Using Process Mining Technology to Understand User Behavior in SaaS Applications

Najah Mary El-Gharib

Thesis submitted to the
Faculty of Engineering
in partial fulfillment of the requirements for the degree of

Master of E-Business Technologies

uOttawa

University of Ottawa
Ottawa, Ontario, Canada
December 2019

Abstract

Processes are running everywhere. Understanding and analyzing business and software processes and their interactions is critical if we wish to improve them. There are many event logs generated from Information Systems and applications related to fraud detection, healthcare processes, e-commerce processes, and others. These event logs are the starting point for process mining. *Process mining* aims to discover, monitor, and improve real processes by extracting knowledge from event logs available in information systems. Process mining provides fact-based insight from real event logs that helps analyze and improve existing business processes by answering, for example performance or conformance questions. As the number of applications developed in a cloud infrastructure (often called *Software as a Service* – SaaS at the application level) is increasing, it becomes essential and useful to study and discover these processes. However, SaaS applications bring new challenges to the problem of process mining.

Using the Design Science Research Methodology, this thesis introduces a new method to study, discover, and analyze cloud-based application processes using process mining techniques. It explores the applications and known challenges related to process mining in cloud applications through a *systematic literature review* (SLR). It then contributes a new *Application Programming Interface* (API), with an implementation in R, and a companion method called *Cloud Pattern API – Process Mining* (CPA-PM), for the preprocessing of event logs in a way that addresses many of the challenges identified in the SLR. A case study involving a SaaS company and real event logs related to the trial process of their online service is used to validate the proposed solution.

Acknowledgment

This thesis would not have been possible without the support of many people. First of all, I would like to thank my supervisor, Professor Daniel Amyot, for his help, support, constructive feedback, and helpful guidance to complete my thesis on time. It is a pleasure for me to work with him, it has been a great and valuable experience. I could have not asked for a better supervisor.

Also, I would like to thank the University of Ottawa for providing me with the Admission Scholarship and the University of Ottawa Excellence Scholarship to continue my master's studies. Thanks to the Government of Ontario for providing me with the Ontario Graduate Scholarship to complete my thesis work.

Additionally, I would like to thank my colleagues within the same team of research. It was a pleasure to know every single one of you. Thanks for all your help and thanks for always being available to answer questions and giving advice.

Many thanks to my family for all their support, care, and encouragement during my studies. Mainly, I would like to thank my parents, siblings, and my uncle Melhem for all their help, love and support, without their support I would have not been able to achieve this milestone. I am so thankful to have such a loving and caring family.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
API	Application Program Interface
AWS	Amazon Web Services
BAB	Best Analytics of Big Data
BPM	Business Process Management
BPMN	Business Process Model and Notation
CPA-PM	Cloud Pattern API – Process Mining
CSV	Comma-Separated Value
DSRM	Design Science Research Methodology
ETL	Extract, Transform, and Load
FHM	Flexible Heuristic Miner
GCP	Google Cloud Platform
ID	Identifier
IDE	Integrated Development Environment
IS	Information Systems
IT	Information Technology
IaaS	Infrastructure as a Service
KPI	Key Performance Indicator
MHMS	M-Health Monitoring System
PaaS	Platform as a Service
PM	Process Mining
POD	Process-Oriented Dependability
SaaS	Software as a Service
SLR	Systematic Literature Review
SWMaaS	Scientific Workflow Mining as a Service
SWMC	Scientific Workflow Mining in Clouds
URL	Unified Resource Locator
XES	Extensible Event Stream

Chapter 1. Introduction

This thesis introduces a new *Cloud Pattern API – Process Mining* (CPA-PM) method and an *Application Programming Interface* (API) with an R implementation in order to preprocess, discover, study, and analyze cloud-based application processes using process mining techniques. This chapter provides the research motivation, thesis objectives, and research questions. The research methodology selected for this thesis is also presented. The research contributions are presented as well. The conclusion of this chapter lists the outline of this thesis.

1.1. Motivation

Processes are running everywhere, and their number is rapidly increasing. A *process* is defined as a set of related activities that happen over time in order to achieve a goal (van der Aalst, 2011). Processes can be found in companies, hospitals, government institutions, universities, and many other places. More and more organizations these days use Information Technology (IT) systems to support their business processes. Many IT-based Information Systems (IS) record what happens in running processes as *event logs* that contain information such as timestamps, activities, amounts, customers, or resources. Event logs are usually stored in log files or collected and stored in log management tools. All these event logs form a large amount of data ready to be discovered, understood, and exploited through process mining.

Process mining (PM) refers to the extraction of process maps (or models) from collected event logs (van der Aalst, 2011). These collected logs are the main starting point for process mining techniques. Process mining helps in discovering the different variations of a process and generates process model visualizations, often in informal representations or in more formal languages such as the Business Process Model and Notation – BPMN (OMG, 2014) or Petri nets (ISO/IEC 2019). However, process mining is not just about collecting data; it

is, also, about analyzing resulting processes to improve them. Process mining is often needed, in reality, because processes are much more complex than the ideal process definition designers and managers have in mind. Different people perform the same process definition in different ways.

The core idea is to analyze data from a process perspective. Process mining enables answering questions such as: “How does my as-is process look like?”, “Are there any deviations from the expected process definition”, “Are there any waste and unnecessary steps that can be eliminated?”, or “Where are the bottlenecks?”. To be able to answer such questions, PM approaches data from a process point of view, mapping this data to process models.

Process mining is about exploiting event logs to obtain meaningful information about the processes that produced them. According to van der Aalst (2011), there are three type of process mining: process discovery, conformance, and enhancement (see Figure 1).

1) *Discovery*: process discovery techniques take an event log as input and produce a process model. The *α-algorithm* is an example of a process discovery technique that takes an event log and produces a Petri net explaining the behavior recorded in the logs.
2) *Conformance:* compares an existing process model with an observed behavior in the event logs. Conformance techniques check if reality, as recorded in the logs, conforms with a predefined (or assumed/desired) model. Conformance checking may be used to detect, locate, and explain deviations from the expected behavior and to measure the severity of such deviation.
3) *Enhancement:* improves and extends an existing process model based on insight generated by process mining. Whereas conformance checking measures the alignment between defined model and reality, enhancement aims to change or extend the a-priori model, or support ways to encourage desired behaviors and discourage undesired behaviors.

This thesis is particularly focusing on process discovery.

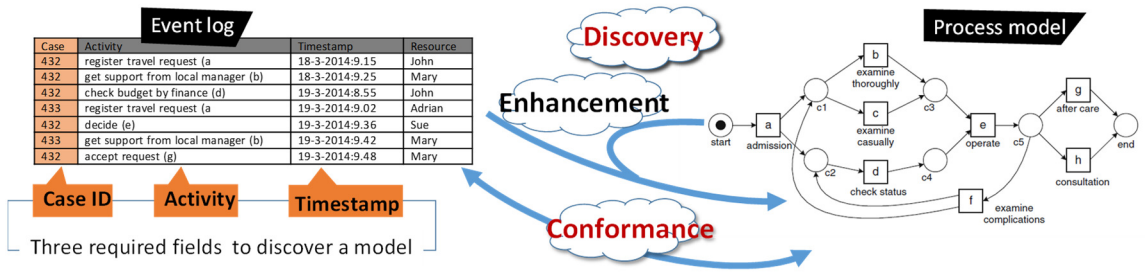

Figure 1: Overview of process mining (with thanks to M. Ghasemi).

There are many log management tools that are being used by companies to monitor their application and to alert when a failure happens. But log management tools do not look at data from a process perspective. These tools collect data for analysis but do not deal with the data as a process from start to end. For all the reasons mentioned above, there is a need to create a process mining model to help companies better understand their processes and how they are executed in reality.

The interest in cloud computing is also rapidly growing in the industry and research communities. *Cloud computing* focuses on sharing resources to reduce costs significantly while simplifying application access, resilience, and management. Cloud computing is also changing the way business processes are managed and supported. Real-life processes collected from cloud-based applications that capture the user actions on such applications are flexible but less structured processes. Cloud applications include SaaS, PaaS, and IaaS applications. This thesis focuses only on SaaS applications. For example, a SaaS application accessed through a Web browser will likely allow users to perform activities in many different orders (e.g., by clicking on different buttons), go back to a previous page, etc. Existing process mining techniques have problems dealing with such type of processes. This research aims to better characterize the challenges that exist in cloud/SaaS applications and propose a method to preprocess cloud event logs to make it more feasible to study such processes.

1.2. Research Questions and Thesis Goals

The research objective is to develop a method to preprocess, study and characterize event logs collected from cloud-based applications to discover the user actions. The objective of this thesis work is to examine how process mining can be applied on clickstream data collected from SaaS applications and discover users' behavioral characteristics.

This thesis aims to answer the following research questions, originating from the thesis objective:

- ***RQ1***: How is process mining utilized on cloud-based systems?
- ***RQ2***: What are the challenges encountered when discovering process models from cloud-based application processes?
- ***RQ3***: What is a suitable method to discover and analyze cloud-based application processes?

A *suitable* method means that it provides high automation (e.g., by being scriptable), high abstraction, high scalability, and low effort to accommodate changes. The focus of such method is also on *preprocessing* event logs, where preprocessed logs can then be fed to a number of process mining tools for visualization and analysis.

1.3. Research Methodology

In order to conduct a good research project and study, an appropriate methodology is needed. The research methodology selected here is inspired from Hevner et al. (2004): Design Science Methodology (DSR) in Information System. DSR's first step about awareness of the problem is in particular addressed here through a systematic literature review that helps identifying several process mining challenges specific to SaaS applications. The detailed methodology is explained in Chapter 4 of this thesis.

1.4. Thesis Contributions

There are three main contributions in this thesis:

- The first contribution, addressing RQ1 and RQ2, is a *Systematic Literature Review* (SLR) about how process mining is being applied to study and analyze cloud-based application processes and clickstream data, with a list of observed challenges. This SLR was conducted using three databases and it analyzes 27 different research papers.
- The second contribution, partially addressing RQ3, is an *API* comprising many functions needed to preprocess event logs. An R implementation of the API is also provided. An API is important as this enables automation through *scripting*, which is not handled by other solutions.
- The third contribution, addressing RQ3, is the *Cloud Pattern API – Process Mining* (CPA-PM) method to preprocess less-structured event logs and enable simpler process discovery. This method consists of three different stages and each stage include several steps used before importing logs into process mining tools to build process maps. These steps invoke the functions of the API. The method and the API are validated using event logs from a real SaaS application, suggesting a high level of automation (through *scripting*), a relevant abstraction level, good scalability, and low effort to accommodate changes to the structure of logs in evolving processes.

1.5. Thesis Outline

This thesis is organized as follows:

- **Chapter 2** presents background information about process mining, together with its characteristics and different types. It also explains what event logs are and introduces common process mining tools. A process mining project methodology is also introduced.
- **Chapter 3** presents a systematic literature review that investigates how process mining is applied to cloud processes. This is the first step of our application of the DSR methodology (to get aware of the problems in that space). It also explores how process mining is applied to study clickstream data collected from

user actions using these different applications. A list of cloud-specific PM challenges is finally presented.

- **Chapter 4** presents the detailed research methodology, based on DSR, that was used to conduct this research. The different stages and iterations of this research methodology are explained as well.
- **Chapter 5** presents the Cloud Pattern API – Process Mining method, which is a new approach to preprocess event logs collected from cloud applications before importing them to process mining tools. The method steps are explained as well, and make use of a new API, for which an R implementation is provided.
- **Chapter 6** presents a case study for real event log datasets provided by a SaaS company. The CPA-PM method steps were applied on two versions of a dataset (for a product trial process) and the results are presented.
- **Chapter 7** discusses the results, scalability issues, threats to validity and limitations.
- **Chapter 8** concludes the thesis, revisits the contributions, and discusses future work.

Chapter 2. Background on Process Mining

This chapter gives a literature overview of the relevant theoretical background related to process mining, its tools, and applications. According to van der Aalst (2011), a *process* is a set of related activities that happen over time in order to achieve a goal. Section 2.1 presents an overview of process mining. Section 2.2 highlights the four main characteristics of PM. Section 2.3 discusses the main parts of a typical event log. Section 2.4 discusses the different types of PM. Section 2.5 presents important quality criteria of discovered process models. Section 2.6 introduces some of the PM tools that are widely used in academia and industry. Then cloud applications, cross-organizational PM, and process pattern recognition are defined from sections 2.7 to 2.9. Lastly, the different steps of a typical PM project methodology are explained.

2.1. Overview of Process Mining

Over the last decade, there has been a massive growth of the digital world that is producing a huge amount of event data to support understanding and compliance. The challenge is to analyze and exploit these collected event data in a meaningful way. This collected data has been a very important aspect and an asset for many organizations. Knowledge extracted from such data is driving businesses in the decisions that are made. *Process mining* aims to discover, monitor, and improve real processes by extracting knowledge from event logs (van der Aalst, 2011a). Classical *data mining* techniques such as classification, clustering, regression, and association rules do not focus on business process models (or timed events) and are often used to analyze specific data points in the overall process. Process mining, on the other hand, focuses on analyzing end-to-end processes and is possible because of the growing availability of event data collected by information systems and of recent process discovery techniques. The starting point of process mining is an *event log*. Further explanation about event logs is explained in section 2.3.

2.2. Characteristics of Process Mining

PM techniques focus on studying and analyzing data from a process perspective and generate business process models that are used to analyze the overall process. According to van der Aalst (van der Aalst, 2012a), the important characteristics of process mining are as follows:

1. **Evidence-based:** PM is claimed to be *evidence-based* since it is based on observed behavior recorded in event logs, on which intelligent techniques are used to extract knowledge (van der Aalst, 2011a; 2012a).
2. **Fact-based:** PM is based on event data and not on opinions, which is why it is considered *fact-based*. The observed behaviors of the end users and the systems are recorded in logs and are the main starting point of process mining (van der Aalst, 2011a; 2012a).
3. **Truly intelligent:** PM is *truly intelligent* because it learns from historic data and uses this knowledge for enabling process model enhancement. PM techniques can be used to discover processes and apply conformance checking (van der Aalst, 2011a; 2012a).
4. **Process-centric:** PM is related to processes, not only to data. It analyzes the data from a process perspective and looks at processes from end to end. It is not a *data-centric* approach like other data mining techniques (van der Aalst, 2011a; 2012a).

2.3. Event Logs

Event logs extracted from information systems are the foundation of PM (van der Aalst, 2011). The main assumption about the event log structure is that it contains data related to a *single process*, but possibly with many interleaved instantiations. Each event in a log refers to an *activity*, which means a well-defined step in a process, and is related to a particular *case* identifying a process instance. The events belonging to a case are sorted or contain a *timestamp* enabling sorting. Events may have additional attributes such as resources, cost, etc. Some of the important terms related to event logs and used in this thesis are discussed next:

Case: A case is a specific instance of a process (Rozinat, 2016). A process contains several cases. Each case has a unique identifier referred as case ID. The case depends on the domain of the process. For example, in a purchasing process, the handling of one purchase order is one case. For every event, we need to know which case it refers to, so that the process mining tool can compare several executions of the process to one another. We need to have at least one column that uniquely identifies a single execution of the process, which represents the **case ID**. The case ID determines the scope of the process instance. It determines where your process starts and where it ends. For example, in Table 1, there are four different cases identified by the **case ID**.

Activity: Each event in a log refers to an activity (Rozinat, 2016). An activity forms one step in the process. It is a well-defined step in some process. Some of the steps might occur more than once for a single case, and not all activities need to happen in a case. The activity name determines the level of detail for the process steps. In Table 1, the **Activity** column represent the event name for each event in the log. For example, the different activities that are recorded include "*Main home page*", "*Home page for product*", "*General website search*", and "*Add to cart*", all these are different events performed by the user.

Timestamp: a timestamp[1] is a sequence of characters or encoded information identifying when a certain event occurred, usually giving date and time of the day, and sometimes it also includes seconds or fractions of a second. This is not only important for analyzing the timing behavior of the process but also to determine the order of the activities in the event logs. In Table 1, the second column, which is the **Timestamp** column, represents the time when each event occurred. In some cases, we have a *start* timestamp and an *end* timestamp for each activity in the process. This helps to analyze the processing time or duration of an activity.

Other Columns: Event log file may contain columns other than the main ones (Case ID, Timestamp and Activity). For example, in Table 1, there are two additional column the "**Product Category"** and the "**Device**". The "Product Category" column includes the

[1] https://en.wikipedia.org/wiki/Timestamp

information about the product the user is browsing for example. Additionally, the "Device" column contains information about the device type that is used by the user. Log files may also contain a "**Resource**" column that is associated with a resource. Not all log files contain resource information. When analyzing cloud-based application processes log files contain many columns associated with each event. These columns represent different properties about each event, these properties could be related to user information or system information.

According to Rozinat (2016), in order to use process mining tools, event logs should contain at least these three elements: *case ID, activity*, and *timestamp*. Table 1 represents an event log dataset, where the rows are the events and the columns their elements, and Figure 2 represents a process model discovered using process mining and capturing the event log. The numbers on the activities (rectangles) and sequence relations (arrows) represent frequencies.

Table 1: A part of event log file.

Case.ID	Timestamp	Activity	City	Weekday	Client	Product.Category	Device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	A	Android
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	N/A	Android
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	D	iPhone
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	B	N/A
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	A	Android
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	X	N/A
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	N/A	Android
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	N/A	N/A
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	N/A	Android
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	N/A	iPhone
…	…	…				…	...

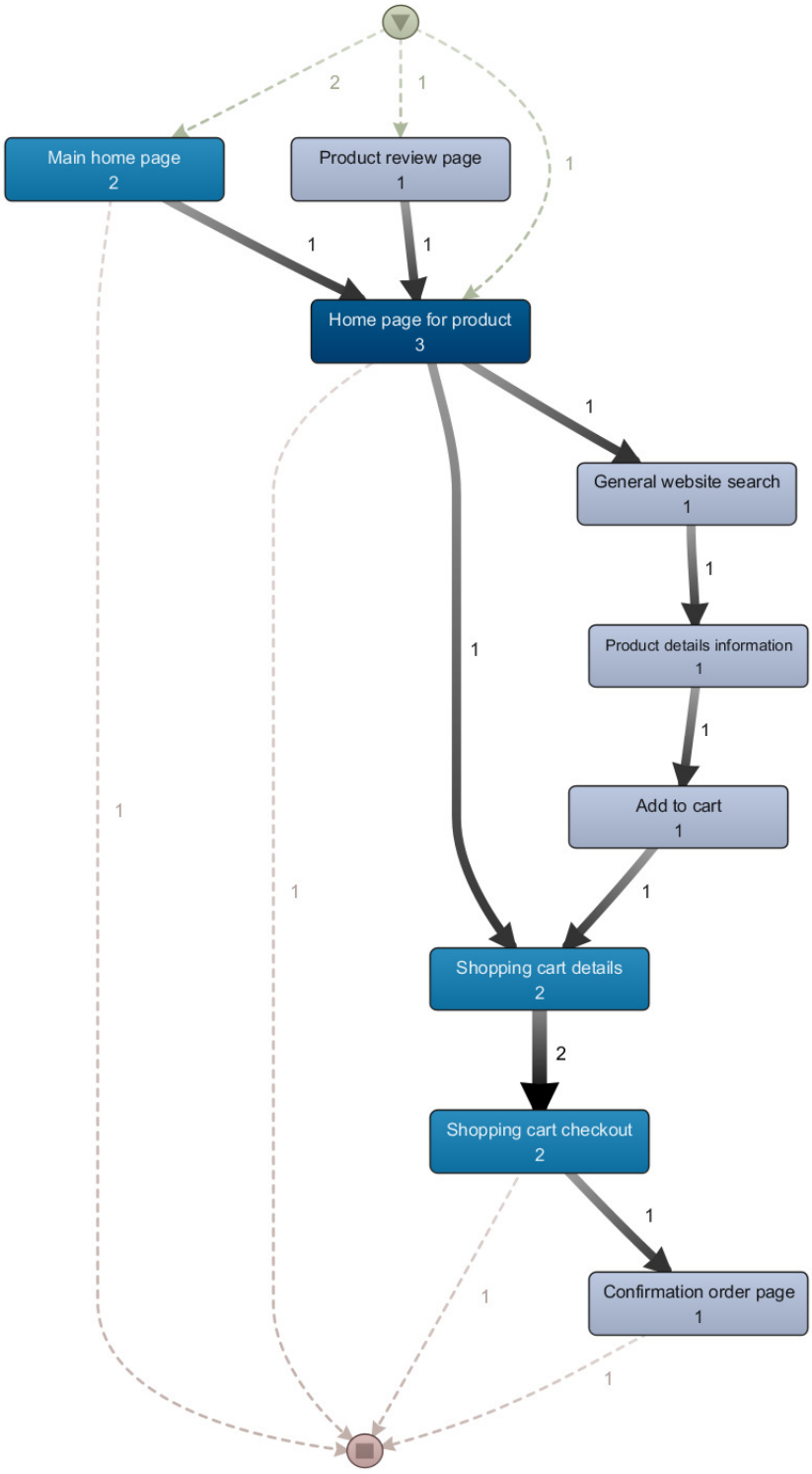

Figure 2: Process map discovered from the event log in Table 1.

2.4. Types of Process Mining

Event logs can be used to enable three main types of process mining, as shown in Figure 3. According to van der Aalst (2011), these types are process *discovery, conformance,* and *enhancement*.

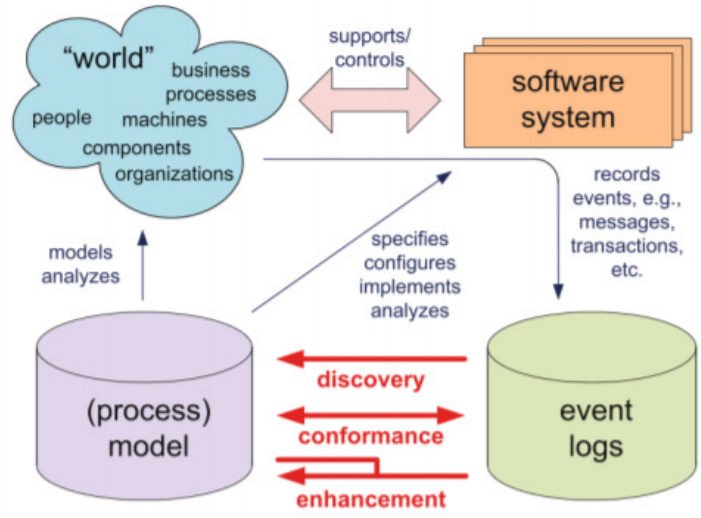

Figure 3: Three main types of process mining: *discovery, conformance, and enhancement* (van der Aalst, 2011)

2.4.1 Process Discovery

A discovery technique takes an event log as input and produces a process model without using any priori information (van der Aalst, 2011). Process discovery is the most popular process mining technique. Let *L* be an event log. A process discovery algorithm is a function that maps *L* onto a process model such that the model represents the behavior seen in the event log. An example of a process discovery algorithm is the α-algorithm (van der Aalst et al., 2004), which takes an event log and produces a Petri net explaining the behavior recorded in the log. Such models are useful for understanding as-is processes based on evidence.

2.4.2 Conformance Checking

In this type, an existing process model is compared with an event log of the same process. Conformance checking is used to verify whether reality, as recorded in the log, conforms to an existing or desired model (van der Aalst, 2011). Conformance checking uses a model

and event logs as input. The modeled behavior and the observed behavior are compared to find similarities and differences. For example, conformance checking can be used for the following purposes:

1. To discover outliers of a process by discovering exceptional behavior observed in reality (van der Aalst, 2011a).
2. To check whether the rules and regulations are obeyed and to discover deviating cases (van der Aalst, 2011a; 2012a).
3. To compare the documented process model with the real process as observed in the event logs and identify what they have in common (van der Aalst, 2012a).
4. As the starting point of process enhancement, since it helps to identify process steps and deviation points (van der Aalst, 2012a).

According to van der Aalst (2011a), there are three approaches for conformance checking: *comparing footprints, token replays,* and *alignments*. They are illustrated below:

1. *Comparing Footprints:* This approach creates an abstraction of the behavior that is allowed by the model. A *footprint* is a matrix showing causal dependencies between activities (van der Aalst, 2011a). For example, in the footprint of an event, log *x* may be followed by *y* but never the other way around. If the footprint of the corresponding model shows that *x* is never followed by *y* or that *y* is sometimes followed by *x*, then the footprints of the event logs and the model disagree on the ordering relationship of *x* and *y* (van der Aalst, 2012a).
2. *Token Replays:* This approach *replays* the event log on the model. In this approach, we have an event log and a Petri net model as input, and the event log dataset is replayed trace by trace. Replaying the event logs on the model helps measuring the fitness of the model (van der Aalst, 2011a; 2012a).
3. *Alignments:* This is one of the most advanced approaches. This approach computes an optimal alignment between each trace in the log and the most similar behavior in the model (van der Aalst, 2012a).

2.4.3 Enhancement

The purpose in enhancement is to extend and improve an existing process model by using information about the actual process recorded in the event logs that is discovered using PM (van der Aalst, 2011a). One type of enhancement is improving the model to better reflect reality. The types of enhancement can be *repair* or *extension* (van der Aalst, 2011a). In *repair*, the model is modified to better reflect how processes are executed in reality. For instance, if two activities are modeled to occur in certain order, but in reality, they can happen in any order, then the model can be modified to reflect this information about events. In *extension*, more information is added to the process model. For instance, adding new data to the analysis of the processes such as resources, properties, metrics, performance attributes, etc.

2.5. Quality Criteria of Discovered Model

Another critical task when applying PM techniques is to measure the quality of the discovered model. It is important to assess whether the discovered model represents the information in the event logs. According to van der Aalst (2011a), the discovered process model can be measured by these four main quality dimensions:

1. *Fitness*: The discovered model should allow for the behavior seen in the event log.
2. *Simplicity*: The discovered model should be simple and clear for the user to understand.
3. *Precision*: The discovered model should not allow for behavior completely unrelated to what was seen in the event log.
4. *Generalization*: The discovered model should generalize the example behavior seen in the event log.

Note that these dimensions often conflict (e.g., generalization promotes emerging behavior not observed in the event log whereas precision aims to prevent it), and their relative importance may depend on the context of use.

2.6. Process Mining Tools

There are several tools that facilitate gathering the logs files and performing process mining.

2.6.1 ProM

The ProM[2] framework (see Figure 4) has been developed as a completely pluggable open-source environment. It is platform independent as it is implemented in Java. ProM is an extensible framework that supports a wide variety of process mining techniques in the form of plugins. Process discovery, conformance checking, social network analysis, organizational mining, and decision mining are all supported by ProM. It also supports common process discovery algorithms, for example alpha and heuristic miner.

