AI REGIO

D5.5 Data Quality and Cleansing for AI Applications

Author:	NISSA
Work Package:	WP5 Beyond INDUSTRY 4.0: AI DIH Industry 5.0 Data Spaces
Delivery date:	21.10.2021
Due date:	30.09.2021
Classification:	Public
Type:	Other

H2020 Innovation Action - This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N. 952003

Status of deliverable

Action/role	Name	Date (dd.mm.yyyy)
Submitted by	NISSA	21.10.2021
Responsible (WP leader)	NISSA	21.10.2021
Approved by (internal reviewer)	AIN	21.10.2021

Revision History

Date (dd.mm.yyyy)	Version	Author	Comments
30.08.2021	0.1	NISSA	Deliverable ToC
14.09.2021	0.3	NISSA	Initial content
23.09.2021	0.5	NISSA	Improvement
28.09.2021	0.7	NISSA	Refinement
21.10.2021	0.8	AIN	Internal review
21.10.2021	1.0	NISSA	Final version

Author(s) contact information

Name	Organisation	E-mail
Nenad Stojanovic	Nissatech	Nenad.Stojanovic@nissatech.com

Executive Summary

The AI REGIO project aims to lowering the barriers preventing AI-driven DIHs from implementing fully effective digital transformation pathways for their Manufacturing SMEs.

In this context, five subsystems are going to be designed and implemented in the context of WP4 "Beyond PLATFORMS: AI DIH Open Platforms and DIH platform" and WP5 "Beyond INDUSTRY 4.0: AI DIH Industry 5.0 and Data Sharing Spaces".

This deliverable focuses on Data4AI Platform, which is a new generation of platforms for ensuring data quality for AI applications based on syntax and semantic context.

It is related to task WP5.3, AI DIH Data Quality and Cleansing for AI Applications which aims to ensure that the data will be available for AI applications in the right, quality, quantity and at right time.

This deliverable accompanies the software code developed in the scope of the task WP5.3.

Contents

Figures

Tables

Table of acronyms

AI	Artificial Intelligence
DoA	Description of Action
IDS	Industrial Data Space
OSS	Open Source Software
SME	Small to Medium Enterprise

1 Introduction

1.1 Scope of the Deliverable

This deliverable is related to task WP5.3, which is responsible for ensuring that the data is available for AI applications in the right, quality, quantity and at right time. The task is based on the MIDIH data preprocessing pipeline, but is extended with the requirements for AI applications. The task is very related to WP4.2.

This deliverable accompanies the software code developed in the scope of the task WP5.3

1.2 Structure of the Document

The deliverable is structured as follows:

- Section 2 explains the context for achieving Data Quality for AI applications.
- Section 3 focuses on the details about the implementation of the Data Pre-processing modules.
- Section 4 lists relevant methods .
- Section 5 contains concluding remarks.

2 Data Quality in Data4AI Platform

This section describes the process of ensuring Data Quality in Data4AI Platform.

2.1 Data4AI Platform

Data4AI platform is a new generation of platforms for ensuring data quality for AI applications. The main goal is to provide an efficient and easy for usage infrastructure for enabling manufacturing SMEs to prepare own data for the usage in AI applications. It includes the adapters for connecting relevant data sources and recipes (workflows) for defining data preparation pipelines (or using the available pre-configured ones). The main objective is to include the domain expertise in the data preparation process, but in a convenient way for non-technical expert. In addition, to ensure the reliability of collected data, methods for checking the completeness and validity of data will be applied on the edge (avoiding two most important problems for the AI: missing and corrupted data

