ABSTRACT
In recent years data has become a valuable asset to many organizations. Several problems, such as missing values and outliers, can have a negative impact on data quality. Poor data quality negatively affects the quality of results obtained from analyzing the data, which in turn negatively impacts the quality of decisions made by organizations. Data profiling, the set of processes to examine data sets and produce metadata, can help in the discovery of such data quality problems. Data profiling is also useful for discovering dependencies on the data, such as functional and inclusion dependencies. Metadata produced by data profiling is of special interest for data integration, in which the data sets to be integrated are often unfamiliar. Pentaho Data Integration (PDI) is a well-known data integration tool with an open source version, open to additions by the community. PDI currently lacks many data profiling functionalities. One of the missing data profiling functionalities that has been deemed relevant by PDI users is outlier detection. The present work is intended to address this flaw by adding the outlier detection functionality to PDI. The system consists of 2 modules: one for detecting outliers and one for visualizing outliers. The detection of outliers is done using four well-know algorithms from the outlier detection literature: DB(p, d)-Outliers, LOF, DBSCAN, and ABOD. A simple outlier ensemble, for adding robustness to the results of individual algorithms, is also included. The visualization of outliers is done with the help of two suitable data visualizations: Scatterplot and Parallel Coordinates. The outlier detection module is completely independent from the visualization module, but the opposite is not true; the visualization module works on the output of the outlier detection module. The results from usability tests show that the system is easy to use and was well accepted by frequent users of PDI. However, users revealed some difficulty in completing a task that integrated our work within a normal PDI workflow.

KEYWORDS
Outlier Detection, Anomaly Detection, Data Profiling, Data Integration, Pentaho Data Integration

1 INTRODUCTION
With vast amounts of data currently available, organizations are focused on exploiting data for competitive advantage [11]. In addition to that, computers have become more powerful, storage capacity has increased dramatically, and many data analysis algorithms have been developed. Together, all these factors boosted the use of data analytics for effective decision making. However, decisions based on results from data analytics are only effective if data quality can be guaranteed.

Many evidences exist that data quality problems affect organizations and result in different types of impact, including substantial social and economical impacts [9]. Data quality problems include, among others, missing attribute values, incorrect attribute values, and different representations of the same data, such as dates represented in different formats. Data quality problems typically arise from human data entry error, data inconsistencies existing between integrated data sets, and noise interference in sensor readings[5]. Data profiling may help revealing such errors and inconsistencies.

Data profiling encompasses a vast array of methods to examine data sets and produce metadata, i.e., data that provides information about data. Typical metadata includes the number of records in a data set, the number of null and unique values of an attribute, and basic column statistics. Other metadata, typically more difficult to obtain, includes outliers and dependencies in the data. Data profiling produces critical insights into data that companies can then leverage to their advantage. Use cases for data profiling include data exploration, data cleansing, and also data integration.

Data integration is the problem of combining data residing at different sources and providing the user with a unified view of those data [10]. In data integration the data sets to be integrated are often unfamiliar, and the integration expert typically wants to know more details about the data sets before integrating them. Those details may be obtained by profiling the data sets.

PDI1 is a well-known data integration tool with an open source version, open to additions by the community. It allows users to load data from multiple sources, transform it, and load the transformed data to multiple sources. Data flows in PDI are built using steps, joined by hops, which pass data from one step to the next. A transformation in PDI is essentially one of those data flows. A simple example of a PDI transformation is illustrated in Figure 1.

![Figure 1: A PDI transformation that reads data from a CSV file, filters the data, sorts it, and loads it to a relational database table.](image)

functionalties their clients wish would be available (natively) on PDI. One of the most pressing needs of clients is outlier detection. The main objective of the present work is to add the outlier detection functionality to PDI. This has been done by designing and developing a new PDI step. The new step, appropriately named Outlier Detection, offers four well-known outlier detection algorithms from the literature, and a simple outlier ensemble, for performing unsupervised outlier detection on multivariate data, i.e., data in which records have several attributes (variables). The Outlier Detection step is accompanied by two data visualizations - Scatterplot and Parallel Coordinates -, since these data visualizations can be a valuable tool for assessing the quality of outlier detection results. The following functional requirements have been identified:

- The step should support outlier detection on multivariate data, with both numerical and categorical attributes;
- Users without background on outlier detection should be able to understand the concepts used in the step without spending time on prior training.

