From Papers to Practice: The openclean Open-Source Data Cleaning Library

Heiko Müller, Sonia Castelo, Munaf Qazi, and Juliana Freire
New York University
{heiko.mueller,s.castelo,munaf,juliana.freire}@nyu.edu

ABSTRACT

Data preparation is still a major bottleneck for many data science projects. Even though many sophisticated algorithms and tools have been proposed in the research literature, it is difficult for practitioners to integrate them into their data wrangling efforts. We present openclean, a open-source Python library for data cleaning and profiling. openclean integrates data profiling and cleaning tools in a single environment that is easy and intuitive to use. We designed openclean to be extensible and make it easy to add new functionality. By doing so, it will not only become easier for users to access state-of-the-art algorithms for their data wrangling efforts, but also allow researchers to integrate their work and evaluate its effectiveness in practice. We envision openclean as a first step to build a community of practitioners and researchers in the field. In our demo, we outline the main components and design decisions in the development of openclean and demonstrate the current functionality of the library on real-world use cases.

PVLDB Reference Format:
Heiko Müller, Sonia Castelo, Munaf Qazi, and Juliana Freire. From Papers to Practice:
The openclean Open-Source Data Cleaning Library. PVLDB, 14(12): 2763 - 2766, 2021.
doi:10.14778/3476311.3476339

1 MOTIVATION

The negative impact of poor data quality on the widespread and profitable use of machine learning [17] makes data cleaning essential in many data science projects. Improving data quality requires data profiling and exploration to gain an understanding of quality issues, and data cleaning to transform the data into a state that is fit for purpose. This process is tedious and costly. A frequently cited survey in 2016 found that data scientists spend 60% of their time on data cleaning and organizing data [16]. In the same survey 57% of the data scientists also stated that they consider data cleaning and organizing data as the least enjoyable task of their job.

Over the years, many tools for profiling, preparing, and cleaning data have been developed, both in academia and industry (see [2, 7, 8] for overviews). These approaches were developed in isolation and in different programming languages with no standardized interfaces. Thus, it is difficult for data scientists to combine existing tools in their data processing pipelines.

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476339

Inspired by the wide adoption of generic machine learning frameworks such as scikit-learn [15], TensorFlow [1], and PyTorch [14], we are currently developing openclean, an open-source Python library for data profiling and data cleaning [11]. Our goals are twofold. First, openclean aims to provide a unified framework for practitioners that brings together open source data profiling and data cleaning tools into an easy-to-use environment. By making existing tools available to a large user-community, and through the integration with the rich Python ecosystem, openclean has the potential to simplify data cleaning tasks. Second, by providing a structured, extensible framework, openclean can serve as a platform with which researchers and developers can integrate their techniques. We hope that by bringing together a community of users, developers, and researchers, we will be in a better position to attack the many challenges in dealing with data quality.

Overview of openclean. The source code for openclean is available on GitHub [11]. We chose Python to implement openclean because of its growing popularity as well as the large number of existing open-source libraries. Figure 1 shows the ecosystem of libraries that openclean currently leverages. openclean is organized in a modular way to account for the large number of approaches and techniques that fall into the areas of data profiling and data cleaning and that we aim to integrate into openclean in the future.

At the center of openclean is the package openclean-core. Within this package we define the relevant APIs for accessing and processing data as well as some basic functionality. The core package depends on well-known Python libraries including pandas [23] and scikit-learn [15] as well as on a set of libraries that were developed in parallel to openclean to address specific needs regarding access to reference data [12], provenance and version management [10], and integration of tools with existing binaries that are not implemented in Python or that require installation of additional software systems [19].

On top of the core library are several extensions that make use of the API's and underlying base packages to provide additional functionality. Current extensions include libraries to discover regular expressions, a Graphical User Interface that is integrated with Jupyter Notebooks, as well as profiling functionality from the Metanome project as one example of how existing tools that are not implemented in Python can easily be integrated into openclean. We are also in the process of integrating other tools like HoloClean [18] and Ditto [9].

In this demo paper, we first introduce the openclean library by giving an overview of the main concepts and the architecture (Section 2). Then, we discuss our demonstration that will show how to use the library based on a few real-world examples and how to add new functionality to the library (Section 3).

