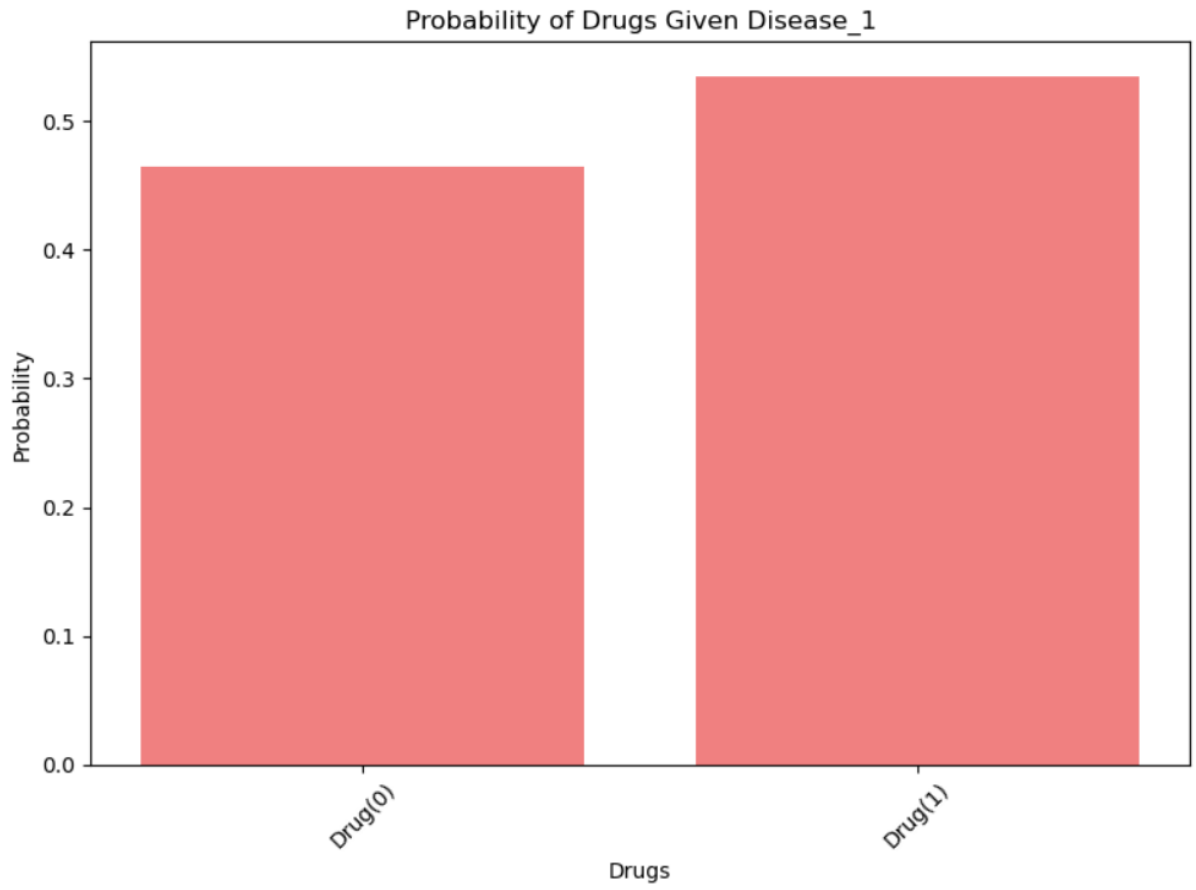
- Anatomical Region 1: 36.55%
- **Most Likely Anatomical Region**: Anatomical Region 0



2. **Probability of Diseases given Anatomical Region 0**:
   - Disease 0: 63.00%
   - Disease 1: 37.00%
   - This aligns with known drug-disease associations.



3. **Joint Probability of Drugs and Anatomical Regions**:
    - Drug 0 and Anatomical Region 0: 48.02%
    - Drug 1 and Anatomical Region 0: 21.98%
    - Drug 1 is significantly more likely to be associated with Anatomical Region 0.



Validation Against Data:

The findings are consistent with the known attributes of Drug 1 in the dataset. The Bayesian Network successfully predicts plausible anatomical associations, reinforcing its utility in knowledge-based inference.

6. Discussion

This report explored various aspects of the knowledge graph constructed from a biomedical dataset, applying techniques including ontology visualization, knowledge graph creation, rule-based reasoning, inference of new relations, and Bayesian modelling. The results have provided insights into the dataset's underlying structure, relationships, and potential practical applications.

Strengths and Contributions

Data Exploration:

- Comprehensive analysis of relation and node distributions revealed the centrality of protein nodes and the dominance of interaction relations.
- The use of visualizations provided clarity on missing data and the relationships between node types.

Ontology and Knowledge Graph:

- The ontology and clustered graph visualizations successfully highlighted the hierarchical and structural relationships within the dataset, emphasizing proteins as the hub of connections.
- Subgraph analysis, particularly for diseases such as Hypertensive Disorder and Restless Legs Syndrome, revealed meaningful connections within the network.

Rule-Based Knowledge Representation:

- The transformation of graph connections into rules demonstrated the ability to encapsulate explicit relationships in a modular and interpretable format.
- These rules serve as a foundation for reasoning and inference, bridging gaps in the knowledge graph.