Usability was identified as a quality requirement:

- It should be easy for users to select attributes from the data given as input to the step, configure the available algorithms in the step, and analyze the results returned by the step;
- To assess compliance with the usability requirement, we have conducted tests with users.

This document is organized as follows. In Section 2 we describe the functionality and implementation of the Outlier Detection step. The same is done for the companion data visualizations in Section 3. Section 4 details the methodology and results of the usability tests, and Section 5 provides a conclusion to this document.

# 2 OUTLIER DETECTION STEP

This section describes the functionality and implementation of the Outlier Detection step. The main goal of the step was to integrate unsupervised outlier detection into PDI.

## 2.1 Functionality

The Outlier Detection step offers four outlier detection algorithms and an outlier ensemble for performing unsupervised outlier detection in multivariate data, both numerical and categorical. More than one algorithm can be selected for each step execution. If this is the case, users may also choose to run the outlier ensemble.

### 2.1.1 Input Data

The input data to the Outlier Detection step is received from other steps in a transformation. Users may select any number of attributes (more commonly referred to as fields in PDI) from the input data. The step supports PDI types Integer and Number, corresponding to Java types Long and Double, respectively. The only categorical data type supported is PDI type String, corresponding to the Java type of the same name. Attributes in the input data that are not of the supported data types are not listed for selection by the users. The step handles categorical attributes by encoding their values as numerical values. For that purpose, two encodings are available: *Inverse Document Frequency (IDF)* and *1-of-n*. These are common encodings and were chosen because they do not require any parameterization by the users.

### 2.1.2 Step Output

The output of the Outlier Detection step is a table that consists of:

- the input fields that were selected for the detection;
- the outlier labels (and optionally the outlier scores) for each selected algorithm;
- the outlier labels (and optionally the outlier scores) for the outlier ensemble, if selected;

The outlier scores for the selected algorithms are optional. In case the outlier ensemble is selected, there is an option for omitting the individual algorithm results from the output, showing only the ensemble results.

The outlier scores assigned by each algorithm are normalized and regularized [7]. This means that all scores are in the interval [0, 1]; inliers will have scores close to 0, while outliers will have scores close to 1. Having normalized and regularized scores not only allows us to combine them in an outlier ensemble, but also facilitates the comparison between methods and the interpretability of the results. We find this very beneficial for users without background on outlier detection, since they will not have to learn how to interpret the scores of the different algorithms, which may differ widely in their scale, their range, and their meaning.

### 2.1.3 Algorithms

The Outlier Detection step includes four algorithms from four different techniques:

1. **DB(p, d)-Outliers** [6] is a distance-based algorithm; an object is a DB(p,d)-Outlier if at least a fraction $p$ of the objects in the data set lies greater than distance $d$ from the object. In addition to parameters $p$ and $d$, a distance function, such as the Euclidean distance, is also required to calculate distances between data points. Although this is originally a labeling algorithm, some authors transform it into a scoring algorithm by eliminating parameter $p$ and keeping only parameter $d$ [12]. In this work we use the scoring version.

2. **LOF (Local Outlier Factor)** [1] is a density-based algorithm; each object in the data set will be awarded a LOF score, which takes in consideration not only the density of the object itself, but also the densities of its nearest neighbors. An object lying in a dense region will have a local density similar to that of its neighbors, whereas an object not lying in a dense region will have a local density lower than that of its neighbors. The former will be considered an inlier, while the latter will be considered an outlier. In addition to a distance function, the algorithm takes as parameter the number of nearest neighbors to consider (originally named $MinPts$).

3. **DBSCAN (Density Based Spatial Clustering of Applications with Noise)** [3] is a clustering algorithm in which an object belongs to a cluster if the neighborhood of a given radius of that object contains at least a minimum number of objects. Objects that do not belong to any cluster are considered noise. The algorithm takes as parameters a radius $\epsilon$, the minimum number of objects to form a cluster ($MinPts$), and a distance function. In outlier detection we may consider noise objects as outliers, while objects belonging to a cluster are considered inliers. DBSCAN is a labeling algorithm, but we can obtain outlier scores from outlier labels by giving a score of 0 to inliers and a score of 1 to outliers.
ABOD (Angle-Based Outlier Detection) [8] compares the angles between pairs of distance vectors to other objects to discern between objects similar to other objects and outliers. If most other objects are located in similar directions, the angles to most pairs of objects will be small and the object will be considered an outlier. On the contrary, if many other objects are located in varying directions, the angles will differ largely and the object will be considered an inlier. ABOD is an alternative to algorithms that uses distances, and is particularly useful for high-dimensionality data sets. A major advantage of ABOD is that users are not required to specify any parameters. We include an option to run an approximate version of this algorithm, known as FastABOD, which only considers the $k$ nearest neighbors of an object $o$, instead of the whole data set.

