# Voyager: Data Discovery and Integration for Data Science

Alex Bogatu[†‡], Norman W. Paton[†], Mark Douthwaite[‡], André Freitas[†∇]

[†]The University of Manchester, UK, [‡]Peak AI Ltd., [∇]Idiap Research Institute, Switzerland
{alex.bogatu,norman.paton,andre.freitas}@manchester.ac.uk
mark.douthwaite@peak.ai

## ABSTRACT

Data discovery and integration have grown to become two important research fields in both academic and commercial domains, mainly fueled by the ever increasing availability of datasets that are stored by organisations without their conceptual meaning or relationships being explicitly known. These tasks can be carried out in different settings and for different purposes; here we focus on the collection of tasks performed by data scientists to acquire the knowledge needed when deciding what analyses to perform on client data. In this paper, we focus on support for three processes often encountered in practice by data scientists: data identification, data understanding and relationship discovery. We describe our practical experience with each of these processes and the means by which we assist data scientists in performing them. We have been informed by real–life use–cases in identifying the tasks carried out routinely by data scientists at Peak AI. The paper reports the design decisions made in the development of a system to support data discovery and integration, and reports on an evaluation that investigates both usability and task efficiency.

## 1 INTRODUCTION

As data analysis algorithms evolve, the need for tools and methods that reduce the complexity of data management becomes increasingly important. Organisations that have data science at their operational core often find themselves past the point where their data repositories can be conceptually modelled automatically. This has prompted organisations to build in-house data lifecycle management solutions to maintain a cohesive order among all their data resources, e.g., *Data Hub* [14], *Amundsen* [2], *Metacat* [29], etc. However, many such solutions play the role of pre-existing data expertise broadcasters, i.e., existing data knowledge is centralized and shared with other data stakeholders. This means that new data still has to be manually curated when it is imported into the system, be it a storage or a metadata management system.

At Peak AI, many data science activities are characterized by frequent interactions with new datasets - most commonly in tabular format. As new tables are imported into internal systems for analysis and dissemination, the productivity of the data discovery and data understanding process becomes a central focal point, and existing metadata management systems are of limited help when the data is new. In practice, data scientists are involved in a mixture of data discovery (e.g., [5, 18, 32]) and data integration (e.g., [16, 39]) tasks. Specifically, given a new collection of tables, a data scientist commonly starts by identifying the ones that are relevant for the intended analysis - often through search processes, i.e., data discovery, and continues to profiling, cleaning and merging them, i.e., data integration. The goal of this process is for the user to also become aware of the data's conceptual model, of its schemata and of the existing relationships between different data artefacts, e.g., tables. This knowledge will be necessary every time data scientists write structured queries or programs that consume that data.

Although both data discovery and data integration are well recognised research areas, in this paper we describe our experience with their practical blending in a common real-world scenario: data scientists having to interact with new, previously unseen, tabular datasets. Our objective is to make the data journey from ingestion to analysis more productive and, ultimately, *to assist data scientists in querying new data using structured queries or programs efficiently.*

In doing so, inspired by practical experience, we see the challenge as stemming from the divergence between the information needs dictated by the task at hand and the data representation format [20]. *What are the concepts to be included in the analysis? How are these represented in the data repository (what tables/columns)? How do the datasets that share the same concepts relate?* In this paper, we address these questions through three activities carried out by data scientists: *conceptual data identification*, *contextual data understanding* and *data relationship discovery.* We discuss each of these branches and propose *Voyager*, a system designed for and evaluated with data scientists. Concretely, our main contributions in this paper are:

- The identification and formalisation of data identification, data understanding and relationship discovery as tasks that are regular precursors to analysis for data scientists.
- A system, called *Voyager*, which is the result of applying recent data discovery research advances[5] to real-world use cases at Peak AI.
- A usability study of the said system with data scientists from Peak AI that investigates *Voyager* in relation to both usability and task efficiency

The remainder of the paper is structured as follows. Relevant related works are discussed next, in Section 2, together with commercial or open source systems. Then, in Section 3, we define conceptual data identification, contextual data understanding and data relationship discovery, and present a real–world case study to support our proposed definitions. Section 4 describes our proposed system, Voyager, that is evaluated using a methodology presented in Section 5 and with results reported in Section 6.

## 2 RELATED WORK

Data discovery is not yet an area with stable and well defined constituents, but results have been reported relating to *search*, *navigation* and *metadata management* that can certainly contribute to the development of data discovery systems. In relation to *dataset search* [9], keyword search can be supported either over the underlying data directly [25], or building on annotations [6]. Of particular relevance to this paper, is index–based dataset similarity search that allows for finding datasets that are related to a given query dataset, as in Aurum [18], Table Union Search [32] and D$^3$L [5]. Typically, such proposals index individual columns using one or several sketches that capture column features, and then a search algorithm combines the evidence from the indexes over the different columns to rank the most similar datasets. Such searches may consider the relationships between datasets (e.g., [5, 43]), but there is a wider requirement, having identified datasets of relevance, to support *navigation* between them, e.g., Seeping Semantics [19], [31].

In contrast with data discovery, data integration [16, 39] is, a well established research area with many associated sub–areas. Of particular interest for this paper is the work on data profiling [1, 33] and functional and inclusion dependencies discovery [36, 37, 41]. While we make use of data profiling concepts to retrieve data summary statistics in Voyager, we do not use traditional inclusion dependencies discovery algorithms. Instead, we rely on efficient search indexes, such as locality sensitive hashing (LSH) [4, 23], for approximating such dependencies and join paths (e.g., [15]) due to the potential scale of and inconsistencies in the data. Beyond that, Voyager merely aims to equip data scientists with the data knowledge that enables them to perform further integration, e.g., schema mapping [27, 28], rather than actually execute data integration processes.

The above techniques for discovering and connecting data sets can also be seen as part of a data catalog solution, and there is recent work on modelling [17] and inferring [42] metadata for sources with lean initial representations. Such models and inferred properties can be useful for populating catalogs, for which there have been proposals in the research literature (e.g., Ground [26], Constance [24]) and in products (e.g., Amundsen [2], LinkedIn's Data Hub [14], Uber's Databook [13], Airbnb's Dataportal [12], Netflix's Metacat [29]).

In this paper, we present a proposal that supports the early stages of data discovery and integration, including search, profiling, navigation and catalogue population, specifically focused on exploratory analysis of data collections. In this regard, our objectives are similar to those of RONIN [34], which also provides an early example of how to combine recent results on similarity search and relationship inference to provide interactive dataset discovery. In this paper, we complement the work in RONIN, with different emphases in relation to search and relationship discovery, giving rise to a rather different user experience. In addition, we provide a comprehensive usability evaluation with users from industry, to explore both how different features are received and the impact of the different capabilities on the efficiency of discovery and integration tasks.

## 3 CONCEPTS AND DEFINITIONS

There is no broad consensus in the field of data science as to what is involved in discovering data and making it available for analysis. In Peak AI, where analysis takes place on customer data using an in-house platform, a starting point involves uploading raw data into an online environment or data repository for future analysis. In practice, coming to terms with the data then involves writing one or more structured queries or a data manipulation program in some programming language, such as Python, R or SQL, with the purpose of identifying the concepts represented in the data, their representation particularities and inter–relationships. This can be seen as an example of *the vocabulary problem of databases* [22] where users' information needs and intents need to be mapped to concepts represented in the data. These concepts refer to different computational resources and structures, such as database schema elements, queries, parameters, file names and data formats, among others [20].

We denote the knowledge necessary for mapping concepts and intents to a collection of data tables as $\mathcal{K}$. We consider $\mathcal{K}$ as a prerequisite for defining a collection of functions, each represented in practice by some structured query over the available tables. Then, the task at hand can be viewed as *the process of acquiring $\mathcal{K}$ necessary in order to define the data queries.*

More formally, given a collection of raw tables $\mathcal{T} = \{T_1, \dots T_n\}$, the requirement is to acquire knowledge $\mathcal{K}$ about the data so as to produce a collection of views $\mathcal{V} = \{V_1, \dots V_m\}$ over $\mathcal{T}$. Each $V_i \in \mathcal{V}$ is the product of some function or structured query defined by some combination of relational operations, e.g., projection, selection, union, join, over one or more tables in $\mathcal{T}$. $V_i$ is then either stored for future analysis or becomes the subject of a data wrangling [21] and/or a data analysis pipeline.