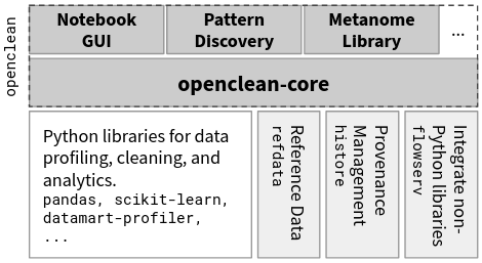


Figure 1: openclean architecture. The core library and extensions depend on standard Python libraries and additional packages that we developed for specific tasks.

2 MAIN FEATURES OF OPENCLEAN

2.1 Data Model and Operators

Datasets and Streams. openclean is primarily intended for tabular datasets, which are represented as data frames: a set of columns with (not necessarily unique) names and a set of rows that contain values for each of the columns.

openclean supports two modes for processing data frames: (i) the data frame can either be loaded completely into main memory (*e.g.,* as a `pandas.DataFrame` [23]), or (ii) it can be streamed from an external storage source, *e.g.,* a comma-separated file. The latter is particularly important for large datasets that cannot fit entirely into main memory. Data streams in openclean are streams of dataset rows. The streaming data processor allows the application of cleaning and profiling operations directly to the rows in the stream, *e.g.,* to filter data of interest that can then be loaded into main memory. Figure 2(b) shows an example in cell #7 using the *NYC Parking Violations* dataset. The dataset contains over 9 million rows and the data file is approx. 380 MB in size. In the example we first select three columns from the dataset and rename column `Registration State` to `State`. We then filter rows that contain a vehicle color and convert the color values to all upper case. Finally, we take a random sample of 1000 rows and return them as a data frame.

Operators. openclean implements two different types of operators: data profiling and data cleaning.

Data Profiling. Profiling operators are used to generate metadata about the data at hand. The generated information can then be used to determine the need for data cleaning or to guide the choice of the cleaning operations. Data profiling operators are either applied on a full data frame, *e.g.,* for the discovery of constraints like functional dependencies, or on sets of values, *e.g.,* on a single column in a data frame for the detection of anomalies (outlier values).

There is a wide variety of profiling operators (see [2] for a classification of profiling tasks). In openclean, we currently provide interfaces and base implementations for the following tasks: computation of basic statistics for dataset columns (*e.g.,* min/max values, entropy, distinct values, etc.), classification of data types, anomaly detection, discovery of regular expressions, and discovery of functional dependencies (FDs) and unique column combinations.

We decided not to standardize the results of profiling operators due to the wide variety of operators and their result types. Instead, a profiling operator either returns a list of values (or objects) or a dictionary. For example, anomaly detection returns a list of values (from a given input list) that were identified as outliers. A profiling operator that computes statistics over the data will return a dictionary with the different computed statistics.

Data Cleaning and Transformation. Data cleaning operators are intended to transform data and they almost exclusively operate on data frames. The following abstract operator types are defined in openclean: transformer, mapper, reducer, and group filter.

A *transformer* maps a given data frame to a new data frame. openclean comes with a set of built-in transformation operations for filtering columns and rows, insertion of new columns and rows, moving and renaming columns, updating of cell values, and for sorting. A *mapper* returns a group of data frames for a given input data frame. Each resulting group is identified by a unique key. The default mapper in openclean is the `groupby` operator that groups rows based on key values that are computed for each row using a given evaluation function (see below). A *reducer* converts a group of data frames into a single data frame. A common example are operators that resolve conflicts when repairing violations of FDs. A *group filter* operates on groups of data frames and transforms one set of groups into a new set of groups.