We decided to mostly include distance-based algorithms since the notion of distance is something most users are familiarized with. We think it would be more difficult for most users to understand parameters of other techniques that require knowledge of more advanced mathematical concepts. We made an exception to this rule by including ABOD.

2.1.4 Outlier Ensemble. Outlier ensembles are used to combine algorithm results in the expectation that the combined results will be more robust. For this reason, we decided to include an outlier ensemble in the Outlier Detection step. The outlier ensemble available in the step performs a weighted arithmetic mean of the outlier scores of the selected algorithms. It requires that two or more algorithms have been selected.

2.2 Architecture

A PDI step performs a specific data processing task on a stream of data rows. Each data row is composed of data and metadata, such as the data type of a field. The life cycle of a PDI step is depicted in Figure 2. The life cycle consists of three stages: Initialization, Row Processing, and Cleanup. The Initialization stage starts when a transformation is preparing to start execution. During initialization we can perform one-time initialization tasks, such as opening files or establishing database connections. Once the transformation starts execution, the step enters the Row Processing stage, where it processes every row until no more rows are left in the input. Once the transformation is complete, the Cleanup stage starts, so that allocated resources can be deallocated.

Figure 2: The 3 stages of a PDI step life cycle: Initialization, Row Processing, and Cleanup.

In Figure 3 we depict the architecture of a PDI step. Within the Step module we represent the four interfaces that every step must implement:

- The class implementing the StepInterface will contain result sets, temporary data, database connections etc;
- The class implementing the StepMetaInterface will hold and serialize step settings, and report changes made to the row stream, e.g., a step may have added or removed fields from the row;
- The class implementing the StepDialogInterface will be responsible for the step dialog.

The Step module contains a step (BaseStep) which already implements these interfaces and that new steps can extend.

Figure 3: Architecture of a PDI step. Represented only are the 4 interfaces that every PDI step must implement.

The Outlier Detection step is composed of two main modules. The Step module contains the four classes that implement the four interfaces required to create a step: OutlierDetectionStep, OutlierDetectionStepData, OutlierDetectionStepMeta, and OutlierDetectionStepDialog. These classes will use the classes in the Outlier Detection module, which contains all the modules that are necessary to handle with data and perform the actual detection of outliers. The Outlier Detection module is depicted in Figure 4.

Figure 4: Outlier Detection module.

The Field Meta module holds metadata such as the name and data type of each field selected for detection, as well as the selected categorical field encodings for categorical fields. The Algorithm Meta module holds algorithm metadata such as the algorithm parameters, their threshold value, and whether the algorithm is selected for running. The Ensemble Meta module is similar to the
Algorithm Meta module, but instead holds ensemble parameter values, such as which algorithms are selected and what are their weights. The Categorical Field Encoding module contains the several categorical field encodings to transform categorical values into numerical values. The Row Storage module holds data rows in their original format, an Object array (Object[]). The Data module holds data rows in a more convenient format for processing, such as Double vectors. The Distance Function module contains the distance functions that the Algorithm module uses to calculate distances between objects. The Result module holds the output of algorithms and ensembles, i.e., labels and scores for each data record. The Algorithm module contains the algorithms used for outlier detection. The Ensemble module contains ensembles for combining algorithm results.

All of these modules were designed for extension. They contain interfaces that define contracts for new classes, so they can be used as any of the existing classes is. In this way it should be easy to add, for example, new algorithms, new ensembles, and new distance functions.

2.3 Implementation

2.3.1 User Interface. The Outlier Detection step user interface (or dialog) was developed with Eclipse SWT\(^3\). SWT, or The Standard Widget Toolkit, is an open source widget toolkit for Java that offers access to the user interface facilities of the operating systems on which it is implemented. This is the standard widget toolkit in PDI. The interface was designed around the three main concepts of the step: data, algorithms, and the ensemble. To that end, we organized the dialog into three tabs: Fields, Algorithms, and Ensemble. Beside the step name, only the Output scores? option is not in a tab, because it applies to both the algorithms and the ensemble.

