Analyza: Exploring Data with Conversation

Kedar Dhamdhere
Google Research
Mountain View, USA
<firstname>@google.com

Kevin S. McCurley
Google Research
Mountain View, USA
<lastname>@google.com

Ralfi Nahmias
Google Research
Mountain View, USA
ralfi@google.com

Mukund Sundararajan
Google Research
Mountain View, USA
mukunds@google.com

Qiqi Yan
Google Research
Mountain View, USA
qiqiyan@google.com

ABSTRACT

We describe Analyza, a system that helps lay users explore data. Analyza has been used within two large real world systems. The first is a question-and-answer feature in a spreadsheet product. The second provides convenient access to a revenue/inventory database for a large sales force. Both user bases consist of users who do not necessarily have coding skills, demonstrating Analyza's ability to democratize access to data.

We discuss the key design decisions in implementing this system. For instance, how to mix structured and natural language modalities, how to use conversation to disambiguate and simplify querying, how to rely on the "semantics" of the data to compensate for the lack of syntactic structure, and how to efficiently curate the data.

Author Keywords

Exploratory data analysis; Natural language

ACM Classification Keywords

H.5.2. User interfaces: Natural language

INTRODUCTION AND MOTIVATION

As the father of exploratory data analysis, John Tukey often argued that scientists should spend a lot of time just looking at data. He described it as being analogous to detective work, where a situation is examined without preconceptions to figure out "what is going on here?"

> Science - and engineering, which here includes agriculture and medicine - does not begin with a tidy question. Nor does it end with a tidy answer.
>
> John W. Tukey [36]

IUI 2017 March 13–16, 2017, Limassol, Cyprus

ACM ISBN 978-1-4503-4348-0/17/03.

DOI: http://dx.doi.org/10.1145/3025171.3025227

We should note that this also applies to all data-driven decision making, including business. In situations that are described by complex high-dimensional data sets, this will often require looking at *many* slices of the data. Moreover, the problem is no longer just about examining a single data set to uncover patterns, but the problem is also about finding the data set that provides the most meaningful insight. We believe that this demands a highly interactive environment in which queries may be quickly created and refined, and the goal of this paper is to describe a system to support data exploration.

There have been several approaches to describe the process of exploring data [12, 18, 19, 26, 38]. A common thread between these is that it is a repeated cycle of dependent phases, including discovery, inquiry, exploration, refinement, summarization and visualization, collaboration, and inference. A complete characterization of the data exploration process is beyond the scope of this paper, but at the very least it entails:

Data discovery The user must be able to find out what kinds of data exist, and what kinds of analysis can be performed on the data. As organizations continue to gather more and more data, this problem is increasing in importance [16].

Data inquiry The user should be able to examine different slices of data to, and hone their inquiry as they inspect more and more data. This process generally produces a sequence of queries on the underlying data sets.

Visualization Just being able to select a slice of data is not enough. Once the data has been produced and transformed, there is also a need to choose a suitable visualization.

Summarization SQL includes various aggregation techniques (e.g., count and sum), but as the volume of data increases, visualizations and understanding may be overwhelmed by the sheer volume of data. For this reason we may often need to employ other summarization techniques. Examples include clustering, bucketing for histograms, linear regression, forecasting, dimensionality reduction, or factor analysis to concentrate on the most important dimensions.

Collaboration Data analysis should be recognized as a collaborative activity, where the results of a user may be presented to others for the purpose of further analysis and the formulation of hypotheses for confirmatory data analysis [3] by an independent process.

Access to data

The problem of designing convenient software tools for the analysis of data is as old as the field of data curation itself. Soon after Codd published his seminal work describing the relational database, the SQL language [7] was proposed. One of the motivations for the original SQL language was to address "an increasing need to bring the non-professional user into effective communication with a formatted data base". They went on to say:

> "There are some users whose interaction with a computer is so infrequent or unstructured that the user is unwilling to learn a query language. For these users, natural language or menu selection (3,4) seem to be the most viable alternatives."

In the same year, Codd [9] also argued that

> "If we are to satisfy the needs of casual users of data bases, we must break through the barriers that presently prevent these users from freely employing their native languages (e.g., English) to specify what they want."

A motivating example

To illustrate some of the problems in data exploration, consider the example from [6] that was presented to extoll the virtues of SQL. There the authors observed that the question "What is the lowest price for bolts?" may be easily translated into the program:

```
SELECT MIN(PRICE)
FROM PRICES
WHERE PARTNO IN
(SELECT PARTNO FROM PARTS
 WHERE NAME = 'BOLT');
```

While this is indeed a relatively simple program, and it should be possible for many people to master the skills required to create this, there are a number of underlying problems that still remain:

- How does the user know that this information is even available in a database, and how do they find the specific database? Those of us who are familiar with SQL will recognize that the user would likely first issue a `SHOW DATABASES` and `SHOW TABLES` command to see what tables exist that might contain their data. This is a crude first step in data discovery.

- How does the user know that the table is called `PRICES` and that it contains references to another table called `PARTS` where the word `'BOLT'` appears? In other words, why should the user have to know anything about the underlying relational schema in order to answer their question? Ordinarily in SQL, the user would have to issue a `DESCRIBE PRICES` command to inspect the schema, followed by a `DESCRIBE PARTS`, probably followed by a `SELECT * from PARTS LIMIT 10` to inspect what kind of data they would see in the `PARTS` table. Only after this sequence of explorations would they have enough knowledge and confidence to construct the query for their answer.

- Once the user has found the lowest price, how do they perform a subsequent analysis, such as estimating the volatility for the price of bolts, or examining the different prices for different size bolts, or to find out how rapidly their stock of bolts is depleted?

- If the user is interested in not just the cheapest price, but also the distribution of prices or the trend for price over time, how can they easily view the distribution in some meaningful way and not inspect numbers from a tabular answer?

Many of these details of dealing with SQL are extraneous to the goal of the user, and require too much understanding of the physical layout of data. These problems describe quite well why a natural language approach is so promising, and explain why it is important that input should be in the user's vocabulary.

Even though the natural language approach was hoped to eventually address the accessibility problem, natural language interfaces for accessing data have had relatively low adoption, perhaps because they run the risk of sacrificing precision. The goal of this paper is to describe how a natural language interface that may be integrated into a multi-modal tool for data exploration, in such a way that users are protected against loss of precision and may freely transition between different modes of interaction.

Due to the iterative nature of data exploration, we believe that conversation is a important requirement for a successful data analysis system. Our experience has also shown that the conversation needs to flow in both directions, so that the user may reinforce their understanding of how the system interprets their questions and commands. While humans may often speak imprecisely to machines, machines should never speak imprecisely to humans.

RELATED WORK

This problem is long standing, and the subject of data exploration has been the focus for a great deal of prior work from different points of view. Starting with the early work of Codd, there has been a considerable amount of work in the field of natural language interfaces for databases, with many prototype systems being deployed using a variety of algorithmic approaches. See [1] for a survey through 1995. Much of our work on natural language parsing follows the approach described in [22, 27, 28].

