Linked Data Science powered by Knowledge Graphs

Mossad Helali*, Shubham Vashisth*, Philippe Carrier*, Katja Hose§, Essam Mansour*
*Concordia University, Canada §Aalborg University, Denmark
{first}.{last}@concordia.ca khose@cs.aau.dk

ABSTRACT

In recent years, we have witnessed a growing interest in data science not only from academia but particularly from companies investing in data science platforms to analyze large amounts of data. In this process, a myriad of data science artifacts, such as datasets and pipeline scripts, are created. Yet, there has so far been no systematic attempt to holistically exploit the collected knowledge and experiences that are implicitly contained in the specification of these pipelines, e.g., compatible datasets, cleansing steps, ML algorithms, parameters, etc. Instead, data scientists still spend a considerable amount of their time trying to recover relevant information and experiences from colleagues, trial and error, lengthy exploration, etc. In this paper, we therefore propose a novel system (KGLiDS) that employs machine learning to extract the semantics of data science pipelines and captures them in a knowledge graph, which can then be exploited to assist data scientists in various ways. This abstraction is the key to enable Linked Data Science since it allows us to share the essence of pipelines between platforms, companies, and institutions without revealing critical internal information and instead focusing on the semantics of what is being processed and how. Our comprehensive evaluation uses thousands of datasets and more than thirteen thousand pipeline scripts extracted from data discovery benchmarks and the Kaggle portal, and show that KGLiDS significantly outperforms state-of-the-art systems on related tasks, such as datasets and pipeline recommendation.

ACM Reference Format:
Mossad Helali*, Shubham Vashisth*, Philippe Carrier*, Katja Hose§, Essam Mansour*. 2023. Linked Data Science powered by Knowledge Graphs. In *Proceedings of the 2023 International Conference on Management of Data (SIGMOD '23).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Data science is the process of collecting, cleaning, and analyzing structured and unstructured data to derive insights or to build models for predicting particular tasks. To achieve these goals, data science is founded on datasets and the pipelines built on top of them. In recent years, we have witnessed a growing interest in data science not only from academia but particularly from companies owning colossal amounts of data and investing in data science platforms to help develop and enhance pipelines analyzing these datasets for various tasks. Furthermore, open data science and collaborative portals, such as Kaggle and OpenML, are expanding in popularity as well; the Kaggle portal, for instance, contains tens of thousands of open datasets and hundreds of thousands of associated data science pipelines[1]. Despite the importance of efforts and their large-scale nature, existing systems do not offer a holistic approach but instead consider data science artifacts, e.g., datasets and pipeline scripts, only in isolation from each other; there is little or no support to help users learn from the experiences and best practices connecting datasets and pipelines.

A data scientist building a pipeline, for instance, is generally interested in datasets relevant to the task at hand, whether on open data portals or within their enterprise, as well as in previously built pipelines using these datasets. Integrating related datasets builds upon traditional approaches for data enrichment (e.g. by having more rows or columns) and data integrity checks, and allows not only to fulfill the initial task but can also enable a broader range of analyses by introducing additional dimensions and perspectives. The associated data science pipelines then allow the data scientist to benefit from the accumulated (public or enterprise) domain knowledge about the datasets, which helps accelerating the development and discovery of new solutions and possibilities. Unfortunately, existing portals and platforms have no or little support to assist data scientists in easily exploring and systematically discovering data science artifacts. Instead, existing platforms offer only isolated and limited support for dataset search or pipeline recommendation.

To overcome this lack of support and to allow for the understanding and maintenance of large repositories of pipelines, a number of techniques [1, 4, 18] has been proposed to provide machine-readable semantic abstractions of software code. However, the majority of these systems targets statically-typed languages like Java, where accurate static code analysis is feasible. However, the vast majority of data science pipelines is not written in Java but in Python[2], which makes these approaches inapplicable. Moreover, even existing approaches for Python code [1] resort to using general-purpose static analysis techniques although for a dynamic language, such as Python, accurate static analysis is challenging or even infeasible in some cases [29]. Hence, there is a need for approaches tailored specifically to the requirements of data science pipelines in Python and their particularities.

What existing platforms, such as Google Dataset Search, or enterprise data catalog tools, such as Amundsen[3], offer is is a rudimentary form of searching for relevant datasets based on keyword matching. Hence, these systems do not allow discovering datasets based on certain characteristics or similar datasets given a sample or a complete dataset [5, 11, 24]. Furthermore, other state-of-the-art

SIGMOD '23, June 18–23, 2023, Seattle, WA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

[1] https://www.kaggle.com/kaggle/meta-kaggle
[2] For example, over 91% (~553,000) of pipelines on Kaggle are written in Python according to the Meta Kaggle dataset at: https://www.kaggle.com/kaggle/meta-kaggle
[3] http://www.amundsen.io

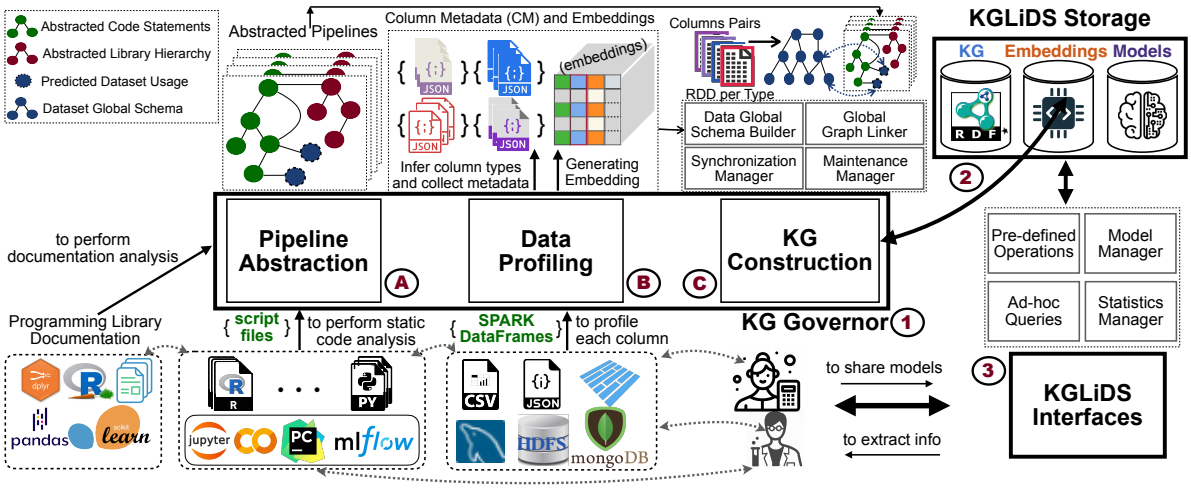

Figure 1: An overview of KGLiDS's main components and the interaction with data sources and pipeline scripts managed by different data science platforms. Data science pipelines and the associated datasets are analyzed via the Pipeline Abstraction and Data Profiling components, respectively. The KG Builder consumes their outputs to construct the LiDS graph and maintains the graph and associated embeddings into an RDF-star and embedding stores. Different users can interact with KGLiDS via the KGLiDS Interfaces to extract information or to share with the community their models developed on top of the LiDS graph.

data discovery approaches, e.g., for measuring table relatedness [34], identifying joinable [36] or unionable [24] tables, work in isolation from each other and do not support discovery based on a combination of diverse features.