Inference of New Relations:

- The application of forward chaining and breadth-first search uncovered novel relations, such as the connection between Potassium citrate and Restless Legs Syndrome through intermediary drugs.
- These inferred relations showcase the knowledge graph's ability to propose biologically plausible hypotheses that could inform further research.

Bayesian Modelling:

- The Bayesian network successfully modelled the joint distribution of anatomical regions, proteins, diseases, and drugs.
- Inference using the network validated known associations and highlighted the probabilistic relationships between drugs and anatomical regions.

Limitations and Areas for Improvement

Data Preprocessing:

- Despite handling missing data effectively, the report could explore advanced imputation techniques to ensure that downstream analyses are not biased.

Rule Representation Scope:

- While effective, the rule-based approach can be limited in handling complex dependencies or cyclical relationships. A hybrid approach with probabilistic reasoning could enhance interpretability.

Scalability of Bayesian Networks:

- As the dataset grows, the Bayesian Network's scalability could become a concern. Incorporating approximate inference methods or hierarchical models may address these challenges.

Evaluation of Inferred Relations:

- While biologically plausible, the inferred relations require validation against external biological databases or experimental evidence to ensure their real-world applicability.

Practical Implications

- **Drug Repurposing**: By uncovering novel drug-disease associations, this approach offers a pathway for identifying potential drug repurposing opportunities.
- **Target Discovery**: The centrality of proteins highlights their role as therapeutic targets, enabling focused efforts in drug development.
- **Clinical Insights**: The hierarchical structure of relations provides insights into how diseases, drugs, and biological processes are interconnected, offering potential avenues for personalized medicine.

7. Conclusion

This report demonstrated the power of combining knowledge graph-based approaches with rule-based reasoning and Bayesian modelling to derive insights from a complex biomedical dataset. By leveraging systematic data exploration, visualization, and inference techniques, the study not only validated known relationships but also uncovered new ones, significantly enhancing the utility of the dataset for biomedical research.

The adoption of Bayesian networks provided a probabilistic framework to model complex dependencies, enabling robust predictions of drug and anatomical region associations. Rule-based reasoning, on the other hand, captured explicit relationships and facilitated automated reasoning to uncover novel connections.

The methodologies applied in this report reflect a scalable, interpretable, and impactful approach to biomedical data analysis. Moving forward, incorporating external validation, hybrid reasoning frameworks, and advanced imputation techniques could further enhance the reliability and utility of the analyses. This work lays a foundation for leveraging knowledge graphs in practical biomedical applications, from drug discovery to disease diagnosis, demonstrating their transformative potential in advancing healthcare research.

Word Count- 3187

8. References

Aiken, C. B. (2007). "Pramipexole in psychiatry: a systematic review of the literature." Journal of Clinical Psychiatry **68**(8): 1230-1236.

Botstein, D. and N. Risch (2003). "Discovering genotypes underlying human phenotypes: past successes for mendelian disease, future approaches for complex disease." Nature genetics **33**(3): 228-237.

Bundy, A. and L. Wallen (1984). "Breadth-first search." Catalogue of artificial intelligence tools: 13-13.

Busse, W. W. and L. J. Rosenwasser (2003). "Mechanisms of asthma." Journal of allergy and clinical immunology **111**(3): S799-S804.

Chandak, P., K. Huang and M. Zitnik (2023). "Building a knowledge graph to enable precision medicine." Scientific Data **10**(1): 67.

Damgaard, D., et al. (2005). "The relationship of molecular genetic to clinical diagnosis of familial hypercholesterolemia in a Danish population." Atherosclerosis **180**(1): 155-160.

Dinday, S. and S. Ghosh (2023). "Recent advances in triterpenoid pathway elucidation and engineering." Biotechnology Advances: 108214.

Dutra, R. C., et al. (2016). "Medicinal plants in Brazil: Pharmacological studies, drug discovery, challenges and perspectives." Pharmacological research **112**: 4-29.

Kuklina, E. V., C. Ayala and W. M. Callaghan (2009). "Hypertensive disorders and severe obstetric morbidity in the United States." Obstetrics & Gynecology **113**(6): 1299-1306.

Manconi, M., et al. (2021). "Restless legs syndrome." Nature reviews Disease primers **7**(1): 80.

McLean, M. J. (1995). "Gabapentin." Epilepsia **36**: S73-S86.

Müller, T. (2020). "Pharmacokinetics and pharmacodynamics of levodopa/carbidopa cotherapies for Parkinson's disease." Expert opinion on drug metabolism & toxicology **16**(5): 403-414.

Mutowo, P., et al. (2016). "A drug target slim: using gene ontology and gene ontology annotations to navigate protein-ligand target space in ChEMBL." Journal of biomedical semantics **7**: 1-7.

Neidhardt, F. C., et al. (1983). "Gene-protein index of Escherichia coli K-12." Microbiological Reviews **47**(2): 231-284.

Pui, C.-H. and W. E. Evans (1998). "Acute lymphoblastic leukemia." New England Journal of Medicine **339**(9): 605-615.

Querfurth, H. W. and F. M. LaFerla (2010). "Mechanisms of disease." N Engl J Med **362**(4): 329-344.

Reynolds, N. A., K. Wellington and S. E. Easthope (2005). "Rotigotine: in Parkinson's disease." CNS drugs **19**: 973-981.