In the following figure we present a high-level view on the AI applications

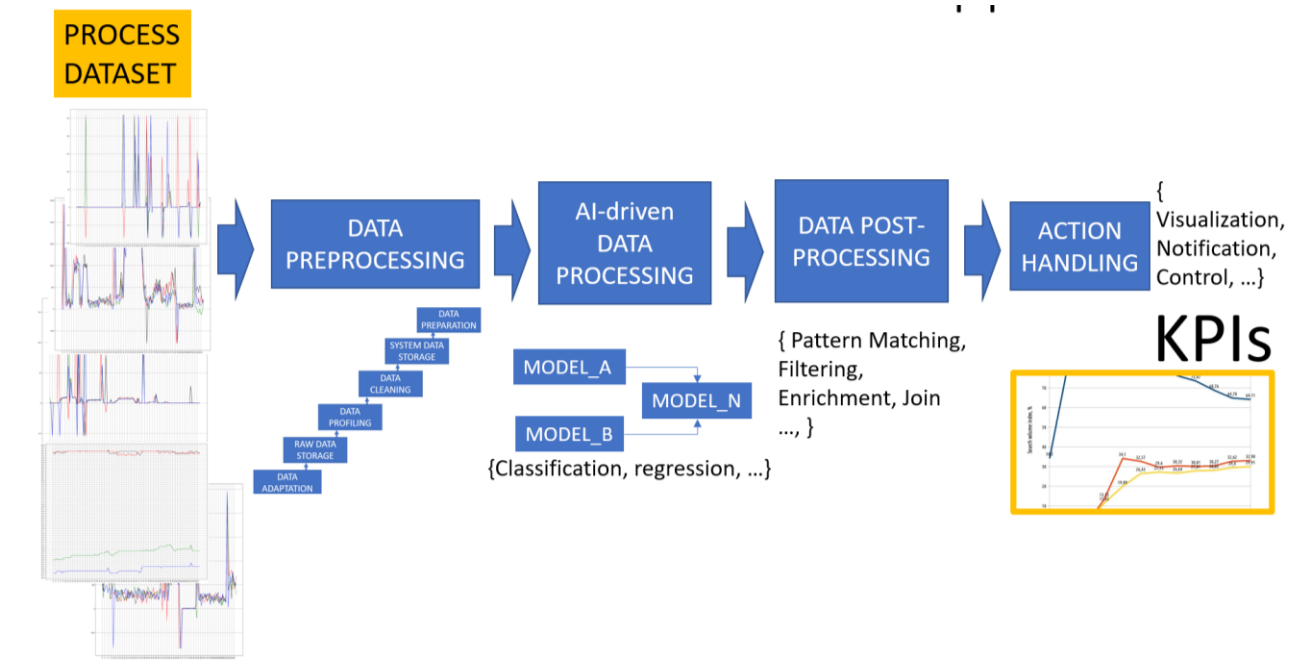

Figure 1: Data processing pipeline for AI applications

We describe the steps in the pipeline briefly:

- Data Preprocessing is a processing pipeline which transforms raw data in the well-formed data (valid structure) that can be processed by various data analysis methods
- AI-driven Data Processing is data analysis which can be done within or outside Data4AIPlatform
- Data Postprocessing enables preparation of the data for output (e.g. filtering)

- Action Handling is related to the delivery of the output to other (control, notification, visualization) systems

2.2 Data Preprocessing pipeline

As depicted in the figure, Data Preprocessing ensures the Data Quality from the syntax point of view (it is in the valid form and can be processed automatically).

In the following figure we describe the Data Preprocessing pipeline.

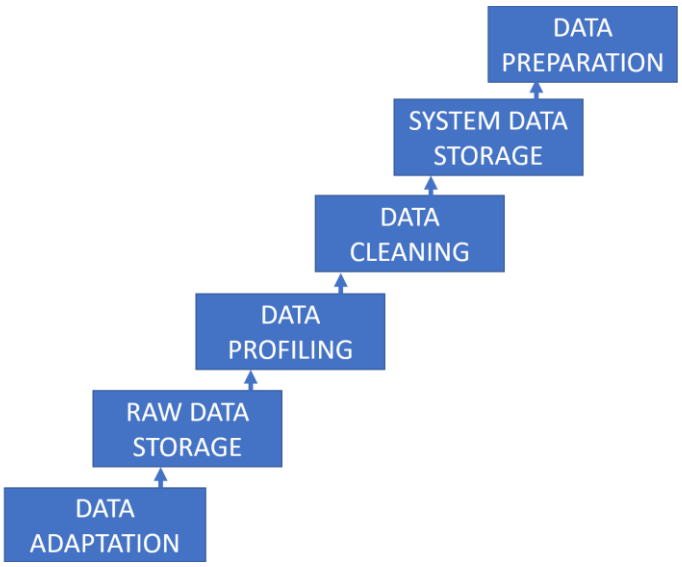

Figure 2: Data Preprocessing pipeline

We briefly describe the particular steps:

