Article

Data Preprocessing Method and API for Mining Processes from Cloud-Based Application Event Logs

Najah Mary El-Gharib and Daniel Amyot

University of Ottawa, Ottawa, Canada
* Correspondence: nelgh031@uottawa.ca

Abstract: **Background:** Process mining (PM) exploits event logs to obtain meaningful information about the processes that produced them. As the number of applications developed on cloud infrastructures is increasing, it becomes important to study and discover their underlying processes. However, many current PM technologies face challenges in dealing with complex and large event logs from cloud applications, especially when they have little structure (e.g., clickstreams). **Methods:** Using Design Science Research, this paper introduces a new method, called *Cloud Pattern API – Process Mining (CPA-PM)*, that enables discovering and analyzing cloud-based application processes using PM in a way that addresses many of these challenges. CPA-PM exploits a new application programming interface (API), with an R implementation, for creating repeatable scripts that preprocess event logs collected from such applications. **Results**: Applying CPA-PM to a case with real and evolving event logs related to the trial process of a Software-as-a-Service cloud application led to useful analyses and insights, with reusable scripts. **Conclusion**: CPA-PM helps producing executable scripts for filtering event logs from clickstream and cloud-based applications, where the scripts can be used in pipelines while minimizing the need for error-prone and time-consuming manual filtering.

Keywords: API; clickstream; cloud applications; process mining; scripting

1. Introduction

Organizations use information systems to support an increasing number of business processes, where a process is a set of related activities that happen over time in order to achieve a goal [15]. Many such systems record what happens in running processes as *event logs* that contain information such as process instances, activities, timestamps, and resources. Event logs form a large amount of data ready to be exploited through *process mining* (PM), which aims to discover processes to improve process understanding, compliance, and quality aspects based on evidence. PM is often needed because concrete processes are often more complex than their idealized process definitions or models.

The interest in information systems exploiting cloud computing is also rapidly growing in the industry and research communities. The number of applications developed in a cloud environment is increasing because of benefits that a cloud infrastructure offers, especially regarding scalability, availability, and maintainability.

Clickstream data is used to represent the electronic record of a user's behavior on Web-based information systems and cloud applications. This data shows the path a user takes while navigating the website or application. This path reflects the different choices that the user takes while navigating. Given that clickstream event logs might include a record of every page visited by the user, they usually form large amounts of behavioral data. Such logs provide opportunities for a detailed look at the decision-making process of users. When applying process mining techniques on such data, process maps can be generated that represent the user actions on web/cloud applications. These process maps give detailed information about users' navigation habits. It is important to visualize and understand user actions while interacting with a cloud application so that the latter be engineered to better satisfy user needs and ensure that users come back to the application.

Unfortunately, the complexity of clickstream data and other event logs from cloud-based applications is typically too difficult to handle by process mining tools, out of the box. A research challenge here includes how to reduce, in a repeatable way, log complexity caused by numerous cloud event attributes, action orderings (e.g., clicks and refreshes), and low-level events. Our research objective is to develop and assess a method to preprocess event logs collected from cloud-based applications to discover user actions and processes. A secondary objective is to examine how process mining can be applied on clickstream data collected from Software-as-a-Service (SaaS) applications and discover users' behavioral characteristics. A suitable method should provide high automation (e.g., by being scriptable), high abstraction, high scalability, and low effort to accommodate changes frequently occurring in cloud-based logging infrastructures. The focus here is also on *advanced preprocessing* of event logs, where the preprocessed logs can then be fed to many existing process mining tools for visualization and analysis.

This research was conducted following Design Science Research in Information Systems [9]. This methodology consists of five iterative steps, 1) awareness of problem, 2) suggestion, 3) development, 4) evaluation, and 5) conclusion, that are illustrated throughout the different sections in this paper.

The main research contribution of this paper is a *Cloud Pattern API – Process Mining (CPA-PM)* method, accompanied by a scriptable application programming interface (API) with an R implementation, to preprocess little-structured cloud-based event logs and enable simpler and repeatable process discovery.

In this paper, Section 2 presents the related work, including important challenges, while Section 3 highlights the research materials and methods. Section 4 introduces the CPA-PM method and Section 5 presents its API. Section 6 shows the application of the method to a SaaS application case study, while Section 7 discusses analysis results and threats to validity. Finally, Section 8 concludes.

2. Related Work and Challenges

Important motivations for applying process mining techniques on a dataset include discovering and generating understandable process models (or *maps*). When analyzing processes with many different cases, much noise, and a high diversity of behaviors, which are typical of cloud-based applications, the generated models are often confusing and difficult to understand. Bose and van der Aalst [2] call such process models *spaghetti models* and explored the iterative use of pattern abstraction to produce simpler and hierarchical models that eliminate irrelevant paths and reduce complexity.

A recent survey on user behavior analysis from clickstream data highlights different approaches based on clustering, classification, and association, but no PM approaches specifically [10]. Our own systematic literature review [7] however assessed 27 peer-reviewed papers related to the application of PM technologies to cloud processes as well as clickstream datasets, including applications to customer journey process modeling [13]. This review shows that the area of cross-organizational process mining (initially explored by van der Aalst [14][17]), common in a cloud-based context, is still challenging. The review also highlights specific challenges faced when preprocessing event logs collected from cloud-based applications, including clickstream data. In particular [7]:

1. Cloud-based logs are generated from different monitoring systems, often from different organizations, leading to ID/case alignment issues especially in the presence of privacy constraints.
2. Cloud-based logs often contain hundreds of attributes that are sparsely populated, which require analysts to select/merge attributes at pre-processing time.
3. Cloud-based processes typically evolve more rapidly than conventional processes, especially in a DevOps context with daily deployments. Different

types of events/attributes can then start/stop appearing in the logs at a very high rate.

4. For real-time processing, the granularity of cloud-based logs is often much finer (hundreds of clicks/events per second) than in conventional processes, which adds to the processing complexity.
5. Given that users are usually free to click anywhere on a Web application, and that the proportion of incomplete process instances is high in cloud-based application, the readability of mined process models is low (and spaghetti-like).
6. Dealing with noisy cloud-based logs, where too many events are tracked, adds to the complexity of models.
7. Many cloud-based events have generic names and must be split according to some of their attributes into different events.
8. The monitoring and trace management tools in cloud environments frequently evolve, which in turn results in changes in the nature of the collected logs.
9. Events whose ordering or even presence in not important (for instance, `click-refresh-click-refresh`) often pollute cloud-based logs and should often be aggregated (e.g., to `click-click-refresh` or even just `click-refresh`) in order to simplify the resulting processes.