Given $\mathcal{T}$, in order to obtain a view $V_i$, a data scientist often has to identify, clean and merge tables from $\mathcal{T}$. $\mathcal{K}$ can therefore be represented by a triple of data knowledge components, *viz.*, *data identification* ($I$), *data understanding* ($U$) and *data relationships discovery* ($R$):

- $I$ is useful for having a conceptual and structural knowledge about data, e.g., table and column names, what concepts they represent, how are these concepts organised in tables, etc. In practice, $I$ could be acquired by *searching and navigating* tables in $\mathcal{T}$.
- $U$ is useful for cleaning and transforming data. This is often relative to an analytical task/context, e.g., identify summary statistics of certain columns, value distributions, and data cleaning needs, such as missing value ratios, duplicate values, etc. In practice, $U$ could be achieved through *data profiling* of individual tables in $\mathcal{T}$.
- $R$ is useful in merging tables by having knowledge about their inter–relationships, e.g., primary key - foreign key (PK/FK) relationships, key candidates, join paths, etc. In practice, $R$ could be achieved by consulting an *entity/relationship (ER) diagram* of tables in $\mathcal{T}$.

Although data scientists may have to carry out additional processes, e.g., data transformation from different data models to tables, *I, U* and *R* have been identified in our experience as the main contributors to the learning curve of becoming familiar with new data.

Note that we do not aim to offer a recipe for performing data cleaning or integration but rather to facilitate such processes by assisting data scientists in becoming knowledgeable about the data at hand. Such knowledge often resides in a data scientist's mind and is often acquired over multiple interactions with the data. Our aim is to streamline this acquisition of $\mathcal{K}$ that can then be shared with other users by centralizing it into a metadata management tool. In this context, our proposed system can be seen as an extension to existing data cataloging tools, such as the ones mentioned in Section 2.

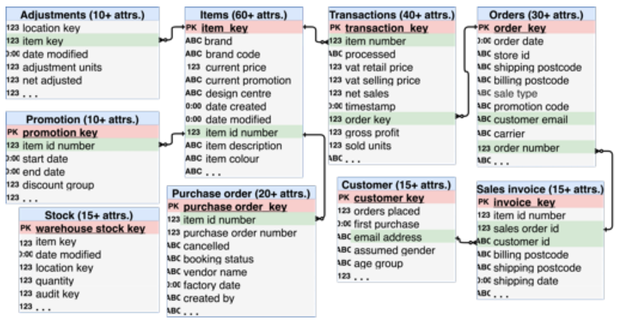

**Figure 1: Use-case entity–relationship diagram with inter–table relationships that are often missing in practice.**

## 3.1 Case Study

In this section, we employ a real–world–inspired[1] use–case to better exemplify and define conceptual data identification ($I$), contextual data understanding ($U$) and data relationship discovery ($R$).

Consider the diagram from Figure 1 where we depict a subset of tables similar to the ones used by data scientists at Peak AI to perform demand forecasting and price optimization for a clothing retailer. We changed the names of the schema elements in Figure 1 to make their semantics self–explanatory. However, this is may not be the case in practice. We observe the following:

(1) The exemplified tables are often part of a much larger collection of tens/hundreds of tables originating from sources outside the control of data scientists and, therefore, with their table and column semantics unknown.
(2) The information stored in such tables is updated at regular intervals, e.g., daily, weekly. However, the new records have to be matched against the prescriptive schema of the already onboarded data. Moreover, similarly to the initial records, they can be riddled with missing values, inconsistent formatting, outliers and duplicates.
(3) The primary key/foreign key (PK/FK) relationships illustrated in Figure 1 are often unknown in practice. This happens, for example, when the information originates from data warehouse systems that do not enforce integrity constraints for relational data or from simple file systems. Furthermore, the tables are not always joinable on their primary keys , e.g., *Items* and *Purchase order*.

## 3.2 Conceptual data identification

Observation (1) from above brings forward the need for mapping the data consumer's intents, often materialised as queries, and the concepts represented in data. There is no quantitative study measuring the dependency between schema size and the effort necessary to understand or even query a dataset [20]. However, together with data scientists at Peak AI, we have identified that this need for conceptually identifying what is stored in a new client data collection is a major step that can often take hours or even days, depending on the input size.

More specifically, given a collection of tables $\mathcal{T} = \{T_1, \dots T_n\}$, e.g., the ones in Figure 1, and an analytical objective, e.g., price optimization, conceptual data identification refers to the recognition of data elements (i.e., tables and attributes) that can be the subject of the intended or future analysis, e.g., attribute *current price* from *Items*, attributes *vat selling price* and *vat retail price* from *Transactions*, etc.

We note that, in the example above, tables *Items* and *Transactions* are the subjects of a direct mapping to the analytical intent because they contain explicit price information. Often however, an indirect mapping is also necessary. With respect to Figure 1, examples include tables such as *Promotion* or *Sales invoice* that can contribute to and influence the price analysis. All these tables have to be merged and queried for the intended analytical task. In this paper, we describe how we leverage *data search* to identify both types of datasets relevant for the given analysis.

## 3.3 Contextual data understanding

Observation (2) from above emphasizes the need for a deeper understanding of data, beyond conceptual awareness. In practice, data science is often preceded by a statistical exploratory analysis of tables of interest, e.g., analyzing value distributions, data types, duplicates ratios, missing value ratios, etc.. At Peak AI, many such tasks are dataset agnostic and, therefore, can be automated.

Continuing with the exemplification based on Figure 1, customer information from tables such as *Customer* or *Orders* is crucial for analysing buying patterns. Therefore, the data scientists have to assess the availability of such data, e.g., missing values, outliers, and even perform a preliminary statistical analysis, e.g., distributions, summary statistics, to determine the potential for value extraction from the data. In this paper, we describe how we leverage *automatic data profiling* to efficiently deliver such data understanding to data scientists.

## 3.4 Data relationship discovery

Observation (3) from above is a manifestation of one of the major obstructions we identified in the productivity of data integration processes: missing integrity constraints between tables. New incoming datasets often originate from storage systems that are not under expert administration and, therefore, their formal interrelationships, e.g., inclusion dependencies, PK/FK, etc., are not explicitly known.

In Figure 1, the highlighted inter–table relationships are often missing in practice. However, they are crucial for efficient querying of multiple tables and for the creation of the unified views. With respect to relationship discovery, in this paper, we propose a method to efficiently approximate an entity–relationship diagram to uplift the efficiency of query writing by data scientists.

## 4 THE VOYAGER SYSTEM

At Peak AI, together with data scientists, we have identified data discovery and integration as a suitable focal point for accelerating the delivery of data science results. We are, therefore, proposing in this section a system called *Voyager*, built on top of indexing–based data discovery [5] and focused on delivering the knowledge $\mathcal{K}$ from Section 3 to data scientists.

Specifically, we guided the development of Voyager by the following design principles:

(1) *Integration*: one system for all of data identification, data understanding and data relationship discovery.
(2) *Automation*: automate the major data derivation processes, to increase data preparation efficiency and productivity.
(3) *Scalability*: read the data once and extract all relevant information for recurrent tasks, i.e., one–off data scanning.
(4) *Accessibility*: focus on easy adoption by data scientists and minimise their learning curve.

[1]This use–case is modeled after a real-world scenario at Peak AI. We cannot use the real use–case because of data and business privacy restrictions.

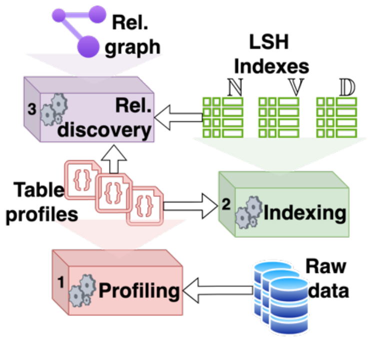

**Figure 2: Voyager's interconnected backend processes and data structures: table profiles, LSH indexes, relationship graph.**

**Table 1: Table & column profile information**

| Profiling info | Description |
|---|---|
| Cardinality | Table cardinality |
| Arity | Table arity |
| Key candidate | The column with the highest likelihood of being a key. |
| Data type | One of categorical, boolean, integer, numeric, date/time, for each column. |
| Summary stats. | Summary statistics for each column, such as counts, unique and missing ratios, min, avg., etc. |
| Histogram | Count histogram for numerical columns. |
| Frequent values | The top-k most frequent values for categorical columns. |
| MinHash[7] | A hash code for each categorical and integer columns used during indexing. |

The result is a system with its main processes and data structures illustrated in Figure 2. Voyager has components for three main exploratory tasks, *viz.*, data profiling, data indexing and relationship graph building, leveraging three types of metadata, each stored in a specialised data structure. These tasks collectively address the design principles mentioned above, as we now describe.

