Road Traffic Fine Management Process

Cassinelli Lorenzo
Mat. 30454A
Master Degree in Computer Science
University of Milan
Business Information Systems
Prof. Paolo Ceravolo
A.Y. 2023/2024

Contents

1 Case Study Description **2**
- 1.1 Dataset . . . 2

2 Organizational Goals **3**

3 Knowledge Uplift Trail (KUT) **4**
- 3.1 Data Understading . . . 5
- 3.2 Data Cleaning and Data Filtering . . . 6
- 3.3 Descriptive Analysis . . . 9
- 3.4 Process Discovery . . . 11
- 3.5 Conformance Checking . . . 13
- 3.6 Machine Learning . . . 14

4 Project Result **15**
- 4.1 Process Discovery Result . . . 15
- 4.2 Conformance Checking Result . . . 15
- 4.3 Machine Learning Result . . . 16

5 Improvements **18**

6 Code **19**

1 Case Study Description

The Italian police developed a system to manage road traffic fines, recording data on over 140,000 cases. The event log from this system is used to analyze and improve the process by (i) increasing fine payment rates, (ii) reducing management costs, and (iii) identifying dysfunctional cases early. The process begins with creating a fine, involving four variables: Amount, Points, Payment, and Dismissal. Fines can be paid at various stages, and if unpaid after 180 days, a penalty is added. Offenders can appeal, potentially leading to fine dismissal. Unresolved cases are eventually sent to credit collection. Process mining is used to map the workflow, enabling the implementation of a machine learning algorithm to predict the duration of cases (short, medium, or long).

1.1 Dataset

The event log, published on 2015, is fully IEEE-XES compliant and contains a total of 150370 cases. In the road traffic fines' event log the some of following attributes are recorded:

- **amount:** The fine amount that the offender is required to pay. This is set when the fine is created. org : (Object) The resource or organizational entity responsible for handling or processing the event, such as the police officer or department involved.

- **dismissal:** The reason for dismissal of the fine, if applicable. For cases where the fine was dismissed, this column records the dismissal reason (e.g., successful appeal), using '#' or 'G', dismissed by Judge or by the prefecture. Otherwise, indicate "NIL."

- **concept:name:** The name of the event or activity associated with the fine case, such as "Create Fine," "Send Fine," or "Payment." This column is central to understanding the process flow.

- **vehicleClass:** The classification of the vehicle involved in the traffic violation.

- **totalPaymentAmount:** The cumulative amount paid towards the fine by the offender, which could reflect partial or full payments.

- **lifecycle:** Indicates the state transition in the process.

- **time:** The exact date and time when each event occurred. This is essential for tracking the sequence and duration of activities within each case.

- **article:** The specific article that was violated, providing legal context for the fine. This could relate to different traffic regulations.

- **points:** The number of points deducted from the offender's driving license due to the violation. Not all violations might involve point deductions.

- **case:concept:** The unique identifier for each fine case, grouping all related events together for a single case.

- **expense:** The costs associated with the fine, such as administrative or postal expenses, which may be added to the total amount due.

- **notificationType:** The type of notification sent to the offender.

- **paymentAmount:** The amount paid in a specific transaction, which could be a partial payment or the full settlement of the fine.

- **matricola:** represent an ID, for tracking purposes.
  This dataset offers a comprehensive view of the fine management process, tracking the financial, legal, and procedural aspects across many cases.

2 Organizational Goals

The organizational goals are used as an approach to optimizing the actual business process, starting form goals a company decided what It would be reached after the investigation. More in details, what can be achived and then, possibily improved, after using the process. We can split them on different leves, starting from general to specific:

1. **Strategic level**: can be reducing shipping times, we can use a specific point, which after reached it, we can stop sending fines to avoids losing time and money, this allows company to send only necessary post.

2. **Operational level**: a company can build predictive models to classify for example: behaviour of a specific case, level of risk, duration of cases, the latter is also showed on my little analysis and will be discussed on this paper.

3. **Tactical level**: Data collection methods will be improved, and the information system will be adjusted to capture more granular data. This refined data will be utilized to develop predictive models tailored to classify behavior or other criteria determined by the company's strategic objectives.

3 Knowledge Uplift Trail (KUT)

In an attempt to reach the goals and answer the possible questions raised by the company, the work has been divided mainly into differents parts. **KUT** is a methodology used to transform raw data, initial dataset, into actionable insights by following a structured sequence of steps that reveal the underlying properties of the data, final parts that allows who execute the process to reach an higher point of data understading. The key techniques involved in this approach include:

- Data Understading using python libraries to understand how dataset is composed to have a better understading of the data.
- Data Cleaning cleaning raw data to be ready for a more granular and reliable analysis.
- Data Filtering allows to remove outliers and have more precision on data analysis using only necessary data.
- Descriptive Analysis using statistics tools to understand the distribution of the data.
- Process Mining using algorithms and techniques to discover the process.
- Conformance Checking indentification of deviations.

Step	**Input**	**Acquired Knowledge**		**Output**
		Analytics/Models	**Type of Knowledge**	
1	Road Traffic Fine Management Process Event logs	Data overview using tools and libraries like pm4py and visualization using graphs like pie chart	Descriptive	First Scan Event Logs
2	First Scan Event Logs	Data Cleaning and Filtering: Removing unnecessary columns, NaN values conversion, Removing anomalies, Filtering on duration != NULL, Deletion of incomplete cases	Descriptive	Cleaned Event Logs without noise and almost ready for process discovery
3	Cleaned Event Logs	Plotting Cumulative and Probability Distribution Function, analysis on 25th, 50th, and 75th percentiles	Descriptive	Analyzed Cleaned Event Logs to understand distribution of cases

Table 1 – continued from previous page

Step	Input	Acquired Knowledge		Output
		Analytics/Models	**Type of Knowledge**	
4	Step 2 or Analyzed Cleaned Event Logs	Process Discovery using Direct Follow Graph (DFG)	Descriptive	High-level overview of activities inside the process
5	Step 3 or After analysis of activities	Process Discovery using Process Mining Algorithms	Prescriptive	Petri Nets of different algorithms were compared and the best one was chosen
6	Fitness	Conformance Checking: Optimization by filtering using the calculated fitness removing cases under a certain level of fitness	Prescriptive	Event Logs Filtered on Fitness
7	Optimized Event Logs on Fitness	Calculation, Extraction and Analysis of Alignments showing Log and Model moves	Descriptive	Log and Models moves
8	Step 6 or Log and Models moves	Dataset manipulation for predictive duration	Descriptive	Prepared Dataset for Machine Learning Algorithm
9	Prepared Dataset for Machine Learning	Comparing different ML algorithms	Prescriptive	Models of ML
10	Models of ML	Prediction of duration using test cases	Predictive	Prediction

