The Journal of Open Source Software

Cuallee: A Python package for data quality across multiple DataFrame APIs

Herminio Vazquez [1*] **and Virginie Grosboillot** [2*]

1 Independent Researcher, Mexico **2** Swiss Federal Institute of Technology (ETH) * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

In today's world, where vast amounts of data are generated and collected daily, and where data heavily influence business, political, and societal decisions, it's crucial to evaluate the quality of the data used for analysis, decision-making, and reporting. This involves understanding how reliable and trustworthy the data are. To address this need, we've created `cuallee`, a Python package for assessing data quality. `cuallee` is designed to be dataframe-agnostic, offering an intuitive and user-friendly API for describing checks across the most popular dataframe implementations such as PySpark, Pandas, Snowpark, Polars, DuckDB and BigQuery. Currently, `cuallee` offers over 50 checks to help users evaluate the quality of their data.

Statement of need

For data engineers and data scientists, maintaining a consistent workflow involves operating in hybrid environments, where they develop locally before transitioning data pipelines and analysis to cloud-based environments. Whilst working in local environments typically allows them to fit data sets in memory, moving workloads to cloud environments involve operating with full scale data that requires a different computing framework, i.e. distributed computing, parallelization, and horizontal scaling.

This shift in computing frameworks requires the adoption of testing strategies that can accommodate testing activities in both local and remote environments, without the need to rewrite test scenarios or employ different testing approaches for assessing various quality dimensions of the data.

An additional argument is related to the rapid evolution of the data ecosystem. Organizations and data teams are constantly seeking ways to improve, whether through cost-effective solutions or by integrating new capabilities into their data operations. However, this pursuit presents new challenges when migrating workloads from one technology to another. As information technology and data strategies become more resilient against vendor lock-ins, they turn to technologies that enable seamless operation across platforms, avoiding the chaos of fully re-implementing data products. In essence, no data testing strategy needs to be rewritten or reformulated due to platform changes.

A last argument is the need for such a quality tool, is the desire of moving quality procedures to the earliest phases of the data product development life-cycle. Whether in industry or academia, the reduction of time allocated for quality activities is unfortunately like a norm, due to the predominant focus on functional aspects. Enabling a declarative, intuitive and flexible programming interface to data quality, allows teams to embed quality into their development, adopting a mindset of building quality in, as opposed to testing quality out.

The Journal of Open Source Software

Checks

(Binney & Tremaine, 2008) is the reference for the checks

Check	Description	DataType
`is_complete`	Zero `nulls`	*agnostic*
`is_unique`	Zero `duplicates`	*agnostic*
`is_primary_key`	Zero `duplicates`	*agnostic*
`are_complete`	Zero `nulls` on group of columns	*agnostic*
`are_unique`	Composite primary key check	*agnostic*
`is_composite_key`	Zero duplicates on multiple columns	*agnostic*
`is_greater_than`	`col > x`	*numeric*
`is_positive`	`col > 0`	*numeric*
`is_negative`	`col < 0`	*numeric*
`is_greater_or_equal_than`	`col >= x`	*numeric*
`is_less_than`	`col < x`	*numeric*
`is_less_or_equal_than`	`col <= x`	*numeric*
`is_equal_than`	`col == x`	*numeric*
`is_contained_in`	`col in [a, b, c, ...]`	*agnostic*
`is_in`	Alias of `is_contained_in`	*agnostic*
`not_contained_in`	`col not in [a, b, c, ...]`	*agnostic*
`not_in`	Alias of `not_contained_in`	*agnostic*
`is_between`	`a <= col <= b`	*numeric, date*
`has_pattern`	Matching a pattern defined as a `regex`	*string*
`is_legit`	String not null & not empty `^\S$`	*string*
`has_min`	`min(col) == x`	*numeric*
`has_max`	`max(col) == x`	*numeric*
`has_std`	`σ(col) == x`	*numeric*
`has_mean`	`μ(col) == x`	*numeric*
`has_sum`	`Σ(col) == x`	*numeric*
`has_percentile`	`%(col) == x`	*numeric*
`has_cardinality`	`count(distinct(col)) == x`	*agnostic*
`has_max_by`	A utilitary predicate for `max(col_a) == x for max(col_b)`	*agnostic*
`has_min_by`	A utilitary predicate for `min(col_a) == x for min(col_b)`	*agnostic*
`has_correlation`	Finds correlation between `0..1` on `corr(col_a, col_b)`	*numeric*
`has_entropy`	Calculates the entropy of a column `entropy(col) == x` for classification problems	*numeric*
`is_inside_interquartile_range`	Verifies column values reside inside limits of interquartile range `Q1 <= col <= Q3` used on anomalies.	*numeric*
`is_in_millions`	`col >= 1e6`	*numeric*
`is_in_billions`	`col >= 1e9`	*numeric*
`is_t_minus_1`	For date fields confirms 1 day ago `t-1`	*date*
`is_t_minus_2`	For date fields confirms 2 days ago `t-2`	*date*
`is_t_minus_3`	For date fields confirms 3 days ago `t-3`	*date*
`is_t_minus_n`	For date fields confirms n days ago `t-n`	*date*
`is_today`	For date fields confirms day is current date `t-0`	*date*
`is_yesterday`	For date fields confirms 1 day ago `t-1`	*date*

Check	Description	DataType
`is_on_weekday`	For date fields confirms day is between `Mon-Fri`	*date*
`is_on_weekend`	For date fields confirms day is between `Sat-Sun`	*date*
`is_on_monday`	For date fields confirms day is `Mon`	*date*
`is_on_tuesday`	For date fields confirms day is `Tue`	*date*
`is_on_wednesday`	For date fields confirms day is `Wed`	*date*
`is_on_thursday`	For date fields confirms day is `Thu`	*date*
`is_on_friday`	For date fields confirms day is `Fri`	*date*
`is_on_saturday`	For date fields confirms day is `Sat`	*date*
`is_on_sunday`	For date fields confirms day is `Sun`	*date*
`is_on_schedule`	For date fields confirms time windows i.e. `9:00 - 17:00`	*timestamp*
`is_daily`	Can verify daily continuity on date fields by default. `[2,3,4,5,6]` which represents `Mon-Fri` in PySpark. However new schedules can be used for custom date continuity	*date*
`has_workflow`	Adjacency matrix validation on `3-column` graph, based on `group`, `event`, `order` columns.	*agnostic*
`satisfies`	An open `SQL expression` builder to construct custom checks	*agnostic*
`validate`	The ultimate transformation of a check with a `dataframe` input for validation	*agnostic*

Controls

This are the controls

References

Binney, J., & Tremaine, S. (2008). *Galactic Dynamics: Second Edition*. Princeton University Press. http://adsabs.harvard.edu/abs/2008gady.book.....B