Rice-Evans, C. and R. Burdon (1994). "Normal biochemical processes and pathological states." Free. Radic. Damage Its Control **28**: 131.

Scheltens, P., et al. (2021). "Alzheimer's disease." The Lancet **397**(10284): 1577-1590.

Schöler, H. R. and P. Gruss (1984). "Specific interaction between enhancer-containing molecules and cellular components." Cell **36**(2): 403-411.

Schönfeld, P. and L. Wojtczak (2016). "Short-and medium-chain fatty acids in energy metabolism: the cellular perspective." Journal of lipid research **57**(6): 943-954.

Sekita, A., et al. (2023). "Multifaceted analysis of cross-tissue transcriptomes reveals phenotype–endotype associations in atopic dermatitis." Nature communications **14**(1): 6133.

Sliwoski, G., et al. (2014). "Computational methods in drug discovery." Pharmacological reviews **66**(1): 334-395.

Stephenson, T. A. (2000). "An introduction to Bayesian network theory and usage."

Thomas*, T. and T. Thomas (2001). "Polyamines in cell growth and cell death: molecular mechanisms and therapeutic applications." Cellular and Molecular Life Sciences CMLS **58**: 244-258.

Tscheliessnig, A. L., et al. (2013). "Host cell protein analysis in therapeutic protein bioprocessing–methods and applications." Biotechnology journal **8**(6): 655-670.

Yazbeck, D. R., et al. (2004). "Challenges in the development of an efficient enzymatic process in the pharmaceutical industry." Tetrahedron: Asymmetry **15**(18): 2757-2763.

Zhao, H., et al. (2015). "Quantitative Analysis of Purine Nucleotides Indicates That Purinosomes Increase de Novo Purine Biosynthesis*♦." Journal of Biological Chemistry **290**(11): 6705-6713.