In conclusion, although some techniques have been proposed for dataset and pipeline discovery, they (i) have been studied in isolation, (ii) come with several shortcomings, and (iii) have not been integrated in existing platforms that can be used in practice. In this paper, we therefore propose KGLiDS, a fully-fledged platform powered by knowledge graphs capturing the semantics of data science artifacts, incl. both datasets and pipelines as well as their interconnections. To the best of our knowledge, KGLiDS is the first system to encapsulate both aspects of data science in a single representation.

In summary, this paper makes the following contributions:

- First fully-fledged platform to enable Linked Data Science[4].
- Scalable deep learning-based data profiling techniques and fine-grained type inference methods to construct a global representation of datasets.
- Capturing the semantics of pipeline scripts in a scalable and accurate way with specific support for Python and its dynamic nature.
- A novel approach for exploring data science artifacts in consideration of datasets, pipelines, and their connections.
- A comprehensive evaluation using two different benchmarks with thousands of datasets and more than thirteen thousand pipeline scripts extracted from data discovery benchmarks and the Kaggle portal – showing the superiority of KGLiDS over the state of the art

The remainder of this paper is organized as follows. In Section 2, we provide an architecture overview of the KGLiDS platform. Section 3 describes the KGLiDS knowledge graph detailing the construction process and the assumptions taken. In Section 4, we discuss the main functionality of KGLiDS to explore and automatically discover and learn from the knowledge graph. Section 5 then presents the results of our comprehensive evaluation. In Section 6, we review existing work. And finally, Section 7 concludes the paper.

[4]https://gitfront.io/r/CoDS-GCS/Ma9Uz7qkpyqu/kglids/

2 KGLIDS AND LINKED DATA SCIENCE

The architecture of KGLiDS is illustrated in Figure 1. The main components are: (1) *KG Governor*, which is responsible for creating, maintaining, and synchronizing the KGLiDS knowledge graph with the covered data science artifacts, pipelines, and datasets, (2) *KGLiDS Storage*, which stores the constructed knowledge graphs as well as embeddings and ML models, (3) *KGLiDS Interfaces*, which allows a diverse range of users to interact with KGLiDS to extract information or share their findings with the community.

2.1 The KG Governor and Data Science Artifacts

KGLiDS captures semantics of data science artifacts, i.e., pipelines and their associated datasets, by applying novel methods for pipeline abstraction and data profiling. During bootstrapping KGLiDS is deployed by enabling the KG Govenor to profile the local datasets and abstract pipeline scripts to construct a knowledge graph as illustrated in Figure 1. The main subcomponents of the *KG Governor* are: (A) *Pipeline Abstraction*, which captures semantics of pipelines by analyzing pipeline scripts, programming libraries documentation, and usage of datasets, (B) *Data Profiling*, which collects metadata and learns representations of data items, e.g., columns or tables, and (C) *Knowledge Graph Construction*, which builds and maintains the LiDS graph and embeddings.

KGLiDS does not only rely on extracted information and relationships between nodes only but can also predict nodes and edges in the knowledge graph using the *Global Graph Linker*, which is part of the *KG Construction* subcomponent. As discussed in more detail in Section 3, KGLiDS adopts embedding-based methods to predict relationships among data items and utilizes MapReduce algorithms to guarantee a scalable approach for constructing the

graph from large growing datasets and pipelines. KGLiDS is not a static platform; as more datasets and pipelines are added, KGLiDS continuously and incrementally maintains the LiDS graph. To avoid having to run the prediction every time a user wants to use the system, predictions are materialized and the resulting nodes and edges annotated with a prediction score expressing the degree of confidence in the prediction.

The Linked Data Science (LiDS) ontology. To store the created knowledge graph in a standardized and well-structured way, we developed an ontology for linked data science: the *LiDS ontology*[5]. Its main types of nodes (classes) are: data items (with related nodes representing datasets, tables, and columns), libraries, and pipeline scripts (with related nodes describing statements) as illustrated in Figure 2. The *LiDS ontology* conceptualizes the data, pipeline, and library entities in data science platforms. The ontology is specified in the Web Ontology Language (OWL 2) and has 13 classes, 19 object properties, and 22 data properties. OWL was chosen as a standard because of its integral support of interoperability and sharing data on the Web and across platforms. Similar to the ontology of DBpedia[6], the URIs of classes and properties (relationships) have the prefix *http://kglids.org/ontology/*, while data instances (resources) have the prefix *http://kglids.org/resource/*. For example, a column *Age* in a table *train.csv* of the *titanic* dataset has the URI *http://kglids.org/resource/kaggle/titanic/train.csv/Age*. Similarly, libraries are identified by unique URIs such as *http://kglids.org/resource/library/sklearn/svm/SVC*.

The Linked Data Science (LiDS) graph. We refer to the graph populating the ontology with instances of the classes as the *LiDS graph*. Using the RDF standard [20] , KGLiDS captures and describes the relationships among these entities and uses Uniform Resource Identifiers (URIs) for nodes and edges in the LiDS graph so that the graph can easily be published and shared on the Web. All entities are associated with an RDF label and RDF type to facilitate RDF reasoning on top of the LiDS graph. Example object properties include column similarity relationships: *http://kglids.org/ontology/data/hasSemanticSimilarity*, and data flow between statements: *http://kglids.org/pipeline/hasDataFlowTo*, while data properties include statement parameters: *http://kglids.org/pipeline/hasParameter*.