- Adapter – Reads the raw data from the relevant data sources (properly defined) and writes data into the raw data storage
- Raw data storage – Stores the raw data from the provided dataset into the previously defined format (d2twin, etc.)
- Profiling – Data Inspection (calculating profiling of the raw data stored in the raw data storage)
- Data cleaning – Data cleaning according the info provided from data profiling (removing irrelevant data from the raw data)
- System data storage – Stores cleaned data after the profiling and cleaning are done
- Data preparation – Getting data from the system data storage and preparing the data for the analytics algorithms

2.3 Architecture

The architecture of the system is presented in the following figure. We describe briefly the two most important components.

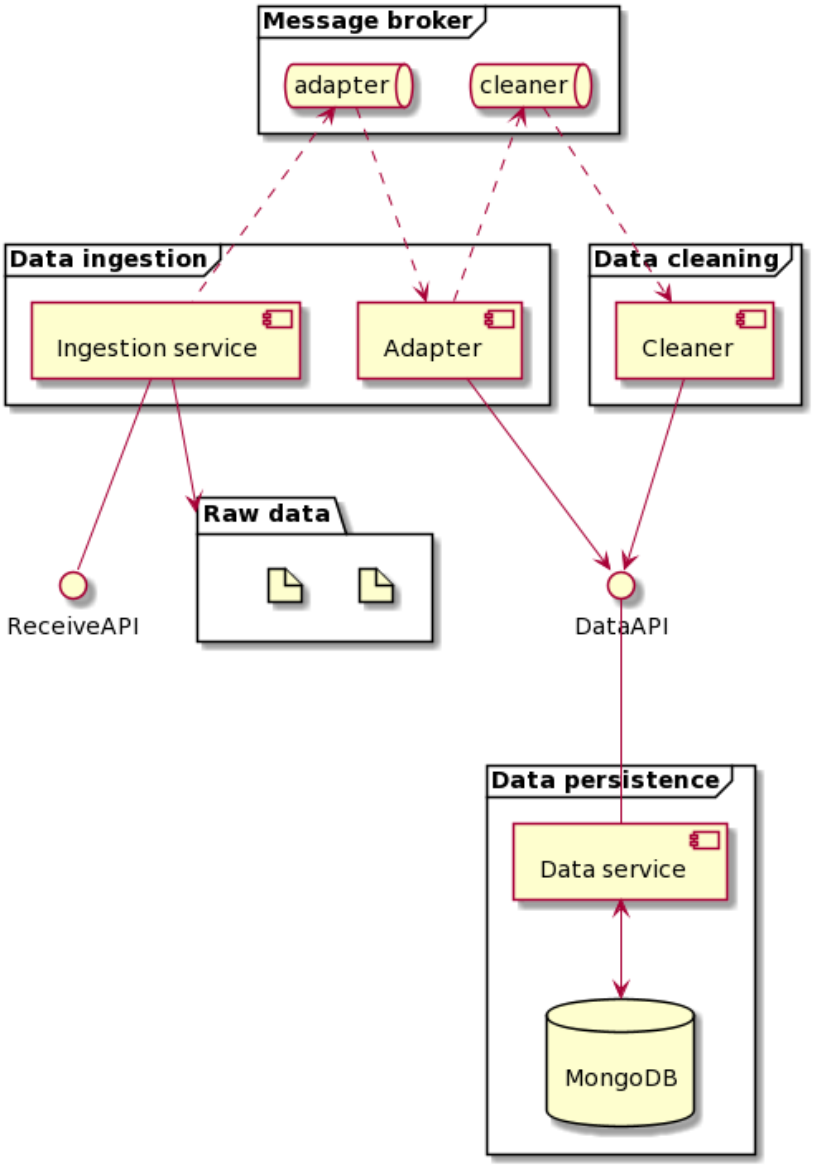

Figure 3: The architecture

2.3.1.1 Adapter

Adapter is a component which is used to transform data from the client to format used in D2Lab. The Adapter is the only component that "knows" how the raw data looks like, and how it should be mapped to our internal format. The rest of the components should be totally decoupled from this information. This also means that implementing a new Adapter is necessary in case the incoming data format is different from the one currently supported.