In the process mining literature, simplification of event logs is often done with a combination of abstraction [5], event/trace filtering (e.g., based on frequencies or attributes) [1], and trace clustering (e.g., K-means clustering) [4]. While filtering techniques alone work well with structured processes, they have problems discovering less structured ones, especially in a cloud-based context [5].

Preprocessing is required to reduce and simplify event logs [11][12]. Preprocessing such logs before applying process mining techniques usually takes up between 40% and 80% of the total effort for most people [18]. Preprocessing is so important that the IEEE Task Force on Process Mining is currently surveying the community regarding challenges and solutions for improving event data preparation for process mining (see https://bit.ly/TFPM-survey2001). Preprocessing is even more needed in a cloud application context. Generic technologies such as *Extract, Transform, and Load* (ETL) certainly have a role to play [3], especially regarding challenge #1 (see Fig. 1), but they are insufficient for the other challenges. There are several process mining tools that support different types of filtering for the event logs, though such tools do not provide much flexibility in terms of the filtering that the analyst can do. The filtering techniques are also decided based on the type of the data and the analysis that has to be done.

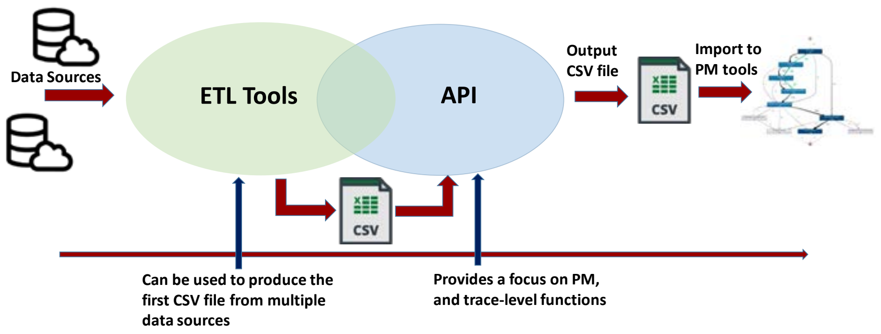
Data Sources
ETL Tools
API
Output CSV file
CSV
Import to PM tools
CSV
Can be used to produce the first CSV file from multiple data sources
Provides a focus on PM, and trace-level functions

Fig. 1. Relationship between ETL and a desirable API for preprocessing.

There is also a need to *automate* this preprocessing, ideally using repeatable scripts exploiting a domain-specific API that focuses on solving some of the PM-related challenges for cloud and clickstream applications. Scripting filters helps minimize repetitive and often error-prone manual labor. Yet, scripting is not even mentioned in a recent review of event log preprocessing for process mining [11].

3. Materials and Method

Our research method follows Hevner's Design Science Research (DSR) in Information Systems (IS) [9], both in terms of research relevance and research rigor. The design science paradigm is based on iterative problem solving. It aims to create innovations that define the ideas, practices, and products through the analysis, design, implementation where the use of IS can be effectively and efficiently accomplished. In this paradigm, an *artifact* should be produced taking into consideration the exiting knowledge in the discipline and should be produced creatively to solve real-world problems. Artefacts are usually constructs, models, methods, and instantiations. We have instantiated DSR's five steps to our research context.

1) Awareness of problem: In this phase, the researcher defines the specific research problem and justifies the value of this research and the proposed solution. We performed a systematic literature review about the challenges in the area of process mining for cloud-based applications [7], whose conclusions are summarized in the previous chapter. Input from key informants of a SaaS application provider was also informally taken into consideration. This also led to our main research question: What is a suitable method to discover and analyze cloud-based application processes?

2) Suggestion: In this phase, from the defined problem and knowledge of what is possible and feasible, we defined the objectives for the solution. A tentative solution was developed that was presented to a SaaS company that provided event logs from their systems. In this phase, a new method (our main artifact) and its different steps were defined, and the solution was illustrated with a manual example. This new method is the topic of Section 4.

3) Development: In this phase, the researcher creates an artifact, which is the solution to the defined problem. The artefact developed here is a method that exploits an API (with an implementation). The development went through several iterations by exploring available features in existing tools (mainly Disco, several ProM plug-ins, and data processing libraries) with open-source datasets and real event logs from the case study. The existing functions relevant to the pre-processing of cloud-based application logs can hence be replicated outside these tools (in the API, in order to support automation in one place, independently of the PM tools used for visualization and analysis) and supplemented by new useful functions in the API. An implementation of the API was done in the R programming language[1]. R was selected as this is a common data manipulation and analysis language, and it was also already used extensively by the SaaS company. This API is presented in Section 5.

4) Evaluation: In this step, the researcher observes and measures how well the artifacts support a solution to the problem. Additionally, the researcher communicates the problem and its importance. The evaluation method that was used here is an *application case study*, combined to a comparative analysis and a performance evaluation. A dataset provided by a SaaS company was used to evaluate the developed method for preprocessing event logs before importing the event logs to the PM tools. After preprocessing the event logs using the proposed method, the Disco and ProM PM tools were both used to build the process maps and visualize how processes are executed in reality. Several iterations are performed on the dataset in order to answer the questions about the business processes. Here, we used two iterations based on two versions of the event log (taken at different times) to assess the complexity of adapting scripts in the context where the structure of logs changes. The case study is presented in Section 6, whereas Section 7 presents the comparative analysis and a performance evaluation.

[1] https://www.r-project.org/about.html

5) Conclusion: This phase is the last step in the research process. Here, the results of the research are finalized, indicating the end of a research cycle. This step concludes the research projects and presents the results. A paper [7] and a thesis [6] were already produced, and this paper (accompanied with code and examples on GitHub) concludes this research.