Table 1: Knowledge Uplift Trail (KUT)

3.1 Data Understading

Data overview is a high-level summary that provides insight into the key patterns, trends, and distributions within a dataset. It's essential for understanding the overall structure of the data, identifying any anomalies or outliers, and guiding further analysis or decision-making processes. The two images provide an overview of activities in a Road Traffic Fine Management system. The bar chart (Figure 1) illustrates the frequency of various activities, highlighting that "Create Fine" is the most common action, followed by "Send Fine" and "Add Penalty." The pie charts (Figure 2) show that all processes start with "Create Fine" (100%), while ending activities are mostly dominated by "Payment" (44.7%) and "Send for Credit Collection" (39.2%), indicating key outcomes of the fine process. The visuals help in understanding the flow and

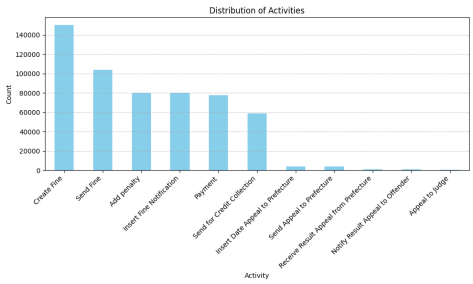


Figure 1: Total Distribution of Activities

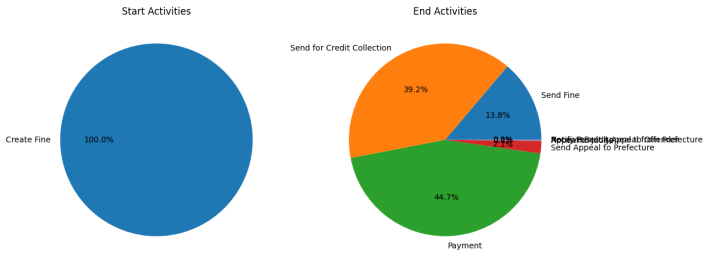


Figure 2: Distribution Start (Left) and End Activities (Right)

outcomes within the system.

3.2 Data Cleaning and Data Filtering

The columns Fig.3 were removed because they were not directly relevant to the analysis of process flow or activity outcomes. For instance, vehicleClass and notificationType focus on specific attributes that don't impact the overall process being studied. lifecycle:transition and lastSent likely contain metadata that are unnecessary for the current analysis. Lastly, matricola is an identifier that doesn't contribute to the insights being sought. Removing these columns helps streamline the data, making the analysis more focused and efficient.

```
[ ] # Removing useless columns
    filtered_log = df.drop(columns=['vehicleClass', 'lifecycle:transition', 'notificationType', 'lastSent', 'matricola'], axis=1)
```

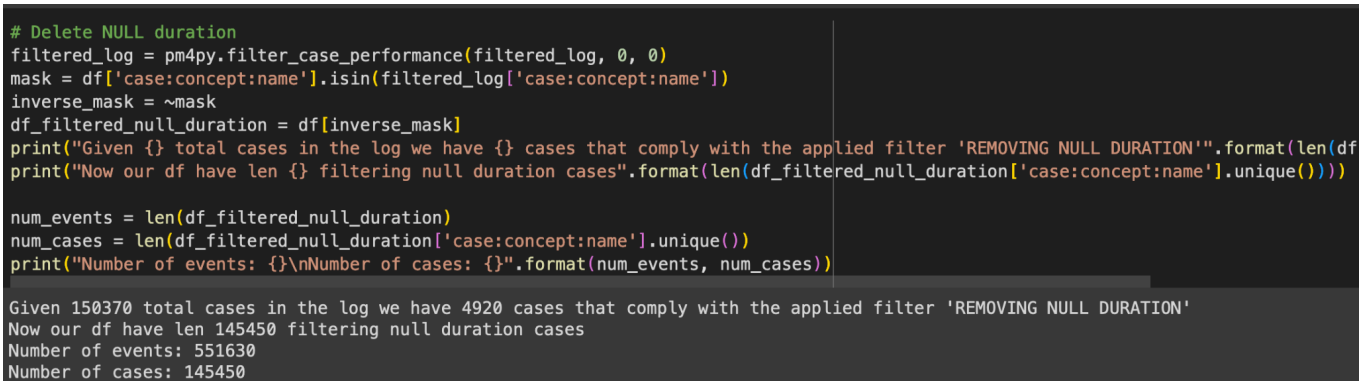
Figure 3: Removing useless columns

Filling NaNs Fig. 4 with 0 treats missing data as absence or zero occurrence, which maintains dataset consistency. This approach prevents errors in calculations and simplifies further analysis. It's particularly useful for numerical fields where NaN logically implies no value or event.

```
# Filling NaN value with 0
filtered_df = filtered_df.fillna(0)
filtered_df
```

Figure 4: Removing NaN values

This step Fig. 5 of data cleaning, focuses on removing cases with zero duration, to ensure that only valid cases are retained for analysis. The *pm4py.filter_case_performance* function filters out cases where the duration is zero. A mask is created to identify cases with valid durations, and an inverse mask is used to exclude cases with null durations. After filtering, the dataset is updated, and a summary of the results is printed. Initially, there are 150,370 total cases, and after filtering, 4,920 cases with null durations are removed. The intremediated cleaned dataset now contains 145,450 valid cases with 551,630 total events, ensuring a cleaner and more accurate dataset for further analysis.



