Automated Data Preparation using Graph Neural Networks

Niki Monjazeb

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

August 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Niki Monjazeb**

Entitled: **Automated Data Preparation using Graph Neural Networks**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Sandra Cespedes

_____ Examiner
Dr. Sandra Cespedes

_____ Examiner
Dr. Abdelhak Bentaleb

_____ Supervisor
Dr. Essam Mansour

Approved by _____
Joey Paquet, Chair
Department of Computer Science and Software Engineering

_____ 2024

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Automated Data Preparation using Graph Neural Networks

Niki Monjazeb

The process of data preparation is a time-consuming portion of data scientists' work. Being able to automate this work will improve the quality of the machine learning results and free data scientists to shift their focus to the machine learning task at hand. My research presents a system to automate this process by learning from the data preparation steps taken from others working on similar datasets. To automate data cleaning and transformation, datasets and their corresponding notebooks were extracted from Kaggle, their information was abstracted before being uploaded into a knowledge graph. Graph Neural Network (GNN) models were trained on those knowledge graphs, and the most commonly used cleaning and transformation operations for similar datasets were inferred. These operations are offered to the user as recommendations that they can apply to their dataset using the corresponding APIs. These recommendations have outperformed their state-of-the-arts counterparts in terms of time, memory consumption, and accuracy. To detect similarity inclusion dependencies (sIND), knowledge graphs from datasets in the Prague Relational Learning Repository were created. From those knowledge graphs, the columns deemed to have an inclusion dependency were studied until features leading to this dependency were observed. These features were used to create a model that could predict the sIND between columns. The resulting model was able to correctly predict more sIND pairs, in a shorter timespan than its competitor. This holistic platform can easily be integrated into any Data Science Pipeline (DSP) and facilitate the data preparation process for data scientists.

Acknowledgments

I would like to express my deepest gratitude to Dr. Essam Mansour, my supervisor for his guidance and mentorship through this learning journey. I extend my sincere appreciation to the members of the CODS lab for providing a stimulating and supportive research environment, with a special thanks to Shubham Vashist, Mossad Helali, and Philippe Carrier for their help and valuable insights.

I would also like to thank my parents for ingraining in me from a young age that I can accomplish whatever I set my mind to, as well as my brother for his unwavering support. Their belief in me has been a constant source of motivation, and I am forever grateful for their love and sacrifices.

Last but not least, I owe a special thanks to my husband. His constant support, unwavering patience, thoughtfulness, inspirational kindness, and empathy are what guided me through this journey, and for that, I will always be grateful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

According to Abdallah et al, (Abdallah, Du, & Webb, 2017), the process of data preparation takes 70-80% of a data scientist's time. This process consists of gathering the relevant data, and cleaning and transforming them. These steps are instrumental in ensuring the production of high quality data which plays a pivotal role in the success of machine learning models. While many techniques and libraries have been developed to assist in this process, it still remains a time consuming and complex endeavor. Our system, KGDataPrep, simplifies this process by providing recommendations for the best cleaning and transformation operations to use and takes a step to facilitate data enrichment via primary-key foreign-key detection by detecting similarity inclusion dependency pairs (sIND).

Despite the inherently time-consuming nature of data preparation, the knowledge gained by the data scientist remains largely unshared due to a low level of communication between individuals and teams within the company. Data scientists in a company might communicate their methods and the resulting findings with a few members within their immediate team, however, the details of their work will largely remain unshared, as illustrated in 1.1. A data scientist might spend a significant amount of time developing a Data Science Pipeline (DSP) which could benefit multiple projects within a company, only for it to be lost within the mountain of documentation created. Furthermore, junior data scientist will not be able to benefit from the knowledge of their more senior colleagues

without having to review documentation.

The redundancy caused by the repetition of similar work by multiple people is detrimental to productivity. It hinders knowledge sharing and takes time away from other tasks that could be done, including the machine learning portion of the DSP, which is crucial for addressing important data science questions. Furthermore, without effective knowledge sharing, it is uncertain whether the data has been optimally cleaned, once again underlining the importance of establishing a culture knowledge-sharing through automation (Mansour, Srinivas, & Hose, 2022).

There are currently many public initiatives that endorse knowledge sharing within the data science field. Collaborative data science platforms such as Kaggle [1] and OpenML [2] offer a wide range of datasets as well as their corresponding DSPs. Data scientists of all levels used these platforms to hone their skills and learn. They mostly do so by participating in competitions and reviewing other data scientist's notebooks and tutorials. Data scientists use these platforms to build on each other's skills and perfect the steps of the DSP related to a particular dataset. These steps range from enrichment, to data cleaning, data transformation, and data selection. Given the different formats in which the datasets can be presented, as well as the different languages in which the DSP can be written, a unifying factor is needed to facilitate the process of knowledge sharing via automation.

Helali et al. (Helali et al., 2024) has found such a unifying factor in knowledge graphs representing the datasets as well as their associated DSP. In this work, we explore the concept of information sharing via knowledge graphs by using them to create models that will facilitate the data preparation process for data scientist.

1.2 Contributions

This thesis introduces a comprehensive platform leveraging a Linked Data Science Powered by Knowledge Graphs (KGLiDS) to streamline the data preparation process. Our system revolves around five distinct models, each serving a specific purpose. Among these models, three are Graph Neural Network (GNN) models tailored for data cleaning and transformation. One GNN model specializes in data cleaning, another focuses on unary transformation, and the third handles scaling

[1] https://www.kaggle.com/
[2] https://www.openml.org/

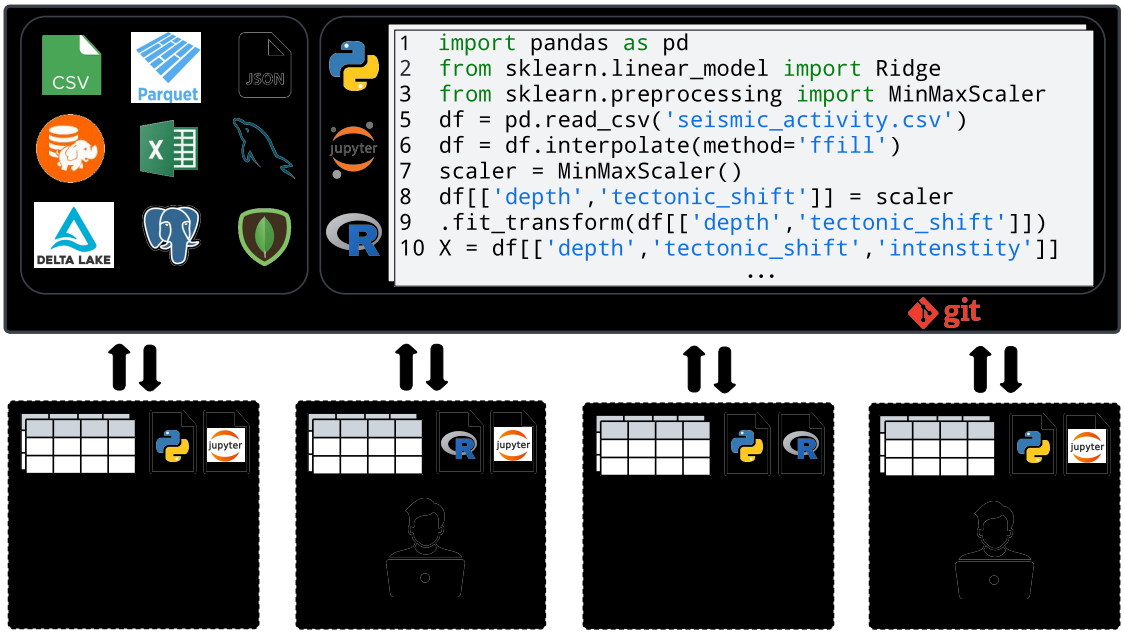

Figure 1.1: Data scientists perform time-consuming, code extensive, and repetitive data preparation in isolation, however, their insights are rarely shared.

transformation. The last two models are a random forest model and an isolation forest model that when combined, are used to detect sINDs. To validate the efficacy of our approach, comprehensive experiments were conducted, benchmarking our models against state-of-the-art (SOTA) systems operating within a similar functional domain.

The data preparation operations, such as data cleaning and data transformation, are formalized as multi-class classification where the goal is to predict the most appropriate cleaning or transformation operations from a predefined set of operations. To create these models, I constructed a data science knowledge graph from top-rated 1000 Kaggle datasets and 13800 pipeline scripts in the form of notebooks with the highest number of votes using KGLiDS (Helali et al., 2024). The most common cleaning and transformation operations used in the extracted notebooks were used as the targets for the multi-class classification. The GNN models were initialized using the embeddings of the related dataset created in the profiling phase of the KGLiDS system. For any new datasets, the embeddings are created before the previously mentioned models are used for inference.

To create the sIND models, the problem of sIND detection is stated as both an outlier detection problem, and a binary classification problem. The problem type is selected based on the size of the dataset and the similarity between its columns. Therefore, a combination of both models is used to

solve this problem. Datasets from the Prague Relational Repository (Motl & Schulte, 2024) were used to build a model for sIND as these datasets contained predefined relationships. 19 of the real-world datasets from the Prague relational repository were used to build the models for sIND. The features used to build these models were extracted from the knowledge graph built by KGLiDS. The features used are the normalized Levenshtein distance between the names of the 2 columns, the similarity score between the 2 columns, the percentage of the values in the dependent column that were distinct, and the ratio of the size of the dependent column over the referenced column. These features are used to build both the isolation forest and the random forest models whose combination leads to our sIND detection system. During inference, KGLiDS is used to create a knowledge graph for the unseen dataset. The knowledge graph is then queried to create the features used for the inference for the model.

The contributions of this thesis are:

- GNN models to be used for data cleaning and data transformation.
- A comprehensive evaluation of 13 datasets containing missing values from the AutoML benchmark and UCI repository using the data cleaning model provided in this thesis as well as SOTA AIMNET(Wu, Zhang, Ilyas, & Rekatsinas, 2020).
- A comprehensive evaluation of 17 datasets from autolearn's experimental evaluation (Kaul et al., 2017), available in the UCI repository (Dua & Graff, 2017) using the data transformation model provided in this thesis as well as SOTA Autolearn (Kaul et al., 2017).
- A user friendly platform allowing for the cleaning and transformation, as well as the detection of Inclusion dependencies (IND) and similarity inclusion dependencies (sIND).
- Models to be used for IND and sIND detection
- A methodology for the evaluation sIND detection systems.
- A comprehensive evaluation of 6 datasets from the Prague repository using the sIND detection model provided in this thesis as well as SOTA SAWFISH (Kaminsky, Pena, & Naumann, 2023).

1.3 Outline

This thesis consists of six chapters that delve into the details behind the approach followed to automate the problem of data preparation. Chapter 2 will be an overview of the related literature in areas related to data cleaning, data transformation, as well as sIND detection. Chapter 3 provides the required background into the KGLiDS system, laying the groundwork for understanding the inner workings of the models used for data cleaning and transformation, as well as the IND and sIND detection. Chapter 4 presents the steps taken to prepare the data cleaning and transformation GNN models, from preparing the KGLiDS knowledge graph, to training the GNN and setting up the inference, as well as the experiments done to establish these models as equal to or exceeding SOTA systems. Chapter 5 presents the problem of sIND detection, and elaborates on the steps used to develop a model to detect sIND, as well as the experiments done to ensure the quality of this model. Chapter 6 is a conclusion of the work done to date, followed by a recommendation for the work to be done in the future.

Chapter 2

Related Works

This chapter surveys previous work done in the data preparation space. Although much work has been done in developing ways to help data scientists prepare their data for machine learning, a majority of that work is focused on improving one aspect of data preparation. Our system allows for both cleaning and transformation of datasets, as well as sIND detection.