## 4.1 Data profiling

Although *data profiling* is an established research area (e.g., [33]), in this paper we limit the reach of profiling to only examining the data from an existing information source (e.g., a database) and collecting descriptive statistics and informative summaries about individual data items (e.g., tables, columns).

In our experience, we have observed that this type of profiling is performed for most analytical tasks early on. Often, profiling is done by programs and scripts written by data scientists that extract metadata of interest for the current task, e.g., using profiling libraries such as the pandas_profiling Python library [35]. We have identified several profiling statistics that are often useful for understanding new data, and we mention the most important ones in Table 1.

While most of the items in Table 1 are self–explanatory, of particular interest are *Key candidate* and *MinHash*. The former identifies a column as being a key candidate in a profiled table following a heuristic approach inspired by the idea of *subject attributes* introduced in [40]. However, a subject attribute is often

**Table 2: Key score factors**

| | | |
|---|---|---|
| $d$ | ratio of distinct values | $\in [0,1]$ |
| $m$ | ratio of missing values | $\in [0,1]$ |
| $s$ | skewness measure | normalized to $\in [0,1]$ |
| $i$ | column ordinal index | normalized to $\in [0,1]$ |

restricted to categorical columns, i.e., the entity the dataset is about. Since we use the identified key candidates in join path approximation, described in Section 4.3, we also consider integer columns. Specifically, we score each column's key candidacy using the following heuristic:

$$k_{score} = d \times (1-m) \times (1-s) \times (1-i) \tag{1}$$

where Table 2 describes the individual factors. Intuitively, a column is more likely to be a key candidate if most of its values are unique and not missing, its count histogram, when data is numeric, is not left or right skewed and the column is positioned to the left of the table.

With respect to *MinHash*, we obtain it using a special hash function with the same name and characterised by outputs with high probability of collision for similar inputs. [7]. It has been proposed for efficient approximate similarity search and, therefore, we are using it during similarity indexing, as described later in Section 4.2.

We use profiling to deliver *data understanding*. We do so efficiently, by automating the extraction of profiling information such as those exemplified in Table 1. Therefore, data profiling contributes to the *automation* principle mentioned before, since new incoming data is presented to data scientists with profiling information already extracted. This information is stored per table in JSON files that act as backend data for subsequent tasks. Therefore, table profiles contribute to the *scalability* principle as well, by acting as data summaries that prevent a re-scanning of the data by the other processes, as we describe next.

## 4.2 Data indexing

By *data indexing* we refer to the process of grouping data columns from different tables based on multiple levels of similarity with the purpose of identifying tables and columns with values drawn from the same domain.

The motivation for data indexing lies in the need for efficient data search and relationship discovery, as we describe in Section 3.4. In Voyager, data indexing is inspired by the $D^3L$ data discovery system [5] and in transferring the ideas advanced by $D^3L$ in practice we rely on three types of evidence for column similarity: names ($\mathbb{N}$), values ($\mathbb{V}$), and domain distributions ($\mathbb{D}$). From column names we derive $q$–grams, meaning that each name is split into its $q$-grams of predefined size. From the value sets of categorical columns we derive tokens meaning that each value of a column is split into its set of word components that is then unioned with all other word sets of the same column. From the value sets of numerical columns we derive domain distributions in the form of histograms of all column values, already computed by the profiler. We thus cover both categorical and numerical data and gather fine–grained similarity evidence for the former, accounting for cases where columns may have names or values that denote the same entity but are represented differently.

Let $c$ and $c'$ be columns with extents $[[c]]$ and $[[c']]$, respectively. Similarly to $D^3L$, we capture name and value similarity

**Algorithm 1** Index construction

**Input**: A collection $C$ of column names with associated profiles $(c_i, P(c_i))$.
**Output**: Indexes $I_{\mathbb{N}}, I_{\mathbb{V}}, I_{\mathbb{D}}$

```
1:  function CreateIndexes
2:      $I_{\mathbb{N}} \leftarrow$ new LSH(); $I_{\mathbb{V}} \leftarrow$ new LSH(); $I_{\mathbb{D}} \leftarrow$ new LSH();
3:      for all $(c_i, P(c_i)) \in C$ do
4:          $Q(c_i) \leftarrow$ get_qgrams($c_i$)
5:          $I_{\mathbb{N}}.insert$(MinHash($Q(c_i)$))
6:          if dtype($c_i$) is numeric then
7:              $H(c_i) \leftarrow$ get_histogram($P(c_i)$)
8:              $I_{\mathbb{D}}.insert$(RandomProjections($H(c_i)$))
9:          else
10:             $MH(c_i) \leftarrow$ get_minhash($P(c_i)$)
11:             $I_{\mathbb{V}}.insert(MH(c_i))$
12:         end if
13:     end for
14:     return $(I_{\mathbb{N}}, I_{\mathbb{V}}, I_{\mathbb{D}})$
15: end function
```

using Locality Sensitive Hashing (LSH) indexing [4, 23] over $q$–gram sets and token sets, respectively. In other words, the similarity between the names of $c$ and $c'$ will be given by the Jaccard similarity of their $q$-gram sets. The similarity between the values of $c$ and $c'$ will be given by the Jaccard similarity between $[[c]]$ and $[[c']]$, modelled using the token sets of each column to account for inconsistent representations. We use MinHash [7] as the hashing function underlying LSH: a hash function characterised by outputs with high probability of collision for inputs with high Jaccard similarity.

With respect to domain distribution, we propose a new LSH–based similarity estimation method, as the one proposed in $D^3L$ does not apply when all columns of a table are numeric or when the key candidate is numeric. Specifically, for numerical columns, we create a new LSH index that groups together columns with similar value distributions represented as histograms. This time, the underlying hash function of the LSH index is *random projections* [10]: a hash function characterised by outputs with high probability of collision for inputs with high Cosine similarity. So the similarity between the domain distributions of $c$ and $c'$, when they are numeric, is given by the Cosine similarity between their value histograms seen as numerical vectors. The overall data indexing procedure of Voyager is shown in Algorithm 1.

In Algorithm 1, functions get_histogram and get_minhash retrieve a column's histogram and MinHash from the profile of the parent table. Therefore, CreateIndexes does not need to access the original data, as per Voyager's *scalability* principle.

*4.2.1 Data search.* The output triple of CreateIndexes provides the backend data structures for the data search functionality offered by Voyager and used to underpin data identification. We describe the motivation of data search with respect to the use case from Figure 1 where a data scientist, needs to identify the concepts represented in the tables. In particular, a data scientist looking for the tables/columns with product sales information needs to identify table *Sales invoice* as being relevant but also *Orders* and *Transactions*. Voyager offers such functionality by implementing a combined search strategy, described in Figure 3, that uses all three indexes, while abstracting from the user the complexity of querying each LSH index.

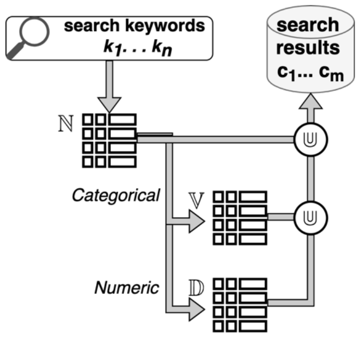

**Figure 3: Voyager's combined search strategy.**

Returning to the product sales example, consider the search keyword set {sales}. As per Figure 3, initially $I_{\mathbb{N}}$ will return result columns such as *Sales invoices.sales order id*, *Orders.sales type*. This result set will be forwarded to $I_{\mathbb{V}}$ and $I_{\mathbb{D}}$, subject to the forwarded column data type. In our example, the value distribution of *Sales invoices.sales order id* will be used to query $I_{\mathbb{D}}$ and the result, {*Orders.order number*} will be unioned with the initial result from $I_{\mathbb{N}}$ and presented to the user.

The intuition behind the search strategy from Figure 3 is to abstract a multi–criteria search behind a simple keyword–based entry point. In this way, users can attend to columns/tables that may be relevant for data identification even if they are not a direct match of the search keywords.