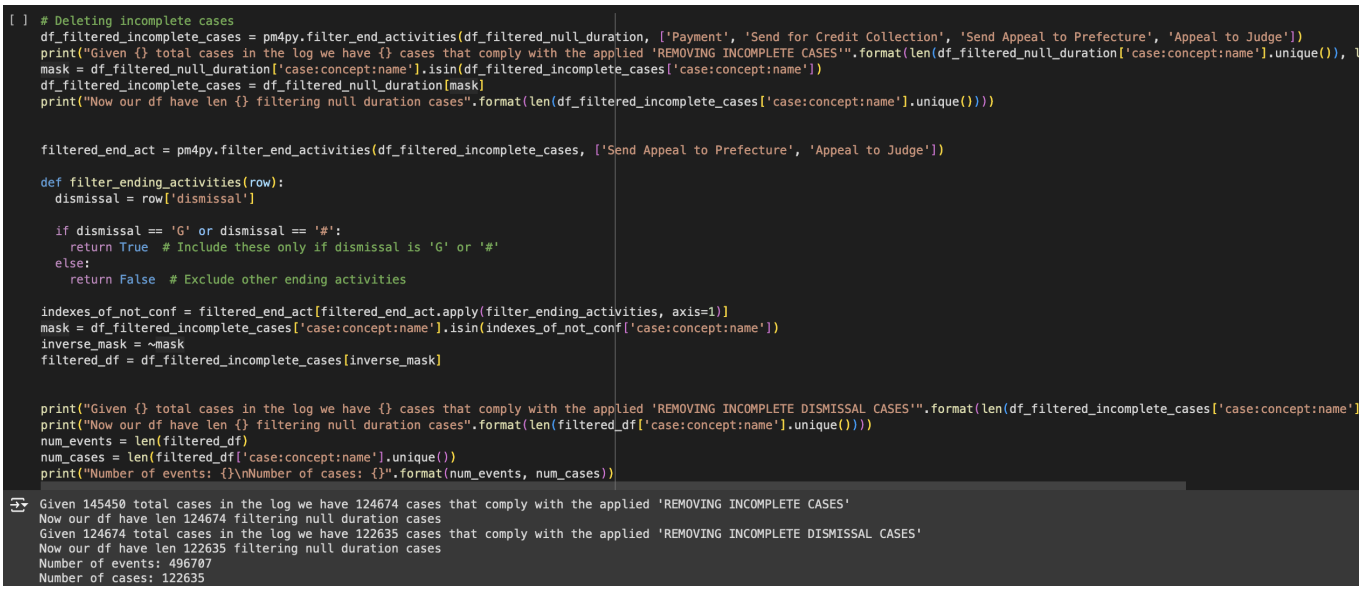
```
# Delete NULL duration
filtered_log = pm4py.filter_case_performance(filtered_log, 0, 0)
mask = df['case:concept:name'].isin(filtered_log['case:concept:name'])
inverse_mask = ~mask
df_filtered_null_duration = df[inverse_mask]
print("Given {} total cases in the log we have {} cases that comply with the applied filter 'REMOVING NULL DURATION'".format(len(df
print("Now our df have len {} filtering null duration cases".format(len(df_filtered_null_duration['case:concept:name'].unique())))

num_events = len(df_filtered_null_duration)
num_cases = len(df_filtered_null_duration['case:concept:name'].unique())
print("Number of events: {}\nNumber of cases: {}".format(num_events, num_cases))
```

```
Given 150370 total cases in the log we have 4920 cases that comply with the applied filter 'REMOVING NULL DURATION'
Now our df have len 145450 filtering null duration cases
Number of events: 551630
Number of cases: 145450
```

Figure 5: Removing cases with duration 0

This code Fig. 6 is focused on refining the dataset by removing incomplete cases, specifically those that do not end with key activities like "Payment," "Send for Credit Collection," "Send Appeal to Prefecture," or "Appeal to Judge", because they count such a valid process iteration and for example: delete from dataset all cases ending like "Create Fine". It first filters the data to retain only cases that conclude with these specified activities, ensuring that only complete processes are considered. Further filtering is applied to exclude cases where the dismissal type is not 'G' or '#', which might indicate an incomplete case, where is present "Send Appeal to Prefecture" or "Appeal to Judge". The result is a cleaned dataset, stripped of incomplete cases, ensuring a more accurate and meaningful analysis. This process is crucial to ensure that the remaining cases in the dataset represent complete and valid processes, enhancing the reliability of the subsequent analysis.



```
[ ] # Deleting incomplete cases
    df_filtered_incomplete_cases = pm4py.filter_end_activities(df_filtered_null_duration, ['Payment', 'Send for Credit Collection', 'Send Appeal to Prefecture', 'Appeal to Judge'])
    print("Given {} total cases in the log we have {} cases that comply with the applied 'REMOVING INCOMPLETE CASES'".format(len(df_filtered_null_duration['case:concept:name'].unique()), l
    mask = df_filtered_null_duration['case:concept:name'].isin(df_filtered_incomplete_cases['case:concept:name'])
    df_filtered_incomplete_cases = df_filtered_null_duration[mask]
    print("Now our df have len {} filtering null duration cases".format(len(df_filtered_incomplete_cases['case:concept:name'].unique())))


    filtered_end_act = pm4py.filter_end_activities(df_filtered_incomplete_cases, ['Send Appeal to Prefecture', 'Appeal to Judge'])

    def filter_ending_activities(row):
      dismissal = row['dismissal']

      if dismissal == 'G' or dismissal == '#':
        return True  # Include these only if dismissal is 'G' or '#'
      else:
        return False  # Exclude other ending activities

    indexes_of_not_conf = filtered_end_act[filtered_end_act.apply(filter_ending_activities, axis=1)]
    mask = df_filtered_incomplete_cases['case:concept:name'].isin(indexes_of_not_conf['case:concept:name'])
    inverse_mask = ~mask
    filtered_df = df_filtered_incomplete_cases[inverse_mask]


    print("Given {} total cases in the log we have {} cases that comply with the applied 'REMOVING INCOMPLETE DISMISSAL CASES'".format(len(df_filtered_incomplete_cases['case:concept:name']
    print("Now our df have len {} filtering null duration cases".format(len(filtered_df['case:concept:name'].unique())))
    num_events = len(filtered_df)
    num_cases = len(filtered_df['case:concept:name'].unique())
    print("Number of events: {}\nNumber of cases: {}".format(num_events, num_cases))
```

```
Given 145450 total cases in the log we have 124674 cases that comply with the applied 'REMOVING INCOMPLETE CASES'
Now our df have len 124674 filtering null duration cases
Given 124674 total cases in the log we have 122635 cases that comply with the applied 'REMOVING INCOMPLETE DISMISSAL CASES'
Now our df have len 122635 filtering null duration cases
Number of events: 496707
Number of cases: 122635
```