Systems such as Holoclean (Wu et al., 2020) and Datawig (Biessmann et al., 2019) have focused on data cleaning, while others such as Autolearn (Kaul et al., 2017) and the framework Learning Feature Engineering (LFE) (Nargesian, Samulowitz, Khurana, Khalil, & Turaga, 2017) which focus on feature transformation. Systems such as DataPrep (Peng, Wu, Lockhart, & et al, 2021) and Data Civilizer (Rezig et al., 2019) have chosen a more holistic approach and cover multiple steps of the pipeline. SAWFISH (Kaminsky et al., 2023) introduces and explores the concept of sIND, separate from any other data preparation techniques.

2.1 Cleaning

Data cleaning is an important step of the data preparation process. While data preparation can encompass a variety of tasks, such as missing value imputation, error correction and outlier detection, our focus will be on systems that have been developed to address the challenge of missing value imputation, as it is the aspect of data cleaning that we have chosen to focus on in this thesis.

2.1.1 Holoclean

Holoclean (Wu et al., 2020) uses statistical learning and inference to unify a range of data-repairing methods. In its most recent version, Aimnet, the user can choose to specify a set of denial constraints to facilitate the data cleaning process. Aimnet cleans datasets of duplicates, spelling errors, and constraint violations. It does so by converting discrete values to embeddings and applying z-score normalization to continuous values. It is trained using self-supervised gradient descent-based end-to-end learning. AimNet applies a multi-task loss approach to datasets containing both numerical and categorical values. This allows it to address both regression (for continuous data) and classification (for discrete data). The Categorical Cross Entropy (CCE) loss is then computed between the predicted probabilities and the actual target value in the dataset. Aimnet has shown promising results for mixed-type data imputation. However, it can be a resources intensive system to run for larger datasets.

2.1.2 Datawig

Datawig (Biessmann et al., 2019) is an open-sourced library for missing value imputation. DataWig uses deep learning feature extractors, combined with automatic hyper-parameter tuning in order to impute values to clean data. Datawig follows the method of Multivariate Imputation by Chained Equations (MICE) (van Buuren & Groothuis-Oudshoorn, 2011), a method that imputes each missing value by considering all other columns in the dataset, for each column in need of cleaning. Datawig also allows for the user to specify similar columns that will facilitate the imputation process. Due to its usage of the MICE method, the training time for this system will increase with the dataset size.

2.2 Transformation

Data transformation can be defined as any systematic change to the data that would enhance its comprehensibility for humans and/or models. Although there exists several systems that transform a dataset's features into more meaningful features, each comes with their set of advantages and disadvantages as explained below.

2.2.1 Autolearn

Autolearn (Kaul et al., 2017) is a regression-based feature learning algorithm. An important advantage of Autolearn is its ability to create and transform features without any domain knowledge. It ranks the features of a dataset based on their information gain. Features with an information gain higher than a user-defined value (0 as default) are then selected. The algorithm employs the distance correlation to identify pairwise correlated features, classifies them into linear and non-linear correlations, and generates informative new features. Stability calculations are then done via sub-sampling as well as the use of selection algorithms (regression, SVM,...). The features are ranked based on stability and information gain, with only the top ones being selected. While Autolearn can provide robust feature transformations for datasets containing both numerical and categorical values, its time and memory consumption remains a challenge. The original dataset's row and feature count, inter-feature correlations, created features, and the chosen feature quantity, all impact Autolearn's memory usage. Hence, the dataset's absolute size is not the primary factor influencing memory usage in Autolearn's transformation. In fact, due to the unpredictable nature of the feature engineering algorithm, it is difficult to estimate the time requirements of the system for a given dataset.

2.2.2 LFE

The LFE framework (Nargesian et al., 2017) offers automated feature engineering for classification tasks by training a set of neural networks to predict the highest impacting transformation to be done on a dataset. These Multi-Layer Perceptron (MLP) classifiers each correspond to a transformation, allowing the classifier to predict whether the transformation will result in a more meaningful feature than the input feature, given the prediction target. LFE is trained on features represented by Quantile Data Sketch (QDS) for 10 unary transformations (log, square-root, frequency, square, round, tanh, sigmoid, isotonic regression, z-score, normalization) and 4 binary transformation (sum, subtraction, multiplication, division) using both the Random Forrest and Logistics Regression models. While LFE has obtained great results, it is limited in the transformation options it offers.

2.3 Holistic systems

Some systems and tools have taken a more holistic approach, with DataPrep (Peng et al., 2021) and Data Civilizer 2.0 (Rezig et al., 2019) being the most prominent ones. However, the data preparation steps provided by these systems are different from those provided by KGDataPrep, preventing their usage as a fair benchmark during experimentation.

2.3.1 DataPrep

DataPrep (Peng et al., 2021) is an Exploratory Data Analysis (EDA) tool for statistical modeling, which aims to facilitate the user's understanding the via data manipulation and visualization. DataPrep aims to be an interactive and easy to use tool providing a wide range of EDA tasks with different levels of granularity. DataPrep allows for data profiling and data cleaning by providing the option to identify missing values and drop them from the dataset. It also helps with feature selection by providing a correlation matrix of features selected by the user.

2.3.2 Data Civilizer 2.0

Data Civilizer 2.0's (Rezig et al., 2019) goal is to provide an end-to-end workflow for data cleaning and machine learning, as well as a debugger and a workflow visualization system. The system is divided into the three following components: The user-defined modules, a debugger, and a visualization tool, with the data cleaning and machine learning happening in the user-defined modules. This module allows for data profiling and data cleaning, in addition to providing data enrichment by allowing the user to joins and unions on the data.

2.4 Similarity Inclusion Dependency (sIND)

The concept of similarity inclusion dependency (sIND) is a novel approach to Inclusion Dependency (IND) detection proposed by (Kaminsky et al., 2023), which aims to identify Inclusion Dependencies (INDs) within real-world datasets. Unlike traditional methods for IND detection, which required clean data where dependencies held without exception, sINDs introduce a more flexible framework. They allow for the detection of INDs between columns even in the presence of

typos or variations in formatting, therefore removing the stringent constraints associated with other IND detection methods. This novel concept expands the applicability of IND detection techniques to potentially imperfect datasets, opening new avenues for data analysis.

2.4.1 SAWFISH

SAWFISH employs a meticulous preprocessing approach to prepare the data for analysis. It initially organizes all unique values within a column based on their length, sorting them from longest to shortest. Subsequently, pairs of values whose string sizes do not align are eliminated. These pairs would include any pair where the shortest value in the dependent column is smaller than the shortest value in the referenced column, minus the user-specified threshold.

SAWFISH then builds an inverted index for the referenced columns. It does so by dividing each value into strings of size T+1 where T is a user-specified threshold. This index facilitates efficient searching and retrieval of relevant information during the validation process. To validate the resulting dependent values, SAWFISH compares all substrings of equal length from the dependent column with the corresponding segment of the inverted index for the referenced column. This rigid and time-consuming validation process ensures that all dependencies between columns in the dataset are correctly identified.

Chapter 3

Linked Data Science Powered by Knowledge Graphs (KGLiDS)

Effective knowledge sharing among data scientists is crucial to both the development of individual data scientists and that of their teams. The lack of knowledge sharing amongst data scientists and its repercussions on their work, has created the need for an automated knowledge sharing system. The concept of linked data science has surfaced as a way to unify data science artifacts from various data science pipelines (DSP) into a cohesive framework by abstracting them. KGLiDS (Knowledge Graph-based Linked Data Science) (Helali et al., 2024) creates a knowledge graph by abstracting datasets and their related DSPs. KGLiDS aims to propel the field of data science forward by dismantling the barriers to knowledge sharing.

As it stands currently, data scientists often work in isolation, or within a small silo consisting of a few team members. This isolation is far from optimal when it comes to knowledge sharing. Work done by a data scientist or a data science team can be forever forgotten if it does not have an immediate business impact. This is not only detrimental to the individual data scientist, but also to the entire company or community. Having a tool to encourage or even automate knowledge sharing is a great advancement to the field.

In the quest to advance data science methodologies, the concept of linked data science (Helali et al., 2024) emerges as a new approach, aiming to forge an integrated framework that seamlessly

interconnects a myriad of data science artifacts. Linked data science represents a shift in paradigm, where datasets, pipeline scripts, and code libraries are no longer viewed as isolated parts, but rather as interconnected elements within a broader ecosystem. By capturing the semantics of these artifacts, linked data science facilitates the formation of novel connections, thereby fostering a richer and more comprehensive understanding of the subject area under investigation.

An essential benefit of linked data science lies in its abstraction of data science artifacts. Through the creation of abstractions for data or data science pipelines, experts from various domains can leverage each other's expertise based on the similarity of their data. This fosters a more enriched environment for learning.

Moreover, the abstraction consolidates the artifacts, simplifying their exploration for all users. Data scientists are no longer required to invest hours in opening multiple Excel files to discern the connections between various elements of a dataset. Instead, they can easily refer to the dataset abstraction, and obtain a significant amount of information. Creating an abstraction of data science artifacts also has the intended effect of making data manipulation scalable.

KGLiDS has been introduced as an innovative platform designed to facilitate the application of linked data science at scale. KGLiDS employs machine learning methodologies to extract valuable insights from data science artifacts, and subsequently create a knowledge graph. This knowledge graph, as seen in Figure 3.1 contains a homogeneous set of nodes and edges describing elements from a datasets such as its tables and columns, thus, embodying the concept of linked data science.

The three fundamental components in KGLiDS are Data Profiling, Knowledge Graph Construction, and Pipeline Abstraction. Data profiling provides insights into the charcteristics of the dataset. The knowledge graph simplifies the process of querying these characteristics and provides significant connection between columns and datasets. The pipeline abstraction, analyzes the pipeline provided in the notebooks associated with datasets and enriches the knowledge graph with information related to the data science pipeline.

