Road Traffic Fine Management Process

Arvin Jafari Moghadam Fard

Business Information Systems

Prof. Paolo Ceravolo

a.a. 2023 - 2024

I. Introduction

This project examines an event log from Italy's road-traffic fines management system. The objective is to uncover insights that improve fine payment completion rates, reduce management costs, and detect process issues early. Through dataset exploration, noise filtering, process discovery, and summarization, this study aims to offer practical strategies for enhancing operational efficiency and achieving better process outcomes.

A. Goals

This project employs process mining techniques to optimize the management of road traffic fines. It begins by analyzing event logs from the police information system to understand statistical properties distinguishing paid and unpaid cases. The objectives include discovering the current process (AS-IS), deriving an optimal process (TO-BE), and analyzing discrepancies, particularly in unpaid cases.

Goals for the Road-Traffic Fines Case Study:

1. Increase Payment Completion Probability: Focus on filtering, summarizing, and analyzing data to improve payment rates promptly after notification.
2. Reduce Process Management Costs: Streamline fine management by filtering out irrelevant cases and analyzing cost-related data to minimize overhead.
3. Early Detection of Process Issues: Identify anomalies in process execution times and appeal outcomes to enhance process efficiency and address potential issues promptly.

B. Knowledge Uplift Trail

The Knowledge Uplift Trail (KUT) is a structured approach to transform raw data into valuable insights. It involves several key steps:

1. **Data Cleaning:** This step standardizes data formats, fills in missing values, and converts categorical data into numerical formats, ensuring the data is ready for analysis.
2. **Data Filtering:** Filters the dataset based on specific criteria, such as keeping only cases where the final activity was a payment, to focus on relevant information.
3. **Descriptive Analysis:** Utilizes statistical tools to analyze the distribution and characteristics of the data, providing a clear understanding of its properties.

4. **Process Mining:** Applies process mining techniques to uncover the underlying process from the event log data, revealing how activities are typically performed.
5. **Conformance Checking:** Identifies any deviations from the expected sequence of events within the process, highlighting potential inefficiencies or irregularities.
6. **Intervention Strategies:** Based on insights gained, strategies are developed to optimize processes. These strategies are continuously refined through ongoing monitoring and measurement to improve outcomes.

The KUT methodology ensures that data is systematically processed and analyzed to derive actionable knowledge, guiding informed decision-making and process improvements.

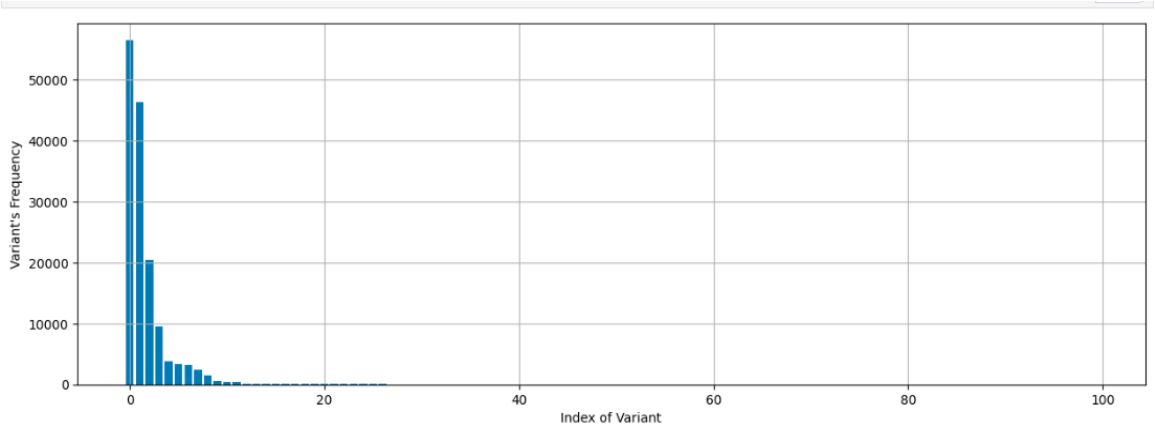
C. Dataset

The dataset used in this study focuses on road-traffic fines managed by the Italian police. documented in an event log to support process management containing over 500,000 logs. The dataset includes various attributes:

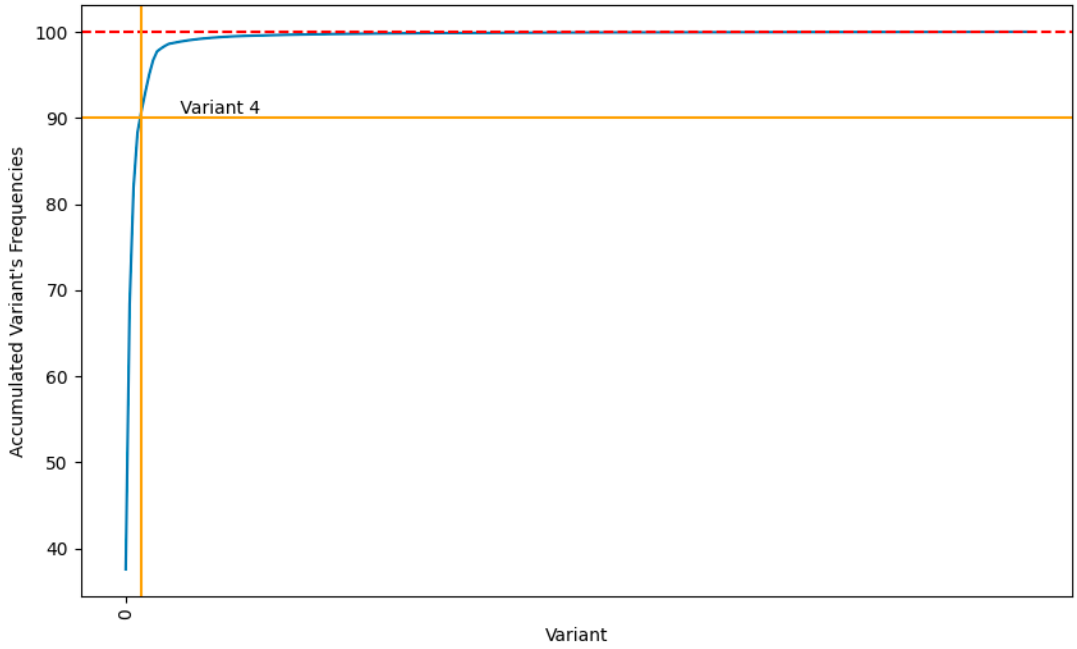
- **case:concept**: Unique identifier for each fine case.
- **concept**: Describes specific activities in the fine process, such as "Create Fine," "Send Fine," or "Payment."
- **time**: Records the exact date and time of each event.
- **dismissal**: Indicates the outcome of the fine, with "NIL" meaning the fine must be paid, and "#" or "G" indicating dismissal by a judge or prefecture.
- **vehicleClass**: Classifies the vehicle involved in the fine.
- **article**: Refers to the regulation under which the fine was issued.
- **points**: Records points deducted from the offender's driving license.
- **amount**: Monetary value associated with fine creation and penalty addition.
- **expense**: Additional costs linked to fine notifications.
- **paymentAmount**: Amount paid during a payment event.