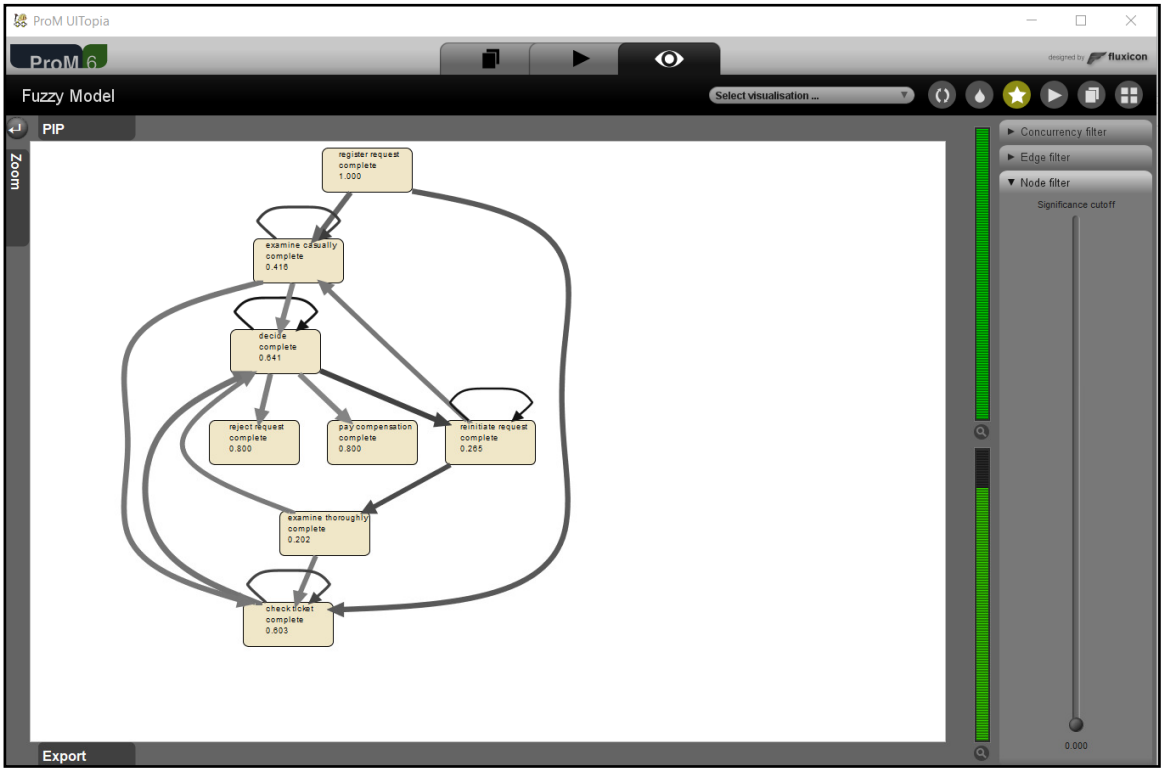

Figure 4: Screenshot for the ProM process mining tool.

[2] http://www.promtools.org/doku.php

2.6.2 Disco

DISCO[3] is a commercial process mining suite supported by a leading academic group from at Eindhoven Institute of Technology. Disco is user friendly and very efficient on large event logs while discovering high-level knowledge. However, it does not support as many algorithms as ProM, just the simpler ones. Figure 5 represents a screenshot of the Disco interface.

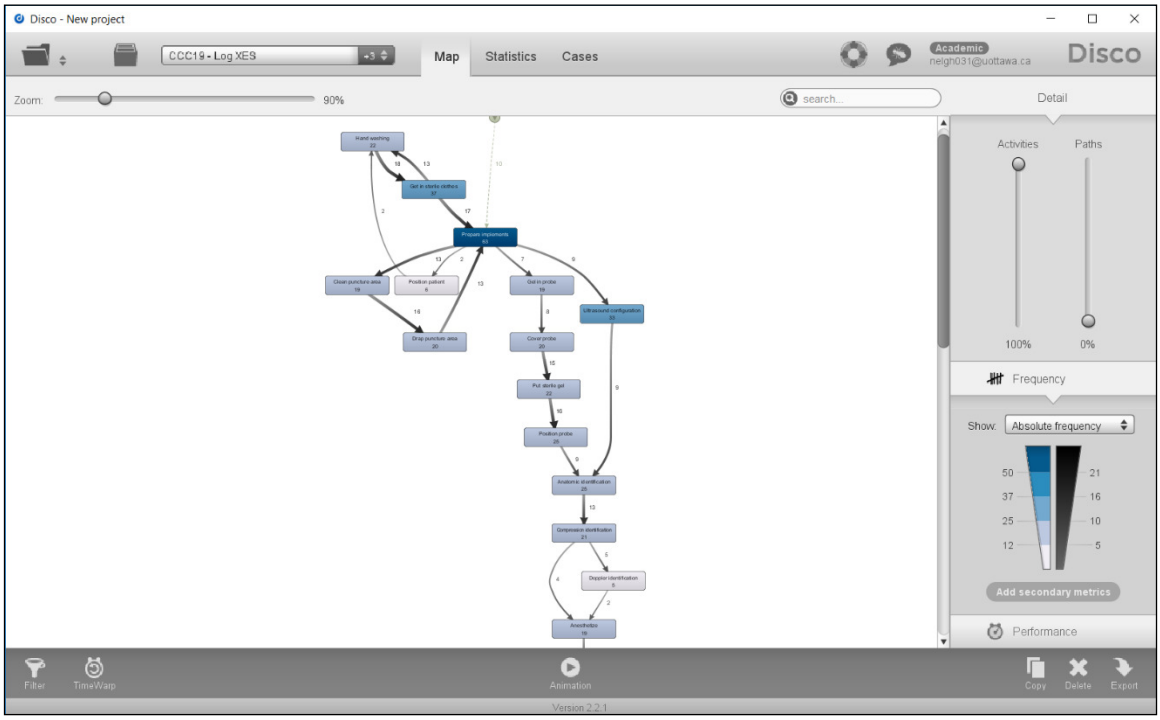

Figure 5: Screenshot for the Disco process mining tool.

Disco has three different view options:

- **Map View**: shows the process map that represents the actual flow of the process based on imported Comma-Separated Value (CSV) data. Frequencies of activities and flows are indicated, and can be used to filter out infrequent cases or activities.
- **Statistics View:** used to gather additional information and detailed performance metrics about the process.

[3] https://fluxicon.com/disco/

- **Case View**: used to view and inspect *individual cases* and see the raw data. This view is used for seeing individual logs entries or complete log entries, individual variants, or list of cases, or for searching particular cases.

2.6.3 OpenXES

OpenXES[4] is a reference implementation of the first Extensible Event Stream (XES)[5] standard for storing and managing event log data. The OpenXES library is implemented in Java and strives for strict XES compatibility, ease of development, and the best possible performance. OpenXES is released as open source and free software.

2.6.4 Celonis

Celonis[6] is a commercial, Web-based process mining tool. The Intelligent Business Cloud Celonis product helps in mining system logs and visualizing processes in real time. Celonis helps in identifying the root causes of any deviation in processes and opportunities for any process improvement. Figure 6 displays a screenshot of the Celonis tool interface.

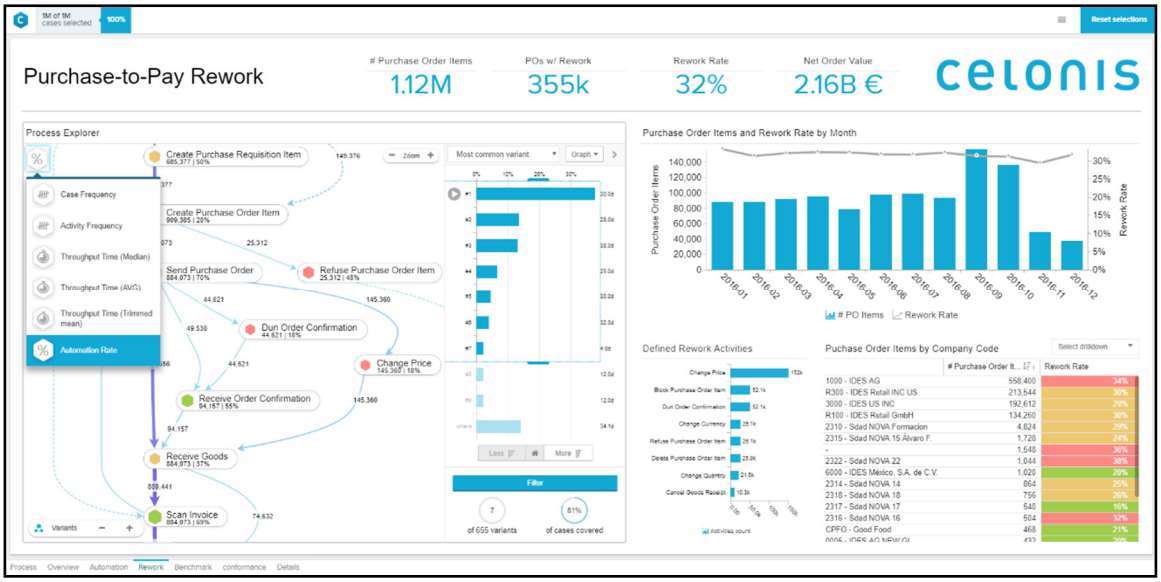

Figure 6: Screenshot of Celonis process mining tool.

[4] http://www.xes-standard.org/openxes/start
[5] http://www.xes-standard.org/xesstandarddefinition
[6] https://www.celonis.com/

2.6.5 Other Tools

There are several other commercial process mining tools and products, for example, Process Gold, minit, PUZZLE Data, QPR, and Lana Labs. However, ProM, Disco, and Celonis are certainly amongst the most popular in academia and industry. A good comparison between PM tools was done by Kebede and Dumas (2015).

2.7. Cloud Applications

According to Armbrust et al. (2010) cloud computing refers to both applications or software services delivered to end users, and hardware and software systems supporting these applications. Cloud resources and services are distributed through a multi-tenant architecture that controls how computing resources are shared among different nodes. *Multitenancy* is a cloud design where computing resources are shared across multiple tenants or organizations in the cloud.

According to Bobrowski (2011), customers represent the different tenants and each tenant is isolated and remains invisible to the other tenants. While using cloud services, organizations only pay for what they use; this is a pay-as-you-go approach, which is different from conventional client/server and data centers approaches. In the traditional model, service providers customize their offering based on customer requests, while in cloud applications there is a main business process that can be configured based on the customer business.

Cloud computing environments are characterized by a high number of applications running on different tenants and by the presence of a significant amount of resources. This context allows systems owners to implement new techniques to scale dynamically their applications and to lease the right amount of resources while adapting to the incoming user demand.

2.8. Cross-Organizational Process Mining

van der Aalst (2010; 2011b) was the first to introduce and discuss the term *cross-organizational* PM and its related challenges and opportunities. In these studies, the author

focuses on stating the different organizational settings in which PM can be applied. Looking at cross-organizational processes can help identify challenges that involve multiple organizations that share the same process or interact together. Also, van der Aalst (2010) proposed configurable process models that represent a family of process models.

2.9. Event Pattern Recognition

The main purpose of applying process mining techniques on a dataset is to discover and generate process maps. When analyzing processes with many different cases and a high diversity of behavior, the models that will be generated will be confusing and difficult to understand. According to Bose and van der Aalst (2009), such process models are called *spaghetti models*. Real-world event logs include a large amount of noise, as a result this will make it more challenging to analyze such processes. Bose and van der Aalst (2009) presented an approach to address this challenge. They worked on discovering hierarchical process models using several plugins implemented in ProM. They applied the Patter Abstraction Plugin in ProM with several iterations in order to recognize patterns of events to be simplified and discover multiple levels of hierarchy for less structured processes. Their event pattern recognition approach can help in creating maps that eliminate irrelevant paths, reduce complexity, display desired traits, and improve process maps clarity.

2.10. Process Mining Project Methodology

van Eck et al. (2015) introduced a methodology to guide the execution of a PM project. This methodology consists of six different stages: *planning, data extraction, data preprocessing, process mining and analysis, evaluation of results,* and *process improvement.*

2.10.1 Planning Stage

The goal of this planning phase is to identify the processes that will be studied. The main activities in this stage are *selecting business processes* and *identifying the analysis questions* (van Eck et al., 2015). Selecting the business processes starts by identifying the business processes to be analyzed and studied in order to be understood or improved. Additionally, the process characteristics and the quality of the event data should be taken into

consideration when selecting the processes that should be analyzed. Identifying the analysis questions starts by translating the goal of the project into analysis questions. These questions are specific for the processes that are studied. Analysis questions can be related to the specific aspects of the process for example, time, quality, resources, and costs.

2.10.2 Data Extraction Stage

After identifying the processes that will be studied, this stage includes extracting the data from the proper IS and databases (van Eck et al., 2015). Based on the analysis questions, event logs are extracted. The scope of the data extraction is determined based on which event data is to be created. For example, one needs to determine the timeframe of the event data that should be included in the analysis. Then, event data can be created by collecting the selected process-related data from the relevant IS. The last step of this phase is to transfer the process knowledge, which includes considering existing process models if there are any.

2.10.3 Data Preprocessing Stage

The objective of this stage is to prepare the extracted event logs to be imported to PM tools, in such a way that is appropriate for the PM and analysis stage (van Eck et al., 2015). The input of this stage is a raw event log dataset and the output is also event logs that are ready to be imported into a PM tool. In this stage, four different activities can be applied on the event log dataset: *creating views, aggregating events, enriching logs, and filtering logs.* **This thesis mainly focuses on this stage, in a cloud-based context**. Chapter 5 will discuss in detail the steps of the PCA-PM method for preprocessing event logs collected from cloud-based applications.

2.10.4 Process Mining and Analysis Stage

In this stage, the PM techniques are applied on the preprocessed event logs, with the aim to answer analysis questions. In this stage, the different process mining types may be applied: process discovery, conformance checking, and enhancement. For example, in this stage, the event logs are imported to a process mining tool (e.g., ProM or Disco) and the process maps are generated. This stage will happen through several iterations. The analysis

can focus on answering specific questions. Given an event log as the input for this stage, process discovery techniques are applied, which returns a fact-based process model as the output.

2.10.5 Result Evaluation Stage

The objective of this stage is to relate the analysis of the findings to the improvement ideas that achieve the project's goals (van Eck et al., 2015). The outputs of this stage are improvement ideas for the current processes or new analysis questions. Validation and verification techniques of the results are also applied here.

2.10.6 Process Improvement and Support Stage

The objective of this stage is to use the gained insights to modify the actual process execution (van Eck et al., 2015). The inputs to this stage are improvement ideas from the evaluation stage and the output are process modifications.

2.11. Chapter Summary

This chapter gave an overview of PM, its characteristics, its three different types, and common tools. In addition, this chapter gives general information about cloud applications, cross-organizational PM, and process pattern recognition techniques. Lastly, it discusses the six different stages of the process mining project methodology, emphasizing that this thesis contributes mainly to the data preprocessing stage for event logs produced by cloud-based applications.

The next chapter proposes a SLR that summarizes the existing applications of PM for cloud-based application processes, together with specific challenges for which this thesis proposes solutions.

Chapter 3. Systematic Literature Review

This chapter presents the results of a systematic literature review (SLR) conducted to investigate the context of PM in cloud applications. The use of clickstream data is also investigated here as this data often represents how users behave while using such online applications. 27 papers are rigorously selected, reviewed and analyzed. This chapter was published in:

El-Gharib, N.M., & Amyot, D. (2019). Process mining for cloud-based applications: A systematic literature review. *9th International Workshop on Model-Driven Requirements Engineering (MoDRE)*, IEEE CS, 34–43.

3.1. Literature Review Methodology

A systematic literature review was undertaken based on the guidelines of Kitchenham et al. (2007). The research questions, search query, and inclusion/exclusion criteria are presented here.

3.1.1 Research Questions

The objective of this research is to explore how PM is being applied on cloud-based application processes, with a particular attention paid to the issue of user behavior described from data, which is especially relevant in a requirements elicitation context. There are two research questions:

- **SLR-RQ1:** How is process mining utilized on cloud-based systems?
- **SLR-RQ2:** How is process mining being applied to understand user behavior from event logs in online applications?

The first research question deals with identifying studies on how PM techniques and algorithms have been used to study cloud-based application processes. For example, what PM techniques have been applied on event logs that are generated by cloud applications, or in

what areas did this happen. The second question aims to discover how PM techniques and algorithms are being applied on clickstream event logs of various online applications. Both questions aim to discover challenges and PM tools that are used together with challenges specific to the cloud context.

3.1.2 Search Process and Query

In this SLR, three of the most relevant search engines in information technologies were used: Scopus, IEEE Xplore, and Web of Science. Scopus and Web of Science are general databased that cover most of the papers in the technology field. Scopus indexes more journals and conferences, but Web of Science has a better coverage of older papers (before 1996). IEEE Xplore is added to cover very recent papers that are not indexed yet by the other two engines, as well as less-known venues such as peer-reviewed workshops.

The main query (performed in January 2019) is looking for papers related to PM and cloud applications:

("process mining" OR "process discovery")

AND

("cloud" OR saas OR "click stream" OR "user behavi*")

within articles' titles, abstracts, and keywords, across the three databases. SaaS (Software as a Service) is the cloud layer where applications reside. Conference reviews and editorial material were excluded from the results. No specific time limit was used. This led to 162 papers (Scopus returned 133 papers, IEEE Xplore returned 32 papers, and Web of Science returned 53 papers, with duplicates). After removing duplicates, 89 unique papers were returned across the three databases.

A second query:

("process mining" OR "process discovery")

AND

("cloud" OR saas OR aws OR "Amazon Web Service" OR azure OR "click stream" OR "user behavi*")

was used to check whether there were additional papers when adding the two main cloud providers to the query. The same number of papers as the first query was returned, suggesting that adding the names of specific platforms does not improve the results.

We also searched for existing literature reviews on how PM technologies are applied to cloud applications, without relevant results.

3.1.3 Inclusion and Exclusion Criteria

The inspection of the 89 papers led to several exclusions based on the following criteria:

- Studies that referred to how cloud computing technologies are used to support PM activities (e.g., for cloud-based PM tools).
- Studies that did not focus on how PM techniques are used to understand cloud application processes.
- Studies that did not focus on how PM techniques are being applied to understand user behavior in online applications.
- Studies that were not written in English.

No inclusion criteria (e.g., based on snowballing) were used. A simple quality assessment ensured that the papers were not published in predatory journals or venues.

3.1.4 Final Selection

A total of 27 papers, listed in Table 2, were selected for this literature review paper based on the criteria of the previous section. All returned papers were published after 2008. Their distribution over years is shows in Figure 7. Most of the papers in this area were published in the past two years.

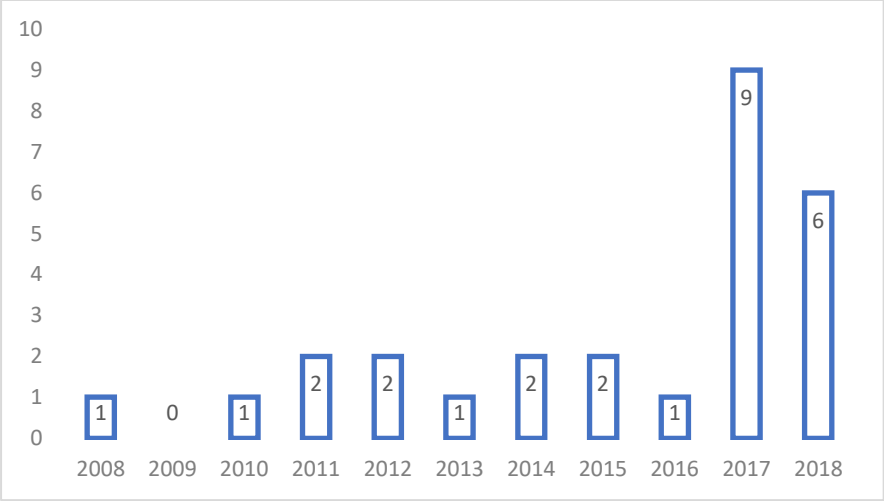

Figure 7: Number of selected papers published per year.

3.2. Review of Selected Studies

After reviewing and analyzing the selected papers, and identifying their key points, three themes became apparent: techniques and algorithms, PM tools, and validation techniques.

3.2.1 Techniques and Algorithms

Some of the most popular process discovery algorithms include the following (van der Aalst, 2011):

1. α-algorithm: takes an event log as an input and generated a Petri net (with concurrency) displaying the behavior included in the event logs.
2. Heuristic Mining: uses a representation similar to causal nets (without concurrency) and takes frequencies of events and sequences into account when constructing a process model. The basic idea is that infrequent paths should not be incorporated into the model.

Each paper discusses a certain process mining technique and such techniques can be applied in cloud-based applications. One proposed technique is a cross-organizational PM approach for cloud computing multi-tenant environments. The cross-organizational PM approach focuses on the integration of multi-tenant architecture of techniques (Bernardi et al., 2015; 2018). The approach of Bernardi et al. (2015) was used to discover declarative process models that represent the variants of a family of business processes as sets of

business rules. Logs can be extracted from process instances running on the process engines to aggregate an event in a centralized event log store (Bernardi et al., 2015). Then, queries can be run to stream events related to one of the variants of the processes followed by a tenant to an online process minter to get more precise knowledge about the variant.

Further work by Bernardi et al. (2018) aims to exploit additional information that is specific to business processes. The considered logs allow recovering the models of the process being executed and extracted useful metrics. The monitoring engine exploits the results obtained from the Online Declare Miner (Bernardi et al., 2018).

Another technique by Buijs et al. (2012) aims to compare collections of process models and their event logs from cross-organizational PM. This technique focuses on analyzing process models and event logs according to three different types of metrics: quality metrics, performance indicators, and comparison.

Best Analytics of Big Data (BAB) is an open-source data analytics tool that combines Hadoop Map-Reduce algorithms with cloud computing and PM to handle the data explosion in business fields (Sutrisnowati et al., 2015). Sutrisnowati et al. (2015) developed BAB's architecture, which consists of three layers for visualizations, analytics, and storage. The analytics layer consists of description, predictive, and perspective layers. BAB uses a four stage, PM-based descriptive analytics approach. First, a data summary stage is created by statistical information extracted from the data dimensions, states, etc. The second stage is concerned with a process model discovered using a process discovery algorithm. The third stage is a performance quality assessment of the discovered process model and is evaluated using conformance checking algorithms. The last stage includes process analysis algorithms used to analyze historical data.

Another use of PM to understand cloud processes is to evaluate software development processes themselves by analyzing how developers use their Integrated Development Environment (IDE). By using PM techniques, for example, discovering processes and checking conformance can help in detecting programmers' coding patterns, behaviors, and

deviations from prescribed processes (Caldeira & Abreu, 2016). PM of IDE event logs was used to provide software developers with higher awareness of their development processes and enhancement opportunities.

PM can also be used in detect errors such as in the *POD-Diagnosis* (Process-Oriented Dependability) approach, which was proposed to model sporadic operations as processes (Xu et al., 2014). Modeling such processes can help determine the order in which processes are executed, as well as use the process context to filter logs, trigger assertion evaluations, visit fault trees, and perform evaluations for online diagnosis and root cause analysis. Xu et al. (2014) evaluated their approach using the Amazon Web Services cloud provider.

Amazon Kinesis (Evermann et al., 2017), a cloud-based event stream infrastructure, was used to evaluate a distributed, streaming implementation of the *Flexible Heuristic Miner* (FHM). Evermann et al. (2017) focused on the separation of the FHM algorithm into individual processing stages, which are suitable for a cloud-based scalable event stream infrastructure, and then distributed these stages to different computation nodes on the cloud infrastructure.

Song et al. (2017) proposed the *Scientific Workflow Mining as a Service* (SWMaaS) solution, which was implemented as a tool for Scientific Workflow Mining in Clouds (SWMC). PM techniques cannot easily be used to study and analyze scientific workflows. Song et al. proposed an approach that supports both types of cloud scientific workflow mining: intra-cloud and inter-cloud workflows.

PM techniques have also been used in healthcare (Ghasemi & Amyot, 2016). A framework of the *M-Health Monitoring System* (MHMS) (Xu et al., 2017), based on a cloud-computing platform, is used to support remote health monitoring, addressing health data integration and interoperation among various medial agencies. PM was integrated into such approach to support treatment selection (Xu et al., 2017). The architecture of the cloud-computing based MHMS is divided into three different layers: 1) cloud storage and multiple tenants access control layer, 2) healthcare data annotation layer, and 3) healthcare data

analysis layer. The PM algorithm, included in the third layer, is used to induce clinic paths from personal healthcare data. The health data analysis model includes two-parts: clinical PM and similar patient searching. The α-algorithm was used for clinical PM to obtain disease treatment paths from historical cases (Xu et al., 2017).

Much of the literature has exploited some techniques and tools developed in the research field of Business Process Management. The focus is often on declarative languages and tools for monitoring how business processes are being executed (Chesani et al., 2017). Chesani et al. (2017) focused on monitoring Map-Reduce applications and performing recovery actions on a cluster. They introduced an architecture where a logic-based monitor is able to detect possible delays and trigger recovery actions.

PM was adopted as a knowledge extraction technique for producing Quality of Service-sensitive customer behavior model graphs. PM is used to study the clickstreams of a user's navigation on a website (Ghavamipoor et al., 2017). PM is also applied to compute the situations of process activities that occur in the cloud. An expandable and stable cloud that needs to scale up and down can be achieved by online PM, and Haung et al. (2011) observe that several concerns should be taken into consideration in that context:

1. Adaptability, for extending the capability of the cloud process activity traces on demand.
2. Stability of transferring reliable activity traces of cloud information and applications.
3. Functionality of the cloud process activities.
4. Scalability, for reducing or enlarging the size of the composition of web services in a cloud.

Bernardi et al. (2018) used a constraint-based declarative process model called *Declare* to represent the business process variants that are generated and collected from the execution of a task on a distributed system. Declare is different from traditional process modeling languages in that it specifies the set of rules and constraints that must be satisfied during

the execution of a process. Declare was also used by Bernardi et al. (2015), Buijs et al. (2012), and Chesani et al. (2017).

Terragni and Hassani (2018) applied PM techniques to analyze user behavior based on events collected from the web. Their goals were to: 1) discover some processes that can illustrate user behavior, 2) extract useful information, 3) compare the processes of the different users' clusters, and 4) discover new key performance indicators. Terragni and Hassani (2018) followed several steps to build their framework: 1) data preprocessing, where each URL is mapped to a specific action, 2) filter the web logs, 3) discover the process and perform the analysis, 4) cluster users according to their behavior on the web, and 5) propose recommendations, followed by an evaluation.

Janes (2017) used PM technique to understand how users interact with a system. The users were programmers and their main goal was to mine event logs and find out which parts of the system are mainly used based on observable user behavior. The main activities of interest involved the generation of test cases by programmers, so they can discover the process of generating tests and how to prioritize their generation.

Poggi et al. (2013) applied Business Process Management (BPM) methodologies to analyze user behavior on an e-commerce website. They focused on generating process maps for user actions to understand the workflow through the different pages of the website that might lead users to either purchase a product or exit the website prematurely. They used three different datasets by saturating, clustering, and biasing the original dataset. They used URLs to classify the different events of the users, which helped them generate a representation of the users' navigations. Poggi et al. applied different algorithms (alpha miner, heuristic miner, and fuzzy miner) to perform the analysis.