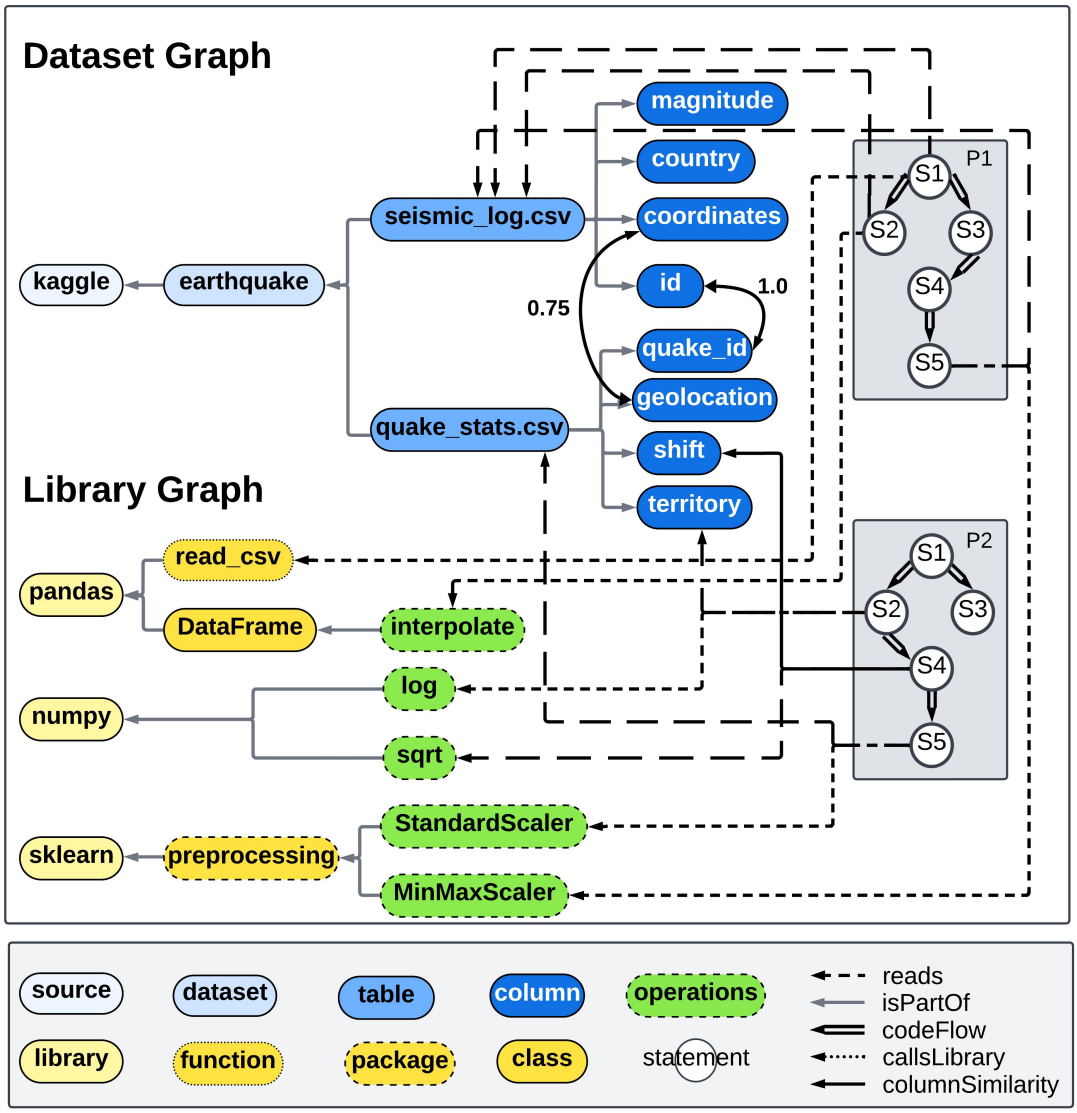

Figure 3.1: An example of the data science KG constructed using KGLiDS and augmented by our system with semantics related to data cleaning and transformation in green.

3.1 Data Profiling

The data profiler learns representations of columns and tables, and uses them to generate fixed-size embeddings based on a column's content and semantics. This profiler undertakes three primary tasks: gathering statistics pertaining to dataset columns, categorizing columns into seven distinct data types, and creating fixed-size column embeddings specific to each column.

For each column in the dataset, the following statistics are calculated: total count of the values, the count of distinct values, the count of missing values, the minimum, maximum, mean, and median

values of the column, as well as its interquartile range. These statistics provide an abstraction of the dataset, and allow for connections to be made between dataset without the need to store large amounts of data. Examples of the used of these statistics to make further connections can be seen in chapter 5.

KGLiDS also classifies columns into 7 fine-grained types: integers, floats, booleans, dates, named entities, natural language texts, and generic strings. Of these datatypes, the classification of the interger, float, boolean, and date types is done using pre-existing libraries such as the Numpy (Harris et al., 2020) and dateparser libraries. Named entities are detected using a pre-trained named entity recognition (NER) model (Peters, Ammar, Bhagavatula, & Power, 2017), allowing for the recognition of 18 different entity types. Natural language texts, which consists of texts such as product reviews, or comments, are predicted based on assessing the presence of corresponding word embeddings for their tokens in the fast text model (Bojanowski, Grave, Joulin, & Mikolov, 2016). All remaining string values are considered generic strings. Detecting the columns' data type is a foundational step for both the column embedding of the data profiler, as well as the graph generation portion of KGLiDS.

During profiling, column embeddings are created and stored as (300,1) fixed-sized arrays. These embeddings are created using pre-trained embedding models specific to each of the 7 fine-grained types. These embeddings are are fixed-sized abstractions of the column. This makes them valuable to be used to generate meaningful relationships between columns during knowledge graph generation, as well as for any other application requiring the identification of relationships between columns as seen in Chapter 5. A major advantage of these embeddings is their fixed size as it allows for a stable runtime regardless of dataset sized.

3.2 Knowledge Graph Construction

KGLiDS leverages the LiDS Ontology to ensure the construction of a standardized knowledge graph. This ontology contains all entities involved in a data science pipeline, including data, pipelines, and libraries, as well as the relationships amongst them. It contains 13 classes, 19 object properties, and 10 data properties, and employs the Web Ontology Language (OWL 2) and Uniform

Resource Identifiers (URLs) to facilitate the publication and sharing of the LiDS graph on the Web. Every entity in the ontology is provided with an RDF label and RDF type, which facilitates RDF reasoning on the LiDS graph.

Artifacts such as datasets, tables, columns, libraries, and pipeline scripts, are depicted as nodes in the knowledge graph. These nodes are interconnected by edges illustrating various relationships, including the hierarchical relationships among artifacts (ie: Column isPartOf Table), the similarity between columns (ie: Column1 columnSimilarity Column2 withCertainty X, where X is a value between 0 and 1), as well as the code flow between pipeline statements (ie: Statement1 hasNextStatement Statement 2).

These connections can be seen in Figure 3.1 where the dataset earthquake which was extracted from Kaggle contains the two tables 'seismic_log.csv' and 'quake_stats.csv', both of which have their own columns, including columns 'id' and 'quake_id'. All of the relationships between columns and their associated table are illustrated using the isPartOf edge. The relationship between the columns 'id' and 'quake_id', however, is illustrated via the edge columnSimilarity with the certainty of that similarity being 1.

Overall, the LiDS graph establishes an effective navigational structure and fosters meaningful connections among physical data science artifacts, empowering data scientists to explore, exchange, and learn from them more effectively.

Given the profiles for the columns generated during data profiling, an individual knowledge graph is constructed for each column of the dataset. This knowledge graph will contain the statistics of the column, its data type, as well as the dataset and table that it belongs to.

Given a columns data type and its column embeddings, as well as a user-defined similarity threshold, the column knowledge graph are linked to one another by their label and content similarities. Label similarities is the similarity between two columns based on their column names using GloVe Word embeddings (Pennington, Socher, & Manning, 2014) and a semantic similarity technique (Goikoetxea, Agirre, & Soroa, 2016). Content similarity is similarity between the embedding of column of the same data type. Both similarities have an associated score from 0 to 1 in the knowledge graph.

A higher similarity threshold set by the user results in less connections to be made. However,

the accuracy of the connections made will be higher. These connections can be used to discover important relationships between columns, as we shall see in the discussion of our similarity inclusion dependency detection technique in Chapter 5.

3.3 Pipeline Abstraction

Datasets from kaggle have many notebooks associated to them. The pipeline abstraction portion of KGLiDS can extract these notebooks from Kaggle and extract the operations used in them. The LiDS graph contains several nodes relating to the abstraction of the statements in the pipelines related to a dataset. These nodes can be libraries, packages, classes or functions. During the pipeline abstraction, the notebook containing the data science pipeline is processed by the statement. Each statement then becomes a node containing various edges such as hasNextStatement which shows the codeFlow of the pipeline from one statement to the next. The statements are then linked to the library or function they call via the callsLibrary and callsFunction edges. The isPartOf relationship between the libraries, packages, classes, and functions are extracted from each library's documentation allowing for the pipeline abstraction code to make these connections.

Figure 3.1 shows the application of several data science operations on columns and tables, such as the application of the "sqrt" operation from the numpy library on the column 'shift' in the pipeline 'P2', statement 'S2'.

The above mentioned pipeline abstraction lays the foundation necessary for building the cleaning and transformation models mentioned in 4.

Chapter 4

On Demand Data Preparation

Data preparation is the process of preparing raw data to be used for machine learning. It is a critical step in the machine learning process that has created a bottleneck in the data analysis process by taking 70-80% of data scientists' time (Abdallah et al., 2017). In other words, time that could be spend analyzing data and building models is being spent preparing the data for these tasks. Automating the data preparation process to free data scientists to do tackle the problems of data analysis and machine learning is the main motivation of this work.

There are various steps that can be used to prepare data for machine learning, chief amongst them are data cleaning and data transformation. Data cleaning involves any operation or series of operations identifying and correcting missing values or inconsistencies in datasets to improve their use in machine learning. The focus of the data cleaning process will be on missing value detection and imputation. Data transformation is the process of converting values in the dataset into more palatable formats for machine learning models. Several data cleaning and transformation operations are integrated in KGDataPrep's recommendation models.

Chapter 3 provided a basis for understanding the knowledge graph created by KGLiDS. This chapter, will elaborate in the use of Graph Neural Networks (GNN) for node prediction and how the information generated by the KGLiDS graph is used to create GNN models that will recommend data preparation steps in KGDataPrep.

Table 4.1: Specifications of the GNN models used for cleaning, scaling transformation and unary transformation used in KGDataPrep

Specification	Cleaning model	Scaling model	Unary model
Number of layers	1	1	1
Hidden Channels	64	128	64
Dropout	0.5	0.5	0.5
Learning rate	0.005	0.005	0.002
Epochs	30	30	15
Runs	3	3	3
Batch size	10000	10000	10000
Walk length	4	4	4
Number of steps	30	30	10

4.1 Graph Neural Networks (GNN)

Knowledge graphs, which are structured as heterogeneous graphs, represent a sophisticated way to model complex domains. In these graphs, nodes signify different classes or entities, while edges illustrate the relationships between them. Heterogeneous Graph Neural Networks (HGNNs) are great tools to analyze knowledge graphs. HGNNs excel in tackling a variety of challenges inherent to knowledge graphs, including node classification, entity alignment, and link prediction. Node classification involves assigning categories to nodes based on their features and the graph's structure. Entity alignment focuses on identifying and linking equivalent entities across different knowledge graphs, facilitating data integration and interoperability. Link prediction aims to infer missing relationships between entities.

Relational Graph Convolutional Networks (R-GCNs) (Schlichtkrull et al., 2018) are an application of HGNNs in both node classification and link prediction problems. By incorporating the relational structure of the graph into the learning process, R-GCNs improves the model's ability to generalize and make accurate predictions across different types of relationships and entities. This capability underscores the transformative potential of HGNNs in harnessing the full power of knowledge graphs for diverse analytical applications.

The RGCN pre-trained models presented in the following section are the building blocks of KGDataPrep. The specification for these models can be found in Table 4.1.

Table 4.2: Cleaning operations used and their associated libraries

Cleaning Operation	Library	Application
Fillna	Pandas	Fill the missing value with the mean in the case of numerical values and the most common value in the case of non-numerical values
SimpleImputer	Scikit-Learn	Impute the missing values using the most frequent value
KNNImputer	Scikit-Learn	Impute for missing values using the k-Nearest Neighbors
IterativeImputer	Scikit-Learn	Imputes missing values by modeling each column containing missing values as a function of other columns in the dataset in a round-robin fashion.
Interpolation	Pandas	Fill the missing value using the linear method for numerical values and the pad method for non-numerical values

4.2 Cleaning

Data cleaning is a fundamental step in the data preparation process. A major task in data cleaning is dealing with missing values, which are often present in most real world datasets. Ensuring that missing values are properly handled will ensure the integrity of the data and improve the results for any data analysis or machine learning that is required to be done on it. This section will delve into the anatomy of the KGLiDS graph used to train the cleaning model, expand on the training and inference of the the data cleaning GNN model, and the experiments performed to ensure its competitiveness with other SOTA systems.

4.2.1 Knowledge graph preparation

In order to build the GNN model for data cleaning, the KGLiDS graph for the training data needs to be built. This data consists of 1085 of of the top voted datasets in Kaggle, containing 3226 tables and 10349 pipelines. The KGLiDS graph is constructed to capture the relationships and features within the data. This graph encompasses various types of nodes and edges, representing essential elements and connections within the datasets.