*4.2.2 Search results ranking.* The multi–criteria search method proposed in Figure 3 implies the need for a result ranking mechanism. With Voyager, we implement a simple ranking function that takes into account the index type the results originate from, the number of columns per table that are similar to the search keywords, and the similarity of each result column to the query, as approximated by LSH. Specifically, taking into account that we want each search result item to be a table, given a keyword $w$, the name–score $r_T^{\mathbb{N}}$ of a candidate table $T$ is given by the average similarity between the column names in $T$ and $w$, as approximated by $I_{\mathbb{N}}$: $r_T^{\mathbb{N}} = \sum_{i=1}^{k} s_i^{\mathbb{N}}/k$, where $s_i^{\mathbb{N}}$ is the similarity between column name $c_i$ of $T$ and $w$. Similarly, the extent–score $r_T^{\mathbb{V}/\mathbb{D}}$ of a candidate table $T$ is given by the average similarity between the column extents in $T$ and the extent of $c_i$ - the name–similar column used as a query for $I_{\mathbb{V}}$ or $I_{\mathbb{D}}$ - adjusted by the name similarity of $c_i$ to the original keyword $w$: $r_T^{\mathbb{V}/\mathbb{D}} = \sum_{i=1}^{k} s_i^{\mathbb{N}} s_i^{\mathbb{V}/\mathbb{D}}/k$.

We, therefore, treat name–similarity and extent–similarity independently and present both rankings to the user. The intuition here is to extend the reach of Voyager's search mechanism beyond simple column/table name similarity, while preserving interpretability of the resulting ranking. Such interpretability would be lost if, for example, Voyager follows a combined ranking function, similar to the one proposed in $D^3L$[5].

Finally, we use data indexing and search to deliver *data identification*. We do so efficiently by building on LSH and, therefore, adhering to the *scalability* principle because LSH is characterised by linear retrieval times w.r.t. the search space size [30]. Moreover, the indexing process in Voyager is fully automated and, therefore, addresses the *automation* principle. With respect to the search process, Voyager abstracts the complexities of LSH querying behind a simple keyword search process[2] and, therefore, addresses the *accessibility* principle we aim for.

## 4.3 Relationship graph creation

Given a collection of tables $\mathcal{T}$, a *relationship graph* is a data structure $G := (T, E)$, with the set of vertices $T \subseteq \mathcal{T}$ and the set of edges $E := sim(T_i, T_j)$, given by a *similarity relationship* between some column in $T_i$ and some other column in $T_j$.

The motivation for building a relationship graph is given by the need for joining tables. We identified this task as a the most time–consuming steps. This is because incoming tables, such as the ones illustrated in Figure 1, often originate from data management systems that do not enforce primary–keys or integrity constraints, e.g., primary-key/foreign-key relationships. Therefore, their relationships are often unknown. We proceed to approximate such relationships in the form of candidate join paths, i.e., partial and full inclusion dependencies (INDs), (e.g., similar, but not limited to the ones illustrated in Figure 1). This approximation is grounded in the notion of similarity, denoted by $sim()$ above, between table columns. In defining $sim()$ we build on top of table profiles and LSH indexes to identify the set of edges $E$ as column extent overlaps. Given two columns $c_i$ and $c_j$, their similarity is quantified by their *value–set overlap* measured using the *overlap coefficient*, $ov(c_i, c_j) = |[[c_i]] \cap [[c_j]]| / \min(|[[c_i]]|, |[[c_j]]|)$. We differentiate between two overlap cases, both describing valid join opportunities:

- *Primary join paths* (PJPs) are approximated full INDs where some key candidate from a table $T_i$ fully includes the values of a column from $T_j$.
- *Secondary join paths* (SJPs) are approximated partial INDs where some key candidate from a table $T_i$ partially includes the values of a column from $T_j$.

More formally, the overlap coefficients of PJPs and SJPs are defined by

$$ov_{PJP}(c_i, c_j) \geq 1 - e \tag{2a}$$

$$ov_{SJP}(c_i, c_j) \geq t \tag{2b}$$

where $e$ is the PJP overlap tolerance coefficient, and $t$ is the SJP overlap threshold. Both are configurable parameters fixed in our experiments to 0.1 and 0.6, respectively. We describe their intuition next.

### 4.3.1 *Primary Join Paths.*

PJP relationships are approximated full INDs that we are using LSH to identify. Alternatively, IND discovery tools (e.g., [36]) could be used. Such approaches, however, require clean data, uncommon when working with external data maintained by non-specialised users. Additionally, the LSH indexes are already created so PJP discovery is actually a data search problem controlled by $e$ - a tolerance factor we configure to take account of the approximate nature of the similarity estimated by LSH.

### 4.3.2 *Secondary Join Paths.*

Similarly, SJP relationships are approximated partial INDs and we are using the same LSH structures to identify them. Their discovery can also be treated as a data search problem controlled by $t$ - a minimum overlap value for two columns to be in a SJP relationship.

[2]The LSH configuration parameters, such as MinHash size and number of inner hash tables, have been empirically fixed to reasonable defaults for a similarity score of at least 0.7: 256 and 50, respectively.

**Algorithm 2** Join Path Discovery

**Input**: Key candidate set $K$ and index $I_{\mathbb{V}}$
**Output**: Collections of $PJPs$ and $SJPs$

```
 1: function GetJoinPaths
 2:     $PJPs \leftarrow \{\}$; $SJPs \leftarrow \{\}$
 3:     for all $kc_i \in K$ do
 4:         $neighbours \leftarrow I_{\mathbb{V}}$.lookup($kc_i$)
 5:         for all $n_j \in neighbours$ do
 6:             if ov($[[kc_i]]$, $[[n_j]]$) >= 1 − e then
 7:                 $PJPs \leftarrow PJPs \cup \{(kc_i, n_j)\}$
 8:             else if ov($[[kc_i]]$, $[[n_j]]$) >= t then
 9:                 $SJPs \leftarrow SJPs \cup \{(kc_i, n_j)\}$
10:             end if
11:         end for
12:     end for
13:     return $(PJPs, SJPs)$
14: end function
```

We rely on $I_{\mathbb{V}}$ to efficiently identify the PJP and SJP candidates[3] using the procedure from Algorithm 2, where the $I_{\mathbb{V}}$ LSH index is used to efficiently approximate the overlap coefficient between column extents[4] (Lines 6 and 8). Specifically, given two columns $c_i$ and $c_j$, similarly to Zhu *et al.* [44], we observe that the overlap coefficient $ov([[c_i]], [[c_j]])$ (denoted simply as $ov$) is *positively correlated* with Jaccard similarity $jacc([[c_i]], [[c_j]])$ (denoted simply as $jacc$) and that $jacc$ can be approximated by counting the number of buckets $c_i$ and $c_j$ share in $I_{\mathbb{V}}$ [23]. Thus, we can use the Jaccard similarity to identify high–overlap candidates in $I_{\mathbb{V}}$. Then, using the set–theoretic inclusion–exclusion principle, it follows that $ov = \frac{jacc \times (|[[c_i]]| + |[[c_j]]|)}{1 + jacc} \times \frac{1}{min(|[[c_i]]|, |[[c_j]]|)}$. With the extent cardinalities stored in table profiles at profile time and $jacc$ given by $I_{\mathbb{V}}$, $ov$ becomes straightforward to compute.

Returning to our relationship graph, the set of relationship graph edges $E = PJPs \cup SJPs$ and the set of vertices $T$ is given by the set union of all tables with columns in some PJP or SJP. We use the resulting graph $G$ to deliver *data relationship discovery*, materialised as set–value overlap. We do so by building on top of the existing LSH indexes and, therefore, preserving the adherence to the *scalability* principle. *Automation* is addressed as well, since all major relationship discovery tasks happen automatically, including updates when necessary, i.e., updated/new tables are (re)profiled and re(indexed) automatically, while the relationship graph is updated in the light of $I_{\mathbb{V}}$'s changes.

## 4.4 User interaction model

In this paper we introduced a collection of tasks data scientists have to perform to overcome the semantic gap between their intents and new incoming data. Therefore, the capabilities of Voyager have to be immediately accessible to data scientists, especially since increased productivity is one of the motivating factors of Voyager. As such, we designed Voyager to be accessible and expose its backend functionalities through a command line interface (CLI), while the frontend is accessible as a JupyerLab [38] extension, as we now describe.