We are also not the first to have built a multi-modal system that combines a natural language interface to data with a visualization framework. Systems having some of these characteristics have been described in [10, 13, 15, 33]. The field of data visualization is also quite active and important for data exploration [20, 13, 33, 10, 35, 11]. While we incorporate visualization into our system and we recognize it as a crucial component in data exploration, it is not a major focus of our work. There is however a stronger overlap with recent work on understanding the role of interaction in visualization systems [38] as well as the role of *conversation* in NLP systems [33].

The problem of data discovery is also an active area of research. Our focus is on identifying tables within a large corpus of tables, but other approaches are mentioned in [5, 16].

A common UI paradigm for visualization in data exploration is that of a "data dashboard" that presents a visual display of the state of data. The term itself is derived from the visual display in an airplane, which presents the current status of various indicators. Dashboards started out as displays to monitor the state of a dataset, using static views on underlying data that may change over time. Over time the user interfaces embraced *exploratation* of data, and provided a user interface that facilitates the creation and navigation through different views on the data, and the ability to change the visualization types [17]. We use dashboards as a starting point for our work, and incorporate NLP and conversation into it.

BRIEF SYSTEM DESCRIPTION

We now give an overview of the various components of the Analyza system; the purpose of this overview is to supply context for the discussion around key design decisions in subsequent sections. A diagram showing the overall system architecture is in Figure 1. Notice that the system has several different types of components for UI, parsing, structured query generation and storage; also, there are two different types of flows depicted. An *online* flow that handles the user's query, and an *offline* flow followed by a curator to improve the system. Both flows are discussed in more detail later in the paper.

Metadata store

This holds three types of metadata. The first type of metadata is the set of intent words, such as "top", "compare", "list" etc., which helps us disambiguate the user's question. The second type of metadata is *schema* information, similar to a SQL database schema, with additional information about the type of the column (e.g. is it a metric, dimension, etc), data formats (e.g. should the number be formatted as a dollar amount), date range defaults, etc. The third type of metadata is a *knowledge base* about entities in the data (e.g., "fr"), i.e., the columns they belong to (e.g., "country"), the lexicon for this entity (e.g., "france"). We also derive an additional lexicon for entities in our knowledge base by joining with a much larger knowledge graph [4, 34].

Parser

This is arguably the most complex part of our system. We use an existing parser for our parsing. The high level architcture for the parser is based on the semantic parsing framework by Liang [22].

Parsing

The role of the parser is to turn the user's input into a intermediate structured representation called the *semantic parse*. The semantic parse is essentially a form with place holders for the *typed* concepts that make up the formal query; the types of these concepts include metrics (numerical columns), dimensions (string valued columns), filters (on dimensions and metrics), sort orders, limits, comparison types, and date ranges. The semantic parse also contains fields that capture the user's intent, requests for types of aggregation and visualizations.

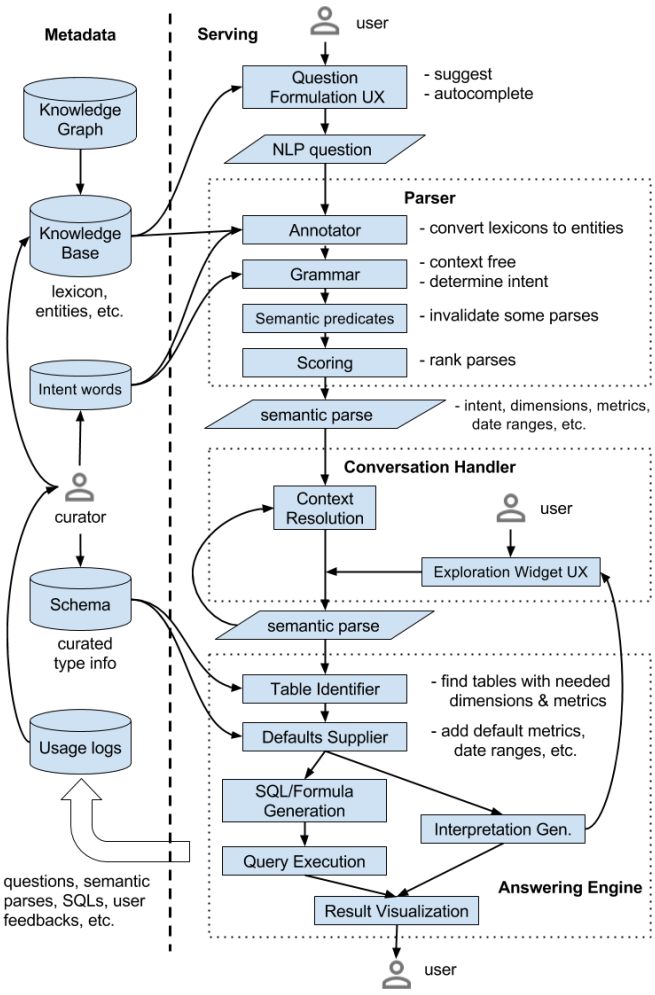

Figure 1. The Analyza system.

Not all the fields of the semantic parse are filled in for every query. See Figure 2 for more details. The components that we implemented are described below.

Annotator

The annotator uses the *knowledge base* to map phrases in the user's query with the entities and intent word types. It uses a combination of string matching, stemming and synonyms to perform this mapping. Phrases can have multiple annotations, and subsequent steps of parsing (such as scoring) perform disambiguation.

Grammar

We use a context-free grammar to parse the annotated query. The grammar rules are written in terms of semantic types. We will show grammar fragments later.

Semantic Predicates

Semantic predicates are used to qualify the validity of a semantic parse. A semantic rule is specified on either a semantic parse or a partial semantic parse, and could specify something like: "no limit can be specified without specifying a dimension", or "average of a metric needs a dimension to average over".

Scoring

While semantic predicates disqualify parses, we also use scoring to produce a soft ranking among parses. A parse with fewer unrecognized words is usually better. A parse that interprets a term "July" as the most recent one is better than an ancient July. Scoring has two manually tuned components; feature generation and feature scores. Feature generation extracts score-worthy aspects of the parse. A simple feature could be the number of unrecognized words in the parse, or the distance of the parsed date from the current time. The feature scores give a linear weighting over features present in the parse.

Table Identifier

We use a simple algorithm to find the smallest table that contains all the columns referenced in the parse, or report that no such table exists. At this stage we may also use access control restrictions to identify which table is appropriate. There is no need for the user to specify the table - this is part of the data discovery phase.

Defaults supplier

Most queries to the system tend to be underspecified. For instance the query may simply be something like "trend for revenue?". We use information in the schema for the table to supply reasonable defaults (e.g. date ranges) that complete the query.

