Pharo DataFrame: Past, Present, and Future

Larisa Safina[1,*,†], Oleksandr Zaitsev[2,*,†], Cyril Ferlicot-Delbecque[1] and Papa Ibrahima Sow[3]

[1]*Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, Lille, France*
[2]*UMR SENS, Cirad, Montpellier, France*
[3]*Ecole Supérieur Polytechnique (ESP), UMMISCO, Dakar, Senegal*

Abstract
DataFrame is a tabular data structure for data analysis. It is a two-dimensional table (similar to a spreadsheet) with an extensive API for querying and manipulating the data. Data frames are available in many programming languages (e.g., pandas in Python or data.frame in R), they are the go-to tools for data scientists and machine learning practitioners. Pharo DataFrame was first released in 2017. Since then, the library underwent many changes and improvements. In this paper, we present the Pharo DataFrame library, show examples of its usage, and compare its API to that of pandas. We overview the changes that have been made since DataFrame v1.0, discuss the limitations of the current implementation, and present the roadmap for future.

Keywords
Pharo, DataFrame, data analysis, data structure

1. Introduction

We live in a data-intensive world, where data in itself constitutes a form of wealth, its amounts are growing exponentially, as well as the level of integration of data-powered tools in our everyday lives. Acquiring data, extracting knowledge from it, and acting based on that knowledge became one of the key activities in modern industries. That is why, modern programming languages and environments are expected to provide tools for data analysis, data visualization, machine learning, data mining, and business intelligence.

Among such tools are data frames — tabular data structures that provide extensive API for data analysis and manipulation. Available in various programming languages (e.g., pandas in Python, data.frame in R), data frames are the go-to tools for data scientists and machine learning practitioners. The first implementation of DataFrame was introduced into Pharo in 2017 [1] as part of the Google Summer of Code project.[1] During the last six years, DataFrame underwent many modifications and improvements: from adding new features and extending its API to larger architectural changes. In this paper, we present the DataFrame library, overview

IWST 2023: International Workshop on Smalltalk Technologies. Lyon, France; August 29th-31st, 2023
*Corresponding author.
†Both authors contributed equally.
larisa.safina@inria.fr (L. Safina); oleksandr.zaitsev@cirad.fr (O. Zaitsev); cyril@ferlicot.fr (C. Ferlicot-Delbecque); papaibrahimasow@esp.sn (P. I. Sow)
0000-0002-4490-7451 (L. Safina); 0000-0003-0267-2874 (O. Zaitsev)

[1]https://summerofcode.withgoogle.com/archive/2017/organizations/5691803940421632

the major changes in its new version, and discuss the future developments that are envisioned by the DataFrame community. We also contribute the early results of exploring the DataFrame API. We study (1) *the evolution of API* by comparing it across two versions of the library and (2) *the completeness of API* by comparing it to the non-exhaustive list of the most important features of pandas.

The rest of this paper is structured in the following way. In Section 2, we explain what are data frames and why do we need them. In Section 3, we overview the major changes and improvements that have been introduced into the library over the last years of development. Section 4 contains the overview of data frames in other programming language and a non-extensive comparison of DataFrame's API to that of pandas. In Section 5, we discuss the envisioned future developments, and finally, Section 6 concludes this paper. Additionally, in Appendix A, we provide an example of how DataFrame can be used to analyse the gender wage gaps dataset.

2. What are DataFrames and Why We Need Them?

Data can be represented in many ways. Tree data structures (e.g., JSON, STON, XML) are often used to model complex objects that can be composed of instances or collections of other objects. Such data structures are simple and powerful because they can be used to store objects of different complexity, from primitive values to large complex object structures with nested elements. However, they are not optimal for analysing data. The more complex such a tree structure becomes, the more difficult it is to write queries, add or remove features, merge multiple datasets, aggregate and group their values. That is why we often prefer to represent data with a more restrictive tabular data structures [2, 3] that can be characterized by the following features:

1. Tables have rows and columns;
2. Each row has same columns and in the same order;
3. Columns are *homogeneous*, meaning that they store values of the same type (*e.g.,* only strings or only integers) while rows can be heterogeneous (one row can contain values of different data types);
4. Rows are stored in a particular order.