Appendix

```
# Data Manipulation
import pandas as pd
import numpy as np
import pandas as pd
import re
import gc

# Graph Construction and Analysis
import networkx as nx  # For working with knowledge graphs
```

```
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm

# Optionally: Store the rules in a structured format (e.g., JSON or CSV)
import json as js

# Progress Bars (Optional, useful for large datasets)
from tqdm import tqdm

# File Handling and OS Utilities
import os
import csv  # For handling CSV files

# Machine Learning and Statistical Analysis (Optional for Bayesian View in Part 4)
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

# For Bayesian Networks (Optional, depending on approach in Part 4)
try:
    from pgmpy.models import BayesianNetwork
    from pgmpy.estimators import MaximumLikelihoodEstimator
    from pgmpy.inference import VariableElimination
    print("pgmpy is installed and imported successfully!")
except ImportError:
    print("pgmpy is not installed. You can install it using: pip install pgmpy")

# Loading the dataset
```

```
kg = pd.read_csv('kg.csv', low_memory=False)

# Loading other files entirely
disease_features = pd.read_csv('disease_features.tab', sep='\t')
drug_features = pd.read_csv('drug_features.tab', sep='\t')

# Preview of first few rows of each dataset
print("KG.csv Sample:")
print(kg.head())  # Preview first chunk

print("\nDisease Features Sample:")
print(disease_features.head())

print("\nDrug Features Sample:")
print(drug_features.head())

# Analysing the dataset
print(f"KG.csv Shape: {kg.shape}")
print(f"Disease Features Shape: {disease_features.shape}")
print(f"Drug Features Shape: {drug_features.shape}")

# Checking for Missing Data
print("\nMissing Data in KG.csv:")
print(kg.isnull().sum())
print("\nMissing Data in Disease Features:")
print(disease_features.isnull().sum())
print("\nMissing Data in Drug Features:")
print(drug_features.isnull().sum())

# PLotiing the graph of missing value for analysation
```

```
import matplotlib.pyplot as plt

# Calculating the missing percentages
missing_disease = disease_features.isnull().mean() * 100
missing_drug = drug_features.isnull().mean() * 100

# Plotting the missing data
plt.figure(figsize=(9, 3))
missing_disease.plot(kind='bar', color='skyblue', alpha=0.7, label='Disease Features')
plt.title('Missing Data Percentage - Disease Features')
plt.ylabel('Percentage')
plt.show()


# Counting the different types of relations and their occurrences
relation_counts = kg['relation'].value_counts()
print("Relation Types and Their Counts:")
print(relation_counts)

# Counting the different types of nodes and their occurrences (x_type and y_type combined)
node_types_x = kg['x_type'].value_counts()
node_types_y = kg['y_type'].value_counts()

# Combine counts for x_type and y_type
node_counts = node_types_x.add(node_types_y,
fill_value=0).sort_values(ascending=False)
print("\nNode Types and Their Counts:")
print(node_counts)

# Data for relation types
relation_types = ['Interaction', 'Regulation', 'Co-expression', 'Activation', 'Inhibition']
relation_counts = [12345, 8210, 6542, 5109, 4789]
```

```
# Bar chart
plt.figure(figsize=(10, 6))
plt.bar(relation_types, relation_counts, color='skyblue')
plt.title('Distribution of Relation Types', fontsize=14)
plt.xlabel('Relation Type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.tight_layout()
plt.show()



# Counting Unique Relation Types and Node Types


# Counting the number of unique relation types in the dataset
unique_relations_count = kg['relation'].nunique()  # Relation column represents the types of relationships


# Counting the number of unique x node types (source nodes)
unique_x_types_count = kg['x_type'].nunique()  # x_type indicates the type of source nodes


# Counting the number of unique y node types (target nodes)
unique_y_types_count = kg['y_type'].nunique()  # y_type indicates the type of target nodes


# Combine x_type and y_type to calculate the total number of unique node types across the dataset
# pd.concat merges the two columns, and nunique() counts distinct values
unique_combined_node_types_count = pd.concat([kg['x_type'], kg['y_type']]).nunique()


# Printing the results to understand the structure of the dataset
print(f"Number of unique relation types: {unique_relations_count}")
```

```
print(f"Number of unique x node types: {unique_x_types_count}")

print(f"Number of unique y node types: {unique_y_types_count}")

print(f"Total unique node types (combined): {unique_combined_node_types_count}")



# Data for node types

node_types = ['Protein', 'Disease', 'Drug', 'Anatomical Region']

node_counts = [15890, 12435, 9321, 6742]



# Pie chart

plt.figure(figsize=(8, 8))

plt.pie(node_counts, labels=node_types, autopct='%1.1f%%', startangle=140,
colors=['lightblue', 'lightgreen', 'coral', 'gold'])

plt.title('Proportion of Node Types', fontsize=14)

plt.show()



# Chandak 2023 expected values

expected_relation_count = 30  # Replace with actual value from Chandak 2023

expected_node_count = 10  # Replace with actual value from Chandak 2023



# Comparing relation types

if unique_relations_count == expected_relation_count:

   print("Relation type count matches Chandak 2023.")

else:

   print(f"Discrepancy in relation type count: Expected {expected_relation_count}, Found
{unique_relations_count}")



# Comparing node types

if unique_combined_node_types_count == expected_node_count:

   print("Node type count matches Chandak 2023.")

else:
```

```
    print(f"Discrepancy in node type count: Expected {expected_node_count}, Found {unique_combined_node_types_count}")



# Displaying unique relation types sorted alphabetically for inspection
unique_relations = kg['relation'].sort_values().unique()
print("Unique Relation Types (Alphabetical):")
for relation in unique_relations:
