Road Traffic Fine Management Process

Andrea Ingoglia

Anno accademico 2023/2024

Contents

1 Introduction

This project aims to leverage process mining techniques to discover the business process to be optimized in the overall management of road traffic fines, setting new organizational goals. In the first section, the event logs collected by the police information system are analyzed, in order to understand the statistical properties that distinguish both paid and unpaid cases. The main objective is to discover the current process (AS-IS) from event logs, identify an optimal process from it (TO-BE) and then analyze the discrepancies about the interpolated process and the segment of unpaid cases. At the end, the operational goals are set and further improvements are proposed.

2 The Knowledge Uplift Trail

The Knowledge Uplift Trail (KUT) is a method used to turn raw data into useful insights. It involves a series of steps that eventually provide a clearer understanding of the underlying properties of the data. Some of the techniques adopted in this methodology are:

1. **Data cleaning** prepares the event log by standardizing data, filling missing values or converting categorical variables to numerical ones;
2. **Data filtering** sets conditions to filter the data, keeping only cases that meet specific criteria, e.g. we might want to keep cases where the final activity was a payment;
3. **Descriptive analysis** uses statistical tools to understand the distribution and the characteristics of the data.
4. **Process mining** uses process mining techniques to discover the process;
5. **Conformance checking** identifies any deviations from the typical sequence of events;
6. **Intervention strategies** are designed based on the insights from the previous steps and they are continuously improved through monitoring and measurement.

2.1 Dataset

This section provides an overview of the dataset used in the study, detailing the different attributes that capture different aspects of the fine life cycle, from creation to payment. Below are the key attributes of the dataset:

- *case:concept:name* identifies the unique case to which the events belong;
- *concept:name* describes the specific activity related to the processing of a fine, such as "Create Fine," "Send Fine," or "Payment";
- *time:timestamp* records the exact date and time when each event occurred;
- *dismissal* indicates whether the fine was dismissed. *NIL* means the fine was not dismissed, while *#* or *G* indicate that the fine was dismissed by a judge or by the prefecture;
- *vehicleClass* specifies the class of the vehicle involved in the fine;
- *article* refers to the regulation under which the fine was issued;
- *points* shows the number of points deducted from the driver's license;
- *amount* contains the monetary value associated with the events "Create Fine" and "Add Penalty";
- *expense* lists additional expenses associated with the event "Send Fine";
- *paymentAmount* records the amount paid in a "Payment" event.

2.2 Data Cleaning

1. In order to ease the processing of event logs, we need a function that consolidates various financial values into a unified *amount* column. The *format_amounts* function operates as follows:

```python
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

event_logs['amount'] = event_logs.apply(format_amounts, axis=1)
```

2. To further increase the clarity of the data, we need to standardize the values in the *dismissal* column facilitating its interpretation. The *format_dismiss* function applies specific criteria to achieve this standardization; it operates as follows:

   (a) If the *dismissal* value is # or *G*, the function returns *Yes*, indicating that the fine was dismissed;

   (b) If the *dismissal* value is *NIL* or (missing) *NaN*, the function returns *No*, indicating that the fine was not dismissed;

   (c) For all other values, the function returns *Unknown*, as these values do not clearly indicate the dismissal status.

```python
def format_dismiss(row):
    if row['dismissal'] in ['#', 'G']:
        return 'Yes'
    elif row['dismissal'] == 'NIL':
        return 'No'
    elif pd.isna(row['dismissal']):
        return 'No'
    else:
        return 'Unknown'

event_logs['dismissal'] = event_logs.apply(format_dismiss, axis=1)
```

3. To maintain the relevance of our dataset, it is essential to filter out cases marked with a *Unknown* dismissal code, ensuring that only cases with clearly defined status are considered in the analysis. The following code accomplishes this:

```python
# Cases with at least an activity is assigned an unknown status to
unknown_dismissal = event_logs[event_logs['dismissal'].isin(['
    Unknown'])]['case:concept:name'].unique()

# Filter
event_logs = event_logs[~event_logs['case:concept:name'].isin(
    unknown_dismissal.tolist())]
```

4. In absence of the value of an attribute, it is important to fill any missing value with a placeholder; the following code demonstrates this process:

```python
event_logs_csv = event_logs.fillna(0)
```