II. Exploring Data

As mentioned in the Dataset section, our data contains 561470 log events and 150370 cases. By analyzing the and extracting variants frequency from our logs and sorting them we can reach the following graph.



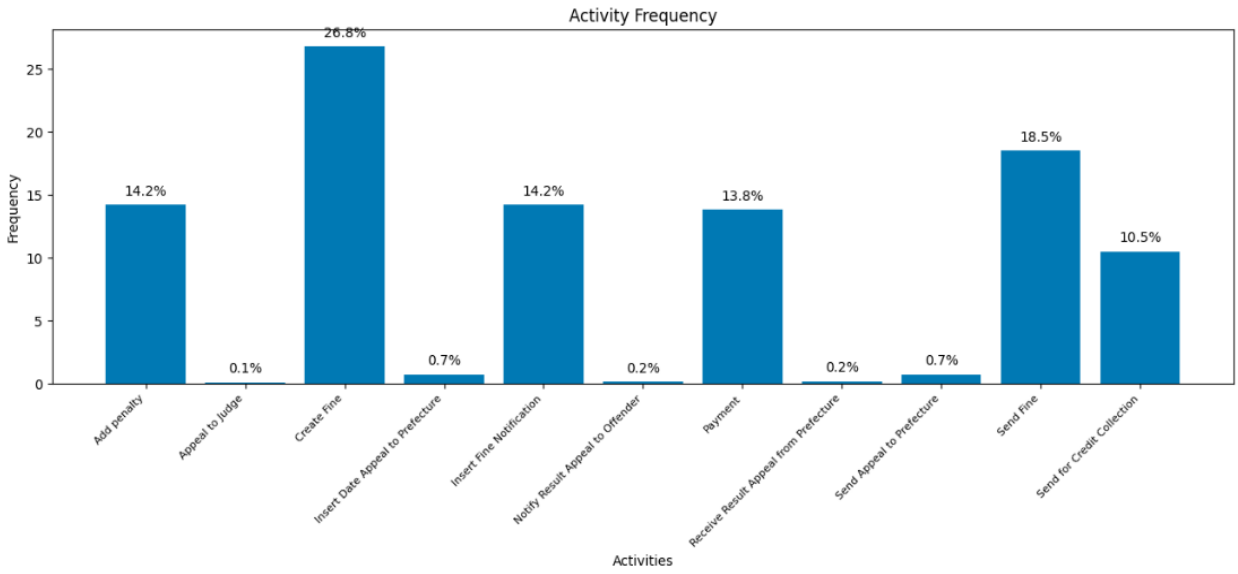
The noticeable thing about the graph is that, most of the information can be obtained by only the first top 10 variants, now for understanding better we can show the accumulative frequency percentage of the variants in the below graph.



So, it is shown in the graph that to obtain at least 90% of the information we need only top 4 of the frequent variants.

Now we can Check the Activity frequency percentages, with plotting the top 10 most frequent activities we will reach the below table and bar chart.

	concept:name	Frequency	Percentage
1	Create Fine	150370	26.781484
2	Send Fine	103987	18.520491
3	Insert Fine Notification	79860	14.223378
4	Add penalty	79860	14.223378
5	Payment	77601	13.821041
6	Send for Credit Collection	59013	10.510446
7	Insert Date Appeal to Prefecture	4188	0.745899
8	Send Appeal to Prefecture	4141	0.737528
9	Receive Result Appeal from Prefecture	999	0.177926
10	Notify Result Appeal to Offender	896	0.159581



For better understanding the distribution of the activities among our log dataset we plotted the word cloud chart of them that is shown in the below.

For checking and identifying the anomalies in our log data we analyzed the start and end activities.

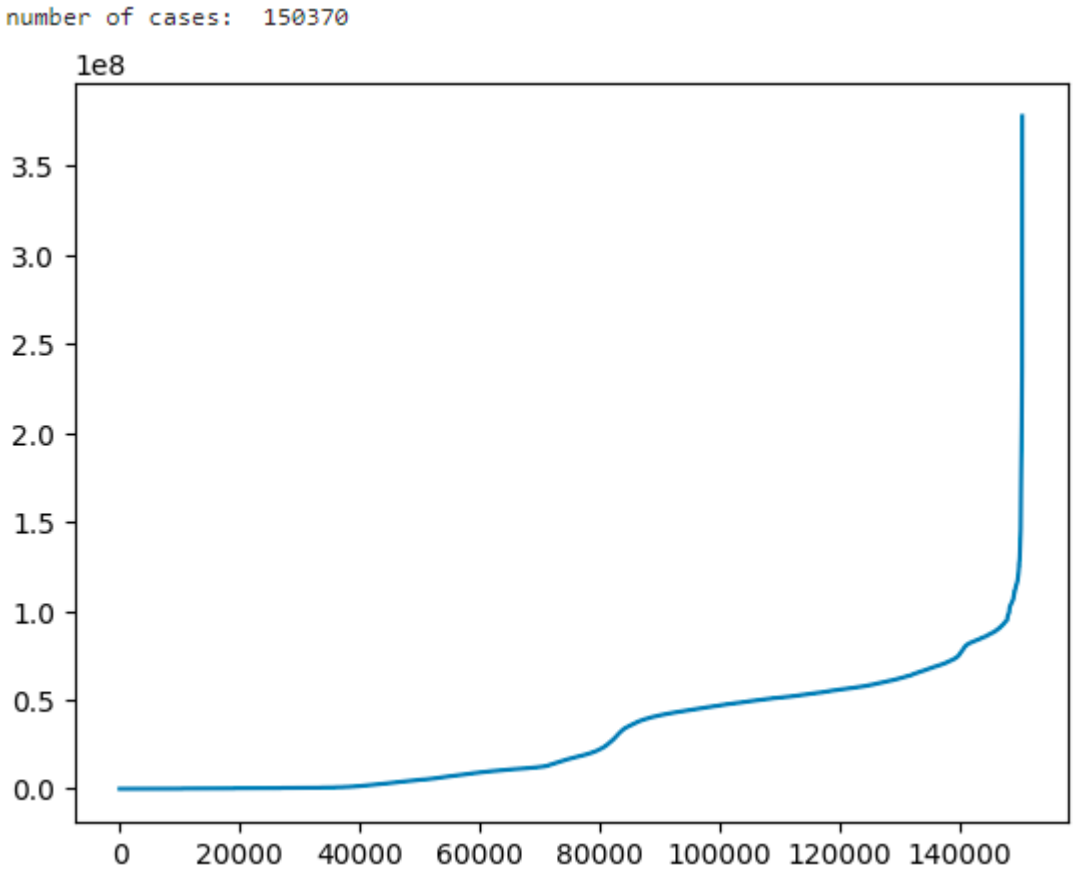
```
log_start
```

```
{'Create Fine': 150370}
```

```
log_end
```

```
{'Send Fine': 20755,
 'Send for Credit Collection': 58997,
 'Payment': 67201,
 'Send Appeal to Prefecture': 3144,
 'Appeal to Judge': 134,
 'Notify Result Appeal to Offender': 86,
 'Receive Result Appeal from Prefecture': 53}
```