2.2 The KGLiDS Storage and Interfaces

KGLiDS maintains different types of information, namely the LiDS graph, the generated embeddings for columns and tables, and the machine learning models for different use cases. The current implementation of KGLiDS adopts the RDF model to manage the LiDS graph and uses Stardog as storage engine [10]. KGLiDS uses RDF-based knowledge graph technology because (i) it already includes the formalization of rules and metadata using a controlled vocabulary for the labels in the graphs ensuring interoperability [13, 31], (ii) it has built-in notions of modularity in the form of named graphs, for instance, each pipeline is abstracted in its own named graph [8], and (iii) it is schema-agnostic, allowing the platform to support reasoning and semantic manipulation, e.g., adding new labeled edges between equivalent artifacts, as the platform evolves [7, 12, 35], and

[5]http://kglids.org/ontology
[6]https://dbpedia.org/ontology

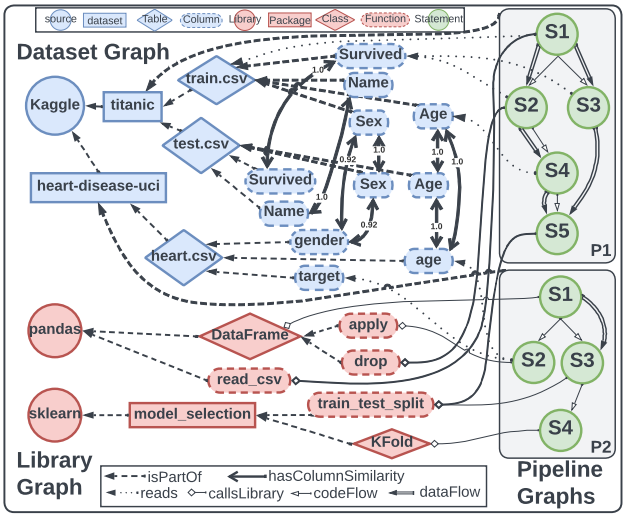

Figure 2: An overview of the LiDS graph, which consists of the dataset, library, and pipeline graphs. Each pipeline is isolated in a named graph.

(iv) it has a powerful query language (SPARQL) to support federated query processing [2, 22, 30]. In extension, KGLiDS uses the RDF-star [16] model, which supports annotating edges between nodes with metadata, which enables us, for instance, to capture the similarity scores for similarity edges between column nodes of datasets.

KGLiDS uses an embedding store, i.e., Faiss [19], to index the generated embeddings and enable several methods for similarity search based on approximate nearest neighbor operations on both CPU and GPU. Thus, KGLiDS enables efficient APIs to interact with the embeddings. This allows users to query the LiDS graph based on the embeddings associated with graph nodes. In Section 4, we discuss the Embedding Similarity Search in KGLiDS. Finally, KGLiDS also stores the ML models developed on top of the LiDS graph.

The KGLiDS portal supports access restrictions to prevent unauthorized users or could be public for anyone. Authorized users have access to query the LiDS graph or embeddings. However, accessing the actual data files in an enterprise may need another level of authorization. A broad range of users with different technical backgrounds can use and benefit from KGLiDS, which provides a simplified interface for non-technical users where pre-configured operations are available, as well as an interface for experts that allows, for instance, editing parameters or formulating ad-hoc queries against the LiDS graph to discover data items and pipelines. Interoperability with other tools was one of KGLiDS's design goals. Hence, KGLiDS, for instance, exports query results as Pandas data frames, a widely used format in data science [28].

3 CAPTURING SEMANTICS

The KGLiDS platform captures semantics of datasets, associated data science pipelines, and programming libraries and constructs (i) pipeline graphs, e.g., each pipeline script is abstracted and stored in a separate named graph, (ii) a library graph for all programming libraries used by the abstracted pipelines, and (iii) a dataset graph.

```
1  import pandas as pd
2  from sklearn.ensemble import RandomForestClassifier
3  from sklearn.model_selection import train_test_split
4  from sklearn.metrics import accuracy score
5  # Read the dataset
6  df = pd.read_csv('titanic/train.csv')
7  X,y = df.drop('Survived', axis=1), df['Survived']
8  X['NomralizedAge'] = (X['Age'] - X['Age'].mean()) / X['Age'].std()
9  # Split to train and test
10 X_train,y_train,X_test,y_test = train_test_split(X,y,test_size=0.2)
11 print(X_train.shape)
12 # Train an RF classifier
13 clf = RandomForestClassifier(50, max_depth=10)
14 clf.fit(X_train, y_train)
15 # Evaluate the classifier
16 print(accuracy_score(y_test, clf.predict(X_test))
```

Figure 3: A running example to demonstrate the KGLiDS Pipeline Abstraction. In this pipeline script, a dataset is loaded using Pandas and split into training and testing portions. Then, a random forest classifier is fitted and evaluated.

3.1 Pipeline Graphs and Library Graph

A data science pipeline, i.e., code or script, performs one or more data science tasks, e.g., data analysis, visualization, or modelling. The pipeline can be abstracted as a control flow graph, in which code statements are nodes and edges are flows of instructions or data. Pipeline P1, in Figure 3, is an abstraction of lines 6-10 in Figure 3. The objective of pipeline abstraction then is to have a language-independent representation of the pipeline semantics, such as code and data flow within a pipeline, invocations of built-in or third-party libraries, and parameters used in such invocations.

KGLiDS combines analysis of code, library documentation, and dataset usage to have a rich semantic abstraction capturing the essential concepts in data science pipelines. Algorithm 1 sketches the process. The main inputs of Algorithm 1 are a dataset $S = \{s\}$, where s is a pipeline script, MD metadata of pipelines, such as information about datasets used in the pipeline, pipeline author, and LD documentations of programming libraries. The algorithm also maintains the *library graph* for the used libraries and a named graph for each pipeline (lines 2 to 3). Our algorithm divides the pipeline abstraction process into a set of independent jobs (line 5), one per pipeline script creating a named graph (line 7) utilizing static code analysis tools, which are natively supported by several programming languages, such as Python (via, for instance `ast` and `astor`) or R (via, for instance, CodeDepends).

For each statement in a script, we store the following: i) *code flow*, i.e., the order of execution of statements, ii) *data flow*, i.e., subsequent statements that read or manipulate the same data variable, and iii) *control flow type*, i.e., whether the statement occurs in a loop, a conditional, an import, or a user-defined function block, which captures the semantics of how a statement is executed, and iv) *statement text*, i.e., the raw text of the statement as it appears in the pipeline. We discard from our analysis statements that have no significance in the pipeline semantics, such as `print()`, `DataFrame.head()`, and `summary()`.