Padidem et al. (2017) performed a study of users shopping behavior processes using clickstream datasets. They used the α-algorithm to generate a Petri net that is then analyzed by heuristic miner and fuzzy miner algorithms. They studied the use of several metrics

such as the fitness of the discovered model, quality, relevant event traces, as well as behavioral and structureal appropriateness (Padidem & Nalini, 2017).

Sahlabadi et al. (2014) proposed another approach that uses PM technologies to detect abnormal behavior in social networks through a fitness function compared to the normal behavior. Song et al. (2008) highlight the issues, challenges, and approaches related to behavior pattern mining. They focus on behavior process mining as a sub-field in process mining. They conclude that existing algorithms in behavior pattern mining should be tested with real world data as future work. The four stage in a process mining project (events recording, preprocessing, mining and discovering, and verification) can be used in a behavior pattern mining project.

3.2.2 Process Mining Tools

The ProM framework developed by van Dongen et al. (2005), is a PM environment extensible through plugins. This framework is flexible in terms of input and output formats and it allows reusing code through the implementation of PM algorithms.

ProM and its plugins represent the tool most frequently used (17 times) by the 27 papers of this SLR. ProM was used to execute the Heuristic Mining Algorithm by Ghavamipoor et al. (2017), and Padidem et al. (2017) used ProM to run the alpha, heuristic, and fuzzy algorithms.

Terragni and Hassani (2018) used the commercial Disco tool to discover different user behaviors from web logs. Additionally, Jane (2017) used Disco to discover the processes related to the generation of test cases by programmers.

In nine papers (Bernardi et al, 2018; Caldeira & Abreu, 2016; Xu et al, 2014; Evermann et al, 2017; Chesani et al, 2017; Haung et al, 2011; Janes 2017; Kalenkova et al, 2018; van der Aalst 2011c) out of the 27 papers the authors developed their own algorithms to apply process mining analysis.

3.2.3 Validation Techniques

Four papers used case studies to discuss the use of process mining technologies in cloud environments (Caldeira & Abreu, 2016; Terragni & Hassani, 2018; Sahlabadi et al., 2014; Liu et al., 2018). A dataset collected from web logs was used in (Terragni & Hassani, 2018). This dataset contains clickstream information for one month, with 10 million events capturing the activities of 2 million users. A dataset from Atrapalo (an online application that offers flights packages), was used by Terragni & Hassani (2018). This dataset consists of clickstream information from visitors and customers of the different products that are offered by this e-commerce website.

Seven papers used clickstream data to test and apply their proposed approaches (Ghavamipoor et al., 2017; Janes, 2017; Poggi et al., 2013; Sahlabadi et al., 2014; Song et al., 2008; Liu et al., 2018; Kalenkova et al., 2018). Data from e-commerce websites were used in some of them. For example, Ghavamipoor et al. (2017) used an e-store website in the analysis of the proposed model. This data includes customer actions related to searching for products, viewing catalogs and requesting products, adding items to shopping cards, and placing their orders. Ghavamipoor et al. (2017) used clickstream datasets collected between June 2012 and September 2013. They were able to extract different customer behaviors from 200,000 records and 10,000 buy and visit sessions.

Six papers (Bernardi et al., 2015; Bernardi et al., 2018; Buijs et al., 2012; Sutrisnowati et al., 2015; Liu et al., 2018; van der Aalst, 2011), discussed how to use process mining and integrate its technologies in cloud environments. The cross-organizational model was applied on real-life examples from the CoSeLoG research project, which investigates how ten Dutch municipalities execute and run their processes (Buijs et al., 2012). Another validation technique involved process data about acknowledging an unborn child. This particular process describes a procedure to be followed when a citizen wants to register a child still to be born (Bernardi et al., 2015).

An experiment was applied to the execution of four process variants of a business process on four different tenants, each following a different process variant of a process family

(Bernardi et al., 2018). In the same study, these processes have the same control flows; however, the execution order of activities varies for the four stores. Each process describes a procedure to be followed when a customer wants to buy some product from online stores. The tenant variants that were studied are invoices, discount codes, reviews, and desired list of products.

Song et al. (2017) used experiments to validate their intra-cloud and inter-cloud scientific workflow mining approach. They considered physical machines belonging to the same cloud and they worked on discovering the corresponding sub-flows in parallel. They also performed a comparison between traditional PM and their proposed approach.

Acampora et al. (2017) evaluated their proposed architecture using a case study. Their approach aims to help with the distribution of resources in a multi-tenant environment. The data that Acampora et al. used was collected from an online shopping website. They also evaluated four process variants for four different tenants.

3.3. Summary and Discussion

3.3.1 Summary

PM allows organizations to get a process perspective of their data, which they can then analyze, leading to the potential discovery of hidden issues supporting the crowed-based elicitation of new system requirements. PM for cloud application is becoming popular, as the number of research papers in this area has been increasing substantially in the last two years (Figure 7). Based on the papers that we reviewed in this SLR, we can see the value of PM being applied to cloud applications.

Table 2 summarizes the 27 papers in the literature review as an attempt to answer SLR-RQ1. The summary is presented along 7 criteria that are selected from the analysis of papers (C1 – C7). The criteria are as follow:

- C1: The paper uses ProM as PM tool.
- C2: The paper uses Disco as PM tool.

- C3: The paper discusses challenges observed while analyzing clickstream event logs; this includes all the data preprocessing that should be done before importing data in the PM tools.
- C4: The paper uses an E-commerce website for the analysis of clickstream datasets.
- C5: The paper discusses business process management.
- C6: The paper is focused on applying PM to software development and in the software engineering field.
- C7: The paper is focused on applying PM technologies to case studies and analyze cloud infrastructure processes.

As shown in Table 2 , for C1, 17 papers use ProM as their main PM tool. They have either developed their own plugin or used existing ones. Only two papers used Disco (C2) to generate process maps for their business processes. The main reason is likely that ProM is an open source tool and allows its users to develop their own extensions. In the nine remaining papers, the authors have implemented and used their own tools to apply process mining techniques.

For C4, only five papers out of the 27 looked at analyzing clickstream data collected from e-commerce websites, which were mainly looking to understand how customers are navigating on the different pages of the website. For C5, 12 papers looked at business process management, and their focus was on business process discovery. Only three papers apply PM techniques in the cloud-based software development fields (C6). In these papers (Caldeira & Abreu, 2016; Janes, 2017; Liu et al., 2018), the authors applied PM to study the behavior of software developers and looked at patterns in development processes. Regarding C3, only two of the papers discussed challenges that are faced when preprocessing clickstream data (Terragni & Hassani, 2018; Song et al., 2008).

Table 2: Summary of related works. Paper (T)ypes are (J)ournal, (C)onference, and (W)orkshop.

Paper	Title	Authors	Year	T	Concept						
					C1	C2	C3	C4	C5	C6	C7
P1	Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining.	van der Aalst	2010	C	Y				Y		Y
P2	Discovering Cross-Organizational Business Rules from the Cloud.	Bernardi et al.	2015	C	Y				Y		Y
P3	Cross-Organisational Process Mining in Cloud Environments.	Bernardi et al.	2018	J							Y
P4	Towards Cross-Organizational Process Mining in Collections of Process Models and their Executions.	Buijs et al.	2012	C	Y						Y
P5	BAB Framework: Process Mining on Cloud.	Sutrisnowati et al.	2015	C	Y	Y					Y
P6	Software Development Process Mining: Discovery, Conformance Checking and Enhancement.	Caldeira et al.	2016	C						Y	
P7	POD-Diagnosis: Error diagnosis of Sporadic Operations on Cloud Applications.	Xu et al.	2014	C							Y
P8	Process Discovery from Event Stream Data in the Cloud – A scalable, distributed implementation of the flexible heuristics miner on the amazon kinesis cloud infrastructure.	Evermann et al.	2017	C							Y
P9	Scientific Workflow Mining in Clouds.	Song et al.	2017	J	Y						
P10	The Design of an m-Health Monitoring System Based on a Cloud Computing Platform.	Xu et al.	2017	J	Y						
P11	Map Reduce Autoscaling over the Cloud with Process Mining Monitoring.	Chesani et al.	2017	C					Y		Y
P12	A QoS-Sensitive Model for e-Commerce Customer Behavior.	Ghavamipoor et al.	2017	J	Y			Y	Y		
P13	Perspectives on Process Mining within Cloud Computing.	Haung et al.	2011	C							Y
P14	Analyzing Customer Journey with Process Mining: From Discovery to Recommendations	Terragni et al.	2018	C		Y	Y	Y	Y		
P15	Test Case Generation and Prioritization: A Process Mining Approach.	Janes	2017	W						Y	
P16	Business Process Mining from e-commerce Web Logs.	Poggi et al.	2013	C	Y			Y	Y		
P17	A Study on Classification of Users Shopping Behavior Process Model Using Click Stream Data.	Padidem et al.	2017	J	Y			Y			
P18	Detecting Abnormal Behavior in Social Network Websites by using a Process Mining Technique.	Sahlabadi et al.	2014	J	Y						
P19	Behavior Pattern Mining: Apply Process Mining Technology to Common Event Logs of Information Systems.	Song et al.	2008	C	Y		Y	Y	Y		
P20	A Two-Layered Framework for the Discovery of Software Behavior : A Case Study.	Liu et al.	2018	C	Y					Y	
P21	User Behavior Discovery from Low-Level Software Execution Log	Liu et al.	2018	J	Y					Y	
P22	E-Government Services : Comparing Real and Expected User Behavior.	Kalenkova et al.	2018	C					Y		
P23	Business Process Configuration in the Cloud: How to Support and Analyze Multi-Tenant Processes?	van der Aalst	2011	C					Y		Y
P24	A Fuzzy-Based Autoscaling Approach for Process Centered Cloud Systems.	Acampora et al.	2017	J	Y				Y		Y
P25	A Fog Computing Based Concept Drift Adaptive Process Mining Framework for Mobile APPs.	Haung et al.	2018	J	Y				Y		Y
P26	Discovering and Exploring State-Based Models for Multi-perspective Processes.	van Eck et al.	2016	C	Y						Y
P27	Distributed Process Discovery and Conformance Checking.	van der Aalst	2012 b	C	Y				Y		

Regarding C7, 13 of the 27 papers can be classified as papers that looked at cloud system processes. The focus is on how to integrate PM tools and techniques into a cloud infrastructure. These papers discussed cross-organizational PM that study a family of process models, with related challenges.

Sutrisnowati et al. (2015) used ProM and Disco as the PM tools to compare them with the BAB tool that they have developed.

Song et al. (2008) mainly discuss behavior pattern mining, together with issues, challenges, approaches and tools used to analysis such processes. Kalenkova et al. (2018) aimed to develop an approach to compare process models when applied to real-life data collected from government services.

3.3.2 Discussion

The previous section provided many faceted answers to SLR-RQ1 on how PM is used on cloud applications. Regarding the specific applicability of PM to the analysis of user behavior in cloud applications (SLR-RQ2), most of the papers that used PM techniques to analyze clickstream data were looking at e-commerce applications, and only three papers looked at software development logs. PM analysis can also be applied to different online applications from different areas, for example visualization and business intelligence applications. Additionally, in social network applications. For example, PM analysis can help with user interface analysis for any type of applications. PM analysis generate maps for the whole process from start to finish, which plays an important role in the analysis to the different features of the application.

When analyzing clickstream data many challenges are faced (SLR-RQ2). Clickstream data often include *all* the clicks that users produced on websites. Depending on the analysis questions, a large data preprocessing effort is required before importing the datasets to the PM tools. More challenges should have been discussed in the papers.

Acampora et al. (2017), focused on explaning the architecture when integrating the Online Declare Engine Miner in the cloud infrastructure. The focus was only on the resource management distribution. They used *Declare*, which is the declarative process modeling language. Their approach helped them discover four Declare models that are used in the analysis of the activites that are executed on each tenant. Liu et al. (2018) proposed an apporach to analyze the real behavior of ProM while excecuting any of its plugins.

van der Aalst (2011) was the first to discuss the challenges of applying process mining analysis to multi-tenant processes in the cloud environment. Some of the highlighted challenges were related to configurable models. Those challenges include that, with real data, it is difficult to represent configurable models while considering all the perspectives in a process. Additionally, analyzing these families of processes is more challenging since the traditional techniques focus on analyzing a single process model. Finally, having configurable process models makes it more challenges to learn from these models and to apply conformance checking techniques.

Buijs et al. (2012) highlighted another challenge, which is about analyzing across several organizations a large collection of the same process model.

van der Aalst (2010) showed that analyzing process variations *across* multiple organizations creates new challenges, and he mentioned several concerns when studying a family of process models. The first concern is how to ensure that each organization has its unique configuration of a process. The second concern is how to trust that the configurable model is correct since it is needed to ensure that all the configurations of the configurable model are correct and represent what is really happening in a process model. The third concern is security, i.e., how to ensure that the processes and the data of the different tenants are secured and isolated from each other (van der Aalst, 2010).

Additionally, and in particular with respect to process configurations in configurable models, van der Aalst (2010) highlighted some of these challenges: 1) the development of a complete collection of high-quality configurable models is essential, 2) the extraction of a

manageable configurable process from a set of models, given that the configurable models including all the perspectives of a process model will likely generate spaghetti processes that are challenging to analyze, and 3) changing a configurable model that is used by multiple tenants, and how to deal with a special process that does not fit a configurable model.

To answer SLR-RQ1 on how process mining is being utilized in cloud-based systems, based on the 27 papers cited in this SLR, several observations have been made. First, the number of papers on process mining applications in cloud systems has been increasing since 2017. The area of cross-organizational process mining is still rather unexplored, and much work remains to be done in that area. As the number of applications that are migrating to cloud infrastructure is increasing, more processes are being generated across all the services that are integrating together. With the introduction of configurable process models and cross-organizational process mining, additional challenges and concerns are introduced.

To answer SLR-RQ2, process mining technologies have been applied on data generated from e-commerce applications to understand how users are navigating websites and which pages are visited more frequently, leading to some outcome (e.g., the buying of a product). This helps analyze the start-to-end processes of user actions. Also, in the software development area, process mining technologies were used to study and assess developers' techniques as well as the behavior of software features.

This SLR hence highlights how PM techniques are being applied in cloud-based applications and how process mining can help businesses better understand the behavior of users exploiting their applications and services. There is a growing interest in process mining research for cloud processes, especially as more businesses are moving to cloud infrastructures.

3.3.3 Challenges

Applying process mining techniques on cloud-based application event logs comes with many challenges, identified explicitly or implicitly in the 27 papers reviewed. The processes for users using cloud applications are less structured than conventional processes in

an industrial context (e.g. an employee working on an assembly line) or in a healthcare context (e.g., a patient following a clinical pathway with coarse-grained events). Users will often click on menus/buttons in different orders, back and forth, while searching what to do/buy, changing their minds several times, or even abandoning the process along the way. The current process mining algorithms do not handle such type of processes very well and more preprocessing for the data is required.

van der Aalst (2012a) is a paper that is representative of many other papers in that field and it summarizes several challenges for applying PM in practice, for systems in general:

1. Merging and cleaning raw event log datasets.
2. Dealing with complex event logs having a variety of characteristics.
3. Mining processes that change over time.
4. Applying PM analysis on event logs in real time.
5. Improving the representation of the process maps for less structured processes.
6. Combining PM analysis with other types of analysis.

The first five of these conventional challenges are often amplified in cloud computing context. In particular:

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 carefully select/merge attributes at preprocessing time.
3. Cloud-based processes typically evolve more rapidly than conventional processes, especially in a DevOps context where an application can be improved and deployed several times per day. Different types of events and 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 (as explained above), and given that the proportion of incomplete processes is much higher in cloud-based application than in conventional processes, the negative impact on the readability of mined process models is quite high.

In addition, I observed the following cloud-specific challenges after exploring real clickstream datasets:

6. Dealing with noisy event logs (where too many events are tracked).
7. Dealing with similar events that must be split according to some of their attributes into different events (as much information is stored in the attributes for very generic event/activity names).
8. The absence of reliable unique identifier (as there are many anonymous users using such applications).
9. The frequent change and evolution of monitoring and trace management tools in cloud and DevOps environments, which in turn result in changes in the nature of the collected logs.
10. 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 event click-refresh) in order to simplify the resulting processes.

Regarding the last point, to efficiently analyze processes for user behavior on an application, it is needed to find usage patterns in such processes in order to simplify and improve the discovery of process models. Finding patterns for cloud application logs and substituting these patterns with simpler and aggregated events create additional challenges in preprocessing such logs. Since cloud logs may include many columns that represent properties of such processes, when aggregating different events into one event, there should be a clearly-defined mechanism to aggregate the properties as well since they might be needed in the analysis. Bose and van der Aalst (2009), worked on exploring common process models for more flexible processes. Their approach focused on discovering patterns in traces from real logs and substituting these patterns with abstract

activities which is part of the preprocessing event logs before generating process maps. However, again, the need for such patterns is amplified in a cloud context.

The available PM tools do not provide solutions for the above problems. Currently, ad hoc and fragile mechanisms (often manual) are used by PM analysts. There are many opportunities for the research community to tackle the above challenges with novel mechanisms and tools.

3.4. Limitations and Threats to Validity

According to Perry et al. (2000) three types of validity threats are discussed in this paper: 1) construct validity, 2) internal validity, and 3) external validity.

Construct validity refers to the quality of the methodology in terms of being helpful to answer the research questions. Even though we included frequent and essential keywords in the queries, and ran these queries on three large and popular databases, there is a possibility that some relevant papers have not been found and ended up not being included in this review. No manual search (e.g., based on snowballing or expert interviews) was used to complement the automated query. More recent papers (in 2019) may also have appeared since this SLR was performed.

Internal validity examines any bias in performing the research. The major threat to internal validity here is that the review of the literature and the coding of the information contained in the selected papers was done by only two people, one with three years of experience in process mining and one with only one year of experience. Some important aspects might have been missed or misrepresented.

External validity considers whether applying the conclusion of this study and the results to other cases or situations is possible. The focus of this SLR is limited to the use of process mining in cloud applications and in discovering user behavior based on event log data collected from online applications. The results and challenges discussed here were based specifically on papers related to these topics and should not be generalized outside that scope.

In particular, the results reflect what the research community has published and may be different from actual industrial practices.

3.5. Chapter Summary

This chapter presented a systematic literature review on the application of PM technologies on cloud processes. As the number of cloud-based applications is increasing (given the management and scalability benefits of such platforms), studying the behavior of users of such applications is becoming a task critical for the success of businesses. By moving to the cloud, these businesses generate many processes within the business itself and in relation to other businesses. From a requirements elicitation perspective, PM offers automated mechanism to model as-is processes rooted in crowd-based event logs.

This SLR discussed and analyzed 27 peer-reviewed papers rigorously selected from automated searches. The contributions of this review include the identification of PM technologies (algorithms, tools, and validation approaches) currently used in the analysis of logs from cloud applications (SLR-RQ1), with special attention given to user behavior recovery (SLR-RQ2). These two points also answer RQ1 of this thesis, defined in section 1.2. In addition, this chapter highlights important challenges this unique context brings to the table, hence answering RQ2 of this thesis. The PM research community is invited to pay further attention to these challenges in the near future as cloud-based applications will play an increasingly important place in our daily lives.

The next chapter will describe the research methodology used for the remainder of the thesis.

Chapter 4. Design Science Research Methodology

In order to conduct effective research, a suitable research methodology should be selected and applied in order to answer the research questions and solve the defined problem. As described in section 1.2, the problem in this thesis involves developing a method to analyze and exploit event logs collected from Information Systems (cloud-based applications) to answer business questions about such processes, Design Science Research in IS has been chosen as the research methodology in this thesis.

This chapter explains what is the Design Science Research Methodology (DSRM) and what are the five steps followed in this thesis's research.

4.1. DSRM Overview

According to Hevner et al. (2004), design science research involves two main activities to understand IS: creating new knowledge through the design of new artifacts and analyzing the artifacts' usage and performance. The design science research methodology in IS involves two complementary paradigms: *behavioral design* and *design science*. Behavioral design aims to develop and verify theories related to human organizational behavior. It aims to study the interaction between people, technology, and organizations. On the other hand, design science has its roots in engineering. It aims to discover and create new artifacts. The design science paradigm is based on 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.

To create and evaluate a method to study and characterize cloud-based processes from clickstream data, this thesis instantiates the Design Science Research Methodology.

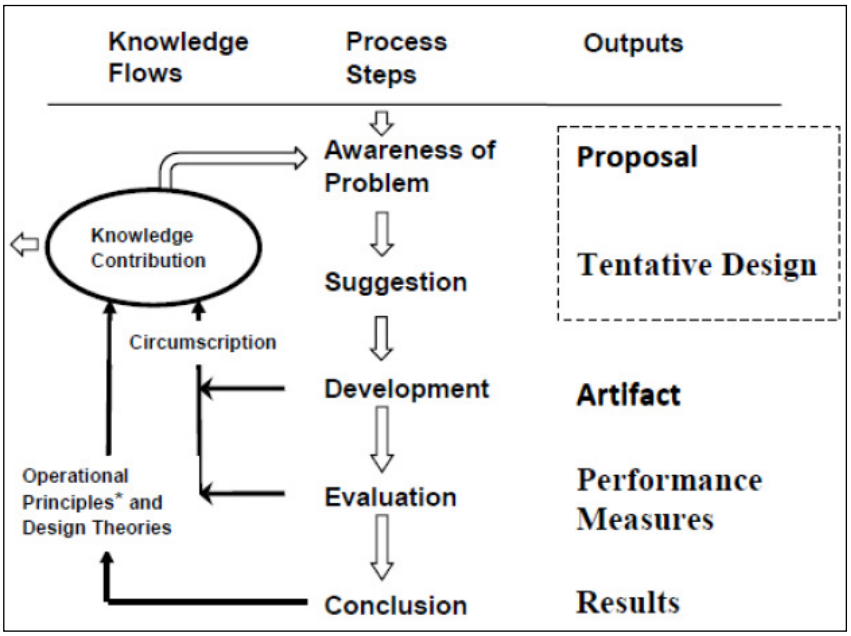

Figure 8: Design Science Research Process Model (Vaishnavi, 2004/19)

As shown in Figure 8 above, the DSRM conceptual process model consists of iterative five steps, which are tailored to perform this thesis's specific research.

4.2. Use of DSRM in this Thesis

Each of DSRM's five steps is detailed below, together with how it was tailored to the context of this thesis.

4.2.1 Awareness of the problem

In this phase, the researcher defines the specific research problem and justifies the value of this research and the proposed solution. The awareness of the research problem comes after conducting a literature review and new applications development in the industry. The output of this phase is a research proposal for a new research problem.

In this phase, I conducted a SLR (previous chapter) on how process mining techniques have been applied in cloud-based applications, together with specific challenges that remain in that area. I defined the research proposal and the solution to the selected problem, which is about developing a method to pre-process event logs from cloud and SaaS

applications in order to support understanding and analyzing their processes. Input from key informants of a SaaS application provider was also informally taken into consideration. The proposal was developed with an overview about the questions that can be answered by this method and what benefits it brings to businesses in answering questions about their existing processes.

Figure 9 highlights the relations between the thesis research questions (Section 1.2) and the SLR research questions (Section 3.1.1). The scopes of RQ1 and SLR-Q1 are the same, i.e., how are process mining discovery techniques used on cloud-based systems (SaaS, but also PaaS and IaaS). SLR Q2 is more focused on user-based behavior, for SaaS applications. The challenges discussed in the SLR to answer RQ2 span both cloud and non-cloud applications. The method proposed in this thesis (answering RQ3) addresses a subset of these challenges. Although it is validated on a SaaS application in this thesis, the method also likely applied to other contexts.

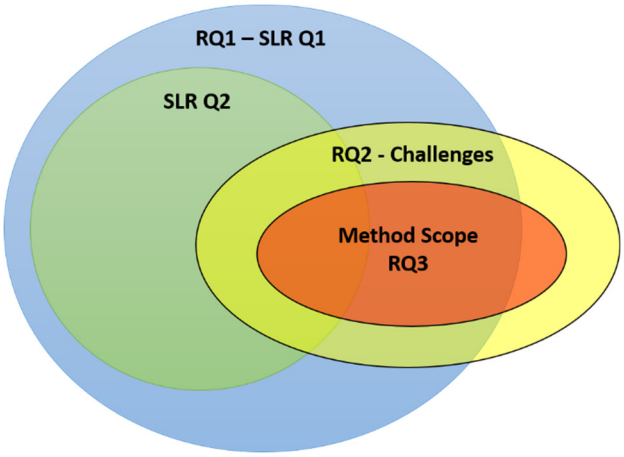

Figure 9: Relations between thesis research questions and SLR research questions

4.2.2 Suggestion

In this phase, from the defined problem and knowledge of what is possible and feasible, I 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, the proposed method and its different steps and stages are defined with the objective of each stage, and the solution was illustrated with an example.

4.2.3 Development

In this phase, the researcher creates an artifact, which is the solution to the defined problem. The artefacts developed here include an API (with an implementation) and a method that exploits it. 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.

The proposed method for preprocessing cloud-based event logs is developed and implemented as the solution to solve the problem. The method consists of several phases and each phase is composed of several steps, which can exploit automated functions. The different functions of the method are regrouped in an API, for which an implementation developed in the R programming language[7] was developed. R was selected as this is a common data manipulation and analysis language, and it was also already used extensively by the SaaS company. Each step in the solution method might go through several iterations to answer the analysis questions. Further details about the proposed method are provided in Chapter 5.

4.2.4 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 the its importance. The evaluation method that was used in my research is a case study. 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

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

performed on the dataset in order to answer the questions about the business processes. At the thesis level, we also use two iterations based on two versions of an event log (taken at different times) to assess the complexity of adapting scripts in the context where the structure of logs changes (e.g., with columns added or removed, new or renamed events, etc.). The case study is presented in Chapter 6.

4.2.5 Conclusion

This phase is the last step in the research process. During this step, the results of the research are finalized, and it indicates the end of a research cycle. This step concludes the research projects and presents the results. One paper was already published (El-Gharib and Amyot, 2019), and this thesis represents a second communication artifact. I expect another publication to be extracted from the thesis, about the method and its implementation and evaluation.

4.3. Chapter Conclusion

The DSRM provides a suitable methodology for carrying out design science research in IS. This chapter described how DSRM was used as a methodology and what were the steps taken at each of its stages.

The next chapter presents the *Cloud Pattern API-Process Mining* (CPA-PM) method, its different steps, and how it should be applied.

Chapter 5. Cloud Pattern API - Process Mining Method

In Chapter 2, the six steps of a process mining project methodology were illustrated. This chapter describes a new *Cloud Pattern API - Process Mining* (CPA-PM) method. This method is part of the data preprocessing step in a process mining project. The objective of this method is to preprocess event logs collected from cloud-based applications so that useful insights can be concluded. The method is supported by a collection of related functions forming an API, with one implement provided (in R). Such an API enables automation through reusable scripts, independently of the PM technology used to discover, visualize, and analyze processes.