In the Fields tab users can select which input fields to use in the detection and which encoding to use for categorical fields. Figure 5 depicts the contents of this tab.

The Algorithms tab contains all the algorithms available in the step. We can see in Figure 6 that each algorithm has its own group with all the algorithm-related parameters. Users must check the Run algorithm? box if they wish to use the algorithm in the detection.

2.3.2 Domain Logic. In the Outlier Detection step the class responsible for the step life cycle is the OutlierDetection class. The OutlierDetection class is responsible for collecting all input data rows, converting them to a more suitable format for processing, performing the outlier detection on the data, and passing the results on to the next steps. The actual outlier detection is performed by the ELKI software library [12]. ELKI is an open source data mining software written in Java, focusing on unsupervised methods in

---

\(^3\)https://www.eclipse.org/swt/
cluster analysis and outlier detection. It provides a large collection of outlier detection algorithms from several techniques.

Input data rows arrive as Object arrays (Object[]) and are stored in a RowStorage object, which stores input data rows in their original format. They are then converted from RowStorage format to Database format. The Database class is part of the Data module and stores input data rows in DoubleVector objects. The DoubleVector class is also part of the Data module and internally stores input data rows in ELKI DoubleVector format. Storing the data in an external library format avoids unnecessary conversions when passing the data to the external library algorithms. Storing the data in RowStorage format also avoids unnecessary conversions later, since the input data will be added to the output of the step in their original format. We use more memory to save CPU time. When converting data from RowStorage format to Database format we also perform the encoding of the categorical fields. This saves another pass from the whole data just to perform the encoding.

When the data conversion is done, we may pass the data on to each one of the selected algorithms. The algorithms have their own classes in the Algorithm module: DBOutliers, LOF, DBSCAN and ABOD. These classes are basically adapter classes for the classes in the external library. When the algorithm’s run method is called, it will call the run method of the same algorithm in the external library. The run method returns a Result object containing the scores and labels for that algorithm. If the outlier ensemble was selected, the Result objects will serve as input to the outlier ensemble. The outlier ensemble also returns a Result object. All the collected results are then converted to RowStorage and are concatenated to the input fields, to serve as output of the step.

3 COMPANION DATA VISUALIZATIONS

In this section we analyze the companion data visualizations to the Outlier Detection step.

3.1 Functionality

The main goal of the companion data visualizations is to provide a convenient way for visualizing the results of the Outlier Detection step, so that users can easily assess the quality of the outlier detection results. As we saw in Section 2.1.2, the output of the Outlier Detection step is a table that always consists of the input fields selected for detection and the outlier labels for each one of the selected algorithms (and possibly the ensemble); the outlier scores of the selected algorithms and the ensemble are optional. The companion data visualizations allow users to select an outlier label from the Outlier Detection step results and visualize the label value assigned to each data record. These visualizations were specifically included for supporting the Outlier Detection step and expect that an outlier label is present in the data source.

The Data Exploration Tool (DET) enables users to easily access visualizations of step data. A set of stock visualizations is included covering the most common chart types. Visualizations in DET adopt an Model-View-Controller (MVC) pattern. The Model identifies the visualization and defines it in terms of its data requirements, such as its visual degrees of freedom (spatial position, color, size, etc.) and any major options that affect its rendering. The visual degrees of freedom in DET are known as visual roles. A visual role specifies, for example, the number of fields that can be mapped to it and what are their data types. A View implements the actual rendering using the chosen technologies (HTML, CSS, Javascript, SVG, etc.) and handles user interaction by dispatching actions on the associated model, such as drilling-down and showing tooltips.

The companion data visualizations are Scatterplot and Parallel Coordinates. Scatterplot can be seen in Figure 8. This visualization has three visual roles: Horizontal Axis, Vertical Axis, and Outlier Label. An outlier label is encoded as color; inliers are shown in green, while outliers are shown in red. The Parallel Coordinates can be seen in Figure 9. This visualization has two visual roles: Axes, and Outlier Label. An outlier label is encoded as color; inliers are shown in green, while outliers are shown in red.

Figure 8: The Scatterplot companion data visualization.

Figure 9: The Parallel Coordinates companion data visualization.