Static analysis is not sufficient to capture all semantics of a pipeline. For instance, it cannot detect that `pd.read_csv()` in line 6 in Figure 3 returns a Pandas DataFrame object. Algorithm 1 further enriches the static program analysis using the documentation of data science libraries (lines 8 to 13). Each statement from static program analysis is enriched by the library it calls, names and values of parameters, including implicit and default ones, and data types of

Algorithm 1: Pipeline Abstraction

Input: Pipeline Scripts $\mathcal{S}$, Pipeline Metadata $\mathcal{MD}$, Programming Library Documentation $\mathcal{LD}$

1 **Main Node:**
2 library_hierarchy ← build_library_hierarchy_subgraph($\mathcal{LD}$)
3 pipeline_metadata ←build_pipeline_metadata_subgraph($\mathcal{MD}$)
4 json.dump(library_hierarchy, pipeline_metadata)
5 $\mathcal{S}_{rdd}.map(analyze\_pipeline\_script)$
6 **Worker (Parallel)** *analyze_pipeline_script(s)*:
7 g ← static_code_analysis(s) ▷ Control and data flow graph
8 **for** *node* ∈ *g* ▷ Documentation Analysis
9 **if** *node calls library* `lib` ∈ $\mathcal{LD}$
10 node.return_type ← $\mathcal{LD}$ [lib].return_type
11 node.parameter_names← $\mathcal{LD}$[lib].parameters.names
12 **for** p ∈ $\mathcal{LD}$ *[lib].parameters and* p ∉ *node.parameters*
13 node.default_parameters += (p.name, p.value)
14 **for** *node* ∈ *g* ▷ Dataset Usage Analysis
15 **if** *node calls* `pandas.read_csv(dataset_name.csv)`
16 node.predicted_dataset_read ← `dataset_name`
17 **if** *node calls* `pandas.DataFrame[column_name]`
18 node.predicted_column_reads += `column_name`
19 json.dump(g) ▷ Save abstracted pipeline graph

return variables. For each class and method in the documentation, we build a JSON document containing the names, values, and data types of input parameters, including default parameters, as well as their return data types. This analysis enables accurate data type detection for library calls to inference the names of implicit call parameters, such as `n_estimators` which is the first parameter in line 13 in Figure 3. Static code analysis cannot support this kind of inference. While performing the document analysis, we use and expand the *library graph* capturing methods belonging to classes, sub-packages, etc. (shown in red in Figure 2).

It also allows the inference of names of implicit call parameters, such as `n_estimators`, the first parameter in line 13 in Figure 3. A useful by-product of documentation analysis is the library graph, indicating methods belonging to classes, sub-packages, etc. (shown in red in Figure 2). This is useful for deriving exciting insights related to data science programming languages. For example, it helps find which libraries are used more frequently than others. KGLiDS, enable retrieving this kind of insight via SPARQL queries against.

Finally, Algorithm 1 uses *Detected Dataset Usage* nodes to capture connections between pipeline statements and the tables or columns used by the pipelines (lines 14–18). If statement reads a table via `pandas.read_csv` (e.g. line 7 in Figure 3) or a column via string indices over DataFrames (e.g. line 8 in Figure 3), such tables or columns are identified as potential reads of actual data. In KGLiDS, the Graph Linker further investigates the detected dataset usage to like these cases into the datasets graph. That helps tracking the usage of datasets, tables, or columns in different pipelines.

3.2 Dataset Graph

Inspired by [23], KGLiDS uses deep learning to generate embeddings for columns and tables based on their content. To also capture semantics, KGLiDS uses a method based on Word Embeddings [14]. Given a dataset, it is first decomposed into independent tables, and

each table is then decomposed into a set of columns, where most computations are done. Statistics and embeddings for tables are then derived from the ones for columns. In the following, we only sketch the implementation and refer the interested reader to the extended version of this paper at the sysname's repository.

KGLiDS makes use of PySpark DataFrame to analyze columns, enabling scalability via distributed computation and using fixed-size embeddings to represent columns and infer their types. In doing so, KGLiDS inherits the PySpark DataFrame support to handle files of different formats, such as CSV, JSON, and Parquet, and connect to relational DB and NoSQL systems. Unlike other systems, KGLiDS breaks down the main data types (numerical, textual, and dates) into fine-grained types based on pattern matching and deep learning methods. For example, Booleans, emails, or postal codes are identified using predefined regular expressions while person, organization, or location names are predicted using named entity recognition (NER) models [27].

KGLiDS generates deep dataset embeddings (*DeDE*) for each column based on a sample of its actual values. We train the DeDE model to capture similarities between columns. DeDE provides three main advantages to KGLiDS; first, higher accuracy of predicted column content similarities in contrast to standard statistical measures [23]. Second, it enables data discovery without exposing the datasets' raw content, which is invaluable in an enterprise setting, where access to the raw data might be restricted. Third, a compact representation of fixed-size embeddings, regardless of the actual dataset size, greatly reduces the storage requirements. Two columns have similar embeddings if their raw values have high overlap, have similar distributions, or measure the same variable – even with different distributions (e.g. `weight_in_kg` is similar to `weight_in_gram`).

To actually construct the *dataset graph* and its links to the *pipeline graphs*, KGLiDS constructs a metadata subgraph containing the hierarchy structure of the datasets and statistics collected for each column. Then, similarity relationship/edges between nodes (columns and datasets) are created by determining their similarities using predefined thresholds – the similarity values are captured as metadata of the edges and encoded using RDF-star. The pairwise comparisons between columns are processed in a MapReduce fashion. To determine the similarity between columns, we use (i) name similarity (similar column names based on GloVe Word embeddings [26] and a semantic similarity technique [15]), (ii) content similarity (similar raw values based on the cosine distance between their column embeddings), (iii) inclusion dependency (numerical columns with similar ranges of values included in one another [9]), and (iv) primary key and foreign key similarity (columns with high uniqueness and content similarity)

4 THE KGLIDS INTERFACES

The primary users of KGLiDS are data scientists in an open or enterprise setting whose objective is to derive insights from their datasets, construct pipelines, and share their results with other users. To achieve this goal, KGLiDS supports offers a number of interfaces (see Figure 1, component 3); many offering access to *Predefined Operations* while some allow users to query the raw LiDS graph directly (*Ad-hoc Queries*). The *Statistics Manager* is collecting and managing statistics about the system and the LiDS graph. The *Model Manager* then enables data scientists to run analyses and train models directly on the LiDS graph to derive insights, e.g., training a GNN [32]-based classifier to detect pipelines used for data exploration. Due to limited space, the remainder of this section focuses on pre-defined operations. More details can be found in the extended version.