The graph includes nodes such as “Table”, “Column,” and “Path”. which are inherent to the actual dataset. Additionally, it incorporates edges like “hasContentSimilarity”, “hasMissingValueCount,” and “isPartOf” to model the relationships within the dataset itself.

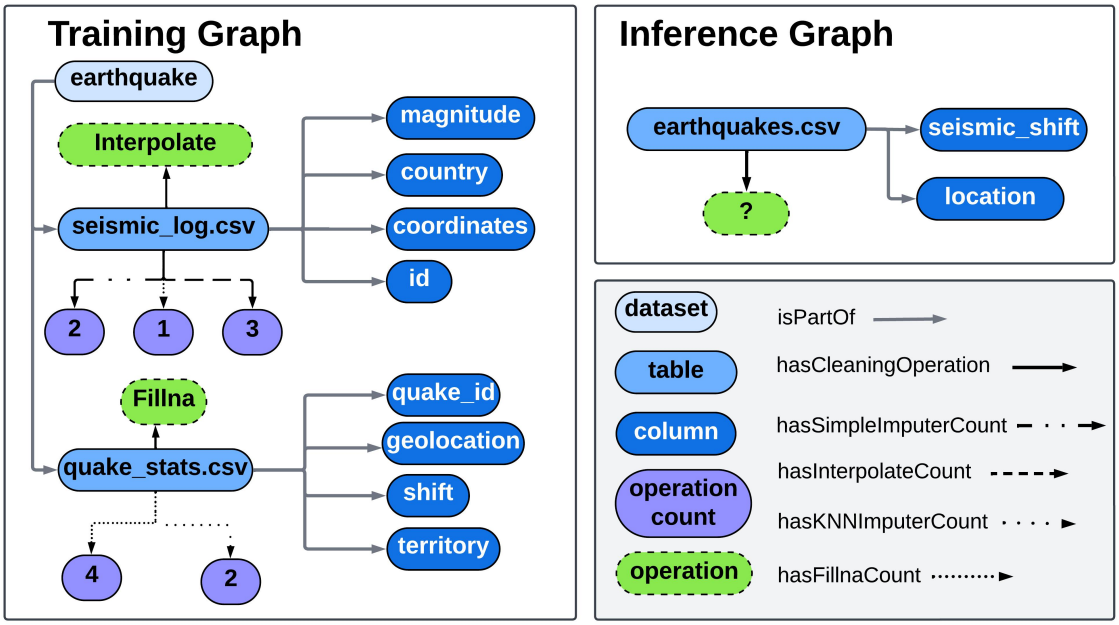

Figure 4.1: An example of the data science KG used for training and inference.

Furthermore, the graph is enriched with nodes such as “OperationCount”, “hasSimpleImputerCount” “hasKNNImputerCount”, and “hasFillnaCount”. These nodes are introduced through the profiling of notebooks associated with the datasets, and represent different data cleaning operations and techniques used in the data preprocessing phase.

Notably, different notebooks may employ distinct cleaning operations on the same table. To express this diversity, two types of triples are used to convey the cleaning operations applied to a table: “<Table><hasCleaningOperation><CleaningOperation>”, and “<Table><hasXCount><OperationCount>” where X is the cleaning operation being counted. An example can be seen in Figure 4.1 , where the triple “<quake_states.csv><HasCleaningOperation><Fillna>” indicates that the “Fillna” operation is the most commonly used cleaning operation for a specific table. This conclusion is reached by inspecting the triples involving this table and the object OperationCount. There exists 3 such triples: “<quake_states.csv><hasSimpleImputerCount><1>”, “<quake_states.csv><hasFillnaCount><3>”, and “<quake_states.csv><hasKNNImputerCount><2>”. Given that the predicate ‘hasFillnaCount” has the highest operation count, meaning that 3 out of the 6 notebooks related to the quake_state.csv table have used this operation in their cleaning process, “Fillna” is chosen as the optimal cleaning operation for this table.

4.2.2 Training

The task of data cleaning recommendation is formalized as a GNN node classification problem. Given a dataset $\mathcal{D}$ with missing values $\mathcal{M} = \{m_1, m_2, ..., m_n\}$ and a machine learning task $\mathcal{L}$, the goal is to predict the cleaning operation $c_r \in \mathcal{C}$ to handle $\mathcal{M}$ such that the accuracy of $\mathcal{L}$ is improved based on the cleaning techniques applied by other data scientists in other datasets similar to $\mathcal{D}$.

In the training phase, the kglids graph is encoded, with the target relation being 'HasCleaningOperation' and the label node designated as 'Table'. A Relational Graph Convolution Network (RGCN) is employed to build a model that will predict the node associated with the edge 'HasCleaningOperation' for a given table node.

The RGCN model is initialized using table embeddings, which are derived from kglids' column embeddings. These embeddings are calculated by averaging the embeddings of the columns in the table that contain missing values. Separate averages are computed for each of our fine-grained types(7) and then concatenated. As kglids' embeddings are of length 300, and boolean types do not have CoLR embeddings, the embeddings used to initialize the RGCN model are of length 1800. Figure 4.2 demonstrates the embedding creation process for the table quake_stats.csv where the embeddings for columns containing missing values with the same data type are averaged, and concatenated together, with the data types not represented in the table being given an embedding of 0. Given that the output of the model can be one of 5 cleaning operations seen in Table 4.2, the RGCN will have 1800 input channels and 5 output channels. The RGCN model will only have one layer as there is only one edge between a given table and its cleaning operation.

The resulting model is capable of predicting the node associated with a given table in the context of cleaning operations.

4.2.3 Inference

During inference, the models previously created are used to make prediction for unseen datasets. To do so, the dataset to be tested is converted to a graph using KGLiDS, before being encoded. The encoded graph and the GNN model previously trained are then used for inference. The results of the inference are then decoded before being output to the user.

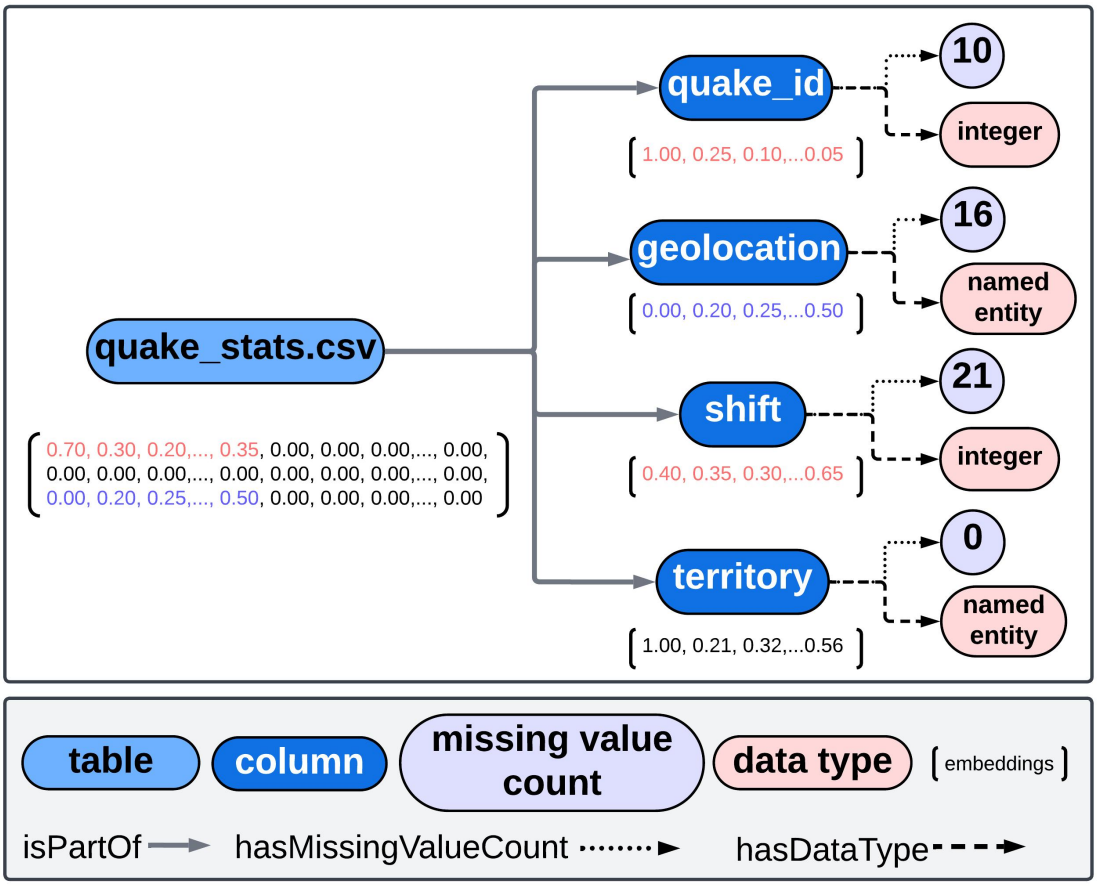

Figure 4.2: The embeddings of all columns containing missing values (quake_id, geolocation, and shift) are aggregated by data type and concatenated to create the table embeddings.

KGLiDS transforms the test datasets into a knowledge graph comprising of "Table" and "Column" nodes, connected by the "isPartOf" edge type. This knowledge graph is then encoded with the same encoding used during training.

The previously created RGCN model is loaded for inference. Given that the test dataset typically consists of a single table, the embedding for that table is calculated by averaging the embeddings of all columns with missing values belonging to the same fine-grained type and concatenating them. These embeddings are used to initialize the RGCN model.

The model's output is a value within the range of 0 to 4, which can then be decoded to determine one of the five possible cleaning operations applied to the table. This inference process allows for the identification of the relevant cleaning operation associated with the table within the given dataset. As seen in Figure 4.3, the top 3 predictions will be output in order of likelihood, giving the user the

	Cleaning Operation	Feature
0	Interpolate	[[gap, dmin, horizontalError, magError, nst, magNst]]
1	IterativeImputer	[[gap, dmin, horizontalError, magError, nst, magNst]]
2	SimpleImputer	[[gap, dmin, horizontalError, magError, nst, magNst]]

Figure 4.3: The output of the cleaning recommendation where the column 'Cleaning Operation' determines the recommended operation while the column 'Features' contains the name of the features said operation should be applied on.

option to use operations other than the top recommendation.

4.2.4 Experiments

This experiment provides a comparison of KGDataPrep's cleaning capabilities against Holoclean, the SOTA general cleaning platform. Holoclean uses statistical learning and inference to unify a variety of data cleaning methods. In order to create a fair testing environment, I used Aimnet (Wu et al., 2020), the most recent version of the Holoclean system. In this version, users are not required to specify a set of denial constraints, which helps put both systems on equal footing. Furthermore, only the *null detector* portion of Holoclean was used and the fine-tuning parameter were set at their default in accordance with Holoclean's GitHub repository [1].

To ensure a natural and varied pattern of missing values in the dataset, 8 datasets from an AutoML benchmark (Helali, Mansour, Abdelaziz, Dolby, & Srinivas, 2022) that contained missing values as well as five datasets from the UCI repository (Dua & Graff, 2017) were used to conduct this experiment. To evaluate the quality of KGDataPrep's data cleaning, the datasets were cleaned using KGDataPrep, as well as Holoclean and a baseline approach in which all rows with missing values are dropped. The cleaned datasets are then used for the machine learning task of classification using the Random Forest classifier and a ten fold cross-validation. The metric used to evaluate KGDataPrep's cleaning capabilities is the F1 scores of the random forest classifier. Given that the purpose of data cleaning in the context of data preparation is to improve the quality of the data used for machine learning, measuring the final machine learning score for the cleaned dataset is an optimal way of measuring the success of the data cleaning process. Furthermore, as an interactive

[1] https://github.com/HoloClean/holoclean/tree/latest-aimnet

data cleaning process is valuable, the systems are also evaluated in terms of their execution time and memory usage.