User interaction is similar for both visualizations. When a user chooses a visualization, the visual roles are automatically filled in; the first outlier label found in the data source is mapped to the Outlier Label role and the input fields selected in the Outlier Detection step are mapped to the remaining roles, i.e., Horizontal Axis and Vertical Axis on Scatterplot, and Axes on Parallel Coordinates. If multiple outlier labels are present in the data source, as is the case in Figure 9, users can drag-and-drop any of them in the Outlier Label role. This allows users to visualize results for all chosen algorithms on the same visualization, although not simultaneously. We chose to restrict users from editing the remaining visual roles because all input fields present in the data source were used in the
detection, i.e., if a user removed a field from one of those roles, the outlier results would still refer to the set of all input fields, not the current set of fields. A companion data visualization is only made available to the users if their data requirements are satisfied, i.e., the number and data type of input fields match the ones required by the visualization. This type of restrictions and automatic mapping of fields to visual roles is a new use case for the application. Users typically have to drag-and-drop fields on visual roles and are allowed to explore the data without restrictions.

3.2 Implementation
The Model and View are specified with the Pentaho Visualization API\(^4\), which provides a unified way to visualize data across the Hitachi Vantara products. The actual rendering of the data was done using two different technologies. The Scatterplot companion data visualization was based on the Scatterplot visualization already available in DET. This visualization uses the Community Charts Component (CCC)\(^5\) charting library, which is built on top of Protovis\(^6\). The Parallel Coordinates companion data visualization uses the D3.js\(^7\) library.

3.3 Limitations
To automatically fill in the visual roles of the companion data visualizations we need a way to distinguish outlier labels - and outlier scores, which may be used in the tooltips - from the input data fields used in the detection. Since the data source does not contain any metadata that allows us to make this distinction, we find outlier labels by simply comparing the names of the fields in the data source with the hard coded names we give to the outlier label columns in the output of the Outlier Detection step: 'DB-Outliers_Label', 'LOF_Label', 'DBSCAN_Label', and 'ABOD_Label'. We consider any fields in the data source whose name contains the suffix '_Label' to be outlier labels. The same methodology is applied for outlier scores; we consider any fields in the data source whose name contains the suffix '_Score' to be outlier scores. Input fields are then the remaining fields found in the data source. This is far from ideal, since it may be the case that the name of some input field used in the detection may contain the suffixes we are waiting for. If this is the case, an input field used in the detection would be wrongly labeled as an outlier label, or score, by the visualization model, leading to incorrect models.

4 USABILITY EVALUATION
Tests with users are important to measure the performance and satisfaction of typical users performing typical tasks in the system under evaluation [4]. Performance measures include, for example, the time spent performing the task, the number of errors committed while performing the task, the time spent using help and documentation, the number of tasks successfully completed, etc. Satisfaction typically involves interviewing the users or asking them to answer satisfaction questionnaires.

For validating the present work, we conducted both formative and summative usability tests. Formative usability tests were important for identifying some problems during the design phase. We concentrate on the summative usability tests, which evaluate the success of the final product.

In the formative tests a user suggested that the outlier ensemble could be automatically selected when a second algorithm is selected, so we decided to use A/B testing to test two slightly different versions of the step: in version A the ensemble is not automatically selected, while in version B it is. Version B required some modifications to the user interface, which are illustrated in Figure 10. In Figure 10a we can see that in this version the Run ensemble? box is in the Algorithms tab, since the ensemble can be automatically selected and if the box remained in the Ensemble tab, users might not realize that it was checked without having clicked on it. This means that in this version the Ensemble tab contains only the ensemble parameters, and thus was renamed Ensemble parameters, as we can see in Figure 10b.

4.1 Experimental Protocol
Twenty users participated in the tests. They were divided into two groups, A and B, to test Outlier Detection step versions A and B, respectively. We initiated the tests by asking users some questions regarding their age and gender, their familiarity with the concepts of Outlier and Outlier detection (domain knowledge), and their level of experience with PDI. Before performing the tasks users watched a presentation on essential concepts of outlier detection, and also a demonstration video of the Outlier Detection step and the companion data visualizations. Users were asked to perform three tasks, which were designed so that users would have to use the step to select both numerical and categorical fields, as well as all the algorithms available and the outlier ensemble. We also wanted the users to visualize more than one algorithm in the visualizations. During the tasks the observer collected three usability measures: (i) the number of errors committed while performing the task, (ii) the time spent performing the task, and (iii) the number of tasks successfully completed. Half the users performed the tasks in one order (1,2,3) and the other half in a different order (2,1,3), to guarantee that the results are independent from the order in which the tasks

\(^5\)https://community.hitachivantara.com/s/article/community-chart-components
\(^6\)http://mbostock.github.io/protovis/
\(^7\)https://d3js.org/
were performed. After performing the tasks users were asked some questions about their satisfaction with the system under evaluation.