Figure 6: Removing incomplete cases

After the initial data cleaning and filtering, the results show that all cases begin with the "Create Fine" activity, indicating a consistent process start. The end activities are primarily split between "Send for Credit Collection" (58,997 cases) and "Payment"

(62,399 cases), highlighting the two main outcomes of the process. A smaller number of cases end with "Send Appeal to Prefecture" (1,193) or "Appeal to Judge" (46), suggesting these are less common but still significant alternate process outcomes. This distribution reflects the primary flow and key variations within the dataset. Next, we will conduct a variant analysis to identify and understand different process flows within the dataset. This will finalize our data filtering, ensuring only the most relevant cases are retained for detailed analysis.
Now Fig. 7 I'm going to filters out specific process variants that are considered anomalies based on their activity sequence. It first identifies all variants in the dataset, then checks each variant to see if "Send Fine" is preceded by activities other than "Create Fine" or "Payment." Variants that do not follow this expected pattern are marked as anomalies. These anomalous variants are then removed from the dataset, resulting in a final filtered dataset that only includes cases following the expected activity sequences, thus ensuring the data's consistency and accuracy for further analysis.

```
[ ]  # Filter Out variants like -> Create Fine;Appeal To Judge Variants and Create Fine;Insert Appel to Prefecture
     variants = pm4py.get_variants(filtered_df)
     anomalies = set()
     count=0
     for variant in variants:
         for i in range(len(variant)):

             if variant[i] == 'Send Fine':
                  if variant[i-1] not in ['Create Fine', 'Payment']:
                     anomalies.add(variant)
                     count+= variants[variant]

     print("Anomalies: ",format(count))

     final_filtered_df = pm4py.filter_variants(filtered_df, anomalies, activity_key='concept:name', case_id_key='case:concept:name', timestamp_key='time:timestamp', retain=False)
     print("FINAL FILTERED ON VARIANTS DF HAS LEN: ",len(final_filtered_df['case:concept:name'].unique()),"cases")
```

Figure 7: Variants Analysis removing anomalies

Index	Variant	Count
0	Create Fine,Send Fine,Insert Fine Notification,Add penalty,Send for Credit Collection	56482
1	Create Fine,Payment	41569
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
5	Create Fine,Send Fine,Payment	3131
6	Create Fine,Send Fine,Insert Fine Notification,Add penalty,Payment,Send for Credit Collection	1515
7	Create Fine,Send Fine,Insert Fine Notification,Insert Date Appeal to Prefecture,Add penalty,Send Appeal to Prefecture	876
8	Create Fine,Send Fine,Insert Fine Notification,Payment,Add penalty,Send for Credit Collection	522
9	Create Fine,Send Fine,Insert Fine Notification,Add penalty,Insert Date Appeal to Prefecture,Send Appeal to Prefecture	190

Figure 8: Top 10 variants result

In this analysis, I began by cleaning and filtering the dataset to remove irrelevant columns, null durations, and incomplete cases. This process ensured that only complete and valid cases were retained, focusing our analysis on meaningful data.Then I moved on to variant analysis (Fig 8 to see the top 10 variants), which is crucial for identifying and eliminating process deviations that do not follow the expected activity sequences. By filtering out these anomalous variants, I've refined the dataset to only include cases that reflect the standard or intended process flows. This final filtered dataset is now more consistent and reliable, providing a solid foundation for accurate and insightful process analysis, free from noise and irregularities. Variant analysis is particularly important as it helps maintain the integrity of the analysis by focusing on the most relevant and representative cases, ultimately leading to more trustworthy conclusions.

3.3 Descriptive Analysis

Now, I will begin the descriptive analysis phase. Descriptive analysis is crucial because it helps us understand the basic characteristics of our data, including its distribution, central tendencies, and variability. This step is essential to summarize and visualize the data in ways that are easy to interpret, setting the stage for more complex analyses.

We start by analyzing the cumulative distribution function (CDF) of case durations. The CDF provides a graphical representation of the probability that a variable takes a value less than or equal to a specific value. In this context, it shows us the distribution of case durations in days, allowing us to understand how quickly or slowly cases are processed. By plotting the CDF, we can identify key percentiles (25th, 50th, 75th), which tell us the duration below which a certain percentage of cases fall.

Moreover, this analysis involves comparing the CDF of the uncleaned distribution (before any data filtering) against the CDF of the cleaned distribution (after filtering). This comparison helps us assess the impact of the cleaning process on case durations. For instance, after filtering out anomalies or incomplete cases, the distribution of case durations may shift, and this shift can be visualized by comparing the two CDFs. We also compare key percentiles before and after the cleaning process to quantify how much the data has changed as a result of our filtering steps.

The CDF 11 plots visually, compare the distribution of case durations before and after data cleaning and filtering, helping to understand the impact of the cleaning process. Before Cleaning 9: 25th percentile (12 days): This suggests that 25% of cases were resolved within 12 days, indicating a quick resolution for a small portion of the cases. 50th percentile (198 days): The median case duration of 198 days shows that half of the cases took longer than 198 days, indicating a more prolonged resolution process for many cases. To conclude, 75th percentile (605 days): The fact that 75% of cases were resolved within 605 days suggests that a significant number of cases experienced very lengthy durations, likely due to outliers or irregularities in the data. After Cleaning 10: 25th percentile (9 days): The slight decrease to 9 days shows that the fastest 25% of cases were still resolved quickly, similar to before cleaning, but now even faster on average. 50th percentile (469 days): The median case duration increased significantly to 469 days. This suggests that while the data cleaning removed certain outliers, it also emphasized the cases that genuinely take longer, leading to a higher median. 75th percentile (644 days): The slight increase in the 75th percentile to 644 days suggests that while most of the outliers were removed, some longer-duration cases were retained, possibly reflecting more complex cases that legitimately take longer.

Overall Impact: The CDF comparison shows that the data cleaning process had a mixed impact on the distribution of case durations. While it removed some anomalies, leading to a clearer picture of the data, the median case duration increased from 198 to 469 days, highlighting that after cleaning, the dataset retained a higher proportion of long-duration cases. This suggests that the cleaning process made the dataset more representative of the actual, possibly complex, case durations, which is crucial for an accurate understanding of the overall process.