The Adapter listens for new *raw data* (adapter queue), parses it to the correct format and passes the result to Data service using its DataAPI for storage purposes. Also, the Adapter broadcasts needed information, in order to notify interested components (ex. Cleaner) that there is new raw data to be processed.

2.3.1.2 Cleaner

Data Profiling results are used in order to perform good cleaning of the data and to make a configuration (a *recipe*, which defines methods and the arguments that will be used) for cleaning. The Cleaner is a component which listens on the *cleaner* queue, and if an appropriate cleaning recipe exists in the system, it performs cleaning of the data and passes the results to the DataAPI for storage. Finally, it also broadcasts information to the broker. Cleaner is the last component in the pipeline that is aware of the raw products data, all other components should only deal with the clean data.

Cleaner is a component which modifies and transforms raw data into clean data, by using different methods for getting individual product data to product. The Cleaner can resample, fill missing values, drop parameters...

3 Data Preprocessing

In this section we provide details about the implementation of the Data Preprocessing modules, as described in previous section.

3.1 Cleaner and Preparation method template

We have proposed the following template for methods related to the Cleaner and Preparation. The goal of using templates is to standardize the methods (development and validation).

```
def do_function(data, some_function:str ='', **arg):
    # getting called args
    arg_names = inspect.getfullargspec(do_function).args
    all_local_variables = locals()
    called_args = dict([(key, all_local_variables[key]) for key in arg_names])
    del called_args['data']


    # boolean variable to track if everyhing is ok
    success = True
    try:
        # checking if all arguments are in right format
        # eval is used to "run" string as python code
        some_function = eval(some_function)
    except:
        success = False
        # if something is wrong, we send False, and data which we recived
        return success, data, called_args

    # doing some work
    new_data = some_function(data)

    # returning new_data which were calculated, called_args for creating recipes
    return success, new_data, called_args
```

Important information:

- Data is in data format explained in section 3.2
- All arguments which are used in the method must be received as string
- **arg is there to receive all mistakenly sent arguments (so program wont crash)
- Called_args is there to help with creating recipes (in real time it won't be used)
- Eval is used to evaluate string, it will return the correct format if it is correctly sent.
- If processing is successful, it should return True, new_data and called args.

From the main code (which will call all the functions in proper order), body of the loop will be something like this:

```
kwargs = read_from_recipe()
```

```
kwargs['data'] = get_data()
kwargs = {'some_function':'lambda x: x+17'}
eval('do_function(**kwargs)')
```

Using eval is like runing a new script, so you can pass something like "do_function(**kwargs)" and it will use already defined kwargs.
It will return something like this:
(True, 72, {'some_function': 'lambda x: x+17'})

3.2 Data format

Data format which was already mentioned will have the following format (example data)

Pandas.index	instance_id	stage	param_name	timestamp	value
0	7019672	Finish_Line	D LIFT Punho LH Down Flush	2019-01-07 12:04:50	-1.51
1	7019672	Finish_Line	D LIFT Roof RH Flush	2019-01-07 12:04:50	-4.05
2	7019672	Finish_Line	D LIFT BS RH Gap	2019-01-07 12:04:50	14.52
3	7019672	Finish_Line	D Lift Roof Delta Flush Flush	2019-01-07 12:04:50	-0.02
4	7019672	Finish_Line	D LIFT Punho RH Down Flush	2019-01-07 12:04:50	-1.03

Table 1. Format example

We discussed using multi index, which is for some casses better than the suggested, but in this essential basic format we have little bit more flexibility. Multistage-related format is presented in the table below (same data as Table 1). Indexes are bolded in both tables.

instance_id	stage	param_name	timestamp	value
7019672	**Finish_Line**	**D LIFT Punho LH Down Flush**	2019-01-07 12:04:50	-1.51
		D LIFT Roof RH Flush	2019-01-07 12:04:50	-4.05
		D LIFT BS RH Gap	2019-01-07 12:04:50	14.52
		D Lift Roof Delta Flush Flush	2019-01-07 12:04:50	-0.02
		D LIFT Punho RH Down Flush	2019-01-07 12:04:50	-1.03

Table 2. Multi index format

Products will be stored in a dictionary where key is ther instance_id, and value is the whole product. Method for transforming that product dictionary to data format is `product_dict_to_df` and found in d2labcore.mongo_classes.ProductInstance

Avaible methods, and starting documentation can be found in Section 4.

