Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 86 (2019) 31–36 www.elsevier.com/locate/procedia

7th CIRP Global Web Conference
"Towards shifted production value stream patterns through inference of data, models, and technology"

# Web-based Platform for Data Analysis and Monitoring

Colin Reiff *, Stefan Oechsle, Florian Eger, Alexander Verl

*Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW), University of Stuttgart, Seidenstrasse 36, 70174 Stuttgart, Germany*

* Corresponding author. Tel: +49-711-685-84512; fax: +49-711-685-82808. *E-mail address:* colin.reiff@isw.uni-stuttgart.de

**Abstract**

With the increasing number of sensors installed in production systems and the associated amount of recorded data, a high potential for optimizing manufacturing processes is offered. Based on the gathered information, data analyzing methods can be used to identify correlations and thus, optimize complex multi-stage production systems with the goal of zero-defect manufacturing. For this purpose, multiple statistical methods have to be applied, which is often a time-consuming task and currently requires trained and experienced specialists. Furthermore, the investigated data must be kept up to date for the early recognition of changes possibly leading to cost-intensive scrap and production downtimes. To address these issues, a web-based platform with an intuitive user interface was developed. The platform can access and process various data sources using modular methods. In addition, as part of continuous monitoring, the results of these analyzing steps can be dynamically calculated. The aim of the platform is to make complex analysis processes accessible to machine operators and thus to combine domain expertise and statistical knowledge. The paper describes the underlying architecture and the relevant interfaces of the platform.



*Keywords:* Data Analysis; Monitoring; Process Optimization

## 1. Introduction

Today's working environment and the general situation of European manufacturing companies are characterized by a high level of dynamism and competition. One of the basic prerequisites for remaining competitive in the mid and long term is the efficient use of advanced technologies that have the potential to improve the value creation at all levels of production and to support people in their work areas. The information and communication technologies are one of these fundamental drivers, which enable to create intelligent, flexible, and autonomous cooperation in production by networking operators, manufacturing processes, workpieces as well as storage and transport systems. These disruptive technologies and the general digital transformation in production are endangering the current market position and competitive advantages of established companies in an already cost-intensive and dynamic environment. Therefore, the ability of companies to react correctly to these increasing challenges and to exploit the potential of new technologies is a key factor for the future success. These aspects show that the constantly changing framework conditions put companies under pressure to act and deal with these new technologies and strategies in order to be able to position themselves successfully on the market. One of these strategies also includes the paradigm of zero-defect manufacturing (ZDM), which has the vision of avoiding costs by the prevention and compensation of defects and guaranteeing high product quality [1]. In this context, the EU research project "ForZDM" was launched with the aim of developing methods and tools [2], which support the industry in this changing environment and enable more efficient manufacturing processes [3]. Especially multi-stage manufacturing systems of high-quality products offer a great potential to avoid defects by a holistic and data-driven approach [4,5]. Therefore, the early and comprehensive analysis of a big amount of data, gathered from the increasing number of sensor systems, is indispensable. These circumstances necessitate the use of various statistical methods as well as trained and experienced specialists [6]. In general, this leads to silos of knowledge, which counteract the idea of shared and interconnected solutions with regard to the zerodefect manufacturing paradigm.

 10.1016/j.procir.2020.01.009

This paper is structured as follows: Section 1 presents the motivation and main objectives of the platform, including fundamental specifications derived from the state of the art. Section 2 contains design decisions and relevant interfaces of the platform. Section 3 shows the implementation and user interface. Section 4 summarizes and critically examines the results and describes future work as well as possibilities offered by the developed software platform.

### *1.1. State of the Art and Motivation*

According to a survey from 2018 by KDnuggets [7], Python is currently the most popular programming language in the field of data science and has replaced R. In order to be able to analyze data using the above languages, sound experience is required, since the necessary logic for processing the data must be programmed. Often, many calculation processes have to be combined to achieve the desired results. In addition, the execution of analysis processes requires advanced statistical knowledge to be able to apply the corresponding methods effectively. Therefore, specialists, which are a cost-intensive factor, are required to carry out the necessary programming steps [8]. However, this creates a gap between the analyst and the actual end user who wants to use the knowledge gained from the analysis for their application. Consequently, there are already a multitude of frameworks and platforms that aim to make these statistical methods accessible to users without basic programming experience. Therefore, visual programming approaches and additional support functions such as question catalogues and wizards are used. According to Gartner [9] the leading tools in the field of data science are: RapidMiner, IBM SPSS and KNIME. Detailed reviews of existing data mining tools can be found in [10] and [11].