Then I've calculated the Probability Distribution Function (PDF) of case durations 12, that provides a detailed view of how case durations are distributed over time. BY taking a look to the distribution shape: is heavily right-skewed, with a significant concentration of cases having very short durations, and a smaller number of cases

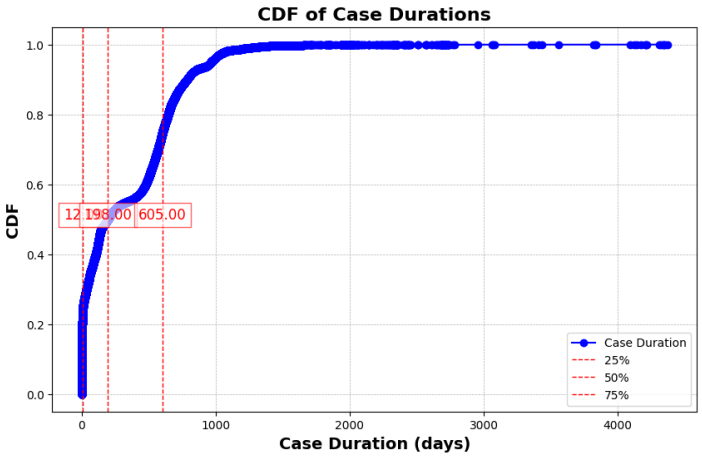


Figure 9: CDF before cleaning and filtering

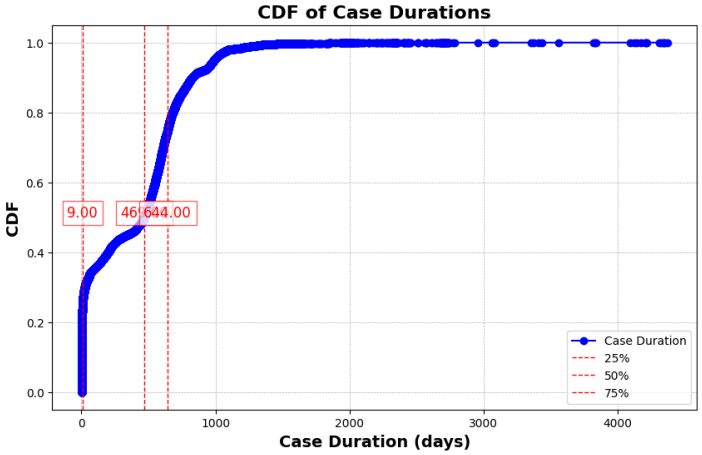


Figure 10: CDF after cleaning and filtering

Figure 11: cumulative distribution function (CDF) of case durations

taking much longer. The Highest Peak, which is at the beginning, suggests that around 30% of cases are resolved extremely quickly, likely within a few days. There are then additional, smaller peaks in the 500-1000 day range, indicating that certain groups of cases take considerably longer to resolve, possibly due to their complexity. Instead the long tail extending beyond 2000 days highlights the presence of a few cases that take an exceptionally long time to conclude. This distribution is important because: reveals a dual nature of case durations—most are resolved quickly, but significant outliers exist, representing cases that drag on for months or years. This information is crucial for understanding inefficiencies in the system and for better resource allocation in case management.

To conclude, the analysis of the cumulative distribution function (CDF) before and after highlights significant variations in the frequency and complexity of the processes involved in fine management. Initially, simpler sequences, such as the combination of 'Create Fine' and 'Payment,' dominated, reflecting a more straightforward workflow for a substantial portion of cases (with a 25th percentile at 9.0). The result of analysis under 25th percentile is: *('Create Fine', 'Payment'): 29936, ('Create Fine', 'Send*

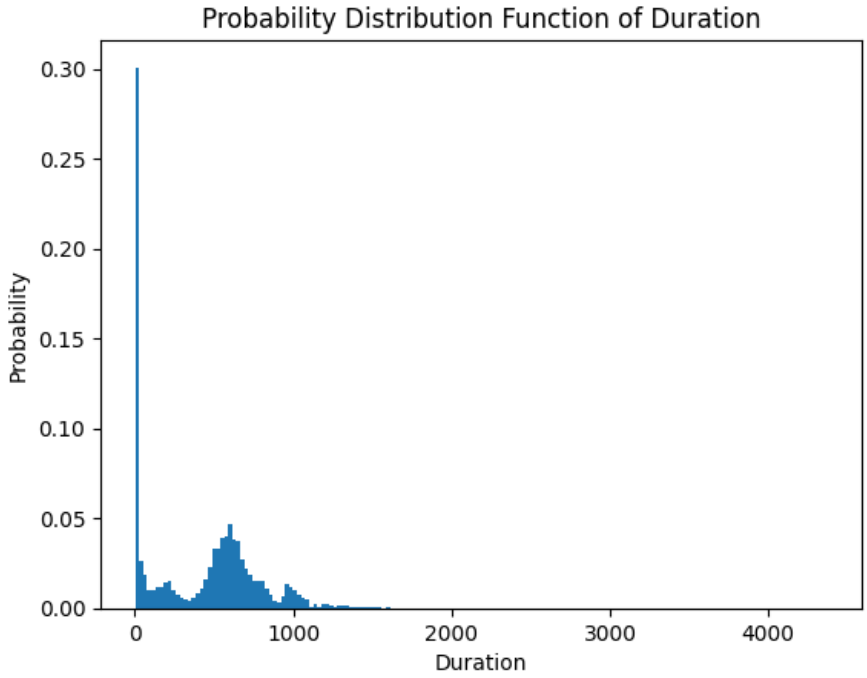


Figure 12: Probability Distribution Function (PDF) of case durations

Fine', 'Payment'): 1, ('Create Fine', 'Payment', 'Payment'): 2 By looking this we can understand the is related to simplest execution of the process. However, after the process becomes complex, there's a noticeable shift toward more complex and varied sequences, as seen with the 75th percentile at 644.0. *('Create Fine', 'Send Fine', 'Insert Fine Notification', 'Add penalty', 'Send for Credit Collection'): 27168, ('Create Fine', 'Send Fine', 'Insert Fine Notification', 'Add penalty', 'Payment', 'Send for Credit Collection'): 662, ('Create Fine', 'Send Fine', 'Insert Fine Notification', 'Add penalty', 'Insert Date Appeal to Prefecture', 'Send Appeal to Prefecture'): 66, ('Create Fine', 'Send Fine', 'Insert Fine Notification', 'Add penalty', 'Payment'): 1212, ...* These sequences often include multiple steps like 'Insert Fine Notification,' 'Add penalty,' and appeals, indicating a more intricate and prolonged case resolution process for a majority of cases, reflecting the inherent complexity and variability in handling fine-related workflows. This shift suggests that while some cases are resolved quickly and simply, a significant portion requires extensive processing, likely due to appeals, penalties, and collections.