Each row in such table can be seen as an object (observation) and each column as a parameter (feature). For example, to store a data about employees, one may create a table where each employee is represented by a row and each column corresponds to certain property: name (`string`), age (`integer`), salary (`float`), etc. The data types of each column are usually of primitive data types, therefore such tables are not optimal for storing complex objects. However, the same limitation makes them well suited for data analysis.

Although tables of data can be implemented with a combination of standard collections (e.g., a list of dictionaries), the data manipulation of such a composed collection would be cumbersome. This raises a need for a dedicated data structure that represents a table and provides a simple API for accessing and manipulating its rows and columns, querying and analysing the data stored in it. Such data sets are called *data frames*. The example in Appendix A provides a hands-on demonstration of data frames and their applications.

3. Pharo DataFrame: past and present

The first version of DataFrame [1] for Pharo was released in September 2017 as a result of Google Summer of Code. Although that version was fully functional for the basic use cases of data frame, the project was rather immature and had several major problems. Most importantly:

- *Lack of functionality.* The API of Pharo DataFrame was far smaller compared to that of other open-source data frame libraries. For example, it did not provide methods for handligh missing values, data loading from CSV could not be configured, etc. Although all those functionalities could still be achieved through the standard API of Pharo collections, there was a clear need for introducing dedicated methods into the DataFrame. As can be seen in Table 1, the number of methods in DataFrame has doubled over the recent years. Although many important functionalities are still missing (see Section 4), the modern version of DataFrame is more complete and according to the community of its developers, today DataFrame covers all the most common use cases.

- *Low performance.* Both in terms of speed and memory capacity, DataFrame was by far inferior to similar libraries in other languages. Some operations that could be performed in less than a second using pandas, would take more than 20 min in the first version of DataFrame. The datasets with several million rows could freeze the Pharo image. Although, performance is still an issue in modern version of DataFrame, most computationally expensive operations have been optimized. There is currently an ongoing effort to make Pharo DataFrame as efficient as its analogues in other languages (see Section 5).

- *Incomplete coherence with Pharo collections.* Although the DataFrame community was always striving for the compatibility with standard API of Pharo collections, v1.0 contained multiple methods that were inconsistent with other collections. The most striking example were the SQL-like querying methods such as select: columnNames where: aBlock which were incoherent with Smalltalk-style select: aBlock methods. Those and many other cases of incoherent API were fixed in the modern version of DataFrame.

- *Dependency on Roassal2.* The first version of DataFrame provided methods for data visualizations using the Roassal2 library [4]. Although, visualizations are very important for data analysis, such a large dependency was unnecessary and hard to manage. In later versions, the community has decided that DataFrame should remain a simple collection for data analysis, and data visualizations must be delegated to a different library that would be expected to support data frames. Today, data frames can be visualized using the Charting packages of Roassal3.[2] In close collaboration with DataFrame community, the developers of Roassal are currenly working to improve those packages and build a powerful data visualization library.

[2]https://github.com/ObjectProfile/Roassal3

- *Lack of detailed documentation.* The first version of DataFrame was only documented through blog posts, a README file on GitHub, and a short paper by Zaitsev *et al.,* [1]. In recent years, DataFrame community has produced several more forms of documentation, including the DataFrame Booklet [5] and examples of applying data frames for machine learning and data mining on pharo-ai Wiki.[3]

In Table 1, we compare two versions of DataFrame: the first stable varsion v1.0 and the most recent pre-release version pre-v3. Both versions were loaded into Pharo 9 on May 29, 2023. As can be seen in the table, the number of methods in both DataFrame and DataSeries classes has doubled. Although the test coverage[4] of DataFrame has always been high, in recent version, it was increased to 95.43% and the total number of test methods was increased almost six times.