However, these tools have limitations that hinder the transfer of knowledge across different corporate domains. Most of these applications are often desktop-based programs. This means that an exchange of data, sharing of results and knowledge are limited. Furthermore, the available functions can often only be adapted to a restricted extent or with increased effort. Additionally, the most platforms are not based on Python. This means that the extensibility is often extremely time-consuming, since the corresponding functions, which are often available in Python, must be programmed from scratch or linked via provided interfaces. Due to the identified deficits this paper aims at the development of an application, which targets the following key requirements:

- Accessibility: different devices and users.
- Usability: visual programming interface.
- Community: sharing of functionalities and results.
- Extensibility: integration and adaptability of Python logic.

### *1.2. Specification*

To overcome the mentioned limitations and achieve the described requirements, this paper presents a web-based platform developed in Python Django (back-end) with an intuitive user interface (front-end) created with Angular. The platform should process data from several sources with modular structured methods, enabling even laypersons to use advanced statistical methods. In addition, the platform should be extendable by implementing and adapting methods using an integrated code editor. The aim of the platform is to provide a user-oriented tool to exchange knowledge gained through data analysis and share created analyzing processes by an integrated community store. The architecture of the developed platform is based on a client-server model. The back-end deployed on the server, provides all functionalities and services. The user can then interact with the front-end of the platform using a web browser, which represents the client. The front-end is responsible for coordinating the graphical representation of the functionalities on the client side, as well as for reacting to user interactions. The **front-end communicates with the back-end via a representational state transfer (REST) application programming interface (API)**. The back-end has the task of managing data in the internal database. These data are for example information about created analysis chains, as well as information about the registered users. Fig. 1 shows the schematic structure of the platform architecture.

## 2. Design

To provide a platform which is as flexible as possible, regarding different use cases, the required methods, and algorithms for performing calculations, are structured in separated processing blocks. These blocks, the so-called stages, are independent of the respective data they are meant to process. The goal in structuring the components required for performing analytical procedures (here called pipelines) is to make these usable for different application purposes. This means that the basic structure of such a composition of individual analytical methods is constant and can be extended accordingly. The components used for this structure are shown schematically in Fig. 2 and will be described in more detail in the following.

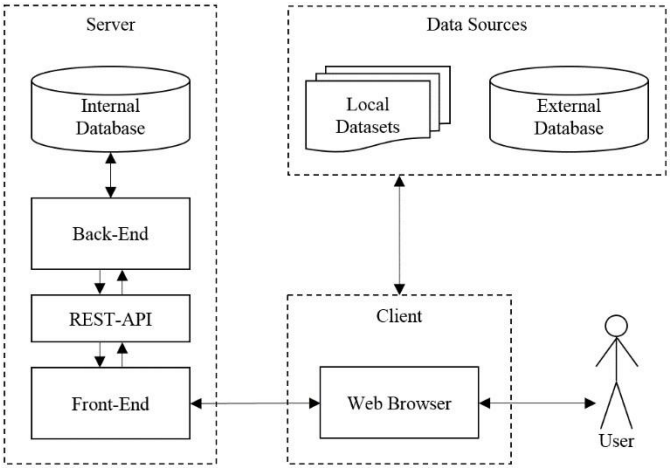


Fig. 1. Visualization of the platform architecture.

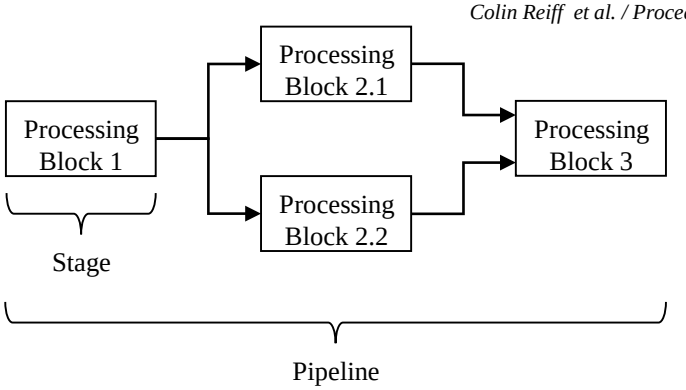


Fig. 2. Schematic representation of a stage and pipeline.