With checking the duration of the cases we can have a good picture of noises and anomalies, in the below line chart the duration distribution of cases is been shown.



III. Applying Filters and Noise removal

First aspect of data that we are considering in order to filter is checking the completion duration cases. With the following stats

```
Number of events: 561470
Number of cases: 150370
Start activities: {'Create Fine': 150370}
End activities: {'Send Fine': 20755, 'Send for Credit Collection': 58997, 'Payment': 67201, 'Send Appeal to Prefecture': 3144, 'Appeal to Judge': 134, 'N
otify Result Appeal to Offender': 86, 'Receive Result Appeal from Prefecture': 53}
```

```
Min Case Duration: 0 days 00:00:00
Max Case Duration: 4372 days 00:00:00
Mean Case Duration: 341 days 16:06:01.029460664
```

After our zero duration filter and considering that not all of the zero duration cases our noises (for example cases with valid traces like {“Create Fine”, ”Payment”} are not noise) we will end up with the following noises that should be discarded.

```
{('Create Fine', 'Send Fine'): 118}
```

The next preprocessing phase is to standardize our dismissal column of dataset and make it more clear with the following function.

```
def format_dismiss(row):
    if row['dismissal'] in ['#', 'G']:
        return True
    elif row['dismissal'] == 'NIL':
        return False
    elif pd.isna(row['dismissal']):
        return False
    else:
        return "Unknown"
```

```
log_df['dismissal'].unique()
```

```
array([False, True, 'Unknown'], dtype=object)
```

It is obvious that cases with ‘unknown’ dismissal value can’t be very informative for us and we discarded them from our dataset.

To make processing event logs easier, we need a function that combines different financial values into one column. The function format amounts work like this:

```
def format_amounts (row):
    if row ['concept:name'] == 'Create Fine':
        return row['amount']
    elif row['concept:name'] == 'Send Fine':
        return row['expense']
    elif row['concept:name'] == 'Add penalty':
        return row['amount']
    elif row['concept:name'] == 'Payment':
        return row['paymentAmount']
    else:
        return 0

log_df["amount"] = log_df.apply (format_amounts ,axis =1)
```

Now with considering the activity of cases we want to keep only cases with legal end activities like 'Payment', 'Send for Credit Collection', 'Send Appeal to Prefecture', 'Appeal to Judge'. And the output will be:

```
Given 150370 total cases in the log we have 129476 cases that comply with the applied filter
```

It is considerable that cases end up with ‘Send appeal to prefecture’ or ‘Appeal to Judge’ can be considered true if values of their dismissal column be True.

Now after our filterings steps we can have a comparison of the case durations between the primary log dataset and filtered log dataset which is shown in the below.

```
TOTAL event log
Min Case Duration: 0 days 00:00:00
Max Case Duration: 4372 days 00:00:00
Mean Case Duration: 341 days 16:06:01.029460664
FILTERED event log
Min Case Duration: 0 days 00:00:00
Max Case Duration: 4372 days 00:00:00
Mean Case Duration: 381 days 01:59:18.849516512
```