5. To ensure that the analysis focuses on meaningful data, it is important to filter out cases where the duration is less or equal then zero; the following code accomplishes this:

```python
# Calculate the duration of each case
case_durations = event_logs.groupby('case:concept:name').agg(
    Duration=('time:timestamp', lambda x: (x.max() - x.min()).days)
    )

filter = case_durations[case_durations['Duration'] > 0]
event_logs = event_logs[event_logs['case:concept:name'].isin(
    filter.index)]
```

2.3 Paid Cases: Data Filtering

1. Filter out cases where the final activity is not a payment, to analyze successful and conformant process executions:

```python
# Identify the last activity for each case
last_activities = paid_cases.sort_values(by=['case:concept:name',
    'time:timestamp']).groupby('case:concept:name').last().
    reset_index()

# Removes cases that do not end with a payment
to_remove_cases = last_activities[last_activities['concept:name']
    != 'Payment']['case:concept:name']

# Update the dataset to include only conformant cases
paid_cases = paid_cases[~paid_cases['case:concept:name'].isin(
    to_remove_cases)]
```

2. Identify cases where the issued amount is equal to the paid amount, excluding cases where both amounts are zero:

```python
# Get all the activities before Payment
issued = event_logs[event_logs['concept:name'] != 'Payment']
# Sum issued amount of each activity
issued_amounts = issued.groupby('case:concept:name')['amount'].sum
    ().reset_index()
issued_amounts.columns = ['case:concept:name', 'issuedAmount']

# Filter and group paid amounts
paid = event_logs[event_logs['concept:name'] == 'Payment']

# Sum paid amount for each case
paid_amounts = paid.groupby('case:concept:name')['amount'].sum().
    reset_index()
paid_amounts.columns = ['case:concept:name', 'paidAmount']

```

```
# Join the Dataframes on case
merged_amounts = issued_amounts.set_index('case:concept:name').
    join(
     paid_amounts.set_index('case:concept:name'), how='left'
)

merged_amounts.reset_index(inplace=True)

# Fill NaN values in with zero
merged_amounts['paidAmount'] = merged_amounts['paidAmount'].fillna
    (0).astype(np.int32)

# Retain only cases where the issued and paid amounts match
    exactly (excluding cases with both amounts as 0).
compare_amounts = merged_amounts[
     (merged_amounts['issuedAmount'] == merged_amounts['paidAmount'
        ]) & (merged_amounts['issuedAmount'] != 0)
]

# Dataset of paid cases
paid_cases = event_logs[event_logs['case:concept:name'].isin(
    compare_amounts['case:concept:name'].tolist())]
```

2.4 Unpaid Cases: Data Filtering

1. To focus on cases where fines are issued but not yet paid, we need to identify records where the issued amount is greater than zero and the paid amount is zero:

```
# Get all the activities before Payment
issued = event_logs[event_logs['concept:name'] != 'Payment']

# Sum issued amount of each activity
issued_amounts = issued.groupby('case:concept:name')['amount'].sum
    ().reset_index()
issued_amounts.columns = ['case:concept:name', 'issuedAmount']

# Filter and group paid amounts
paid = event_logs[event_logs['concept:name'] == 'Payment']
# Sum paid amount for each case
paid_amounts = paid.groupby('case:concept:name')['amount'].sum().
    reset_index()
paid_amounts.columns = ['case:concept:name', 'paidAmount']

# Join the Dataframes on case
merged_amounts = issued_amounts.set_index('case:concept:name').
    join(
     paid_amounts.set_index('case:concept:name'), how='left'
)

merged_amounts.reset_index(inplace=True)
# Fill NaN values in with zero
merged_amounts['paidAmount'] = merged_amounts['paidAmount'].fillna
    (0).astype(np.int32)

# Retain only cases where the issued and paid amounts don't match
```

```
compare_amounts = merged_amounts[
  (merged_amounts['issuedAmount'] != merged_amounts['paidAmount'])
      &
  ((merged_amounts['issuedAmount'] != 0) & (merged_amounts['
     paidAmount'] == 0))
  ]

# Dataset of unpaid cases
unpaid_cases = event_logs[event_logs['case:concept:name'].isin(
    compare_amounts['case:concept:name'].tolist())]
```