### 2.1. Stage

A stage is the smallest unit of a data processing operation and defines individual processing steps. These processing steps can represent different tasks. For example, the task of a stage can be to load data from an Excel file and to make it available for further processing. The task of a subsequent stage in turn can be to calculate a correlation from this data. The function of a stage is defined by a Python script and is stored on the server. By defining setting parameters, the program code belonging to the stage and thus also the execution can be directly adapted to specific use cases. A stage has a definable number of inputs and outputs. When connecting an output of a stage with the input of another stage, the input data is defined. Furthermore, the developer of a stage has the option to declare it as public. This makes it possible to make this stage accessible for other users of the platform. In addition, public stages can be evaluated, making it easier to identify well-functioning and useful stages.

### 2.2. Pipeline

A pipeline contains all the stages used in a analysis process as well as the associated parameters and thus specifies the results of the calculation created by the user. The processing sequence is determined by the mentioned connection of the stages with each other. The user also has the option of connecting one output of a stage to several inputs of other stages. This ensures that different processing steps can be done in parallel. An application case can be for example a comparison of the correlation coefficient before and after the used data has been preprocessed. Similar to stages, pipelines can also be published and made available to the community. This makes it possible to perform typical and repetitive analyzing processes without reassembling the necessary stages. In addition, predefined pipelines created by data scientists or experienced persons can be shared to all other users of the platform. This enables also untrained employees to use advanced and, in principle, complex data processing algorithms for their specific use cases without the need of programming.

### 2.3. Task

A task corresponds to the entirety of an analysis process and consists of the associated pipeline and additional information. The purpose of a task is to execute the complete pipeline and display defined results on the dashboard of the platform. A result corresponds to a marked output of a stage. In this way, the user has the possibility to link certain results of the used stages with the task and share those with other users. Furthermore, a task can coordinate the cyclic execution of the according pipeline in order to enable continuous monitoring. The task initiates all necessary steps for updating the data source at an interval defined by the user. For this purpose, the platform can request the latest data of a connected external database. The visual representation of task results on the dashboard is updated as well.

### 2.4. Internal Database and Models

The internal database (SQLite) is used to store the necessary information for providing the described functionalities. Fig. 3 shows the models which are used, as well as their attributes and relationships to each other as "entity-relationship-models", in an UML diagram. The user model includes the login information for the platform. It is linked to a profile to be able to store additional information relating to a user. In this profile, the company affiliation and the role of the user are defined. Furthermore, a user has the possibility to add stages and pipelines, marked as public, to his collection. The stages used in a pipeline are stored in a so-called build file, together with the connections between the stages. The parameters set by the user during the creation of an analysis process are stored in the *ParameterConfig* model and linked to the pipeline. The model contains a list of parameter value pairs (*ParameterValue*), which consist of the combination of the corresponding parameter and the value selected by the user.

In order to integrate the data to be analyzed into the platform, the user can upload them in the form of various file formats or connect an external database using data queries and receive thereby the returned data in form of a HTTP response in the JavaScript Notation (JSON) format. These files are subsequently stored in the data model. The assignment of the data file to a corresponding stage, which expects a file as input, is again done by a parameter-value pair. This abstraction is necessary, because a stage can be used by several users with different data inputs and therefore, a direct link between the stage and the data model is not possible. Likewise, the file is

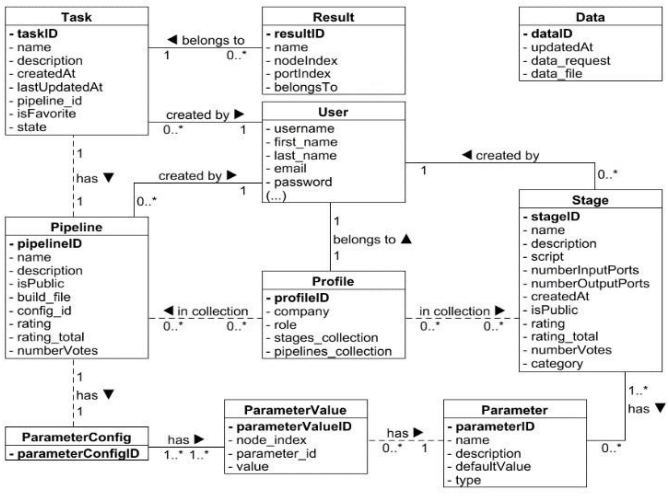


Fig. 3. Models of the internal database and their relationships.

not stored directly in the task or pipeline, so that more than one data source can be included. To avoid cyclic dependencies between models, that would lead to an endless hierarchical structure, certain models are not linked by their primary key (bold). Instead, these keys are only stored as attributes in the models mentioned. With this structure of the internal database, the required modularity can be achieved as well as scalability regarding the number of users and the associated amount of data.