3.4 Process Discovery

In the process discovery stage, the goal is to uncover the underlying business processes based on event logs using various algorithms. One of the key outputs is the **Directly Follows Graph (DFG)**. This graph 13 provides a high-level overview of the activities in the process and the order in which they occur. It shows how events are connected, revealing the most common paths between activities. It's a useful tool to understand the basic flow before diving deeper with more sophisticated models. Next, different process discovery algorithms were applied using Petri nets, each of which was evaluated

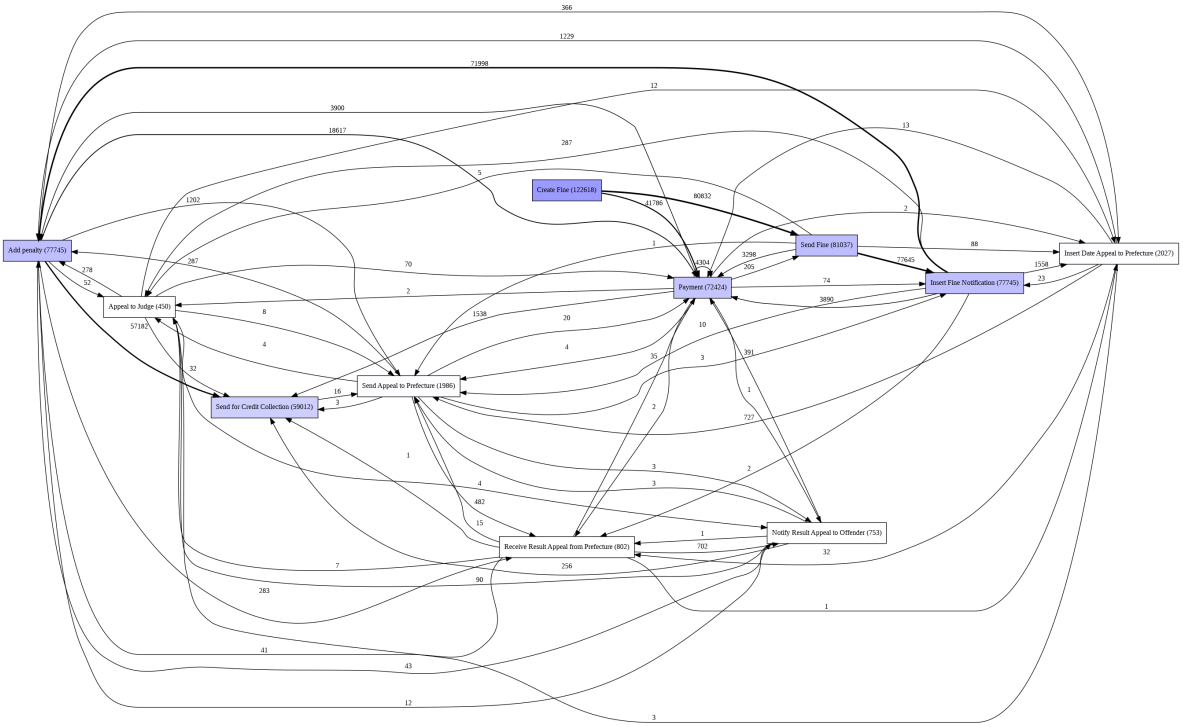


Figure 13: Directly Follows Graph (DFG)

based on three key metrics:

- **Precision**: Measures how much of the discovered behavior conforms to the actual behavior in the event log.
- **Fitness**: The degree to which the model fits the actual event log data, measured using token-based replay.
- **Simplicity**: Evaluates how easy the model is to understand (lower complexity is preferred).
- **Generalization**: Reflects how well the model can handle unseen behavior (how well it generalizes to future cases).

Algorithms:

- **Alpha Miner** (pm4py.discover_petri_net_alpha):
  Precision: 0.575
  Simplicity: 0.684
  Generalization: 0.983
  Alpha Miner produces a simple and highly generalized model, but its precision is relatively low, meaning it doesn't capture the behavior as accurately as desired.
- **Inductive Miner** (pm4py.discover_petri_net_inductive):
  Precision: 0.744
  Simplicity: 0.631
  Generalization: 0.965

The Inductive Miner strikes a good balance, providing a solid mix of high precision, reasonable simplicity, and good generalization, making it a strong choice for more detailed analysis.

- **Heuristics Miner** (pm4py.discover_petri_net_heuristics):
  Precision: 0.999
  Simplicity: 0.553
  Generalization: 0.908
  Heuristics Miner achieves almost perfect precision, meaning it closely matches the actual behavior in the logs. While its simplicity is lower than other models, it still maintains strong generalization. The trade-off is acceptable, as the high precision ensures an accurate reflection of real process behavior.

- **ILP Miner** (pm4py.discover_petri_net_ilp):
  Precision: 0.617
  Simplicity: 0.188
  Generalization: 0.984
  ILP Miner produces a highly generalized model but with a very low simplicity score, meaning the model is complex and harder to interpret. Its precision is also suboptimal.

3.5 Conformance Checking

Conformance checking is a critical step in process mining that evaluates how well the actual event log matches the discovered process model. The goal is to identify discrepancies, also known as deviations, between the expected process (as represented by the model) and the real process behavior (from the event log). This step is essential for verifying the model's accuracy and for uncovering process inefficiencies or unexpected behaviors. Steps in the Conformance Checking Process that I've done in my project:

- **Filtering Cases by Fitness**: After the process discovery phase, we calculated the fitness of the process model using token-based replay, which checks how closely the log follows the model. Cases with fitness scores less than 0.69 were identified as low-fitness cases. These cases indicate significant deviations from the expected process model were filtered out to clean the data. A total of 11 cases were found with a fitness score below 0.69. After filtering, the dataset contained 122,609 cases and 496,490 events. This step ensures the focus is on higher-fitness cases that are closer to the modeled process.