Table 1
Comparing two versions of Pharo DataFrame: v1.0 and pre-v3

	v1.0 (2017)	**pre-v3 (2023)**
Methods in DataFrame class	73	186
Methods in DataSeries class	63	108
Test methods	103	595
Test coverage	72.02%	95.43%

In addition to the changes listed above, the DataFrame community has also introduced several new tools that can be used todether with data frames. Those include the new Spec-based data inspector tool,[5] data imputers[6] that provide different strategies for filling the empty (nil) values in columns by filling them with zeroes, average values, etc. and remembering the statistical properties to ensure reproducibility, and a tool for loading default datasets.[7]

4. DataFrame Outside of Pharo

Being an essential structure in data analysis, data frames have been implemented in many programming languages [6], the most popular of which are pandas (Python) and data.frame (R). Pandas DataFrame is widely used in data analysis and provides a flexible, high-performance way to manipulate, analyse, and visualize data. In R, the data.frame is a fundamental data structure used for storing and manipulating structured data as well. In both libraries data frames offer wide range of functionalities, including data alignment, indexing, merging, filtering, and statistical operations. Their differences are not critical and mostly related to the question of syntax, and spesific strategies of indexing, handling missing vlues etc.

In this section we compare Pharo DataFrame with pandas. We explore commonly used data analysis features based on the litrature in the field [2, 7] and review operations for data import

[3] https://github.com/pharo-ai/wiki
[4] Test coverage was calculated using the DrTests tool in Pharo 9 as the percentage of methods from the core package (DataFrame-Core package in v1.0 and DataFrame package in pre-v3) that are covered by tests from DataFrame-Tests package
[5] https://github.com/pharo-ai/data-inspector
[6] https://github.com/pharo-ai/data-imputers
[7] https://github.com/pharo-ai/datasets

and export, data manipulation (aggregation, grouping, joining, merging, sorting and ranking), visualization, performance optimization and operations for providing more specific ways of analysis (time series, statistical analysis, handling categorical data, etc). We do not go deep into commands details, and do not cover command specific parameters. In Table 2, we show which features from a non-exhaustive list of features from the corresponding categories are present in DataFrame.

Table 2
Selected features of Python's pandas and their presence in Pharo DataFrame.

Data Import / Export:	
CSV	*yes*
Excel	*yes*
SQL	*no*
XML	*no*

Data Manipulation:	
Select data	*yes*
Filter data	*yes*
Add/remove column/row	*yes*
Transpose	*yes*
Handle missing values	*yes*
Grouping and Aggregation	*yes*
Join (inner, outer, left, right)	*yes*
Merge	*yes*
Sort	*yes*
Rank	*no*

Time Series Analysis:	
Handle date/time	*no*
Resample	*no*
Frequency conversion	*no*
Time shifting	*no*
Rolling window	*no*

Statistical Analysis:	
Descriptive statistics	*yes*
Correlation	*yes*
Covariance	*yes*
Regression	*no*

Handling Categorical Data	
Encode categorical variables	*no*
Transform categorical variables	*no*
Create dummy variables	*no*
Categorical data analysis	*no*

As can be seen, the Pharo implementation supports all the listed methods with limited support for Regression analysis (implemented in the Pharo-ai), and some input and output formats (currently XML and SQL are missing). Functionality which is fully missing in DataFrame for the moment, concerns time series analysis and handling categorical data.

5. Pharo DataFrame: future

During the several years of using and developing DataFrame, we have collected the list of improvements, that we plan to implement in the future, which has been extended now by the information received from comparing DataFrame with pandas. The aspects we would like to focus on in the short term regard the library documentation, functional improvements, and issues with a library's general performance. Later we would like to work on adding suport for managing big data in DataFrame.

5.1. Functionality Enhancements

Information retrieved from the comparison with pandas shows that as a first step it will be necessary to add support for time series analysis which is a statistical method for analyzing and forecasting data points collected over time and which is useful for understanding patterns, trends, and dependencies in sequential data (e.g. stocks, weather patterns). The next step will be to add supoprt for handling categorical data for managing and analyzing variables, that represent categories or groups. Although those operations could be performed using the standard Pharo API (*e.g.*,DateAndTime class), this would require writing multiple lines of code for a simple operation. DataFrame could therefore benefit from a dedicated API for time series analysis and handling categorical variables. We also plan to improve support for handling missing values by introducing NaN (*"not a number"* values), and adding support for them to the numerical algorithms of DataFrame.