## 3. Implementation of the Platform

In the following, the concrete implementations of the various functionalities of the platform are discussed. The frontend was created using the Angular framework. The communication between the front-end and the back-end takes place using the REST API paradigm, which is implemented with the Django REST Framework. Furthermore, the REST API is structured in such a way that the individual stages can also be called and executed from outside the platform. This means that processing steps can also be used by other, external applications.

### 3.1. Structure of the Data Model

All data used as inputs and outputs of the stages have the same structure and are implemented as a class. The attributes of this data class are: *sensor_id*, *description*, *resource_id*, *measuring_unit*, *values*, *time_stamps* and *error* (see Fig. 4). The error attribute can be used to pass generated errors within a stage to the platform and display them to the user. This data structure contributes to the modularity of the stages and enables a uniform processing. This ensures that all stages, as well as the front-end, can access the attributes of the data in the same way.

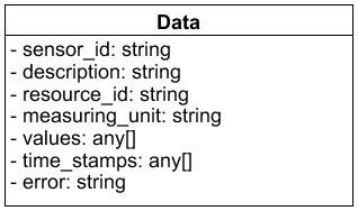


Fig 4. Attributes of the data model.

### 3.2. Construction of a Stage

In order to ensure a uniform structure of the stages, they are derived from a template called *AbstractStage*. The *AbstractStage* defines the interfaces (*input* and *output*) and execution of the included methods (*init* and *execute*). The two properties *input* and *output* represent the interface to previous and following stages respectively and transfer data objects. Whereas *init* and *execute* represent abstract methods, which are used for the initialization of the parameters required by the stage as well as for the actual execution.