Query Generator

The Query generator turns the semantic parse into an executable query, i.e., either a SQL query or a spreadsheet formula as appropriate. We implement this using a template-based generation system. We then use the appropriate query engine to fetch the data; notice we use existing query engines and did not implement our own.

Interpretation Generator

This module recasts the semantic parse in natural language. It is used to provide feedback to the user about the system's comprehension of the question. Similar to query generation, a template-based system is used, with English templates instead.

User interface

There are various subcomponents in the user interface including ones that perform autocomplete, suggest questions, and select and render the appropriate visualizations.

LESSONS FROM IMPLEMENTATIONS

A unique aspect of our work is the scale at which we have gathered data from interaction by users, and the fact that our components apply to multiple use cases. We have built several different instances of the Analyza system, with slightly different user interfaces that all utilize a natural language interface.

Our initial prototype was built as a web interface and native mobile application that allowed sales people to submit questions and get very quick responses. The underlying data store for this prototype uses SQL. We ran this system for over a year and collected over 70,000 questions during this time

```
SemanticParse {
  vector<Metric> metrics;
  vector<Dimension> dimensions;
  vector<DimensionFilter> filters;
  vector<MetricFilter> metric_filters;
  vector<DateRange> date_ranges;
  # Bounds the number of output rows.
  int limit;
  SortOrder sort_order;
  # Ignored terms from the user question.
  vector<UnusedTerm> unused_terms;
  # Is the query a continuation.
  bool conversation;
  # Indicates a user request for type of
  # visualization; e.g, barchart, trend.
  VisualizationHint hint;
    ...
}
# There are analogous data structures for
# Dimensions, Filters etc.
Metric {
  string name;
  string domain;
  # Dollar value, percentage etc.
  Format format;
  # Type of aggregation (e.g., SUM, AVG).
  Aggregator aggregator;
  # Whether there is a filter bound to the
  # metric, e.g. for the query revenue in
  # france, the filter france is bound to
  # the metric revenue.
  vector<DimensionFilter> bound_filters;
  ...
}
```

Figure 2. Pseudocode for the Semantic Parse. The semantic parse is our central abstraction; it is the output of the parsing process and the input to SQL/formula generation. It is also used to convey context in conversation.

from over 2000 users. The mobile application has proved to be very useful for quick answers to ad hoc questions that arise in business situations.

We also built a prototype that is now incorporated into a spreadsheet product [30]. This implementation bears some similarity to what was described in [15]. In a spreadsheet interface, the data exposed to the user is much smaller than the previous two prototypes, and the schemas are completely unspecified. Moreover, a spreadsheet may itself encapsulate several tables within the grid of cells, and we are often faced with the problem of identifying what range of cells is being referred to by the question. The query engine is also quite different than the previous two examples, since it consists of formulas that are written in the API that is commonly used by creators of spreadsheets. It is worth noting that only 20% of spreadsheet users indicated that they "do programming", which suggests that a large majority of spreadsheet users are consumers rather than producers [31]. It is our hope that this natural language tool will make it significantly easier for the consumers of spreadsheets to easily construct formulas and make more sophisticated use of the data stored in spreadsheets.

KEY DESIGN DECISIONS

In this section we describe the key decisions for parsing, curation and UI.

Why not Machine Learning

There is the potential that machine learning (ML in short) techniques can be applied to parts of the Analyza system, including grammar rules induction, and even the translation from natural language directly to spreadsheet formula/SQL generation [25]. However, we argue that as a first implementation, there are at least two reasons that we cannot rely on ML. First, we did not have sufficient amount of high quality training data to start with, which is needed by most ML systems. Second, ML systems tend to not be very robust, and may not be able to achieve the required level of precision for a data system.

Now that the Analyza system has launched in several forms, we have accumulated real queries from a large set of users, along with the correct formulae or SQL for each query. The first issue on lack of training data is therefore somewhat resolved. It remains a research challenge whether the second or other issues can be resolved toward a ML-based Analyza system.

Semantics over Syntax

What queries should we cover?

Since SQL is the industry standard for data analysis, we could aspire to support all SQL features via natural language. SQL can be used to write very complex, nested queries. For example the following query:

```
SELECT PART FROM PARTS
WHERE PART IN (
  SELECT PART, SUM(SALES) as TOTAL_SALES
  FROM PARTS
  WHERE DATE BETWEEN '2016-01-01' AND '2016-01-31'
  GROUP BY PART ORDER BY TOTAL_SALES DESC LIMIT 20)
  AND INVENTORY < 100;
```

The corresponding natural language question is a tongue-twister: "show me the products that have inventory level below 100 among the 20 top products in january". Furthermore, this question has added ambiguity: is the inventory level from january or current? It is much cleaner to break it up into two questions ("top 20 products in january" and "which of them have inventory level below 100"). We will describe how Analyza supports this form of conversation in a subsequent section.

When we inspected the usage logs from our system, we found that most of the questions could be answered without supporting arbitrary nesting and joins in SQL. We manually inspected a random sample of 200 queries from the sales system, and found only 3% that would require a join or a nested query. Some nested queries may be addressed by followup conversation (e.g., "which of these...").

Natural language is often ambiguous and imprecise. Furthermore, the exposure of users to web search engines in recent years has shaped user expectations for how to phrase their queries. According to [37], only 3.2% of search engine queries are formulated as natural language questions, and about 10.3% were identified to have question intent. Moreover, [2] reports that a majority of the search engine queries are noun phrases. This also reflects our experience with the natural language questions that we observed in Analyza. Here are some of the typical questions we observed from our prototype for sales users.

Example 1 "france revenue and clicks"

Example 2 "top products"

Example 3 "average clicks and revenue by country"

Example 4 "clicks from france and uk"

Example 5 "mobile clicks from france"

It's clear that these queries have very little syntactic structure. From the previously mentioned manual inspection of 200 queries from the sales system, we also observed that 22 could not be parsed, 6 had too many unrecognized terms, resulting in a questionable parse. 2 made no sense at all to us, 4 were refused because the user did not have access to the data, and 4 failed because the user asked for data that did not exist in the database.

Semantics and types

In Analyza, we categorize entities into *metrics*, *dimensions* and *filters*. Metric entities correspond to columns contain numeric values. The numeric values can have different types of aggregations applicable: e.g. *revenue* is additive. Averaging is a more reasonable aggregation for *Age*. It is also possible to have metrics that cannot be added up. Conditions on metrics such as clicks > 100 are *metric filters*. In the SQL query, metric filters are mapped to the *HAVING* clause. Thus clicks > 100 becomes HAVING SUM(clicks) > 100.

Dimensions correspond to columns containing string values or numeric identifiers. Examples include country, state, age group. The dimensions may form a hierarchy. The individual values in a dimension column are *dimension filters*. For example, "france" represents the dimension filter country = france. While executing the formula or SQL, *dimension filters* are translated to equality conditions applied to each row of data.