As shown in table 4.3 KGDataPrep's F1-scores are consistently better or comparable to those of Holoclean as well as the baseline method. Furthermore, the dataset size and its percentage of missing value does not hinder the completion of the cleaning task as can be the case for both the baseline method and Holoclean.

However, the main advantage of KGDataPrep is in its ability to perform as an interactive system as it outperforms Holoclean significantly in 85% of datasets in terms of execution time. This processing speed allows for data scientists to integrate my system within their data science pipeline without the need to account for the time required for the cleaning to be completed.

Furthermore, KGDataPrep outperforms Holoclean in 38% of datasets in terms of memory usage, as illustrated in Figure 4.4. Holoclean failed with out-of-memory errors while attempting to clean datasets $\#11$, $\#12$, and $\#13$, and was therefore unable to complete the cleaning process for these datasets with the existing RAM of 189GB. It is important to note that the largest dataset being tested, albert, has a size of 156 MB, presenting a genuine scalability challenge for Holoclean. This is due to Holoclean generating multiple tables containing dataset information throughout its cleaning process. These newly generated tables cause Holoclean's memory requirements to increase as the dataset size increases. In contrast, KGDataPrep's memory usage remains relatively stable regardless of the dataset size due to its models' fixed-size embeddings.

KGDataPrep's cleaning system provides competitive data cleaning, both in terms of the resulting machine learning score, as well as its seamless integration into an interactive data science pipeline due to its fast processing and low memory usage.

4.3 Transformation

Data transformation is the process of transforming data from one representation to another. There can be many motivations behind data transformation, such as changing the distribution of a feature to generate a more symmetric distribution, improve visualization, or the compatibility of the data with modeling processes (Abdallah et al., 2017). Unlike data cleaning, several data

Table 4.3: F1-Scores for Data Cleaning: The performance of our system vs Holoclean's Aimnet using multiple ML tasks on 13 datasets. Our system slightly outperforms Holoclean in small datasets and successfully completes the task for larger datasets while Holoclean encountered an out-of-memory (OOM) issue.

ID - Dataset	Baseline	Holoclean	KGDataPrep
1 - hepatitis	**69.76**	67.78	69.35
2 - horsecolic	00.00	82.28	**85.38**
3 - housevotes84	96.10	**96.64**	95.89
4 - breastcancerwisconsin	**97.43**	95.93	96.85
5 - credit	88.11	86.95	**88.17**
6 - cleveland_heart_disease	**28.31**	27.51	25.50
7 - titanic	70.68	81.89	**82.63**
8 - creditg	00.00	65.63	**66.63**
9 - jm1	**61.59**	60.55	61.55
10 - adult	79.15	78.49	**79.46**
11 - higgs	71.70	OOM	**71.73**
12 - APSFailure	**91.49**	OOM	90.89
13 - albert	00.00	OOM	**66.70**

transformation operations can be applied on the same data. Choosing the correct data transformation operation requires an understanding of the data distribution as well as the field the data is related to. In this section, I will discuss the anatomy of the KGLiDS graph used to train our transformation model, expand on the training and inference of the KGDataPrep's data transformation GNN model, and the experiments performed to ensure its competitiveness with other SOTA systems.

4.3.1 Knowledge graph preparation

The recommendation models for transformation have as their starting point the same 1085 datasets as the data cleaning model. KGLiDS is used to profile and create a knowledge graph. However, the transformation knowledge graph is enriched by profiling of notebooks associated with the datasets with nodes containing transformation operations such as "StandardScaler", "MinMaxScaler", "RobustScaler", or "sqrt" and "Log" and edges "HasUnaryOperation", "HasScalingOperation", "StandardScaler", "MinMaxScaler", "RobustScaler", "Sqrt", and "Log". The existence of two edges instead of one is due to the nature of data transformation as some data transformation operations are more suitable to be applied only to specific columns, hereafter called unary operations,

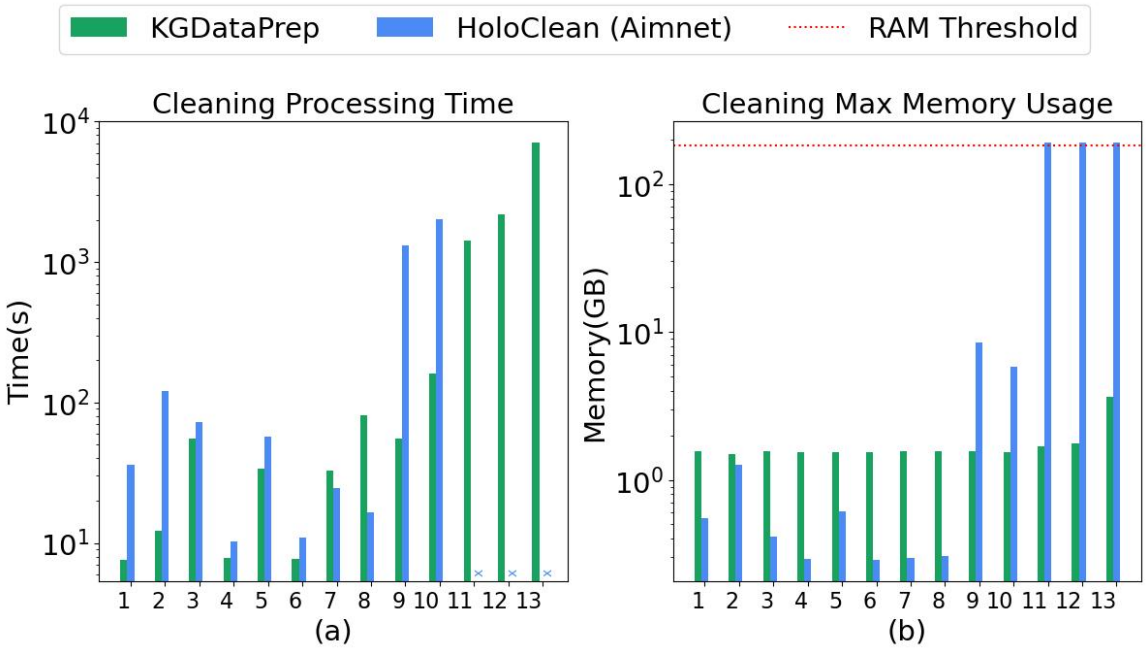

Figure 4.4: The performance of KGDataPrep vs Holoclean's Aimnet on the 13 datasets from 4.3. These results are obtained using a VM with 189 GB of RAM. Datasets are sorted by size in increasing order. (a) The X-axis represents the dataset ID, and the Y-axis is the time consumed by each system. KGDataPrep provides a processing time considerably lower than that of Holoclean. (b) The X-axis represents the dataset ID, and the Y-axis is the memory usage consumed by each system. KGDataPrep works with a near constant memory usage while Holoclean's memory usage changes significantly depending on the dataset.

while others tend to be applied to the entirety of the dataset, hereafter called scaling operation.

Notably, different notebooks may employ distinct transformation operations on the same table. To express this diversity, four types of triples are used to convey the transformation operations applied to a table. For instance, a triple like "<Table><HasScalingOperation><StandardScaler>" indicates that the "StandardScaler" operation is the most commonly used transformation operation for a specific table. In a similar fashion, a triple like "<Column><hasUnaryOperation><Sqrt>" indicates that the "Sqrt" operation is the most commonly used transformation operation for a specific column. Another triple, "<Table><hasStandardScalerCount><Value>" signifies the number of notebooks that applied the "StandardScaler" operation to the table, while "<Column><hasSqrtCount><Value>" indicated the number of columns on which Sqrt was applied.

Table 4.4: Transformation operations used and their associated libraries

Transformation Type	Transformation Operation	Library	Application
Scaling Operation	StandardScaler	Scikit-Learn	This function standardizes all features in the dataset by removing the mean and scaling them to unit variance.
	RobustScaler	Scikit-Learn	This function removes the median and scales all features according to their default quantile range of 25 to 75.
	MinMaxScaler	Scikit-Learn	This function transforms all features in the dataset by scaling them to the default range of 0 to 1.
Unary Operation	Sqrt	Numpy	This function applies the Sqrt transformer to the absolute value of all features in the dataset for which it has been recommended.
	Log	Numpy	This function applies the Log transformer to the absolute value of all features in the dataset for which it has been recommended.
	NoUnary	N/A	This function does not apply any Unary transformation on the features of the dataset for which it has been recommended.

4.3.2 Training

The task of data transformation recommendation was formalized as a GNN node classification problem. Given a dataset $\mathcal{D}$ with features $\mathcal{F} = \{f_1, f_2, ..., f_n\}$ and a machine learning task $\mathcal{L}$, the recommendation task is to predict a set of transformations $\mathcal{T} = \{t_1, t_2, ..., t_m\}$ to improve the accuracy of $\mathcal{L}$ based on the transformation techniques applied by other data scientists in the past on datasets and features similar to $\mathcal{D}$ and $\mathcal{F}$, respectively.

The graph used to create the GNN model contains the tables used for training, their features, including their columns, as well as the associated applied data transformation operations. These operations consist of scaling transformations provided in the *Scikit-learn* library: *StandardScaler*, *MinMaxScaler*, *RobustScaler* and common unary transformation operations from the *NumPy* and *Scikit-learn* libraries: *Log*, and *Sqrt*.

Scaling transformations are performed on the entire dataset, ensuring that all features are appropriately scaled. Therefore, to create the scaling recommendation model, the CoLR embeddings for all columns of the same data type in a table are averaged, and the averaged CoLR embeddings of all

data types are concatenated to create a comprehensive representation of the table. Unary transformations are performed on columns. Thus, the columns of a table are associated with their respective CoLR embeddings. The scaling and unary models are initialized with the table and column CoLR embeddings respectively and will in turn recommend a scaling transformation for tables and unary transformations for columns.

4.3.3 Inference

The inference phase is very similar to that of data cleaning. The dataset to be tested is transformed into a knowledge graph using KGLiDS. This graph is subsequently encoded.

The inference for the scaling and unary operations is done separately, as each have their own RGCN models. First, the scaling model is loaded. Given that the test dataset typically consists of a single table, the embedding for that table is calculated by averaging the columns of the same fine-grained data type and concatenating them. These embeddings are used to initialize the RGCN model. The model's output is a value within the range of 0 to 2, which can then be decoded to determine one of the three possible scaling operations applied to the table. This inference process allows for the identification of the relevant scaling operation associated with the table within the given dataset.

Once the inference for scaling is completed, the RGCN for unary operations is loaded. In this model, recommendation is for transformation operations to be applied on individual columns, therefore, the initialization is done using column embeddings. For each column, the output is a value from 0 to 2. The columns are then grouped by output and the numerical output values are mapped to a Sqrt operation, a Log operation or no unary operation at all.

The 3 scaling as well as the 3 unary predictions will be output in order of likelihood as show in Figure 4.5. This gives the the user the option to use operations other than the top recommended operation.

	Recommended_transformation	Recommendation	Feature
0	RobustScaler	rec1	All
1	Log	rec1	[magError, magNst]
2	NoUnary	rec1	[depth, depthError, dmin, earthquake_id, gap, horizontalError, latitude, longitude, magnitude, nst]
3	MinMaxScaler	rec2	All
4	NoUnary	rec2	[magError, magNst]
5	Sqrt	rec2	[depth, depthError, dmin, earthquake_id, gap, horizontalError, latitude, longitude, magnitude, nst]
6	StandardScaler	rec3	All
7	Log	rec3	[depth, depthError, dmin, earthquake_id, gap, horizontalError, latitude, longitude, magnitude, nst]
8	Sqrt	rec3	[magError, magNst]

Figure 4.5: The output of the transformation recommendation where the column 'Recommended_transformation' determines the recommended operation, the column 'Recommendation' contains the priority of the recommendation with rec1 being the top recommended operation, and the column 'Feature' contains the name of the features said operation should be applied on.