For achieving this we used the System Usability Scale (SUS) [2] and additional questions specific to the step and the visualizations.

### 4.2 Group A Results

Of the 10 users in this group 9 are men and 1 is a woman. The average age is 34.9 years old. On average, users are not very familiar with the concepts of Outlier and Outlier Detection. The average level of experience with PDI is 3.7 out of 5.0.

The values for all collected measures of group A users are shown in Table 1. Overall, 66.7% of the users completed the tasks with success. All users in this group completed at least task 1 and 2 users completed all tasks. Task 3 was largely responsible for the lower percentages.

<table>
<thead>
<tr>
<th>ID</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task Completion (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Errors</td>
<td>Time</td>
<td>Errors</td>
<td>Time</td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>03:10</td>
<td>0</td>
<td>05:13</td>
</tr>
<tr>
<td>A2</td>
<td>0</td>
<td>01:49</td>
<td>0</td>
<td>03:10</td>
</tr>
<tr>
<td>A3</td>
<td>0</td>
<td>06:06</td>
<td>1</td>
<td>11:17</td>
</tr>
<tr>
<td>A4</td>
<td>0</td>
<td>03:09</td>
<td>0</td>
<td>07:06</td>
</tr>
<tr>
<td>A5</td>
<td>0</td>
<td>03:11</td>
<td>0</td>
<td>06:04</td>
</tr>
<tr>
<td>A6</td>
<td>0</td>
<td>02:56</td>
<td>0</td>
<td>05:19</td>
</tr>
<tr>
<td>A7</td>
<td>1</td>
<td>03:35</td>
<td>0</td>
<td>06:14</td>
</tr>
<tr>
<td>A8</td>
<td>0</td>
<td>02:46</td>
<td>0</td>
<td>07:02</td>
</tr>
<tr>
<td>A9</td>
<td>0</td>
<td>03:11</td>
<td>0</td>
<td>10:19</td>
</tr>
<tr>
<td>A10</td>
<td>0</td>
<td>03:18</td>
<td>1</td>
<td>10:59</td>
</tr>
<tr>
<td>Mean</td>
<td>0.10</td>
<td>03:18</td>
<td>0.20</td>
<td>07:11</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>0.32</td>
<td>01:05</td>
<td>0.42</td>
<td>02:47</td>
</tr>
<tr>
<td>Median</td>
<td>0.00</td>
<td>03:10</td>
<td>0.00</td>
<td>06:38</td>
</tr>
</tbody>
</table>

Table 1: Usability measures - Group A.

We analyze the SUS scores with the help of Figure 11. The median score is 73.8, with a low variability. One user is marked as an outlier, with a score of 100; this user completed 2 out of 3 tasks without committing errors.

![Figure 11: SUS scores - Group A.](image)

Two users in this group are color blind, and had difficulty with the color scheme used in the visualizations.

### 4.3 Group B Results

Similarly to group A, of the 10 users in this group 9 are men and 1 is a woman. The average age is 37 years old; users of his group are, on average, 2.1 years older than users of group A. On average, users are not very familiar with the concepts of Outlier and Outlier Detection, but are slightly more familiar than group A users. The average level of experience with PDI is 3.4, slightly less than group A users.

The values for all collected measures of group B users are shown in Table 2.