4. Cloud Pattern API – Process Mining Method

Cloud Pattern API – Process Mining (CPA-PM), as the main artifact of this research paper, is a method for preprocessing event logs (including clickstream data) collected from cloud-based applications, that tackles challenges #2 to #9 from the previous section. Through an API, currently implemented in R, CPA-PM enables scripting for log manipulation and for replacing patterns before applying process mining techniques on the event logs. The CPA-PM method consists of three main steps (see Fig. 2) that enable preprocessing cloud-based application event logs, which usually include a large number of columns (i.e., event attributes) that need to be taken into consideration when aggregating rows in the dataset.

The input for this method is an event log dataset (in CSV) and the output is a cleaned log (CSV) that can be imported by process mining tools for building simpler process maps and enable further analysis. The single input CSV file can itself be a result of a prior extraction and processing from multiple data sources where, for example, conventional technologies (ETL) and their tools can be used. The steps of the CPA-PM method might go through several iterations to get answers for analysis questions.

Note that CPA-PM and its API offer tools to automate the simplification of event logs, but they do not guarantee that the resulting logs will preserve the right information to support understanding. This remains the responsibility of the analyst.

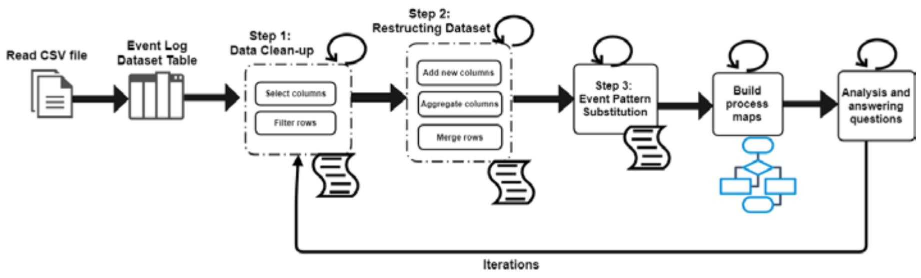

Fig. 2. Overview of the CPA-PM method and its three main iterative steps.

4.1. Clean-up Step

In this step, initial cleaning of the dataset is done. Seven sub-steps are included here, and the last two sub-steps (1.6 and 1.7) can be interleaved.

Step 1.1: Read the CSV log file, which must include the three main columns that are needed to apply process mining (case ID, event, and timestamp). It may also include other columns, for example resources and other properties.

Step 1.2: Clean column identifiers, e.g., by removing white spaces and dots from the column's names and ensuring their unicity. This returns a dataframe dataset with clean names that consist only of lowercase letters, numbers, and underscores.

Step 1.3: Ensure the time format of the timestamps reflects the granularity needed. For example, if the timestamp must include detailed information about the minutes, seconds, and milliseconds for when the event occurred, this may require merging separate date and time columns, or change the encoding of the date/time information.

Step 1.4: Ensure that the cells in a given column comply with their expected column type and transform or cast the column, if necessary, to the required data type.

Step 1.5: Select only the columns needed for the analysis questions. For clickstream data, the analyst should select a) the Case Identifier, Event, and Timestamp columns minimally required by PM techniques; b) the resource columns relevant to the analysis questions; and c) the attribute columns that refine events. Consider also excluding columns that are empty or that threaten re-identification.

Step 1.6: Filter the event log. Filtering is a popular preprocessing technique whose goal is to reduce the complexity or to focus on a specific part of the process during the analysis. This technique is usually performed multiple times, iteratively.

Step 1.7: Remove incomplete cases. An *incomplete case* is a truncated case/trace that is missing the start and/or the end event(s) in a process instance.

4.2. Dataset Restructuring Step

Restructuring for the event log is performed in such a way that process mining algorithms can be applied depending on the understanding or analysis questions that should be answered. These different techniques, some of which implementing several known event abstraction approaches [17], can be applied during this step.

- *Enriching the event log dataset:* event logs can be enriched with additional attributes, e.g., by adding external additional data collected or by combining two different datasets. This is however often done before the generation of the original CSV file (e.g., using ETL). Another way is by deriving or computing additional or *derived* events and data attributes based on the existing attributes in the data, resulting in new columns calculated from other columns. Such additions depend on the required analysis, e.g., for calculating the time difference between the occurrence of two events to reason about durations.
- *Aggregating events:* aggregating events into coarser-grained or more abstract events (e.g., along "is-a" or "part-of" relationships between events) can help reduce complexity and improve the structure of clickstream datasets, leading to better and simpler processes.
- *Aggregating rows*: this technique includes aggregating several rows into one row, to reduce complexity and (unnecessary) repetition of the same event. For example, let us consider that event *A* is repeated multiple times in a given case, but needs to be considered only once. Options here include keeping only the first occurrence of event *A*, keeping only its last occurrence, or merging all the *A* rows into one by applying maximum, minimum, average, and other such functions on the columns' attributes for the repeated event *A*. This will be illustrated in more detail in the API functions and their implementations.

4.3. Event Pattern Substitution Step

In this step, complex event patterns are substituted with a new event to reduce the complexity of the resulting process maps. If several events patterns are repeated and are known to occur together, then they can be substituted by a single, more abstract event. Defining, selecting, and substituting patterns in this step is done by domain knowledge experts. Several common patterns are defined as rewrite rules at this stage and can be used on the cloud-based application event log dataset. These and other patterns can be defined by the analyst depending on the analysis context.

For example, consider a common event in a cloud application case: a page *refresh*. A pattern could consist of a non-empty sequence of *refresh* events (*refresh*$^+$). As whether the page is refreshed once or many times without any other events in between, that sequence *refresh*$^+$ could be replaced by only *refresh* event. This would simplify many traces, and make them equivalent along the way, which would help simplify the resulting process model generated through process mining, without negatively affecting its usefulness. More complex patterns involving sequences of interleaving *click* and *refresh* events (*click-*

{click | refresh}-refresh*) could similarly be simplified by substituting them to a single *click-refresh* pair.

5. CPA-PM Application Programming Interface (API)

This section presents the API functions, mainly described in terms of addition, removal, or modification of rows, columns, and traces. As the purpose here is to simplify and shorten event logs, the API offers mainly removal and modification functions. Please note that in the following, an event log table with a heading row is also called a *dataframe*. Most of the functions (except `writeCSV`) return a new dataframe.