5.2. Performance

DataFrame performance remains to be a weak spot of the library from both volume and velocity points of view. DataFrame users and developers have observed certain delays in managing data comparing to the other data frames libraries. We plan to benchmark most used operations in DataFrame and compare them to the corresponding implementation in pandas to find the most costly ones and reimplement them to support better velocity. Some operations of DataFrame are also limited by the amount of data that they can handle. Additional study would be required to test DataFrame on the datasets of different sizes and identify the maximum number of rows and columns that can be processed by this library. It must be noted that the performance issues are not due to the programming environment (Pharo) but are caused by the poorly optimized implementation of certain methods in DataFrame. As indicated by Zaitsev *et al.*, [8], numerical algorithms in Pharo can be as fast as those in Python (numpy, pandas, scikit-learn) if implemented using the same low-level optimization techniques. At the moment, Pharo data frames are not capable of handling infinite streams of data (operations should be limited to a given window, e.g. 1000 rows). We would like to improve data frames in this regard that would allow us to work with machine learning projects involving streams of real-time data (e.g. stock predictions, financial transactions etc) We can take the inspiration from the windowing aggregation operations from the `streamz.dataframe` module.

5.3. Big Data Support

Big Data [9, 10] is the concept that has started to gain popularity since the late 90s, which refers to a large, complex and often heterogeneous sets of data that are hard to be effectively manipulated by the traditional data processing techniques due to their inhereted velocity, volume and variety. Big Data is widely used in research and industry fields (including the critical sectors of banking, security, healthcare etc.) and serves as an input for data science and machine learning algorithms. Advantages gained by adoption of Big Data (better information extraction and discovering of patterns and correlations that can lead to improved decision-making and a deeper understanding of underlying (business) processes) create a serious demand for building more tools capable of effectively dealing with it.

Pharo approaches this demand with the Spa framework [11] used in Pharo-based parallel and distributed applications. Spa supports a "spark-alike" MapReduce programming model [12] with different debugging features enabled, allowing to deploy and coordinate various instances and threads of the same Pharo image using different Pharo VMs. At the moment, DataFrame in Pharo is not adapted to be used with big data. DataFrame can not process more data than a Pharo image can allocate and can not be scaled. However, being implemented as a façade providing user with a frontend (API containing all public methods) and hiding a backend (internal data frame representation: a collection for storing data and a set of 29 main methods for manipulating it that all other methods of API are based on), DataFrame provides the possibility to easily substitute its backend implementation. This can be used, for example, to provide optimised data storage, database connection, etc. We would like to change the internal implementation of DataFrame to make it able to treat big data by making it agnostic to the underlying data structure and adding support for various stand-alone database connectors or for distributed framework as Hadoop,[8] Spark,[9] and Spa: a Pharo-native framework. Due to the DataFrame architecture, these changes will be seamless and invisible for library users.

5.4. Better synchronisation with PolyMath and pharo-ai

PolyMath[10] is a Pharo library for scientific computing, similar to existing libraries like NumPy,[11] SciPy[12] for Python or SciRuby[13] for Ruby. It provides basic support for complex and quaternions extensions, random number generators, fuzzy algorithms, automatic differentiation, KDE-trees, numerical methods, Ordinary Differential Equation (ODE) solvers, etc. [13]. Although DataFrame was never part of the PolyMath library itself, it was originally implemented by the community of PolyMath developers and under the umbrella of the PolyMath Organization.[14]

Pharo-ai[15] is another Pharo library that implements algorithms of artificial intelligence. Most of those are algorithms of shallow machine learning (the one that does not rely on deep neural networks), but it also provides tools for data mining, natural language processing, graph algorithms, etc. All algorithms of pharo-ai are meant to be compatible with DataFrame, which comes in handy because training machine learning models is often preceded by data manipulations (cleaning, preprocessing) — the task for which DataFrame is well suited. That being said, users of pharo-ai are not required to use DataFrame as every algorithm can also accept any OrderedCollection of correct shape (i.e. a collection of collections representing a tabular dataset of shape $m \times n$). This is possible thanks to the fact that DataFrame is coherent with the standard API of Pharo collections. The algorithms of pharo-ai do not reference DataFrame directly but only use the methods of Pharo collections. DataFrame has the responsibility to implement those methods.