Pre-defined Operations

Let us consider a scenario where a data scientist is interested in predicting heart failure in patients and illustrate how KGLiDS's pre-defined operations can help achieve this goal.

Search Tables Based on Specific Columns. To get started, the data scientist would like to find relevant datasets using keyword search, the following operation supports this:

```
search_keywords([['heart', 'disease'], 'patients'])
```

KGLiDS will then perform a search in LiDS using the conditions that are passed by the user who has the possibility to express conjunctive (`AND`) and disjunctive conditions (`OR`) using nested lists. In the example above 'heart' and 'disease' are conjunctive and 'patients' is a disjunctive condition. Let us assume the data scientist has found the following two datasets of interest: `heart-failure-prediction`, `heart-failure-clinical-data` and `world-happiness-report`.

Discover Unionable Columns. In the next step, let us assume that the data scientist would like to combine the two tables into one. Since it is very unlikely that the two tables in our example come with exactly the same schema, the data scientist is seeking assistance to identify matching (unionable) columns expressing the same information. KGLiDS provides support to automatically recommend a schema for the merged table containing all columns from both input tables that can be matched (unionable columns). The output will be a Pandas DataFrame. This can be expressed as:

```
find_unionable_columns (dataset1, table1, dataset2, table2)
```

Join Path Discovery. Let us now assume that the data scientist would like to join the obtained table with a table from another dataset to obtain a richer set of features for downstream tasks. Let us further assume that the table is not directly joinable. Hence, KGLiDS suggests intermediate tables (restricted to a join path with 2 hops in our example) and displays the potential join paths. This is done by computing an embedding of the given DataFrame (`df`), finding the most similar table in the LiDS graph, and determining potential join paths to the given target table. This could be expressed as follows:

```
get_path_to_table(df, target_dataset, target_table, hops=2)
```

[7] KGLiDS also supports more challenging variations, such as identifying the shortest path between two given tables.

Library Discovery. Since the data scientist would like to build an ML pipeline to predict heart failure and is not an expert in building ML pipelines, KGLiDS helps the data scientist by providing a way to become familiar with the most used libraries in ML pipelines for similar tasks. This can be expressed as follows:

[7] Katja: What is df? If it's the unioned table from above, then df should be introduced as the output above

```
get_top_used_libraries(k=10, task='classification')
```

where k specifies the number of libraries to return.

Pipeline Discovery. After reading more about the most used libraries, Pandas, Scikit-learn, and XGBoost, the data scientist is interested to see example pipelines where the following components are used: `pandas.read_csv`, `XGBoost.XGBClassifier`, and `sklearn.f1_score`. This can be expressed as:

```
get_pipelines_calling_libraries('pandas.read_csv',
    'XGBoost.XGBClassifier', 'sklearn.metrics.f1_score')
```

KGLiDS returns a list of pipelines matching the criteria along with other important metadata, such as author, language, etc.

Feature Recommendation. Exploiting the observation that similar datasets often have similar pipelines, KGLiDS offers a function to recommend a set of features (columns) for a given dataset. The data scientist in our example can make use of this function as follows:

```
recommend_features_for_task(df, task='classification')
```

KGLiDS determines the features (columns) by first computing an embedding for the given DataFrame (`df`) and then using this embedding to search for the most similar dataset S. Afterwards, KGLiDS retrieves the set of S's columns that are most often used in pipelines and returns the those matching the given DataFrame (`df`).

Classifier Recommendation. Afterwards, the data scientist needs to decide which classification model to use and would like to retrieve suggestions:

```
recommend_ml_models(df, task='classification')
```

Using a similar technique as for feature engineering, KGLiDS uses embeddings of the used DataFrame to find the top-k pipelines and classifiers that are used with similar datasets.

Hyperparameter Recommendation. In the final step, the data scientist would like KGLiDS to provide help with finding a promising configuration for the hyperparameters of the chosen classifier. This can be expressed as follows:

```
recommend_hyperparameters(myDataFram, model)
```

Again, KGLiDS uses embeddings of the provided DataFrame and the given criteria to find classifiers used on similar datasets and retrieves the most commonly used configurations for hyperparameters. The data scientist can then use these values to train a model.

5 EXPERIMENTAL EVALUATION

This section presents an empirical evaluation of KGLiDS. To the best of our knowledge, KGLiDS is the first platform to capture the semantics of data science artifacts, i.e., pipeline scripts and datasets. Related systems support programming scripts or data discovery in isolation from each other. Our evaluation compares the performance of KGLiDS's components with these systems and analyzes the quality of modelling different tasks on top of our LiDS graph.

Compared Systems We evaluated KGLiDS in terms of pipelines abstraction against GraphGen4Code[1], a toolkit to generate a knowledge graph for code [8]. For data discovery, we evaluated KGLiDS against D^3L [5] and Aurum [11]. Both systems construct a navigational data structure for datasets consisting of tables and csv files.

Pipeline Scripts and Datasets We collected from Kaggle [9] 13,800 data science pipeline scripts used in the top 1000 datasets, which includes *3,775* tables and *141,704* columns. We selected these pipelines and datasets based on the number of user votes on the platform. For each dataset, we select up to 20 most voted Python pipelines. The datasets are related to various supervised and unsupervised tasks in different domains, such as health, economics, games, and product reviews. The pipeline scripts have different categories of pipelines, such as pure data exploration and analysis (EDA), and machine learning modelling. We also used the benchmarks provided by D^3L, namely `SmallerReal` and *Synthetic*. `SmallerReal` is a collection of 654 tables. It is collected from the UK open data portal for different domains such as business, health, transport, etc. [10]. *Synthetic* is a collection of 5044 tables. It is created based on the Canad open portal [11].

RDF Engines and Computing Infrastructure. We used Stardog version *XXX* as SPARQL endpoints. We deployed the endpoint, KGLiDS, and GrapGen4Code on the same local machine. We used two different settings for our experiments. In the first setting, we used a Linux machine with 16 cores and 90GB of RAM. We used a SPARK cluster of XX machines and XXGB RAM in the second setting to construct our dataset graphs. Unlike KGLiDS, both D^3L and Aurum are not implemented to use multiple machines. Thus, we compared them in terms of Precision (P) and Recall (R) and reported their results in [5].

The KGLiDS Setup. We analyzed Python pipeline scripts (including Jupyter Notebooks) and analyzed the documentation of the top three Python data science libraries, namely, `Pandas`, `Sklearn`, and `Numpy`. For our dataset graph, we set the similarity thresholds α (XXX similarity) , β (XXX similarity), θ (XXX similarity) and γ (XXX similarity) to 0.75, 0.95, 0.90, and 0.60, respectively.