4.3.4 Experiments

I evaluated the transformation recommendation model to assess its effectiveness. To achieve this, I compared KGDataPrep's transformation recommendation model to the SOTA system, Autolearn (Kaul et al., 2017). This system was chosen as our comparison system of choice, as it was the most recent system offering transformation with an available codebase, allowing us to run experiments to compare both systems. Autolearn is a regression-based feature learning algorithm, as well as a baseline for which no data transformation was applied to the dataset. I evaluated the accuracy of the final machine learning task applied on the datasets, the processing time of the systems as well as their memory usage. The results from these comparisons situates our transformation recommendation model within the data transformation space.

I used 17 of the datasets from Autolearns experimental evaluation, which are available in the UCI repository (Dua & Graff, 2017), as seen in Table 4.5. Similar to the data cleaning evaluation, I applied the transformation techniques from KGDataPrep and Autolearn to the dataset before training and testing the data using a Random Forest 5-fold cross-validation classification task.

As the metrics used in the Autolearn paper was the accuracy, I also presented the results in terms of the accuracy of the final classification task. I did not manage to reproduce the reported results of Autolearn using the default parameters of its GitHub repository [2]. Therefore, both the published results from the Autolearn paper and the results I obtained by running the system are shown in Table 4.5. Based on the results in this table, KGDataPrep consistently matches or surpasses

[2] https://github.com/saket-maheshwary/AutoLearn/tree/master

Autolearn's accuracy.

I also measured the processing time required by each of these methods. To ensure interactivity, a three-hour time limit was imposed on the systems. KGDataPrep significantly outperformed Autolearn in execution time, particularly with larger datasets. While Autolearn timed-out while transforming these datasets, KGDataPrep successfully completed the transformation in a time frame orders of magnitude smaller.

Finally, the memory usage of both systems was compared. As shown in Figure 4.6, KGDataPrep maintains a stable memory usage as data size grows. This is due to its use of fixed-size embeddings. However, to create its features, Autolearn employs distance correlation to identify pairwise correlated features, and classifies them into linear and non-linear correlations. Due to this elaborate process, the original dataset's row and feature count, the inter-feature correlations, the created features, and the number of chosen features all impact Autolearn's memory usage. Hence, the dataset's absolute size is not the primary factor influencing memory usage in Autolearn's transformation. So while Autolearn has a lower memory usage for some datasets, it is difficult to predict its memory usage for a given dataset, while KGDataPrep has a near constant memory usage with a small increase for larger datasets.

I compared KGDataPrep to both the SOTA data trnsformation system Autolearn as well as a general baseline. The accuracy results of KGDataPrep are comparable or better than those of Autolearn. While the processing time and memory consumption of Autolearn can be smaller than ours, they are affected by many factors and difficult to predict. Overall, KGDataPrep's data transformation is largely advantageous in terms of the accuracy score and is likely to be more beneficial than its SOTA alternative for larger datasets.

4.4 APIs

KGDataPrep includes several Application Programming Interfaces (APIs). These APIs aid in the seamless integration of my data preparation models with any data science pipeline and enable data scientists to directly select and apply the recommended data cleaning operations to their dataset without requiring explicit code. APIs allowing for recommendation of a data cleaning operation,

Table 4.5: Accuracy for Data Transformation: The performance of our system as compared to AutoLearn on 17 datasets for machine learning classification tasks. Autolearn results are formatted as Y(X) where Y is the reported accuracy in Kaul et al. (2017) and X is the outcome of reproducing Autolearn experiments. TO signifies that Autolearn timed out in three hours, while OOM indicates that Autolearn ran out-of-memory.

ID - Dataset	Baseline	Autolearn	KGDataPrep
14 - fertility_Diagnosis	82.00	84.00 (**86.12**)	85.00
15 - haberman	68.63	65.34 (71.89)	**71.92**
16 - wine	96.07	97.20 (**98.33**)	97.17
17 - Ecoli	82.73	86.59 (81.23)	**88.10**
18 - pima diabetes	**75.37**	73.05 (75.13)	75.14
19 - Banke Note	99.05	99.56 (**99.93**)	98.91
20 - ionosphere	93.15	92.30 (**93.46**)	93.44
21 - sonar	73.55	77.87 (78.83)	**78.86**
22 - Abalone	22.91	22.21 (**24.96**)	24.56
23 - libras	71.94	70.22 (79.13)	**81.39**
24 - waveform	82.10	81.12 (TO)	**85.00**
25 - letter recognition	93.96	94.14 (TO)	**96.46**
26 - opticaldigits	96.38	96.57 (TO)	**98.10**
27 - featurepixel	95.5	94.20 (TO)	**97.65**
28 - shuttle	**99.97**	99.81 (TO)	99.96
29 - featurefourier	79.9	79.31 (TO)	**82.55**
30 - poker	68.1	72.26 (OOM)	**75.32**

application of a cleaning operation, as well as the recommendation and application of transformation operations are provided by KGDataPrep.

KGDataPrep aims to reduce or even remove the need for Exploratory Data Analysis (EDA) by data scientists by recommending options from the most common libraries. For this goal to be realistic, KGDataPrep needs to be easily integrated into Data SCience Pipelines (DSPs). My recommendation APIs provide options from the most common libraries in a itemized manner from the most likely to least likely. My application APIs allow the data scientist to apply not only the highest recommendation but any of the top three recommendations so the user can compare the results if need be. This results in an easily integrable system.

When asking for a cleaning operation, the API:

```
cleaning_ops = recommend_cleaning_operations(df)
```

can be used. When an unseen dataset is provided as a DataFrame, KGLiDS's Column Learned

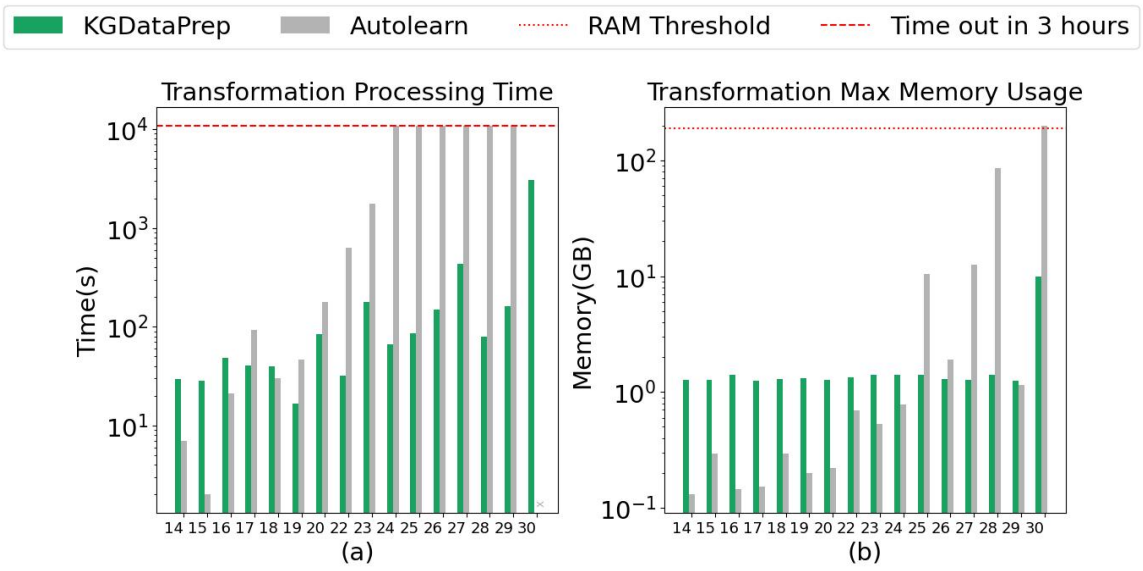

Figure 4.6: The performance of KGDataPrep vs Autolearn on the 17 datasets from 4.5. These results are obtained using a VM with 189 GB of RAM. Datasets are sorted by size in increasing order. (a) The X-axis represents the dataset ID, and the Y-axis is the time consumed by each system. KGDataPrep provides a processing time considerably lower than that of Autolearn, with Autolearn timing out in several datasets. (b) The X-axis represents the dataset ID, and the Y-axis is the memory usage consumed by each system. KGDataPrep works with a near constant memory usage while Autolearn's memory usage changes significantly depending on the dataset and is unable to complete the task for one dataset due to an OOM error.

Representation (CoLR) embeddings are calculated for each column of the dataframe. They are then aggregated by data type of missing columns of missing values and the resulting embeddings are concatenated. The resulting embedding is used to initiate the task-specific pre-trained GNN model built for data cleaning. A dataframe similar to that shown in Figure 4.3 is output as the value for the cleaning_ops variable. The recommended cleaning operations are sorted in this dataframe from the most highly recommended operation to the least likely recommendation. The recommended operation chosen by the user can be input into the application API by specifying its row in the cleaning_ops dataframe.

The API used to apply the recommendations to the original dataframe is:

```
apply_cleaning_operations(cleaning_ops.loc[0],df)
```

This API takes the base dataframe and selected cleaning operation as input and returns the cleaned dataset as output.

Similar APIs exist for data transformation. The recommendation API for transfomration is:

```
transformation_ops = recommend_transformation_operations(df)
```

Given an unseen dataset, the COLR embedding for each column in the dataset is calculated. To allow for the Scaling recommendation, the COLR embeddings are aggregated by data type and the resulting embeddings are concatenated. As scaling operations are applied on the dataset as a whole, the top three recommendations are sorted. The order of the recommendations can be seen in the column 'Recommendation' of the dataframe with rec1 being the top recommendation for a transformation category and column combination, rec2 being the second recommendation, ect. As seen in Figure 4.6, The top scaler recommendation is RobustScaler which should be applied to all features, while the top unary recommendation for the columns magError and magNst is Log transformation and the top unary recommendation for all other columns of the dataset is to not have any unary transformation.

The API used to apply transformation operations to a dataset is:

```
apply_transformation_operations(transformation_ops.loc[0],df)
```

The user can choose to apply any or all the top recommendations (in the example in Figure 4.6, that would be applying RobustScaler to all columns and Log to columns magError and magNst) or apply a mix and match of the recommended transformations, such as applying the top recommendation for the scaling operations (RobustScaler) but the third recommendation for the Unary transformations (Sqrt for columns magError and magNst and Log for the remaining columns).

Chapter 5

Similarity Inclusion Dependency (sIND) Detection

5.1 Introduction

The concept of sIND was introduced in the paper "Discovering Similarity Inclusion Dependencies" (Kaminsky et al., 2023) as a means to detect inclusion dependencies IND detection system is motivated by the assumption that real world data will be dirty due to typos, misspellings, or other human errors. They introduce their sIND detection algorithm: Similarity AWare Finder of Inclusion dependencies via a Segmented Hash-index (SAWFISH). While this paper reports great results, I decided to take a different approach at solving the same problem. KGDataPrep's sIND detection system uses features queried from the KGLiDS graph to create models capable of sIND detection. This chapter provides a deeper exploration of SAWFISH and the motivations and steps used to develop a new sIND detection model.

SAWFISH, aims to detect all sINDs in a dataset, by looking for values that are similar. Similarity is defined by the user via the edit distance parameter. Any two values having a Levenshtein distance smaller than the user-defined edit distance are considered similar. Although SAWFISH initially uses heuristics to prune the list of potential sIND pairs, it eventually resorts to a pairwise calculation of the Levenshtein for the shortlisted pairs. Depending on the dataset, the resulting shortlist may vary in size, with longer potential sIND pair lists requiring a longer processing time for the

pairwise Levenshtein distance comparisons. The variation in the calculation time of the SAWFISH methodology depending on the size and content of the dataset is the motivation in developing a new sIND detection model.