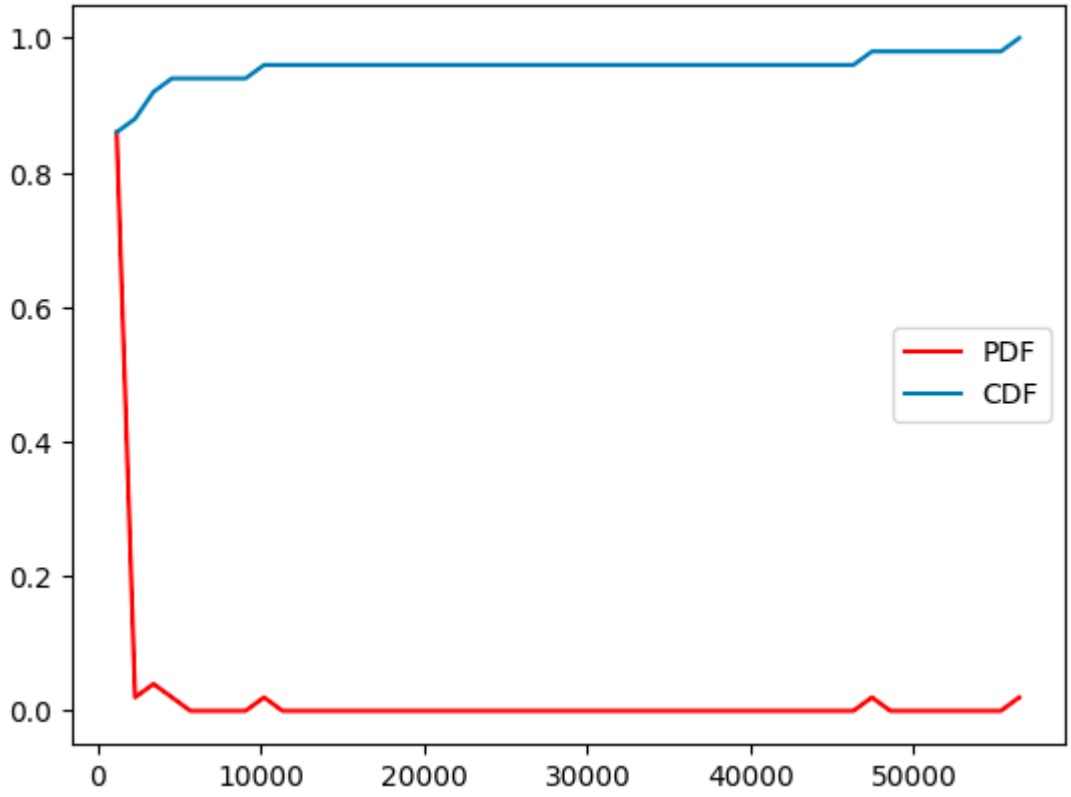
IV. Summarizing

For the Summarizing first we can take a look to the variants sorted by their frequency.

	Variant	Count
0	(Create Fine, Send Fine, Insert Fine Notification, Add penalty, Send for Credit Collection)	56482
1	(Create Fine, Payment)	46371
2	(Create Fine, Send Fine, Insert Fine Notification, Add penalty, Payment)	9520
3	(Create Fine, Send Fine, Insert Fine Notification, Add penalty, Payment, Payment)	3736
4	(Create Fine, Send Fine, Insert Fine Notification, Payment, Add penalty, Payment)	3301
...	...	...
310	(Create Fine, Send Fine, Insert Fine Notification, Add penalty, Appeal to Judge, Create Fine, Send Fine, Insert Fine Notification, Payment, Add penalty)	1
311	(Create Fine, Payment, Payment, Send Fine, Insert Fine Notification, Add penalty, Payment)	1
312	(Create Fine, Send Fine, Insert Fine Notification, Insert Date Appeal to Prefecture, Send Appeal to Prefecture, Receive Result Appeal from Prefecture, Add penalty, Notify Result Appeal to Offender, Appeal to Judge, Create Fine, Payment, Create Fine, Send Fine, Insert Fine Notification, Add penalty, Payment, Payment, Create Fine)	1
313	(Create Fine, Send Fine, Insert Fine Notification, Add penalty, Insert Date Appeal to Prefecture, Appeal to Judge, Send Appeal to Prefecture)	1
314	(Create Fine, Send Fine, Insert Date Appeal to Prefecture, Send Appeal to Prefecture, Insert Fine Notification, Appeal to Judge, Add penalty, Payment)	1

315 rows × 2 columns

For having a better sight of the variants distribution we have plotted the Probability Density Function (PDF) and Cumulative Distribution Function (CDF)



Both the CDF and PDF start with very high values, indicating a large number of events in the first few variants. After this, the PDF drops sharply. There are a few small spikes later, showing some variants with slightly higher event counts, but most have very low counts compared to the first ones.

With choosing top 10 frequent variants we will proceed with another step in filtering and we will end up with following stats.

```
Number of events: 491845
Number of cases: 125454
```

For the last step of filtering repetition of activities are checked. For example, it is not meaningful for a case, we have multiple delete invoice. With the following snippet of code, However, all of the samples passed this filter.

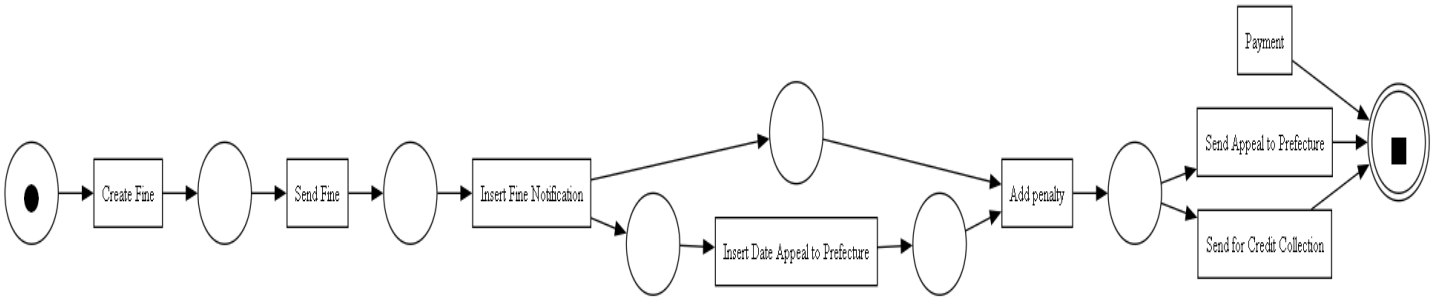
```
cases_to_exclude = pd.concat([
    pm4py.filter_activities_rework(filtered_log, "Send Fine", 2),
    pm4py.filter_activities_rework(filtered_log, "Send for Credit Collection'", 2)
    ])
cases_to_exclude = cases_to_exclude["case:concept:name"].unique()

print("cases to exclude:",len(cases_to_exclude))
print("total cases:", len(filtered_log["case:concept:name"].unique()))
filtered_log = filtered_log[~filtered_log['case:concept:name'].isin(cases_to_exclude)]
print("total cases:", len(filtered_log["case:concept:name"].unique()))
```

```
cases to exclude: 0
total cases: 125454
total cases: 125454
```

V. Process Discovery

In this section we try to apply different process discovery algorithm and do a comparison among them for four criteria of generalization, fitness, precision, and simplicity.