- **Conformance Checking Using Alignments**: Alignments are used for detailed conformance checking, which compares each trace in the event log with the process model and identifies deviations. Alignments show:
  Log Moves: Steps present in the log but missing in the model.
  Model Moves: Steps expected by the model but missing in the log.
  The alignment provides a detailed mapping of each event in the log to its corresponding step in the model.

- **Extracting and Analyzing Deviations**: The deviations between the log and the model were extracted. If a step in the log didn't match a step in the model

(or vice versa), it was counted as a deviation. These deviations provide valuable insights into where the real-world process differs from the modeled process.

Deviations can uncover process inefficiencies, non-compliance, or opportunities for process improvement. For example:
A large number of log moves like 'Payment' not modeled suggests that the model might need updating to reflect real-world behaviors. Numerous model moves such as 'Send for Credit Collection' missing in logs indicate that expected activities may not be happening in practice, potentially pointing to bottlenecks or skipped steps. By understanding these deviations, organizations can:
Improve process accuracy and compliance.
Optimize processes by removing unnecessary steps.
Update models to better reflect actual behavior.
To conclude conformance checking, especially using alignments, is a powerful technique to ensure that the discovered process model accurately reflects the actual business processes. Filtering out low-fitness cases improves the accuracy of the analysis, and the identified deviations provide actionable insights for process improvement.

3.6 Machine Learning

In this part of the project, I aiming to build a machine learning model to classify cases based on their duration into three categories: short, medium, and long. This classification is useful for predicting how long a case might take to resolve based on features like amount, points, and vehicle class. To achieve this, we first preprocess the data to create a suitable dataset for training and testing, followed by scaling, encoding, and resampling techniques to handle class imbalances. Detailed Explanation of the Process:
Duration Calculation: The duration of each case is calculated as the difference between the first and last event for each case. This helps in understanding the time span of each case.
Categorization: Cases are categorized into 'short', 'medium', and 'long' based on their duration in days. These categories are useful for classification.
Oversampling: Since there might be more short-duration cases, oversampling is applied to ensure a balanced dataset. This improves model performance by giving equal representation to each class.
Encoding: Categorical features such as vehicleClass are encoded into numerical values so that machine learning algorithms can process them.
Scaling: Feature scaling is performed to normalize the feature values between 0 and 1, which is important for many machine learning models to perform optimally.
Train-Test Split: The dataset is split into training and testing sets, ensuring that 20% of the data is used to evaluate the model's performance after training.
The code that I've written prepares the dataset for training a machine learning model by classifying cases into short, medium, or long categories. After creating the dataset, handling class imbalance, encoding, and scaling, the data is split for training and testing the machine learning model, laying the foundation for accurate case duration prediction.

Then after this pre-processing part I've choosen different machine learning algorithms, **Random Forest Classifier**, **XGBoost (Extreme Gradient Boosting)** and **Support Vector Machine (SVM)**. They were selected for classifying case du-

rations into three categories: short, medium, and long. Below 4.3 section, there is a detailed discussion of each algorithm, the results, and how they were optimized.

4 Project Result

In this project discussion part, I'm going to describe the result obtained on different execution part of my Process Mining Project:

4.1 Process Discovery Result

First of all, referring to 3.4 section part the Heuristics Miner Fig. 14

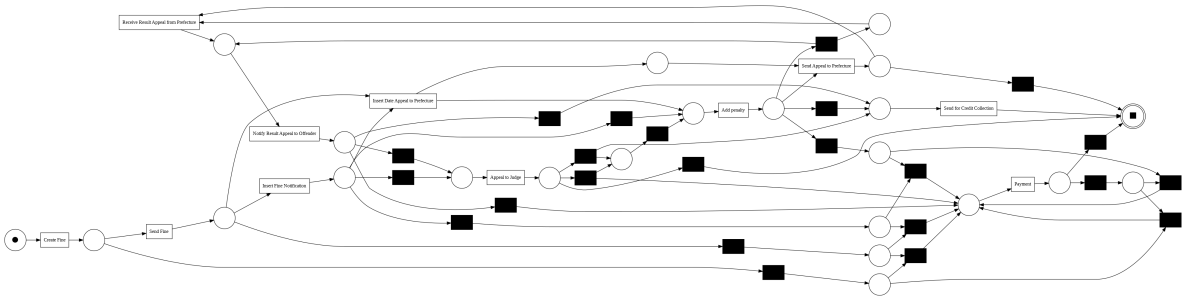


Figure 14: Heuristic Miner

emerged as the best-performing algorithm for process discovery, primarily due to its outstanding precision of 0.999. This near-perfect precision indicates that the model generated by the Heuristics Miner aligns extremely closely with the actual behavior recorded in the event logs. This is critical for ensuring that the model represents real-world processes accurately, without omitting important steps or activities.
While the simplicity of the model is lower compared to other models, meaning it may be more complex to interpret, the trade-off is justified by the high precision. In practice, having a more complex model is acceptable when it significantly enhances the accuracy of the representation of the real process. Simplicity is important, but in this case, ensuring that all variations of the process are captured accurately holds greater value.
Furthermore, the Heuristics Miner achieves a strong generalization score of 0.908, respect to the other, which demonstrates that the model can handle unseen behavior effectively. This generalization capability is essential for future-proofing the model, as it ensures that even if new or unexpected cases arise, the model will still perform reliably. In conclusion, the Heuristics Miner strikes the optimal balance between accurately reflecting real-world processes and maintaining flexibility to handle variations. Even though its simplicity is lower, the exceptional precision and strong generalization make it the most reliable choice for process discovery in this scenario, making it the ideal solution for generating insights and driving process improvements.

