The Journal of Open Source Software

Visions: An Open-Source Library for Semantic Data

Simon Brugman[*1] **and Ian Eaves**[2]

1 Radboud University **2** Independent

DOI: 10.21105/joss.02145

Software

- Review
- Repository
- Archive

Editor: Matthew Sottile

Reviewers:

- @1313e
- @fccoelho

Submitted: 10 February 2020
Published: 13 April 2020

Introduction

Many common data workflows such as loading tabular data from plain text files, data compression, and machine learning data processing rely on *semantically* meaningful representations of the data's type. Most type inference algorithms, including those used by pandas and within the tidyverse employ rule-based heuristics tightly coupled to the *machine* type implementation used by the library. In practice these two representations are distinct. For example, while the set of real numbers between 0 and 1 are stored on the computer disk as float, their *semantics* might instead be a probability.

`Visions` is an expressive, user-configurable framework for capturing the semantic relations between data types forming a development bedrock supporting a range of potential applications.

Statement of Need

While data processing libraries like pandas and the tidyverse are primarily focused on the machine representation of data on disk, there does not exist a systematic solution to expose semantic representations of data. Additionally, most current libraries solve the challenge of type inference through a suite of fixed heuristics. In practice, this restricts developers to the limited set of machine type representations embedded within the libraries implementation.

These heuristics usually consist of rules like "if the series can be cast to integer then do so, otherwise try float," which fail to support the myriad, often hierarchical, semantic representations deployed by users in real-world cases. Virtually every data practitioner has experienced the frustration of reading a column of integers with trailing decimal zeros as float. Despite being obvious to most any human, the heuristics in our tooling usually fail this simple test case. We refer to `visions` documentation for other, similar examples.

Providing a general solution to customized data types is subtly more difficult than it might first appear.

The usual approach is to define a type inference strategy for each machine type where a sequence of potential coercions is applied until one succeeds or the options are exhausted. For example, the string value `'1.0'` might first be cast to integer, which would naively fail, before being cast to float. The heuristic would not provide further inference, as coercion already succeeded.

Defining inference strategies in this manner introduces hard coupling between types, e.g. strings are required to know about integers and floats to attempt casting. Introducing a new type like probability necessitates modifying the string inference strategy to account for the possibility of the new type. Worse still, a user introducing a new semantic type like "probability" which is a subset of float, means modifying float's inference strategy similarly. This rapidly becomes infeasible as adding a second new type means modifying string, float,

*Both authors contributed equally to this paper

and probability. In effect, each new type requires rewriting the heuristics for each type that came before. Further, it introduces strong dependency on the heuristics ordering - if the float test is applied before probability it always infer float. In general, there is no systematic method to determine a "correct" ordering for hierarchically defined generic types in this manner.

Proposed Solution

`Visions` exposes a lightweight and flexible API for dynamically defining semantic data types and the relationships between them. Following the `visions` model of types allows users to build domain-specific groupings of types, referred to as type systems or typesets, using composition as seen below.

In this way, users gain a consistent mechanism for detecting, inferring, and casting data into their chosen type system without being locked into fixed, heuristic-based approaches.

A rich library of pre-defined types is available in the package alongside an expressive API for third-party contributions.