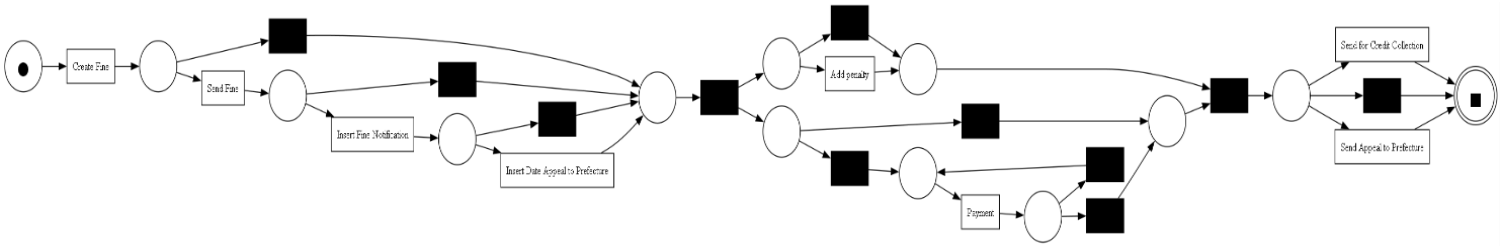
The first algorithm we use is Alpha Miner, which discovers process models from event logs by identifying causal dependencies between activities. It analyzes the order of events to infer relationships like precedence or concurrency. The algorithm uses the 'directly follows' relation to construct a process model, usually as a Petri-net or flowchart. Applying this algorithm to our filteredlog file generates the following Petri-net



Results of the Aplha Miner:

replay based fitness: {'perc_fit_traces': 0.6982639054952413, 'average_trace_fitness': 0.8354134213112436, 'log_fitness': 0.8309338322219397, 'percentage_of_fitting_traces': 0.6982639054952413}
replay based precision: 0.67744527826278
generalization: 0.9888759824824958
simplicity: 0.8888888888888888

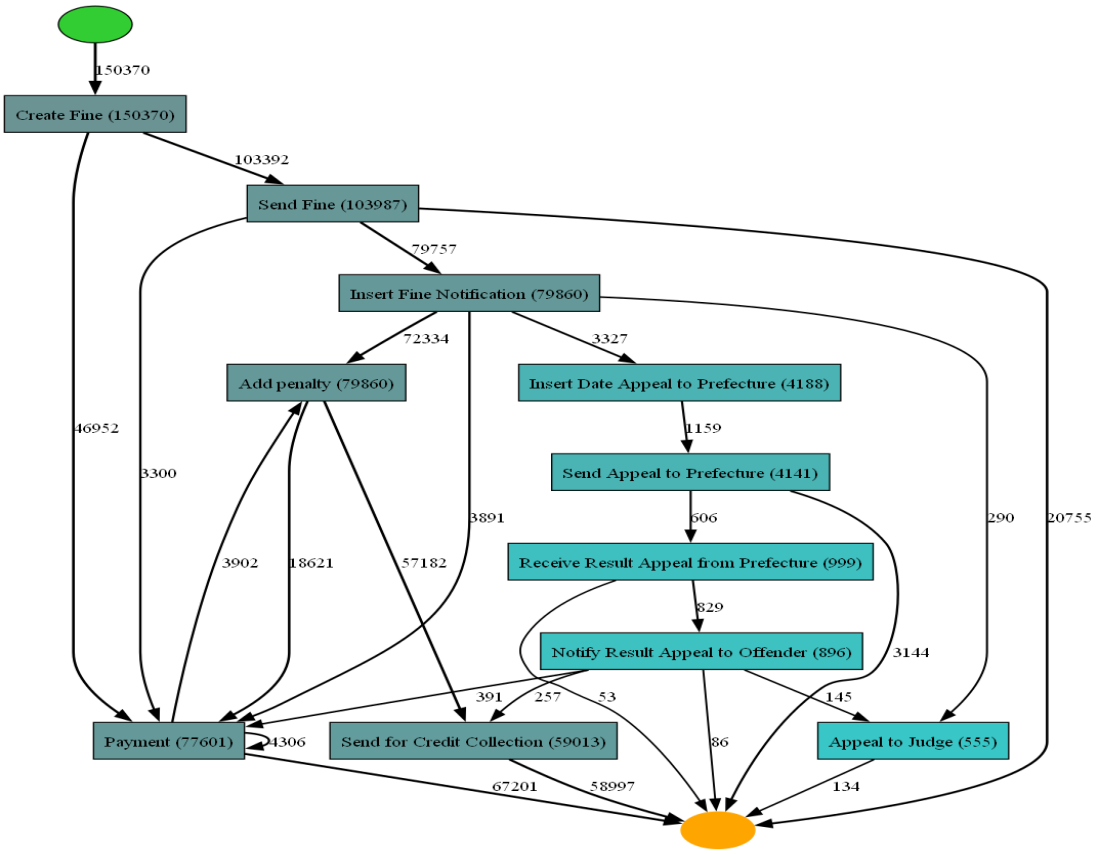
Next is the Inductive Miner algorithm, which discovers process models from event logs by inferring their structure, including choices and loops. Unlike Alpha Miner, it uses a bottom-up approach, starting with individual activities and merging similar behaviors. It handles complex behaviors like exclusive choices and parallelism. Applying this algorithm to our filtered log file produces the following Petri-net.

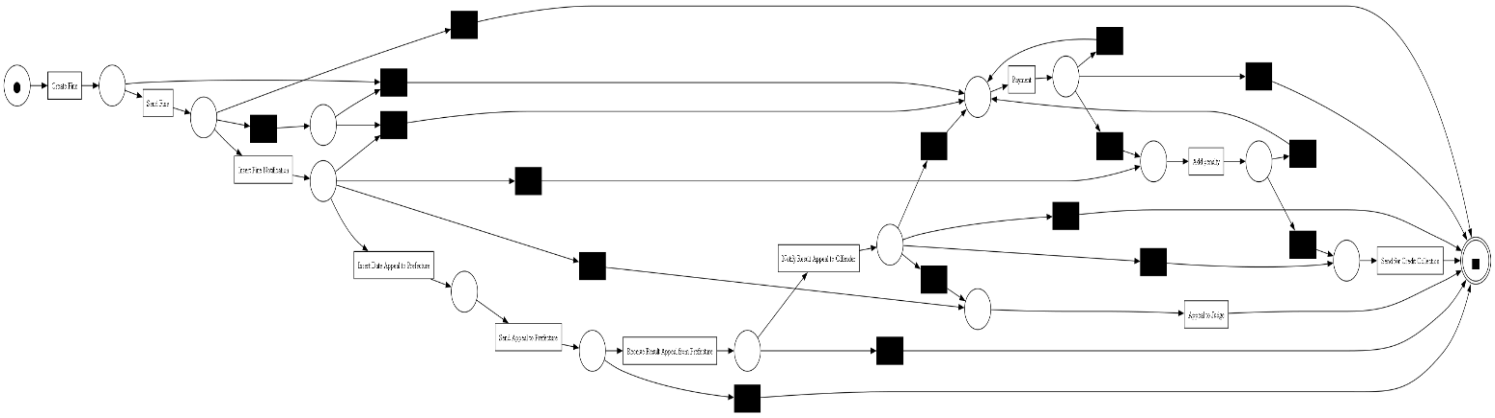


Results of the Inductive Miner:

replay based fitness: {'perc_fit_traces': 100.0, 'average_trace_fitness': 1.0, 'log_fitness': 1.0, 'percentage_of_fitting_traces': 100.0}
replay based precision: 0.6004576446831842
generalization: 0.9919500285557509
simplicity: 0.6666666666666666