Despite having different purposes, the three libraries, PolyMath, pharo-ai, and DataFrame,

[8] https://hadoop.apache.org
[9] https://spark.apache.org
[10] https://github.com/PolyMathOrg/PolyMath
[11] https://numpy.org/
[12] https://scipy.org/
[13] http://sciruby.com/
[14] https://github.com/PolyMathOrg
[15] https://github.com/orgs/pharo-ai

have a lot in common. There is an intersection between the communities of their core developers. Those libraries are meant to be compatible with each other (although sometimes this is not the case) and are often used together in the same application. However, for the historical reasons, the open-source development process of those three libraries is not always synchronized. There are cases when machine learning algorithms are placed into PolyMath, mathematical algorithms are implemented in DataFrame, and data processing algorithms are stored in pharo-ai. Although in some cases it makes sense (e.g., data preprocessing and data partitioning algorithms are in pharo-ai because they are designed specifically for machine learning purposes), overall the three communities would benefit from better communication, clearly defined scopes, and more synchronized development process.

6. Conclusion

In this paper, we have discussed the DataFrame library in Pharo programming language. We have presented its current state, and its evolution and provide a high-level comparison with the pandas framework. This gave us valuable insight into the future directions for the library's improvement. As a future work, we foresee the rigorous implementation of the features missing in Pharo DataFrame that are presented in other libraries and are in high demand by library users. We would also like to tackle the performance issues the library possesses at the moment and evaluate the results by benchmarking DataFrame and comparing results with other data frame libraries. We are especially interested to apply future improvemnts for the scenario of handling big data, and we plan rethink the back-end implementation of DataFrame so it could profit from using the external data storages: databases and distributed clusters, and to support the Spa: Pharo-native framework for distributed handling of big data.

References

[1] O. Zaytsev, N. Papoulias, S. Stinckwich, Towards exploratory data analysis for pharo, in: Proceedings of the 12th edition of the International Workshop on Smalltalk Technologies, 2017, pp. 1–6.

[2] W. McKinney, Python for data analysis: Data wrangling with Pandas, NumPy, and Jupyter, " O'Reilly Media, Inc.", 2022.

[3] D. Petersohn, S. Macke, D. Xin, W. Ma, D. Lee, X. Mo, J. E. Gonzalez, J. M. Hellerstein, A. D. Joseph, A. Parameswaran, Towards scalable dataframe systems, arXiv preprint arXiv:2001.00888 (2020).

[4] A. Bergel, Agile Visualization, LULU Press, 2016. URL: http://agilevisualization.com/.

[5] C. F.-D. Oleksandr Zaitsev, Data analysis made simple with pharo dataframe, 2023. URL: https://github.com/SquareBracketAssociates/Booklet-DataFrame.

[6] Awesome Dataframes, https://github.com/jcmkk3/awesome-dataframes, 2023. Accessed: 2023-05-29.

[7] A. Géron, Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems, O'Reilly Media, Sebastopol, CA, 2017.

[8] O. Zaitsev, J. M. no Sebastian, S. Ducasse, How fast is ai in pharo?

benchmarking linear regression, in: IWST 2022-International Workshop on Smalltalk Technologies, 2022.
[9] J. Zakir, T. Seymour, K. Berg, Big data analytics., Issues in Information Systems 16 (2015).
[10] V. Mayer-Schonberger, K. Cukier, Big Data: A Revolution That Will Transform How We Live, Work, and Think, Houghton Mifflin Harcourt, Boston, 2013. URL: http://www.amazon.com/books/dp/0544002695.
[11] M. Marra, A live debugging approach for big data processing applications, Ph.D. thesis, Vrije Universiteit Brussel, 2022.
[12] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Commun. ACM 51 (2008) 107–113. URL: https://doi.org/10.1145/1327452.1327492. doi:10.1145/1327452.1327492.
[13] D. H. Besset, Object-Oriented Implementation of Numerical Methods An Introduction with Pharo, Square Bracket Associates, 2016.

A. Example: Using DataFrame to Analyse Gender Wage Gaps

In this example, we use DataFrame to analyse a dataset of wage gaps between male and female employees in different countries and find the countries with largest and smallest wage gaps in a given year. We use a public Gender Wage Gap dataset provided by OECD (Organisation for Economic Co-operation and Development).[16] The dataset is provided as a CSV file which can be loaded as into Pharo as DataFrame:

```
wageGapFile := 'data/wagegap.csv' asFileReference.
data := DataFrame readFromCsv: wageGapFile.
```