2. It is important to focus on cases that have progressed to the point where they are sent for credit collection, identifying cases arrived to the final stage of the process execution; it is addressed by the following code:

```
# Identify the last activity for each case
last_activities = unpaid_cases.sort_values(by=['case:concept:name'
    , 'time:timestamp']).groupby('case:concept:name').last().
    reset_index()

# Removes cases that end with "Send for Credit Collection"
to_keep_cases = last_activities[last_activities['concept:name'] ==
     'Send for Credit Collection']['case:concept:name']

# Update the dataset to include only conformant cases
unpaid_cases = unpaid_cases[unpaid_cases['case:concept:name'].isin
    (to_keep_cases)]
```

2.5 Dataset After Cleaning

This section outlines the attributes of the dataset after data cleaning. The cleaned dataset includes the following attributes:

- *case:concept:name* identifies the unique case to which the events belong;
- *concept:name* describes the specific activity related to the execution of an event, such as "Create Fine," "Send Fine," or "Payment";
- *time:timestamp* records the exact date and time when each event occurred;
- *amount* normalizes the previous "amount," "expense," and "paymentAmount" columns;
- *dismissal* indicates whether an event is associated with the dismissal of the fine or not.

3 Paid Cases: Knowledge uplift trail

Step	**Input**	**Analytics/Models**	**Acquired Knowledge**	**Type of Knowledge**	**Output**
Step 1	Event logs	Data Cleaning	Normalization of amounts, filling NaN values and translation of dismissal codes	Descriptive	Cleaned event logs
Step 2	Cleaned event logs	Descriptive Analysis	Identifying **paid cases**	Descriptive	Cases where the issued amount matches with the paid amount
Step 3	Paid cases	Descriptive Analysis	Data filtering	Descriptive	Cases with duration greater than zero days where the final activity is a ”Payment”
Step 4	Filtered Paid Cases	Descriptive Analysis	Benchmarking payment timelines: average shipping time from creation	Descriptive	Data distribution parameters
Step 5	Plots and indicators of the data distributions	Statistical Testing	Confidence Intervals on the average shipping time	Descriptive	Statistical confidence that the observed data is representative of typical performance
Step 6	Filtered Paid Cases	Process mining	Process discovery	Descriptive	Petri nets describing the process of the most frequent variants
Step 7	Petri nets	Prescriptive	Process optimization and simplification	Prescriptive	Design of a simplified process to better understand the correct sequences of the activities.

4 Unpaid Cases: Knowledge Uplift Trail

Step	**Input**	**Analytics/Models**	**Acquired Knowledge**	**Type of Knowledge**	**Output**
Step 1	Event logs	Data Cleaning	Normalization of amounts, filling NaN values and translation of dismissal codes	Descriptive	Cleaned event logs
Step 2	Cleaned event logs	Descriptive Analysis	Identifying **unpaid cases**	Descriptive	Cases where the issued amount is greater than zero and the paid amount is zero.
Step 3	Unpaid cases	Descriptive Analysis	Data filtering	Descriptive	Cases with duration greater than zero days where the final activity is payment
Step 4	Filtered Unpaid Cases	Descriptive Analysis	Average shipping time from creation	Descriptive	Data distribution parameters
Step 5	Plots and indicators of the data distributions	Statistical Testing	Confidence Intervals on the average shipping time	Descriptive	Statistical confidence that the observed data is representative of typical performance
Step 6	Filtered Paid Cases	Process mining	Process discovery	Descriptive	Petri nets describing the process of the most frequent variants
Step 7	Petri nets	Prescriptive	Process optimization and simplification	Prescriptive	Design of a simplified process to better understand the correct sequences of the activities.

5 Conformance Checking: Knowledge Uplift Trail

Step	**Input**	**Analytics/Models**	**Acquired Knowledge**	**Type of Knowledge**	**Output**
Step 1	Petri nets of both paid and unpaid cases	Prescriptive	Construction of a simplified model that generalizes the overall process	Prescriptive	Specifications
Step 2	Event logs and specification	Conformance checking	Calculate the deviation of unpaid cases from the objective process	Descriptive	Fitness
Step 3	Fitness	Prescriptive	Conformance to the TO-BE process and identification of possible weaknesses.	Prescriptive	Future improvements

