Process Mining for Python (**PM4Py**): Bridging the Gap Between Process- and Data Science

Alessandro Berti*, Sebastiaan J. van Zelst*†, Wil M.P. van der Aalst*†
*RWTH Aachen University
Process and Data Science group, Lehrstuhl für Informatik 9, 52074 Aachen, Germany
{a.berti,s.j.v.zelst,wvdaalst}@pads.rwth-aachen.de
†Fraunhofer Gesellschaft
Institute for Applied Information Technology (FIT), Sankt Augustin, Germany
{sebastiaan.van.zelst,wil.van.der.aalst}@fit.fraunhofer.de

***Abstract*—Process mining, i.e., a sub-field of data science focusing on the analysis of event data generated during the execution of (business) processes, has seen a tremendous change over the past two decades. Starting off in the early 2000's, with limited to no tool support, nowadays, several software tools, i.e., both open-source, e.g., ProM and Apromore, and commercial, e.g., Disco, Celonis, ProcessGold, etc., exist. The commercial process mining tools provide limited support for implementing custom algorithms. Moreover, both commercial and open-source process mining tools are often only accessible through a graphical user interface, which hampers their usage in large-scale experimental settings. Initiatives such as RapidProM provide process mining support in the scientific workflow-based data science suite RapidMiner. However, these offer limited to no support for algorithmic customization. In the light of the aforementioned, in this paper, we present a novel process mining library, i.e., Process Mining for Python (PM4Py), that aims to bridge this gap, providing integration with state-of-the-art data science libraries, e.g., pandas, numpy, scipy and scikit-learn. We provide a global overview of the architecture and functionality of PM4Py, accompanied by some representative examples of its usage.**

***Index Terms*—Process Mining; Data Science; Python.**

I. Introduction

The field of *process mining* [1] provides tools and techniques to increase the overall knowledge of a (business) process, by means of analyzing the event data stored during the execution of the process. Process mining received a lot of attention from both academia and industry, which led to the development of several commercial and open-source process mining tools. The majority of these tools supports *process discovery*, i.e., discovering a process model that accurately describes the process under study, as captured within the analyzed event data. However, process mining also comprises *conformance checking*, i.e., checking to what degree a given process model is accurately describing event data, and *process enhancement*, i.e., techniques that enhance process models by projecting interesting information, e.g. case flow and/or performance measures, on top of a model. The support of such types of process mining analysis is typically limited to open source, academic process mining tools such as the ProM Framework [2] and Apromore [3].

Both ProM and Apromore put a significant emphasis on non-expert usability, i.e., by means of providing an easy to use graphical user interface. Whereas such an interface helps to engage non-expert users and, furthermore, helps to showcase process mining to a larger audience, it hampers the usability of the tools for the purpose of large-scale scientific experimentation [4]. To this end, the RapidProM [5], [6] initiative allows for repeated execution of large-scale experiments with process mining algorithms in the RapidMiner[1] suite. However, RapidProM provides neither easy algorithmic customization nor an easy way to integrate custom developed algorithms. As such, the aforementioned tools fail to support customizable process mining algorithms and large-scale experimentation and analysis.

To bridge the aforementioned gap, i.e., the lack of process mining software that i) is easily extendable, ii) allows for algorithmic customization and iii) allows us to easily conduct large scale experiments, we propose the *Process Mining for Python (PM4Py)* framework. To achieve the aforementioned goals, a fresh look on the currently available programming languages and libraries indicates that the Python programming language[2], along with its ecosystem, is most suitable. In particular, the data science world, both for classic data science (pandas, numpy, scipy ...) and for cutting-edge machine learning research (tensorflow, keras ...), is heavily using Python. Other libraries, albeit with a lower number of features, exist already for the Python language (PMLAB [7], OpyenXES [8]). The bupaR library [9] supports process mining in the statistical language R, that is widely used in data science. The main focal points of the novel PM4Py library are:

- Lowering the barrier for algorithmic development and customization when performing a process mining analysis compared to existing academic tools such as ProM [2], RapidProM [5] and Apromore [3].
- Allow for easy integration of process mining algorithms with algorithms from other data science fields, implemented in various state-of-the-art Python packages.