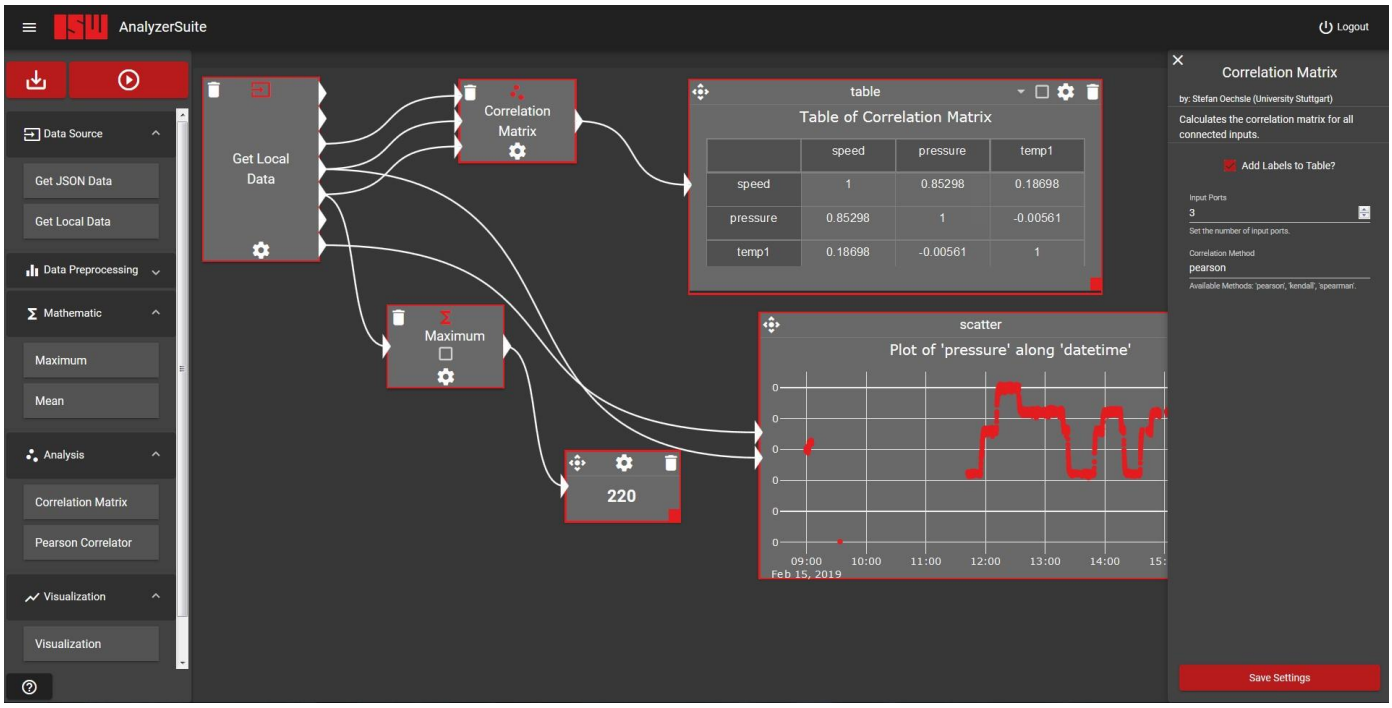


Fig. 5. Screenshot of the user interface for modelling new pipelines.

### 3.3. Creating a New Task

The user interface for creating a new pipeline, as shown in Fig. 5, is divided into three sections: a list of available stages, a visual programming area and a view for setting parameters of a selected stage. On the left side of the interface, all stages which are available for the specific user are listed. This list includes, for example, self-created stages, but also public stages, which are added to the collection by the user. The available stages are displayed in their respective categories and can be added to the modeling level by double-clicking. The different categories are: *Data Source*, *Data Preprocessing*, *Mathematic*, *Analysis*, *Visualization* and *Uncategorized*. To save a pipeline or a task, additional information, such as a description of whether this pipeline should be public, can be entered via a dialog window.

On the visual programming area, which is located in the middle of the user interface, all selected stages are represented as blocks. The inputs of a stage are listed on the left side of such a block. The number of these input ports is either defined during creation or can be set later using a parameter. The outputs are located on the right-hand side. The number of outputs is dynamically adjusted to the number of data objects returned by a stage. There are also icons for deleting and opening the parameter settings within a stage block. A checkbox allows the user to view the results of this stage on the dashboard. Stage block connections are made by pressing and holding the left mouse button on a stage output. The user is then shown a Bézier curve from this output to the current mouse position. By releasing the mouse button on an input of another stage, the two ports can be connected. The connections are displayed as Scalable Vector Graphics (SVG) paths. If the user moves a block within the modeling plane, the associated paths are also updated. As soon as all inputs of a stage are occupied with connections, the execution of this stage takes place automatically. By clicking on the gear symbol of a stage, the parameters linked to the stage as well as the description of the stage are displayed on the right side of the user interface. Here, the user has the possibility to make and apply the desired settings. This also executes a recursive algorithm that updates all subsequent stages accordingly. For the storage of the entire pipeline, all used stages, together with further additional information, are summarized in the mentioned build file. This additional information is, for example, the positions on the modeling level. Furthermore, the connections created by the user are also stored in this build file. The build file is then saved on the server and linked to the database entry of the associated pipeline. This allows to completely reconstruct a pipeline later.

### *3.4. Dashboard – Overview of Created Tasks*

The so-called *dashboard* represents the entry page of the platform, after a successful login of the user. Here, the tasks created by the user or shared by others, are displayed with the corresponding results. The user has the possibility to edit, delete and mark tasks as favorites. Furthermore, all tasks can be filtered according to their execution status (finished, running, pending, and failed). When creating a task, the user can choose which results are to be displayed on the dashboard. To display these results, a distinction is made between the values involved, i.e. how many value series are used. If it is not a single value (or text), the following representations are possible: t*able*, *line chart*, *bar chart*, *scatterplot*, *heatmap* and *contour diagram*. The displaying modes, which can be selected by the user, depend on the number of data transmitted. The open source library Plotly.js [12] as well as an extension for the use under Angular are used for the actual representation of the diagrams. Fig. 6 shows a part of the dashboard containing an executed task and the calculated results. The cyclic execution of the tasks starts automatically, as soon as the user is on the dashboard. For the displaying on the dashboard, only the stages marked as result are of interest. Accordingly, the entire pipeline of a task is not executed each time, but only the stages that are necessary for the calculation of the result. Therefore, a recursive algorithm is used. This algorithm starts with the result stages and follows all connections that are linked to the inputs of these stages. The respective stages reached are added to the list to be processed and the same procedure is applied to them. The algorithm terminates as soon as all stages have been processed and they no longer have incoming connections.