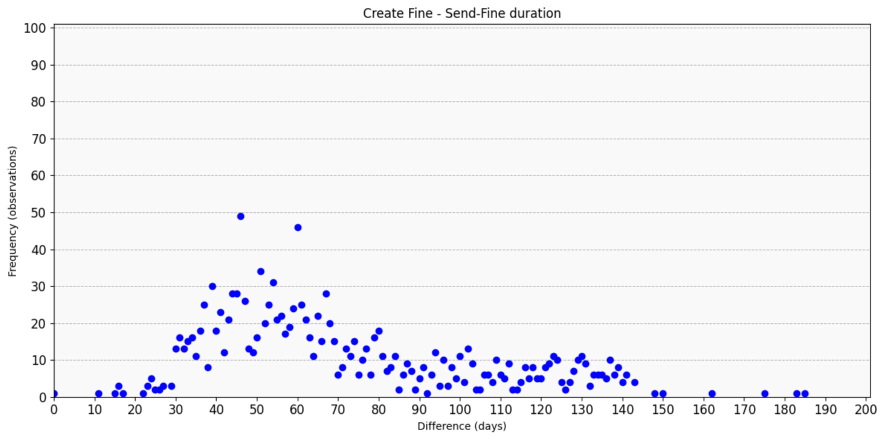
6 Statistical Analysis on Data Segments

The main objective of this section is to run descriptive statistics on the data distributions representing both paid and unpaid cases, in order to extract timing reference level KPI that characterize the process.

6.1 Paid Cases: The Create Fine-Send Fine Timeline

The primary assumption to asses the aforementioned objective, is that the time frame from the creation of the fine to its shipping is a critical factor determining the success of a case. To validate this assumption, we will:

- Measure the key performance indicators (KPIs) from the paid cases to establish benchmarks;
- Compare these benchmarks to the timelines of unpaid fines to demonstrate that unpaid fines have a longer time frame;
- Support the business hypothesis that reducing the send time can increase the success rate of fine payments.



The central tendency suggests that half of the fines are shipped within the first 60-70 days, with a high variability indicated by a standard deviation of 31 days. Statistical tests were conducted to assess the normality of the data distribution. Both the Shapiro-Wilk and Jarque-Bera tests returned p-values greater than 0.05, suggesting that the data can be considered normally distributed. The fact that the mean is higher than the median indicates a slight positive skew in the data. However, this skewness is not significant enough to challenge the assumption of normality, affirming the suitability of the data for parametric statistical analyses. Such variability suggests that while some fines are shipped promptly, others experience significant delays. This insight is crucial for identifying potential inefficiencies or bottlenecks in the process.

6.1.1 Paid Cases: Normality Test

The Shapiro-Wilk and the Jarque-Bera test are statistical test used to assess whether a dataset follows a normal distribution or not. In particular, the Jarque-Bera test works by calculating the skewness and kurtosis of the dataset, which are measures of the shape of the distribution. These values are then compared to what would be expected under a normal distribution. If the dataset is significantly different from a normal distribution, the Jarque-Bera test will flag it.

```
import scipy.stats as stats
import pandas as pd

# Test for normality using Shapiro-Wilk
```

```
shapiro_test = stats.shapiro(frequency_diff_emission['Difference'])
print('Shapiro-Wilk Test P-Value:', shapiro_test.pvalue)

# Test for normality using Jarque-Bera
jarque_bera_test = stats.jarque_bera(frequency_diff_emission['
    Difference'])
print('Jarque-Bera Test P-Value:', jarque_bera_test.pvalue)

# If P-Value > 0.05, we fail to reject the null hypothesis, meaning
    data is likely normal
if shapiro_test.pvalue > 0.05 and jarque_bera_test.pvalue > 0.05:
    print("There is not enough evidence to reject the null hypothesis
        that the data is normally distributed.")
else:
    print("The data is not normally distributed.")

### Output

# Shapiro-Wilk Test P-Value: 0.0719853863120079
# Jarque-Bera Test P-Value: 0.1969621165468974
# There is not enough evidence to reject the null hypothesis that the
    data is normally distributed.
```

6.1.2 Paid Cases: Normal distribution Confidence Interval

The confidence level is the percentage of times you expect to reproduce an estimate between the upper and lower bounds of the confidence interval, and is set by the alpha value. In this case the value to test, is the average shipping time from creation.

```
# Calculate the sample mean and standard error
mean_difference = pivot['Days'].mean()
std_dev_difference = pivot['Days'].std()
n = pivot.shape[0]
sem = std_dev_difference / (n**0.5)

# Calculate the 95% confidence interval
confidence_level = 0.95
degrees_freedom = n - 1
confidence_interval = stats.t.interval(confidence_level,
    degrees_freedom, mean_difference, sem)

print(f"Mean of the differences: {mean_difference}")
print(f"95% confidence interval for the mean difference: {
    confidence_interval}")

### Output

# Mean of the differences: 69.91117478510029
# 95% confidence interval for the mean difference: (68.27613042451182,
     71.54621914568875)
```