### 4.4.1 *Backend CLI.*

Voyager's backend processes, *viz.* data profiling, data data indexing, relationship graph creation, are

[3]We also include integer–valued columns in $I_{\mathbb{V}}$ by considering each integer value as a word/token.
[4]$D^3L$ [5] has shown good performance in approximating value set overlaps using an index such as $I_{\mathbb{V}}$, although the index stores value tokens rather than full values.

**Table 3: Voyager's backend CLI commands.**

```
# voyager create -help

Usage: voyager create [OPTIONS] COMMAND [ARGS]

Options:
  -help : Show this message and exit.

Commands:
  profiles : Create a profile for all tables in a database.
  indexes : Create indexes for all profiled tables.
  graph : Create relationship graph for all profiled tables.
```

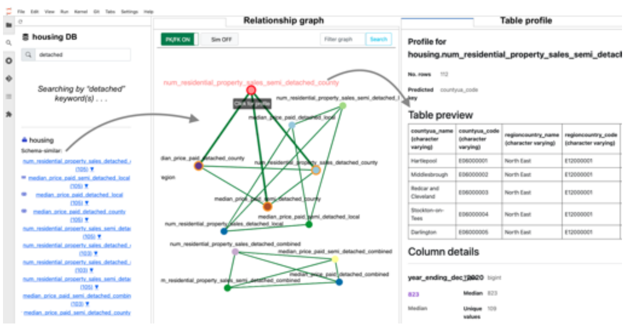

**Figure 4: Voyager's frontend where the relationship graph and table profiles can be accessed from the search interface.**

accessible to data scientists as CLI commands that can be used to create automatic processes, e.g., *cron jobs*, to add new data into the table profiles, LSH indexes, and a relationship graph. Voyager's backend is controlled through the three sub–commands from Table 3. The `ARGS` are sub–command specific and include configuration parameters, such as MinHash size for `profiles`, number of LSH hash tables for `indexes`, or PJP tolerance coefficient and SJP overlap threshold for `graph`.

*4.4.2 Frontend extension.* Voyager exposes data profiles, indexes and the relationship graph through an extension to Jupyter Notebook [38], one of the main development environments used by data scientists. Figure 4 illustrates a simple version this frontend that we used for this paper's experiments. The left most interface consists of a table browsing component that also acts as a user interface for consulting the results of the search strategy from Figure 3. The relationship browsing interface from the middle exposes the relationship graph where PJPs and/or SJPs can be filtered out to simplify the view. Graph nodes, i.e., tables, are labeled with their corresponding table names and can also be filtered to avoid a convoluted graph. Thirdly, the right most table profile interface presents profiling information, such as the one from Table 1. This can be accessed either from the browsing interface or by clicking on a relationship graph node. Using this three interfaces, users can explore the different tables in a database, their inter–relationships and zoom–in on individual table profiles.

Such a design, with decoupled backend and frontend components, allows users to configure backend processes to create and keep data structures up to date using the CLI commands. The results can then be consulted in the user interface so that users acquire the knowledge $\mathcal{K}$ from Section 3 faster than using

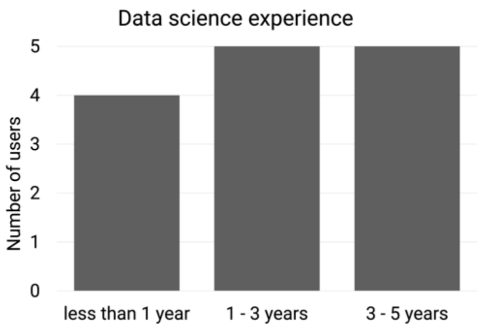

**Figure 5: Data science experience distribution.**

some other data exploration method, as we next describe in our evaluation.

## 5 EXPERIMENTAL DESIGN

The main motivating factor for building Voyager is the observed time costs of data discovery and integration. Data scientists interact with new data often, and overcoming the semantic gap discussed in Section 3 by identifying, understanding and querying different data components can lead to productivity costs in the delivery of data analysis results. In this section, we design experiments to evaluate the hypothesis that Voyager facilitates data identification, data understanding and relationship discovery and, consequently, increases the productivity of data scientists. We do so by relying on implementations of the backend processes in Python 3.7 and of a simple version of the frontend from Figure 4 as a Jupyter Notebook 6.4 extension. We then ask data scientists to perform six specifically designed tasks, as described next.

### 5.1 Methodology

We evaluate our usability and productivity hypothesis by employing a three–factor analysis:

(1) We use a general usability measurement framework in the form of the 10–question *System Usability Scale* (SUS) questionnaire [3, 8] to measure the subjective assessment of Voyager's usability by data scientists.
(2) We compose an additional 9–question questionnaire to analyse the usefulness of Voyager's individual components, *viz.* data profiles, data indexes, and relationship graph, in acquiring each of data identification, data understanding and relationship discovery.
(3) We perform a comparative time–analysis between two data scientist sub–groups with and without the use of Voyager to assess the time–cost impact of using Voyager.

Our subject pool for Voyager's evaluation consists of Peak's *14 data scientists* with a balanced data science background between a few months and 5 years. Figure 5 shows the distribution of the users experience in data science.

We ask our subjects to perform *six analytical tasks* over *40 new datasets*. To this end, we randomly split the data scientists pool into two equal groups: group $\mathcal{V}$, or subjects who use Voyager, and group $\mathcal{O}$, or subjects who use some other method for exploring the data, e.g., SQL querying, Python scripts, etc.. The six analytical tasks are given to group $\mathcal{V}$ in a Jupyter Notebook with the Voyager extension installed and with the instruction to use it for searching and exploring the data and its relationships. A Jupyter Notebook with the same tasks but without the Voyager

Table 4: Generic and concrete usability study tasks.

| Task | Generic desc. | Concrete task |
|---|---|---|
| *1. (I)* | Identify relevant tables and columns that store specific concepts. | *Which tables contain information about the number of detached and semi-detached house sales between 1995 and 2020 at the level of local authorities?* |
| *2. (I)* | | *Which tables contain information about the prices paid for detached and semi-detached houses between 1995 and 2020 at the level of local authorities?* |
| *3. (U)* | Aquire statistical descriptions of concepts represented in the data. | *What was the median number of detached property transactions across local authorities in 2020?* |
| *4. (U)* | | *What was the highest price paid for a semi-detached house across local authorities in 2019 ?* |
| *5. (R)* | Identify how relevant tables relate. | *Merge the tables from tasks 1.1 and 1.2 into a unified view.* |
| *6. (I,U,R)* | Perform a data analysis task that requires data discovery and integration. | *How many terraced properties have been transacted and what was their average price in Manchester between 2018 and 2020 ?* |

extension is presented to group $O$. So these users perform the tasks without the use of Voyager. We measure the time required to perform the six tasks by each group and perform evaluation method (3) by cross–analysing the results. Then, we swap the groups and ask them to perform the tasks again. So group $O$ will have Voyager access while $\mathcal{V}$ will not. Consequently, each user has the chance to use Voyager and then answer the questionnaires from evaluation methods (1) and (2).

The six analytical tasks are described in Table 4. They are based on 40 tables, further discussed in Section 5.2, that are new to the experiment subjects so data discovery and/or integration is necessary in order to perform the tasks. Specifically, tasks 1 and 2 require data identification - the tables can be identified using Voyager's search functionality. Tasks 3 and 4 require data understanding - the statistics can be extracted from the table profiles. Task 5 requires relationship discovery - the value overlap relationships can be identified in the relationship graph. Task 6 requires all of data identification, data understanding and relationship discovery since it implies the need for aggregations over a join result of multiple tables.

The subjects are required to perform all tasks in order. As such, the role of tasks 1, 3, and 5 is to elicit the need for different knowledge when confronted with new datasets and analytical jobs. The purpose of tasks 2 and 4 is to elicit the contribution of identification and understanding abilities when performing repeated actions on the data (i.e., tasks 2 and 4 will be easier once tasks 1 and 3 have been performed because the user will be more familiar with the data). Similarly, the purpose of task 6 is to elicit the ease of performing data analysis once the semantic gap has been overcome. After group swapping, each subject re–executes the same tasks on the same data so the semantic gap will be even narrower. However, we hypothesise that the advantages of using Voyager will still be recognisable.