The Heuristic Miner algorithm creates process models from event logs by finding patterns and relationships, considering event frequency, choices, and parallel activities. It handles noisy data and complex behaviors, using a heuristic-based approach. Below is the output of applying Heuristic Miner to discover Heuristic-net and the petri net of our Heuristic Miner:

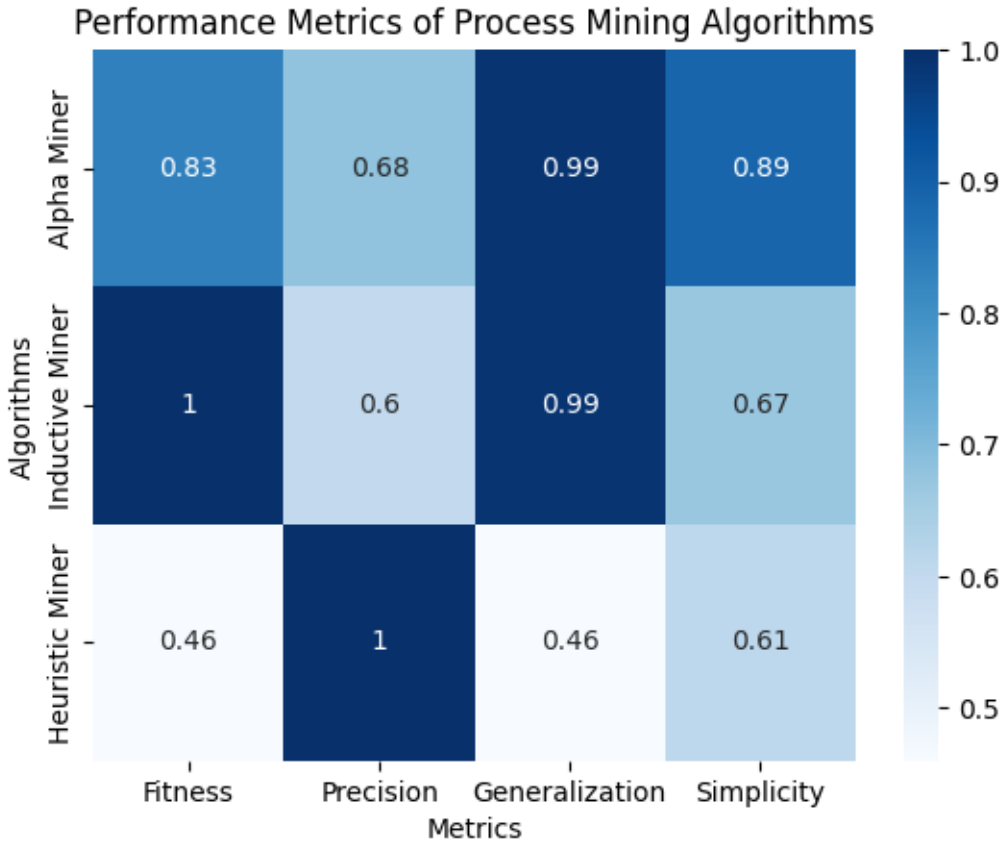




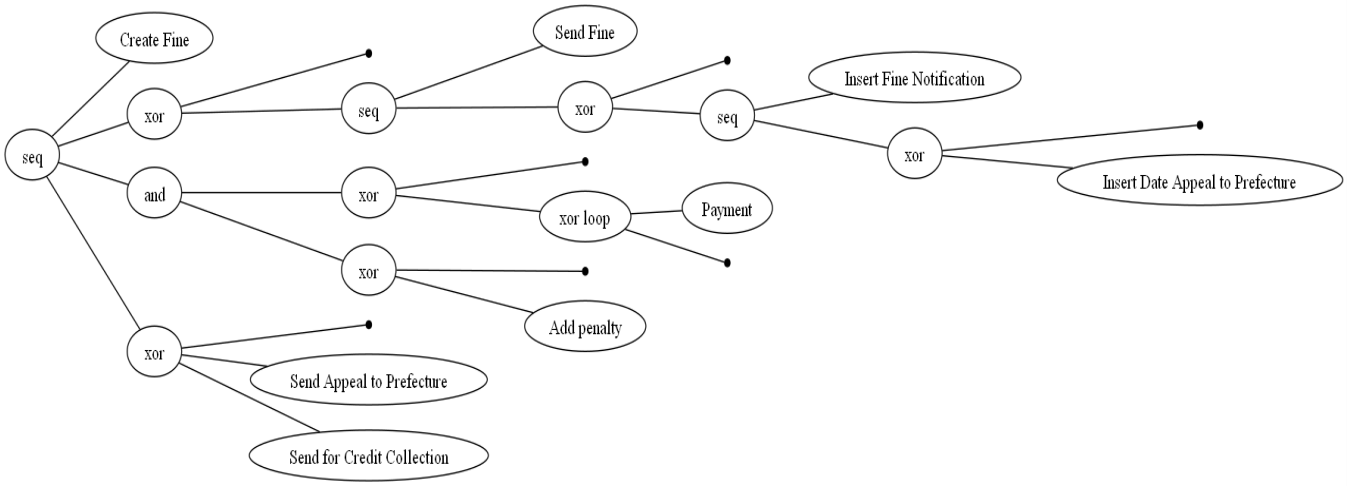
Results of the Heuristic miner:

replay based fitness: {'perc_fit_traces': 0.0, 'average_trace_fitness': 0.46074208949289197, 'log_fitness': 0.604007655769335, 'percentage_of_fitting_traces': 0.0}
replay based precision: 1.0
generalization: 0.45933925857280344
simplicity: 0.6111111111111112

To understand perfomances for each algoirthm we use heatmap to compare their results.