6.2 Paid Cases: Process Discovery

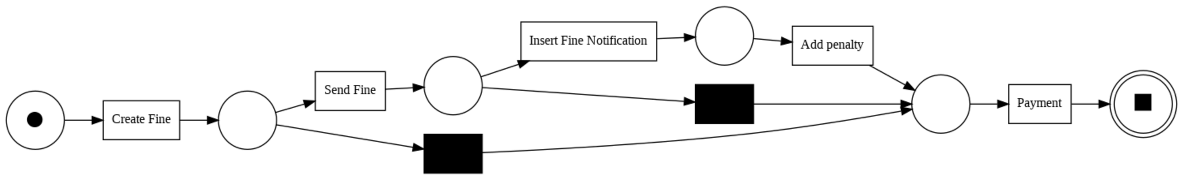
The following table summarizes the five most frequent observed variants for paid cases:

Table 1: Five most frequent variants of paid cases

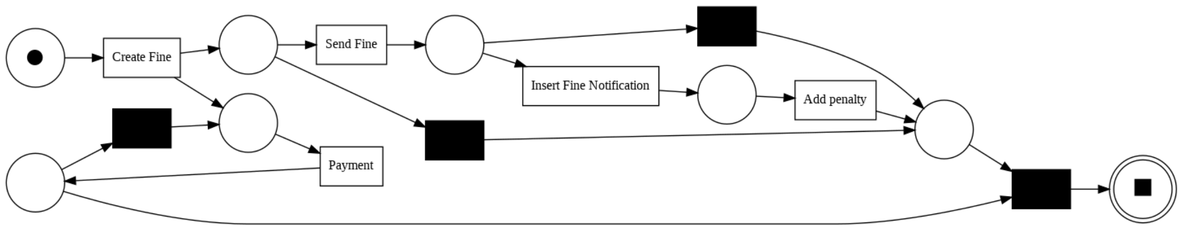
Variant	Count
Create Fine,Payment	30346
Create Fine,Send Fine,Payment	1360
Create Fine,Payment,Send Fine,Payment	18
Create Fine,Send Fine,Payment,Payment	12
Create Fine,Send Fine,Insert Fine Notification,Add penalty,Payment	3

The goal of this section is to provide a global representations of these variants and how the activities are related one each other. The following process discovery techniques from the PM4Py library are used:

- The **alpha miner** algorithm identifies the basic ordering relations between activities and constructs a Petri net that captures these relations;
- The **heuristic miner** extends the alpha miner by incorporating frequency and dependency metrics to better handle noise in the data, by considering the most frequent and significant patterns within the event log:



- The **inductive miner** works by recursively dividing the event log into smaller parts until simple models can be discovered and then combining these models:



6.3 Paid Cases: Specification of The Process Flow

Due to the complexity of the previous process representations, simpler and more effective representation of the underlying workflows are considered. In fact, by breaking down the process into these manageable segments, we can better understand the primary mechanisms of the most frequently occurring scenarios.