5.1. Overview of CPA-PM

CPA-PM method is a method for preprocessing event logs collected from cloud-based applications and replacing patterns before applying process mining techniques on the event logs. It aims to help preprocess less structured event logs that are collected from cloud applications, for example clickstream datasets. The CPA-PM method consists of three main steps and each step has sub-steps that will enable to preprocess cloud-based application event logs, which usually include a large number of columns that need to be taken into consideration when aggregating rows in the dataset. The input for this method is an event log dataset (CSV) and the output is a cleaned event log (CSV) that can be imported by process mining tools for building process maps and enable further analysis. The single input CSV file can itself be the result of prior extraction and processing from multiple data sources where, for example, conventional technologies such as *Extract, Transform, and Load* (ETL) and their tools can be used (see Figure 10). The API in CPA-PM can have functions that overlap those found in ETL tools, but it will also contain functions specific to the cloud and process mining contexts that are not typically found in ETL solutions (de Murillas et al., 2019). The steps of the CPA-PM method might go through several iteration to get answers for analysis questions.

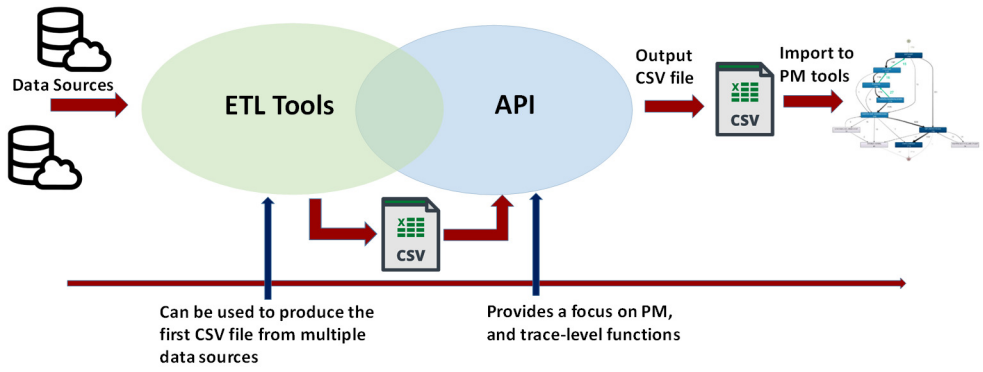

Figure 10: Where the CPA-PM method and its API fit.

5.2. Data Preprocessing

The main objective of this stage is to prepare the event logs dataset to be fed into the process mining tools to build process maps. The CPA-PM method is applied to the dataset during the data preprocessing stage of the process mining project. The CPA-PM method consists of three different major steps: 1) *Clean-up*, 2) *Restructuring*, and 3) *Pattern Substitution*, illustrated in Figure 11.

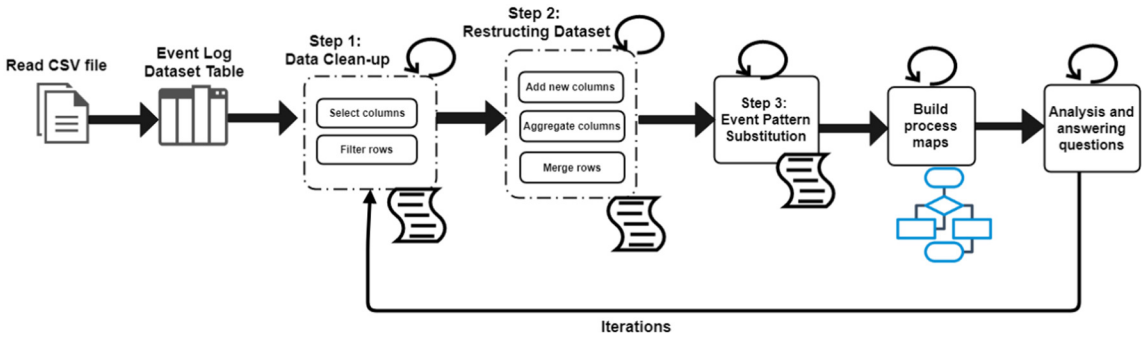

Figure 11: CPA-PM method.

5.3. CPA-PM Steps

This section provides the details for each of the three steps of the CPA-PM method, to be supported by the API summarized in Appendix A.1. An ongoing example involving an e-commerce cloud application is used to illustrate the steps.

5.3.1 Clean-up Step

In this step, initial cleaning of the dataset is done that includes removing unnecessary columns or rows from the event log dataset. Additionally, having the correct format and including all the fields needed to apply process mining techniques are both checked. Seven sub-steps are included here, and the last two (Steps 6 and 7) can be performed in any order or even be interleaved.

- **Step 1:** Read the CSV log file. The CSV file, containing the event log produced by the cloud-based application, 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. Additionally, the rows in the CSV file represent records for each of the events.
- **Step 2:** Clean column identifiers: that includes removing white spaces and dots from the columns' names and ensuring the unicity of the names. This returns a dataframe **dataset** with clean names. The resulting names consist only of the _ character, lowercase letters, and numbers. By using the same example from Table 1, replicated in Table 3, as the original event log, we can see that the column identifiers are not consistent. For example, some of them may start with uppercase letters or include special characters. Cleaning the identifiers results in a modified heading row, shown in Table 4.

Table 3: Event log file sample.

Case.ID	Timestamp	Activity	City	Weekday	Client	Product.Category	Device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	A	Android
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	N/A	Android
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	D	iPhone
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	B	N/A
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	A	Android
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	X	N/A
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	N/A	Android
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	N/A	N/A
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	N/A	Android
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	N/A	iPhone
…	…	…				…	...

Table 4: Resulting dataset after applying the clean_names() function.

case_id	timestamp	activity	city	weekday	client	product_category	device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	A	Android
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	N/A	iPhone
…	…	…				…	...

- **Step 3:** Ensure the time format of the timestamps reflect 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 for instance, require merging separate Date and Time columns, or change the encoding of the date/time information. Existing functions from the underlying programming language can be used to ensure the right format is used. By referring to the example in Table 3, the timestamp field contains detailed information about when the event occurred, and the timestamp has this format: **YYYY-MM-DD hh:mm:ss**.

- **Step 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 5:** Select the columns needed for the analysis. According to the processes that we are analyzing and studying and depending on the analysis questions, some of the columns in the original event log might not be required. When working with clickstream data collected from a cloud application, each event log includes several properties about that particular event. The user should select only the columns needed for the analysis:
    - Ensure the presence of **Case Identifier**, **Event**, and **Timestamp** columns. These columns are required to be able to apply process mining techniques. In the example above (refer to Table 4), the **Case Identifier** is represented by the *case_id* column, the **Event** is represented by *activity* column, and the **Timestamp** is represented by *timestamp* column. These three columns are necessary to be able to apply process mining algorithms.
    - Consider keeping the resource columns that are needed based on the analysis questions.

- Consider keeping other attribute columns that refine events. For example, columns *city, weekday, client, product_id,* and *device* represent other column that give more information about the events and may be useful in answering some of the analysis questions.
- Consider excluding columns that are all empty or that contain re-identifying information, for example, emails, company names, addresses, phones number, etc.

- **Step 6:** Filter the event log dataset. Filtering is a well-known and popular technique used during the data preprocessing stage. The main aim for filtering event logs is to reduce complexity or to focus on a specific part of the process during the analysis. This technique is usually performed multiple times and it goes through several iterations to understand and analyze the data from different perspectives. van Eck et al. (2015) discuss three different types of filtering techniques: *slice and dice, variance-based filtering,* and *compliance-based filtering.*
  - *Slice and dice:* this technique includes removing events or traces based on values recorded in the event, log for example specific activity names, case identifiers, resources, or timestamp. That also includes keeping traces that occurred in a certain time interval.
  - *Variance-based filtering:* this technique includes grouping similar traces, into clusters, for example by partitioning the event log in order to find less complexed process models for each of the partitions in the original process.
  - *Compliance-based filtering:* this technique removes traces or events that do not comply with a specific rule in a process model. These rules are selected based on the process that is being analyzed.

  Filtering can be applied on:
  - Events that have a very low frequency. For example, in Table 5, rows 28 and 29 could be removed from the dataset. The activity **FAQ page** has a low frequency compared to the frequencies of the other events. However, in some cases, such events are important to keep if we are looking for outliers

or depending on the analysis questions that we are trying to answer and the processes we are studying.

- Events and noise rows that are unrelated to a specific process we are analyzing or from other processes. For example, keep only the rows that have activities related to the process being analyzed.
- Case identifiers that are empty. For example, in the event log of Table 5, rows 16, 19, and 21 have an empty (e.g., unknown) case_id.

Table 5: Sample event log dataset.

row #	case_id	timestamp	activity	city	weekday	client	Product_category	device
1	121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	N/A	Android
2	121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	A	Android
3	121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	N/A	Android
4	122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	D	iPhone
5	123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	B	N/A
6	121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	A	Android
7	123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	X	N/A
8	121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	N/A	Android
9	123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	N/A	N/A
10	123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	N/A	N/A
11	123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	N/A	N/A
12	121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	N/A	Android
13	121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	N/A	Android
14	124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	N/A	iPhone
15	121	2018-08-10 00:04:53	Refresh page	Ottawa	Wednesday	N/A	N/A	N/A
~~16~~		~~2018-08-10 00:04:54~~	~~Main home page~~		~~Wednesday~~		~~N/A~~	~~N/A~~
17	124	2018-08-15 00:06:28	Refresh page	Halifax	Monday	Phone App	N/A	iPhone
18	124	2018-08-13 00:06:30	Refresh page	Halifax	Saturday	Phone App	N/A	Android
~~19~~			~~Refresh page~~					~~N/A~~
20	128	2018-08-14 00:06:30	Shopping cart details	Quebec City	Sunday	Phone App	N/A	iPhone
~~21~~			~~Checkout page~~					~~N/A~~
22	128	2018-08-14 00:05:55	Main home page	Quebec City	Sunday	Online Browser	N/A	N/A
23	123	2018-08-07 00:06:30	Shopping cart details	Montreal	Sunday		N/A	N/A
24	124	2018-08-15 00:05:01	Shopping cart details	Halifax		Phone App	N/A	Android
25	125	2018-08-16 00:05:01	Refresh page	Vancouver	Tuesday		N/A	Android
26	121	2018-08-10 00:04:50	Refresh page	Ottawa		Phone App	D	iPhone
27	121	2018-08-10 00:04:51	Refresh page	Ottawa		Phone App	X	iPhone
28	124	2018-08-15 00:05:01	FAQ page	Halifax	Monday	Phone App	N/A	Android
29	121	2018-08-10 00:03:01	FAQ page	Ottawa	Wednesday	Phone App	N/A	Android
30	…	..	…	…	…	…	..	…

- Events that have very high frequency in the data and that do not add value to understanding the processes. For example, in the dataset below, the **Refresh page** activity has a high frequency and complexify the resulting process. Yet, as a user can refresh any page many times, such activity does not add any knowledge to the analysis, so these rows can be eliminated.
- Filter the traces that only include a very small number of events. Traces that include 1, 2, or 3 events could be removed from the dataset since these will unlikely add value to the analysis. As usual, the analyst decides which traces to remove according to the number of events.

- **Step 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 (see Figure 12). There are different reasons why incomplete cases exist in a real dataset.
  1- The data extraction was based on a certain timeframe. For example, let us assume an event log was extracted for a certain timeframe from February 2018 to May 2018, but there is a case (process instance) that started in January 2018 and finished in March 2018. In the extracted event log, this truncated case will still be included but will be missing the starting point and all the steps that were executed in the month of January 2018.
  2- Some cases did not finish. When extracting the data there are some cases that have not finished yet within the timeframe. Let us consider the same context explained above, but with a case starting in April 2018 and finishing after May 2018 (say, in June 2018). Then this process will show in the process map but missing the finish event.
  3- Some traces might never finish. For example, a user started executing the process but abandoned the execution along the way (so the end events will be missing). It is a challenge to determine whether a user actually abandoned the process; a maximum expected duration can be used in such situation.

  According to the three reasons mentioned above, removing incomplete cases is a very important step in cleaning the dataset before starting the analysis. Leaving

such incomplete cases would confuse process mining algorithms into thinking there are other (invalid) process start/end points for the process.

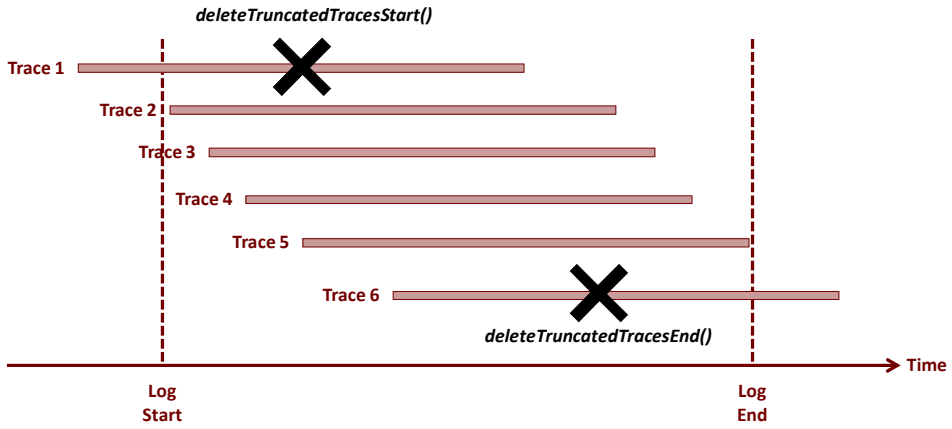

Figure 12: Example of incomplete traces with missing start events (Trace 1) of end events (Trace 6).

5.3.2 Restructuring Step

In this step, restructuring for the event log is performed in a way that process mining algorithms can be applied depending on the questions that should be answered. For example, restructuring includes *enriching the event log dataset*, *aggregating events*, and *aggregating rows*. These different techniques can be applied during this step.

- *Enriching the event log dataset*: event logs can be enriched with additional attributes. One way is by adding external additional data collected, for example by combining two different datasets. This is however often done before the generation of the original CSV file. 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. The need for such additional columns will depend on the analysis that is being done. For example, calculating the time difference between the occurrence of two events might be useful to enable process mining algorithms to reason about durations.
- *Aggregating events*: aggregating events into coarser-grained or more abstract events can help reduce complexity and improve the structure of clickstream datasets, leading to better and simpler processes. Again, the number of events remains unchanged

here. van Eck et al. (2015) distinguish between two different types of aggregations: *is-a* and *part-of*.

- The *is-a* aggregates two different events that belong to the same class, but the number of events stay the same. For example, two events labeled with *Start Help Guide* and *Exit Help Guide* are considered instances of *Help Guide* but they remain as two event instances.
- The *part-of* aggregation merges multiple events into a coarser-grained event, for example, when defining hierarchy events based on event attributes, for city (smaller) or country (larger).

- *Aggregating rows:* this technique includes aggregating several rows into one row, to reduce complexity and (unnecessary) repetition of the same event. Let us consider that event *A* is repeated multiple times in a given case, but it needs to be considered only once. There are several options for the aggregation in this case, including keeping only the first occurrence of event *A*, keeping only the last occurrence of event *A*, 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 details in the API functions and their implementations.

5.3.3 Pattern Substitution Step

In this step, complex events 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 are done according to the domain knowledge experts. Several common but non-exhaustive patterns are defined as rewrite rules in this step and can be used on the cloud-based application event log dataset (for each case). These and other patterns can be defined by the analyst depending on the analysis context. In the following, + indicates 1 or many instances, and _ represents the absence of events (in their deletion).

Pattern 1: $A^+ \rightarrow A$

Let us consider *A*, an event in case. In this pattern, a non-empty sequence of consecutive events *A* is replaced by only one *A*.

Pattern 2: $A^+B^+ \rightarrow AB$

Let us consider two distinct events *A* and *B* in a case. In this pattern, a non-empty sequence of *A* events followed by a non-empty sequence of *B* events are replaced by one *AB* event pair. First all *A*s are aggregated, and then all *B*s are aggregated.

Pattern 3: $AB \rightarrow C$

Let us consider two distinct events *A* and *B* in a case. In this pattern, an *A* immediately followed by a *B* is replaced by a new event *C*.

Pattern 4: $A^+ \rightarrow \_$

Let us consider an event *A* in a case. In this pattern, non-empty sequences of *A*s are removed from the case.

Pattern 5: $AB^+C \rightarrow A\_C$

Let us consider three distinct events *A*, *B*, and *C* in a case. In this pattern, an event *A* has to be followed by sequence of *B* events, and then by a *C* event. The sequence of *B*s is removed from the case and the result is the sequence *AC*.

Pattern 6: $AB^+C \rightarrow ABC$

Let us consider three distinct events *A*, *B*, and *C* in a case. In this pattern, an event *A* has to be followed by a sequence of *B*s and then by an event *C*. The sequence of *B*s is aggregated into one *B* event. The result sequence is *ABC*.

Note that the aggregation function in these patterns also need to handle the events' attributes (especially when a sequence is replaced by one event), and design decisions will be required from the analyst when implementing the details.

5.4. Event Log Preprocessing Functions

This section defines and illustrates the functions that compose the API needed to support the abstract steps discussed in the previous section. Table 6 provides an overview of 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, a table with a heading row is also called a *dataframe*.

Table 6: Summary of API functions.

	Row	Column	Trace	Other
Add		concatenateColumns() eventIsRepeated()		
Remove	keepFirstEvent() keepLastEvent() filter() removeEvents- LowFrequency() deleteAllEvents()	selectColumns() deleteColumns()	deleteTraceLengthLessThan() deleteTruncatedTracesStart() deleteTruncatedTracesEnd() deleteTracesWithTimeLess()	
Modify	cleanHeaders() arrangeRows() mergeRows()			
Other				readCSV() writeCSV()

5.4.1 Read a CSV file

readCSV(String file)

The *readCSV* function reads the CSV file into a dataframe that it creates.

Arguments

- *file* is the CSV file in the select work directory.

Returns a dataframe.

Example

`readCSV(file="DataSet.csv”)→ DataSet`

This invocation reads the CSV file DataSet.csv into a dataframe that it creates and that is called DataSet. Table 7 gives an example of a dataframe file. The rows represent different records in the dataset and the columns represent the attributes.

Table 7: Dataframe table created from CSV file.

Case.ID	Timestamp	activity	City	Weekday	Client	Product.Category	Device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	A	Android
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	N/A	Android
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	D	iPhone
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	B	N/A
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	A	Android
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	X	N/A
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	N/A	Android
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	N/A	N/A
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	N/A	Android
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	N/A	iPhone
…	…	…				…	...

5.4.2 Write to a CSV file

writeCSV(table DataSet, String file)

The *writeCSV* function writes the dataframe to a CSV file.

Arguments

- *DataSet* is the name of dataframe to save.
- *file* is the name of the CSV output file.

Returns a CSV file

Example

```
writeCSV(DataSet, file="DataSet.csv")
```

This writes the dataframe DataSet to the DataSet.csv file.

5.4.3 Clean columns headers

cleanHeaders(table DataSet)

This function cleans the headers of the columns from spaces and other special characters. It only keeps lower case letters, numbers, and underscores (_). The spaces are replaced by '_' and the special characters are removed.

Arguments

- *DataSet* is the name of the dataframe to modify.

Returns a dataframe with clean header names.

Example

```
cleanHeaders(EventLogs)-> EventLogs
```

Table 8 represents the output after applying the cleanHeaders() function to Table 7. The column names are all in lower case.

Table 8: Dataframe after applying the cleanHeaders() function to Table 7.

case_id	timestamp	activity	city	weekday	client	product_category	device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	A	Android
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	N/A	Android
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	D	iPhone
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	B	N/A
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	A	Android
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	X	N/A
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	N/A	Android
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	N/A	N/A
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	N/A	Android
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	N/A	iPhone
…	…	…				…	...

5.4.4 Select columns from a dataframe

selectColumns(table DataSet, string columnName, …)

This function selects/keeps the list columns needed for analysis from the dataset. Only the list of selected columns/attributes are included in the dataset. Note that after applying the *cleanHeaders()* function, the column names might change. The new names will be used from now in the rest of the functions.

Arguments

- *DataSet* is the name of the dataframe.
- *columnName* is the name of the column to keep in the dataset. Many can be listed, separated by commas.

Returns a dataset including only the list of columns/attributes that are selected.

Example

```
selectColumns(EventLogs, case_id, timestamp, activity)
      -> EventLogs
```

Table 9 represents the output dataframe after applying the selectColumns() function. Only the columns selected are included in the dataframe.

Table 9: Dataframe after applying the selectColumns() function to Table 8.

case_id	timestamp	activity
121	2018-08-01 00:04:22	Main home page
121	2018-08-01 00:04:27	Home page for product
121	2018-08-01 00:04:48	General website search
122	2018-08-10 00:04:49	Home page for product
123	2018-08-07 00:05:40	Product review page
121	2018-08-01 00:04:52	Product details information
123	2018-08-07 00:05:58	Home page for product
121	2018-08-01 00:04:59	Add to cart
123	2018-08-07 00:06:04	Shopping cart details
123	2018-08-07 00:06:10	Shopping cart checkout
123	2018-08-07 00:06:22	Confirmation order page
121	2018-08-01 00:05:05	Shopping cart details
121	2018-08-01 00:05:15	Shopping cart checkout
124	2018-08-13 00:06:28	Main home page
…	…	…

5.4.5 Delete columns from a dataframe

deleteColumns(table DataSet, string columnName, …)

This function drops the listed columns from the dataset. The columns listed by column-Name/header will be deleted/dropped from the DataSet dataframe.

Arguments

- *DataSet* is the name of the dataframe.
- *columnName* is the name of the column to drop from the dataset. Many can be listed, separated by commas.

Returns a dataframe excluding the columns/attributes that are selected to be dropped.

Example

```
deleteColumns(EventLogs,-(client,product_category,device))
      → EventLogs
```

Table 10 represents the output result after applying the deleteColumns() function on Table 8. The three columns *client*, *product_category*, and *device* are deleted from the dataset.

Table 10: Dataframe after applying the deleteColumns() function to Table 8.

case_id	timestamp	activity	city	weekday
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday
121	2018-08-01 00:04:48	General website search	Ottawa	Monday
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday
…	…	…		

5.4.6 Filter rows

filter(table DataSet, condition Conditions)

The filter function keeps records/rows based on the conditions specified. Only the rows where the condition is TRUE are kept in the DataSet. The filter function supports multiple functions, for example: ==, >, <, >=, <=, &, | , ! .

Arguments

- *DataSet* is the dataframe.
- **Conditions** represents the logical conditions defined in terms of the column names; it is possible to have multiple condition separated by &.

Returns a dataframe where the conditions specified are TRUE.

Example

```
filter(dataset, company_id != ""
      & city != "Ottawa"
      & weekday != "Wednesday") → dataset
```

Table 13 represent the output dataframe after applying the *filter()* function, starting with the dataframe represented in Table 11. Table 12 represents the dataframe with the rows that were deleted according to the filter function.

Table 11: Dataframe representing the dataset before applying the filter() function.

case_id	timestamp	activity	city	weekday
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday
121	2018-08-01 00:04:48	General website search	Ottawa	Monday
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday
…	…	…		

Table 12: Dataframe with the rows that were deleted by the filter() function.

case_id	timestamp	activity	city	weekday
~~121~~	~~2018-08-01 00:04:22~~	~~Main home page~~	~~Ottawa~~	~~Monday~~
~~121~~	~~2018-08-01 00:04:27~~	~~Home page for product~~	~~Ottawa~~	~~Monday~~
~~121~~	~~2018-08-01 00:04:48~~	~~General website search~~	~~Ottawa~~	~~Monday~~
~~122~~	~~2018-08-10 00:04:49~~	~~Home page for product~~	~~Toronto~~	~~Wednesday~~
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday
~~121~~	~~2018-08-01 00:04:52~~	~~Product details information~~	~~Ottawa~~	~~Monday~~
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday
~~121~~	~~2018-08-01 00:04:59~~	~~Add to cart~~	~~Ottawa~~	~~Monday~~
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday
~~121~~	~~2018-08-01 00:05:05~~	~~Shopping cart details~~	~~Ottawa~~	~~Monday~~
~~121~~	~~2018-08-01 00:05:15~~	~~Shopping cart checkout~~	~~Ottawa~~	~~Monday~~
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday
…	…	…		

Table 13: Output dataframe after applying the filter() function.

case_id	timestamp	activity	city	weekday
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday
…	…	…		

5.4.7 Remove events with low frequency from dataset

removeEventsLowFrequency(table DataSet, String eventName, integer freq)

This function removes the events from the dataset that have a frequency below the value *freq* grouping by *eventName*.

Arguments

- *DataSet* is a dataframe.
- *eventName* is the name of the value to group by.
- *freq* is the threshold value that is used to filter out rows whose count is less than freq.

Returns a dataframe containing only the rows that meet the specified minimum frequency.

Example

```
removeEventsLowFrequency(EventLogs, activity, 5) → EventLogs
```

5.4.8 Delete traces with low number of events

deleteTraceLengthLessThan(table DataSet, String groupID, integer num)

This function removes the traces where the number of events is less than a specified value.

Arguments

- *DataSet* is a dataframe.
- *groupID* is the name of the variable to group traces by.
- *num* is the required minimum number of events in a trace.

Returns a dataframe containing only the traces that have at least num events.

Example

```
deleteTraceLengthLessThan(EventLogs, case_id, 2) → EventLogs
```

5.4.9 Remove truncated traces

deleteTruncatedTracesStart(table DataSet, String groupID, String eventColumn, String value)

This function removes the traces/cases that do not start with the required event value.

Arguments

- *DataSet* is a dataframe.

- *groupID* is the name of the variable to group traces by.
- *eventColumn* is the name of the event column.
- *value* is the value of eventColumn that represents the required start event name.

Returns a dataframe only including the traces that start with an event value.

Example

```
deleteTruncatedTracesStart(EventLogs, case_id, activity, "SignUp")
      → EventLogs
```

In this example, the function will remove all the traces that do not start with a *SignUp* event. This is used to eliminate incomplete traces whose beginning was truncated.

deleteTruncatedTracesEnd(table DataSet, String groupID, String eventColumn, String value)

This function removes the traces/cases that do not end with the required event value.

Arguments

- *DataSet* is a dataframe.
- *groupID* is the name of the variable to group traces by.
- *eventColumn* is the name of the event column.
- *value* is the value of eventColumn that represents the required end event name.

Returns a dataframe only including the traces that end with an event value.

Example

```
deleteTruncatedTracesEnd(EventLogs, case_id, activity, "purchase
      completed")→ EventLogs
```

In this example, the function will remove all the traces that do not end with a *purchase completed* event, in order to eliminate incomplete traces whose end was truncated.

5.4.10 Delete traces with insufficient duration

deleteTracesWithTimeLess(table DataSet, String groupID, String timestampColumn, integer t)

This function removes the traces/cases that were running for a total duration less than t.

Arguments

- *DataSet* is a dataframe.

- *groupID* is the name of the variable to group traces by.
- *time*stampColumn is the name of the timestamp column.
- *t* is the value of the minimum total duration for each trace/case, in seconds.