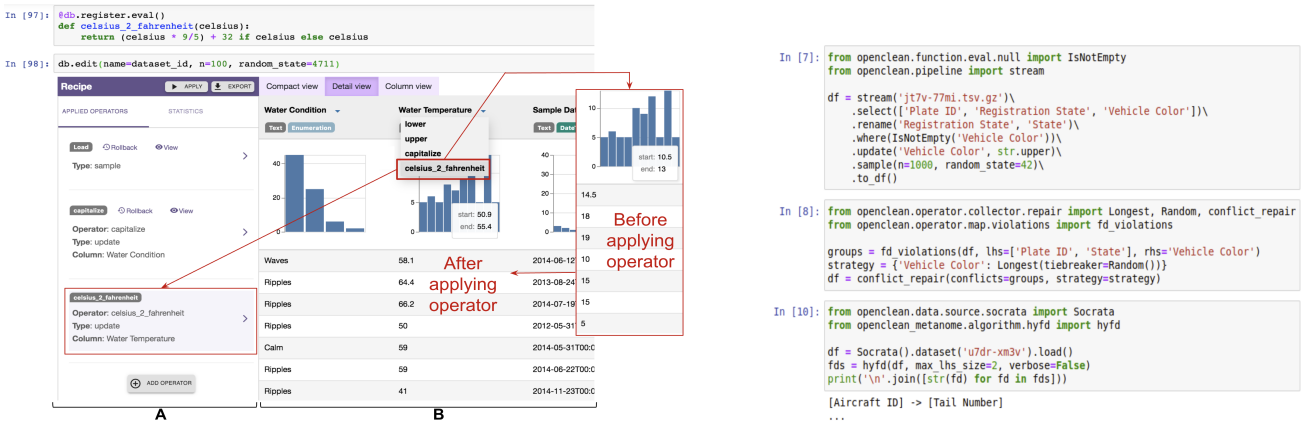
Functions. Well-defined APIs are essential for a framework that aims to be flexible and extensible. In openclean we define APIs for common data wrangling tasks in a modular way that makes it easy for a user to combine them as well as to include new functionality at different levels of granularity. One means to ensure such flexibility is the use of functions as arguments.

In openclean, many cleaning and transformation operators take functions as their arguments. For example, the `filter` and `update` operators shown in Figure 2(b) [#7] take functions `IsNotEmpty` and `str.upper` as arguments to select and manipulate rows. By having functions as operator arguments it becomes easy to change and customize behavior. In the example, we can easily use any (user defined) function with a single argument instead of the standard `str.upper` function to transform the data using the update operator.

Figure 2(b) [#8] shows another example for the power of composability in openclean. Here we use the `fd_violations` operator (a combination of mapper and group filter) to get groups of rows that violate a given functional dependency. We then define a repair strategy using the function `Longest` knowing that violations are caused by abbreviations like BLK and BRW for BLACK and BROWN. Again, it is easy for the user to define their own domain-specific repair strategy by providing a custom data manipulation function for the sets of rows that violate the functional dependency.

2.2 GUI - Integration with Jupyter Notebooks

Data profiling and cleaning are inherently exploratory tasks. In many scenarios the user needs to visualize data and profiling results (statistics) to get a better understanding of data properties and existing quality issues, or may identify a data quality issue by examining the predictions of a machine learning model. Many existing cleaning tools like OpenRefine [5] or Trifacta Data Wrangler [22] come with graphical user interfaces (GUIs) to make it easier for users to explore and clean their data. Instead of relying on a dedicated GUI, openclean can be used in many different environments, including Jupyter Notebooks [6]. Working in a Python or notebook



(a) (b)

Figure 2: (a) The openclean interface in a Jupyter Notebook allows users to inspect the contents of a dataset in the spreadsheet view (B), including column statistics and data types. The recipe view (A) captures the series of operations applied to the data. Users can register user-defined functions and later apply them through the visual interface. (b) Use of openclean for functional dependency violation repair.

environment allows users to more easily integrate data cleaning tasks with their data science pipelines. In addition to the ability to leverage existing libraries, openclean provides a spreadsheet-like GUI shown in Figure 2(a), which enables users to visualize and interact with the data from a Jupyter Notebook.

Data Summary. The spreadsheet view provides easy-to-use visualizations for data analysis and exploration. It displays profiling results for each column, including inferred data types (e.g., in Figure 2, `Water Condition` is detected as a categorical attribute) and statistical information such as mean, standard deviation, and unique values, at different levels of detail (compact, detail, and column views) together with histograms of column values and inferred column types. By default, profiling information is generated using the `datamart-profiler` [21]. openclean also provides an API for users to create and use their own data profilers.