5.1 Pipeline Graphs: Abstraction and Quality

We compared KGLiDS to GraphGen4Code[1] using the 13,800 Kaggle pipelines. Our comparison focuses on pipeline abstraction and accuracy of models trained on the generated graphs.

Semantic Pipeline Abstraction. For this set of pipelines, KGLiDS consumed 113.7 minutes to generate our LiDS graph, which contains 13M triples. GraphGen4Code consumed 2,255.4 minutes to generate a graph of 97M triples for the same set of pipelines. In GraphGen4Code, the focus is abstracting semantics from general code for programs. Thus, GraphGen4Code capture unnecessary semantics from pipeline scripts. Unlike GraphGen4Code, KGLiDS capture semantics related only to data science. Thus, KGLiDS achieved a graph reduction of more than 86% and even captured semantics, which GraphGen4Code did not manage to capture, as shown in Table 1. For example, capturing semantics related to *statement location*, *variable names*, or *function's parameter order* is irrelevant to pipeline automation or discovery. In contrast, capturing semantics related to datasets, such as *dataset reads*, or libraries, such as *library hierarchy*, is essential for automating data science pipelines, such as discovering pipelines for similar datasets and highly effective libraries.

[8] GrapGen4Code is obtained from https://github.com/wala/graph4code as of Dec, 2021

[9] The script to download these datasets is available at KGLiDS's repository

[10] https://github.com/alex-bogatu/DataSpiders.git

[11] https://github.com/RJMillerLab/table-union-search-benchmark.git

Table 1: A comparative analysis between KGLiDS and GraphGen4Code in terms of pipeline abstraction. For each category in each graph g of T triples, we reported $t(r\%)$, where t is number of triples and r is the ration of t to T.

Modelled Aspect	KGLiDS	GraphGen4Code
Library call	500.5k (3.8%)	15,272k (15.6%)
Code flow	2,110.7k (15.9%)	20,304k (20.8%)
Data flow	1,272.4k (9.6%)	13,295k (13.6%)
RDF node types	2,249.2k (17.0%)	-
Control flow type	809.6k (6.1%)	1,124k (01.2%)
Column reads	212.5k (1.6%)	1,997k (02.0%)
Dataset reads	26.1k (0.2%)	-
Function parameters	3,699.3k (28.0%)	7,511k (7.7%)
Library hierarchy	101 (0.0%)	-
Statement text	2,124.5k (16.0%)	7,944k (8.1%)
Statement location	-	3,972k (4.1%)
Variable names	-	987k (1.0%)
Function parameter order	-	25,133k (25.8%)
Total	13,221.9k	97,538k

Modelling Accuracy. We evaluated the quality of the generated graphs for pipelines using four graph classification tasks (two binary and two multi-class), as summarized in Table 2. Task *T1* classifies pipelines into pure EDA or Modelling ML. Task *T2* classifies pipelines into classical ML-based or DL-based pipelines. In task *T3*, ML pipelines are classified by the ML task as either regression, classification, or clustering. In task *T4*, the pipelines are classified based on the application domain of the pipeline. We selected the four most popular domains: finance, education, health, and entertainment. The ground truth for these tasks is based on the tags associated with each pipeline. These tags are provided by the authors who uploaded the pipelines to Kaggle, i.e., human labelled data. We split the pipelines into training, validation, and testing sets for each task with ratios 80%, 10%, and 10%, respectively. Further, we down-sample the majority class, if any, to account for model bias.

We train a Deep Graph Convolutional Neural Network (DGCNN) [33] classification model, which takes a pipeline graph and its class/label as input. Additionally, the model requires for each node the associated node embedding. To this end, we train a structural node embedding model, namely TransE [6], for each pipeline graph (both KGLiDS and GraphGen4Code) separately and feed the trained node embeddings to the DGCNN classifier. We use the same training configuration and hyperparameters when training on KGLiDS or GraphGen4Code. KGLiDS outperformed GraphGen4Code in most tasks, showing that KGLiDS's 86% graph reduction and more than 90% acceleration in graph construction comes lead also to a graph of high quality. Moreover, our LiDS graph helps automate different aspects of pipelines in faster time as these models are dominated by the number of vertices in the graph, which is significantly more in the graphs generated by GraphGen4Code.

5.2 The Datasets Graph and Data Discovery

We compared KGLiDS to D^3L [5] and Aurum [11] in terms of different data discovery tasks, such as table relatedness and joinable paths, using the same benchmarks used by D^3L. We used both

Table 2: A comparison between KGLiDS and GraphGen4Code in terms of modelling. For each task, we report $a(t)$, where a is the classification accuracy and t is the training time in min.

System	Binary Tasks		Multi-class Tasks	
	T1	T2	T3	T4
KGLiDS	59.8 (14.6)	**63.5** (8.8)	**45.1** (3.2)	**36.7** (6.5)
GraphGen4Code	**66.1** (69.5)	60.9 (39.8)	38.7 (13.8)	34.8 (19.4)
Baseline Accuracy	50.4	50.5	36.4	27.6
No. Pipelines	1974	1188	551	543

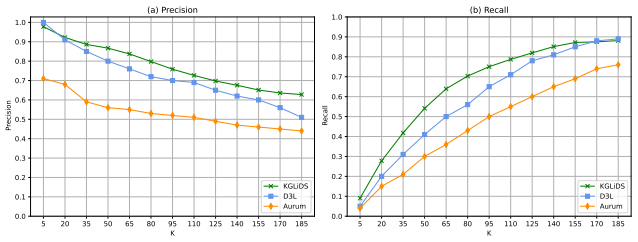

Figure 4: Average precision (a) and recall (b) of table relatedness for KGLiDS, D^3L, and Aurum on `SmallerReal` dataset as k increases from 5 to 185.

`SmallerReal` and *Synthetic* datasets of D^3L. Both datasets have an average number of 10.7 columns per table. However, average number of rows per table is 12,207 in `SmallerReal` and 1,908 in *Synthetic*. Thus, `SmallerReal` is more challenging as each table contains 84% more rows of real data. We got similar results in both datasets. We show only our results in `SmallerReal` for lack of space.