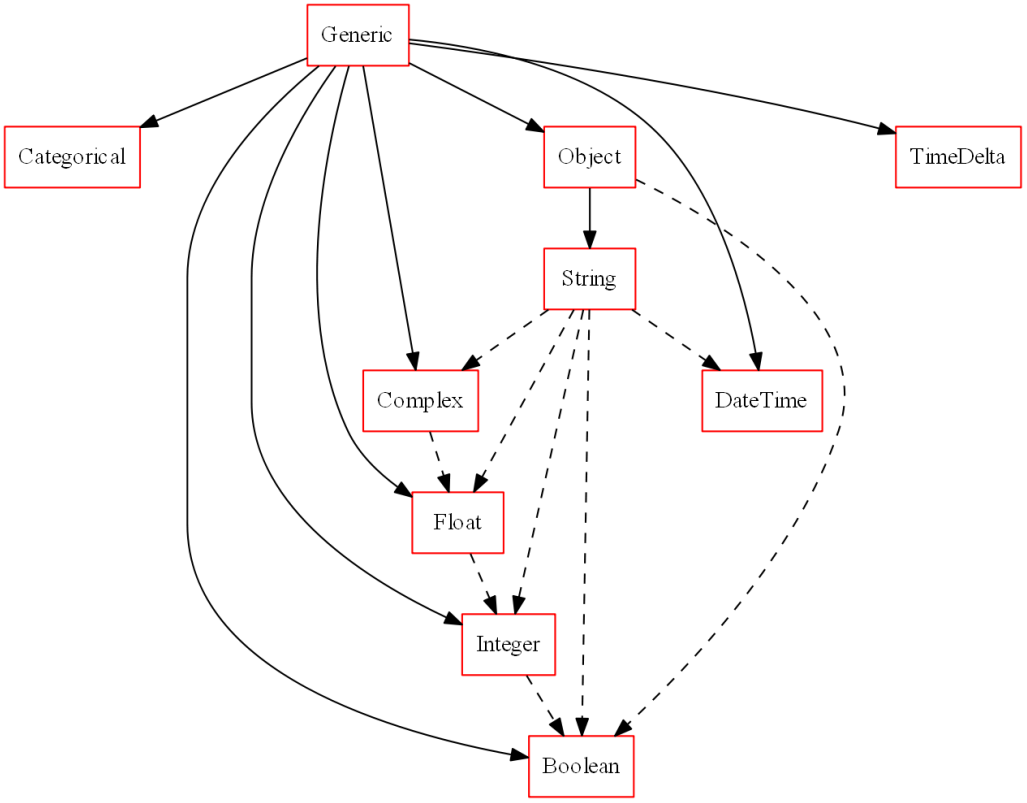

Figure 1: An example of semantic types and their hierarchical relations in the standard typeset. Solid edges denote inference relations where no casting is required, dashed edges denote that casting is required to traverse the edge.

Research purpose and Contributions

The need to capture semantic and machine type distinctions has been a key challenge facing the development of pandas 2.0 where tight coupling has limited type-level expressivity for users and lead to code complexity where type inference heuristics are expanded to support multiple divergent needs pandas2-docs.

`visions` was designed to be used by researchers and practitioners to experiment with more flexible data workflows. The package aims to make defining and relating data types simple while maximizing expressive capability. This enables users to define, relate, modify and share types to solve data tasks and has direct applications to exploratory data analytics (EDA) problem such as those found in (Brugman, 2020) where the appropriate statistics for data of different types varies greatly. It further paves the way for exciting future opportunities to identify and understand semantically useful data abstractions across a variety of potential domains.

`visions` builds on `pandas` (**???**; team, 2020) and `numpy` (Walt, Colbert, & Varoquaux, 2011) for machine type representations. Other work has created specific semantic data types for pandas, namely `cyberpandas` (1, 2, 3) and `geopandas` (Jordahl et al., 2019). Moreover, `networkx` (Hagberg, Schult, & Swart, 2008) is employed to construct type relation graphs.

Potential Applications

Data summarization

Producing highly representative descriptive statistics, also known as data summarization is a critical first step in most exploratory data analysis (EDA). However, machine type representations are rarely sufficiently informative to fully guide initial, exploratory, steps. Consider the 5-star movie rating scale - although numeric, measures like simple means fail to capture the statistical characteristics of the measure. Work on the automated EDA tool (Brugman, 2020) regularly faced this challenge and served as initial motivation for developing `visions`.

Simplifying Data Workflows with `visions`

Decoupling semantic and machine types allows programmers to reuse type-specific logic in many subsequent tasks and hence can be used to simplify data workflows. For example, data validation covering meaningful properties of the data, has to include some form of data summarization, which in turn relies on meaningful types. Similarly, in machine learning the encoding of variables depends on their statistical data type, which are also used for data summarization. Potential workflows that could be simplified range from general data workflows to domain-specific applications including machine learning.

We refer to the application section in the project documentation for a more elaborate list of potential applications.

Acknowledgements

The `visions` package is part of the Dylan Profiler ecosystem. This work was supported by SIDN Fonds under project number 191118.

References

Brugman, S. (2020). pandas-profiling: Exploratory data analysis reports in Python. https://github.com/pandas-profiling/pandas-profiling.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring Network Structure, Dynamics, and Function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15). Pasadena, CA USA.

Jordahl, K., Bossche, J. V. den, Wasserman, J., McBride, J., Fleischmann, M., Gerard, J., Tratner, J., et al. (2019). *Geopandas/geopandas: V0.6.2*. Zenodo. doi:10.5281/zenodo.3545747

team, T. pandas development. (2020). *Pandas-dev/pandas: Pandas 1.0.1*. Zenodo. doi:10.5281/zenodo.3644238

Walt, S. van der, Colbert, S. C., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, *13*(2), 22–30. doi:10.1109/MCSE.2011.37