Table 3
Five rows of the Gender Wage Gap dataset before preprocessing

LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	YEAR	Value
PRT	WAGEGAP	EMPLOYEE	PC	A	2014	15.321756895
BRA	WAGEGAP	EMPLOYEE	PC	A	2013	16.363636364
LUX	WAGEGAP	SELFEMPLOYED	PC	A	2020	22.295013428
SVK	WAGEGAP	EMPLOYEE	PC	A	2003	20.689655172
GRC	WAGEGAP	EMPLOYEE	PC	A	2002	23.565754634

In Table 3, we show five randomly selected columns of that DataFrame. The original dataset combines records from employees and self-employed people. To keep only the data about employees, we use select: method which is supported by all Pharo collections. In case of DataFrame, at each iteration, the select block will be evaluated with one row.

```
data := data select: [ :row | (row at: 'SUBJECT') = 'EMPLOYEE' ].
```

To answer the questions listed above, we only need three columns of the dataset: country, year, and wage gap. The columns: method creates a subset of DataFrame with only the given columns. In the next line, we rename those columns using the columnNames: setter.

[16] https://data.oecd.org/earnwage/gender-wage-gap.htm

```
data := data columns: #('LOCATION' 'TIME' 'Value').
data columnNames: #('Country' 'Year' 'Gap').
```

At the final step of data preprocessing, we replace the three-letter country codes with full country names. To do that, we load another dataset which contains the list of ISO-3166 country and dependent territories with UN regional codes.[17]

```
countryCodesFile := 'data/countryCodes.csv' asFileReference.
countryCodes := DataFrame readFromCsv: countryCodesFile.
```

To get the mapping in the form of a dictionary, we first set the *alpha-3* column (country codes) as row names and then access column *name* of the new DataFrame. The column will be an object of class DataSeries which is a kind of OrderedDictionary containing country codes as its keys and country names as its values.

```
countryCodes rowNames: (countryCodes column: 'alpha-3').
countryCodeMapping := countryCodes column: 'name'.
```

The original Wage Gaps dataset contains two values which can not be recognized as valid country codes: *OECD* and *EU27*. We remove them and then apply the mapping to the *Country* column.

```
data := data reject: [ :row |
    #('OECD' 'EU27') includes: (row at: 'Country') ].

data toColumn: 'Country' applyElementwise: [ :each |
    countryCodeMapping at: each ].
```

Table 4
The cleaned dataset ready to be analysed.

Country	Year	Gap
Portugal	2014	15.321756895
Brazil	2013	16.363636364
Slovakia	2003	20.689655172
Greece	2002	23.565754634

In Table 4, we show the same rows were randomly selected for Table 3, after applying the preprocessing described above. The data is now clean and ready to be analysed. We first select the year for which we will compare the gender wage gaps across different countries. The most recent year in the dataset (2022) only contains one entry. We therefore search for the year after 2013 years which has the most entries (we are not interested in data from more than 10 years ago). To do that, we first filter data by year. In the next line, we group the data by years and aggregate it by counting the number of country names per year. Then, using the argmax method, we get the year for which the count value is the largest. That year is 2018.

```
selectedYear := ((data select: [ :row | (row at: 'Year') > 2013 ])
    group: 'Country' by: 'Year' aggregateUsing: [ :group | group size ])
    argmax.
```

[17] https://github.com/lukes/ISO-3166-Countries-with-Regional-Codes

We take a subset of data for only the selected year and sort it in the descending order of wage gaps. Using methods head and tail, we get the top-5 and the bottom-5 countries in the dataset. The result can be seen in Tables 5 and 6.

```
oneYearData := (data select: [ :row | (row at: 'Year') = selectedYear ])
    sortDescendingBy: 'Gap'.

oneYearData head. "top-5 rows"
oneYearData tail. "bottom-5 rows"
```

Table 5
Countries with largest wage gaps

Country	Gap
Korea, Republic of	34.10
Japan	23.53
Estonia	22.68
Israel	22.65
Latvia	20.28

Table 6
Countries with smallest wage gaps

Country	Gap
Hungary	5.06
Denmark	4.86
Romania	3.49
Belgium	3.40
Bulgaria	3.03

This simple example demonstrated the typical use case for a DataFrame library. Although the same preprocessing and analysis steps could be performed on standard Pharo collections, DataFrame makes this process simpler and more intuitive by providing a dedicated API for data manipulation.