The result of the recursive algorithm is a kind of tree structure of stages, which are also executed recursively, starting with the "leaves". The last stage executed, i.e. the "root" of the tree structure, is the newly calculated result. The use of this algorithm is intended to improve the performance, since stages that are not required for the result will not send data over the network. This is particularly important when processing a large amount of data.

### *3.5. Expert Page – Online Code Editor for Stages*

The modular structure of the platform allows users to add their own stages in the form of Python scripts. The expert page provides an integrated code editor for this purpose. For embedding the editor Ace [13] is used together with an angular integration. In this editor, the corresponding source code of the stage can be entered directly. It contains the basic structure of an exemplary implementation of a stage (*AbstractStage*). The user can also define the desired number of inputs, outputs, and link parameters to the stage. In addition, the user has the possibility to reuse existing parameters or to create new ones. A new parameter is defined by its name, type, default value and description. Furthermore, the user can directly test the created stage online. To save the stage, the user can enter additional information about this stage in a dialog. The additional information can be the category, the description or whether this stage should be public (shared with other users) or not. The name of the stage, on the other hand, is automatically derived from the class name used in the source code.

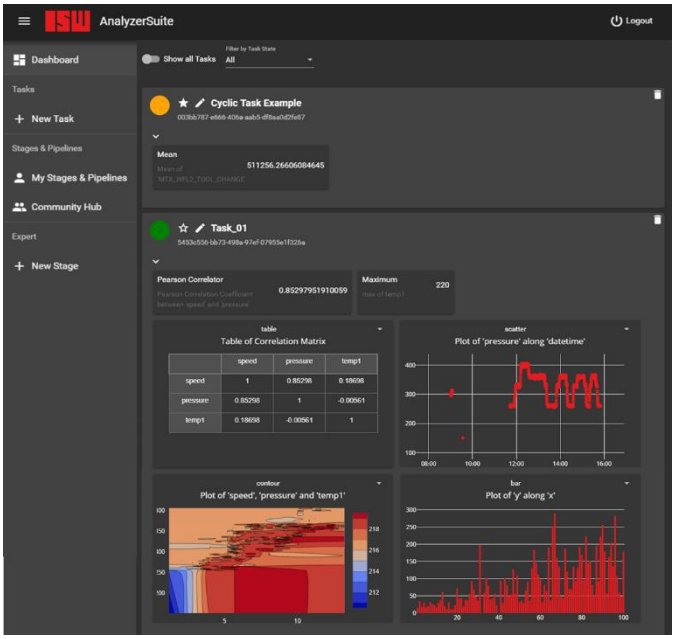


Fig. 6. Screenshot of the dashboard with a task.

## 4. Conclusion and Future Work

This paper introduces a web-based platform developed in Python Django for the modular analysis of data received from local files and external databases. The modularity is achieved by encapsulating single calculations and algorithms in socalled stages. These stages can be combined in processing pipelines. Through the integration of visual programming for modeling the analysis steps, a high usability is achieved. The platform offers the possibility of a cyclical execution of the created pipelines and thus enables continuous monitoring of processes. The developed platform shows a possibility to combine the exploitation and analysis of data with a community idea. The possibility of extending and adapting methods using an online Python code editor enables a high growth potential and thus a universal applicability result. Due to the modular architecture, a broad range of application scenarios can be addressed. The possibility of sharing stages and predefined pipelines in combination with modularity enables to map specialist knowledge and skills with relevant use cases from non-specialists.

Nevertheless, there are possibilities for improvement and extension of the platform. On the one hand, the implementation of further stages, especially stages for the integration of machine learning algorithms, would be conceivable. Control instructions could also be implemented, which would allow branching in the pipeline depending on the input values. Furthermore, it would be useful for certain applications if the parameters of one stage were set by the output value of another stage. This would enable the user to create pipelines that dynamically adapt to the data to be processed. A service could also be integrated that stores certain user activities (e.g. deleting stages, creating tasks, etc.) in a corresponding log file. This could make it easier to find the cause of errors. At the same time, a kind of archive or "recycle bin" would be conceivable, for example for restoring accidentally deleted tasks. With regard to rights management, this could be restructured in such a way that the various functionalities can be permitted or prohibited individually for each user. This would allow a more detailed distribution of rights. With regard to the user interface, a profile page would be useful, on which personal information, such as name and company, can be customized. Furthermore, a function for requesting certain rights or a certain role could be integrated, whereby the request would have to be checked by the administrator.