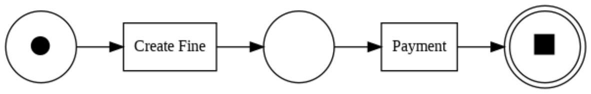


Figure 1: Process flow including only the payment

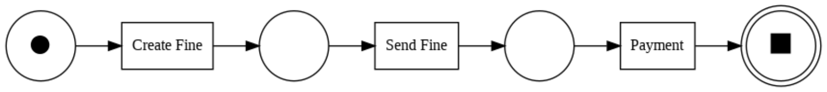


Figure 2: Process flow including shipping

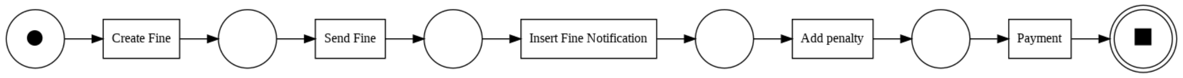


Figure 3: Process flow including penalties

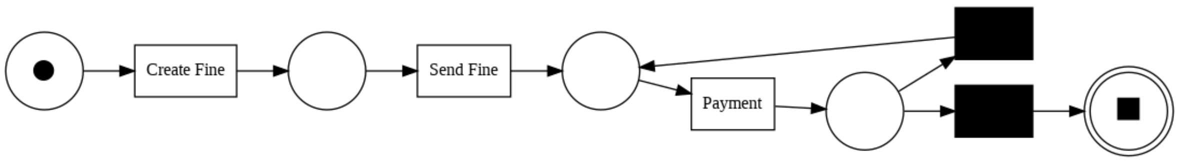
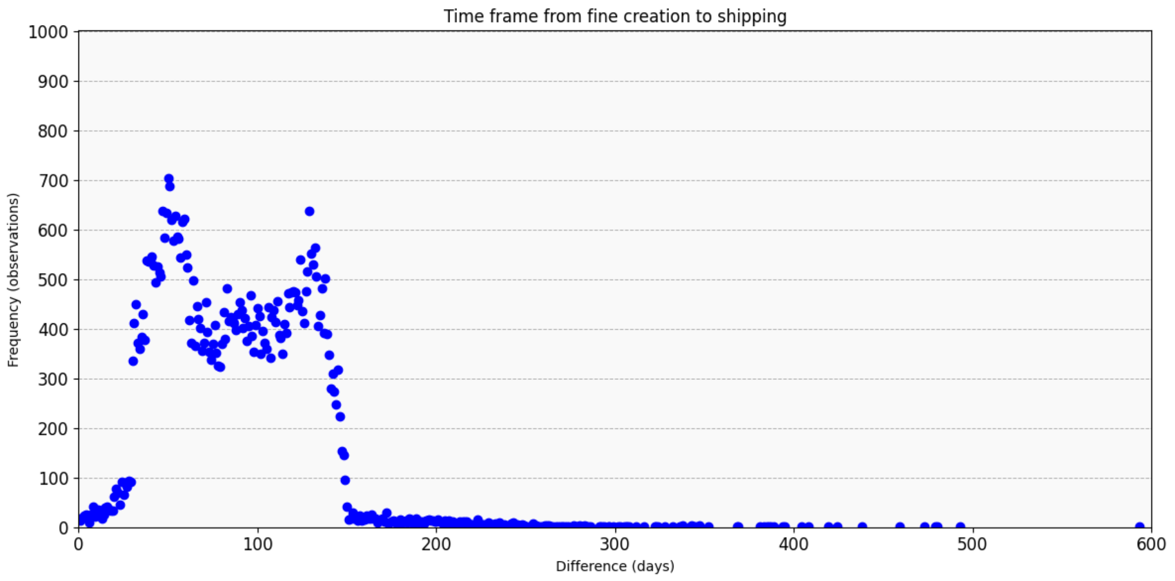


Figure 4: Process flow with recurrent payments

6.4 Unpaid Cases: The Create Fine-Send Fine Timeline

In this section, unpaid cases are analyzed to extract the properties of the variants distribution, in order to validate our primary assumption: the shipping time frame from the fine creation is one of the main factors for the success of a case. This comparison aims to demonstrate that unpaid fines have longer time frames, supporting the hypothesis that reducing the delays can increase the success rate of fine payments.



The central tendency of this distribution suggests in average that the fines are shipped within the first 83 days, with a high variability indicated by a standard deviation of 42 days, suggesting that there is a considerable variability in the shipping times. Statistical tests were conducted to assess the normality of the data distribution. Both the Shapiro-Wilk and Jarque-Bera tests returned p-values less than 0.05, indicating that the data cannot be considered normally distributed. Since the assumption of normality is not respected, non parametric statistical analyses, such as bootstrap methods should be used.

6.4.1 Unpaid Cases: Bootstrap Confidence Interval

The bootstrap method creates multiple resamples (with replacement) from a single set of observations, and computes the effect size of interest on each of these resamples. The bootstrap resamples of the effect size can then be used to determine the 95% CI.

```python
# Number of bootstrap samples
B = 10000
n = len(pivot['Days'])
bootstrap_means = np.zeros(B)

# Generate bootstrap samples and compute the statistic
for i in range(B):
    bootstrap_sample = np.random.choice(pivot['Days'], size=n, replace
       =True)
    bootstrap_means[i] = np.mean(bootstrap_sample)

# Compute confidence intervals
lower_bound = np.percentile(bootstrap_means, 2.5)
upper_bound = np.percentile(bootstrap_means, 97.5)

print("Bootstrap 95% CI for the mean:", (lower_bound, upper_bound))

### Output

# Bootstrap 95% CI for the mean: (83.63655009584777,
   84.32921158614867)
```