- **`readCSV`**`(String file)`: reads the CSV file into a dataframe that it creates.
- **writeCSV**(table DataSet, String file): writes the dataframe to a file.
- **`cleanHeaders`**`(table DataSet)`: cleans the dataset's headers of the columns from spaces (replaced with _) and other special characters (removed). This function only keeps lower case letters, numbers, and underscores (_).
- **`selectColumns`** `(table DataSet, string columnName, …)`: selects/keeps the list of columns needed for analysis from the dataset. Only the list of selected columns/attributes are included in the dataset.
- **deleteColumns** (table DataSet, string columnName, …): drops the listed columns from the dataset.
- **`filter`**`(table DataSet, condition Conditions)`: keeps records/rows based on the conditions specified. Only the rows where the condition is `true` are kept in the DataSet. Comparison and composite functions are supported with ==, >,<, >=, <=, &, |, and !.
- **`removeEventsLowFrequency`**`(table DataSet, String eventName, integer freq)`: removes the events from the dataset that have a frequency below the value *freq* (count integer) when grouping by *eventName*.
- **`deleteTraceLengthLessThan`**`(table DataSet, String groupID, integer num)`: removes the traces where the number of events is less than a specified value. *groupID* is the name of the variable to group traces by, and *num* is the minimum number of events.
- **`deleteTruncatedTracesStart`**`(table DataSet, String groupID, String eventColumn, String value)`: removes the traces/traces that do not start with the required event value. *groupID* is the name of the variable to group traces by, and *eventColumn* is the name of the event column. *value* is the value of the eventColumn that specifies the required start event name.
- **`deleteTruncatedTracesEnd`**`(table DataSet, String groupID, String eventColumn, String value)`: similar to the previous one, but removes the traces/cases that do not *end* with the required event.
- **`deleteTracesWithTimeLess`**`(table DataSet, String groupID, String timestampColumn, integer t)`: removes the traces/cases that ran for a total duration less than *t*. *timestampColumn* is the name of the timestamp column, and *t* is the value of the minimum total duration for each trace/case, in seconds.
- **`concatenateColumns`**`(table DataSet, String newColumn, String col1, String col2, …)`: concatenates two or more columns (*col1*, *col2*, …) in a dataframe into a *newColumn* that is added to the dataframe.
- **`arrangeRows`**`(table DataSet, String columnName, …)`: sorts the rows by *columnName*, in increasing order. *columnName* is a list of unquoted names.
- **`eventIsRepeated`**`(table DataSet, String groupIDCol, String eventCol, String newCol)`: creates a new column *newCol* that indicates whether the event has been repeated or not. The event is represented by the *eventCol* parameter. The new column will only include values 0 (when not repeated) or 1 (when repeated).

- **`deleteAllEvents`**`(table DataSet , String event, String eventName)`: deletes all records/rows with an *eventName* value in the *event* column.
- **`keepFirstEvent`**`(table DataSet, groupID, String eventAttribute, String eventName)`: keeps the first occurrence of an event (i.e., the first occurrence of several repeated events). *groupID* represents the column for grouping the records, *eventAttribute* the event attribute column, and *eventName* the event name whose first occurrence in each group we want to keep.
- **`keepLastEvent`**`(table DataSet, groupID, String eventAttribute, String eventName)`: similar to the previous one, but keeps the *last* occurrence of an event (i.e., the last occurrence of several repeated events).
- **`mergeRows`**`(table DataSet, list groupByVariables, list columnArguments)`: aggregates several rows into one row. *groupByVariables* represent the variables to group the dataframe by, and *columnArguments* represents the new column arguments based on aggregation functions. Each column of the aggregated rows requires one predefined aggregation operator from: mean, median, min, max, sum, first, last, n (count), of logical operators. String concatenation is supported too. Those aggregation operators are defined by the analyst.

As `mergeRows()` is the most complex function of this API, we are illustrating its application on a simple sequence of events extracted from a real event log. This function works by splitting the data into groups, applying some analysis to each group of data, and then combing the results. It works by aggregating several rows for each column. For instance, the first three rows of Table 1 target the same activity of the same case identifier. The analyst may decide to keep only one such event (as in the patterns described earlier). The issue here is to decide how to aggregate the information in the various columns.

Table 1 Simple event log example, with targeted events to be merged highlighted in blue

case_id	timestamp	activity	city	weekday	client	num_ items	product_ category	device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	3	B	Android
121	2018-08-01 00:04:23	Main home page	Ottawa	Monday	NA	4	B	Android
121	2018-08-01 00:04:25	Main home page	Ottawa	Monday	NA	6	D	Android
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	4	A	Android
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	4	N/A	Android

When aggregating the rows grouping by *case_id* and *activity* attributes, the analyst may decide to keep the *timestamp* and *weekday* of the first event; produce lists of unique *activity*, *city*, *client*, and *device* values for their respective columns; calculate the sum of the merged elements in the *num_items* column, and keep all the values of the *product_category* column (without uniqueness). The invocation of this API in R on the log named *EventLogs* is as follows, with the result shown in Table 2.

```
mergeRows(EventLogs, .(case_id, activity), summarise,

   timestamp = first(timestamp),

   activity = paste(unique(activity), collapse = ','),

   city = paste(unique(city), collapse=','),

   weekday = first(weekday),

   client = paste(unique(client), collapse=','),

   num_items = sum(num_items),

   product_category = paste((product_cateogry), collapse=’,’),
device = paste(unique(device), collapse=',')) -> EventLogs
```

Table 2 Event log after merging the first three rows of Table 1 into one row, in blue

case_id	timestamp	activity	city	weekday	client	num_ items	product_ category	device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	13	B,B,D	Android
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	4	A	Android
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	4	N/A	Android

This API contains basic functions to load/save CSV files and filter rows and columns, but also advanced ones that support the various steps of CPA-PM more directly and efficiently. For example, `deleteTruncatedTracesStart()` and `deleteTruncatedTracesEnd()` are essentially used to support Step 1.7 of the CPA-PM method (Section 4.1). These methods exclude from the event log the incomplete traces (i.e., all the events for one case) that either started too early or finished too late (or did not finish) according to the time frame where the log was produced, see Fig. 3.