Table Relatedness. In an extensive collection of datasets, such as a data lake, one of the data discovery tasks is to find related tables. As defined in [5] for a certain table T in a dataset D, (*i*) table relatedness is calculated in terms of Precision@k and Recall@k for different values of k, where k is the number of related tables to T, and (*ii*) average precision and recall is calculated per 100 target tables sampled at random without replacement for different values of k. KGLiDS consistently outperforms D^3L and Aurum for both precision and recall on the `SmallerReal` dataset, as shown in Figure 4. These results demonstrate the effectiveness of our table unionability relationships by breaking down column data types into sub-types using both a named entity recognition model and pattern matching. This decreases the number of false-positive relationships in these majority string benchmark datasets. The results also show the accuracy of our table similarity score, which considers the number of similar columns and their similarity scores.

Join Paths. In this experiment, we evaluate the impact of join paths on the *attribute precision* of the top-k related tables on T. A join path starting from a table S is a chain of tables that can be joined on one or more columns to populate S with more columns. Therefore, for a target table T, tables $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ on join paths starting from either of the top-k related tables are also promising candidates for being related to T. We evaluate the attribute precision with and without considering join paths and average the results for 100 randomly sampled target tables at different values of k. For both the settings i.e. with and without join paths, KGLiDS outperforms

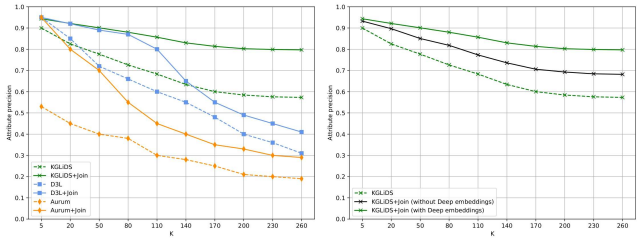

Figure 5: (a) The impact of considering join paths for attribute precision on `SmallerReal` dataset for $k \in [5, 260]$. (b) Using deep dataset embeddings to detect join paths improves the attribute precision in KGLiDS.

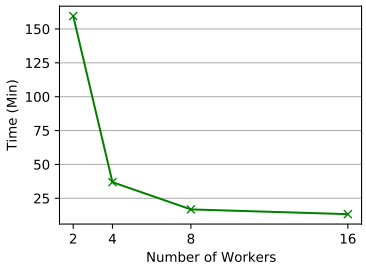

Figure 6: Shared-nothing; strong scalability test for KGLiDS' data profiling and dataset graph builder on the 1000 Kaggle datasets including 3,775 tables and 141,704 columns.

Aurum for all values of k, as shown in Figure 5. Attribute precision with join paths is comparable between KGLiDS and D^3L for lower values of k. However, KGLiDS outperforms D^3L for higher values of k with a significantly high margin. This is because KGLiDS generates deep embeddings of column values which serve as the basis of generating the `primary-key-foreign-key` relationships. The quality of the joins is clearly demonstrated as it significantly improves attribute precision when considered. In addition, Figure 5 (b) shows the improvement gained by using deep dataset embeddings to predict join paths.

A Scalable Datasets Graph Construction.

@Mossad write for the final one. Not the one in the figure :) Based on the settings we agreed on.

In this experiment, we evaluate the scalability of our system by performing a strong scalability test for the construction of the dataset graph on a collection of 250 datasets from Kaggle ($\tilde{6}$70 tables and $\tilde{2}$0,000 columns). For the same task, we vary the number of workers in the cluster from 2 to 16, doubling the number of workers with each increment. Figure 6 shows the time taken as we vary the number of workers, where the construction time drops from 159 minutes with 2 workers to 13 minutes with 16 workers, which demonstrates that KGLiDS can efficiently leverage distributed computation to construct the LiDS graph.

6 RELATED WORK

While some existing systems capture the semantics of code and others support data discovery, KGLiDS is the first platform to combine these aspects in a single system and implement our vision of federated data science [21].[12] While existing platforms do rarely use open standards[13], our platform is powered by knowledge graph technologies to enable the data science community to explore, exchange, and automatically learn from data science artifacts, namely pipeline scripts and datasets.

Capture Pipeline Semantics. The use of knowledge graphs to support applications on top of source code has seen much traction in recent years. Several approaches [1, 3, 4, 18] have been proposed to provide a semantic representation of source code with knowledge graphs. These techniques cannot be generalized to data science pipelines due to their heavy reliance on the detailed static analysis of Java, where information such as method input and return types is straightforward to determine. GraphGen4Code [1] is a toolkit based on pure static code analysis tool, WALA[14].[15] Unlike GraphGen4Code, KGLiDS combines static code analysis with library documentation and dataset usage analyses to have a rich semantic abstraction that captures the essential concepts in data science pipelines.

Data Discovery Systems design metadata collected from datasets as different navigational data structures. Examples of these systems are RONIN [25], which builds a hierarchical structure D^3L [5] that constructs hash-based indexes[16], and Aurum [11], which creates an in-memory linkage graph. We demonstrated the capabilities of KGLiDS as a data discovery system as referred to this part of the system as KGLac [17].[17] We are the first to design navigational data structures to capture the semantics of datasets and pipelines based on knowledge graph technologies. Furthermore, we enable several data discovery operations, such as unionable tables, joinable tables, and join path discovery. Unlike [24, 34, 36], KGLiDS also enables users to query datasets based on embeddings and combined easily different operations in one pipeline script as demonstrated in [17].[18]

7 CONCLUSION

This paper describes KGLiDS, a fully-fledged platform to enable Linked Data Science powered by knowledge graphs and machine learning. KGLiDS constructs a highly interconnected knowledge graph capturing the semantics of datasets, pipeline scripts, and code libraries. KGLiDS implements specialized static code analysis that infers information not otherwise obtainable with general-purpose static analysis, resulting in a richer, more accurate, and more compact abstraction of data science pipelines. KGLiDS further utilizes an advanced data profiler empowered by machine learning to analyze data items, incl. datasets, tables, and columns. In combination,

[12] Katja: We need to be very careful about this. The rest of the paper advertises Linked data science for some reason. This needs to be aligned!
[13] Katja: Please check if this is true.
[14] https://wala.github.io/
[15] Katja: The rest of the sentence is missing! How does WALA connect here?
[16] Katja: Please fix the first half of this sentence - unclear what belongs to which part (grammar problem)
[17] Katja: The previous sentence is out of context - please fix
[18] Katja: This would be a good spot to write something about AutoML.

these components enable novel use cases for discovery, exploration, reuse, and automation in data science platforms. [19]

REFERENCES

[1] Ibrahim Abdelaziz, Julian Dolby, James P. McCusker, and Kavitha Srinivas. 2020. A Toolkit for Generating Code Knowledge Graphs. *ArXiv* (2020). https://arxiv.org/abs/2002.09440