6.5 Unpaid Cases: Process Discovery

The following table summarizes the first five most recurrent variants for unpaid cases:

Variant	Count
Create Fine,Send Fine,Insert Fine Notification,Add penalty,Send for Credit Collection	56473
Create Fine,Send Fine,Insert Fine Notification,Appeal to Judge,Add penalty,Send for Credit Collection	107
Create Fine,Send Fine,Insert Fine Notification,Insert Date Appeal to Prefecture,Add penalty,Send Appeal to Prefecture,Receive Result Appeal from Prefecture,Notify Result Appeal to Offender,Send for Credit Collection	88
Create Fine,Send Fine,Insert Fine Notification,Insert Date Appeal to Prefecture,Send Appeal to Prefecture,Add penalty,Receive Result Appeal from Prefecture,Notify Result Appeal to Offender,Send for Credit Collection	81
Create Fine,Send Fine,Insert Fine Notification,Add penalty,Insert Date Appeal to Prefecture,Send Appeal to Prefecture,Receive Result Appeal from Prefecture,Notify Result Appeal to Offender,Send for Credit Collection	49

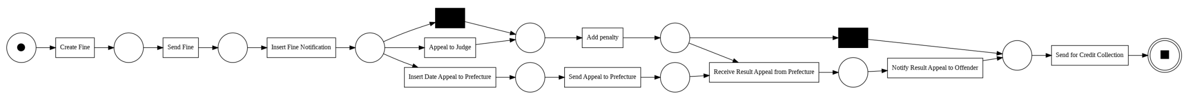
To gain insights on the most common patterns of the unpaid cases, the first five most recurring variants are considered. The heuristics miner algorithm is then used to gain a complete graphical perspective of the process through Petri nets:

```python
# From unpaid cases dataset get only the first most recurrent variants
filtered_log = pm4py.filter_variants_top_k(unpaid_cases, 5)

# Process discover using heuristics miner
```

```
net, im, fm = pm4py.discover_petri_net_heuristics(filtered_log)
pm4py.view_petri_net(net, im, fm, format='png')
```

The algorithm provides the following representation:



Create Fine
Send Fine
Insert Fine Notification
Appeal to Judge
Insert Date Appeal to Prefecture
Add penalty
Send Appeal to Prefecture
Receive Result Appeal from Prefecture
Notify Result Appeal to Offender
Send for Credit Collection

The global process can be simplified into two main sub processes

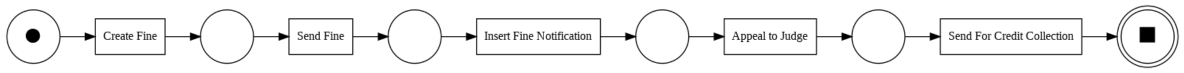
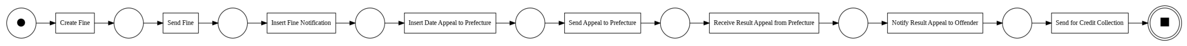


Figure 5: Unpaid cases with appeal to judge



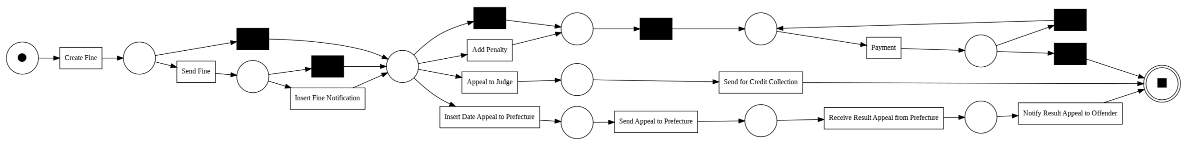
Create Fine
Send Fine
Insert Fine Notification
Insert Date Appeal to Prefecture
Send Appeal to Prefecture
Receive Result Appeal from Prefecture
Notify Result Appeal to Offender
Send for Credit Collection

Figure 6: Unpaid cases with appeal to prefecture

An additional requirement that can be set to reduce the variability of the execution instances is to constraint the "Add Penalty" activity position in the traces, by allowing to place it just after the "Insert Fine Notification" activity.