[1] http://rapidminer.com
[2] http://python.org

```
from pm4py.algo.discovery.alpha import versions
from pm4py.objects.conversion.log import factory as log_conversion
ALPHA_VERSION_CLASSIC = 'classic'
ALPHA_VERSION_PLUS = 'plus'
VERSIONS = {ALPHA_VERSION_CLASSIC: versions.classic.apply,
ALPHA_VERSION_PLUS: versions.plus.apply}
def apply(log, parameters=None, variant=ALPHA_VERSION_CLASSIC):
    return VERSIONS[variant](log_conversion.apply(log, parameters, log_conversion.TO_EVENT_LOG), parameters)
```

Figure 1: Example factory method (Alpha Miner). Different variants (the Alpha and the Alpha+) are made available.

- Create a collaborative eco-system that easily allows researchers and practitioners to share valuable code and results with the process mining community.
- Provide accurate user-support by means of a rich body of documentation on the process mining techniques made available in the library.
- Algorithmic stability by means of rigorous testing.

The remainder of this paper is structured as follows. In Section II, we present the architecture and an overview of the features provided by PM4Py. In Section III, we present some representative examples (process discovery, conformance checking). Section IV discusses the maturity of the tool and Section V concludes this paper.

II. Architecture and Features

In order to maximize the possibility to understand and re-use the code, and to be able to execute large-scale experiments, the following architectural guidelines have been adopted on the development of PM4Py:

- A strict separation between *objects* (event logs, Petri nets, DFGs, ...), *algorithms* (Alpha Miner [10], Inductive Miner [11], alignments [12] ...) and *visualizations* in different packages. In the *pm4py.object* package, classes to import/export and to store the information related to the objects are provided, along with some utilities to convert objects, e.g., process trees into Petri nets; while in the *pm4py.algo* package, algorithms to discover, perform conformance checking, enhancement and evaluation are provided. All visualizations of objects are provided in the *pm4py.visualization* package.
- Most functionality in PM4Py has been realized through *factory methods*. These factory methods provide a single access point for each algorithm, with a standardized set of input objects, e.g., event data and a parameters object. Consider the factory method of the Alpha Miner, depicted in Fig. 1. The Alpha (`variant='classic'`) and the Alpha+ (`variant='plus'`) are made available. Factory methods allow for the extension of existing algorithms whilst ensuring backward-compatibility. The factory methods typically accept the name of the variant of the algorithm to use, and some parameters (shared among variants, or variant-specific).

In the remainder of this section, we present the main features of the library, organized in objects, algorithms, and visualizations.

A. Object Management

Within process mining, the main source of data are *event data*, often referred to as an *event log*. Such an event log, represents a collection of events, describing what activities have been performed for different instances of the process under study. PM4Py provides support for different types of event data structures:

- *Event logs*, i.e., representing a list of *traces*. Each trace, in turn, is a list of events. The events are structured as key-value maps.
- *Event Streams* representing one list of events (again represented as key-value maps) that are not (yet) organized in cases.

Conversion utilities are provided to convert event data objects from one format to the other. Furthermore, PM4Py supports the use of `pandas data frames`, which are efficient in case of using larger event data. Other objects currently supported by PM4Py include: heuristic nets, accepting Petri nets, process trees and transition systems.

B. Algorithms

The PM4Py library provides several mainstream process mining techniques, including:

- *Process discovery*: Alpha(+) Miner [10] and Inductive Miner (IMDF [11]).
- *Conformance Checking*: Token-based replay and alignments [12].
- Measurement of fitness, precision, generalization and simplicity of process models.
- Filtering based on time-frame, case performance, trace endpoints, trace variants, attributes, and paths.
- Case management: statistics on variants and cases.
- Graphs: case duration, events per time, distribution of a numeric attribute's values.
- Social Network Analysis [13]: handover of work, working together, subcontracting and similar activities networks.

```
1 from pm4py.objects.log.importer.xes import factory as xes_importer
2 from pm4py.algo.discovery.alpha import factory as alpha_miner
3 from pm4py.visualization.petrinet import factory as pn_vis_factory
4 log = xes_importer.apply("C:\\receipt.xes")
5 # discovers a Petri net along with an initial (im)
6 # and a final marking (fm)
7 net, im, fm = alpha_miner.apply(log)
8 gviz = pn_vis_factory.apply(net, im, fm)
9 pn_vis_factory.view(gviz)
```

Figure 2: PM4Py code to load a log, apply Alpha Miner and visualize a Petri net.