Using the schema information, we can make sense of the underspecified questions. In Example 5, revenue and clicks are both metrics, hence the aggregation type average applies to both.

In Example 5, both france and uk represent filter on the dimension country, hence we interpret "france and uk" as a logical OR: country = france OR country = uk.

In Example 5, we have two dimension filters: *mobile* (device = mobile) and *france* (country = france). Since these are on different dimensions, we combine them using logical AND.

This information about entity types allows us to infer meaning for the seemingly vague and underspecified questions. We discuss how we obtain the semantic information later in this section.

Using default values

As seen in the examples given above, many of the questions were underspecified or have implicit attributes. In Example 5 ("top products") does not explicitly specify the quantity (i.e.

metric) to sort the products by. We add a pre-configured default metric (e.g. revenue) to the *semantic parse* in this case. Other attributes that need to be supplied include limit (how many products to show), and a relative date range over which to compute the metric. A similar idea was presented in [13]. In a subsequent section on user interface, we describe how the user is informed when these default values are applied.

Domains or entity scopes

In Example 5 given above ("france revenue and clicks"), the metric revenue could represent revenue from many different products. However, the metric clicks is relevant only to online ads, helping us disambiguate revenue.

To achieve this, entities are further grouped into "domains", which represent an abstraction on how entities are related to each other. The overall goal of a domain is to provide scope for where an entity remains semantically consistent. In the narrowest sense, a domain may represent a single spreadsheet where entities are next to each other in a sheet, or a domain may group the columns of a single SQL table. More generally, multiple tables may share a consistent entity across them, e.g. country code or language specification will often be used to represent entities in different tables. The parser insists that all the entities from a semantic parse belong to the same domain.

Grammar

In order to make the grammar work reliably even in absence of syntactic structure, we chose to use a context-free grammar. The grammar uses two kinds of rules. The first kind are floating rules that match the question as sets of entities. Following is an example of floating rules which use Kleene star.

$$\begin{aligned} \text{Entities} &\leftarrow (\text{Entity})* \\ \text{Entity} &\leftarrow \text{Metric} \\ \text{Entity} &\leftarrow \text{Dimension} \\ \text{Entity} &\leftarrow \text{Stopword} \end{aligned}$$

The second kind of grammar rules use phrase-level information to capture dependencies between the words. We give example rules in the figure below. The first one matches "clicks more than 100". The second rule matches "france vs uk". It uses a semantic predicate, validate, that only allows the derivations where both filters apply to the same dimension. Hence phrases like "france vs mobile" won't be accepted.

$$\begin{aligned} \text{MetricFilter} &\leftarrow \text{Metric Comparison Bound} && (1) \\ \text{Comparison} &\leftarrow \text{"more than"} \\ \text{Bound} &\leftarrow \text{Number} \\ \text{SliceComparison} &\leftarrow \text{validate(Filter Against Filter)} && (2) \\ \text{Against} &\leftarrow \text{"vs"} \end{aligned}$$

The previous work in this area has used alternative approaches based on syntactic parsing. [21] used constituency-trees while [13, 27] have used dependency trees in their parsing.

Portability

As mentioned previously, the Annotator maps terms to typed entities. The rest of the system, including grammar, semantic predicates and query generator work on the semantic types. Furthermore, the *semantic parse* corresponds to an abstract data specification that can be translated via a template-based system to queries for the underlying query engine.

These two things allowed us to use the same implementation of Analyza for two very different applications, one with relational databases and one with spreadsheets as underlying data storage.

In summary, when we observed the usage of Analyza, our key takeaways were:

- We rely heavily on semantics of data to make up for lack of syntax.
- Supporting all of SQL via natural language is neither necessary nor sufficient.

Curation Process

Next, we describe the process of populating the *Metadata store* described previously.

The *Metadata store* holds two types of metadata that need curation. The first is schema information, i.e. which columns are metrics, which are dimensions, dates etc. The second type of metadata is a *knowledge base* about entities in the data., i.e., the columns they belong to, and the lexicon for this entity. As we describe below, we use a combination of automated and manual process of curating this data.

Inferring semantics from structured data

In most cases, the semantics of entities are inferred from the structure exposed by the underlying data storage system. The dimensions and metrics are columns in the tables and the values in dimension columns are collected as dimension filters.

For databases, we are able to re-use the work put in by the database adminstrator in creating the table schema. When creating the table schema, the columns represent coherent types, e.g. all the values in country column will be countries. Furthermore, the columns have data types. This type information from table schema allows us to identify columns as metrics, dimensions and date columns.

For spreadsheets, we found that the terminology used to name columns in spreadsheets is more amenable to direct usage in an NLP system. Additionally, spreadsheets have additional information such as visual hints of headers, formatting of numerical values, currencies, dates, and cells automatically filled using formulae. This allows us to infer semantics about the columns, e.g. which aggregations are applicable to a metric. While the schemas are wide open, the fact that spreadsheets are often shared between individuals means that they are often populated with meaningful column headings and visual markup that indicates the structure of the underlying data set.

Manual curation

The column names typically used in databases are often chosen to satisfy goals other than understandability of the data, and the closest analogy for how these names are chosen is probably found in the goals by which programmers select variable names. The goal is often toward maintaining

short variable names that are easily distinguished from others. There have been multiple studies [14] that support the hypothesis that programmers and database engineers often use non-linguistic constructions. These include use of compound names like `productname`, the use of snake case like `product_name`, the elimination of vowels to produce things like `cntr`, and the use of terse acronyms like `CTR`. Simple techniques such as word segmentation allow us to build a lexicon for the columns of a table, though the process may sometimes result in awkward or unnatural utterances, because programmers and users may use different language. In many cases, the inferred semantics can be further enriched by a manual curation step.

We needed to augment the lexicon for entities in our *knowledge base*. For instance, even if there is a table called `customers` in which there is a field called `name`, we would still need further context to know whether customer names are company names or human names. If a database table contains a column named `PRICE`, we would automatically recognize that the utterance "price" from a user may be mapped to this column, but in some circumstances there may be alternative synonyms like `cost`, `charge`, `amount`, `fare`, etc.

The curation also augments the schema information. This include whether a metric is an additive or ratio metric, whether "best" means bigger or smaller values of the metric, or whether a metric represents a currency. The ability to add manually curated information depends on the scale of the problem in terms of number of data sets. In cases where the number of potential users is large (e.g., our sales force use case), there is considerable value from having someone manually add utterances. One approach to this would be to use crowdsourcing, but as the number of users grows, the number of desirable views on data will also increase so any effort made to tailor static views is probably better spent on refining the lexicon. For the implementation of Analyza on spreadsheets, we had to forego the manual part of the curation process.

Ongoing curation

We periodically reviewed and summarized the logs from interactions of sales users with our first prototype. We used these to identify classes of questions that should have been answered, and alternate lexicon to refer to entities within the system. This involved automatically finding phrases in the questions that were not mapped to any entity and clustering all the questions based on these phrases.