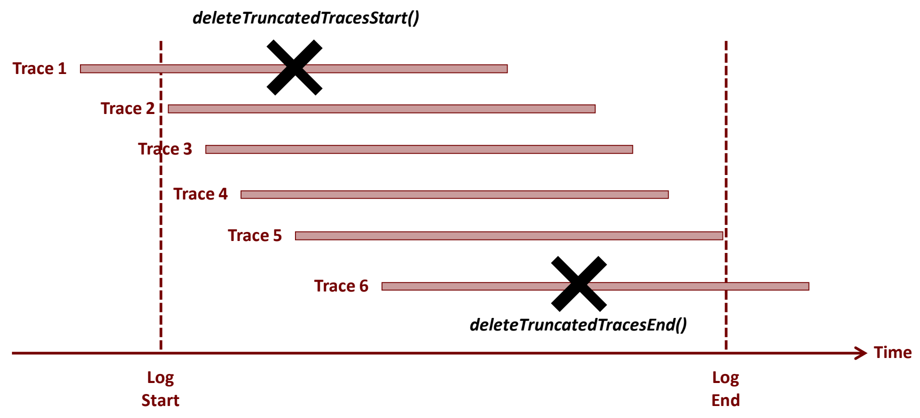

Fig. 3. Example of incomplete traces with missing start events (Trace 1) of end events (Trace 6)

This API addresses challenges #2 to #9 from the literature review. It is currently implemented in R, but a Python version is also being considered. More detail about the API and its usage is available in [6], and its R implementation is available online at https://github.com/NajaElgharib/CPA_PM.

6. SaaS Application Case Study

The CPA-PM method is applied to a real clickstream event log dataset in order to assess the effect of the CPA-PM method on the preprocessing of datasets collected from a SaaS application on the process mining results. The logs' events include user actions from when users start the trial version of the application until they either become real clients or give up along the way. CPA-PM is then applied to a second dataset extracted from the same log management tool for the same process taken several months later to understand and assess the impact on the initial script of the evolution of the processes and of the log monitoring infrastructure after a long time in a SaaS company.

6.1. Case Study Overview

The aim of the case study is to apply the CPA-PM method steps to preprocess the raw event log dataset in order to discover better structured process maps than without using preprocessing. The dataset was provided and anonymized by a SaaS company, but it still cannot be shared publicly for confidentiality reasons. The dataset was collected from a cloud-based data visualization application that allow users to create interactive dashboards. The logs' events include user actions from when users start the *trial* version of the application until they become real clients, or give up along the way.

6.2. Dataset

Data extraction is the first step in such PM project, which is then followed by data preprocessing. Data extraction involves getting the event log from the (possibly many) log management tools used by the SaaS company. The starting point in this case study is the event log dataset related to the trial version, provided as one CSV file after anonymization, and renaming of some of the events (to preserve the confidentiality).

The dataset includes three months (August 2018 to October 2018) of events, with 1,602,438 different records/rows and a very large number (152) of variables/columns. The dataset includes records from 4,462 different cities. From the case study, only 24 different variables are needed based on the process-oriented questions that we want to answer. These questions pertain, for example, to the situations under which the users stop using the trial version before becoming (paying) customers.

6.3. Case Study Planning

The main columns in the dataset that are required before applying PM techniques are the case identifier, the timestamp, and the activity.

1) **Case ID:** The Company ID attribute is chosen as the case ID, as it refers to cross-site identifier used to differentiate users. The Company ID does not change from the starting point of the process (user sign up for trial version) until the last activity within the time frame that is included in the dataset, even if the user uses the trial version of the application sporadically over several days.
2) **Timestamp:** each event occurs at a particular moment. The timestamp used here is in this format: YYYY-MM-DD hh:mm:ss.
3) **Activity**: from the event log dataset, the event attribute was selected as the activity. The event attribute represents the clicks and actions done by users while using the online application. Table 3 displays some event names (among 93) from the dataset, with their descriptions.
4) **Other attributes:** In addition to the main three attributes that are needed for PM analysis, the dataset includes additional columns that represent other attributes and properties that exist when an activity/action occurs. Here, those attributes include data source, data source number, help guide name, template name, account type, number of users, device type, browser, city, weekday, number of dashboards, dashboard type, and others. These attributes are considered as other necessary properties that will be used in the analysis.
5) **Process mined from the original event log:** The initial process map generated using the Disco tool [8] for the original event log is much too complex and unreadable to be presented here. With high numbers of variants with respect to the cases, such spaghetti process maps are generated but are not helpful during analysis. Preprocessing and restructuring of the dataset are required.

Table 3 Some event names with their descriptions

Event Name	Description
Add graph from library	User adds a graph from the library that provides several ones
Add new graph button	User adds a new button to the graph
Add template	User adds a template to build a graph
Build graph from library	User builds a new graph from existing ones in the library
Connect to data source	User connects to a data source to import data into a graph
Create dashboard	User creates a dashboard
Create data source	User creates a new data source to import their own datasets

6.4. Data Preparation and Preprocessing

CPA-PM is illustrated on the SaaS application dataset, starting with the *Clean-up* step. The following six points correspond to steps 1.1 to 1.6 in Section 4.1.

1) The first step involved importing the dataset (a CSV file) to RStudio, the R environment we used to execute the R functions from the CPA-PM API. The following step is executed in order to read the CSV dataset file:

EventLogs <-**readCSV**(file="full_DS.csv",header=TRUE,sep=",")

The EventLogs dataframe is created in RStudio. Then the dataset was explored to compute the total number of unique events that exist. The EventLogs dataset includes 93 unique events executed by users during the three-month period.

2) After importing the CSV file, the cleanHeaders() function was applied to make sure there is consistency with the column headers names:

cleanHeaders(EventLogs)-> EventLogs

3) In this step, the time format was check and it was already appropriate.

4) We ensured that the attributes needed for the analysis were available in the SaaS application dataset, with the right type.