Table 5: Number of residential property sales by local authority - sample.

| Authority name | Region name | Dec 2020 | Sep 2020 | ... | Dec 1995 |
|---|---|---|---|---|---|
| Hartlepool | North East | 39 | 81 | | 213 |
| Middlesbrough | North East | 155 | 288 | | 257 |
| Darlington | North East | 128 | 199 | | 215 |

Table 6: *SUS* statements.

| No. | Statement |
|---|---|
| *1.* | *I think that I would like to use Voyager frequently.* |
| *2.* | *I found Voyager unnecessarily complex.* |
| *3.* | *I thought Voyager was easy to use.* |
| *4.* | *I think that I would need the support of a technical person to be able to use Voyager.* |
| *5.* | *I found the various functions in Voyager were well integrated.* |
| *6.* | *I thought there was too much inconsistency in Voyager.* |
| *7.* | *I would imagine that most people would learn to use Voyager very quickly.* |
| *8.* | *I found Voyager very cumbersome to use.* |
| *9.* | *I felt very confident using Voyager.* |
| *10.* | *I needed to learn a lot of things before I could get going with Voyager.* |

## 5.2 Evaluation dataset

We design our evaluation based on a *40–table* data collection originating from the UK's Office for National Statistics[5]. The tables contain information about the number of property transactions and the median price paid for these transactions across various geographies in England and Wales, on an annual basis, updated quarterly, between 1995 and 2020. The data is organised in tables by property types, e.g., detached houses, flats, etc., and administrative levels, e.g., region, city, borough, etc.. Table 5 shows a sample of the information about the number of residential property sales by local authority in the North East of England.

## 5.3 Reported measures

The results of each evaluation method from Section 5.1 are quantified using the following measures:

*SUS Score*: The SUS questionnaire used with evaluation method (1) is composed of the 10 statements from Table 6 that are scored from 1 to 5, i.e. strong disagreement to strong agreement. Each odd statement measures a positive judgement, while each even statement measures a negative judgement. Then, following the proposal from [8], the *SUS score* is given by:

$$2.5 \times \left[ \sum_{1 \leq i \leq 10, \text{i odd}} (s_i - 1) + \sum_{1 \leq j \leq 10, \text{j even}} (5 - s_j) \right] \quad (3)$$

where $s_{i/j}$ is the user–given score for statement $i/j$. Equation 3 thus normalizes the positive and negative character of the statements to a *[0, 100]* score. We report and analyse the scores in Section 6.1.

*Functional usefulness percentages*: The questionnaire used with evaluation method (2) consists of the 9 statements from Table

[5]https://www.ons.gov.uk/peoplepopulationandcommunity/housing/bulletins/housepricestatisticsforsmallareas/yearendingdecember2020

Table 7: Functional usefulness statements.

| No. | Statement |
|---|---|
| *1.* | *I found the table profiles useful for data identification.* |
| *2.* | *I found the relationship graph useful for data identification.* |
| *3.* | *I found the search functionality useful for data identification.* |
| *4.* | *I found the table profiles useful for data understanding.* |
| *5.* | *I found the relationship graph useful for data understanding.* |
| *6.* | *I found the search functionality useful for data understanding.* |
| *7.* | *I found the table profiles useful for rel. discovery.* |
| *8.* | *I found the relationship graph useful for rel. discovery.* |
| *9.* | *I found the search functionality useful for rel. discovery.* |

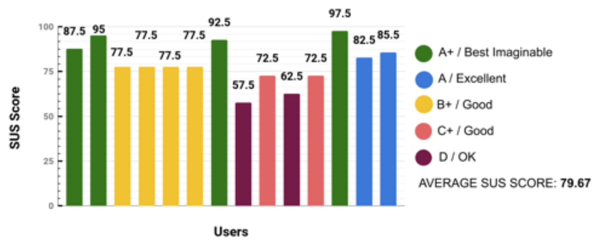

**Figure 6: System Usability Study score for Voyager showing that 10/14 users rate Voyager's usability at least *Good.***

7, scored from 1 to 5. Each statement assesses the usefulness of one of Voyager's main functions. In Section 6.2, we report and analyse the users responses the the nine statements.

Additionally, we asked each data scientist to score from 1 to 5 Voyager's overall potential for increasing productivity using one statement: *I think that productivity increases when using Voyager, as opposed to using other data exploration methods.* We report and analyse the responses in Section 6.2.

*Task times*: For evaluation method (3) we report the times required by data scientists in groups $\mathcal{V}$ and $O$ to perform the six tasks when each group first encounters the data, i.e., before swapping the two groups. We do not include the after–swap times in the analysis because, by that point, each subject is already familiar with some of the tables, i.e., the semantic gap is narrower. We report and analyse the results for this experiment in Section 6.3.

## 6 RESULTS

In this section we report the results for each of the three evaluation methods from Section 5.1.

### 6.1 Evaluation method (1): System Usability Scale

In this experiment, we evaluate Voyager's perceived usability quantified by the SUS score from Equation 3. We include in our measurement all 14 subjects once they had the chance to use Voyager. The results are reported in Figure 6, including the individual score values returned by Equation 3 and the corresponding adjective and grades according to SUS's alternative scales proposed by Bangor *et al.* [3]. While three out of 14 subjects graded Voyager's usability with *A+* and another two with *A*, the average SUS score of *79.67* corresponds to grade *A-*. This speaks about Voyager's potential for being recommended to other data scientists and used frequently. In fact, 13 subjects agreed or strongly agreed when specifically asked about a more frequent use of Voyager.

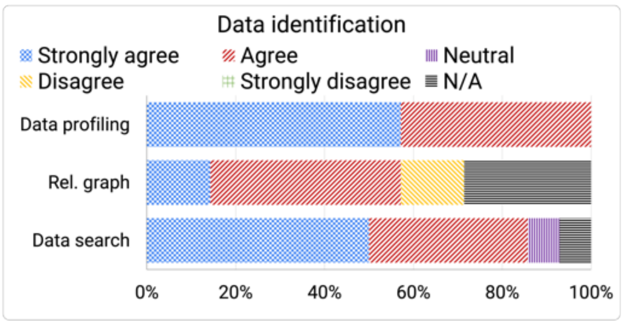

**Figure 7: Evaluation of Voyager's main functionalities w.r.t. data identification showing that 12/14 users agree with the usefulness of data search.**

Conversely, four users graded Voyager's usability with *D* or *C+*. A closer analysis of the reasons behind these grades suggested that the respective users considered the three main functions of Voyager not well integrated and observed inconsistencies in how these can be accessed. In addition, the two users who graded the system with *D* considered Voyager either cumbersome to use or desired technical support at their first use of the tool. We observe that most of these issues may result from the prototype nature of Voyager's interface. A more advanced user interface design, together with more user training could therefore lead to an increase in the SUS score.

### 6.2 Evaluation method (2): Functional usefulness

In this experiment we specialize our evaluation study in the direction of usefulness. To this end, we asked the 14 data scientists to evaluate the usefulness of Voyager's main functionalities, *viz.*, data profiling, data searching, and relationship graph creation, with respect to the three data branches introduced in Section 3, *viz.*, data identification, data understanding, and relationship discovery.

The motivation for this experiment is our hypothesis that data identification can be facilitated by Voyager's indexing & search functionality, data understanding can be facilitated by Voyager's table profiles and data relationship discovery can be facilitated by Voyager's relationship graph.

*6.2.1 Data identification.* Figure 7 shows that 12 out of 14 subjects have found Voyager's data search functionality useful for Tasks 1, 2 and 6 from Table 4. The two users who did not agree used only the browsing capabilities of Voyager's interface, illustrated in Figure 4, and the table profiles to identify the intended tables for the same tasks. This is in line with the *100%* agreement of data profiling usefulness illustrated in Figure 7. Moreover, the relationship graph has also proven useful for identifying data for more than half of the subjects. This is because the tables and columns needed to be identified for Tasks 1, 2, 6 are neighbours in the relationship graph.