3.3 DataPreprocessing library

Data Preprocessing is a Python library which can be imported to a local environment by using pip. It contains python scripts:

- Mongo_classes.py - Contains class representation of MongoDB objects, it supports working with JSON format (Reading from and writing to JSON)
- Database_worker.py - Has a class DataProvider which methods will be used to optimally get and write data to database.
- Profiler.py
- Cleaner.py
- Preparation.py

Profiler, cleaner and preparation contain methods so other components can use them.

3.3.1 Profiler characteristics

- Profiler is a set of methods (each method produces one document in collection or are all gathered in one document)
- Profiler can be called for raw and clean data.
- Profiler can be called on data which were used for clustering (on whole products or on used in clustering)
- Profiler uses same frame for storing results (frame should be the same so different queries can be performed, but content of it should be different)
- Profiler shouldn't affect the data (change it in any way) so it will not bias the next method
- Profiler will be used for a data scientist to decide what to do for cleaning and for showing results on the Portal.
- Two different types of results must be made (one for experimenting-Data scientist, and one for the visualization)

3.3.2 Cleaner characteristics

- Cleaner should check if there are all the stages that are defined, otherwise it must skip this product (in real time, we must process the product only when all stages are present)
- Cleaner consists of a set of steps which must be performed
- Cleaner can be issued for different periods of time (eg per shift, time range per day, per business days or over a period of several months, when the data format changes)
- Cleaner should resolve different situations depending on condition (for instance if one peak is bigger than usual, method should be harsher)
- All methods for cleaning must take and return data in the same format
- Methods must be modular
- Methods can't transform data (standardize, normalize, etc.)
- At the end of a cleaning process, all products must have:
    a. Same stages
    b. Same parameters in each stage
    c. Same lengths per parameter for timeseries
    d. Same frequency per parameter for timeseries
    e. All values for parameters must be filled (no nulls)
- Adding new methods should be easy. In a way that does not change the architecture and many changes are not needed for the project.

- Specific method for use case can be a good workaround, but it should always try to generalize the approach so it can be reused
- Cleaner can be performed on a batch (for clustering) and on individual product (real time anomaly detection)

3.3.3 Preparation

- Preparation is the one component that can transform data (e.g. by standardizing), or shape it (take less parameters, or less values for timeseries)
- Preparation can use methods from cleaning to remove parameters and cut time series parameters.
- All methods in Preparation must take and return data in the same data format (so we can change order of methods).
- Preparation shouldn't have use-case specific methods, all methods should "work" for all use-cases
- Preparation can be performed on a batch (for clustering) and on individual product (real time anomaly detection)

3.4 Cleaning_recipes

Cleaning recipe collection contains recipes for cleaning raw products.

Dummy document for collection that will store cleaning recipes looks like:


```
{
   "_id":"MongoDB-generated ID",
   "recipe_id":"vw-123",
   "context":{
      "context_type":"cleaning",
      "process_instance_id":"Process id",
      "start_time":1234567890123,
      "end_time":1234567890123,
      "shift":"all"
   },
   "generation_time":1234567890123,
   "needed_stages":[
      "Framing",
      "Finish_line"
   ],
   "tags":[

   ],
   "operations":[
      {
         "method":"FirstMethod",
         "arguments":{
            "arg1":"5",
            "arg2":"some text"
         }
      },
```

```
        {
            "method":"SecondMethod",
            "arguments":{

            }
        }
    ]
}
```

It is visualized on the next graph.