According to the heatmap Alpha Miner performed better In general than the others
A process tree visually represents a process model, capturing its control flow and behavior. It's hierarchical, showing relationships between activities, decisions, and loops. Process trees are intuitive, displaying activity order, choices, and looping. They help analyze process behavior, identify bottlenecks, and optimize processes. After using pm4py's process tree generator, the graph below is obtained:



The footprints matrix is a visual representation generated from event logs using process mining techniques. It illustrates the frequency and dependencies of activities in a process. Each cell in the matrix shows how often one activity directly follows another. This helps in identifying common patterns, sequence of activities, and potential bottlenecks in the process flow

	Add penalty	Create Fine	Insert Date Appeal to Prefecture	Insert Fine Notification	Payment	Send Appeal to Prefecture	Send Fine	Send for Credit Collection
Add penalty	#	#	<	<	‖	>	#	>
Create Fine	#	#	#	#	>	#	>	#
Insert Date Appeal to Prefecture	>	#	#	<	#	#	#	#
Insert Fine Notification	>	#	>	#	>	#	<	#
Payment	‖	<	#	<	‖	#	<	>
Send Appeal to Prefecture	<	#	#	#	#	#	#	#
Send Fine	#	<	#	>	>	#	#	#
Send for Credit Collection	<	#	#	#	<	#	#	#

VI. Segmentation

A. Decision Tree

A Decision Tree is a powerful and intuitive classification algorithm used in data analysis. It works by splitting the data into subsets based on feature values, creating a tree-like model of decisions. Each node represents a feature, and each branch represents a decision rule. This allows for easy interpretation and understanding of the decision-making process.

In analyzing the logs, a Decision Tree can effectively classify events and predict outcomes, such as payment completion. By training the model on historical data, it learns the patterns and relationships between features and the target variable. In this project, the Decision Tree achieved a high accuracy of 98%, demonstrating its effectiveness in predicting fine payments based on various attributes of the event logs. This insight can help optimize the management process and improve operational efficiency.