*6.2.2 Data understanding.* Figure 8 shows that 13 out of 14 subjects have found Voyager's data profiling functionality useful for Tasks 3, 4 and 6 from Table 4. This is because the profiles for the corresponding tables either contain the values of the required statistics (e.g., tasks 3 and 4) or can provide information to compute them (e.g., task 6). In addition, the relationship graph

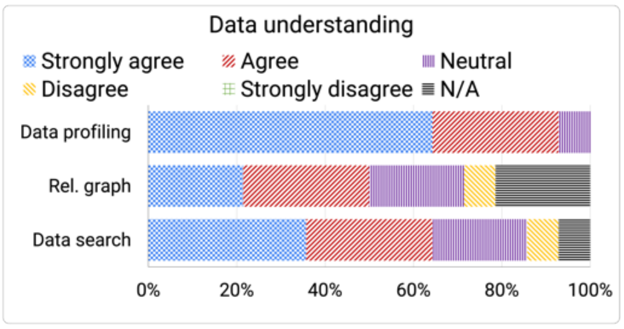

**Figure 8: Evaluation of Voyager's main functionalities w.r.t. data understanding showing that 13/14 users agree with the usefulness of data profiling.**

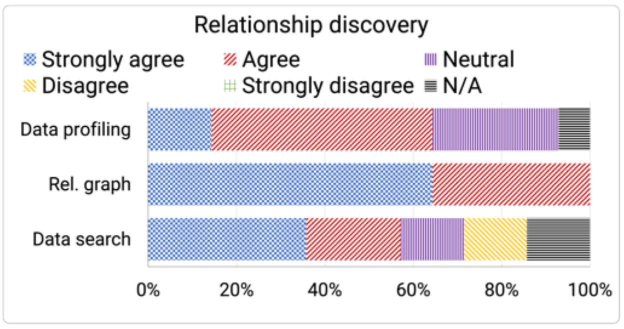

**Figure 9: Evaluation of Voyager's main functionalities w.r.t. relationship discovery showing that all users agree with the usefulness of the relationship graph.**

and data search functionalities have been used by at least half of the users when trying to understand the data.

*6.2.3 Relationship discovery.* Figure 9 shows that all 14 subjects have found Voyager's relationship graph useful for Tasks 5 and 6 from Table 4. This is because the relationship graph emphasizes approximated join paths that, as is often the case in practice, are not explicitly defined in our evaluation data collection. Data scientists can, thus, use the graph paths to identify joining keys. Additionally, 9 users consider the information from table profiles useful for joining dataset. However, in scenarios with more than a few tables to join, visually analyzing table profiles to identify joining keys becomes impractical. Searching for similar tables in Voyager could also be used to find potential joining columns and at least 8 users have identified this possibility. This can be done using the value–similar search results to extract tables with value–sharing columns.

*6.2.4 Productivity increase.* In addition to the nine statements about individual functionalities, we asked users to evaluate Voyager's overall potential for increasing productivity. The results reported in Figure 10 suggest that the vast majority of subjects, i.e., 12 out of 14 users, agree with our hypothesis that Voyager has such a potential. One user was undecided, while one other user disagreed with our hypothesis. After a closer analysis we conclude that users who were not convinced by Voyager's increased productivity potential also wished for more technical support in using the tool. This suggests that a better clarification of Voyager's technical concepts and supporting the tool's initial adoption could further reveal its productivity growth potential.

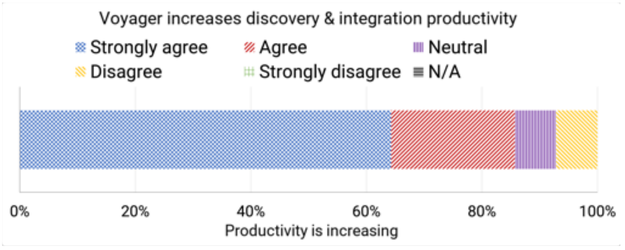

**Figure 10: Evaluation of Voyager's productivity potential showing that 12/14 users agree that use of Voyager can increase productivity.**

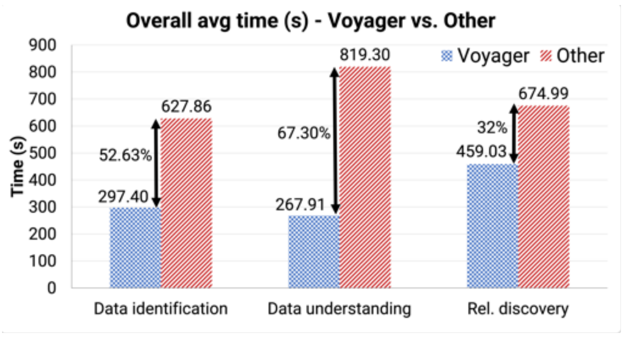

**Figure 11: Time measurements between group $\mathcal{V}$ (i.e., Voyager) and group $O$ (i.e., Other) showing time savings when using Voyager.**

## 6.3 Evaluation method (3): Task times

In this experiment we evaluate our hypothesis that Voyager increases productivity with respect to time. To this end, we report and cross–analyze the times required for user groups $\mathcal{V}$ and $O$ to perform the tasks from Table 4.

*6.3.1 Overall time analysis.* Figure 11 emphasizes the difference in the average time measurements between the two user groups. Each graph bar reports the average group times over the sums of two task, i.e., we sum the times of two tasks for each user in the group and report the average of all sums in that group. More specifically, data identification includes times for tasks 1 and 2, data understanding includes times for tasks 3 and 4, and relationship discovery includes times for tasks 5, 6. Although task 6 covers all three branches, the most challenging part is identifying table relationships. Therefore, we include task 6 in relationship discovery measurements.

The figure suggests that identifying tables and columns using Voyager's browsing or search interface is 52% more efficient than using some other exploration method. This difference increases to 67% for extracting summary statistics about identified data elements. This is because data understanding often involves writing exploratory queries, e.g., in SQL, and the table profiles relieve the users from writing some of these queries.

Discovering table relationships and writing join queries is 32% more efficient when using Voyager. The improvement is, therefore, smaller than in the case of data identification and data understanding because by the time they reach tasks 5 and 6, both user groups have overcome part of the semantic gap - recall that users perform the tasks in order. This could also explain why data understanding tasks turned out more challenging for group $O$ users than relationship discovery. The 32% improvement also

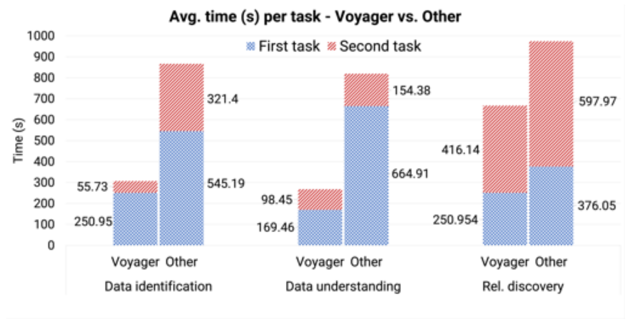

**Figure 12: Per–task time measurements between group $\mathcal{V}$ (i.e., Voyager) and group $O$ (i.e., Other) showing significant time savings for each task when using Voyager.**

suggests that even after users become more familiar with the data, Voyager can still enhance the data querying process.

Lastly, we performed two–sample t–tests on the time data corresponding to each of data identification, data understanding and relationship discovery. In each case, the two samples are given by group $\mathcal{V}$'s times and group $O$'s times, each with 7 samples. The three tests did not yield statistically significant results. This is because we have prioritised obtaining results from professional data scientists, but the consequence of this is that the sample sizes are quite small. If we consider the results of each of the tasks together, thus asking if individual (non–specific) tasks were quicker with Voyager than using some other data science existing techniques (a total of 21 samples in each group) then the differences are found to be statistically significant: p–value = *0.035*. This statistical significance has been obtained using a Wilcoxon Signed-Rank test [11] instead of a t–test. This is because the latter assumes the samples are normally distributed, a potentially unreasonable assumption when we include measurements from all three tasks.

*6.3.2 Per–task time analysis.* Figure 12 shows a consistent trend at individual task level with the one previously observed in Figure 11. For each user group, *First task* denotes tasks 1, 3, or 5 depending on the branch, i.e., data identification, data understanding or relationship discovery. Similarly, *Second task* denotes tasks 2, 4, or 6. In each case, we report the average over times recorded for each user in their respective group. Once again, we include task 6 in the relationship discovery time analysis.

Data identification and data understanding proves up to three times more efficient when conducted using Voyager than when some other data exploration method is used. In the case of both groups, the second data identification and data understanding tasks took significantly less time to perform than the corresponding first tasks. This pattern emphasizes the importance of overcoming the data semantic gap, i.e., once the user familiarises with the data, repetitive tasks become easier to perform.

With respect to relationship discovery, we first observe that the difference between the times corresponding to the two groups is smaller than in the previous two cases. This is because tasks 5 and 6 require users from both groups to write SQL queries. However, these queries require the pre-identification of joining keys and the relationship graph assists users from group $\mathcal{V}$ in finding them.