5) In this step, the selectColumns() function was called to select the list of columns needed for the analysis, which are here. Here is a list of the columns that were selected for the analysis:

selectColumns(EventLogs, company_id, event, time, client, template, guide_name, guide_type, number_of_graphs, number_of_graphs_on_dashboard, graph_origin, source, kpi_count, template_name, account_type,graphs_owned, data_format, data_sources_owned, dashboard_template_name,
connector_backend, x_city, weekday) -> EventLogs

6) Then, some initial filtering was done. This filtering includes cleaning the dataset from some of the attributes that are not needed:

Remove the empty case identifiers:

EventLogs %>% **filter**(company_id != "") -> EventLogs

Remove the records where the client attribute value is *Phone App*, since for this analysis we are not interested in including the phone app users:

```
EventLogs %>% filter(client != "Phone App") -> EventLogs
```

Remove the records where the event is *View Dashboard*, because this is an event with a very high frequency and many variants as it is clicked many times by users, and it will not add useful information to the analysis:

EventLogs %>% **filter**(event != "View Dashboard") -> EventLogs

Other records with very low frequency (10 and less) and not needed in the analysis were removed:

removeEventsLowFrequency(EventLogs, company_id, 10) -> EventLogs

Sort the events in the dataset according to *company_id* and *time* variables:

```
arrangeRows(EventLogs, company_id, time) -> EventLogs
```

After doing the initial cleaning of the dataset EventLogs, the resulting dataset was imported to Disco to start the analysis. *Company_id* was selected as the case identifier, *event* as the activity variable, and *time* as the timestamp variable. According to Disco, the resulting EventLogs dataset includes 960,919 events over time (a 40% reduction from the initial dataset); 52,084 cases; 85 activities; and 13,848 different variants.

7) The removal of incomplete cases (Step 1.7 in Section 4.1) is an important data preparation step that needs to be done before starting the PM analysis. For example, truncated cases may appear to be faster than they really are and distort case durations. Removing incomplete cases can also help simplify the process map, because incomplete cases inflate the process map layout by adding spurious end or start points to the process map. In this case study, we removed all the traces that did not start with the *Trial Sign Up Completed* event:

deleteTruncatedTracesStart(EventLogs, company_id, event, "Trial Sign Up Completed") -> EventLogs

After applying this function, the dataset got simpler, with 816,125 different records and 15,190 unique cases. Note that deleting cases truncated at the end with the API's `deleteTruncatedTracesStart()` function is not performed here because we want to analyze the reasons why users abandon trials before buying the application, and so these incomplete process instances must be preserved.

Another type of filter based on very short traces (Step 1.5 in Section 4.1) can be considered. For example, there are 520 cases that have only one event "Trial Sign Up Completed", and 149 cases with only two events; "Trial Sign Up Completed" and "Help Guide Start". The cases (traces) with very few events per trace (1 to 2 events) are not interesting for the analysis since they do not provide much information about the processes (the users simply started the trial version of the cloud application but did not really use it). These traces should be removed from the dataset. In this case study, the traces that included only one or two events were deleted:

deleteTraceLengthLessThan(EventLogs, company_id, 2)
-> EventLogs

Simple loops involving one event (e.g., the "Help Guide Start" activity) were also not deemed interesting, and they were filtered out to further simply the logs:

eventIsRepeated(EventLogs, company_id, event, isRepeated) -> EventLogs

EventLogs %>% **filter**(isRepeated != "1")-> EventLogs

More complex simplifications involving pattern-based substitutions and the merging of rows were also used (see [6]). These were all captured in an R script, available online.

6.5. Process Discovery, Mining, and Analysis

After preprocessing the event logs using the CPA-PM method, the resulting dataset was imported to two PM tools, namely Disco and ProM, for process discovery (the specific type of process mining of interest in this case study) and analysis. The preprocessing steps can go through several iterations before getting to the final answers to the analysis questions. Even after importing the dataset to PM tools, the analyst can iterate again through filtering, aggregations, and pattern substitutions before importing the resulting dataset again to PM tools (see Fig. 2).

Different variants of our R script were produced to answer different questions of our stakeholders, such as "what are the steps followed by trial users who give up before registering (non-converters)?", "what is the main process for converters?", and "how different are the processes when non-converters give up within 5 minutes from when they use the application between 1 to 14 days?". Fig. 4 shows one of these answers in the form of a Disco process model, and the other models can be found in [6].

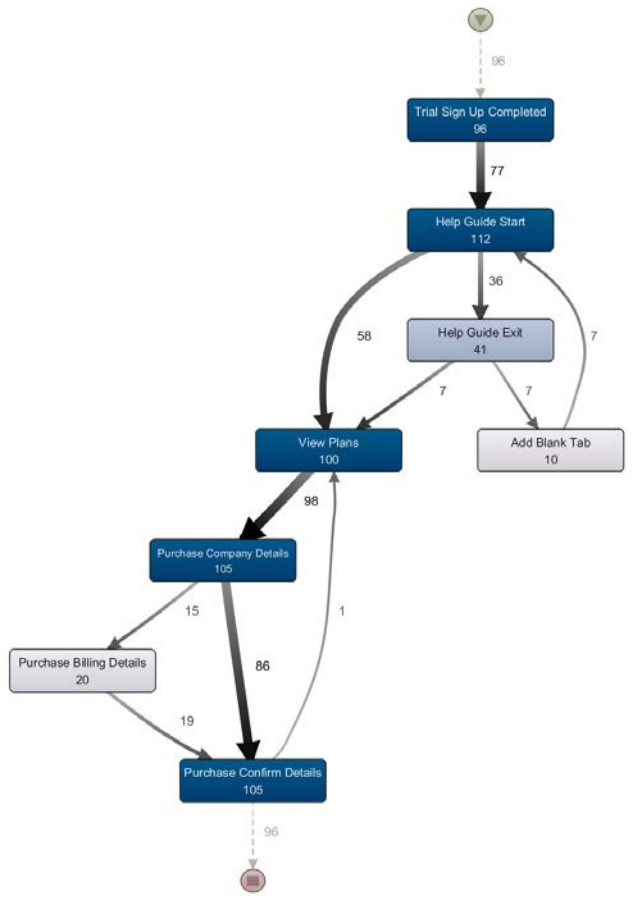

Fig. 4. Process map for users that converted to customers within 5 minutes, generated with Disco from an event log with 96 cases and 30 variants.