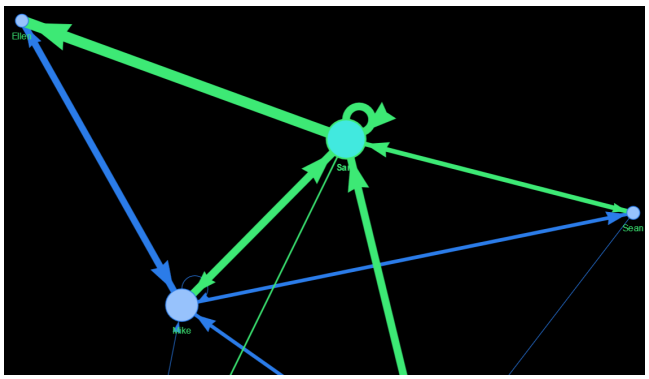

Figure 4: Social Network Analysis (Handover of Work metric) using Pyvis visualization.

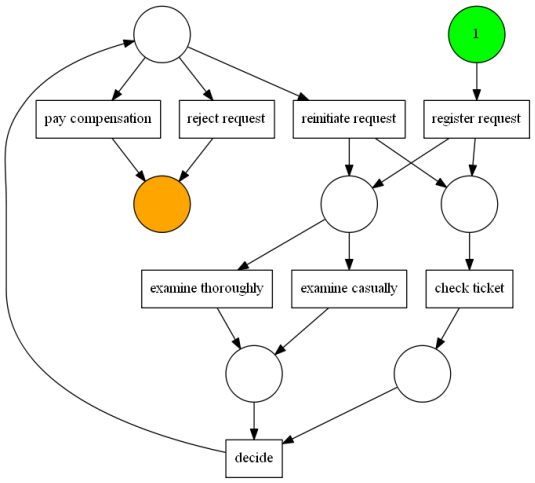

Figure 5: PM4Py in action: process discovery with the Alpha Miner.

C. Visualizations

The following Python visualization libraries have been used in the project:

- GraphViz: representation of directly-follows graphs, Petri nets, transition systems, process trees.
- NetworkX: static representation of social networks.
- Pyvis: web-based, dynamic representation of social networks (see Fig. 4).

III. Examples

In this section, we provide some examples of the use of PM4Py.

A. Process Discovery

Fig. 2 shows example code to perform process discovery using Alpha Miner and visualize the process model. The factory methods that are needed (XES importer, Alpha Miner and Petri net visualization) are loaded (line 1-3). Then, an XES log is imported (line 4), the Alpha Miner is applied providing the log object (line 7), and the visualization is obtained: a factory method is applied to layout the graph (line 8), and the result is shown in a window (line 9). The result is shown in Fig. 5.

```
1 from pm4py.algo.conformance.alignments import factory as alignments
2 # alignments accepts a log and an accepting Petri net, i.e.
3 # a Petri net along with an initial (im) and a final (fm) marking
4 aligned_traces = alignments.apply(log, net, im, fm)
5 for index, result in enumerate(aligned_traces):
6     print(index, result['alignment'])
```

```
[('register request', 'register request'), ('>>', None), ('check ticket', 'check ticket'),
('examine thoroughly', 'examine thoroughly'), ('>>', None), ('decide', 'decide'), ('>>', None),
('reject request', 'reject request')]
```

Figure 3: PM4Py code to perform alignments between a log and a model, and print the alignments. The output of the alignment of a trace on an example log and model is reported.

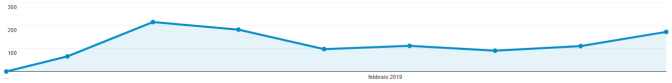

Figure 6: Users that accessed the PM4Py website in February 2019

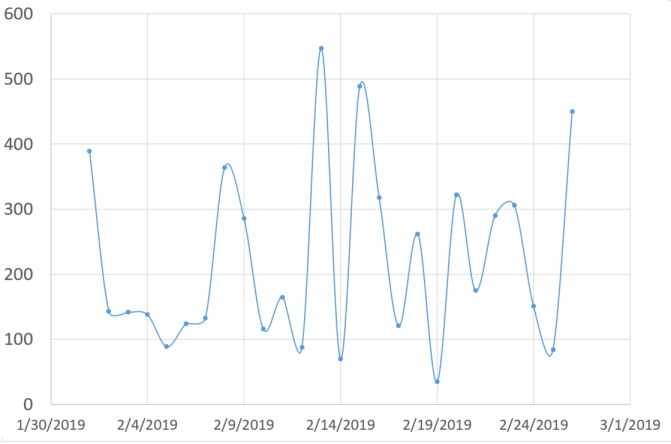

Figure 7: Daily downloads of PM4Py from Pypi during the month of February 2019

B. Conformance Checking

Fig. 3 shows example code to apply alignments and display the result. First, the alignments factory method is loaded (line 1). Then, the alignments between a log object and a process model are obtained (line 4). For each aligned trace (line 5) the alignment result is displayed on the screen (line 6). The alignment of a trace is reported in the lower part of Fig. 3.