KGDataPrep's sIND detection system forgoes a pairwise similarity comparison in favor of a more heuristic approach. I used the KGLiDS graph mentioned in Chapter 3 as a foundation for this work. This graph creates an abstraction of the data which it then stores in a knowledge graph. KGDataPrep queries this graph to obtain a pool of column pairs deemed to have a similar content. Using these pairs as a starting point, features that are indicative of a IND relationship are also queried from the graph and are used to build KGDataPrep's sIND detection models.

Using heuristics to detect sIND pairs and using the KGLiDS graph to create features related to the datasets provides an advantage in regards to computational speed. KGLiDS' use of a fixed sized column embedding via data sampling and its use of PySpark to enable a distributed computation results in a fast graph generation. Furthermore, the heuristics used for feature extraction bypass the need for a pairwise comparison, further improving the speed of sIND detection for KGDataPrep.

5.2 Implementation

Inspired by the paper "A Machine Learning Approach to Foreign Key Discovery" (Rostin, Albrecht, Bauckmann, Naumann, & Leser, 2009), several features were reviewed, tested, and modified to be adapted to help with a sIND detection task. Given the importance placed on the time required to complete the sIND detection task, the features selected were required to be easily obtained by querying the previously generated KGLiDS graph. Two different models were then trained on the generated features to create KGDataPrep's final sIND detection model.

5.2.1 Features

The main part of sIND detection was developing a model that could take in the content similarity pairs detected by the KGLiDS' knowledge graph and prune them to find the sIND pairs. To do so, I extracted four features from the KGLiDS knowledge graph and used them to develop our model. These four features are:

Distance between column names (N1, N2): This feature examines the relationship between column names N1 and N2, respectively belonging to columns C1 and C2. A similarity in the column names can be an indication of a similarity in the values of their columns. To measure the similarity of the column names, the normalized Levenshtein distance is calculated. The Levenshtein distance calculates the smallest number of edits needed in order to transform string N1 to string N2. This value is then normalized by dividing it by the longest of the two strings.

Content Similarity (C1,C2): This feature measures the content similarity between the two columns C1 and Col2. It can be found by querying the KGLiDS graph where C1 and C2 nodes are linked via the <data:hasContentSimilarity>edge as seen in Figure 3.1. The value of the content similarity is calculated using the normalized Euclidean distance between the CoLR embedding representations of the two columns. This feature shows the measure of similarity between two columns, assigning a higher similarity score to columns with a higher similarity. These scores, in turn help with the learning of the models as columns with a higher similarity are more likely to be sIND pairs.

Cardinality (C1): This feature exhibits the cardinality of the referenced column (C1) using the <data:hasDistinctValueCount>predicate in the KGLiDS graph. The KGLiDS profiler counts the number of distinct values present in the referenced column, C1. A higher cardinality indicates a higher number of unique values in the referenced column, C1. This relationship holds true as referenced columns often have more unique values than their dependent columns.

Table size ratio (C1,C2): This feature calculates the ratio of the number of rows in the dependent column (C2) to the number of rows in the referenced column (C1) using the ⟨data:hasTotalValueCount⟩ relationship in the KGLiDS graph. This value is obtained by the profiler by measuring the length of each column and stored in the KGLiDS graph. Dependent rows are expected to have a higher number of rows than their referenced columns as they will often repeat a value more than once. Therefore, the higher the table size ratio feature, the higher the probability of a column pair being a sIND pair.

A summary of the above features and their implementation can be seen in Table 5.1.

Table 5.1: Features used to build the sIND pair detection models and their associated KGLiDS predicates

Feature	Feature implementation in KGLiDS
Column Name Distance	None
Content Similarity	data:hasContentSimilarity
Cardinality	data:hasDistinctValueCount
Table Size Ratio	data:hasTotalValueCount

5.2.2 Modeling

Depending on the relative number of sIND paire with respect to the size of the tables the sIND pairs belong to, the problem of sIND pair detection can be classified as a binary classification problem or an outlier detection problem.

In the binary classification approach, given a dataset $\mathcal{D}$ consisting of n data points $\{x_1, x_2, \ldots, x_n\}$, each associated with a label $y_i \in \{0, 1\}$, where 1 implies a sIND pair and 0, a non-sIND pair, the objective is to learn a function $f : \mathcal{X} \rightarrow \{0, 1\}$ that maps an input $x \in \mathcal{X}$ to a predicted label $\hat{y} \in \{0, 1\}$.

The binary classification model used to build KGDataPrep's sIND detector is sklearn's RandomForestClassifier [1]. It is a combination of multiple decision trees to create a more accurate model. The training data is divided into subgroups which are used to train each tree. During the inference phase, each tree independently predicts a class for their given sample. The final prediction is the aggregation of all the trees. The metric used for our model is the mean predicted probability of a sIND pair based on the trees in the forest.

In the outlier detection approach, given a dataset $\mathcal{D}$, with n data points $\{x_1, x_2, \ldots, x_n\}$, the goal is to identify a subset of data points $\mathcal{O} \subseteq \mathcal{D}$ such that the points in $\mathcal{O}$ are considered outliers or anomalies.

The outlier detection model used to build KGDataPrep's sIND detector is sklearn's IsolationForest [2]. This model returns the anomaly score of each column pair by recursively dividing the values in a particular feature. The number of times this partitioning is required in order to isolate a

[1] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

sample is indicative of the normality of the sample with a shorter path indicating a higher anomaly score for that sample. The contamination parameter for this class denotes the estimated proportion of outliers in the dataset according to the user.

While for datasets with a larger sIND pair to dataset size a binary classification approach was successful, this approach failed to successfully detect sIND pairs in larger dataset with a lower sIND ratio. The outlier detection approach was therefore implemented in its place. However, the outlier detection approach failed to detect the sIND pairs in datasets where they were more frequent, despite its contamination ratio being increased. This resulted in a combination of both approaches, where both the probability score of the binary classification model and the anomaly score of the outlier detection model are calculated and the highest one is selected for KGDataPrep's sIND detection model. This approach favors the classification model for datasets with a higher concentration of sIND pairs and the outlier detection model for datasets with a lower concentration of sIND pairs.

5.2.3 API

To obtain the sIND pairs, the API: get_sIND(database) can be used where the input to the function is the database for which we want the sIND pairs. This function will look for a knowledge graph by the same name as the database and will construct one if one does not exist. It will then use the knowledge graph to build the features mentioned above and use our model to detect the sIND pairs. The output of the function will be the detected sIND pairs.

5.3 Experiments

To validate the effectiveness of KGDataPrep's sIND detection technique, experiments were conducted comparing it to SAWFISH both in terms of its ability to detect sIND pairs accurately as well as its scalability when applied to large datasets. In both aspects, KGDataPrep's sIND detection technique proved itself to be a viable and effective technique, surpassing its SAWFISH counterpart.

5.3.1 sIND detection

The datasets used in this experiment were the largest datasets from the Prague Relational Learning Repository (Motl & Schulte, 2024). These datasets were all originally clean, with no typos or errors. SAWFISH with a threshold of 0 (Levenshtein distance of 0 between the values being compared) was ran on these datasets to find the IND pairs in each dataset. These pairs are the baseline used throughout the following experiments. The datasets are then injected with errors. In two sets of experiments shown in Figure 5.1 5% and 10% of the value in each table are set to be erroneous. The extent of the injected errors is determined by the edit distance threshold set, where an edit distance threshold of 1 signifies that the errors introduced will result in a maximum Levenshtein distance of 1 between the original value and its erroneous counterpart. To explore the effects of different error levels, dataset with edit distance threshold of 1, 2, or 3 were created. Using the above techniques, a total of six sets of datasets were created, with two sets of three datasets. The first set contains datasets with a 5% induced error and edit distance thresholds of 1, 2, and 3, while the second set's datasets have a 10% induced error with the same edit distance thresholds.

These datasets are used to test KGDataPrep's sIND detection by comparing the sIND pairs detected in the erroneous datasets with the IND pairs detected in the clean data. To test SAWFISH, its edit distance threshold parameter was adjusted to the edit distance threshold introduced in the dataset (A edit distance threshold of 1 was set for SAWFISH when testing the dataset with an edit distance threshold of 1). For KGDataPrep, no adjustments were required to account for the different edit distances as KGDataPrep's models are built on clean datasets.

My findings correspond with the findings declared in the SAWFISH paper. Increasing the Levenshtein distance threshold in the SAWFISH algorithm improves its ability to detect sIND pairs. However, its runtime increases exponentially in the process. Meanwhile, KGDataPrep maintains a near constant runtime as well as a good and stable ability to detect sIND pairs. It can also be noted that the percentage of introduced error does not affect either system in terms of runtime or their ability to detect sIND pairs.

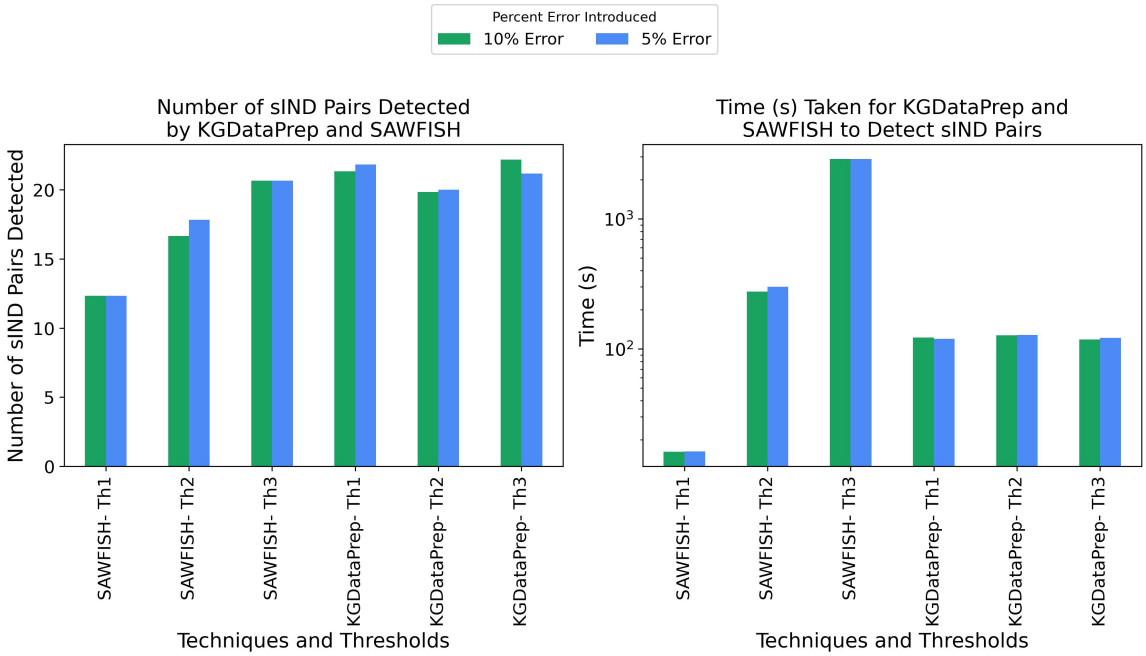

Figure 5.1: (a) The number od sIND pairs correctly detected by SAWFISH and KGDataPrep with all combination 0f 5% and 10% induced error and edit distand thresholds of 1 to 3. SAWFISH's performance improves with an increase in threshold while ours remains relatively constant. (b) The time required for sIND detection by SAWFISH and KGDataPrep with all combination 0f 5% and 10% induced error and edit distand thresholds of 1 to 3. SAWFISH's processing time increases exponentially with an increase in threshold while KGDataPrep's remains relatively constant.