This required designing our parser in a way that accounts for each term in the input question. It could either be mapped to a known entity from the schema or it'd be identified as an *unused term*. Keeping track of every term in the question was mentioned in [28]. In a subsequent section, we describe the use of *unused terms* in the user interface.

From inspecting the query logs, we noticed many questions using the construction "ratio of X and Y". This prompted us to support various ways of combining metrics in Analyza. We also found that users asked for dimensions that didn't exist in the data set (e.g. gender). We added a feature to the system to detect questions including such known unknowns that the system can't answer. A helpful error message was displayed in these cases.

On the sales database we went through two rounds of iteration in which we examined a sample of questions to determine precision, as well as an ordered list of the most common unrecognized terms. The first curation round raised the precision from 75% to 81%, and the second curation round raised the precision from 81% to 90%. Upon completion of these two rounds, we observed that queries had an average of 6.3 terms, and 2.7% of terms ended up being unused in the parse. We suspended manual curation at this point, because there is a diminishing return from trying to manually track down unused terms from the long tail, and we should always expect such terms to appear.

Inferring Semantics from Knowledge Graph

We use additional heuristics to derive information from columns and expand the lexicon by using a much larger proprietary knowledge graph [34, 4]. This allows us to perform two kinds of inference from the *content* of the database.

1. When we see a column whose name is LOC that contains strings such as "Chicago", "New York", "Springfield", and "Sunnyvale", we can use the knowledge graph to identify that these strings all belong to a collection of entity names called `/location/citytown` that has various utterances used to describe it, including "city", "town", "locality", "village", "municipality", etc. This allows us to infer synonyms for the name of columns.

2. Once a column has been associated with a collection of entities from the knowledge graph, we may further infer that individual entities may have alternative utterances. Thus in a database of US election campaign contribution data, an entry of "Rafael Edward Cruz" may be recognized by the more common name of "Ted Cruz". A column containing entries such as AMZN, MSFT, and AAPL would be recognized as stock ticker symbols for publicly held companies, and allows us to recognize them by the utterances "Amazon", "Microsoft", or "Apple".

The efficacy of these techniques is strongly dependent upon the scope of the knowledge graph and the nature of the database. Our knowledge graph is human-vetted, so it is of high precision but modest coverage. In the case of the sales data set (which contains countries, fairly large companies, etc) it was able to detect entities for approximately 50% of the values in the tables. For another data set from the US Federal Election Commission, it only identified 9% of the entities. This data set contains many less well-known entities such as individual contributors, their companies, etc. The algorithm was tuned to have very high precision (about 98%) at the cost of lower recall, but the recall still varies quite a bit.

Approximate matches

We increase the value of the curated lexicon in our knowledge base by using approximate matches. We used multiple ways of establishing semantic similarity between a question term and the lexicon. Two terms are similar if they have low edit-distance, or if they have the same stem [29], or if they are

synonyms. Similar techniques for semantic similarity were employed in [13]. It should be noted that spell correction should be done against terms of the vocabulary from the lexicon, as the statistics will be different from a generic corpus of documents.

User interface design

The goal of a user interface for an data exploration system is to make it easy to perform the iterative process that was described in the introduction. We propose using a a multi-modal interface that combines natural language with a dashboard-style method of specifying a data slice. A key component of this design involves a two-way *conversation* between the user and the system. This has two main purposes. First, complex queries are most naturally formed through a conversation, and second, we need to confirm to the user how their input was understood by the system. The latter concept is related to the theory of grounding [8] in communication, where two participants come to a common understanding of what has been said.

Framing the mental model for the user

Exploratory dashboards have proved to be quite effective, but they suffer from one major weakness, namely as the complexity of the data set(s) increases and the number of dimensions and metrics grows, it becomes increasingly difficult to navigate through the space of possible options. It is not uncommon to see database tables that contain dozens or even thousands of individual fields in a record, and this is infeasible to show with widgets, menus, or even autocomplete. The curse of dimensionality is as much a problem in user interfaces as it is in algorithms. For mobile user interfaces this problem is even more acute.

The NLP interface makes it easy for the user to specify their data selection, but also suffers from the problem that a novice user will be unaware of the grammar or vocabulary that the system supports, and must therefore be helped to learn these. This problem is similar to the process by which humans learn a second language, where the user needs to learn both the grammar of recognized questions as well as the vocabulary that corresponds to their known concepts. The user challenge is less severe than learning a second language, because they need only learn which subset of their existing grammar and vocabulary will be recognized.

In the study of how humans learn a second language, a great deal of research has been concentrated on the feedback given, both negative and positive. The two primary concepts that have evolved in this discipline are *recast* and *noticing*. The goal of recast is to echo back what the learner said, but with corrections to the sentence structure. This can be seen as either negative or positive reinforcement. According to Schmidt's noticing hypothesis [32], users will only realize that they are making mistakes or asking correct questions when they become consciously aware of some stimulus in their environment. Long [23] has suggested that in order for recasts to be useful, the learner must recognize the negative feedback contained within, as opposed to just recognizing it

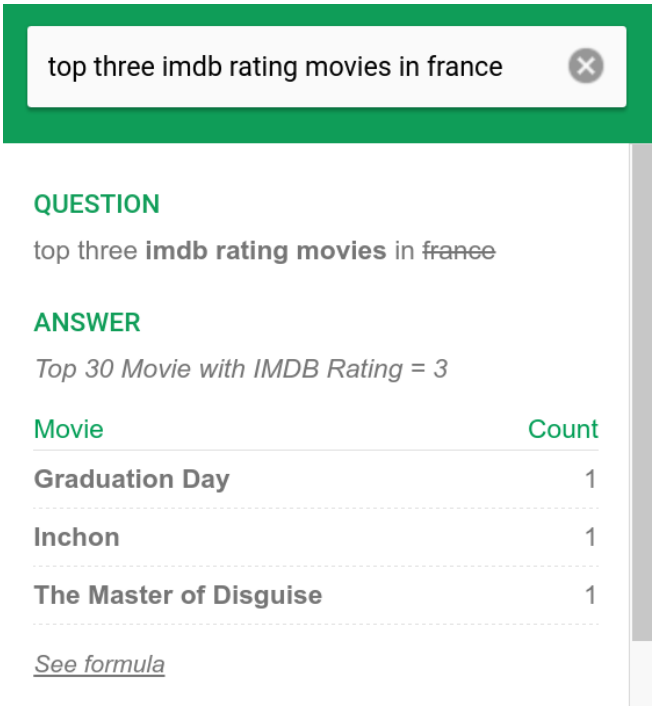

Figure 3. An answer card in the spreadsheet user interface. The user entered a question asking for movies in France, but the data does not contain this information so the display of the original question has this with strikethrough. The recast of the original question shows this by omission, but the strikethrough is used to enhance the user's ability to notice the omission.

as a rephrasing of the original question. We believe that simply restating the question as it is understood by the system is not enough when there are terms that are unused in the user's original question. In this case, a mere restatement of the question omitting reference to the unused phrases places too high a cognitive burden on the user to notice the omission of terms that they originally specified. For this reason, we advocate using both explicit positive feedback and well as explicit negative feedback about unused terms. The design we have chosen for this is to repeat the original user's question with strikethrough to highlight the terms that were ignored, along with a recast of the system's understanding of the original question in a canonical format that is easy to understand (see Figure 3). We omit strikethrough on relatively unimportant terms such as conjunctions.