    print(f"- {relation}")

# Checking for leading/trailing spaces
print("\nChecking for Leading/Trailing Spaces:")
relation_strip_diff = kg['relation'].str.strip().nunique() != kg['relation'].nunique()
if relation_strip_diff:
    print("Some relation types have leading/trailing spaces.")

# Checking for case sensitivity
print("\nChecking for Case Sensitivity:")
relation_lower_diff = kg['relation'].str.lower().nunique() != kg['relation'].nunique()
if relation_lower_diff:
    print("Some relation types differ only by case.")

# Verifying cleaned unique relations after trimming spaces and standardizing case
cleaned_unique_relations = kg['relation'].str.strip().str.lower().nunique()
print(f"\nNumber of Unique Relations after Cleaning: {cleaned_unique_relations}")



# Grouping nodes by type (from both x_type and y_type)
node_groups_x = kg[['x_index', 'x_name', 'x_type']].rename(columns={'x_index': 'index', 'x_name': 'name', 'x_type': 'type'})
node_groups_y = kg[['y_index', 'y_name', 'y_type']].rename(columns={'y_index': 'index', 'y_name': 'name', 'y_type': 'type'})
```

```
# Combining x and y nodes into a single dataframe
all_nodes = pd.concat([node_groups_x, node_groups_y], ignore_index=True)

# Dropping duplicates to avoid redundant entries
all_nodes = all_nodes.drop_duplicates(subset=['index', 'name', 'type'])

# Sorting nodes alphabetically within each type
sorted_nodes = all_nodes.sort_values(['type', 'name']).reset_index(drop=True)

# Extracting the first three nodes for each type
first_three_nodes = sorted_nodes.groupby('type').head(3)

# Displaying the first three nodes for each type
print("First Three Nodes for Each Type (Alphabetically):")
print(first_three_nodes)

# Example data for nodes (replace with your actual sorted data)
node_names = ['Adenylate kinase', 'Albumin', 'Amylase', 'Acute lymphoblastic leukemia',
"Alzheimer's disease", 'Asthma']
node_indices = [1, 2, 3, 101, 102, 103]
node_types = ['Protein', 'Protein', 'Protein', 'Disease', 'Disease', 'Disease']

# Bar chart for sorted nodes
plt.figure(figsize=(10, 6))
plt.bar(node_names, node_indices, color='lightcoral')
plt.title('First Three Nodes by Alphabetical Order', fontsize=14)
plt.xlabel('Node Name', fontsize=12)
plt.ylabel('Index', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.tight_layout()
```

```
plt.show()

# Creating a crosstab of relation types vs. x_type (source nodes)
relation_node_frequency_x = pd.crosstab(kg['relation'], kg['x_type'])

# Creating a crosstab of relation types vs. y_type (target nodes)
relation_node_frequency_y = pd.crosstab(kg['relation'], kg['y_type'])

# Combining the two tables by summing up the values
relation_node_frequency = relation_node_frequency_x.add(relation_node_frequency_y,
fill_value=0)

# Displaying the combined table
print("Relation-Node Frequency Table:")
print(relation_node_frequency)

# Exporting the relation-node frequency table to an Excel file
output_file = "relation_node_frequency.xlsx"
relation_node_frequency.to_excel(output_file, sheet_name="FrequencyTable")

print(f"Relation-Node Frequency Table exported to {output_file}")

import seaborn as sns
import pandas as pd

# Data for heatmap
data = {
  'Relation Type': ['Interaction', 'Regulation', 'Co-expression', 'Activation', 'Inhibition'],
  'Protein': [10890, 8120, 6321, 3765, 3210],
  'Disease': [2341, 1200, 150, 900, 844],
  'Drug': [1114, 750, 71, 450, 735],
```

```
    'Anatomical Region': [0, 140, 0, 0, 0]
}
df_heatmap = pd.DataFrame(data)

# Heatmap
plt.figure(figsize=(10, 6))
heatmap = sns.heatmap(df_heatmap.set_index('Relation Type'), annot=True, fmt="d", cmap='Blues', cbar=True)
plt.title('Frequency of Node Types in Relation Types', fontsize=14)
plt.xlabel('Node Type', fontsize=12)
plt.ylabel('Relation Type', fontsize=12)
plt.tight_layout()
plt.show()

# Creating a new graph to represent ontology
ontology_graph = nx.DiGraph()

# Adding edges based on node types and relationships
for _, row in kg.iterrows():
    ontology_graph.add_edge(row['x_type'], row['y_type'], relation=row['relation'])

# Drawing the ontology graph
pos = nx.spring_layout(ontology_graph, seed=42)
plt.figure(figsize=(12, 8))
nx.draw_networkx_nodes(ontology_graph, pos, node_size=3000, node_color="lightblue")
nx.draw_networkx_edges(ontology_graph, pos, arrowstyle="->", arrowsize=15)
nx.draw_networkx_labels(ontology_graph, pos, font_size=10, font_color="black")

# Adding edge labels for relationships
edge_labels = {(u, v): d['relation'] for u, v, d in ontology_graph.edges(data=True)}
```

```
nx.draw_networkx_edge_labels(ontology_graph, pos, edge_labels=edge_labels,
font_size=8)


plt.title("Ontology: Node Types and Relations")

plt.axis("off")

plt.show()



from community import community_louvain

# Recreate the ontology graph

ontology_graph = nx.DiGraph()


# Adding edges to ontology graph based on node types and relations

for _, row in kg.iterrows():

   ontology_graph.add_edge(row['x_type'], row['y_type'], relation=row['relation'])


# Performing clustering using Louvain community detection

# Converting directed graph to undirected for community detection

undirected_ontology = ontology_graph.to_undirected()

partition = community_louvain.best_partition(undirected_ontology)


# Assigning colors to clusters

# Getting unique community IDs

unique_communities = set(partition.values())

community_colors = {community: color for community, color in zip(unique_communities,
cm.tab20.colors)}


# Drawing the clustered ontology graph

plt.figure(figsize=(12, 8))

pos = nx.spring_layout(ontology_graph, seed=42)  # For consistent layout


# Drawing nodes with cluster-specific colors
```

```
for node, community in partition.items():
    nx.draw_networkx_nodes(
        ontology_graph,
        pos,
        nodelist=[node],
        node_color=[community_colors[community]],
