Detecting Abstract Type Violations Between Data Wrangling & Analysis Code

Lukas Blass (1950003), Eunice Jun (1623594) & Sam Kaufman (1022827)

All empirical fields rely on valid statistical analyses. Unfortunately, conducting valid statistical analyses is challenging. There are two interdependent challenges: preparing the data for analysis ("wrangling") and selecting/executing the correct statistical test to test a hypothesis ("analyzing"). While frequently treated as distinct stages in a developer's workflow, the data produced during the wrangling stage must be designed so that it is compatible with the analysis stage (e.g. how statistical functions anticipate the data to be shaped, typed, etc.). Incompatibilities between stages, however subtle, can lead to defects.

We propose a tool to support early debugging of a particular such defect: abstraction violations where the *abstract types* suggested by one stage are incompatible with usage in the other stage [1]. To scope our project, we will focus on client code of the widely used Python libraries Pandas, Numpy, Scipy, and Statsmodels [2]. While abstract type inference is a technique that already exists, we are unaware of any applications of the technique to the data science domain. [1]

In the data analysis domain, we expect the problem of abstraction violation to be exacerbated by statistical analysis packages' frequent requirement that data be in long format, a common denormalized format. Anecdotally, this translation into and out of this format can be confusing because it eliminates the structure which distinguishes different abstract types of data.

As an example, consider an analyst's task of comparing the prices of cars across two different years. Suppose the dataset contains the following columns: "Car Model", "Price in 1990 (measured in *dollars*)", and "Price Change in 2000" (measured as a *percentage change* from the price in 1990). When the analyst reformats the dataset into long format, they collapse the two price columns into one and add a year column. The reformatted dataset now has the following columns: "Car Model," "Price," and "Year." However, the analyst forgot to change the units of the "Price Change in 2000." As a result, the resulting "Price" column has values of mixed units (implicit). All the values are still numbers, but they have semantically different meanings. Some refer to dollar prices. Others refer to percentages. In other words, data in the same column have incompatible abstract types.

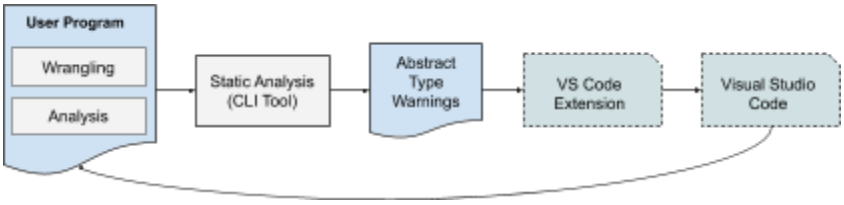
[1] This type system-based technique is orthogonal in approach, if not in goal, from existing work on data debugging, such as the ExceLint and Melford systems [3, 4].

We focus on novice data analysts because we think our tool could benefit novices most. In general, novices make the highest number of errors. Also, anecdotally, we have observed novices repeatedly make abstraction errors.

Our **central research question** is: Can our tool help novice data analysts more quickly and accurately detect and resolve abstraction violations? Given that no similar tools for statistical analysis exist, we **hypothesize** that our tool will help users detect and resolve abstraction violations more quickly and accurately than without the tool.

Approach

Our tool will operate on a single program which can be cleanly divided into two modules: a data wrangling module and a data analysis module. Our tool will statically infer abstract type lattices for each module by analyzing API usage and iteratively unifying abstract types, such as when one DataFrame column is assigned to another (in the wrangling module) or a non-parametric test is performed (in the analysis module). Wherever usage in one module disagrees with the lattice from the *other* module, a potential abstraction violation has occurred, and our tool will raise a warning[2].



We will implement a simple command line interface tool to perform analysis over on-disk files, as well as a Visual Studio Code extension to embed and parse output from that tool in the user interface of the VS Code text editor.

Timeline and Study Design

We plan to evaluate our tool in two phases: a case study and a controlled experiment.

For our case study, we will recruit 3-5 CSE PhD students who regularly conduct statistical analyses. We will ask the students to download and use our tool for 2 weeks. We will develop pre- and post-surveys to gather qualitative data about our tool. Based on the case studies, we will develop two tasks to use in our controlled user study. The tasks will ask people to identify and debug abstraction errors. Both tasks will have the same number of abstraction bugs.

After our case study, we plan to conduct a controlled user study. We will recruit 12 UW undergraduates who have taken CSE 160 (Data Programming), have experience writing Python, and express an interest in data analysis. Our study will have a mixed-design: order of

[2] Depending on the number of unwanted warnings, we may want to extend our static analysis tool to understand some kind of explicit type refinement annotations.

tasks (between subjects) x tool/no tool (within subjects). All participants will do both tasks, but the order of tasks will be counterbalanced. Additionally, six participants will do a task *with* our tool first and then another task *without* our tool. The other six participants will do a task *without* our tool first and then *with* our tool. We will measure participants' **accuracy** in detecting defects and fixing them, the **time** to identify and fix defects, and the **usability** and **usefulness** of our tool (via surveys).

We doubt we can recruit 12 students and conduct all user studies before the final report is due for the course (on June 3). We will try to conduct as many as we can and plan to continue after the final report/course.

Schedule

Case study starts	Week of May 13
Controlled study starts	Week of May 27

References

[1] Guo,P., Perkins, J., McCamant, S., and Ernst, M. 2006. Dynamic inference of abstract types. In Proceedings of the 2006 international symposium on Software testing and analysis (ISSTA '06). ACM, New York, NY, USA, 255-265.

[2] Bobriakov, I. Top 15 Python Libraries for Data Science in 2017. https://medium.com/activewizards-machine-learning-company/top-15-python-libraries-for-data-science-in-in-2017-ab61b4f9b4a7

[3] Barowy, D.W., Berger, E.D. and Zorn, B., 2018. ExceLint: automatically finding spreadsheet formula errors. Proceedings of the ACM on Programming Languages, 2 (OOPSLA), p.148.

[4] Singh, R., Livshits, B. and Zorn, B., 2017. Melford: Using neural networks to find spreadsheet errors. Microsoft Tech. Report.