Returns a dataframe only including the traces that have a total duration greater than or equal to *t*.

Example

```
removeTracesWithTimeLess(EventLogs, company_id, time, 300)
      → EventLogs
```

In this example, this function will remove all the traces that were running with a total duration less than 5 min.

5.4.11 Concatenate two or more columns

concatenateColumns(table DataSet, String newColumn, String col1, String col2, …)

The concatenateColumns() function concatenates two or more columns in a dataframe into a new column that is added to the dataframe.

Arguments

- *DataSet* is a dataframe.
- *newColumn* is the name of the new column which is the result of the concatenation of the columns.
- *col1* is the name of a column in the dataframe.
- *col2* is the name of a second column in the dataframe, and so on.

Returns a dataframe with the new concatenated column added.

Example

```
concatenateColumns(EventLogs, event, activity, client)→ EventLogs
```

Table 14 represents a dataframe before applying the concatenateColumns() function, which is to concatenate the *activity* and *client* columns into another one named *event*. The resulting dataframe is represented in Table 15, which has the additional *event* column.

Table 14: Dataframe before applying the concatenateColumns() function.

case_id	timestamp	activity	city	weekday	client
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App
…	…	…			

Table 15: Dataframe after applying the concatenateColumns() function to Table 14.

case_id	timestamp	activity	city	weekday	client	event
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	Main home page Phone App
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	Home page for product Phone App
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	General website search Phone App
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	Home page for product Phone App
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	Product review page Online Browser
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	Product details information Phone App
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	Home page for product Online Browser
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	Add to cart Phone App
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	Shopping cart details Online Browser
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	Shopping cart checkout Online Browser
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	Confirmation order page Online Browser
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	Shopping cart details Phone App
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	Shopping cart checkout Phone App
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	Main home page Phone App
…	…	…				

5.4.12 Arrange rows

arrangeRows(table DataSet, String columnName, …)

This function sorts the rows by columnName, in increasing order.

Arguments

- *DataSet* is a dataframe table.
- *columnName* is a list of unquoted columns/attributes names.

Returns a dataframe that is arranged in increasing order according the column specified.

Example

`arrangeRows(EventLogs, case_id, timestamp)` → `EventLogs`

The *arrangeRows()* function orders the rows/records in increasing order grouping by *case_id* and then *timestamp* (see Table 16).

Table 16: Output table after applying the arrange() function.

case_id	timestamp	activity	city	weekday	client	product_category	device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:04:27	Home page for product	Ottawa	Monday	Phone App	A	Android
121	2018-08-01 00:04:48	General website search	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	A	Android
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	N/A	Android
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	N/A	Android
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	D	iPhone
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	B	N/A
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	X	N/A
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	N/A	N/A
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	N/A	N/A
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	N/A	iPhone
…	…	…				…	...

5.4.13 Create isRepeated column

eventIsRepeated(table DataSet, String groupIDCol, String eventCol, String newCol)

This function creates a new column called newCol that indicates whether the event has been repeated or not. The event is represented by the eventCol parameter. The newly created column newCol will only include 0 (not repeated) or 1 (repeated).

Arguments

- *DataSet* is the dataframe table.
- *groupIDCol* is the column/attribute name that we are grouping by.
- *eventCol* is the name of the event column whose repetition is to be indicated.
- newCol is the name of the new column to be created that will include only 0 or 1.

Returns the dataframe with an additional *isRepeatedEvent* column.

Example

`eventIsRepeated(EventLogs, case_id, activity, isRepeated)` → `EventLogs`

5.4.14 Delete all events

deleteAllEvents(table DataSet , String event, String eventName)

This function deletes all records/rows with an eventName value for the event attribute.

Arguments

- *DataSet* represent the dataframe.
- *event* represents the name of the attribute column that represents the activity information.
- *eventName* represents the event of the rows to be deleted.

Returns a dataframe that has excluded all the records where the event attribute equals eventName.

Example

```
deleteAllEvents(dataset, activity, "Add to Cart") → dataset
```

In this example, all the rows that have activity value equals “Add to Cart” are deleted from the dataset.

5.4.15 Keep the first occurrence of an event

keepFirstEvent(table DataSet, groupID, String eventAttribute, String eventName)

This function keeps the first occurrence of an event (i.e. the first occurrence of several repeated events).

Arguments

- *DataSet* represents the dataframe.
- *groupID* represents the column for grouping the records.
- *eventAttribute* represents the event attribute column or the activity column name.
- *eventName* represents the event name whose first occurrence in each group we want to keep.

Returns a dataframe with only the first occurrence of an event name that is specified.

Example

```
keepFirstEvent(EventLogs, case_id, activity, "go to home page")
      → EventLogs
```

5.4.16 Keep the last occurrence of an event

keepLastEvent(table DataSet, groupID, String eventAttribute, String eventName)

This function keeps the last occurrence of an event (i.e. the last occurrence of several repeated events).

Arguments

- *DataSet* represents the dataframe.
- *groupID* represents the column for grouping the records.
- *eventAttribute* represents the event attribute column or the activity column name.
- *eventName* represents the event name whose last occurrence in each group we want to keep.

Returns a dataframe with only the last occurrence of an event name that is specified.

Example

```
keepLastEvent(EventLogs, case_id, activity, "go to home page")
      → EventLogs
```

5.4.17 Aggregate/merge several rows

This important and complex function aggregates several rows into one row. This function works by splitting the data into groups (representing related events in one trace), applying some analysis to each group, and then combining the results. It works by aggregating several rows for each column. For example, users can apply mean, median, min, or max aggregations on a per column basis. This function is needed because when we want to merge multiple rows into one row to reduce the number of repeated events, we have to handle the values in all their attributes/columns properly. Deciding which aggregation operator to use for which column will be done by the analyst.

First, let us see how this function work on the dataset in Table 17. If the analyst wants to merge the first three rows (highlighted in blue) into one row, she must decide what to do with the values of attributes timestamp, city, weekday, client, and the rest of the columns. She could to create a set of unique values for the city, weekday, client, product_category and device columns, while computing the sum of the #_of_items, and preserving the first

timestamp in the sequence. Table 18 shows the result of the aggregation (first row highlighted in blue).

Table 17: Dataset example.

case_id	timestamp	activity	city	weekday	client	#_of items	product_category	device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	3	N/A	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
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	6	A	Android
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	7	N/A	Android
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	23	N/A	Android
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	34	N/A	Android
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	22	D	iPhone
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	22	B	N/A
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	NA	X	N/A
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	3	N/A	N/A
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	4	N/A	N/A
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	5	N/A	N/A
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	6	N/A	iPhone

Table 18: Dataset result after aggregating the first three rows of Table 17.

case_id	timestamp	activity	city	weekday	client	#_of items	product_category	device
121	2018-08-01 00:04:22	Main home page	Ottawa	Monday	Phone App	13	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
121	2018-08-01 00:04:52	Product details information	Ottawa	Monday	Phone App	6	A	Android
121	2018-08-01 00:04:59	Add to cart	Ottawa	Monday	Phone App	7	N/A	Android
121	2018-08-01 00:05:05	Shopping cart details	Ottawa	Monday	Phone App	23	N/A	Android
121	2018-08-01 00:05:15	Shopping cart checkout	Ottawa	Monday	Phone App	34	N/A	Android
122	2018-08-10 00:04:49	Home page for product	Toronto	Wednesday	Phone App	22	D	iPhone
123	2018-08-07 00:05:40	Product review page	Montreal	Sunday	Online Browser	22	B	N/A
123	2018-08-07 00:05:58	Home page for product	Montreal	Sunday	Online Browser	NA	X	N/A
123	2018-08-07 00:06:04	Shopping cart details	Montreal	Sunday	Online Browser	3	N/A	N/A
123	2018-08-07 00:06:10	Shopping cart checkout	Montreal	Sunday	Online Browser	4	N/A	N/A
123	2018-08-07 00:06:22	Confirmation order page	Montreal	Sunday	Online Browser	5	N/A	N/A
124	2018-08-13 00:06:28	Main home page	Halifax	Saturday	Phone App	6	N/A	iPhone

This function can be applied to any number of rows as long as the selection is done. Also, it can be applied not only for rows with same event names, but also for rows that have different event names (e.g., to support patterns such as the ones described in section 5.3.3).

The *mergeRows()* function works by splitting the dataframe, applying aggregation functions, and returning the result in a dataframe.

mergeRows(table DataSet, list groupByVariables, list columnArguments)

Arguments

- DataSet represents the dataframe table that we are working with.
- groupByVariables represent the variables to split dataframe by
- columnArguments represents the new column arguments.

For example, in *new_time = first(time)*, *new_time* is the new name of the column that will result after the merge application, whereas *first* is the operation to keep the value of the first row, and *time* is the name of the column as it is in the dataset before we apply mergeRows().

The aggregation operations that can be applied on the different columns are:

- mean(): computes the mean of the aggregated values.
- median(): computes the median of the aggregated values.
- min(): uses the minimum value among the aggregated values.
- max(): uses the maximum value among the aggregated values.
- sum(): computes the sum of the aggregated values.
- first(): keeps the first value.
- last(): keeps the last value.
- n(): count the number of values.
- any(): logical function where any of the values is true.
- all(): logical function where all the values are true.

Returns a dataframe with events and their attributes aggregated as specified.

Example

```
mergeRows(.dataset, .(company_id, event), time=first(time),
                                  number_of_graphs=mean(number_of_graphs),
                                  kpi_count=sum(kpi_count),
                                  city = paste(unique(city)))
```

This invocation merges the rows grouping by *company_id* and *event* attributes. It keeps the first time in the *time* column, calculates the mean for the *number_of_graphs* column, calculates the sum for the *kpi_count* column, and creates a list of unique cities for the *city* column.

5.5. Chapter Conclusion

This chapter described the CPA-PM method, its different steps and more detailed activities, and the illustrated API functions that support these activities. The R implementation of this API is provided in Appendix A.1. The *mergeRows()* function is a key part of this API.

In Chapter 6, the CPA-PM method is applied to a real clickstream event log dataset provided by a SaaS company, where the identifiers and some of the events were anonymized using pseudonyms. Additionally, a new dataset is provided afterwards for another analysis aiming to test how well the method works with other dataset and how many updates to the scripts invoking the methods are needed in order to adapt to newer datasets.

Chapter 6. SaaS Application Case Study

In this chapter, the CPA-PM method is applied to a real clickstream event log dataset (anonymized through the use of pseudonyms for identifiers and some activities) in order to assess the effect of the CPA-PM method in the preprocessing of datasets collected from a SaaS application on the process mining results.

Section 6.1 gives an overview of the case study. Section 6.2 introduces the dataset that is used in this case study and on which the CPA-PM method is applied. Section 6.3 focuses on planning the case study, which includes selecting the *Case ID, Timestamp,* and *Event* from the dataset. Section 6.4 describes data preparation and preprocessing; this is where the CPA-PM method steps are applied on the event log dataset. Section 6.5 presents process discovery and mining using ProM and Disco, and then the analysis. Finally, section 6.6 highlights extraction information from the process maps.

6.1. Case Study Overview

This case study is about discovering process maps from a clickstream event log dataset. The aim 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 is 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.

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 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 customers.

6.2. Dataset

The dataset was provided and anonymized by a SaaS company. *Data extraction* is the first step in such PM project, which is then followed by data preprocessing, and then process discovery and analysis. Data extraction involves getting the event log for the correct log management tool that is used by the SaaS company. In some cases, depending on the process that is being selected for analysis, data might need to be extracted from multiple sources. 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 confidential nature of the company).

6.3. Planning the Case Study

As explained in section 2.3, the main columns in the dataset that are required before applying PM techniques are the case identifier, the timestamp, and the activity.

6.3.1 Case ID

From the available set of attributes in the event log dataset, *Company ID* 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 SaaS application sporadically over several days.

6.3.2 Timestamp

Each event occurs at a particular moment. The timestamp used here is in this format: YYYY-MM-DD hh:mm:ss.

6.3.3 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 19 displays some events from the dataset and their description.

Table 19: Some event names and their description.

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
Edit dashboard	User edits a dashboard
Edit data source	User edits a data source, e.g., to change the data source type
Edit graph	User performs an edit graph action
Refresh	User refreshes a page
Graph saved	User saves the graph created
Help guide	User selects the help guide
Insert formula	User inserts a formula to do some calculations
Select dashboard template	User selects a template for the dashboard from the template library
Share dashboard	User shares a dashboard with another user in the same company
Sidebar help	User clicks on the sidebar help button
Trial sign up	User signs up for a trial version of the application
View dashboard	User clicks to view the dashboard page

6.3.4 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 are attributes are considered as *other* properties that will be used in the analysis.

6.3.5 Process Mined from the Original Event Log

Figure 13 shows the initial process map generated in Disco for the original EventLogs dataset. With the high number 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.

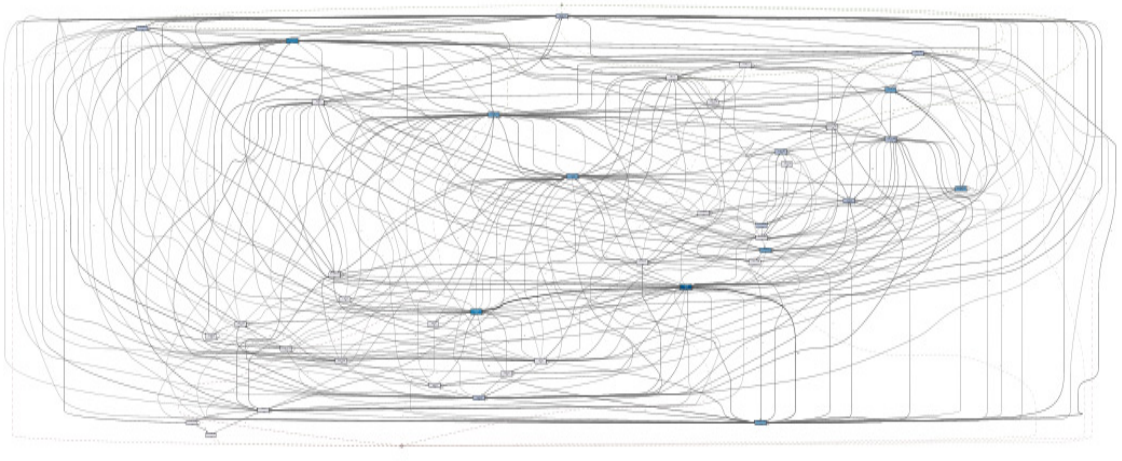

Figure 13: Initial process map for the original EventLogs dataset.

6.4. Data Preparation and Preprocessing

In order to prepare the data for the analysis, the CPA-PM method is used here, starting with the *Clean-up* phase, illustrated on the SaaS application dataset. The following six points correspond to steps 1 to 6 in section 5.3.1.

1- The first step involved importing the dataset (a CSV file) to RStudio, the R environment 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 ← read.csv(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- As mentioned in Section 6.3, I ensured that all the attributes needed for the analysis were available in the SaaS application dataset, with the right type.

5- In this step, I selected the list of attributes/columns that I need for my analysis. I called the selectColumns() function to select the list of columns and returns a data-frame with the list specified. 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 and not needed in the analysis were removed as well. The decision was to keep only events that have occurred more than 10 times.

```
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 is selected as the case identifier, event as the activity variable, and time as the timestamp variable (Figure 14).

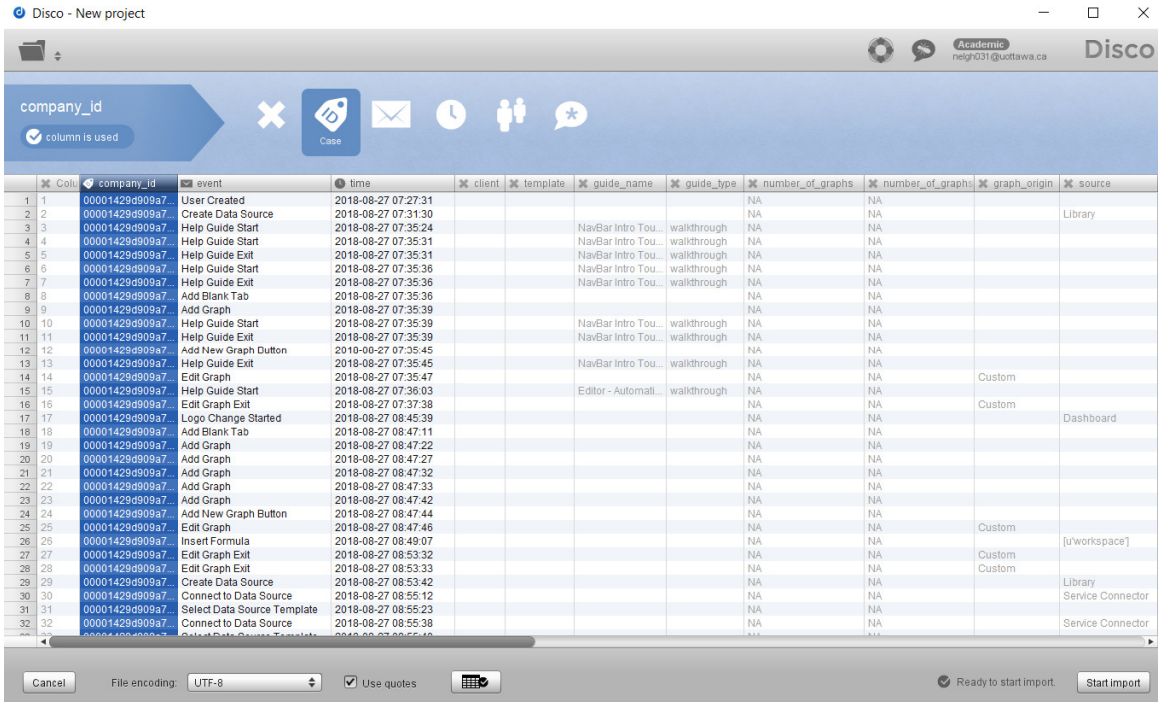

Figure 14: Cleaned dataset imported in Disco.

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.
- 13,848 different variants.

Table 20: Top events according to frequency.

Activity	▲ Frequency
Edit Graph	97,359
Graph Saved	64,076
Help Guide Start	58,756
Account Active Today	58,414
Graph Edit from Dashboard	51,149
Add Graph	50,518
Add Graph from Library	41,253
Graph Library	40,167
Select Graph from Library	39,710
Add Template	35,624
Help Guide Exit	35,068
Template Connect Button	32,272
Template Connected	30,989
Connect to Data Source	27,854
Edit Graph Exit	24,192
Create Data Source	23,909
Add Template DS Settings	21,320
Select Data Source Template	16,001
Connector Search	15,824
Trial Sign Up Completed	15,625
Data Source Error	14,858
Sidebar Help	14,852
Add New Graph Button	14,114
Add Blank Tab	13,963
Data Source Created	10,854
View Plans	10,093
Insert Formula	9,361
Dashboard Configured	7,581
Data Sources	6,363
Build Graph from Library	6,261
Add Existing Tab	6,184

After applying the steps mentioned above, the process map was generated (see Figure 15). Although much cleaner and understandable than the original one (Figure 13), this process map is still complicated to understand, and finding paths from start to end for processes executed by users remains challenging. Thus, further preprocessing is required.

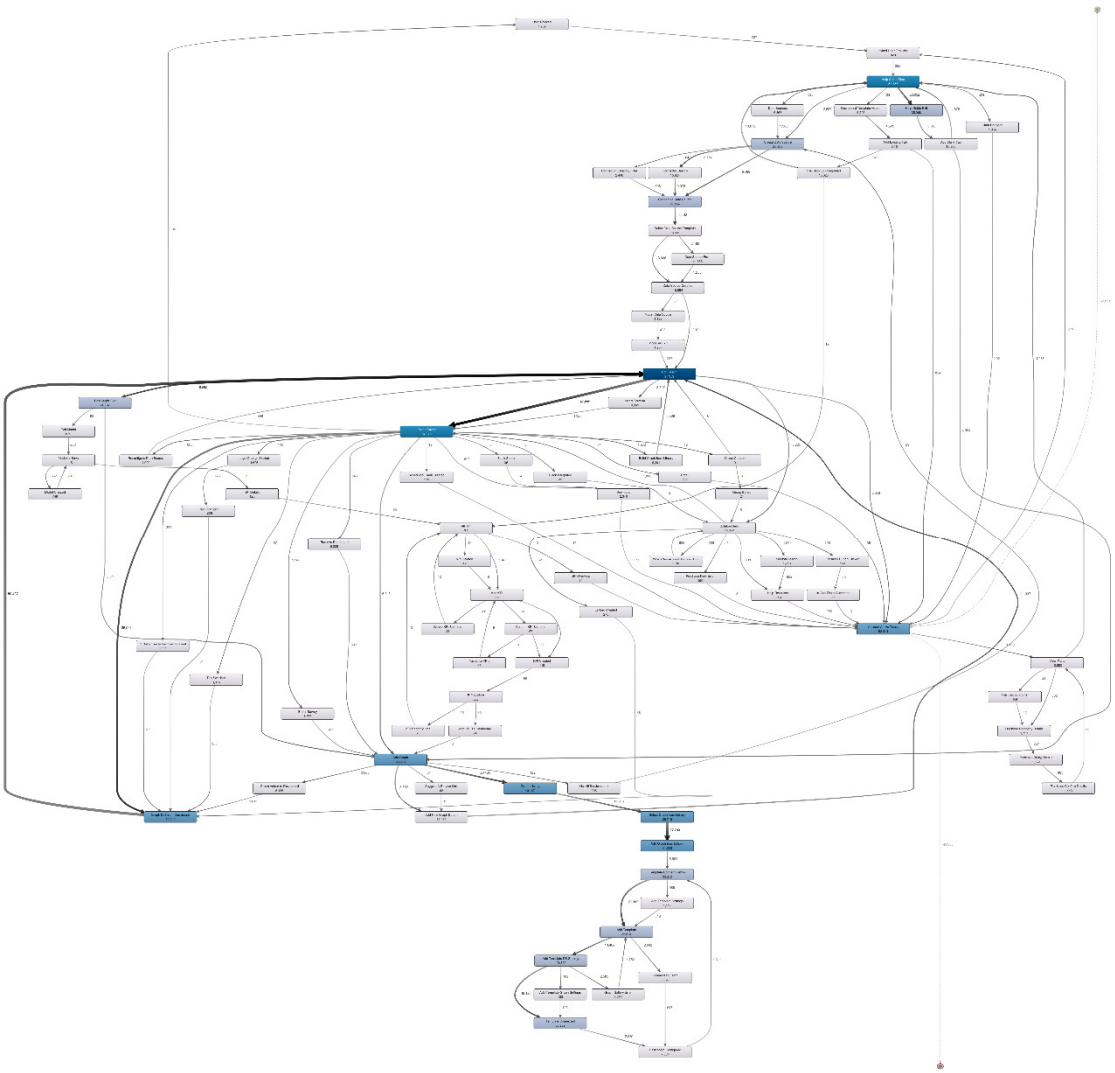

Figure 15: Process map after initial cleaning.

Removing Incomplete Cases

The removal of incomplete cases (step 7 in section section 5.3.1) 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 to 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, I 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 get simpler, with 816,125 different records and 15,190 unique cases (down from 52,084 after the initial cleanup). Note that deleting cases truncated at the end with `deleteTruncatedTracesEnd()` is not done here because we do want to analyze the reasons why users abandon trials before buying the application, and so these incomplete process instances must be preserved.

After importing the dataset to Disco and selecting the caseID, activity, and timestamp attributes, a new process map was generated, which is displayed in Figure 16.

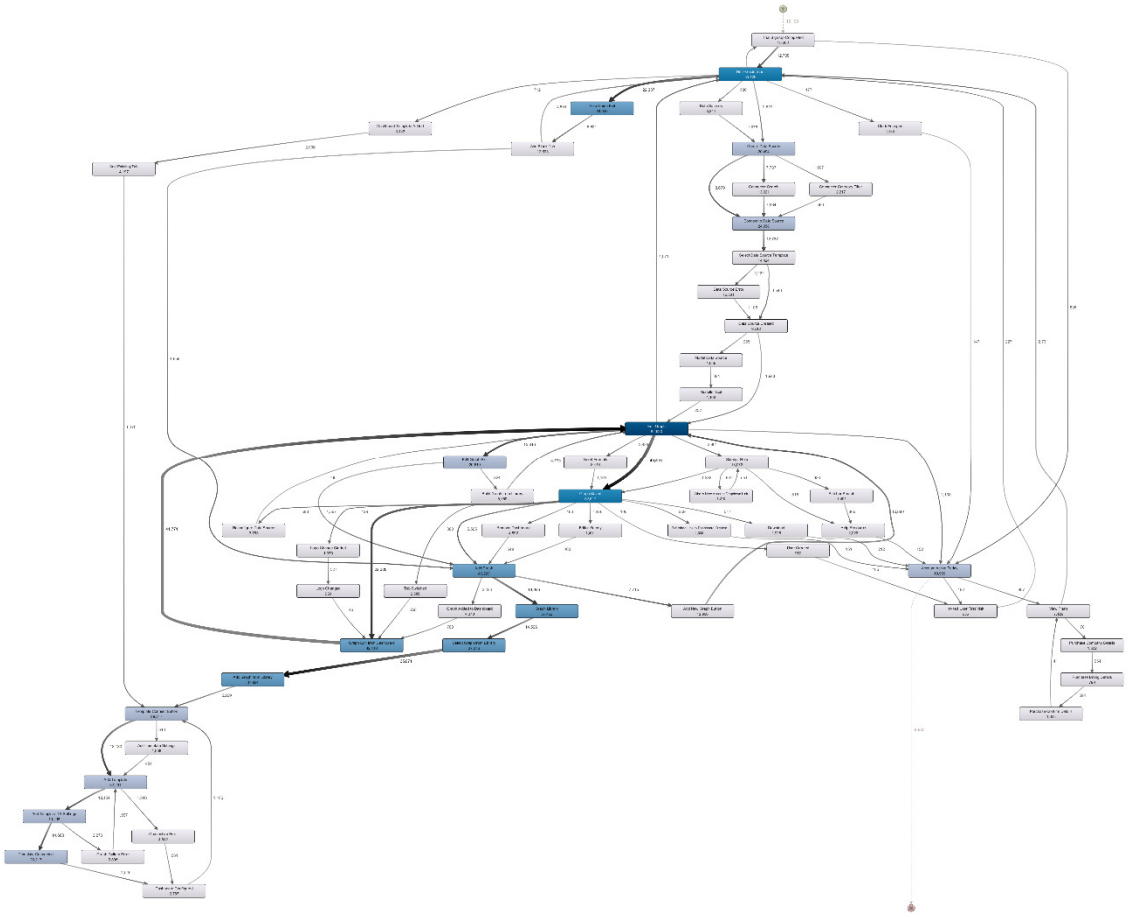

Figure 16: Process map after removing traces that do not start with the *Trial Sign Up Completed* event.

Removing Short Traces

Again, this process map is still too complex to be helpful in discovering valuable information, and more preprocessing is required. One type of filter based on very short traces (step 5 in section section 5.3.1) can be considered. For example, there are 520 cases

where each case only has one event “Trial Sign Up Completed”. Also there are 149 cases where each case only has 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 but did not really used 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
```

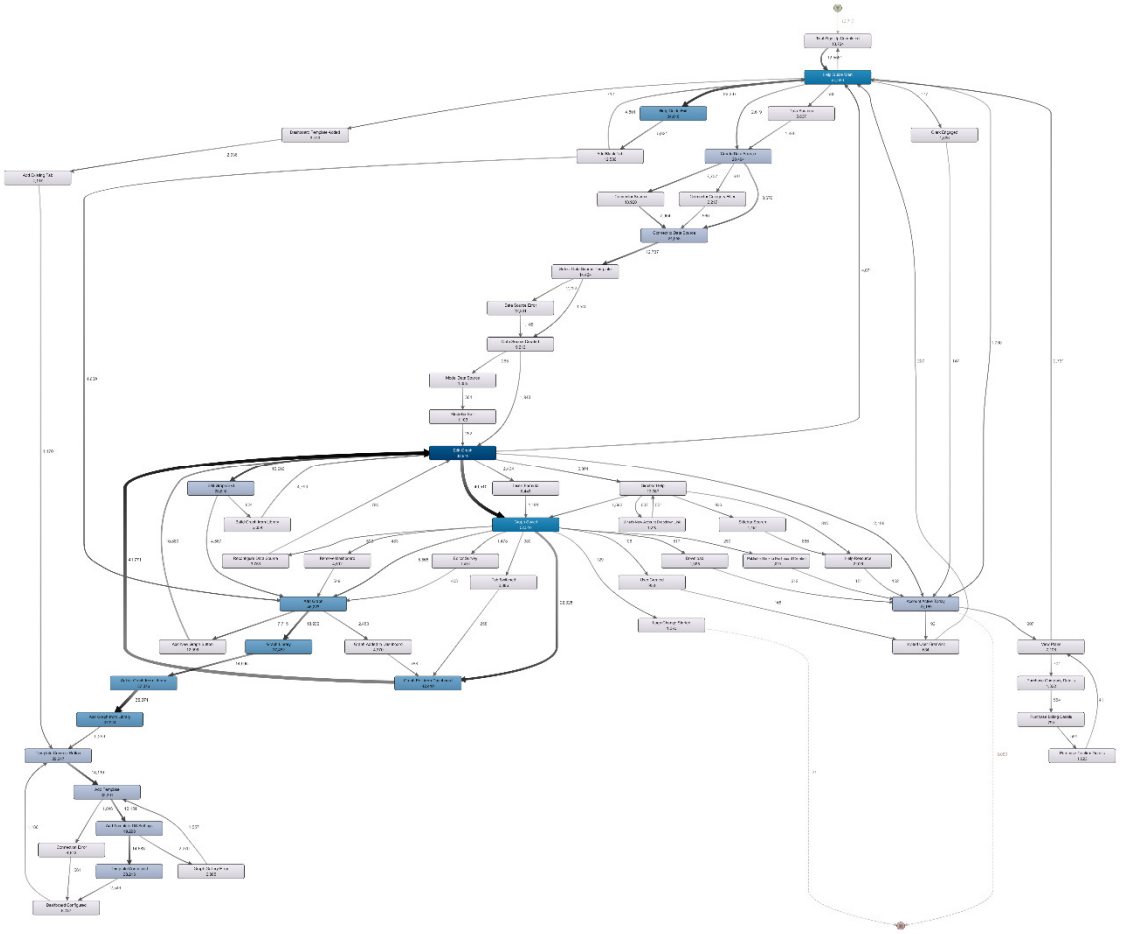

Figure 17: Process map generated after deleting the traces with 1 or 2 events.