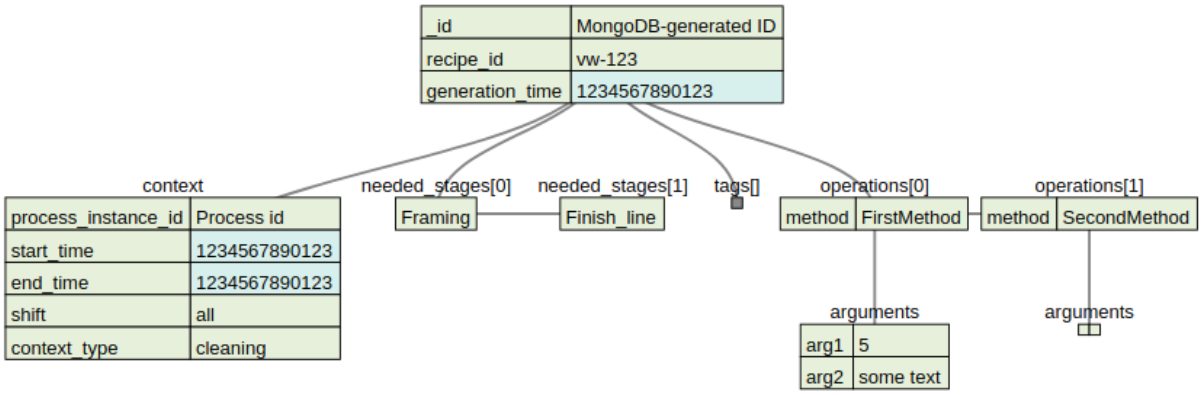

Figure 4: Cleaning recipe example

Used fields explanation:

- **recipe_id** (str) - used to identify different recipes, should be some logical name
- **generation_time** (datetime) - when this recipe is created
- **context** (Dict) - represents cleaning context needed to distinguish required recipe for cleaning
    - **process_instance_id** (str) - from which process cleaning recipe is used
    - **start_time** (datetime) - is datetime of the start datapoint (multiple recipes can be used on one product type if needed, e.g. Format of the data is changed; therefore, different recipe is needed from one point in time)
    - **end_time** (str) - is datetime of end datapoint (multiple recipes can be used on one product type if needed, e.g. Format of the data is changed; therefore, different recipe is needed from one point in time)
    - **shift** (str) - recipes can be used on specific stage
    - **context_type** (str) - used to distinguish context for cleaning rather than preparation
- **needed_stages** (List[str]) – they are a list of stage names which is required to begin the cleaning. If all stages aren't present (there can be more than specified), cleaning skips this product in cleaning process.
- **tags** (List[str]) - are used for easier filtration, if needed
- **operations** (List) - List of operations which can be used from d2core.methods.cleaning and d2core.methods.preparation
    - **method** (str) - exact name in d2core
    - **arguments** (Dict) - needed for the specified method

3.5 Preparation_recipes

Preparation recipes represent ordered methods that need to be called to prepare data for analysis. Preparation recipes are contained in the operations. Operations is a list of methods from either d2core.methods.cleaning or d2core.methods.preparation, here it is expected to find operations that will remove some parameters from analysis, transform data in some way, etc.

Dummy document for collection which will store preparation recipes looks like:


```
{
   "_id":"MongoDB-generated ID",
   "recipe_id":"RecipeID",
   "context":{
      "context_type":"preparation",
      "product_type":"ProductType"
   },
   "generation_time":1234567890123,
   "needed_stages":[
      "Framing",
      "Finish_line"
   ],
   "tags":[

   ],
   "operations":[
      {
         "method":"FirstMethod",
         "arguments":{
            "arg1":"5",
            "arg2":"some text"
         }
      },
      {
         "method":"SecondMethod",
         "arguments":{

         }
      }
   ]
}
```


This is visualized on the next graph.