<table>
<thead>
<tr>
<th>ID</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task Completion (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Errors</td>
<td>Time</td>
<td>Errors</td>
<td>Time</td>
</tr>
<tr>
<td>B1</td>
<td>0</td>
<td>04:03</td>
<td>0</td>
<td>06:34</td>
</tr>
<tr>
<td>B2</td>
<td>0</td>
<td>02:41</td>
<td>1</td>
<td>07:02</td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
<td>03:01</td>
<td>0</td>
<td>04:17</td>
</tr>
<tr>
<td>B4</td>
<td>1</td>
<td>01:22</td>
<td>0</td>
<td>04:22</td>
</tr>
<tr>
<td>B5</td>
<td>0</td>
<td>03:16</td>
<td>0</td>
<td>06:07</td>
</tr>
<tr>
<td>B6</td>
<td>0</td>
<td>02:25</td>
<td>0</td>
<td>04:17</td>
</tr>
<tr>
<td>B7</td>
<td>0</td>
<td>03:00</td>
<td>0</td>
<td>03:41</td>
</tr>
<tr>
<td>B8</td>
<td>0</td>
<td>02:41</td>
<td>0</td>
<td>08:24</td>
</tr>
<tr>
<td>B9</td>
<td>0</td>
<td>02:14</td>
<td>1</td>
<td>05:43</td>
</tr>
<tr>
<td>B10</td>
<td>0</td>
<td>03:06</td>
<td>0</td>
<td>06:25</td>
</tr>
<tr>
<td>Mean</td>
<td>0.10</td>
<td>02:47</td>
<td>0.20</td>
<td>05:42</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>0.32</td>
<td>00:42</td>
<td>0.42</td>
<td>01:30</td>
</tr>
<tr>
<td>Median</td>
<td>0.00</td>
<td>02:52</td>
<td>0.00</td>
<td>05:55</td>
</tr>
</tbody>
</table>

Table 2: Usability measures - Group B.

We start by analyzing the SUS scores with the help of Figure 12. The median score is 90, with a low variability. Only 1 user had the minimum score (72.5); this user was consistently fast and completed 2 tasks. Two users scored 95, the maximum score in this group.

![Figure 12: SUS scores - Group B.](image)

### 4.4 Discussion

As we saw in Section 4.1, three usability measures were collected: (i) the number of errors committed while performing the task, (ii) the time spent performing the task, and (iii) the number of tasks successfully completed.

The major source for errors was task 3, which involved using and parameterizing other steps than the Outlier Detection step. Users took more time to perform tasks 2 and 3 than task 1. In task 1 users...
were not required to use any visualization. Task 3 was planned for taking more time, so we consider the resulting times to be normal. On average, only 53.3% users completed all tasks. In task 3, as we saw before, users committed many errors related to other steps that were necessary to complete the task successfully. We also collected satisfaction measures after the users performed the tasks. One of them was the SUS score. Based on the high scores in both groups, especially in group B, we can conclude that users were mostly satisfied with the step and the visualizations. We also asked specific questions about the step and the visualizations. The question with the highest score was the question 'The step was easy to configure'. This is in line with what we intended for the step. Two users were color blind and responded 1 and 2 out of 5 to the question 'The color scheme was appropriate for the task'. We took this into consideration and will include a new color scheme in the release version of the visualizations.

Since we performed A/B testing with two different groups of users, we need to perform a statistical inference test, such as the Student’s t-Test, for obtaining reliable results for a population of users.

We define our null hypothesis $H_0$: "step version B is more usable than step version A". We performed a one-tailed test, with $\alpha = 0.05$, for all usability and satisfaction measures. The results for usability measures are shown in Table 3 and for satisfaction measures in Table 4. The font color is green if the hypothesis was not rejected and red if the hypothesis was rejected.

Table 3: Student’s t-Test values for the Usability Measures.

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task Completion (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>Time</td>
<td>Errors</td>
<td>Time</td>
</tr>
<tr>
<td>p-value of Student’s t-Test</td>
<td>0.500</td>
<td>0.112</td>
<td>0.500</td>
</tr>
</tbody>
</table>

For the number of errors committed and the time spent performing the task, we reject $H_0$ if $p-value < \alpha$. This is because we want this measures to be lower in step version B. For the percentage of tasks completed and satisfaction measures, we reject $H_0$ if $p-value > \alpha$. This is because higher percentages of tasks completed and scores is what we want for step version B We can see that our hypothesis is not rejected for the number of errors and the time spent performing the task - for all three tasks -, as well as for the SUS score. Our hypothesis is rejected for the number of tasks successfully completed and satisfaction measures D, E, F, G, H, and I. We do not consider the rejection of the hypothesis by the number of tasks successfully completed to be important since we saw that the majority of users did not complete task 3 due to PDI steps other than the Outlier Detection step.

Table 4: Student’s t-Test values for the Satisfaction Measures.