5.3.2 Time scaling

To further investigate the effects of a dataset's size on the sIND detection time, the TPCDS dataset, a common scalable benchmark, was used. The dataset was generated at sizes: 0.1GB, 1GB, and 5GB. The use of a scalable benchmark dataset was necessary to truly be able to investigate the processing time, as in these datasets, the number of IND pairs will remain the same regardless of their size.

Both the SAWFISH sIND detection technique and KGDataPrep's were applied on each dataset variation. Given that the focus of this experiment was the scaling time, the similarity threshold was set to zero to facilitate the experiment. As observed in Figure 5.2, while KGDataPrep's detection time is nearly constant, a steep increase can be seen in Sawfish's time. KGLiDS creates an embedding based on a sampling of the data, allowing for a near constant time for sIND detection regardless of dataset size, while SAWFISH's last step requires a pairwise comparison of values in suspected sIND pairs, which causes an increase in processing time as the dataset size increases.

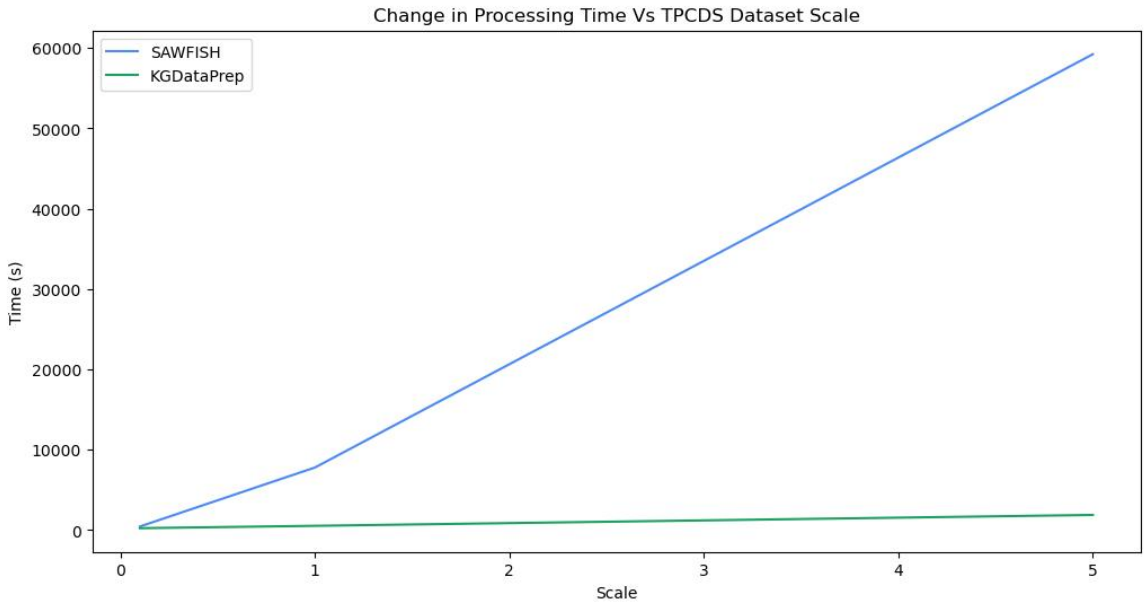

Figure 5.2: The measure of time in seconds required for Sawfish and KGDataPrep models to detect sIND. While a larger scaling factor for the TPCDS dataset results in an increase in the time required for sIND detection, the time requirement increase for KGDataPrep is insignificant due to the data sampling strategy used in KGLiDS.

In this chapter, I introduced the concept of sIND detection, reviewed the previous work done on this problem and presented our own work aiming to solve it. KGDataPrep's sIND detection system was evaluated alongside the SOTA algorithm, SAWFISH, and showed good results both in terms of the number of sIND pairs detected and the processing time of the system.

Chapter 6

Conclusion and Future Work

In this thesis I set to present a new data preparation system including a data cleaning, data transformation and sIND detection system. I introduced a knowledge graph for linked data science, KGLiDS as the foundation on which these systems are built. I elaborated on how the GNNs used in our on demand data preparation systems are built on top of the KGLiDS graph, and explored how these systems are integrated into a DSP and their competitiveness with existing SOTA systems. I presented the topic of sIND detection and illustrated how such a system can be built by using the KGLiDS graph as its foundation.

KGLiDS provides a means to map relational datasets and their associated pipelineinto a knowledge graph, expanding what could traditionally be done with such datasets. It creates and abstraction of the datasets and their pipelines, which makes them more scalable and shareable. This abstraction is done by converting elements of datasets, libraries, and pipeline scripts into nodes which can be linked via up to 29 different predicates. The KGLiDS graph provides a strong foundation for my systems to be built on.

The on demand data preparation systems are GNN models built on the KGLiDS graph. They aim to facilitate the data cleaning and transformation processes by providing access to the GNN models via their own APIs. Both the cleaning and transformation models' performances are comparable or better to that of the SOTA systems used in my experiments. However, they are consistently faster than their counterparts and provide a more stable memory consumption, making them more desirable for use in a DSP.

While these systems provide a great advancement in the field of data preparation, there remains several steps in the data preparation pipeline that could also benefit from having such models, such as data visualization or feature selection.

While sIND detection is a newly posed problem, I have presented models that can detect sIND pairs as well as the methodology to test these models. my sIND detection system uses features queried from the KGLiDS graph to build sIND detection models. It also provides an API to use these models on unseen datasets. My system can find more sIND pairs than its SOTA counterpart, Sawfish, regardless of the error percentage or the extent of error within a dataset. Furthermore, it can do so at a stable pace, without being affected by the error threshold. These results were obtained by introducing errors into clean datasets in a controlled manner and comparing the sIND detection to that of the original dataset. Future work could explore the possibility of using this sIND system as a foundation to build a primary-key foreign-key detection system. Primary-key and foreign-key pairs are used for data enrichment.

Further work in this area could be to use GNN models for the sIND detection. GNN models could be beneficial as they will take into consideration other features of a column, providing the model with a more thorough understanding of the columns deemed identical. An exploration into building a primary key-foreign key detection system around the sIND detection to allow for automated data enrichment would also be of value.

This work has contributed to the automation of a portion of the DSP by providing an on demand data preparation system recommending cleaning and transformation. It has also set the stage for further contributions by developing a sIND detection technique. These contributions have all performed well when compared to their SOTA counterparts and can easily be integrated into any DSP using their APIs.

Appendix A

Master's Coursework and Contributions

A.1 Master Coursework

Table A.1: Course work

Course	Course Code	Semester	Grade
NATURAL LANGUAGE ANALYSIS	COMP6751	Fall 2022	A
DEEP LEARNING	COMP691	Winter 2023	A-
BIG DATA ANALYTICS	SOEN6111	Winter 2023	A+
FOUNDATIONS/SEMANTIC WEB	COMP6531	Winter 2024	A+

A.2 Publications

Helali, M., Monjazeb, N., Vashisth, S., Carrier, P., Helal, A., Cavalcante, A., Ammar, K., Hose, K., Mansour, E. (2024). Kglids: A platform for semantic abstraction, linking, and automation of data science.

References

Abdallah, Z. S., Du, L., & Webb, G. I. (2017). Data preparation. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of machine learning and data mining* (pp. 318–327). Boston, MA: Springer US. Retrieved from https://doi.org/10.1007/978-1-4899-7687-1_62 doi: 10.1007/978-1-4899-7687-1_62

Biessmann, F., Rukat, T., Schmidt, P., Naidu, P., Schelter, S., Taptunov, A., ... Salinas, D. (2019). Datawig: Missing value imputation for tables. *Journal of Machine Learning Research*, *20*(175), 1–6. Retrieved from http://jmlr.org/papers/v20/18-753.html

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Dua, D., & Graff, C. (2017). *UCI machine learning repository.* Retrieved from http://archive.ics.uci.edu/ml

Goikoetxea, J., Agirre, E., & Soroa, A. (2016). Single or multiple? combining word representations independently learned from text and wordnet. In *Proceedings of the thirtieth conference on artificial intelligence (AAAI)* (pp. 2608–2614). Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11777

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020, September). Array programming with NumPy. *Nature*, *585*(7825), 357–362. Retrieved from https://doi.org/10.1038/s41586-020-2649-2 doi: 10.1038/s41586-020-2649-2

Helali, M., Mansour, E., Abdelaziz, I., Dolby, J., & Srinivas, K. (2022). A scalable automl approach based on graph neural networks. *Proceedings of the VLDB Endowment*, *15*(11), 2428-2436.

doi: 10.14778/3551793.3551804

Helali, M., Monjazeb, N., Vashisth, S., Carrier, P., Helal, A., Cavalcante, A., ... Mansour, E. (2024). *Kglids: A platform for semantic abstraction, linking, and automation of data science.*

Kaminsky, Y., Pena, E. H. M., & Naumann, F. (2023). Discovering similarity inclusion dependencies. *Proceedings of the ACM on Management of Data*, *1*, 1-24. doi: 10.1145/3588929

Kaul, A., Maheshwary, S., & Pudi, V. (2017). Autolearn - automated feature generation and selection. In V. Raghavan, S. Aluru, G. Karypis, L. Miele, & X. Wu (Eds.), *2017 IEEE international conference on data mining, ICDM 2017, new orleans, la, usa, november 18-21, 2017* (pp. 217–226). IEEE Computer Society. Retrieved from https://doi.org/10.1109/ICDM.2017.31 doi: 10.1109/ICDM.2017.31

Mansour, E., Srinivas, K., & Hose, K. (2022, jan). Federated data science to break down silos [vision]. *SIGMOD Rec.*, *50*(4), 16–22. Retrieved from https://doi.org/10.1145/3516431.3516435 doi: 10.1145/3516431.3516435

Motl, J., & Schulte, O. (2024). *The ctu prague relational learning repository.*

Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E. B., & Turaga, D. S. (2017). Learning feature engineering for classification. In C. Sierra (Ed.), *Proceedings of the twenty-sixth international joint conference on artificial intelligence* (pp. 2529–2535). Retrieved from https://doi.org/10.24963/ijcai.2017/352

Peng, J., Wu, W., Lockhart, B., & et al. (2021). Dataprep.eda: Task-centric exploratory data analysis for statistical modeling in python. In *Sigmod, 2021* (pp. 2271–2280). ACM.

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543). Retrieved from https://doi.org/10.3115/v1/D14-1162

Peters, M., Ammar, W., Bhagavatula, C., & Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the association for computational linguistics (acl)* (Vol. 1, pp. 1756–1765). Retrieved from https://doi.org/10.18653/v1/P17-1161

Rezig, E. K., Cao, L., Stonebraker, M., Simonini, G., Tao, W., Madden, S., ... Elmagarmid, A. K.

(2019). Data civilizer 2.0: A holistic framework for data preparation and analytics. *Proc. VLDB Endow.*, *12*(12), 1954–1957. Retrieved from http://www.vldb.org/pvldb/vol12/p1954-rezig.pdf doi: 10.14778/3352063.3352108

Rostin, A., Albrecht, O., Bauckmann, J., Naumann, F., & Leser, U. (2009). A machine learning approach to foreign key discovery. In *International workshop on the web and databases.* Retrieved from https://api.semanticscholar.org/CorpusID:11636431

Schlichtkrull, M., Kipf, T. N., Bloem, P., vanÂ den Berg, R., Titov, I., & Welling, M. (2018). Modeling relational data with graph convolutional networks. In A. Gangemi et al. (Eds.), *The semantic web* (pp. 593–607). Cham: Springer International Publishing.

van Buuren, S., & Groothuis-Oudshoorn, K. (2011). mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, *45*(3), 1-67. doi: 10.18637/jss.v045.i03

Wu, R., Zhang, A., Ilyas, I. F., & Rekatsinas, T. (2020). Attention-based learning for missing data imputation in holoclean. In *Conference on machine learning and systems.* Retrieved from https://api.semanticscholar.org/CorpusID:211482719