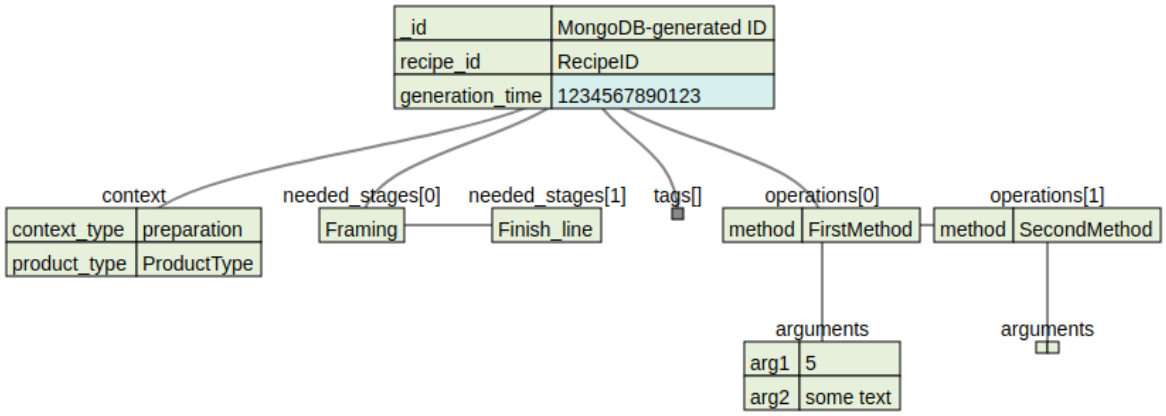

Figure 5: Preparation recipe example

Used fields explanation:

- **recipe_id** (str) - is unique identifier, should be some logical name to easily identify its purpose
- **generation_time** (datetime) - when this recipe is created
- **context** (Dict) - represents cleaning context needed to distinguish required recipe for cleaning
    - **product_type** (str) - for which product is used
- **needed_stages** (List[str]) - are a list of stage names that is required for preparation to begin.
- **tags** (List[str]) - are used for easier filtration, if needed
- **operations** (List) - List of operations that can be used from d2core.methods.cleaning and d2core.methods.preparation
    - **method** (str) - exact name in d2core
    - **arguments** (Dict) - needed for the specified method

4 Methods

In this section detailed lists of the cleaner and the preparation methods is provided

4.1 Cleaner methods

No	Name	Explanation	Method name	Input	Input type
1	Removing parameters	Remove parameters that failed profiling tests or that were determined (by us or the customer) to be unimportant	remove_params	Raw/clean parameter; param_name	Measurements, timeseries
2	removing stage	Removing specific stage from all parameters	remove_stage	Raw/clean parameter; stage_name	Measurements, timeseries
3	removing products	If the product doesn't contain data for the parameter, remove it	remove_products	raw/clean parameter; stage, parameter name	Measurements, timeseries
4	Make new params	Make new params using a formula (like param3 = param1 - param2)	make_new_param	Raw/clean product; add_stage, add_param_name, formula, dict_of_params	parameter
5	Remove outliers	If the parameter has a big outlier, it will be removed. Using specified limits, or something like that	remove_outliers	Raw/clean parameter; stage, param_name, lower_boundary, higher_boundary	Measurements, timeseries
6	resampling	Resampling timeseries to specified frequency	resample_parameter	Raw/clean timeseries; param_name, freq	timeseries
7	resampling	Resampling timeseries to specified frequency	resample_parameters	Raw/clean timeseries; params (list of dictionaries with stage_name, param, etc), freq	timeseries
8	recentring timeseries	Recentring timeseries to avoid being "out of phase"	_recenter_timeseries	Raw/clean timeseries; arguments for the method	timeseries
9	find left bound	Find left limit of time series	_find_left_bound	Raw/clean timeseries;	timeseries

				arguments for the method	
10	fing right bound	Find right limit of time series	_find_right_bound	Raw/clean timeseries; arguments for the method	timeseries
11	cutting timeseries	Cut time series to have the same values	cut_timeseries("Djuradj" method)	Raw/clean timeseries; arguments for the method	timeseries
12	Pick detection and removal	Use peak detection (D2Filter method) to detect peaks and remove them with some method (smoothing, bffill, ffill)	eliminate_picks_using_quartiles	Raw/clean timeseries; boundary for pick detection, method	timeseries
13	Shape checker	Check if you have the correct form, if it does not depend on the input, remove the product, the parameter or something else	remove_bad_shaped_products	Raw/clean parameter; what to do	shape_dict
14	remove products with less data	remove product if length of datapoint(s) is less than specified as argument	remove_products_with_less_datapoints	raw/clean product; shape dict which specifies which stage, which param and how many values	Measurements, timeseries
15	remove products with less parameter data	remove products if it does not contain a specific parameter	remove_product_missing_parameter	raw/clean product; parameter to check for	Measurements, timeseries
16	remove products with missing values	Removes products if it has more than specified number of missing values	remove_products_with_missing_values	raw/clean product; parameters_no, datapoints_no, no_of_allowed_missing_values	Measurements, timeseries
17	Filling parameters with value	Fill parameters with value	fill_param_with_value	raw/clean parameter; stage_name, param_name, value	Measurements, timeseries
18	Resample malca data	Use-case specific method which prepares data for selected dataset	resample_malca_data	raw/clean parameter; stage_name, min_no_of_rows_needed, max_seconds_missing, max_rows_after, resample_format	timeseries
19	truncate timeseries	Remove datapoints with values outside limits	truncate_timeseries	Raw/clean timeseries;	timeseries