After removing very short traces, the total number of records is now down to 813,691 with 13,713 unique cases and 11,922 variants. The total number of variants is still high (87%) with respect to the number of unique cases. The process map in Figure 17 is generated.

This map displays 63% of the activities with the highest frequencies (this is a visualization feature of Disco).

Restructuring by Minimizing Loop-Based Variants

Variants show how much users underestimate complexity in processes, especially when dealing with cloud-based application processes. When applying PM analysis, the number of variants can be an interesting metric to distinguish the common and exceptional behavior. However, to analyze the variants in a meaningful way, we need to have the dataset described at the right level of abstraction.

A *loop* means that an activity is repeated several times within one case. In some situations, adding additional details can be helpful to answer questions about the number of times these repetitions occurred and to analyze them in more detail. When analyzing cloud-based application processes, event repetitions tend to occur often in the dataset. What can be seen in the process map is that there are many self-loops (see Figure 18). These repetitions come from multiple clicks on the same web page. They come from a refresh, an automated redirection, or an internal post back to the same page. They are more of a technical nature than an actual repetition of the same process step. In Figure 18, loops are highlighted with red boxes. For example, *Help Guide Start* event has 15,021 self loops and *Edit Graph* has 88,815 self loops.

Figure 18 only shows a few examples of the self loops in the whole event log. Theses repetitions are not meaningful for analyzing the actual user experience for this process. Even worse, these repetitions also create many more variants than there actually are from a high-level process perspective.

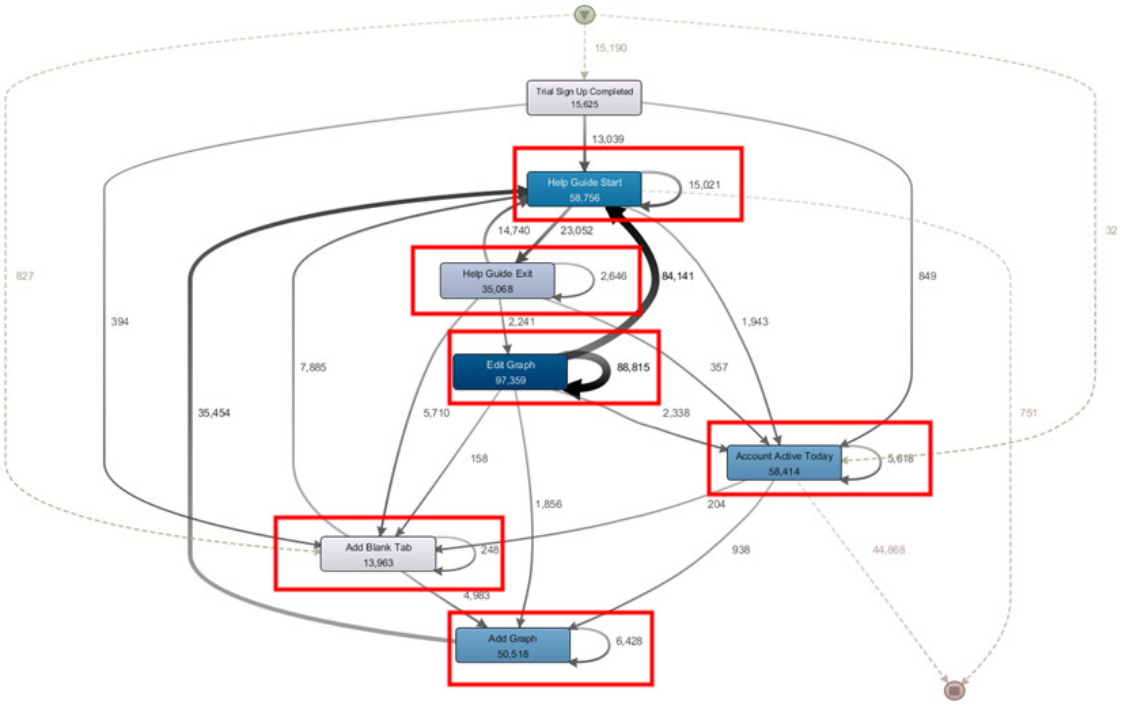

Figure 18: Process map example for loops.

For example, in the case shown in Figure 17, one variant includes these activities: *Trial Sign Up Completed* → *Help Guide Start* → *Help Guide Start* → *Help Guide Exit* → *Help Guide Start* → *Account Active Today.*

Another variant includes these activities: *Trial Sign Up Completed* → *Help Guide Start* → *Help Guide Start* → *Help Guide Exit* → *Account Active Today.*

These variants, which differ by one irrelevant *Help Guide Start* event, will make it more challenge to analyze such clickstream event logs. In the current dataset there are 11,922 different variants to analyze. Most of these variants are generated from different orderings of clicks on a page. Repetitions introduce many variations that are not relevant from a high-level view of the process. Hence, what we would like to do is to be able to exclude these repetitions from our analysis (as suggested in section 5.3.2).

We can do this by adding an additional column named *isRepeated* to the dataset, which indicates whether an event is a repetition or not:

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

This column includes only values 0 (not repeated) or 1 (repeated). This column can then be used in filtering the dataset and excluding the repeated events:

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

Pattern-Based Substitution

After filtering out the repeated events and keeping only the non-repeated ones, the total number of variants was reduced to 11,186. This is still too high a number of variants, so more preprocessing is still required. As suggested in section 5.3.3, pattern substitution can be considered.

Note that when we removed repeated events from the dataset, we have kept the first occurrence of the event in the sequence. For example, for the trace $A \rightarrow \underline{A} \rightarrow B \rightarrow \underline{B} \rightarrow C \rightarrow A \rightarrow B \rightarrow C$, we keep the events in this order $A \rightarrow B \rightarrow C \rightarrow A \rightarrow B \rightarrow C$. The second consecutive *A* and the second consecutive *B* from the original trace were removed. Keeping the rows where the *isRepeated* value 0 is the same as keeping only the first occurrence of an event in a repeated sequence. For example, we can run the *keepFirstEvent()* function, which keeps the first occurrence of an event. After keeping the first occurrence of an event, the analyst can execute again the *deleteTraceLengthLessThan()* function to delete very short traces in case some were generated after running *keepFirstEvent()*. These steps can be executed multiple times through several iterations.

In the red rectangle part of Figure 19, there are paths going both ways between the two events *Help Guide Start* and *Help Guide Exit*. These two events can occur in any order. They could possibly be replaced by one, more abstract event *Help Guide*. This is an example of event pattern substitution where two events are replaced by one event in order to reduce variations within the same process.

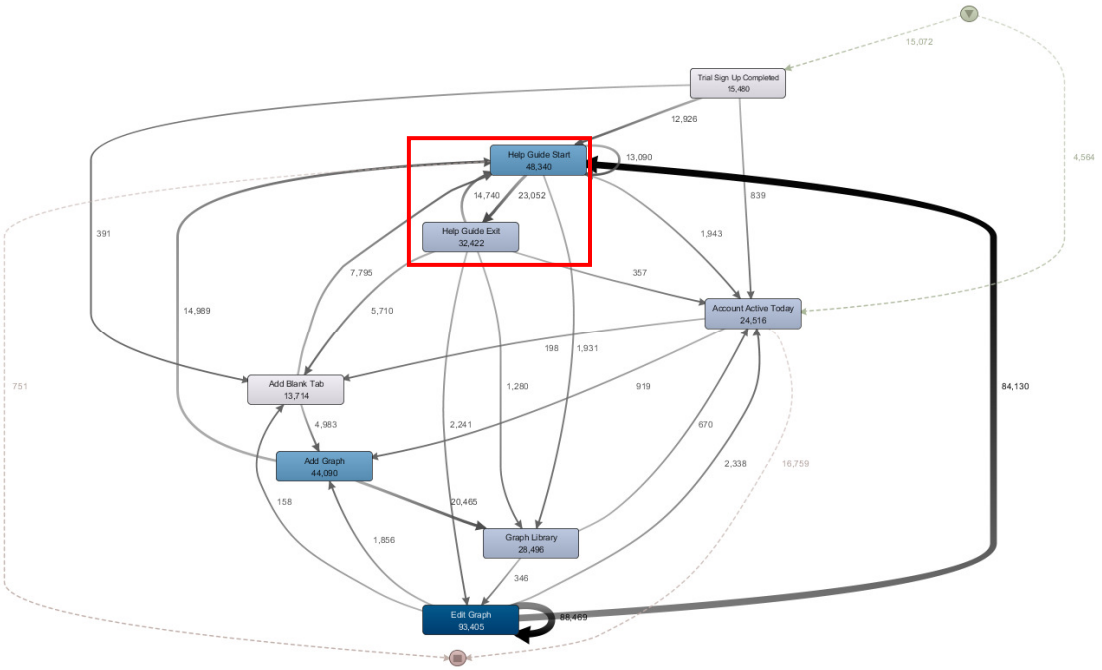

Figure 19: Process mining map.

In a process map, when there are arrows back and forth between two events, there are obvious benefits in grouping these two events together. In some cases, there are a number of events whose ordering is not important, and they can be replaced by one event. However, when aggregating rows together, several decisions need to be made regarding the other attributes included in the data, so valuable information is not lost. Table 21 shows some examples about events patterns and the new event names that replace a certain pattern.

Table 21: Examples of events patterns.

New Event Name	Events Patterns
Help Guide	Help Guide Start
	Help Guide Exit
Sidebar Help	Help Source
	Sidebar Help
	Sidebar Search
Graph Gallery	Graph Selected from Gallery
	Graph Added from Gallery
Connect Template	Template Connected
	Template Connect Button
Edit Graph	Edit Graph Exit
	Edit Graph
Add Graph	Graph Saved
	Add Graph
	Edit Graph from Dashboard

Again, it is up to the analyst to select the patterns and the events that should be replaced by one event. In some cases, the analyst may know in advance that some groupings of events can be done. In other cases, such patterns can be discovered. A *pattern-abstractions plug-in*, available for ProM (Bose & van der Aalst, 2009), can be used as a start point to discover meaningful abstractions of events, but this plug-in is useful for simple patterns only.

The next step is to replace a sequence of repeated events by one event name. This is done first by extracting a subset of the dataset according to a specific sequence of events. Then the analyst must aggregate the extracted rows into one row, according to the list of specified conditions. The resulting single row is then inserted into the original dataset.

The resulting script, which includes the above function invocations, is presented in Appendix A.2.

6.5. Process Discovery, Mining and Analysis

After preprocessing the event logs using the CPA-PM method, as applied in section 6.4, the dataset was imported to two PM tools, Disco and ProM, for process discovery (the specific type of process mining of interest in this thesis) and analysis. The preprocessing steps can go through several iterations before getting to the final answer. 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.

6.5.1 Disco Usage

The SaaS company provided us with some questions that they would like to see answered about their processes. Generating process maps can help with answering questions regarding processes. In addition, process mining may highlight new metrics that are affecting the processes and that should be measured.

The first question that we want to explore from the process maps aims to understand and analyze processes per segments, for example cities, countries, and weekdays, by clustering process users into two groups: trial converters and non-converters. Important sub-questions

include “How do processes change between weekdays for both trial converters and non-converters?” and “How do processes change between different cities?”. Another question to explore aims to understand the processes that are happening in the first 5 minutes, 15 minutes, and 20 minutes of the process, for trial converters and non-converters. The CPA-PM method helped in processing the large number of event logs and the variations in initially unstructured processes, in order to generate simpler and better-structured process maps that can help answering those questions.

In this case study, there are several challenges with respect to answering these specific questions. For example, analyzing the process for the non-converter users implies that the starting event is *Trial Sign Up Completed* and the end event could be any event except *Purchase Confirm*. However, I can only say that these users did not convert within the timeframe from where the event logs were extracted. Maybe they have converted a few days later and these events are not included in the extracted dataset that is being currently used in the analysis. In such situation, the analyst must make clear assumptions, e.g., that these users within this timeframe did not convert and are clustered as non-converters.

Another challenge exists when deleting events with a low frequency. Selecting the appropriate threshold is not trivial. If the analyst deletes all events that have frequency less than or equal to 10, then the events with frequency 11 are still included in the dataset. Are those important? Should they be considered or deleted as well? Again, assumptions and decisions must be documented to best interpret the results.

Converter Users

In this case, we are looking at processes for users that started a trial version of the application and converted into real customers. For this process, the starting event is “Trial Sign Up Completed” and the finish event is “Purchase Confirm Details”. In order to get only the dataset that include those converter users, the following functions are invoked:

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

```
deleteTruncatedTracesEnd(EventLogs, company_id, event, "Purchase
Confirm Details") -> EventLogs_converters
```

The total number of records for the converter dataset is 144,255 records/rows. The following dataset was imported to Disco to build the process map (Figure 20).

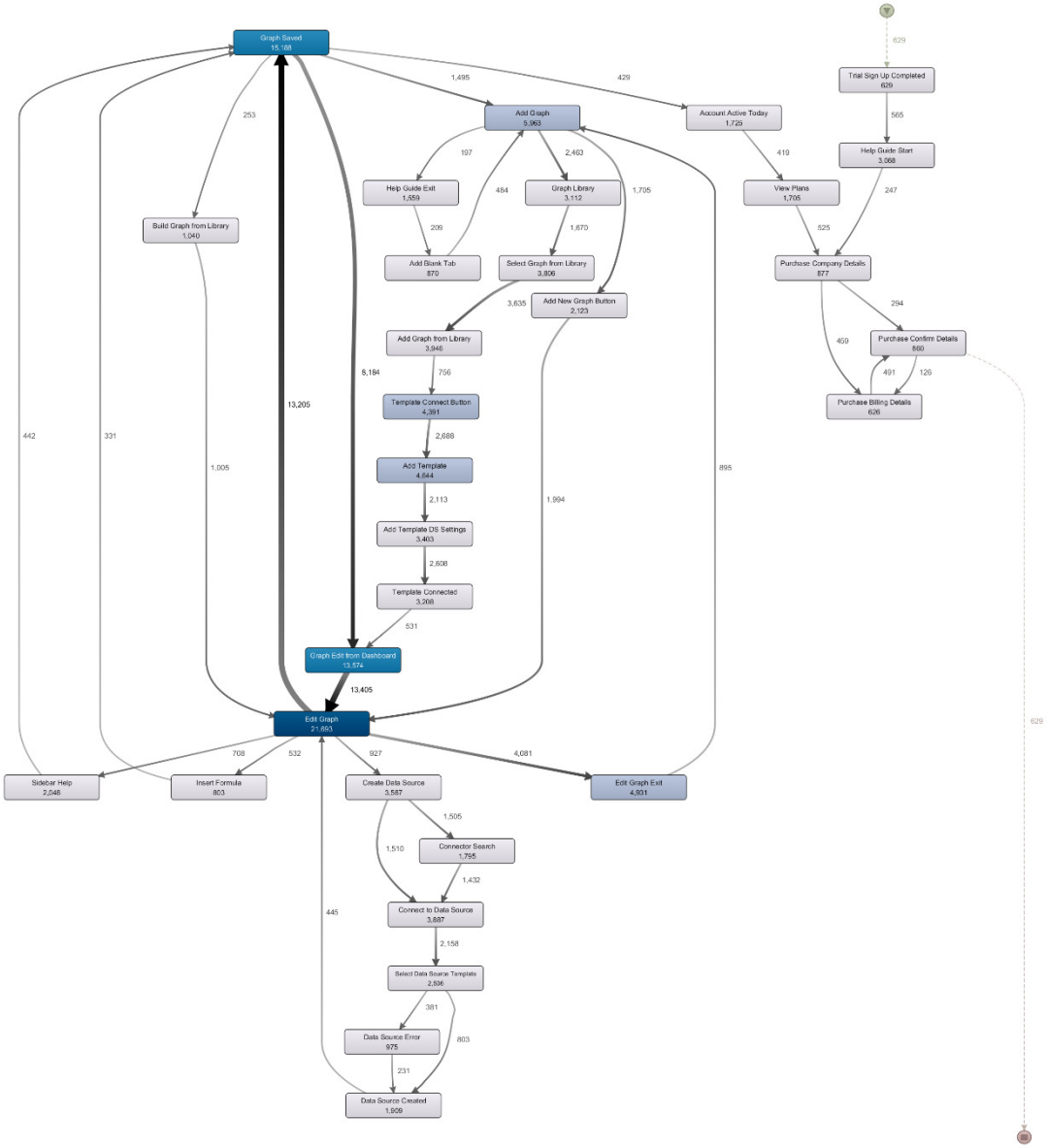

Figure 20: Process map generated for users converted to real customers.

The most frequent path in this process is:

Graph Saved → Graph Edit from Dashboard → Edit Graph.

There are in total 629 cases with 550 variants. When I filter the dataset according to performance, that is, by filtering the cases according to case duration (which is the total time it took for a user to execute the whole process from start to finish), 30% of the total cases converted within less than 5 minutes. There are 96 cases with a total of 30 variants here (Figure 21).

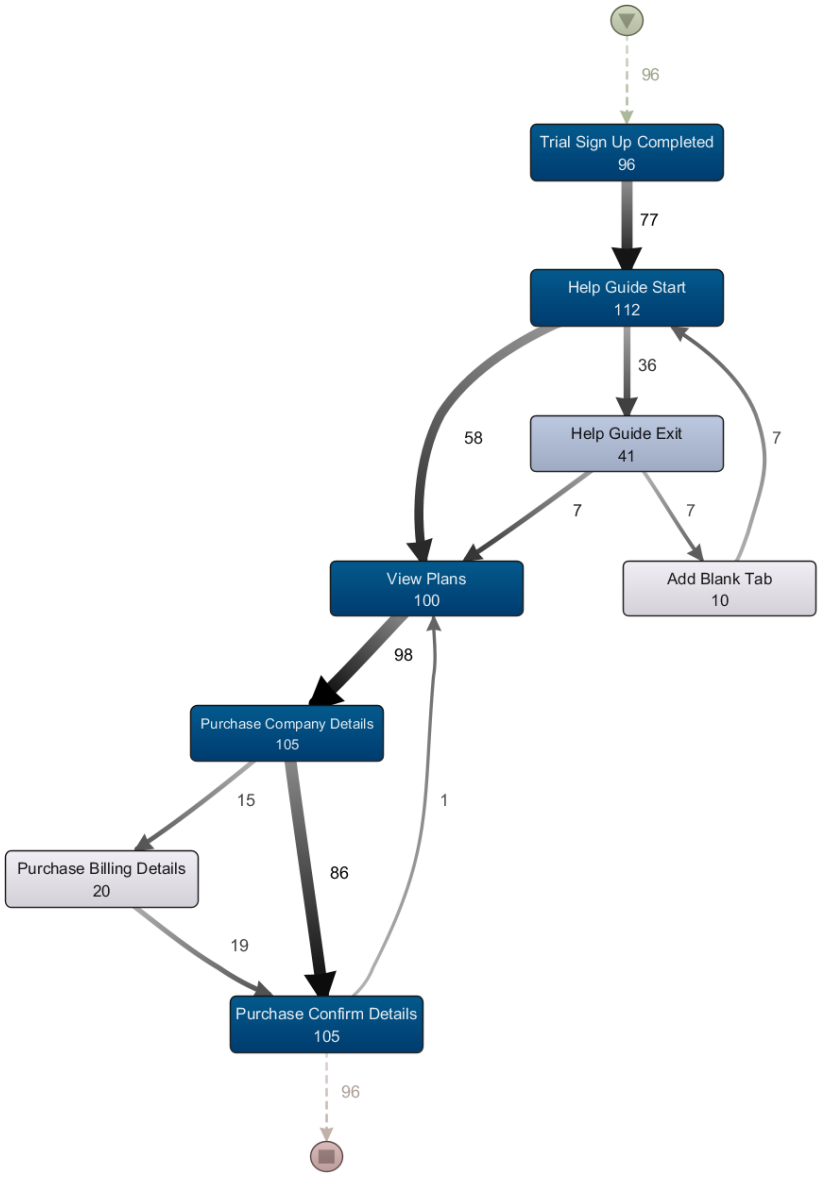

Figure 21: Process map for converter users.

The most frequent path in this case is:
Trial Sign Up completed → Help Guide Start → View Plans → Purchase Company Details → Purchase Confirm Details.

Additionally, if we look at more statistics, the highest conversion rate happens on Tuesdays (24%) and Wednesdays (19%), and the least ones on Saturdays (7%) and Sundays (6%).

As another example, there are 26% of the cases that converted between 1 day and 14 days duration (Figure 22). The most frequent path according to the process map generated is *Graph Saved → Graph Edit from Dashboard → Edit Graph*. In addition, 50% of the cases took more than 14 days to convert, with a maximum duration of 90 days and 13 hours.

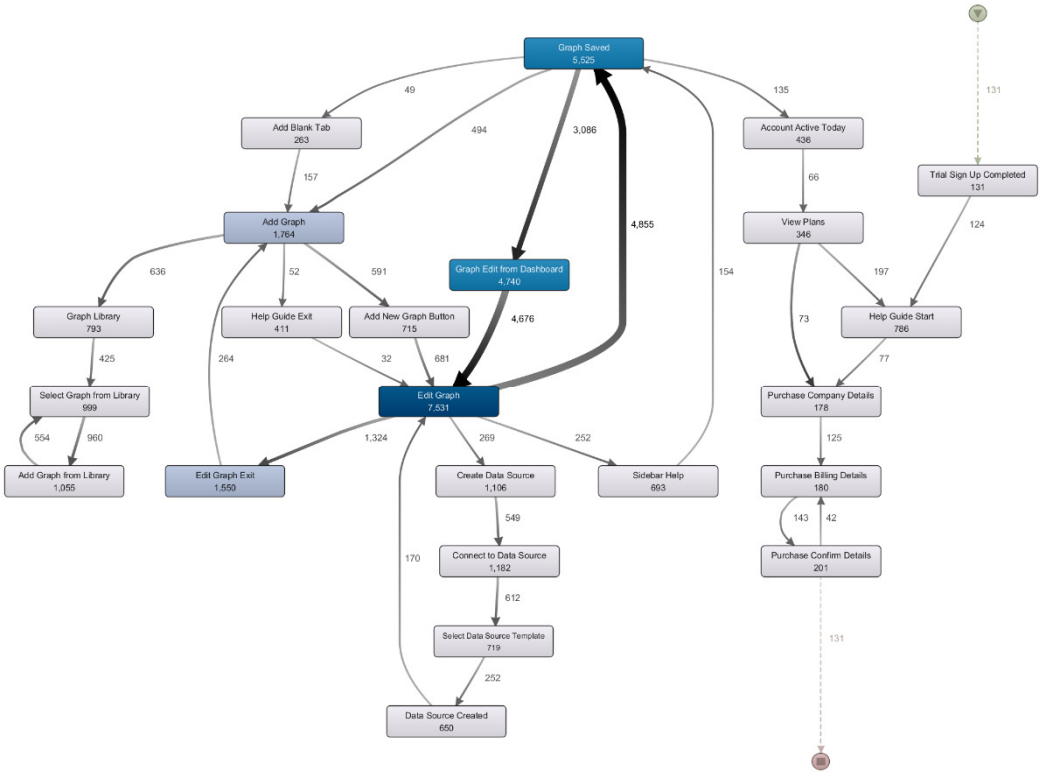

Figure 22: Process maps for converter users with case duration less than 14 days.

