DataSense: Display Agnostic Data Documentation

Poonam Kumari, Michael Brachmann,
Oliver Kennedy
University at Buffalo, SUNY
{poonamku,mrb24,okennedy}@buffalo.edu

Su Feng, Boris Glavic
Illinois Inst. of Technology
{sfeng14@hawk,bglavic@}.iit.edu

Producing and consuming documentation is an essential step when creating and analyzing data. Good documentation helps a consumer of data to understand the context of the data and its schema, including (i) semantics (e.g., the currency of an account balance), (ii) data collection techniques (e.g., which assay was used to measure blood iron), (iii) limitations or caveats (e.g., that missing values are due to sensor failure), and (iv) assumptions made (e.g., missing geographical locations were inferred through geocoding). Misunderstanding the context of a dataset can lead to statistical errors and mistakes with potentially serious, life-threatening consequences. In short, good data documentation is critical. Unfortunately, extensive documentation can overwhelm users, making it hard to find elements in the documentation that are relevant for the task at hand. We need a better way to interact with documentation than the current state of the art: dozens or even hundreds of pages of word documents.

Modern code development environments (IDEs) present a compelling solution for code: (i) Syntax highlighting is a high-level overview of the document structure, helping users to survey it; and (ii) Mouseover detail views show users the specifics of functions, types, classes, etc. We argue that a similar paradigm is needed for datasets, a paradigm we term **display-agnostic data documentation** (**DAD**). DAD facilitates both *discovery* and *lookup* of context-relevant data documentation by inlining them directly and *interactively* into the data display. This documentation can often be derived from the data or by analyzing provenance. Many such techniques already exist, from fully automated data documentation techniques like data profiling, provenance summarization, to user-provided prose annotations. For example:

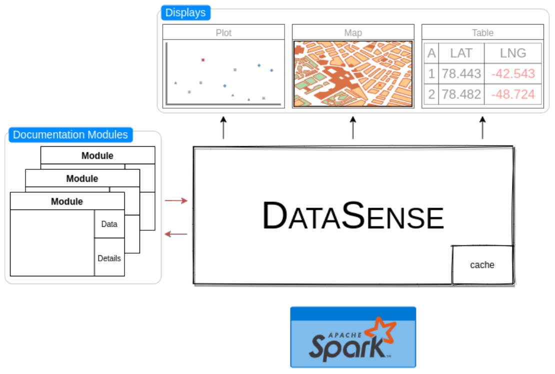
Outliers: Highlighting outliers can assist in finding errors or interesting data points. Prose annotations can help to explain them.

Missing Values: Under SQL's `NULL` Semantics, aggregates silently ignore null values. Highlighting data derived from null values reveals data errors like failed `CAST` operations.

Cell Provenance: Spreadsheet expression cells (e.g., `'=$A22+$B22'`) can help users to interpret the role of the cell, for example when the cell's column name is uninformative. Similar information for database query results (e.g., `AVERAGE(ST_Distance(trip.start, trip.end))`). i.e., schema-level provenance, can be just as helpful.

Annotations: Semistructured documentation is used in programming languages like Python (`__doc__` or PEP484) and Java (Javadoc), and leveraged by IDEs for mouse-over contextual documentation.

Unlike IDEs where users interact with code only through a text editor, a dataset may be displayed and accessed through many modalities: as a table, in a graph, on a map, by writing queries, and more. The need to support multiple modalities poses a challenge for inline documentation, as each new form of data documentation needs to be individually adapted for each display modality. A DAD system needs to separate logic for generating data documentation



A	LAT	LNG
1	78.443	-42.543
2	78.482	-48.724

Figure 1: The DataSense abstraction layer

and displaying data with the documentation inlined. Creating a standard interface between these components makes it possible to implement a documentation generator once, and have it work with a range of display modalities. This leads us to four central goals:

Process Agnostic: A DAD should support many different types of documentation generation processes and formats, including free-form prose, structured properties, provenance, and more.

Declarative: A DAD should generate documentation agnostic to the modality in which it will be displayed. For example, declarative roles (e.g., highlight, detail) can help avoid explicit formatting rules.

Context Sensitive: A DAD should dynamically adapt documentation to what is relevant for a user to avoid overwhelming them.

Unobtrusive: A DAD should (i) facilitate *discovery* of relevant documentation without impeding the user's normal data interactions, while also (ii) making context-relevant documentation accessible.

Automatic tracking: A DAD should ensure that associations between data and documentation are preserved when data is transformed, e.g., during data wrangling. This saves the user from having to manually generate documentation for derived data.

Unlike IDEs however, which manage human-generated (and thus moderately-sized) programs, datasets (which are much larger) require a more scalable and flexible way to associate documentation with data. Figure 1 shows one approach to DAD that we call DataSense: An abstraction layer that links *documentation modules* generates contextual documentation, and relational data *display managers* renders (e.g., as a table, plot, or map) the data and its documentation. We envision DAD allowing users to declaratively document dataset elements. This will require improving the process for inferring documentation from source data, for example by automatically discovering rules for propagating annotations. A prototype of DataSense is implemented as part of the Vizier workflow-notebook (https://vizierdb.info), with support for user- and heuristically-generated prose annotations and profiling metadata, and with support for provenance metadata available soon.