The recast used in our system has the additional function of showing how the system canonicalizes utterances by the user (e.g, replacing "Click through rate" with CTR). This provides hints on synonyms and shortcuts the user might benefit from. It also allows us to display supplied defaults in the parse process. In the case of spreadsheets, we also highlight the regions of the spreadsheet that are referenced by the question in order to reinforce which parts the recast refers to.

It should perhaps be noted that the feedback mechanism also bears a close resemblance to the use of *readback* in communication between pilots and air traffic control [24], where precision is imperative. The basic principle is to have the pilot and

controller repeat information back to each other to confirm the accuracy of their communication.

Suggestions

We also lean heavily on the common UI paradigm of interactive suggestions as the user types or speaks to the system. We use two kinds of suggestions, namely entities from the data to inform the user about vocabulary, as well as full questions that inform the user about the grammar that is recognized by the system. In Figure 4 we show an implementation of this, in which we display entities from the data that may be concatenated to the user's question, as well as full questions (in this case from the user's history). Full questions may also be constructed from templates generated from the grammar and the data.

Speed of interaction

In order to scale this approach to large data sets, it is important that the NLP understanding part be decoupled from the data fetch to present an answer. Our parser typically takes only 100ms to completely resolve a question into a query, so the user may receive immediate feedback on how the system understands their question and is given a chance to review this while the system fetches the data resulting from their query. If the recast represents an incorrect interpretation, the user is then given a chance to rephrase their request before the system fetches the data.

Dealing with ambiguity

Due to the vague nature of natural langauge, it is sometimes impossible for the system to recognize the user intent. We may either have too little confidence in the parse, or we may be unable to distinguish between two semantically different parses. In the case where multiple parses cannot be resolved, we present the recast of these to the user and ask them to choose one. In case there is insufficient confidence in a parse, we may present alternatives to the user that have strong overlap with the user's original question, or simply ask them to rephrase their question. The case of ambiguity is fairly common, and other approaches are described in [13].

There are a number of other possible failure modes that need to be communicated to the user, including when the question is recognized but the user does not have access to the data, or the query engine is incapable of generating the requested data set. For example, this could happen if the date range is not present in the data, or the user requests hourly aggregation but the data only contains aggregation by day.

Role of conversation

Once a user has asked a question like *what is the trend for revenue in France in q1?* and seen an answer, they may ask a followup question that reuses the state from their previous question. Thus a subsequent question like *what about Germany?* or *how does it compare to last year?* should be recognized within the context of the answer just provided to the user. Such "elliptical" sentences tend to be common in discourse [1, section 4.6].

The system supports substitution of a single element of the semantic parse (e.g., metric or dimension or daterange), but the user may also continue the conversation from the results

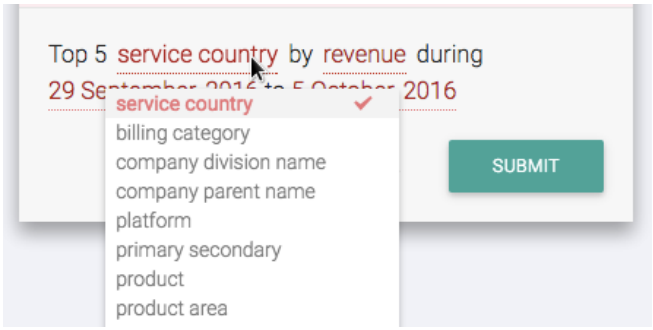

Figure 5. The recast of a question is used to create an additional user widget that facilitates exploration of similar questions.

that were shown in the answer. Thus the context for a conversation consists of both the previous question but also the data from the previous query. As an example, after a question like *top US cities by population*, the user can ask a question like *which of these have a female mayor?*. The result would be the subset of cities that have a female mayor, ordered by population. Note that the followup question need not even be from the same table.

Modes of conversation

While a user may engage in natural language conversation with the system, this places a burden on the user to construct followup questions. We also provide an alternate mode of interaction by creating click targets in the recast to show some of the possible substitutions of metrics, dimensions, filters, and date ranges (see Figure 5). This is similar to the ambiguity widgets of [13], but is used to generalize the question rather than disambiguate the question.

A natural language interface has some natural limitations for very complex queries that require nested queries or subclauses. By providing a conversational interface, we make it much easier for a user to express these nested queries. Moreover, we can support a "sticky" form of context that spans multiple independent queries, through the linguistic command such as "set scope to France". This would then be displayed in the user interface as something that becomes part of the user's question, but may also be dismissed.

Code generation and multi-mode extension

Once a user has submitted a question and received an answer, there are several actions that might be performed other than conversation. A representative card from the mobile application is shown in Figure 6, where there are icons to "share" or "star" (bookmark or save) a given answer. The action of "starring" a card creates a question that becomes part of their customized dashboard. If they revisit the card in the future, the question is reparsed and the answer is recreated using current data. This allows a user to create an on-the-fly dashboard.

We believe that other data exploration paradigms such as SQL and dashboards have their own advantages, and we seek to amplify those rather than replace them. For this reason, we expose the underlying generated SQL and formulas to the

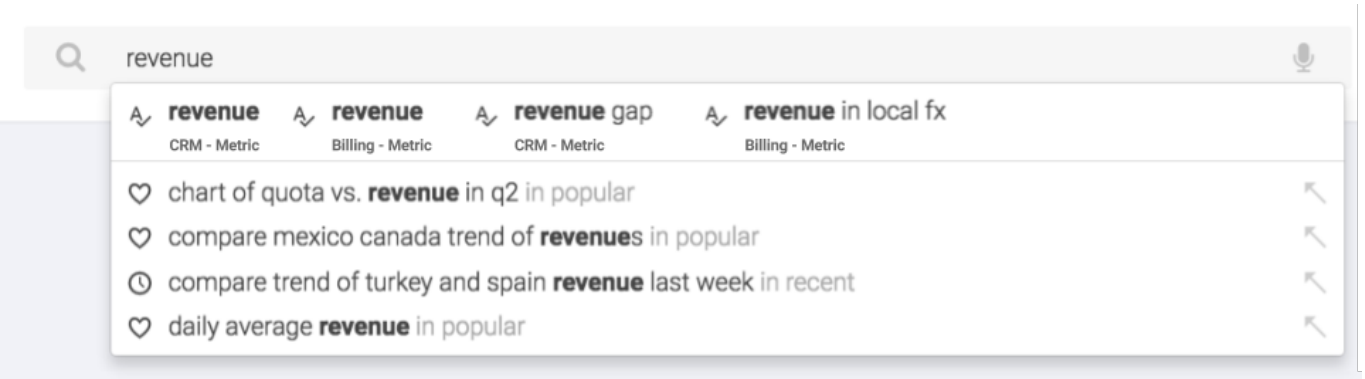

Figure 4. The use of suggestions in a prototype user interface. As the user types, we autocomplete entities from the data that can be used to extend their question, and we show the domain of the entity. We also match against full questions that the user may ask.