The process map in Figure 22 indicates that some sequences of events are occurring in a certain order, e.g., the path *Create Data Source → Connect to Data Source → Select Data Source Template → Data Source Created*. If we aggregate all these events together, regardless of the number of times they occurred in this order, we will be able to reduce the number of variants for the different cases. Another example is the path *Graph Saved → Graph Edit from Dashboard → Edit Graph*, which is the most frequent path in this scenario; the analyst might decide to delete those events from the analysis as they are known to occur multiple times because those are the main activities executed by the user. The other option is to aggregate those three events into one event, which would also reduce the number of variations within the same process.

Non-converters Users

In this context, we are looking at users that did not convert within the timeframe for when the data was collected (Figure 23).

In terms of the time duration for the processes, 76% of the total cases have a duration shorter than 5 minutes. The process map in Figure 24 displays the paths executed by the non-converter users. The most frequent path is *Trial Sign Up Complete → Help Guide Start → Help Guide Exit*.

There are some users that executed between 30 to 60 events between 2 minutes and 5 minutes. Some of the cases that were executed for less than a minute include:

- *Trial Sign Up Completed → Help Guide Start → Help Guide Exit → Help Guide Start*
- *Trial Sign Up Completed → Help Guide Start → Help Guide Exit → Help Guide Start → Add Blank Tab*
- *Trial Sign Up Completed → Help Guide Start → View Plans*
- *Trial Sign Up Completed → Help Guide Start → Data Sources*

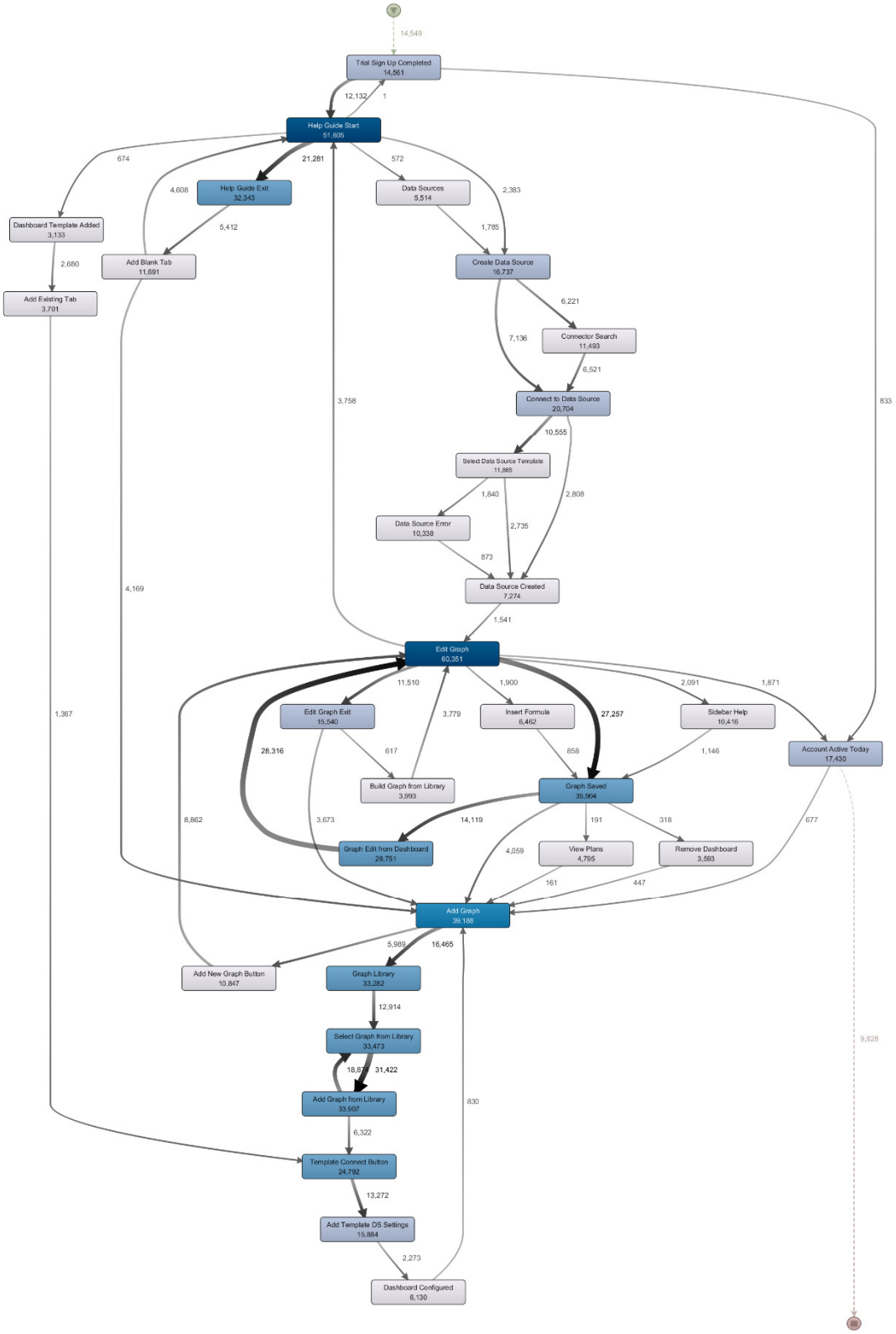

Figure 23: Process map for non-converter users after initial preprocessing.

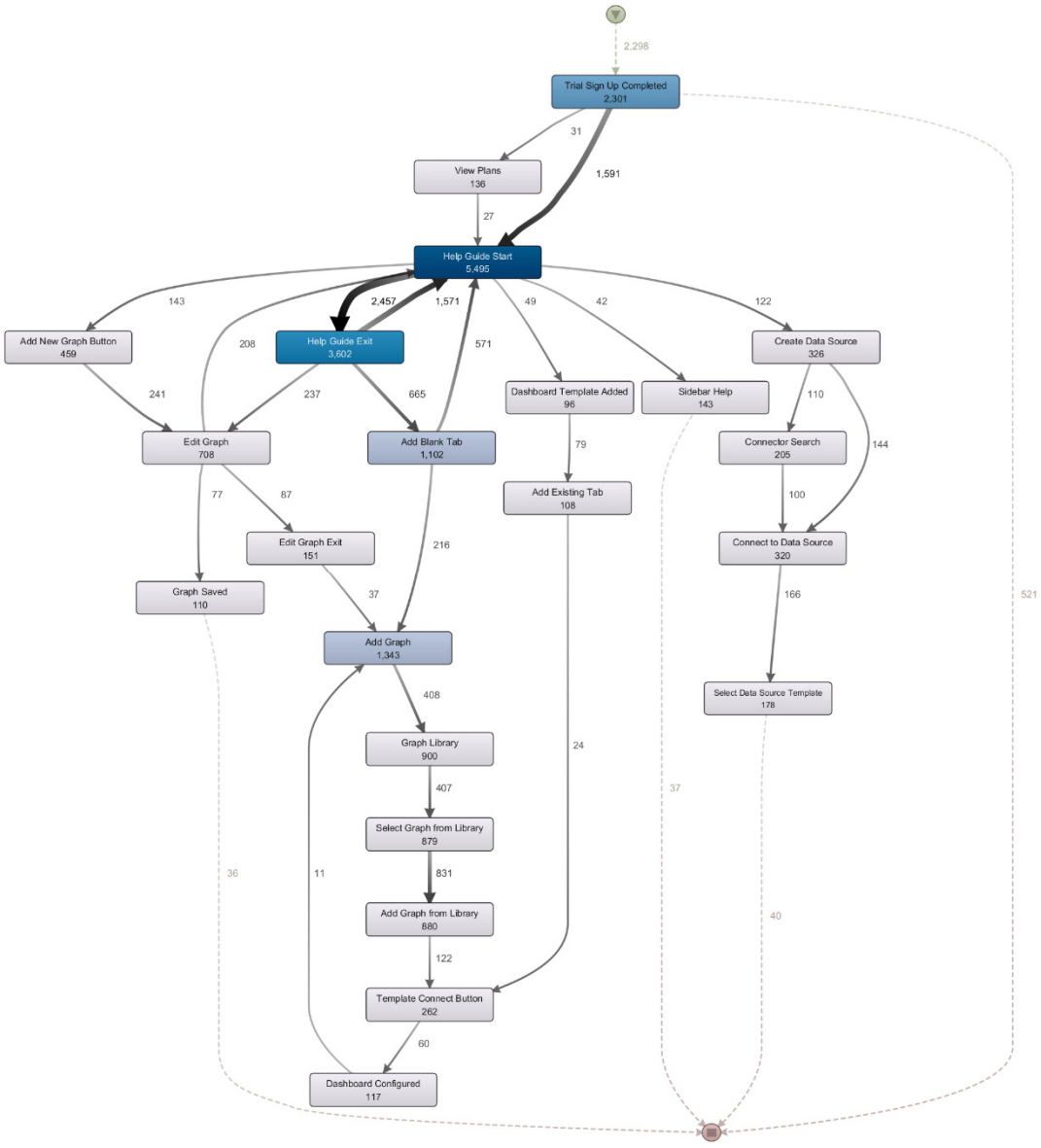

Figure 24: Process map for non-converters with total duration less than 5 minutes.

By filtering for case duration and looking at the cases that were running for more than a day and less than 14 days, we get 17% of the total cases. As shown by the process map in Figure 25, when users are spending more time using the application, the most frequent paths include the following activities: *Edit Graph, Graph Saved, Add Graph, Graph Library, Graph Edit from Dashboard, Add Graph from Library.*

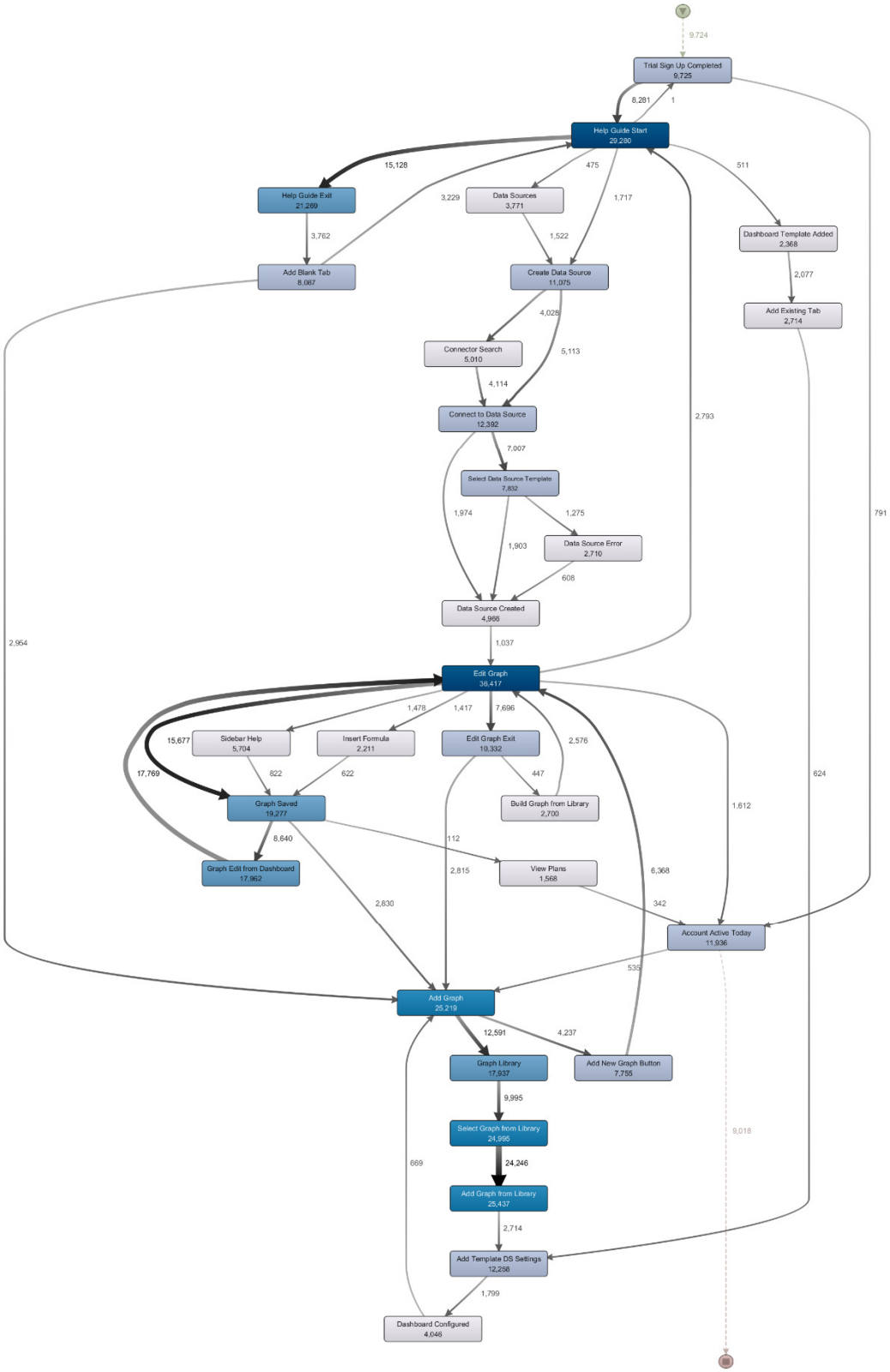

Figure 25: Process map for non-converter users with case duration between 1 and 14 days.

The results that were provided in this section are only some examples of how process mining can help answering different analysis questions about processes and the steps users are taking when using a cloud-based application. More analysis can be done after the dataset is imported to Disco or any other process mining tool. For example, in Disco, several filtering options are available: filtering by different attributes, endpoints, subsequences in a process, number of cases, number of events per case, and process variants. These filters can be applied in a certain order and through several iterations. Those decisions are taken by the analysis according to the analysis questions.

6.5.2 ProM Usage

ProM provides several algorithms that can be used to discover process maps, each with some benefits and drawbacks. While discovering the process model for a clickstream data set, various PM discovery algorithms can be used. A process model must offer a balance between fitness, precision, generalization, and simplicity. Because clickstream event logs are generally unstructured, a PM algorithm able to handle loops and parallelism is likely beneficial. The *alpha miner* algorithm (van der Aalst et al., 2004), which is not able to detect loops and parallel activities, is not entirely suitable for clickstream event logs.

On the other hand, the *heuristic miner* algorithm (Weijters et al., 2006) is able to detect loops and handle noise. It mainly takes frequencies into account, detect short loops, and detects skipping activities. The Heuristic Miner algorithm works on mining the control flow of activities with a process model. It only considers the order of events for each case (Weijters et al., 2006).

The *fuzzy miner* (Günther & van der Aalst, 2007) is able to deal with an unstructured sequence of activities and it outputs a process graph, which is very easy to understand, but it does not differentiate between choices and parallelisms.

The *inductive miner* (Leemans et. al, 2013) is suitable when we are interested in discovering choices and parallelism between activities in a clickstream dataset.

6.6. A Newer 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 changes should be made to the script in order to adapt it to new, more recent dataset. This second version of the script is available in Appendix A.3.

The new dataset was collected from the period of January 2019 – August 2019, and it includes 21 different variables/attributes, 2,144,210 different 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 more 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.

Figure 26 shows the process map of this new event log, before doing any preprocessing. Disco was used to visualize the most frequent activities (73% of the activities) and paths (15% of the total paths); the complete map would actually be much more complex than this figure.

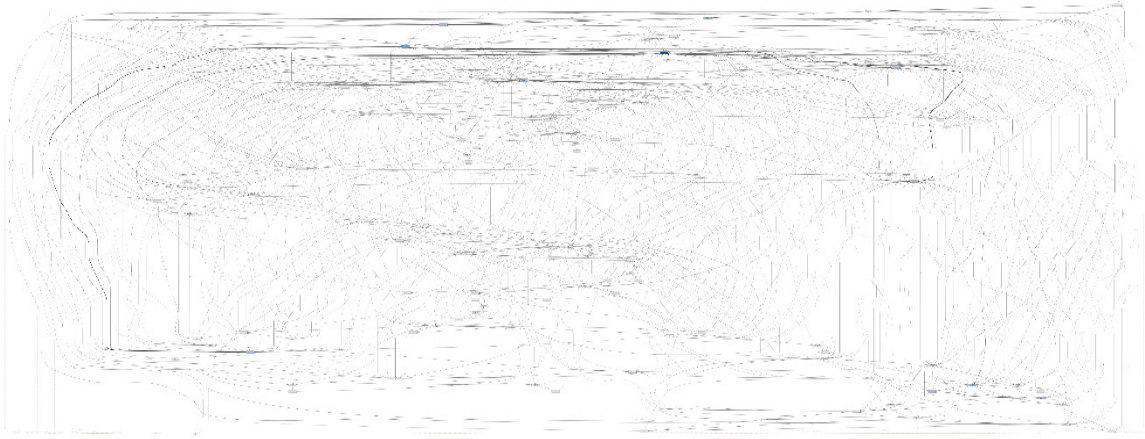

Figure 26: Process map representing the new dataset event logs.

The initial preprocessing script was applied again on this dataset. Here are the steps that were applied:

1- Read the CSV file dataset into a dataframe.

```
EventLogs <- read.csv(file = "dataset.csv",header=TRUE,sep=",")
```

2- Clean the headers of the columns.

```
cleanHeaders(EventLogs) -> EventLogs
```

3- Select columns/attributes needed for analysis.

```
selectColumns(EventLogs,
      company_id ,event, time, account_type, account_status,
      number_of_clients, client_status) -> EventLogs
```

4- Filter the records that do not have company_id values.

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

5- Remove the records with event *View Dashboard*.

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

The number of records was reduced to 1,821,499.

6- Remove events with very low frequency.

```
removeEventsLowFrequency(EventLogs, event, 20) -> EventLogs
```

The number of records was reduced to 1,821,472.

7- Remove incomplete traces that do not have the start event as *Trial Sign Up Started.*

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

The number of records was reduced to 1,788,326.

8- Remove traces that include less than 2 events.

```
deleteTraceLengthLessThan(EventLogs, company_id, 2)
      -> EventLogs
```

The number of records was reduced to 1,778,321.

9- Arrange dataset according to the time attribute.

```
EventLogs %>% arrange(company_id, time) -> EventLogs
```

The resulting process map, presented in Figure 27, visualized the new dataset event logs after applying the CPA-PM method.

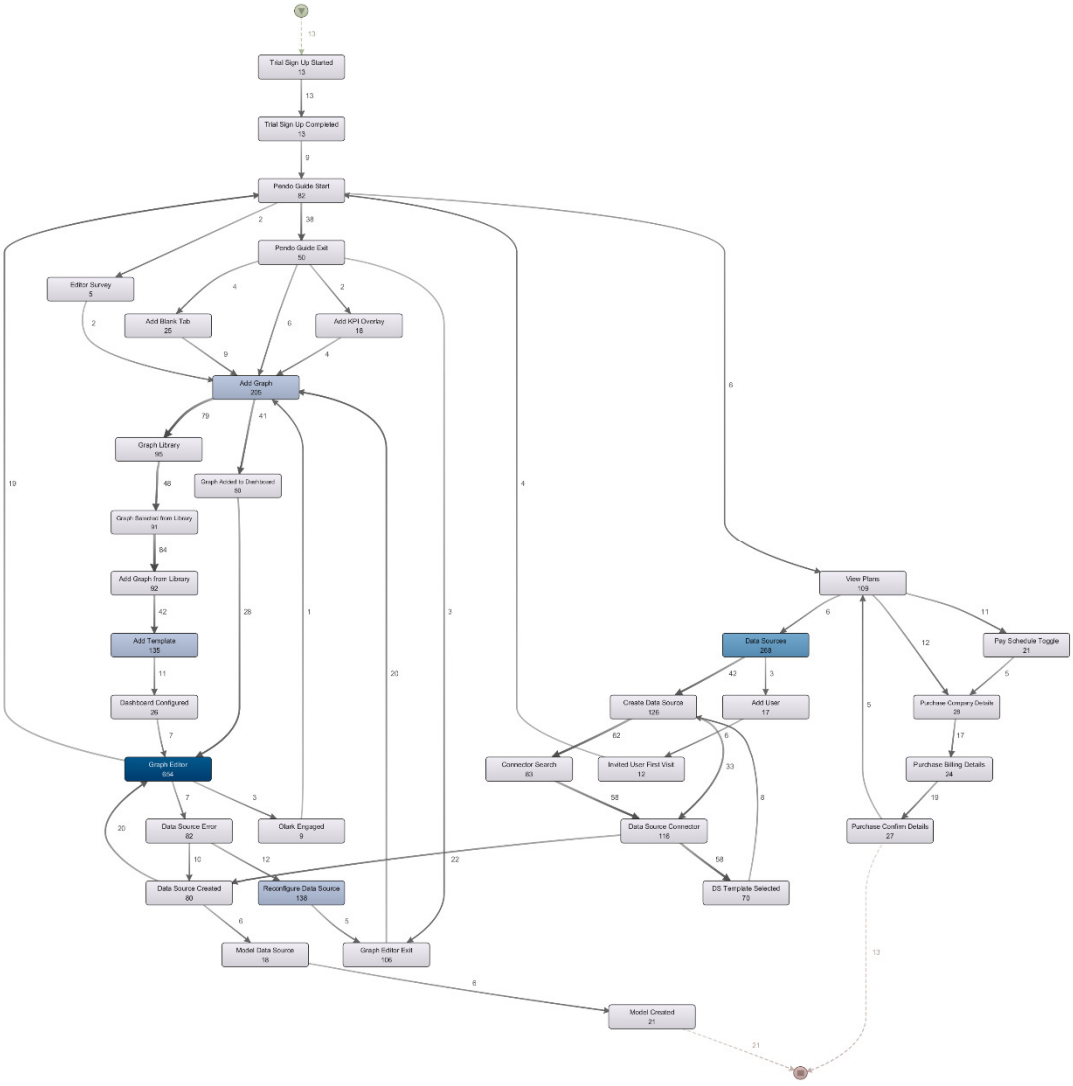

Figure 27: Process map representing the new dataset event log after preprocessing.

The same functions that were used to preprocess the first dataset, were also used to preprocess the second dataset. The CPA-PM method was also used on the second dataset by only modifying the event names accordingly, even though the second dataset was collected for a longer timeframe and included more events than then first one. After applying the CPA-PM method, the event logs were reduced, and the process maps were less complex. Redundant repetitions of events that would not add meaningful information for the analysis were removed, which resulted in a much simpler dataset. The noisy event logs including loops, truncated traces, cases with very short duration, events with low frequencies, and cases

with missing ID were deleted from the dataset, which removed many records that are useless for the analysis and for answering the analysis questions. Substituting patterns in the dataset reduced the number of case variants, which in turn reduced the complexity of the generated process map, because many rows were being merged into one row.

6.7. Chapter Conclusion

This chapter illustrated how the CPA-PM method can be used to answer analysis questions on two different datasets provided by a SaaS company. It also listed the steps taken to simplify and preprocess the event logs to generate more meaningful process maps. The chapter also highlighted some of the questions that can be answered. It finally emphasized where it is up to the analyst to make decisions on the processes that should be analyzed and what events to keep in the dataset.

Chapter 7. Analysis and Discussion

This chapter analyses the CPA-PM method, which is proposed to help analysts better preprocess event logs. Additionally, a scalability analysis for the method is performed. Lastly, this chapter discusses the main threats to validity related to this thesis and how some were mitigated.

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 analysis purpose. These *spaghetti* processes are not incorrect per se. The problem however is that such process maps are not useful for discovery, since they are too complicated 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 contain millions of events capturing user clicks anywhere on cloud application pages, as well as page refreshes. The CPA-PM method proposed in this thesis can help in cleaning such datasets in a suitable way in order to build more meaningful process maps. After applying the steps of the CPA-PM method, the resulting process map are less complex and more meaningful 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 functions used to support the CPA-PM steps for a particular event log can be collected as a script that can be invoked 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 thesis, Disco was mainly used to display process maps, and ProM was investigated as well.

The analyst 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, the analyst may discover that some activities that should be present are actually not displayed in the process map since they are not being logged in the first place.

7.2. Scalability Analysis

As the number of records in event logs increases, the resulting process maps 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 as 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 Table 22 and visualized in Figure 28.

Table 22: Script execution results.

Dataset #	Number of records per dataset	Average runtime for 3 iterations
1	6,250	1 min 25 sec
2	12,500	2 min 45 sec
3	25,000	3 min 35 sec
4	50,000	4 min 46 sec
5	100,000	5 min 10 sec
6	200,000	5 min 45 sec
7	400,000	9 min 13 sec
8	800,000	11 min 25 sec
9	1,600,000	14 min 34 sec
10	2,144, 210	18 min 12 sec

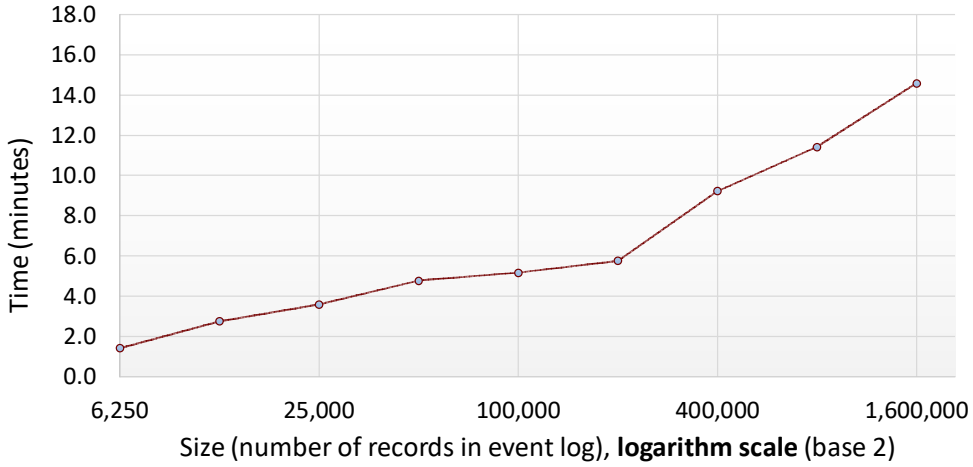

Figure 28: Script execution performance.

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 Table 22 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 removing many of the records.

In some situations, the dataset might include more attributes (higher number of columns), which would require more computations when merging the rows and hence result in longer runtime. The impact of such additional attributes was not assessed here. In situations where more pattern iterations are executed, the number of rows would be reduced, which might shorten the average runtime. Again, this situation was not tested here.

Still, in general, having to wait 10 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 identified and discussed here. Other threats to validity related to the systematic literature review were discussed in section 3.4.

7.3.1 Construct Validity

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 thesis 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. In order to mitigate some of these threats, a second dataset was provided to the method and script could be used again. The limitation remains that the new dataset was provided by the same stakeholder and from the same system, just from a different period of time.

7.3.2 Internal Validity

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 the thesis author perform the evaluation and analyze the results of the case study. A similar issue occurs in the SLR, because it was mainly done by one person. This threat was mitigated by having my supervisor also evaluating 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 the API themselves.

7.3.3 External Validity

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. Another mitigation would be to get datasets from different cloud eco-systems and infrastructures.