<table>
<thead>
<tr>
<th>SUS Scores</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value of Student’s t-Test</td>
<td>0.008</td>
<td>0.052</td>
<td>0.149</td>
<td>0.283</td>
<td>0.500</td>
<td>0.194</td>
</tr>
</tbody>
</table>

Although version B had more success in the usability measures and the SUS score, we decided to use version A as the final version of the Outlier Detection step. This is mainly for one reason, which we explain next. Group B users, which were testing the version that automatically selects the outlier ensemble upon selection of a second algorithm, could have unchecked the Run ensemble? box in task 2, in which the ensemble was not required. Only 1 user reminded to uncheck the Run ensemble? box. This is an indicator that users did not remember that the outlier ensemble was not necessary for this task, or did not realize the outlier ensemble was automatically selected. Leaving the Run ensemble? box checked in this case means that we are going to waste memory and computation for obtaining results we are not interest in. We could not choose version B over version A as it is, for this reason. Version B would need improvements in regard to this feature.

5 CONCLUSIONS

Data is increasingly valued by companies looking to take advantage of data analytics methods. The efficiency of such methods largely depends on the quality of the data. Several problems, such as the presence of outliers, have a negative impact on the results of data analytics methods. Data profiling can help in the discovery of such data quality problems. One of the fields that largely benefits from using data profiling methods is data integration, in which the data sets to be integrated are often unfamiliar. PDI is a well-known data integration tool which currently lacks many data profiling functionalities, in particular, outlier detection.

In this document we describe how we integrated outlier detection into PDI, by developing a PDI step and a set of companion data visualizations for the DET. The Outlier Detection step offers four well-know outlier detection algorithms from the literature, and a simple outlier ensemble, for performing unsupervised outlier detection on multivariate data, supporting both numerical and categorical data. Our approach was essentially to develop the step architecture and an Outlier Detection module, which delegates the actual detection to an external framework. The companion data visualizations - Scatterplot and Parallel Coordinates - , use the output from the Outlier Detection step for enabling users to assess the quality of the outlier detection results.

We conducted usability tests to validate the Outlier Detection step and the companion data visualizations. The usability measures and the user responses to the satisfaction questionnaires show us that users were mostly satisfied with the Outlier Detection step and had no major difficulties in configuring it. However, users had difficulty in completing a task that integrated our work within a typical PDI workflow. Overall, we think that users will benefit with the addition of the Outlier Detection step and the companion data visualizations to PDI.

5.1 Limitations and Future Work

The Outlier Detection step has several aspects that could be improved. Currently the step only allows one parameterization for each algorithm; if users want to run the same algorithm with different parameterizations, they need to create several instances of the step. However, the results obtained from the different parameterizations of the algorithm cannot be combined into an outlier ensemble, since the ensemble is integrated in the step. We could either detach the ensemble from the Outlier Detection step by creating an Outlier Ensemble step, or change the Outlier Detection step
in order to allow different parameterizations of the same algorithm in a single instance.

Another limitation surfaces with the increasing dimensionality of the data sets. High-dimensional data becomes very sparse, and traditional data structures, such as arrays, will waste memory representing null values. The Outlier Detection step currently makes use of Java arrays and lists to hold the data in memory. All attributes in an incoming data row are stored, whether they are null or not, thus wasting memory in case the data is sparse. When the data is sparse, we could replace those data structures with data structures similar to Dataframes in the Python library pandas. pandas Dataframes are dictionary-like containers for data rows, holding (attribute, value) pairs. An entry is only added to the dictionary if the value for the attribute is not null - or zero, in the case of numerical data -, thus saving memory in case the data is sparse.

During the usability tests, we noticed that users had difficulties with the companion data visualizations. The typical use case for DET involves exploring the data by adding and removing fields from any visual roles in a visualization, while our use case is slightly different: input fields are automatically assigned to their visual roles and users are only allowed to add and remove outlier labels from the Outlier Label role. Some users found this confusing. This automatic assignment of input fields to predetermined visual roles also does not allow users to change the encoding for an attribute, e.g., changing, in a scatterplot, the encoding for input field A from x-position to y-position and the encoding for input field B from y-position to x-position. We have to rethink our use case so that users have the freedom of reassigning input fields to the visual roles of their choice, while guaranteeing that all input fields are assigned to visual roles.

Future work for the Outlier Detection step may include support for more data types (booleans, dates), adding more algorithms (possibly of different techniques), and adding other types of outlier ensembles.

Both the Outlier Detection step and the companion data visualizations are open source software, and thus will be open to contributions from other developers and suggestions from the community. The Outlier Detection step will also be added to the Pentaho Marketplace, so that users can easily download and integrate the step into PDI.

REFERENCES