Task 6 takes longer than task 5 in the case of both groups. This is because task 5 refers to tables and columns used in the previous tasks as well, while task 6 uses new tables and columns that have to be identified first.

*6.3.3 Summary.* Overall, the analysis from this section shows considerable time–cost reductions when using Voyager. Therefore, it supports our productivity desideratum and encourages further development efforts to address requirements resulted from our user study described in this evaluation.

## 7 CONCLUSIONS

Data discovery and integration have been the focus of considerable research interest. However, there has been less work reported that builds on these foundations to provide platforms that address specific data science needs and support putting these concepts into practice. This paper takes some steps in this direction by proposing a system that blends elements of data discovery and data integration to assists data scientists in interacting with new tabular data. Specifically, in this paper, we address the contributions claimed in the introduction by:

- *The introduction and formalisation of data identification, data understanding and relationship discovery tasks:* Our work is motivated by the requirement within Peak AI to make effective use of customer data for analysis. We have proposed both indexing techniques and a user interface that supports these activities in an integrated manner.
- *A practical system for supporting data discovery and integration, called Voyager.* In the light of the identified functionalities, and the associated performance requirements, we have developed techniques for data profiling, indexing and relationship graph creation that follow a collection of design principles, specifically the integration of the activities in a single environment, the automated derivation of the required data, the scalable population and use of the necessary data representations, and the provision of tools with minimal learning curves.
- *A usability study of Voyager with data scientists.* We have evaluated the resulting system with data scientists from Peak AI, where the data scientists carry out a representative tasks using their existing techniques or using Voyager. In particular, we investigated: (i) the subjective usability of Voyager through a System Usability Scale questionnaire; (ii) the usefulness of the specific components of Voyager through a questionnaire on the utility of the different functionalities for specific tasks; and (iii) the time taken to carry out the different data preparation activities with and without Voyager. This evaluation provided focused, and in general positive, feedback from experienced data scientists, which gives evidence that the specific functionalities can be useful and can improve productivity, even on first use of the system.

## ACKNOWLEDGMENTS

Research supported by a Knowledge Transfer Partnership involving The University of Manchester and Peak AI Ltd., funded by Innovate UK (KTP11540).

## REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDB J.* 24, 4 (2015), 557–581.

[2] Amundsen — Lyft's data discovery & metadata engine [n.d.]. https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9. Accessed: 2021-09-21.

[3] A. Bangor, P.T. Kortum, and J.T. Miller. 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of Human–Computer Interaction* 24, 6 (2008), 574–594.

[4] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web, WWW*. ACM, 651–660.

[5] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *36th IEEE International Conference on Data Engineering, ICDE*. IEEE, 709–720.

[6] Dan Brickley, Matthew Burgess, and Natasha F. Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *The World Wide Web Conference, WWW*. ACM, 1365–1375.

[7] A. Broder. 1997. On the Resemblance and Containment of Documents. In *SEQUENCES*.

[8] John Brooke. 1996. *"SUS-A quick and dirty usability scale." Usability evaluation in industry*. CRC Press.

[9] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. 2020. Dataset search: a survey. *VLDB J.* 29, 1 (2020), 251–272.

[10] M. S. Charikar. 2002. Similarity Estimation Techniques from Rounding Algorithms. In *STOC*.

[11] William Jay Conover. 1999. *Practical nonparametric statistics* (3. ed ed.). Wiley.

[12] Data portal, the data centric tool of AirBnB [n.d.]. https://zeenea.com/data-portal-the-data-centric-tool-of-airbnb/. Accessed: 2021-09-21.

[13] Databook: Turning Big Data into Knowledge with Metadata at Uber [n.d.]. https://eng.uber.com/databook/. Accessed: 2021-09-21.

[14] DataHub: A generalized metadata search & discovery tool [n.d.]. https://engineering.linkedin.com/blog/2019/data-hub. Accessed: 2021-09-21.

[15] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. 2017. SilkMoth: An Efficient Method for Finding Related Sets with Maximum Matching Constraints. *Proc. VLDB Endow.* 10, 10 (2017), 1082–1093.

[16] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration*. Morgan Kaufmann.

[17] Rebecca Eichler, Corinna Giebler, Christoph Gröger, Holger Schwarz, and Bernhard Mitschang. 2021. Modeling metadata in data lakes—A generic model. *Data & Knowledge Engineering* 136 (2021), 101931.

[18] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *34th IEEE International Conference on Data Engineering, ICDE*. IEEE Computer Society, 1001–1012.

[19] Raul Castro Fernandez, Essam Mansour, Abdulhakim Ali Qahtan, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery. In *34th IEEE International Conference on Data Engineering, ICDE*. IEEE Computer Society, 989–1000.

[20] A. Freitas. 2015. *Schema–agnostic queries over large-schema databases: a distributional semantics approach*. Ph.D. Dissertation. National University of Ireland, Galway.

[21] Tim Furche, Georg Gottlob, Leonid Libkin, Giorgio Orsi, and Norman W. Paton. 2016. Data Wrangling for Big Data: Challenges and Opportunities. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT*. OpenProceedings.org, 473–478.

[22] W. Furnas G, T. K. Landauer, L. M. Gomez, and S. T. Dumais. 1987. The Vocabulary Problem in Human-System Communication. *Commun. ACM* 30, 11 (1987), 964–971.

[23] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*. Morgan Kaufmann, 518–529.

[24] Rihan Hai, Sandra Geisler, and Christoph Quix. 2016. Constance: An Intelligent Data Lake System. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD*. ACM, 2097–2100.

[25] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Managing Google's data lake: an overview of the Goods system. *IEEE Data Eng. Bull.* 39, 3 (2016), 5–14.

[26] Joseph M. Hellerstein, Vikram Sreekanti, Joseph E. Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, Mark Donsky, Gabriel Fierro, Chang She, Carl Steinbach, Venkat Subramanian, and Eric Sun. 2017. Ground: A Data Context Service. In *8th Biennial Conference on Innovative Data Systems Research, CIDR*. www.cidrdb.org.

[27] Lacramioara Mazilu, Norman W. Paton, Alvaro A. A. Fernandes, and Martin Koehler. 2022. Schema mapping generation in the wild. *Inf. Syst.* 104 (2022), 101904.

[28] Giansalvatore Mecca, Paolo Papotti, and Salvatore Raunich. 2012. Core schema mappings: Scalable core computations in data exchange. *Inf. Syst.* 37, 7 (2012), 677–711.

[29] Metacat: Making Big Data Discoverable and Meaningful at Netflix [n.d.]. https://netflixtechblog.com/metacat-making-big-data-discoverable-and-meaningful-at-netflix-56fb36a53520. Accessed: 2021-09-21.

[30] Renée J. Miller. 2018. Open Data Integration. *Proc. VLDB Endow.* 11, 12 (2018), 2130–2139.

[31] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. 2020. Organizing Data Lakes for Navigation. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD*. ACM, 1939–1950.

[32] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. 2018. Table Union Search on Open Data. *Proc. VLDB Endow.* 11, 7 (2018), 813–825.

[33] Felix Naumann. 2013. Data profiling revisited. *SIGMOD Rec.* 42, 4 (2013), 40–49.

[34] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *Proc. VLDB Endow.* 14, 12 (2021), 2863–2866.

[35] pandas-profiling [n.d.]. https://pandas-profiling.github.io/pandas-profiling. Accessed: 2021-10-22.

[36] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Profiling with Metanome. *Proc. VLDB Endow.* 8, 12 (2015), 1860–1863.

[37] Thorsten Papenbrock and Felix Naumann. 2017. Data-driven Schema Normalization. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT*. OpenProceedings.org, 342–353.

[38] Project Jupyter [n.d.]. https://jupyter.org/. Accessed: 2021-11-05.

[39] Michael Stonebraker and Ihab F. Ilyas. 2018. Data Integration: The Current Status and the Way Forward. *IEEE Data Eng. Bull.* 41, 2 (2018), 3–9.

[40] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering Semantics of Tables on the Web. *Proc. VLDB Endow.* 4, 9 (2011), 528–538.

[41] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M. Procopiuc, and Divesh Srivastava. 2010. On Multi-Column Foreign Key Discovery. *Proc. VLDB Endow.* 3, 1 (2010), 805–814.

[42] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD*. ACM, 1951–1966.

[43] 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 2019 International Conference on Management of Data, SIGMOD*. ACM, 847–864.

[44] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proc. VLDB Endow.* 9, 12 (2016), 1185–1196.