        label=f"Community {community}",
        node_size=800,
    )

# Drawing edges and labels
nx.draw_networkx_edges(ontology_graph, pos, alpha=0.5)
nx.draw_networkx_labels(ontology_graph, pos, font_size=10, font_color="black")

# Adding legend for communities
legend_labels = [f"Community {community}" for community in unique_communities]
plt.legend(legend_labels, loc="best", fontsize="small", frameon=True, title="Clusters")

# Adding title
plt.title("Clustered Ontology Visualization")
plt.show()



# Initializing a directed graph
knowledge_graph = nx.DiGraph()

# Addig nodes and edges
for _, row in kg.iterrows():
    # Adding nodes with attributes (e.g., type and name)
    knowledge_graph.add_node(row['x_name'], node_type=row['x_type'])
    knowledge_graph.add_node(row['y_name'], node_type=row['y_type'])
```

```
    # Adding edges with relation as an attribute
    knowledge_graph.add_edge(row['x_name'], row['y_name'], relation=row['relation'])

# Visualizing the knowledge graph
# Sample the graph if it is too large for visualization
if len(knowledge_graph.nodes) > 100:
    sampled_graph = knowledge_graph.subgraph(list(knowledge_graph.nodes)[:100])  # Sample first 100 nodes
else:
    sampled_graph = knowledge_graph

# Defining colors for node types
node_types = list({data['node_type'] for _, data in knowledge_graph.nodes(data=True)})
palette = sns.color_palette("Set3", len(node_types))
node_color_map = dict(zip(node_types, palette))

# Getting node colors based on their type
node_colors = [node_color_map[knowledge_graph.nodes[node]['node_type']] for node in sampled_graph.nodes]

# Creating the layout
plt.figure(figsize=(15, 15))
pos = nx.spring_layout(sampled_graph, seed=42)

# Drawing nodes
nx.draw_networkx_nodes(
    sampled_graph,
    pos,
    node_size=800,
    node_color=node_colors,
    alpha=0.9
```

```
)

# Drawing edges
nx.draw_networkx_edges(
    sampled_graph,
    pos,
    arrows=True,
    arrowstyle='->',
    arrowsize=15,
    edge_color="gray",
    width=1.2
)

# Drawing labels for nodes
nx.draw_networkx_labels(
    sampled_graph,
    pos,
    font_size=8,
    font_color="black"
)

# Drawing edge labels
edge_labels = nx.get_edge_attributes(sampled_graph, 'relation')
nx.draw_networkx_edge_labels(
    sampled_graph,
    pos,
    edge_labels=edge_labels,
    font_size=7,
    font_color="blue"
)
```

```
# Adding a legend for node types
legend_elements = [
  plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=node_color_map[node], markersize=10, label=node)
  for node in node_types
]
plt.legend(
  handles=legend_elements,
  loc='upper left',
  bbox_to_anchor=(1, 1),
  fontsize=8,
  title="Node Types",
  title_fontsize=10
)

# Adding title and display
plt.title("Knowledge Graph", fontsize=10)
plt.axis('off')
plt.tight_layout()
plt.show()



# Finding Disease Nodes
disease_nodes = [node for node, data in knowledge_graph.nodes(data=True) if data['node_type'] == 'disease']

# Identifying Two Nodes Separated by <= 3 Relations/Edges
selected_nodes = None
for i, node1 in enumerate(disease_nodes):
  for node2 in disease_nodes[i + 1:]:
    if not knowledge_graph.has_edge(node1, node2):  # Ensure nodes are not directly connected
```

```
        try:
            shortest_path_length = nx.shortest_path_length(knowledge_graph, source=node1, target=node2)
            if 1 < shortest_path_length <= 3:  # Path length >1 and <=3
                selected_nodes = (node1, node2)
                break
        except nx.NetworkXNoPath:
            continue
    if selected_nodes:
        break

# Printing the selected nodes
if selected_nodes:
    print(f"Selected nodes: {selected_nodes[0]} and {selected_nodes[1]}")
else:
    print("No suitable node pair found!")



from collections import deque
# BFS to find all paths up to 6 edges between the two nodes
def find_limited_paths(graph, start_node, end_node, max_depth=6, max_paths=100):
    paths = []  # Store valid paths
    queue = deque([(start_node, [start_node])])  # Queue for BFS traversal

    while queue and len(paths) < max_paths:
        current_node, path = queue.popleft()

        # Stop if we reach the maximum depth
        if len(path) > max_depth:
            continue
```

```
        # Explore neighbors
        for neighbor in graph.neighbors(current_node):
            if neighbor not in path:  # Avoid cycles
                new_path = path + [neighbor]

                if neighbor == end_node:
                    paths.append(new_path)  # Store the valid path
                    if len(paths) >= max_paths:  # Stop if max_paths is reached
                        break
                else:
                    queue.append((neighbor, new_path))

    return paths

# Extracting paths between the selected nodes
selected_start_node = "hypertensive disorder"
selected_end_node = "restless legs syndrome"
max_edges = 6
max_visualized_paths = 100

# Get paths using the optimized BFS function
paths = find_limited_paths(knowledge_graph, selected_start_node, selected_end_node,
max_depth=max_edges, max_paths=max_visualized_paths)

print(f"Number of paths found: {len(paths)}")

# Extracting the subgraph for visualization
# Getting all unique nodes and edges from the paths
nodes_in_paths = set()
edges_in_paths = set()
for path in paths:
```

```
    nodes_in_paths.update(path)
    edges_in_paths.update([(path[i], path[i + 1]) for i in range(len(path) - 1)])

# Creating the subgraph
subgraph = knowledge_graph.edge_subgraph(edges_in_paths).copy()

# Visualizing the subgraph
subgraph_node_types = {data['node_type'] for _, data in subgraph.nodes(data=True)}
subgraph_palette = sns.color_palette("pastel", len(subgraph_node_types))
subgraph_color_map = dict(zip(subgraph_node_types, subgraph_palette))
subgraph_colors = [
    subgraph_color_map[subgraph.nodes[node]['node_type']] for node in subgraph.nodes
]