[2] Ibrahim Abdelaziz, Essam Mansour, Mourad Ouzzani, Ashraf Aboulnaga, and Panos Kalnis. 2017. Lusail: A System for Querying Linked Data at Scale. *Proceedings of the VLDB Endowment* 11, 4 (2017), 485–498. http://www.vldb.org/pvldb/vol11/p485-abdelaziz.pdf

[3] Awny Alnusair and Tian Zhao. 2010. Component search and reuse: An ontology-based approach. In *IEEE International Conference on Information Reuse & Integration*. 258–261. https://doi.org/10.1109/IRI.2010.5558931

[4] Mattia Atzeni and Maurizio Atzori. 2017. CodeOntology: RDF-ization of Source Code. 20–28. https://doi.org/10.1007/978-3-319-68204-4_2

[5] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *International Conference on Data Engineering (ICDE)*. 709–720. https://doi.org/10.1109/ICDE48307.2020.00067

[6] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 26. https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf

[7] Damian Bursztyn, François Goasdoué, Ioana Manolescu, and Alexandra Roatis. 2015. Reasoning on web data: Algorithms and performance. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 1541–1544. https://doi.org/10.1109/ICDE.2015.7113422

[8] Jeremy J. Carroll, Christian Bizer, Patrick J. Hayes, and Patrick Stickler. 2005. Named graphs, provenance and trust. In *Proceedings of the international conference on World Wide Web (WWW)*. 613–622. https://doi.org/10.1145/1060745.1060835

[9] Falco Dürsch, Axel Stebner, Fabian Windheuser, Maxi Fischer, Tim Friedrich, Nils Strelow, Tobias Bleifuß, Hazar Harmouch, Lan Jiang, Thorsten Papenbrock, et al. 2019. Inclusion dependency discovery: An experimental evaluation of thirteen algorithms. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 219–228.

[10] The Stardog RDF Engine. 2022. http://stardog.com.

[11] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *International Conference on Data Engineering (ICDE)*. 1001–1012. https://doi.org/10.1109/ICDE.2018.00094

[12] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.* 24, 6 (2015), 707–730.

[13] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the International World Wide Web Conference (WWW)*. 413–422. https://doi.org/10.1145/2488388.2488425

[14] Josu Goikoetxea, Eneko Agirre, and Aitor Soroa. 2016. Single or Multiple? Combining Word Representations Independently Learned from Text and WordNet. In *AAAI*.

[15] Josu Goikoetxea, Eneko Agirre, and Aitor Soroa. 2016. Single or Multiple? Combining Word Representations Independently Learned from Text and WordNet. In *Proceedings of the Thirtieth Conference on Artificial Intelligence (AAAI)*. 2608–2614. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11777

[16] Olaf Hartig. 2019. Foundations to Query Labeled Property Graphs using SPARQL. In *SEM4TRA-AMAR@SEMANTICS*, Vol. 2447. http://ceur-ws.org/Vol-2447/paper3.pdf

[17] Ahmed Helal, Mossad Helali, Khaled Ammar, and Essam Mansour. 2021. A Demonstration of KGLac: A Data Discovery and Enrichment Platform for Data Science. *PVLDB* 14, 12.

[18] Azanzi Jiomekong, Gaoussou Camara, and Maurice Tchuente. 2019. Extracting ontological knowledge from Java source code using Hidden Markov Models. *Open Computer Science* 9 (08 2019), 181–199. https://doi.org/10.1515/comp-2019-0013

[19] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[20] Graham Klyne, Jeremy J. Carrol, and Brian McBride. 2014. RDF 1.1 Concepts and Abstract Syntax. World-Wide Web Consortium.

[21] Essam Mansour, Kavitha Srinivas, and Katja Hose. 2021. Federated Data Science to Break Down Silos [Vision]. *SIGMOD Record* 50, 4 (2021), 16–22. https://doi.org/10.1145/3516431.3516435

[22] Gabriela Montoya, Hala Skaf-Molli, and Katja Hose. 2017. The Odyssey Approach for Optimizing Federated SPARQL Queries. In *Proceedings of the International Semantic Web Conference (ISWC)*, Vol. 10587. 471–489.

[23] Jonas Mueller and Alex Smola. 2019. Recognizing Variables from Their Data via Deep Embeddings of Distributions. In *International Conference on Data Mining (ICDM)*. 1264–1269. https://doi.org/10.1109/ICDM.2019.00158

[24] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825. http://www.vldb.org/pvldb/vol11/p813-nargesian.pdf

[25] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *PVLDB* 14, 12 (2021).

[26] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. https://doi.org/10.3115/v1/D14-1162

[27] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the Association for Computational Linguistics (ACL)*, Vol. 1. 1756–1765. https://doi.org/10.18653/v1/P17-1161

[28] Devin Petersohn. 2021. Scaling Data Science does not mean Scaling Machines. In *Conference on Innovative Data Systems Research (CIDR)*. http://cidrdb.org/cidr2021/papers/cidr2021_abstract11.pdf

[29] Vitalis Salis, Thodoris Sotiropoulos, Panos Louridas, Diomidis D. Spinellis, and Dimitris Mitropoulos. 2021. PyCG: Practical Call Graph Generation in Python. *International Conference on Software Engineering (ICSE)* (2021), 1646–1657.

[30] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. 2011. FedX: A Federation Layer for Distributed Query Processing on Linked Open Data. In *ESWC (2) (Lecture Notes in Computer Science, Vol. 6644)*. Springer, 481–486.

[31] Boris Villazón-Terrazas, Nuria García-Santa, Yuan Ren, Kavitha Srinivas, Mariano Rodriguez-Muro, Panos Alexopoulos, and Jeff Z. Pan. 2017. Construction of Enterprise Knowledge Graphs (I). In *Exploiting Linked Data and Knowledge Graphs in Large Organisations*. 87–116. https://doi.org/10.1007/978-3-319-45654-6_4

[32] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

[33] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018). https://ojs.aaai.org/index.php/AAAI/article/view/11782

[34] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *SIGMOD*. 1951–1966. https://doi.org/10.1145/3318464.3389726

[35] Weiguo Zheng, Lei Zou, Wei Peng, Xifeng Yan, Shaoxu Song, and Dongyan Zhao. 2016. Semantic SPARQL Similarity Search Over RDF Knowledge Graphs. *Proceedings of the VLDB Endowment* 9, 11 (2016), 840–851. http://www.vldb.org/pvldb/vol9/p840-zheng.pdf

[36] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 847–864. https://doi.org/10.1145/3299869.3300065

[19] Katja: Maybe once more the link to github, full paper, etc.