6.6. A Second Dataset

Another dataset was provided by the SaaS company for analysis. The purpose here is to *re-apply* the functions for preprocessing the dataset (essentially captured in a reusable script) and check how much change the original script is required in order to adapt it to new, more recent datasets. This addresses challenges #3 and #8 from Section 2.

The new dataset was collected for the period from January 2019 to August 2019, and it includes 21 different variables/attributes, 2,144,210 records/rows, and 34,315 different traces. Additionally, the new dataset includes 142 unique events, which is more than the number of events in the first dataset. That could mean that the process activities have changed over time, or that new types of events were collected while using the application. The aim here is to evaluate how well the analyst can execute the original script on this dataset in order to adjust for processes and logging environments that change over time. The initial preprocessing script was hence applied again.

Although scripts can be used many times, especially in a short time period, at some point, as processes and technologies evolve in an enterprise, the scripts many need to evolve as well. Indeed, the script developed for the initial dataset, which was used as is on other datasets from 2018, could not be reused as is here because of the new types of events that were observed in logs from 2019. Step 1.5 of the CPA-PM method was hence re-applied to handle these new events (i.e., to decide whether to keep them or not). This took a few minutes of manual effort, mainly to understand the nature of the new events. The rest of the script did not have to be modified, and its execution led to similar reductions in log and process complexity as previously observed for the first dataset. Overall, modifications to the script took a few minutes, suggesting high reusability and good tolerance to the evolution of logs over long periods of time.

7. Discussion

This section analyses the proposed CPA-PM method for helping analysts better preprocess event logs. Additionally, a scalability analysis for the method is performed. Threats to validity are also discussed.

7.1. CPA-PM Method Analysis

Often, real-life cloud-based processes are so complex that the resulting process maps are too complicated to interpret and use for discovery and analysis purpose. These *spaghetti* processes are not incorrect per se. The problem however is that such process maps are too large and unstructured to derive any useful insights or information that can be used to answer analysis questions that may lead to process or application improvement. At this level, what is needed is to simplify such complicated process maps, or their source event logs. Clickstream datasets can contain millions of events capturing user clicks anywhere on application pages, as well as page refreshes.

The CPA-PM method proposed here can help in cleaning such datasets in a suitable way in order to build more meaningful process maps automatically and repeatedly for different datasets using the same logging infrastructure. After applying the steps of the CPA-PM method, the resulting process maps are less complex and more understandable in terms of displaying the main events that were executed in the process. Many traces and events can be deleted or aggregated without affecting the answers to analysis questions. This method also focuses on reducing variations in one same process. Finally, the API invocations used to support the CPA-PM steps for a particular event log can be collected as a script that can be run each time a new version of the event log becomes available, hence helping automate the analysis while reducing manual labor. This preprocessing method is applied on the dataset *before* importing the reduced dataset to process mining tools. In this paper, Disco was used to display process maps.

Analysts can also discover along the way that a better logging mechanism should be used in their cloud system. For example, not all user activities may be logged properly in the log management tools. After applying process mining techniques and building process maps, analysts may discover that some activities that should be present are actually not displayed in the process maps since they are not properly logged.

Note however that the logs may also be simplified too much, and that important information may be removed along the way. CPA-PM does not compute similarity metrics or information loss (as in [12]); such features could be added in the future.

7.2. Scalability Analysis

As the number of records in event logs increases, the resulting process maps often become more complex. More preprocessing iterations and effort are hence usually required. It is important to test the scalability of the CPA-PM method and scripts when the size of event logs increases. A scalability analysis was performed on subsets of a real datasets that vary in size (by doubling the initial one 9 times). The same script with the same API function invocations was run on each of the datasets. The script was run three times on each dataset, and the average execution times are reported in Fig. 5.

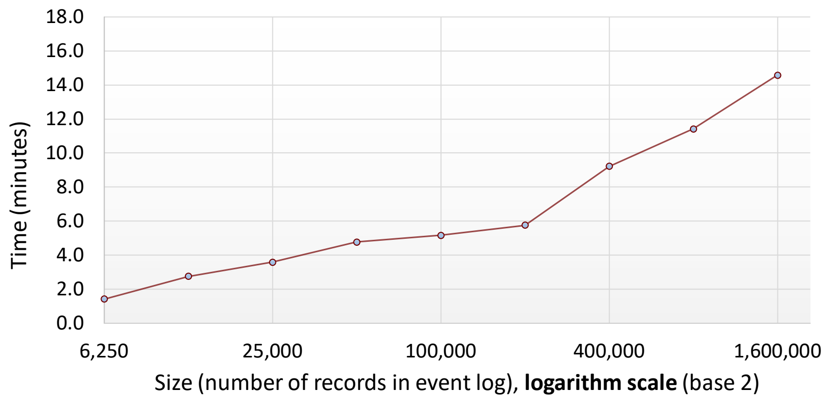

Fig. 5. Script execution performance (the X axis uses a logarithmic scale).

In term of environment, the scalability analysis was performed with RStudio running on a laptop with Windows 10 (64-bit), 16 GB RAM, and an Intel® Core™ i5-8250U processor. No other application was running in parallel.

As expected, the higher the number of *events* in the log, the higher the average runtime. This might cause issues in some contexts where this part of the analysis is time sensitive. What is interesting in Fig. 5 however is that the increase in average duration is less than linear with respect to the number of events in the log, likely because the initial filtering steps are efficiently removing many of the records up front.

In some situations, the dataset might include more *attributes* (i.e., more columns), which would require more computations when merging the rows and hence result in longer run times. The impact of such additional attributes was not assessed explicitly here. However, in situations where more pattern iterations are executed, the number of rows would be reduced, which might in turn shorten the average runtime.

Still, in general, having to wait 10-12 minutes to pre-process automatically a million events, in a repeatable way, is still much better than doing so manually, as the latter approach would take much longer and be more prone to errors.

7.3. Threats to Validity

Several potential threats to the validity of this research are discussed here.

Construct validity aims to assess the extent to which the tests performed actually measure what our method claims to be doing. An important threat here is that the chosen case study may not reflect all the different analysis questions that process maps can answer. The data quality may have been an issue in answering some further questions about the processes in the case study. Additionally, there was only one case study in this paper and the data was provided only by one source. Another threat is that only two process mining tools were used to build the process maps. To mitigate some of these threats, a second dataset was provided to the method and the script could be used again. The limitation remains that the new dataset was provided by the same stakeholder and from the same system, just at a different time and for a different period.