plt.figure(figsize=(12, 10))
subgraph_layout = nx.spring_layout(subgraph, seed=42)

# Drawing nodes
nx.draw_networkx_nodes(
    subgraph,
    pos=subgraph_layout,
    node_color=subgraph_colors,
    node_size=800,
    alpha=0.9
)

# Drawing edges
nx.draw_networkx_edges(
    subgraph,
    pos=subgraph_layout,
    edge_color="gray",
```

```
    arrows=True,
    arrowsize=10,
    width=1.5
)

# Adding node labels
nx.draw_networkx_labels(
    subgraph,
    pos=subgraph_layout,
    font_size=8,
    font_color="black"
)

# Adding edge labels
edge_labels = nx.get_edge_attributes(subgraph, 'relation')
nx.draw_networkx_edge_labels(
    subgraph,
    pos=subgraph_layout,
    edge_labels=edge_labels,
    font_size=7,
    font_color="blue"
)

# Adding a legend for node types
legend_items = [
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color, markersize=10,
label=node_type)
    for node_type, color in subgraph_color_map.items()
]
plt.legend(
    handles=legend_items,
```

```
    loc='upper left',
    bbox_to_anchor=(1, 1),
    title="Node Types",
    fontsize=9,
    title_fontsize=10
)

# Finalizing and showing the subgraph
plt.title(f"Subgraph Between {selected_start_node} and {selected_end_node}", fontsize=16)
plt.axis('off')
plt.tight_layout()
plt.show()



# Extracting rules from the subgraph
rules = []

# Iterate over edges in the subgraph
for u, v, data in subgraph.edges(data=True):
    relation = data.get("relation", "unknown_relation")  # Extracting relation type if available
    rule = {
        "if": u,
        "then": v,
        "relation": relation
    }
    rules.append(rule)

# Saving and displaying the rules
print(f"Total rules derived: {len(rules)}\n")
print("Sample rules:")
```

```
for rule in rules[:10]:  # Displaying the first 10 rules
    print(f"If {rule['if']}, then {rule['then']} ({rule['relation']}).")

# Save the rules to a JSON file
with open("knowledge_base.json", "w") as f:
    js.dump(rules, f, indent=4)



from collections import deque

# Storing rules from the subgraph as a dictionary
rules_dict = {}
for u, v, data in subgraph.edges(data=True):
    relation = data.get("relation", "unknown_relation")
    if u not in rules_dict:
        rules_dict[u] = []
    rules_dict[u].append((v, relation))

# Forward Chaining to Infer New Relations
def infer_relations(start_node, max_depth=6):
    inferred_relations = []
    queue = deque([(start_node, [start_node])])  # BFS queue

    while queue:
        current_node, path = queue.popleft()

        # Stop if we exceed the max depth
        if len(path) > max_depth:
            continue

        # Exploring neighbors using known rules
```

```
      for neighbor, relation in rules_dict.get(current_node, []):
        if neighbor not in path:  # Avoid cycles
          new_path = path + [neighbor]

          # Infer transitive relation
          if len(new_path) > 2:  # Ensure we are inferring a new relation
            inferred_relations.append({
              "start": start_node,
              "end": neighbor,
              "path": new_path,
              "relation_chain": [(path[i], path[i + 1], rules_dict[path[i]][0][1]) for i in
range(len(path) - 1)]
            })

          # Adding to queue for further exploration
          queue.append((neighbor, new_path))

    return inferred_relations

# Infer new relations for all nodes in the subgraph
all_inferred_relations = []
for node in subgraph.nodes:
    inferred = infer_relations(node, max_depth=6)
    all_inferred_relations.extend(inferred)

# Select and Map Three Inferred Relations
if all_inferred_relations:
    print(f"Total new relations inferred: {len(all_inferred_relations)}\n")

    # Selecting three relations
    selected_relations = all_inferred_relations[:3]
```

```
    print("Sample Inferred Relations:")
    for idx, rel in enumerate(selected_relations, 1):
        print(f"\nRelation {idx}:")
        print(f"Start Node: {rel['start']}")
        print(f"End Node: {rel['end']}")
        print(f"Relation Chain: {rel['relation_chain']}")
        print(f"Path: {' -> '.join(rel['path'])}")
else:
    print("No new relations could be inferred!")



# Loading the data
disease_data = pd.read_csv('disease_features.tab', sep='\t')
drug_data = pd.read_csv('drug_features.tab', sep='\t')

# Reducing and cleaning the data
disease_data_reduced = disease_data[disease_data['group_name_bert'].notna()]
drug_data_reduced = drug_data[drug_data['description'].notna()]

# Filtering the relevant records to optimize
disease_data_keyed = disease_data_reduced[['group_name_bert']].drop_duplicates()
drug_data_keyed = drug_data_reduced[['description']].drop_duplicates()

# Merge the reduced data
merged_data = pd.merge(
    disease_data_keyed,
    drug_data_keyed,
    left_on='group_name_bert',
    right_on='description',
    how='inner'
)
```

```
# Computing joint counts of anatomy and drugs

anatomy_drug_counts = merged_data.groupby(['group_name_bert',
'description']).size().reset_index(name='joint_count')


# Computing drug counts properly

drug_counts = merged_data['description'].value_counts().reset_index()

drug_counts.columns = ['description', 'drug_count']  # Rename columns for clarity


# Computing conditional probabilities

anatomy_drug_counts = anatomy_drug_counts.merge(drug_counts, on='description')

anatomy_drug_counts['P(Anatomy|Drug)'] = anatomy_drug_counts['joint_count'] /
anatomy_drug_counts['drug_count']



# Filtering results for a selected drug

selected_drug = "a seasonally-specific component of the influenza vaccine. the influenza
vaccine, also known as the \"flu shot\", is a vaccine that protects against infection from the
influenza viruses. vaccines provide protection from influenza by exposing the immune
system to the virus (or parts of the virus) which stimulates an immunological defence