7.4. Chapter Summary

This chapter presented a brief analysis of the CPA-PM method and it effects on building process maps from cloud-based applications. A simple scalability analysis, where a real script was used on real datasets of different sizes, also suggests that the method and its API can be used in practice to shorten the time needed to perform the necessary preprocessing steps prior to process mining and analysis. Lastly, this chapter highlights different threats to the validity of this research, with current and future mitigations.

Chapter 8. Conclusion and Future Work

This chapter summarizes the contributions of this thesis and answers to its three research questions. In addition, it discusses some research opportunities and future directions to improve the preprocessing of different dataset qualities and the challenges that come with such datasets.

8.1. Contributions

In order to properly use process mining from event logs collected from cloud-based applications, intensive and iterative preprocessing of such logs must be done in order to minimize datasets and simplify the resulting mined processes, without loss of analysis power.

This thesis introduced a CPA-PM method to preprocess event logs before feeding them to process mining tools and building process maps. This method includes the different steps needed to better clean and restructure cloud-based event logs, which come with their own set of specific challenges, including a large number of events and attributes, spurious and repetitive events, and unstructured orderings of events. The method is supported by a set of functions, grouped as an API, with an implementation in the R language.

The last contribution of this thesis is an SLR that highlights how process mining has been used to better understand and study processes generated from cloud-based systems, with the challenges specific to that context.

8.2. Answers to Research Questions

The thesis answers three main research questions related to the application of process mining (PM) techniques in cloud-based systems and the challenges encountered.

RQ1. How is process mining utilized on cloud-based systems?

Answer: The SLR in Chapter 3 describes how process mining is being applied to analyze cloud processes in different types of applications, including the use of clickstream data. The SLR analyzes 27 different research papers collected from three different databases and discusses application areas and the relevant techniques/tools used along the way. The results also show a significant increase, in the last two years, in the number of publications about process mining for cloud systems, which suggests an increasing interest in this research area. Out of 27 papers, 17 papers used ProM as PM tool and 2 papers used Disco. In terms of applications, 13 papers looked at cloud infrastructure processes, were concerned with 12 business process management, 5 looked at clickstream data from e-commerce applications, 3 used PM techniques in cloud-based software development processes, and only 2 papers discussed the challenges faced when preprocessing clickstream data.

In terms of the concepts seen in Chapter 3 and used to assess the literature in Table 2, this thesis provides additional challenges (C3) as well as new insights related to clickstream data from e-commerce applications (C4) to the few experiences reported in prior publications. The thesis also adds a new case study (C7) where the Disco tool is used (C2). Table 23 captures the line corresponding to this thesis as an addition to Table 2.

Table 23: Concept categorization of this thesis according to Chapter 3.

Concept						
C1	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**
	Y	**Y**	**Y**			**Y**

RQ2. What are the challenges encountered when discovering process models from cloud-based application processes?

Answer: Several challenges more specific to cloud-based applications were observed when conducting the SLR in Chapter 3. Those challenges, which go beyond the conventional ones observed in the application of process mining to business processes, include:

1. Cleaning and merging raw event logs dataset.
2. Dealing with complex event logs having many characteristics/attributes.

3. Applying PM analysis on event logs in real time.
4. Applying PM analysis on less structured processes and improving the representation of such process maps.
5. Combining PM analysis with other types of analysis.

Other challenges were discovered when applying the CPA-PM method on real datasets that represent clickstream data collected from cloud-based visualization applications, including dealing with:

6. Noisy event logs in clickstream datasets.
7. Similar events that must be split according to some attributes into different events.
8. Missing some unique identifiers for some processes.
9. The frequent changes in processes and their events, which might change the structure of the event logs that are collected.
10. Events whose ordering is not important, for example refreshing a page.

This thesis proposes solutions particularly applicable to problems 1, 2, 4, 6, 7, 9, and 10.

RQ3. What is a suitable method to discover and analyze cloud-based application processes?

Answer: A *Cloud Pattern API- Process Mining* method was proposed in Chapter 5 to preprocess event logs generated from cloud-based and SaaS applications. This method consists of three stages to preprocess event logs: dataset cleaning, dataset restructuring, and pattern substitutions. The method steps are supported by API functions aiming to automate event log preprocessing. These functions are implemented in R and validated through a real dataset provided by a SaaS company.

The method's *suitability* for SaaS applications, as defined in Section 1.2, was demonstrated in the case study of Chapter 6 by providing:

- high automation through the method being scriptable, an original feature of this method that enables automation of pre-processing in one place, independently of the PM technology used to discover, visualize, and analyze processes;
- high abstraction by showing that it handles low-structured processes like click-stream applications;
- high scalability by the efficient handling of logs with over a million events to understandable process models that fit on a page;
- and low effort to accommodate changes in the scripts needed to handle drifting and evolving datasets and processes.

8.3. Limitations

Several limitations were discussed in section 7.3. Both datasets included in the case study were provided by the same stakeholder, for the same process, and from the same environment and infrastructure. The method should be applied on datasets collected from applications that are running in different cloud environments and from different providers for example Microsoft Azure, Amazon Web Services, and Google Cloud Platform. Additionally, event logs extracted from different log management tools might need more preprocessing depending on the quality of the raw data. The method was not applied on different types of processes. For example, in some cases it is important to keep the events with low frequencies since they might be interesting for the analyst as outliers.

The stakeholders did not apply the method themselves, which remains a limitation of this thesis. The usability of the method was not assessed by analysts to check how easy it is to understand the different functions and apply them on different datasets.
An additional limitation, is that the resulted preprocessed event logs were only imported into the Disco (mainly) and ProM PM tools.

8.4. Future Work

There are many opportunities to extend this research work:

1. To address some of the limitations from section 8.3, the CPA-PM method can be applied on datasets collected from different cloud infrastructure environments. 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 (e-commerce applications, banking, online publishing, online booking, online auction, etc.). The method and its API might need to be changed or adjusted according to the application type and the actions that users perform while using such applications.

2. The CPA-PM method and its API implementation could be extended to start from an earlier phase, e.g., to handle multiple data sources (instead of assuming the existence of a single CSV input file). In some cases, the same process might be recorded in different log management tools and these event logs should be combined and matched using the identifiers.

3. Some of the functions could likely be generalized. For example, the two functions proposed to delete truncated traces in event logs (section 5.4.9) could be extended to support multiple start events or end events.

4. The API functions could be also implemented in other programming languages, for example Python or SAS.

5. The API performances might also be improved with hand-made functions that do not rely on external libraries, and perhaps with support for real-time preprocessing.

6. The API could include functions that would discover patterns of events automatically rather than having the analyst find them.

7. The method and API could be tested on IaaS and PaaS levels, in addition to the current SaaS level, in order to generalize them to a wider range of cloud-based contexts.

8. This research would benefit from a usability study of the method and its API. Also, getting regular feedback from stakeholders and analysts to have a better understanding about their needs, the type of processes the analyze, and how well the method works for them would help support the continuous improvement of the method and its API.

References

Acampora, G., Bernardi, M., Cimitile, M., Tortora, G., & Vitiello, A. (2017). A Fuzzy-based Autoscaling Approach for Process Centered Cloud Systems. *IEEJ Transactions on Electrical and Electronic Engineering, 13*(11), 1624-1632.

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, et al. (2010). A view of cloud computing. *Communications of the ACM, 53*(4), 50-58.

Bernardi, M. L., Cimitile, M., & Maggi, F. (2015). Discovering cross-organizational business rules from the cloud. *IEEE SSCI 2014 - 2014 IEEE Symposium Series on Computational Intelligence - CIDM 2014: 2014 IEEE Symposium on Computational Intelligence and Data Mining, Proceedings*, pp. 389-396.

Bernardi, M. L., Cimitile, M., & Mercaldo, F. (2018). Cross-organisational Process Mining in Cloud Environments. *Journal of Information & Knowledge Management, 17*(2), 1-27.

Bobrowski, S. (2011). Optimal multitenant designs for cloud apps. *2011 IEEE 4th International Conference on Cloud Computing,* pp. 654-659.

Bose, J. C., & van der Aalst, W. (2009). Abstractions in Process Mining: A Taxonomy of Patterns. *International Conference on Business Process Management,* LNCS 570, Springer, pp. 159-175.

Buijs, J., van Dongen, B., & van der Aalst, W. (2012). Towards Cross-Organizational Process Mining in Collections of Process Models and Their Executions. *Business Process Management Workshop. BPM 2011*. LNBIP 100, part 2, pp. 2-13.

Caldeira, J., & Abreu, F. (2016). Software Development Process Mining: Discovery, Conformance Checking and Enhancement. *2016 10th International Conference on the Quality of Information and Communications Technology*, pp. 254-259.

Chesani, F., Ciampolini, A., Loreti, D., & Mello, P. (2017). Map reduce autoscaling over the cloud with process mining monitoring. *6th International Conference on Cloud Computing and Services Science*, pp. 109-130.

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

Diamantini, C., Genga, L., & Potena, D. (2016). Behavioral process mining for unstructured processes. *Journal of Intelligent Information Systems, 47*(1), 5-32.

El-Gharib, N.M., & Amyot, D. (2019). Process mining for cloud-based applications: A systematic literature review. *9th International Workshop on Model-Driven Requirements Engineering (MoDRE)*, IEEE CS, pp. 34-43.

Evermann, J., Rehse, J., & Fettke, P. (2017). Process discovery from event stream data in the cloud - A scalable, distributed implementation of the flexible heuristics miner on the Amazon kinesis cloud infrastructure. *Proceedings of the International Conference on Cloud Computing Technology and Science*, pp. 645-652.

Ghasemi, M. (2018). What Requirements Engineering can Learn From Process Mining. *Proceedings - 2018 1st International Workshop on Learning from other Disciplines for Requirements Engineering, D4RE 2018*, pp. 8-11.

Ghasemi, M. & Amyot, D. (2016). Process Mining in Healthcare: A Systematised Literature Review. *International Journal of Electronic Healthcare,* 9(1), 1-28.

Ghasemi, M. & Amyot, D. (2019). From event logs to goals: a systematic literature review of goal-oriented process mining. *Requirements Engineering,* 1-27 (to appear).

Ghavamipoor, H., Golpayegani, S., & Shahpasand, M. (2017). A QoS-sensitive model for e-commerce customer behavior. *Journal of Research in Interactive Marketing, 11*(4), 380-397.

Günther, C.W., & van der Aalst, W.M. (2007). Fuzzy mining-adaptive process simplification based on multi-perspective metrics. *International conference on business process management*, LNCS 4714, Springer, pp. 328-343.

Haung, K. Z., Chen, Y., & Chung, Y. (2011). Perspectives on process mining within cloud computing. *2011 3rd International Conference on Advanced Computer Control, ICACC 2011*, pp. 656-660.

Huang, T., Xu, B., Cai, H., Du, J., Chao, K. -., & Huang, C. (2018). A fog computing based concept drift adaptive process mining framework for mobile APPs. *Future Generation Computer Systems, 89*, 670-684.

Hevner, A. R., March, S., Ram , S., & Park, J. (2004). Design Science in the Information Systems Discipline. *MIS Quarterly, 32*(4), 725-730.

ISO/IEC (2019). *ISO/IEC 15909-1:2019 Systems and software engineering — High-level Petri nets — Part 1: Concepts, definitions and graphical notation*. https://www.iso.org/standard/67235.html

Janes, A. (2017). Test Case Generation and Prioritization: A Process-Mining Approach. *10th IEEE International Conference on Software Testing, Verification and Validation Workshops*, pp. 38-39.

Kalenkova, A. A., Ageev, A. A., Lomazova, I. A., & van der Aalst, W. M. P. (2018). E-government services: Comparing real and expected user behavior. *Business Process Management Workshops. BPM 2017*. LNBIP 308, pp. 484-496.

Kebede, M., & Dumas, M. (2015). *Comparative evaluation of process mining tools*. Master's thesis, Faculty of Mathematics and Computer Science, University of Tartu, Finland.

Kitchenham, B., & Charters , S. (2007). *Guidelines for performing Systematic Literature Reviews in Software Engineering,* Technical report, Ver. 2.3, EBSE, UK.

Leemans, S. J., Fahland, D., & van der Aalst, W. M. (2013). Discovering block-structured process models from event logs containing infrequent behaviour. *Business Process Management Workshops*, LNBIP 171, Springer, pp. 66-78.

Liu, C., Wang, S., Gao, S., Zhang, F., & Cheng, J. (2018). User Behavior Discovery from Low-Level Software Execution Log. *IEEJ Transactions on Electrical and Electronic Engineering, 13*(11), 1624-1632.

Liu, C., Zhang, Li, G., Gao, S., & Zeng, Q. (2018). A two-layered framework for the discovery of software behavior: A case study. *IEICE Transactions on Information and Systems, 101*(8), 2005-2014.

Padidem, D. K., & Nalini, C. (2017). A study on classification of users shopping behavior process model using click stream data. *Journal of Engineering and Applied Sciences, 12*(12), 9548-9553.

Poggi, N., Muthusamy, V., Carrera, D., & Khalaf, R. (2013). Business process mining from e-commerce web logs. *11th International Conference on Business Process Management*, LNCS 8094, pp. 65-80.

OMG (2014). *Business Process Model and Notation*, version 2.0.2. OMG formal/13-12-09, January 2014. https://www.omg.org/spec/BPMN/

ProM Tools. [Online] Available at: http://www.promtools.org/doku.php [Accessed June 2019].

Rozinat, A. (2016). *Disco User's Guide.* [Online]. Available at: https://fluxicon.com/disco/files/Disco-User-Guide.pdf [Accessed March 2019].

Sahlabadi, M., Muniyandi, R., & Shukur, Z. (2014). Detecting abnormal behavior in social network websites by using a process mining technique. *Journal of Computer Science, 10*(3), 393-402.

Song, J., Luo, T., & Chen, S. (2008). Behavior pattern mining: Apply process mining technology to common event logs of information systems. *IEEE International Conference on Networking, Sensing and Control, ICNSC*, pp. 1800-1805.

Song, W., Chen, F., Jacobsen, H., Xia, X., Ye, C., & Ma, X. (2017). Scientific workflow mining in clouds. *IEEE Transactions on Parallel and Distributed Systems, 28*(10), 2979-2992.

Sutrisnowati, R. A., Bae, H., Pulshashi, I., Putra, A., Prastyabudi, W., Park, S., et al. (2015). BAB Framework: Process Mining on Cloud. *The Third Information Systems International Conference BAB*, pp. 453-460.

Terragni, A., & Hassani, M. (2018). Analyzing Customer Journey with Process Mining: From Discovery to Recommendations. *2018 IEEE 6th International Conference on Future Internet of Things and Cloud*, pp. 224-229.

Vaishnavi, V., Kuechler, W., and Petter, S. (Eds.) (2004/19). *Design Science Research in Information Systems*. January 20, 2004, last updated June 30, 2019. URL: http://www.desrist.org/design-research-in-information-systems/.

van der Aalst, W. M. (2010). Configurable Services in the Cloud: Supporting Variability While Enabling Cross-Organizational Process Mining. *9th Confederated*

International Conferences on On the Move to Meaningful Internet Systems, LNCS 6426, pp. 8-25.

van der Aalst, W. M. P. (2011a). *Process Mining: Discovery, Conformance and Enhancement of Business Processes.* Berlin, Heidelberg: Springer.

van der Aalst, W. (2011b). Intra- and inter-organizational process mining: Discovering processes within and between organizations. *IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling*, pp. 1-11.

van der Aalst, W. M. P. (2011c). Business process configuration in the cloud: How to support and analyze multi-tenant processes? *2011 IEEE Ninth European Conference on Web Services*, pp. 3-10.

van der Aalst, W. M.P. (2012a). Process Mining: Overview and Opportunities. *Process-Aware Information Systems: Bridging People and Software through Process Technology, 3*(2), 235-255.

van der Aalst W.M.P. (2012b). Distributed Process Discovery and Conformance Checking. *International Conference on Fundamental Approaches to Software Engineering*, Springer, pp. 1-25.

van der Aalst, W., Weijters, A., & Maruster, L. (2004). Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering, 16*(9), 1128 - 1142.

van Dongen, B., de Medeiros, A., Verbeek, H., Weijters, A., & van der Aalst, W. (2005). The ProM Framework: A New Era in Process Mining Tool Support. *26th International Conference on Applications and Theory of Petri Nets 2005, ICATPN 2005*, LNCS 3536, pp. 444-454.

van Eck, M., Lu, X., Leemans, S., & van der Aalst, W. (2015). PM2: a Process Mining Project Methodology. *27th International Conference on Advanced Information Systems Engineering CAiSE 2015*, LNCS 9097, pp. 297-313. Stockholm; Sweden.

van Eck M.L., Sidorova N., van der Aalst W.M.P (2016, September). Discovering and Exploring State-Based Models for Multi-perspective Processes. *International Conference on Business Process Management*, Springer, pp. 142-157.

Weijters, A.J.M.M., van Der Aalst, W.M., & De Medeiros, A.A. (2006). Process mining with the heuristic miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP, 166,* 1-34.

Xu, B., Xu, L., Cai, H., Jiang, L., Luo, Y., & Gu, Y. (2017). The design of an m-Health monitoring system based on a cloud computing platform. *Enterprise Information Systems, 11*(1), 17-36.

Xu, X., Zhu, L., Weber, I., Bass, L., & Sun, D. (2014). POD-Diagnosis: Error diagnosis of sporadic operations on cloud applications. *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 252-263.

Appendix A: Application Programing Interface

This appendix lists the functions in the API and its implementation as an R library. It also provides a sample script that uses the API functions, and a revised version that adapts the first scripts to handle a new version of the event log, provided many months later. Please note that this implementation relies on several libraries:

- **dplyr**: a grammar for data manipulation [8].
- **plyr**: set of tools for splitting, applying, and combining data [9].
- **janitor**: set of tools for examining and cleaning dirty data [10].
- **tidyverse:** set of packages that are used in data analysis, including ggplot2, tidyr, readr, and stringr [11].

A.1. List of Functions

```
#Import libraries (needed for the API to run)
library(dplyr)
library(plyr)
library(janitor)
library(tidyverse)

#read a CSV file
readCSV <- function(filename){
      read.csv(file=filename,header=TRUE,sep=",")
}

#write to CSV file
writeCSV <- function(.dataset, filename){
      write.csv(dataset, file=filename)
}
#clean column headers
cleanHeaders <- function(dataset){
      dataset %>% clean_names() -> dataset
}
```

[8] https://www.rdocumentation.org/packages/dplyr/versions/0.7.8
[9] https://www.rdocumentation.org/packages/plyr/versions/1.8.4
[10] https://www.rdocumentation.org/packages/janitor/versions/1.2.0
[11] https://www.tidyverse.org/packages/

```
#arrange records/rows
arrangeRows <- function(.dataset, ...){
      arrange(.dataset, ...) -> dataset
}

#select dataset columns for analysis
selectColumns <- function(dataset, …){
      dataset %>%
            select(…) -> dataset
}

#delete columns from the dataset
deleteColumns <- function(dataset, …){
      dataset %>%
            select(-(…))-> dataset
}

#filter rows
filter(.dataset, …) -> dataset

# remove rows with low frequency
# num represents the frequency number
# n represents the count produced by tally
removeEventsLowFrequency <- function(.dataset, event, num){
eventCount <- .dataset %>%
      group_by(event) %>%
      tally

frequentEvents <- eventCount %>%
      filter(n > num) %>%
      select(event)

.dataset <- .dataset %>%
      filter(event %in% frequentEvents$event)
}


# delete traces with number of events less than a specific number
(num)
deleteTraceLengthLessThan <- function(.dataset, groupID, num){
traceCounts <- .dataset %>%
      group_by(groupID) %>%
      tally

frequentTraces <- traceCounts %>%
      filter(n >= num) %>%
      select(groupID)

.dataset <- .dataset %>%
      filter(groupID %in% frequentTraces$groupID)
}
```

```
# delete traces that do not start with a specific start event
# enquo quotes the value of startEvent and captures the environment
# where the deleteTruncatedTracesStart() function is called
deleteTruncatedTracesStart <- function(dataset, groupID, startEvent,
value) {
      groupID <- enquo(groupID)
      startEvent <- enquo(startEvent)

      dataset %>%
            group_by(!!groupID) %>%
            filter(first(!!startEvent) == value) -> dataset
}

# delete traces that do not finish with a specific end event
deleteTruncatedTracesEnd <- function(dataset, groupID, lastEvent,
value){
      groupID <- enquo(groupID)
      lastEvent <- enquo(lastEvent)

      dataset %>%
            group_by(!!groupID) %>%
            filter(last(!!lastEvent) == value) -> dataset
}

# delete traces with total duration less than t
deleteTracesWithTimeLess(dataset, groupID, time, t){
      groupID <- enquo(groupID)
      time <- enquo(time)

      dataset %>%
      group_by(!!groupID) %>%
            filter((last(time)- first(time))< t)
}

# concatenate two columns
concatenateColumns <- function(dataset, newCol, col1, col2){
      newCol <- enquo(newCol)
      col1 <- enquo(col1)
      col2 <- enquo(col2)
      dataset %>%
            mutate(!!newCol := paste(!!col1, !!col2)) ->
            dataset
}

# create isRepeatedEvent column
eventIsRepeated <- function(dataset, groupID, event,
      isRepeated) {
            groupID <- enquo(groupID)
            event <- enquo(event)
            isRepeated <- enquo(isRepeated)
```

```
      dataset %>%
            group_by(!!groupID) %>%
                  mutate(!!!repeated := as.numeric(!!event ==
                  lag(!!event, 1)))
}


# keep first event
keepFirstEvent <- function(.dataset, groupID, eventName, value){
      groupID <- enquo(groupID)
      eventName <- enquo(eventName)
      .dataset %>%
            filter(eventName == value & isRepeated != "1")
            -> .dataset
}


# keep last event
keepLastEvent <- function(.dataset, groupID, eventName, value){
      groupID <- enquo(groupID)
      eventName <- enquo(eventName)
      .dataset %>%
            filter(eventName == value & last(isRepeated))
            -> .dataset
}

# delete all events
deleteAllEvents <- function(.dataset, event, eventName){
      filter(.dataset, event != eventName) -> .dataset
}


# merge rows
mergeRows <- function(.dataset, .variables, …){
      ddply(.dataset, .variables, summarize, …)
}
```

A.2. Sample Usage Script

This script was used in section 6.4.

```
#import libraries

library(dplyr)
library(plyr)
library(janitor)
library(tidyverse)

## read CSV file
readCSV("dataset.csv") -> EventLogs
```

```
## clean column headers
cleanHeaders(EventLogs) -> EventLogs

## select columns for 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


## filter rows where company_id is empty
EventLogs %>% filter(company_id != "") -> EventLogs

## filter rows to remove some events
EventLogs %>% filter(client != "Phone App"
                  & event != "View Dashboard")-> EventLogs

## filter rows for events with low frequency

removeEventsLowFrequency(EventLogs, event, 10)
      -> EventLogs

## delete cases with few events
deleteTraceLengthLessThan(EventLogs, company_id, 5)
      -> EventLogs

## arrange rows according to variables company_id and time
arrangeRows(EventLogs, company_id, time) -> EventLogs

## delete cases that do not start with “Trial Sign Up Completed”
event
deleteTruncatedTracesStart(EventLogs, company_id, event, "Trial Sign
Up Completed") -> EventLogs

## delete cases that do not finish with “Purchase Confirm Details”
deleteTruncatedTracesEnd(EventLogs, company_id, event, "Purchase
Confirm Details") -> EventLogs


## concatenate two columns (optional)
concatenateColumns(EventLogs, act, event, source)
      -> EventLogs

## create isRepeatedEvent column
eventIsRepeated(EventLogs, company_id, event, isRepeated)
      -> EventLogs
```

```
## keep first occurrence of certain event
keepFirstEvent(EventLogs, company_id, event, "Edit Dashboard")
      -> EventLogs

## keep first occurrence of all events
EventLogs %>%
      filter(isRepeated != "1") -> EventLogs

## keep last occurrence of certain event
keepLastEvent(EventLogs, company_id, event, "Edit Data Source")
      → EventLogs

## delete all events
deleteAllEvents(EventLogs, event, "Slack Share")
      -> EventLogs

## merge rows
mergeRows(EventLogs,
           .(company_id, event),
           summarise,
           time = first(time),
           kpi_count = median(kpi_count),
           graphs_owned = median(graphs_owned),
           guide_name = paste(unique(guide_name),
           collapse=','),
           guide_type = paste(unique(guide_type),
           collapse=','),
           number_of_graphs = max(number_of_graphs),
           number_of_graphs_on_dashboard = sum(num-
           ber_of_graphs_on_dashboard),
           connector_backend= paste(unique(connector_backend), col-
           lapse = ','),
           x_city = paste(unique(x_city), collapse = ','),
           weekday = paste(unique(weekday), collapse = ',')) ->
EventLogs

## write to CSV file
writeCSV(EventLogs, "dataSet.csv")
```

A.3. Sample Usage Script on the New Dataset

This script was applied in section 6.6.

```
#import libraries

library(dplyr)
library(plyr)
library(janitor)
library(tidyverse)
```

```
## read CSV file
readCSV("dataset.csv")-> EventLogs

## clean column headers
cleanHeaders(EventLogs) -> EventLogs

## select columns for analysis
      selectColumns(EventLogs, company_id ,event, time, ac-
      count_type, account_status, number_of_clients,
      client_status) -> EventLogs


## filter rows where company_id is empty
EventLogs %>% filter(company_id != "") -> EventLogs

## filter rows to remove some events
EventLogs %>% filter(client != "Phone App"
                    & event != "View Dashboard")
                       -> EventLogs

## filter rows for events with low frequency

removeEventsLowFrequency(EventLogs, event, 10)
      -> EventLogs

## delete cases with few events
deleteTraceLengthLessThan(EventLogs, company_id, 2)
      -> EventLogs

## arrange rows according to variables company_id and time
arrangeRows(EventLogs, company_id, time) -> EventLogs

## delete cases that do not start with “Trial Sign Up Completed”
event
deleteTruncatedTracesStart(EventLogs, company_id, event, "Trial Sign
Up Completed") -> EventLogs


## create isRepeatedEvent column
eventIsRepeated(EventLogs, company_id, event, isRepeated)
      -> EventLogs

## keep first occurrence of certain event
keepFirstEvent(EventLogs, company_id, event, "Edit Dashboard")
      -> EventLogs

## keep first occurrence of all events
EventLogs %>%
      filter(isRepeated != “1”) -> EventLogs

## keep last occurrence of certain event
keepLastEvent(EventLogs,  company_id,  event,  "Edit  Data  Source")->
EventLogs
```

```
## delete all events
deleteAllEvents(EventLogs, event, "Slack Share")
      -> EventLogs

## merge rows
mergeRows(EventLogs,
           .(company_id, event),
           summarise,
           time = first(time),
           account_type = paste(unique(account_type),
           collapse=','),
           account_status = paste(unique(account_status),
           collapse=','),
           number_of_clients = sum(number_of_clients),
           client_status = paste(unique(client_status),
           collapse=','))
-> EventLogs


## write to CSV file
writeCSV(EventLogs, "dataset.csv.csv")
```