Internal validity aims to estimate and evaluate the degree to which conclusions about the analysis of the proposed method can be made based on the case study and the data provided. The first threat here is that bias might have been introduced by having to perform the evaluation and analyze the results of the case study. This threat was partially mitigated by having two people evaluate the results of the case study. Also, this could be further mitigated by having the stakeholders apply the method on other datasets and having them evaluate it themselves.

External validity aims to estimate whether results of the evaluation can be generalized to other cases. Although there is no reason to believe the proposed method and API cannot be used on other datasets (even outside the context of cloud-based applications), there is currently no evidence they can. To mitigate this threat in the future, we can apply the method and API functions on dataset provided by different stakeholders and for different types of processes.

8. Conclusions

This paper contributes a data preprocessing method (CPA-PM) for event logs generated by cloud-based information systems, with an emphasis on clickstream data. To properly apply process mining on such logs, advanced and iterative preprocessing must be done in order simplify the resulting mined processes without losing valuable information. It is also important that the preprocessing steps be repeatable. CPA-PM includes different steps needed to clean and restructure cloud-based event logs and address relevant challenges in that area (especially #2 to #9 in Section 2), including a large number of events and attributes, spurious and repetitive events, and unstructured orderings of events. As an additional contribution, the method is supported by a new API with an implementation in R, which enable the scripting and reuse of preprocessing steps. CPA-PM goes beyond what is usually found in ETL tools and in the preprocessing capabilities of process mining tools.

There are many opportunities to extend this research work, some of which being identified in the previous section. We also plan to extend the CPA-PM method to include more functions for preprocessing event logs, provide a Python implementation, and integrate the API to an existing PM tool. Additionally, more case studies can be done to apply the method on event logs that represent user actions while navigating the different parts of an application, or different application domains. The integration of similarity and information loss metrics, together with support for multiple, different logging systems in cloud environments, would also be beneficial. The usability of the API also deserves further attention in the future.

Supplementary Materials

The R implementation of the API and an example are freely available online at: https://github.com/NajaElgharib/CPA_PM.

Acknowledgement

We thank Dr. Alireza Pourshahid and Tomasz Ogrodzinski for their help in providing event logs and assessing the resulting process models. The first author thanks the University of Ottawa and the Ontario Graduate Scholarship for funding this work.

Author Contributions

Conceptualization, N.M.E.-G. and D.A.; Methodology, N.M.E.-G. and D.A.; Software, N.M.E.-G.; Validation, N.M.E.-G. and D.A.; Formal Analysis, N.M.E.-G.; Investigation, N.M.E.-G. and D.A.; Resources, N.M.E.-G.; Data Curation, N.M.E.-G.; Writing – Original Draft Preparation, N.M.E.-G. and D.A.; Writing – Review & Editing, N.M.E.-G. and D.A.; Visualization, N.M.E.-G.; Supervision, D.A.; Project Administration, N.M.E.-G. and D.A.; Funding Acquisition, D.A.

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 31(4), pp. 686–705 (2018)

[2] Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in Process Mining: A Taxonomy of Patterns. In: *Int. Conference on Business Process Management*, LNCS 5701, pp. 159–175. Springer (2009)

[3] de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.M.P.: Connecting databases with process mining: a meta model and toolset. *Software & Systems Modeling*, **18**(2), pp. 1209–1247 (2019)

[4] De Weerdt, J., Vanden Broucke, S., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. *IEEE Transactions on Knowledge and Data Engineering*, 25(12), pp. 2708–2720 (2013)

[5] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: *Fundamentals of business process management*, Springer (2013)

[6] El-Gharib, N.M.: *Using Process Mining Technology to Understand User Behavior in SaaS Applications*. Master's thesis, University of Ottawa, Canada, December (2019). DOI:10.20381/ruor-24202

[7] El-Gharib, N.M., Amyot, D.: Process mining for cloud-based applications: a systematic literature review. In: *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pp. 34–43. IEEE CS (2019)

[8] Günther, C.W., Rozinat, A.: Disco: Discover Your Processes. In: *BPM (Demos)*, CEUR-WS 940, pp. 40–44 (2012). https://fluxicon.com/disco/

[9] Hevner, A., Chatterjee, S.: Design science research in information systems. *Design Research in Information Systems*, vol. **22**, pp. 9-22. Springer (2010)

[10] Kumar Padigela, P., Suguna, R.: A Survey on Analysis of User Behavior on Digital Market by Mining Clickstream Data. In: *Third International Conference on Computational Intelligence and Informatics*, AISC 1090, pp. 535–545. Springer (2020)

[11] Marin-Castro, H.M., Tello-Leal, E.: Event Log Preprocessing for Process Mining: A Review. *Applied Sciences*, 11(22), 10556 (2021)

[12] Sani, M.F.: Preprocessing Event Data in Process Mining. In: *CAiSE (Doctoral Consortium)*, CEUR-WS 2613, pp. 1–10 (2020)

[13] Terragni, A., Hassani, M.: Analyzing Customer Journey with Process Mining: From Discovery to Recommendations. In: *2018 IEEE 6th Int. Con. on Future Internet of Things and Cloud*, pp. 224-229 (2018)

[14] van der Aalst, W.M.P.: Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining. In: *9th Confed. Int. Conf. Move to Meaningful Internet Systems*, LNCS 6426, pp. 8–25. Springer (2010)

[15] van der Aalst, W.M.P.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag (2011)

[16] van der Aalst, W.M.P.: Intra- and Inter-Organizational Process Mining: Discovering Processes within and between Organizations. In: *The Practice of Enterprise Modeling. PoEM 2011*, LNBIP 92, pp. 1–11. Springer (2011)

[17] van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A. Event abstraction in process mining: literature review and taxonomy. *Granular Computing*, 6, pp. 719–736 (2021)

[18] Wynn, M.T. et al.: Rethinking the Input for Process Mining: Insights from the XES Survey and Workshop. In: *Process Mining Workshops. ICPM 2021*. LNBIP 433, pp. 3–16. Springer (2022).