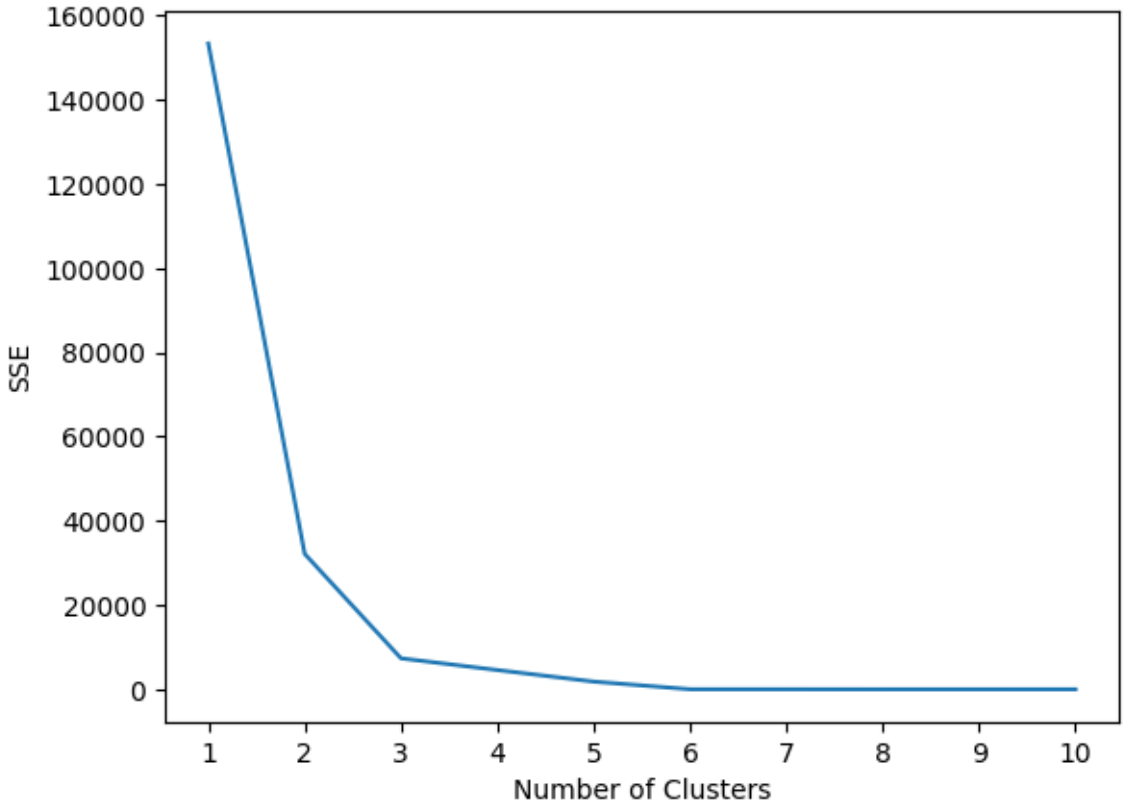
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9837389872160205
```

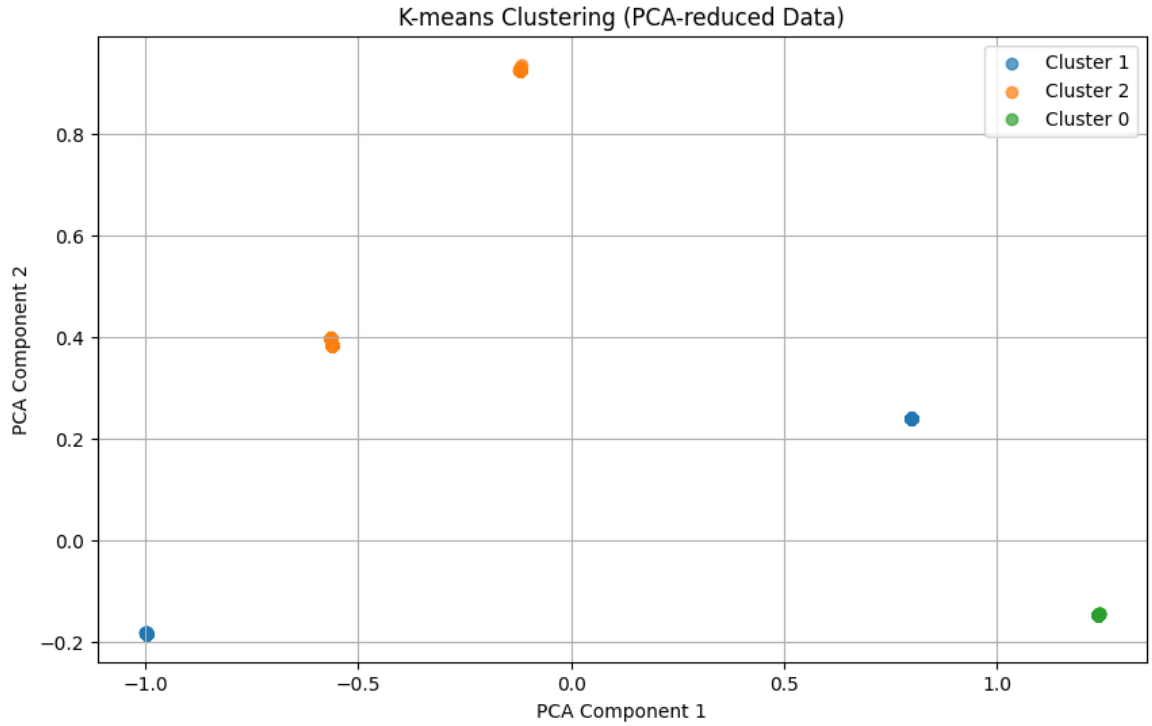
B. Clustering

Clustering is a technique used to group similar data points together based on their features. In this context, it segments event logs into clusters of traces that exhibit similar patterns. The K-means algorithm is a popular clustering method that partitions data into K clusters, where each data point belongs to the cluster with the nearest mean.

In the provided code, after normalizing the features of event logs, K-means is applied to identify clusters of similar traces. The 'elbow method' graphically determines the optimal number of clusters (K) by plotting the sum of squared errors (SSE) against different K values. The point where the SSE begins to level off ('elbow point') suggests an appropriate K value. Finally, each trace in the event log is assigned to a cluster, enabling further analysis and insights into process behavior.



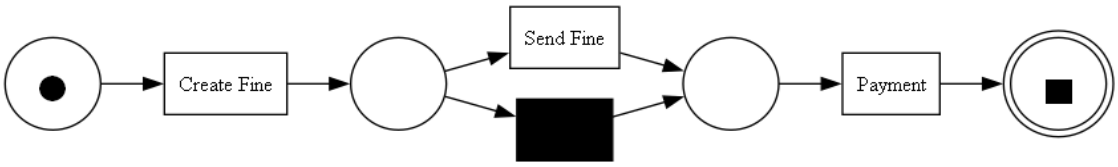
Choosing K=3K = 3K=3 based on the elbow method indicates that three clusters effectively capture the variation in the data, balancing model complexity and clustering quality as suggested by the elbow plot

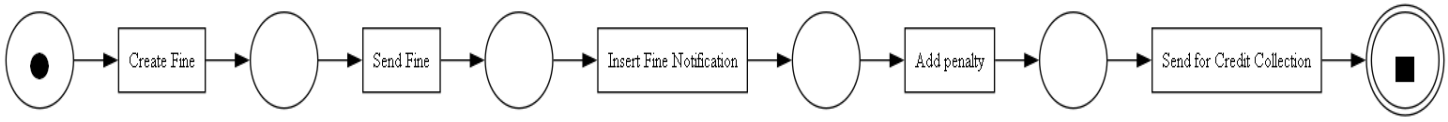


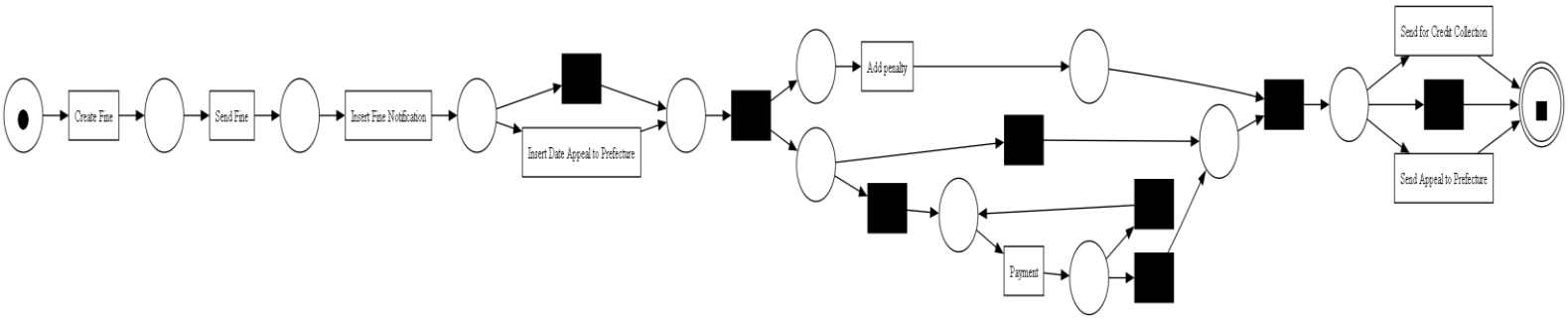
.

Segmenting cases based on their cluster:
Number of events in cluster 0: 92742
Number of cases in cluster 0: 46371
Number of events in cluster 1: 291803
Number of cases in cluster 1: 59613
Number of events in cluster 2: 107300
Number of cases in cluster 2: 19470

Petri-nets of the Inductive Miner algorithm for each cluster have been shown respectively:







Here's a summarized and short conclusion for your project based on the K-means clustering results with three clusters:

Cluster 0: This cluster includes 46,371 cases where fines are predominantly paid in full without significant penalties or credit collection. The average payment amount is 0.94% of the total fine amount.
Cluster 1: With 59,613 cases, this cluster shows fines where there's a mix of payments and penalties, indicating some cases involve partial payments or delays. The average payment amount is notably lower at 0.07% of the total fine amount.
Cluster 2: This smaller cluster comprises 19,470 cases where fines typically involve higher penalty rates and instances of credit collection. The average payment amount is 95.5% of the total fine amount, suggesting these cases often result in full payment after initial penalties.
These clusters provide insights into different patterns of fine management behaviors, guiding interventions for optimizing payment completion rates and reducing management costs in the road-traffic fines process.

VII. Conclusions

This project successfully analyzed Italy's road-traffic fines management process, revealing significant insights through process mining techniques. By segmenting cases into three distinct clusters, we identified patterns in fine payment behaviors. Cluster 0 primarily involves cases with full payments and minimal penalties, Cluster 1 shows a mix of payments and penalties, and Cluster 2 features higher penalties and credit collections but often results in full payments. These findings highlight opportunities to enhance payment completion rates, streamline management costs, and address process inefficiencies, ultimately contributing to a more efficient fines management system.

Link to Github repo