20	resample_timeseries	Resample timeseries data. Optimised for S4F use case.	resample_timeseries	Raw/clean timeseries; frequency, list of parameter names, list of stage names	timeseries
21	interpolate_timeseries	Interpolate timeseries data. Optimised for S4F use case.	interpolate_timeseries	Raw/clean timeseries, interpolation method,list of parameter names, list of stage names	timeseries
22	strech_timeseries	Reduce timeseries to the same duration.	strech_timeseries	Raw/clean timeseries, treshold, duration, interval, order, list of parameter names, list of stage names	timeseries
23	remove_long_and_short_duration_timeseries	Remove all parameters that do not have a duration d that is a>d<b. Where a and b are function parameters.	remove_long_and_short_duration_timeseries	Raw/clean timeseries, lower bound, upper bound, time units,list of parameter names, list of stage names	timeseries
24	remove_long_and_short_length_timeseries	Remove all parameters that do not have the length l that is a>l<b. Where a and b are function parameters.	remove_long_and_short_length_timeseries	Raw/clean timeseries, lower bound, upper bound, list of parameter names, list of stage names	timeseries
25	Filling timeseries	Fill timeseries data (bfill, ffill, average fill)		Raw/clean parameter;	timeseries
26	Filling parameters with values	Fill parameters with values	fill_params_with_values	Raw/clean data, filling dictionary	Measurements, timeseries
27	Replace strings in numeric columns	Replace strings in numeric columns	replace_strings_in_numeric_columns	Raw/clean data, filling dictionary	Measurements, timeseries

4.2 Preparation methods

No	Name	Explanation	Method name	Input	Input type
1	filtering data by date	Remove product which isn't in a specified date range	filter_data_by_date	Clean parameters; after, before, stage_name	Measurements, timeseries

2	Scaling	Scaling should support different methods (standardization, normalization...)	scale_data	Clean parameters; method, arguments for method (min, max for normalization; average, std_dev for standardization...)	Measurements, timeseries
3	Concatenate	Concatenate ALL products sent to this method	concatenate_timeseries	Clean paramters; timeseries_name	Measurements, timeseries
4	Vectorize timeseries	Create one instance of all sent data. Take the first data point of a parameter from all instances and create a new time series parameter from them	vectorize_timeseries	Clean parameters	timeseries
5	Merge stages	Merge multiple stages into a single stage (change stage_name)	merge_stages	Stage names to merge, new stage name	Measurements, timeseries
6	Truncate timeseries	Truncate timeseries so that it takes from left to right limit	truncate_timeseries	Clean paramters; left_bound, right_bound	timeseries
7	Smoothing	Smoothing parameters using a window	smooth_timeseries	Clean parameters; windows size	timeseries
8	Transformation	Wavelet, discretization		Clean parameters; method; arguments for the method	Measurements, timeseries
9	Differencing	Differentiation is a method to subtract two time series		clean parameters; order	timeseries
10	Dimesionallity reduction	Dimensionality reduction		Clean product; method; method arguments	product
11	Aggregation				
12	Generalization				
13	Different tests and algorithms				

5 Conclusion

This deliverable focuses on Data4AI Platform, which is a new generation of platforms for ensuring data quality for AI applications based on syntaxis and semantic context.

It is related to task WP5.3, AI DIH Data Quality and Cleansing for AI Applications which aims to ensure that data is available to AI applications in the right quality, quantity and time.

This deliverable accompanies the software code developed in the scope of the task WP5.3.