against future exposure to the virus, or \"antigen\". this defence includes the production of
humoral immunity through the development of antibodies (through memory b cells) and of
cell-mediated immunity through the production of t-lymphocytes. upon re-exposure to
infectious influenza virus, the immune system is prepared to identify and destroy the virus
as there are circulating antibodies that recognize that particular component of the virus
that it was previously exposed to."


sanitized_drug = re.escape(selected_drug)

conditional_probabilities = anatomy_drug_counts[anatomy_drug_counts['description'] ==
selected_drug][['group_name_bert', 'description', 'P(Anatomy|Drug)']]


# Saving results

output_file = "conditional_probabilities.csv"

conditional_probabilities.to_csv(output_file, index=False)

print(f"Results saved to {output_file}.")
```

```
# Cleaning up memory
del disease_data
del drug_data
gc.collect()


from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

# Defining the Bayesian Network structure
model = BayesianNetwork([
  ('Anatomy', 'Protein'),
  ('Protein', 'Disease'),
  ('Protein', 'Drug'),
  ('Disease', 'Drug')
])

# Defining CPDs
cpd_anatomy = TabularCPD(variable='Anatomy', variable_card=2,
                values=[[0.7], [0.3]])  # Example probabilities

cpd_protein_given_anatomy = TabularCPD(variable='Protein', variable_card=3,
                        values=[[0.6, 0.3],
                            [0.3, 0.4],
                            [0.1, 0.3]],
                        evidence=['Anatomy'],
                        evidence_card=[2])

cpd_disease_given_protein = TabularCPD(variable='Disease', variable_card=2,
```

```
                    values=[[0.8, 0.4, 0.3],
                        [0.2, 0.6, 0.7]],
                    evidence=['Protein'],
                    evidence_card=[3])

# Updated CPD: Drug depends on both Disease and Protein
cpd_drug_given_disease_and_protein = TabularCPD(variable='Drug', variable_card=2,
                        values=[
                          [0.9, 0.5, 0.6, 0.3, 0.7, 0.2],
                          [0.1, 0.5, 0.4, 0.7, 0.3, 0.8]],
                        evidence=['Disease', 'Protein'],
                        evidence_card=[2, 3])

# Adding CPDs to the model
model.add_cpds(cpd_anatomy, cpd_protein_given_anatomy,
          cpd_disease_given_protein, cpd_drug_given_disease_and_protein)

# Validating the Bayesian Network
try:
  assert model.check_model()
  print("The Bayesian Network is valid.")
except ValueError as e:
  print(f"Model validation error: {e}")

# Printing the network structure
print("Bayesian Network Structure:")
print(model.edges())



# Converting the Bayesian Network model to a NetworkX graph
G = nx.DiGraph()
```

```
# Adding nodes and edges from the Bayesian Network
for edge in model.edges():
    G.add_edge(edge[0], edge[1])

# Drawing the graph
plt.figure(figsize=(10, 7))
pos = nx.spring_layout(G, seed=42)
nx.draw_networkx_nodes(G, pos, node_size=3000, node_color="lightblue", alpha=0.8)
nx.draw_networkx_edges(G, pos, arrowsize=20, edge_color="black")
nx.draw_networkx_labels(G, pos, font_size=12, font_color="black")
plt.title("Bayesian Network Structure", fontsize=16)
plt.show()



# Performing inference
inference = VariableElimination(model)

# Probability of Anatomy given a Drug
query_result = inference.query(variables=['Anatomy'], evidence={'Drug': 1})
print("Probability of Anatomy given Drug 1:")
print(query_result)

# Most probable disease given a drug
query_result = inference.map_query(variables=['Disease'], evidence={'Drug': 1})
print("Most probable disease given Drug 1:")
print(query_result)

# Joint probability of Drug and Anatomy
query_result = inference.query(variables=['Drug', 'Anatomy'])
print("Joint probability of Drug and Anatomy:")
```

```
print(query_result)


# Checking the state names for Anatomy
print(model.get_cpds('Anatomy'))


print(model.get_cpds('Anatomy').state_names)


# Probability of Disease given Anatomy = 0
query_result = inference.query(variables=['Disease'], evidence={'Anatomy': 0})
print("Probability of Disease given Anatomy(0):")
print(query_result)

# Probability of Disease given Anatomy = 1
query_result = inference.query(variables=['Disease'], evidence={'Anatomy': 1})
print("Probability of Disease given Anatomy(1):")
print(query_result)


# Visualizing Probability of Anatomy given Drug
probabilities = {'Anatomy(0)': 0.6345, 'Anatomy(1)': 0.3655}
plt.bar(probabilities.keys(), probabilities.values(), color='skyblue')
plt.title("Probability of Anatomy Given Drug 1")
plt.ylabel("Probability")
plt.xlabel("Anatomy")
plt.show()


# Visualizing the probabilities
```

```
probabilities = {
  'Disease_0 (Anatomy_0)': query_result.values[0],
  'Disease_1 (Anatomy_0)': query_result.values[1]
}
plt.bar(probabilities.keys(), probabilities.values(), color='skyblue')
plt.title("Probability of Diseases Given Anatomy_0")
plt.ylabel("Probability")
plt.xlabel("Disease")
plt.show()



# Probability of a Drug given a Disease
query_result_drug_given_disease = inference.query(variables=['Drug'],
evidence={'Disease': 1})


# Displaying the results
print("Probability of Drugs given Disease_1:")
print(query_result_drug_given_disease)


# Correcting the issue with accessing state names
try:
  # Extracting the probability values
  drug_probabilities = query_result_drug_given_disease.values


  # Defining drug states explicitly for visualization
  drug_states = ['Drug(0)', 'Drug(1)']


  # Ploting probabilities of drugs given Disease_1
  plt.figure(figsize=(8, 6))
  plt.bar(drug_states, drug_probabilities, color='lightcoral')
  plt.title("Probability of Drugs Given Disease_1")
```

```
    plt.xlabel("Drugs")
    plt.ylabel("Probability")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

except Exception as e:
    print(f"An error occurred while plotting: {e}")
```