IV. Maturity of the tool

PM4Py 1.0 has been released on 21/12/2018 and was used by 200 students in the "Introduction to Data Science" course held by the Process and Data Science group in the RWTH Aachen University. Already two academic projects have been supported by PM4Py and are publicly available:

- Usage of probabilistic automata for compliance checking (*https://github.com/lvzheqi/StreamingEventCompliance*).
- Prefix alignments for streaming event data [14] (*https://gitlab.com/prefal/confo*).

PM4Py 1.1 has been released on 22/02/2019 with additional features. There are some integrations of the PM4Py library in other projects:

- bupaR R process mining library uses PM4Py to handle alignments and get models using the Inductive Miner.
- A data analytics web interface was written in Vue.JS (*https://git.bogdan.co/b0gdan/beratungsleistungen*).

In Fig. 6, some statistics taken from Google Analytics are reported about the number of accesses to PM4Py web site during the month of February 2019. In Fig. 7, some statistics about the downloads of the PM4Py library from PIP are reported. Issues are managed through Github. The XES certification, with maximum score, has been awarded to the PM4Py library.

V. Conclusion

In this paper, the PM4Py process mining library (*http://www.pm4py.org*) has been introduced. PM4Py supports a rapidly growing set of process mining techniques (discovery, conformance checking, enhancement ...). A video presenting the library and some example applications (log management, process discovery, conformance checking) has been made available[3]. The library can be installed[4] through the command *pip install pm4py*. Extensive documentation is provided through the official website of the library. Moreover, the Github repository supports a collaborative eco-system where users could signal problems or contribute to the code.

References

[1] W. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[2] B. F. Van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst, "The prom framework: A new era in process mining tool support," in *International conference on application and theory of petri nets*. Springer, 2005, pp. 444–454.

[3] M. La Rosa, H. A. Reijers, W. van der Aalst, R. M. Dijkman, J. Mendling, M. Dumas, and L. García-Bañuelos, "Apromore: An advanced process model repository," *Expert Systems with Applications*, vol. 38, no. 6, pp. 7029–7040, 2011.

[4] A. Bolt, M. de Leoni, and W. M. van der Aalst, "Scientific workflows for process mining: building blocks, scenarios, and implementation," *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 6, pp. 607–628, 2016.

[5] R. Mans, W. van der Aalst, and H. E. Verbeek, "Supporting process mining workflows with RapidProM." in *BPM (Demos)*, 2014, p. 56.

[6] W. van der Aalst, A. Bolt, and S. J. van Zelst, "RapidProM: Mine your processes and not just your data," *CoRR*, vol. abs/1703.03740, 2017. [Online]. Available: http://arxiv.org/abs/1703.03740

[7] J. Carmona Vargas and M. Solé, "Pmlab: a scripting environment for process mining," in *Proceedings of the BPM Demo Sessions 2014: Co-located with the 12th International Conference on Business Process Management (BPM 2014) Eindhoven, The Netherlands, September 10, 2014*. CEUR-WS. org, 2014, pp. 16–20.

[8] H. Valdivieso, W. L. J. Lee, J. Munoz-Gama, and M. Sepúlveda, "Opyenxes: A complete python library for the extensible event stream standard."

[9] G. Janssenswillen and B. Depaire, "Bupar: business process analysis in r," 2017.

[10] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[11] S. J. Leemans, D. Fahland, and W. van der Aalst, "Scalable process discovery with guarantees," in *International Conference on Enterprise, Business-Process and Information Systems Modeling*. Springer, 2015, pp. 85–101.

[12] A. Adriansyah, N. Sidorova, and B. F. van Dongen, "Cost-based fitness in conformance checking," in *2011 Eleventh International Conference on Application of Concurrency to System Design*. IEEE, 2011, pp. 57–66.

[13] W. van der Aalst and M. Song, "Mining social networks: Uncovering interaction patterns in business processes," in *International conference on business process management*. Springer, 2004, pp. 244–260.

[14] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, and W. van der Aalst, "Online conformance checking: relating event streams to process models using prefix-alignments," *International Journal of Data Science and Analytics*, pp. 1–16, 2017.

[3] http://pm4py.pads.rwth-aachen.de/pm4py-demo-video/

[4] Additional prerequisites, available at the page http://pm4py.pads.rwth-aachen.de/installation/ have to be installed.