Additionally, a mobile version of the platform could be developed to extend the use of the platform in an industrial environment, especially at field level. This would enable users to see and interpret results off the dashboard directly on a tablet or on similar devices. The user-friendliness can also be further increased, making it easier for non-technical users to get started and use the software. This can be done, for example, by a kind of setup wizard when creating new pipelines. Defined questions built up in a decision tree could lead the user step by step to the desired pipeline, which is then generated automatically. Possible questions are, for example, what kind of data is involved, whether pre-processing is desired and which benefit of the analysis is aimed at.

## Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 723698. This paper reflects only the author's views and the Commission is not responsible for any use that may be made of the information contained therein.

## References

[1] Eger, F., Coupek, D., Caputo, D., Colledani, M., Penalva, M., Ortiz, J.A., Freiberger, H., Kollegger, G. Zero Defect Manufacturing Strategies for Reduction of Scrap and Inspection Effort in Multi-stage Production Systems. In: Procedia CIRP, 67, 2018. pp. 368–373.

[2] Reiff, C., Eger, F., Korb, T., Freiberger, H., Verl, A. User Interface for the Acquisition and Characterization of Defects and Performed Rework in Multi-Stage Production Systems. In: Procedia CIRP, 78, 2018. pp. 243–248.

[3] Eger, F., Reiff, C., Colledani, M., Verl, A. Knowledge Capturing Platform in Multi-Stage Production Systems for Zero-Defect Manufacturing: Submitted. In: 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), 2018.

[4] Reiff, C., Eger, F., Tempel, P., Magnanini, M.C., Ortiz, J.A., Colledani, M., Verl, A., Sarries, I. Smart centering for rotation-symmetric parts in multi-stage production systems for zero-defect manufacturing. In: Procedia CIRP, 79, 2019. pp. 27–32.

[5] Wang, K.-S. Towards zero-defect manufacturing (ZDM)—a data mining approach. In: Advances in Manufacturing, 1 (1), 2013. pp. 62–74.

[6] Eger, F., Reiff, C., Brantl, B., Colledani, M., Verl, A. Correlation analysis methods in multi-stage production systems for reaching zero-defect manufacturing. In: Procedia CIRP, 72, 2018. pp. 635–640.

[7] Piatetsky, G., 2018. Python eats away at R: Top Software for Analytics, Data Science, Machine Learning in 2018: Trends and Analysis. KDnuggets. https://www.kdnuggets.com/2018/05/poll-tools-analyticsdata-science-machine-learning-results.html. Accessed 14 August 2019.

[8] Shang, Z., Zgraggen, E., Buratti, B., Kossmann, F., Eichmann, P., Chung, Y., Binnig, C., Upfal, E., Kraska, T., 2019. Democratizing Data Science through Interactive Curation of ML Pipelines, in: Proceedings of the 2019 International Conference on Management of Data - SIGMOD '19, Amsterdam, Netherlands. 30.06.2019 - 05.07.2019. ACM Press, New York, USA, pp. 1171–1188.

[9] Idoine, C., Krensky, P., Brethenoux, E., Linden, A., 2019. Magic Quadrant for Data Science and Machine Learning Platforms. Gartner, Inc. https://www.gartner.com/doc/reprints?id=1-65WC0O1&ct=190128&st=sb. Accessed 15 August 2019.

[10] Rangra, K., Bansal, K.L. Comparative study of data mining tools. In: International journal of advanced research in computer science and software engineering, 4 (6), 2014.

[11] Jovic, A., Brkic, K., Bogunovic, N., 2014 - 2014. An overview of free software tools for general data mining, in: 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia. 26.05.2014 - 30.05.2014. IEEE, pp. 1112–1117.

[12] Plotly Technologies Inc., 2015. Collaborative data science. https://plot.ly.

[13] ACE, 2010. Ajax.org Cloud9 Editor. https://ace.c9.io.