4.2 Conformance Checking Result

Conformance Checking Result 3.5 section, here using statistics of deviations I've: Identified a total of 213,384 deviations. This indicates significant differences between the

log and the model. The deviations were broken down into:
Log Moves: 149,956 instances where the log contained events not represented in the model.
Model Moves: 63,428 instances where the model expected events that were missing from the log.
These results are essential for understanding where and why the real-world process diverges from the expected model.
Top 5 Deviations:
The most frequent deviations give an idea of the recurring discrepancies between the log and the model.
('Payment', '≫') (Count: 57,600):
The log contains "Payment" events that are not captured in the model.
('≫', 'Send for Credit Collection') (Count: 11,211):
The model expects "Send for Credit Collection" but it is missing from the log.
('≫', 'Send Fine') (Count: 41,581):
"Send Fine" is expected in the model but missing from the log.
('≫', 'Insert Fine Notification') (Count: 44,872):
"Insert Fine Notification" is expected in the model but not present in the log.
('≫', 'Appeal to Judge') (Count: 52,092):
"Appeal to Judge" is missing from the log, despite being expected in the model

4.3 Machine Learning Result

Random Forest Classifier The Random Forest Classifier was initially trained with default parameters (n_estimators=100, random_state=42), and the following performance metrics were observed:
Accuracy: 0.464
Precision: 0.471
Recall: 0.464
F1 Score: 0.461
These results indicate that the model's performance is not ideal, as the accuracy and other metrics are around 46%. This could be due to the complexity of the dataset or the need for further optimization. Then using the confusion matrix, which was plotted to provide a visual understanding of how well the model classifies the three categories. I've used that because, It helps to identify misclassifications and gives insights into where the model struggles (e.g., predicting long durations as short, or vice versa). To improve performance, GridSearchCV was used to perform hyperparameter tuning. This method systematically tests different combinations of parameters to find the best configuration for the Random Forest model. The parameters tested included:
n_estimators: Number of trees in the forest (100, 200, 300).
criterion: Function to measure the quality of a split ("gini", "entropy", "log_loss").
max_depth: Maximum depth of the tree (5, 10, 15). min_samples_split: Minimum number of samples required to split an internal node (2, 5, 10).
max_features: Number of features to consider when looking for the best split ('sqrt', 'log2', or None).

After implementing GridSearchCV to optimize the model's hyperparameters, the results unfortunately did not demonstrate any significant improvement over the initial model's performance. This suggests that the current hyperparameter configuration, even after exploring various combinations, is not a limiting factor in achieving better predictive accuracy.

XGBoost Classifier XGBoost is a powerful and popular gradient boosting algorithm that is highly efficient in handling classification problems, especially for structured/tabular data.
The XGBoost model was trained with the following parameters:
objective: 'binary:logistic"
(though it's better suited for binary classification, we are using weighted scoring for multi-class classification).
n_estimators: 100 (number of boosting rounds).
learning_rate: 0.01 (controls the contribution of each tree to the final prediction).

Despite XGBoost being a powerful and popular gradient boosting algorithm known for its efficiency in handling classification problems, especially with structured/tabular data like in this case, the model's performance remained underwhelming.

Comparison:
Despite these efforts, the XGBoost model, underperformed. Its performance on average stayed below 0.46, notably lagging behind the benchmark set by the Random Forest Classifier.

Support Vector Machine (SVM) In addition to Random Forest and XGBoost, I explored the potential of Support Vector Machines (SVM), another powerful algorithm known for its effectiveness in high-dimensional spaces. Using the default parameters, I hoped that SVM could potentially outperform the previously established Random Forest benchmark. Comparison:
However, despite SVM's strengths, it also struggled to achieve a satisfactory performance level. The model's accuracy remained consistently below 0.46, falling short of the Random Forest Classifier's results. This suggests that the default parameters might not be ideal for our specific dataset and further hyperparameter tuning might be necessary to unlock SVM's full potential. Even with the application of GridSearchCV to optimize the SVM hyperparameters, the model's performance saw no significant improvement. This suggests that the challenge lies not in finding the optimal hyperparameter configuration for SVM, but potentially in the model's suitability for the dataset's inherent characteristics. Despite exploring various parameter combinations, the accuracy remained stubbornly below the 0.46 benchmark still setted by the Random Forest Classifier.

Conclusion:
While both XGBoost and Support Vector Machines (SVM) held promise for improving upon the baseline Random Forest model, ultimately, neither algorithm managed to surpass its performance. This was observed even after meticulous hyperparameter tuning using GridSearchCV for both XGBoost and SVM.
This result highlights a crucial point in machine learning: the superior algorithm on paper isn't always the winner in practice. **Dataset characteristics** (main cause), feature engineering choices, and even the nuances of the specific problem can significantly

influence model performance.
Both Random Forest and XGBoost possess strengths and weaknesses. Random Forest shines in its ease of implementation and interpretation. It often provides a robust baseline, especially when dealing with diverse datasets. XGBoost, on the other hand, generally boasts superior performance, particularly with structured data exhibiting complex patterns. Its sophisticated handling of boosting rounds and regularization often leads to greater precision.
The key distinction often emerges in the precision and generalization capabilities. A well-tuned XGBoost model often demonstrates superior precision and recall, especially when tackling complex cases with higher variability. These cases might pose challenges for Random Forest, leading to less accurate predictions.
Although Random Forest serves as a reliable starting point, extensive hyperparameter tuning may not elevate it to the level of refinement achievable with XGBoost.
In my scenario, despite the theoretical advantages of XGBoost and SVM, **Random Forest emerged as the most effective model**. This highlights the importance of a comprehensive model selection and evaluation process, where theoretical expectations are tested against actual performance on the specific dataset.

5 Improvements

Enhancing Model Accuracy: First and foremost, achieving better accuracy on my ML algorithms implementation is crucial. This can be accomplished by experimenting with various model architectures, tuning hyperparameters, and employing advanced techniques such as ensemble methods or deep learning models. By iteratively refining our approach and validating it with diverse datasets, aiming to improve predictive performance and reduce error rates.
Incorporating Additional Data Attributes: To gain a deeper understanding of driver behavior and enhance the effectiveness of our models, it is essential to incorporate additional data attributes. For example, including information such as the driver's age or the duration since obtaining their driving license can provide valuable insights into their skill level and experience. By integrating these variables, we can develop a more nuanced understanding of driver profiles, leading to improved model predictions and a more accurate assessment of risk levels.
Feature Engineering and Data Enrichment: Beyond adding basic demographic information, further feature engineering and data enrichment can significantly enhance model performance. This includes creating derived features, such as the frequency of past violations, the severity of infractions, and patterns in payment behavior. Enriching the dataset with external sources, such as traffic conditions or weather data, can also provide context that influences driving behavior and compliance.
By focusing on these areas for improvement, we can not only enhance the accuracy and reliability of our machine learning algorithms, but also for the entier process, leading to more informed decision-making and better outcomes in managing road traffic fines and related processes.

6 Code

To see Road Traffic Fine Management Process's complete code, I've linked Github Repo here: https://github.com/thelori91/Business-Information-Systems-PJR