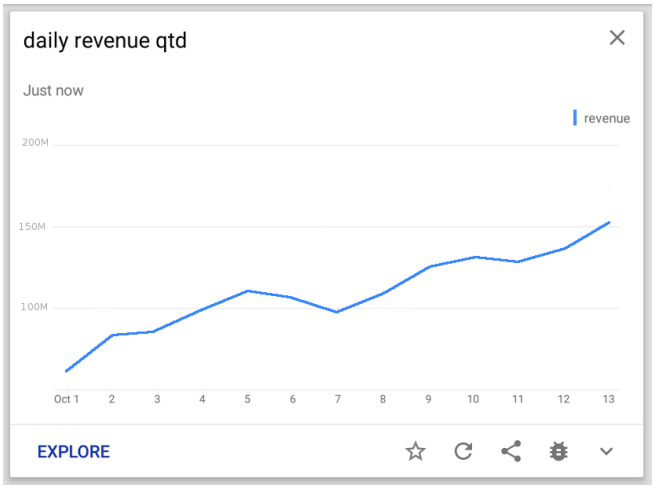

Figure 6. A representative card from the mobile interface, showing icons to perform subsequent actions such as "star" or "share". In this implementation, clicking on "Explore" brings up the widget in Figure 5.

user so that they may use them as starting points for investigations that a user may express more naturally through code rather than natural language.

In the spreadsheet case, a user may use drag and drop to insert the generated formula to a cell in the spreadsheet, and use the existing programming interface to make appropriate modifications. This is similar to [15], where they directly modify the spreadsheet. In the case of a dashboard backed by a relational database, we expose the SQL query as a starting point for refining their analysis when the situation calls for it. We also augment static dashboards by allowing a user to "star" or "bookmark" a query so that they may refer to it later. This provides the capability for a user to create a personalized on-the-fly dashboard.

SUMMARY AND CONCLUSIONS

We wouldn't truly have succeeded in democratizing access to this data until we allow users without coding skills to explore and interact with this data effectively. Analyza is a system that uses natural language processing to do just this. It combines the ease-of-use of a natural language interface with the precision of a structured interface, enabling users to explore data effectively, without resorting to a computer language.

Analyza has been used within two large-scale real world systems – one that enables a large sales force to interact with a revenue/inventory database, and another which provides natural language abilities to a spreadsheet product. Both user bases have a large fraction of users who are unfamiliar with coding, but are able to interpret data and make data-driven decisions.

In the process of designing and implementing this system, we identified several fundamental design choices around abstraction, parsing, curation, multimodality, and portability. We believe that several of the ideas could also apply to intelligent personal (software) assistants.

One of the difficulties in evaluating a system of this type lies in the heterogeneous nature of data sets and the varying sophistication of users. There is some danger in attempting to extrapolate from the observed metrics on our system, because there is no such thing as a "generic database" or a "generic user", and observations are dependent on the nature of data and the mental model of data that is held by the user. We have applied our system to multiple user bases as well as data sets spanning at least least seven orders of magnitude in size. This leaves us hopeful that our observations have at least some generality.

We envision different lines of possible future work; to add the ability to build simple statistical models, to add a data-driven alert functionality, and to build a conversational feature that suggests followup questions, taking into account prior questions of the user.

ACKNOWLEDGEMENTS

The process of turning a research project into a product involves the work of many individuals, and this process produces a lot of insights. We thankfully acknowledge the contributions of the following people: Ajay Nainani, Amir Najmi, Arijit De, Bill MacCartney, Daniel Gundrum, Diane Tang, Garima, Jakob Uszkoreit, Kannan Pashupathy, Koen Dirckx, Mateo Slanina, Nikunj Agrawal, Nishant Redkar, Peter Danenberg, Serge Yaroshenko, Shruti Gupta, Shrikant Shanbhag, Yogita Mehta, and Xiaoqiang Luo.

REFERENCES

1. I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. 1995. Natural language interfaces to databases - an introduction. *Natural Language Engineering* 1(1) (1995), 29–81. **https://arxiv.org/pdf/cmp-lg/9503016.pdf**

2. Cory Barr, Rosie Jones, and Moira Regelson. 2008. The Linguistic Structure of English Web-search Queries. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '08)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 1021–1030. **http://dl.acm.org/citation.cfm?id=1613715.1613848**

3. E. Bier, Card S. K., and J. W. Bodnar. 2007. Entity-based collaboration tools for intelligence analysis. In *IEEE Symposium on Visual Analytics Science and Technology (VAST '07)*, W. Ribarsky and O. Keim (Eds.). IEEE Computer Society Press, Los Alamitos, CA, 99–106. DOI: **http://dx.doi.org/10.1109/VAST.2008.4677362**

4. Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*. ACM, New York, NY, USA, 1247–1250. DOI: **http://dx.doi.org/10.1145/1376616.1376746**

5. Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: Exploring the Power of Tables on the Web. *Proc. VLDB Endow.* 1, 1 (Aug. 2008), 538–549. DOI: **http://dx.doi.org/10.14778/1453856.1453916**

6. Donald D. Chamberlin, Morton M. Astrahan, Michael W. Blasgen, James N. Gray, W. Frank King, Bruce G. Lindsay, Raymond Lorie, James W. Mehl, Thomas G. Price, Franco Putzolu, Patricia Griffiths Selinger, Mario Schkolnick, Donald R. Slutz, Irving L. Traiger, Bradford W. Wade, and Robert A. Yost. 1981. A History and Evaluation of System R. *Commun. ACM* 24, 10 (Oct. 1981), 632–646. DOI: **http://dx.doi.org/10.1145/358769.358784**

7. Donald D. Chamberlin and Raymond F. Boyce. 1974. SEQUEL: A Structured English Query Language. In *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control (SIGFIDET '74)*. ACM, New York, NY, USA, 249–264. DOI: **http://dx.doi.org/10.1145/800296.811515**

8. Herbert H. Clark and Susan E. Brennan. 1991. *Grounding in Communication*. Vol. 13. American Psychological Association, 127–149.