Recipe. The GUI allows users to apply transformation operators on the spreadsheet using a set of pre-defined evaluation functions. The history of applied functions is shown as part of the spreadsheet view, *i.e.,* the *recipe*. openclean provides the ability to extend the set of available functions. Figure 2(a) shows an example of how a user-defined function can be registered using Python decorators (cell [#97]). All registered functions can later be applied in the user interface. openclean supports materialization of registered functions, *e.g.,* in Json files or using a relational database, which makes it possible to re-use the functions in different notebooks or share them among trusted users. Operations in a recipe can also be exported as a new transformer that can be applied to other datasets.

2.3 Reference Data

Reference data is an important tool for data cleaning and openclean supports the use of different reference datasets. For example, the restcountries.eu project [20] provides a curated list of country names for all countries in the world together with their official ISO 3166-1 abbreviation codes. This data can be used to identify invalid values (*e.g.,* misspellings) in a given dataset column with country names. Using openclean's string similarity search we can also find possible replacements for misspelled names as close matches in the curated list. The mappings of country names to different abbreviations can also be used to standardize representations for columns containing a mix of two- and three-letter country codes.

We created the open-source library `refdata` [12] to provide access to reference datasets on the Web. Individual datasets are hosted by data maintainers on the platform of their choice. Information about available dataset is maintained in a central index (*i.e.,* a Json file) hosted on the Web. Users download copies of the datasets for local access.

2.4 Data Provenance Management

Because cleaning is an exploratory task, it often is necessary to undo operations, compare the current state of a dataset to a previous version, or view the history of how the values in a data cell were changed by different cleaning operations. For openclean we developed `histore` [10], a library to maintain the history of a dataset. `histore` is based on techniques that were developed in [4]. `histore` maintains all snapshots of a dataset together with descriptors of the operators that created them. Similar to popular version control systems like `git`, `histore` provides the ability to commit, checkout, and rollback different snapshots of a dataset.

In the spreadsheet view shown in Figure 2(a), for example, the user can click on any of the operations in the recipe to view the dataset snapshot as it was after the execution of the respective operation. At this point the user has the option to rollback all changes that occurred after the selected operation.

2.5 Integration of External Tools

openclean provides mechanisms to integrate and execute existing tools (binaries) from within Python scripts. Some tools are implemented in different programming languages, *e.g.,* Metanome [13] is implemented in Java, while others require installations of additional software systems, *e.g.,* HoloClean [18] requires an installation of PostgreSQL. Our aim is to avoid re-implementing existing tools

while giving users a choice of how to run them in case that they do not have the ability or privileges to install additional software.

To this end, `openclean` makes use of `flowserv`, a library that allows to define wrappers for existing tools and run them from within Python. A typical scenario for running an external program on a given dataset is as follows: (i) write the dataset to a file on disk, (ii) run the binary using the dataset file plus other user-provided parameters from the command line, and (iii) read results (*e.g.,* a modified dataset output file) and convert them into Python objects.

The first and the third step can easily be executed within the same thread that is running the `openclean` script. The second step could be executed using Python's `subprocess` package if the user has the required runtime environment (*e.g.,* Java JRE) and binaries (*e.g.,* Jar file) installed on their machine. As an alternative, external programs can be run using container engines like Docker. In this case, the user needs to have Docker installed locally but could easily run many programs that are implemented in different languages without further installations.

`flowserv` allows developers to define workflows like the one above as a Python script. Users can then configure the system to allow them to choose if individual workflow steps are executed as sub-processes or as Docker containers. In addition, we are currently working on options to run workflow steps that require a large amount of resources on High-performance computing clusters or using commercial cloud service providers.

Figure 2(b) [#10] shows an example from the user perspective for the seamless integration of existing profiling operators from Metanome into `openclean` using the *Aircraft Tail Numbers and Models at SFO* dataset.

3 DEMONSTRATION

We will demonstrate `openclean`'s abilities to profile and clean data using different datasets. All examples shown will be made available as Jupyter Notebooks together with other use cases that are already in the `openclean` GitHub repository and which will be available on Google Colab. This will enable attendees and those that cannot attend in person to experiment with `openclean` on their computers.