9. E. F. Codd. 1974. Seven Steps to Rendezvous with the Casual User. In *Proc. IFIP TC-2 Working Conference on Data Base Management Systems*. **https://exhibits.stanford.edu/feigenbaum/catalog/cp353fq9623** published in "Data Base Management", ed. J. W. Klimbie and K. I. Koffeman, North-Holland 1974.

10. Kenneth Cox, Rebecca E. Grinter, Stacie L. Hibino, Lalita Jategaonkar Jagadeesan, and David Mantilla. 2001. A Multi-Modal Natural Language Interface to an Information Visualization Environment. *International Journal of Speech Technology* 4, 3 (2001), 297–314. DOI: **http://dx.doi.org/10.1023/A:1011368926479**

11. Usama Fayyad, Georges G. Grinstein, and Andreas Wierse (Eds.). 2002. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

12. Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining*, Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy (Eds.). American Association for Artificial Intelligence, Menlo Park, CA, USA, 1–34. **http://dl.acm.org/citation.cfm?id=257938.257942**

13. Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. 2015. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 489–500. DOI: **http://dx.doi.org/10.1145/2807442.2807478**

14. Latifa Guerrouj, Massimiliano Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2014. An Experimental Investigation on the Effects of Context on Source Code Identifiers Splitting and Expansion. *Empirical Softw. Engg.* 19, 6 (Dec. 2014), 1706–1753. DOI: **http://dx.doi.org/10.1007/s10664-013-9260-1**

15. Sumit Gulwani and Mark Marron. 2014. NLyze: Interactive Programming by Natural Language for Spreadsheet Data Analysis and Manipulation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. ACM, New York, NY, USA, 803–814. DOI: **http://dx.doi.org/10.1145/2588555.2612177**

16. Alon Halevy, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing Google's Datasets. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 795–806. DOI: **http://dx.doi.org/10.1145/2882903.2903730**

17. Patrick Hertzog. 2015. Binary Space Partitioning Layouts To Help Build Better Information Dashboards. In *Proceedings of the 20th International Conference on Intelligent User Interfaces (IUI '15)*. ACM, New York, NY, USA, 138–147. DOI: **http://dx.doi.org/10.1145/2678025.2701383**

18. Andreas Holzinger, Matthias Dehmer, and Igor Jurisica. 2014. Knowledge Discovery and interactive Data Mining in Bioinformatics - State-of-the-Art, future challenges and research directions. *BMC Bioinformatics* 15, S-6 (2014), I1. DOI: **http://dx.doi.org/10.1186/1471-2105-15-S6-I1**

19. S. Jolaoso, R. Burtner, and A. Endert. 2015. Toward a Deeper Understanding of Data Analysis, Sensemaking, and Signature Discovery. In *Human-Computer Interaction–INTERACT, 2015*. Springer, 463–478.

20. Daniel A. Keim. 2002. Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (Jan. 2002), 1–8. DOI: **http://dx.doi.org/10.1109/2945.981847**

21. Fei Li and H. V. Jagadish. 2016. Understanding Natural Language Queries over Relational Databases. *SIGMOD Rec.* 45, 1 (June 2016), 6–13. DOI: **http://dx.doi.org/10.1145/2949741.2949744**

22. Percy Liang. 2016. Learning Executable Semantic Parsers for Natural Language Understanding. *Commun. ACM* 59, 9 (Aug. 2016), 68–76. DOI: **http://dx.doi.org/10.1145/2866568**

23. Michael H. Long. 1996. *The role of the linguistic environment in second language acquisition*. Academic Press.

24. Daniel Morrow, Alfred Lee, and Michelle Rodvold. 1993. Analysis of Problems in Routine Controller-Pilot Communication. *The International Journal of Aviation Psychology* 3, 4 (1993), 285–302. DOI: **http://dx.doi.org/10.1207/s15327108ijap0304_3**

25. Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2015. Neural Programmer: Inducing Latent Programs with Gradient Descent. *CoRR* abs/1511.04834 (2015). **http://arxiv.org/abs/1511.04834**

26. P. Pirolli and S. Card. 2015. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of 2005 International Conference on Intelligence Analysis*. **http://www.phibetaiota.net/wp-content/uploads/2014/12/Sensemaking-Process-Pirolli-and-Card.pdf**

27. Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*. Association for Computational Linguistics, Stroudsburg, PA, USA, Article 141. DOI: **http://dx.doi.org/10.3115/1220355.1220376**

28. Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a Theory of Natural Language Interfaces to Databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces (IUI '03)*. ACM, New York, NY, USA, 149–157. DOI: **http://dx.doi.org/10.1145/604045.604070**

29. Martin F Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.

30. Ritcha Ranjan. 2016. Explore in Docs, Sheets and Slides makes work a breeze — and makes you look good, too. (September 2016). **https://docs.googleblog.com/2016/09/ExploreinDocsSheetsSlides.html**.

31. Christopher Scaffidi, Mary Shaw, and Brad Myers. 2005. Estimating the Numbers of End Users and End User Programmers. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '05)*. IEEE Computer Society, Washington, DC, USA, 207–214. DOI: **http://dx.doi.org/10.1109/VLHCC.2005.34**

32. Richard Schmidt. 1990. The Role of Consciousness in Second Language Learning. *Applied Linguistics* 11 (1990), 129–158. **http://nflrc.hawaii.edu/PDFs/SCHMIDT%20The%20role%20of%20consciousness%20in%20second%20language%20learning.pdf**

33. Vidya Setlur, Sarah E Battersby, Melanie K Tory, Rich Gossweiler, and Angel X Chang. 2016. Eviza: A Natural Language Interface for Visual Analysis. In *29th ACM User Interface Software and Technology Symposium (UIST 2016)*. ACM, New York, NY. DOI: **http://dx.doi.org/10.1145/2984511.2984588**

34. Amit Singhal. 2012. Introducting the Knowledge Graph: things, not strings. Google Blog. (May 2012). **https://blog.google/products/search/introducing-knowledge-graph-things-not/** See also **https://www.google.com/intl/bn/insidesearch/features/search/knowledge.html**.

35. Chris Stolte, Diane Tang, and Pat Hanrahan. 2008. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Databases. *Commun. ACM* 51, 11 (Nov. 2008), 75–84. DOI: **http://dx.doi.org/10.1145/1400214.1400234**

36. John W. Tukey. 1980. We need both exploratory and confirmatory. *The American Statistician* 34 (1980), 23–25. **https://www.jstor.org/stable/2682991?seq=1**

37. Ryen W. White, Matthew Richardson, and Wen-tau Yih. 2015. Questions vs. Queries in Informational Search Tasks. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15 Companion)*. ACM, New York, NY, USA, 135–136. DOI:**http://dx.doi.org/10.1145/2740908.2742769**

38. Ji Soo Yi, Youn ah Kang, John Stasko, and Julie Jacko. 2007. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (Nov. 2007), 1224–1231. DOI: **http://dx.doi.org/10.1109/TVCG.2007.70515**