In what follows, we describe two use cases we plan to show during the demo. We note that `openclean` includes an adaptor for the Socrata Open Data API [3] which gives us immediate access to thousands of datasets that can be used during the demo.

NYC Parking Violations. Parking Violations Issuance datasets for New York City contain violations issued during a fiscal year. The datasets are available from the Socrata API and contain, among other information, details about parking meters,*e.g.,* their ID and street address, and the vehicle that was involved in the violation.

In our demo we use the dataset for the year 2014 that contains over 9 million rows. We first show how to filter data columns and rows using the streaming operator, *e.g.,* remove rows with missing `Meter ID`. We then identify violations of functional dependencies, *e.g.,* `Meter ID` → `Address`, and show how to define violation repair strategies in `openclean`, *e.g.,* using the most frequent value involved in a violation. We further show how reference data on street types is used to standardize street addresses, *e.g.,* W35 Street vs. West 35 Str. Finally, we demonstrate the use of `openclean-pattern` to discover patterns for column `Meter ID` that are then used to identify outliers that do not meet the expected value format.

Ethiopia Vaccination Data. Ethiopia Vaccine Delivery dataset contains monthly vaccine deliveries to various administrative levels of the Oromia region in Ethiopia over a time period of 2 years.

We demonstrate `openclean`'s extensibility by first computing the equivalent Gregorian dates from Ethiopic calendar values spanning multiple columns by implementing a new reusable operator. We then showcase built-in standardization support by identifying and fixing misspelled 'Woreda' names in the dataset by comparing them with a master vocabulary of official spellings after building a Mapping using `openclean`'s string matching operators. The demo also stresses the importance of having a user with domain knowledge in the loop to resolve conflicts in any standardization and cleaning process.

ACKNOWLEDGMENTS

This work was partially supported by the DARPA D3M program, NSF awards OAC-1640864 and OAC-1841471, and the NYU Moore Sloan Data Science Environment.

REFERENCES

[1] Martín Abadi and et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/
[2] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling Relational Data: A Survey. *The VLDB Journal* 24, 4 (Aug. 2015).
[3] apilayer. 2021. Restcountries Project. http://restcountries.eu.
[4] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang-Chiew Tan. 2004. Archiving Scientific Data. *ACM Trans. Database Syst.* 29, 1 (March 2004).
[5] OpenRefine Developers. 2010. OpenRefine. https://openrefine.org/.
[6] Jupyter development team. 2016. Jupyter Notebooks - a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, Netherlands.
[7] Mazhar Hameed and Felix Naumann. 2020. Data Preparation: A Survey of Commercial Tools. *SIGMOD Rec.* 49, 3 (Dec. 2020).
[8] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning*. Association for Computing Machinery, New York, NY, USA.
[9] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (Sept. 2020).
[10] Heiko Müller. 2018. Data Frame History Store. https://github.com/heikomuller/histore.
[11] openclean Development Team. 2018. openclean - A Data Cleaning Library. https://github.com/VIDA-NYU/openclean.
[12] openclean Development Team. 2021. Reference Data Repository. https://github.com/VIDA-NYU/reference-data-repository.
[13] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Profiling with Metanome. *Proc. VLDB Endow.* 8, 12 (2015).
[14] Adam Paszke and et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc.
[15] F. Pedregosa and et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011).
[16] Gill Press. 2016. Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says. *Forbes* (Mar 2016). https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/
[17] Thomas C Redman. 2018. If Your Data Is Bad, Your Machine Learning Tools Are Useless. *Harvard Business Review* (Apr 2018). https://hbr.org/2018/04/if-your-data-is-bad-your-machine-learning-tools-are-useless
[18] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (2017).
[19] Project Scailfin. 2019. Reproducible and Reusable Data Analysis Workflow Server. https://github.com/scailfin/flowserv-core.
[20] Socrata. 2021. Discovery API. https://socratadiscovery.docs.apiary.io.
[21] Auctus Development Team. 2018. Data profiling library for Datamart. https://pypi.org/project/datamart-profiler/.
[22] Trifacta. [n.d.]. Data Wrangling Software and Tools. https://www.trifacta.com/.
[23] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*.