7 Specifications and Conformance Checking

A general, complete process incorporating the previous workflows is now generated using the inductive miner algorithm. This representation offers a comprehensive and well-generalized view of the possible scenarios that can occur during a process execution. This process model should be able to capture a wide range of potential situations, providing a robust overview of the entire process.


Create Fine
Send Fine
Insert Fine Notification
Add Penalty
Appeal to Judge
Insert Date Appeal to Prefecture
Send Appeal to Prefecture
Send for Credit Collection
Payment
Receive Result Appeal from Prefecture
Notify Result Appeal to Offender

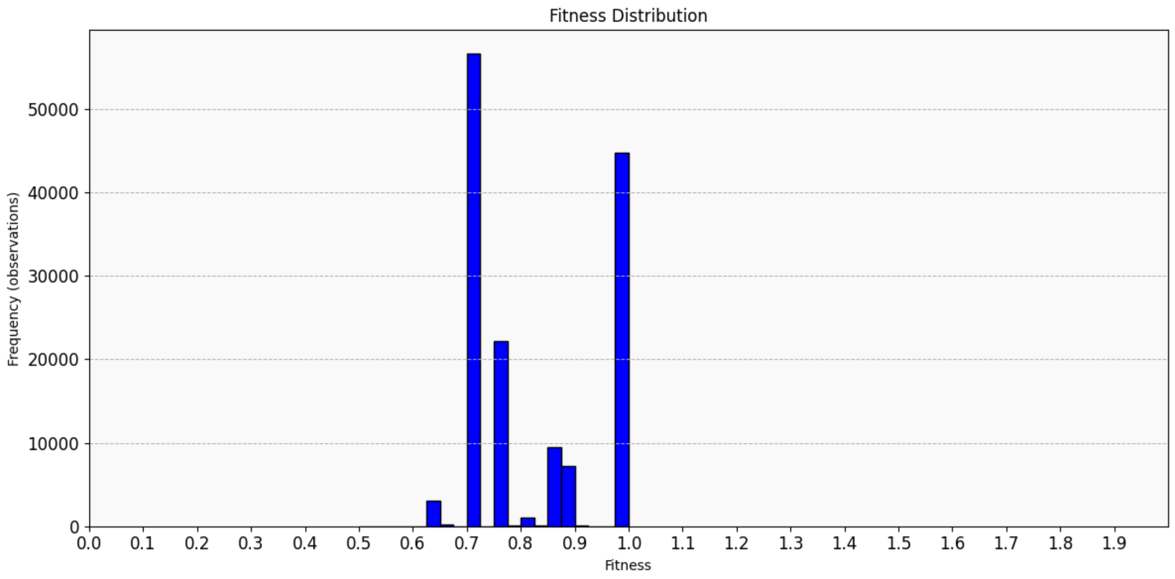
In the following lines of code, all the unpaid fines (AS-IS) cases are compared to the (TO-BE) prescripted process:

```
# Discover process as specifications require
net, im, fm = pm4py.discover_petri_net_inductive(a)

pm4py.view_petri_net(net, im, fm, format='png')

# Find diagnostic table
diagnostics = pm4py.conformance_diagnostics_alignments(event_logs, net
    , im, fm, return_diagnostics_dataframe=True)
```

The resulting histogram is:



Representing a general fitness score greater than 60%. As shown by the conformity analysis, the logs seem generally compliant with the theoretical model. Therefore, the only area of improvement is the initial hypothesis: unpaid cases have longer shipping times. Thus, I can define the reduction of shipping times as a strategic objective, impacting the operational and tactical levels in order to reduce the variability of this events.

8 Organisational goals

The organizational goals diagram illustrates an approach to optimizing the actual business process. At the strategic level, the primary focus is on reducing shipping times, which involves measures like deciding to stop sending fines after a certain period to avoid unnecessary costs.
At the operational level, the goals are to increase process compliance and identify bottlenecks, optimize resource allocation, and build predictive models to classify the risk level of cases. To achieve these, real-time monitoring dashboards based on process mining techniques will be implemented to continuously monitor if cases are managed appropriately and to detect if a case has progressed too far. Additionally, automated systems will be put in place to send reminders about due dates and the benefits of timely payments to offenders to cut postal costs. Regular training sessions are held to equip staff to manage dysfunctional executions effectively.
Moreover, data collection methods will be enhanced, and the information system will be adapted to extract more detailed information, which will be used to build predictive models for classifying the risk level of cases.

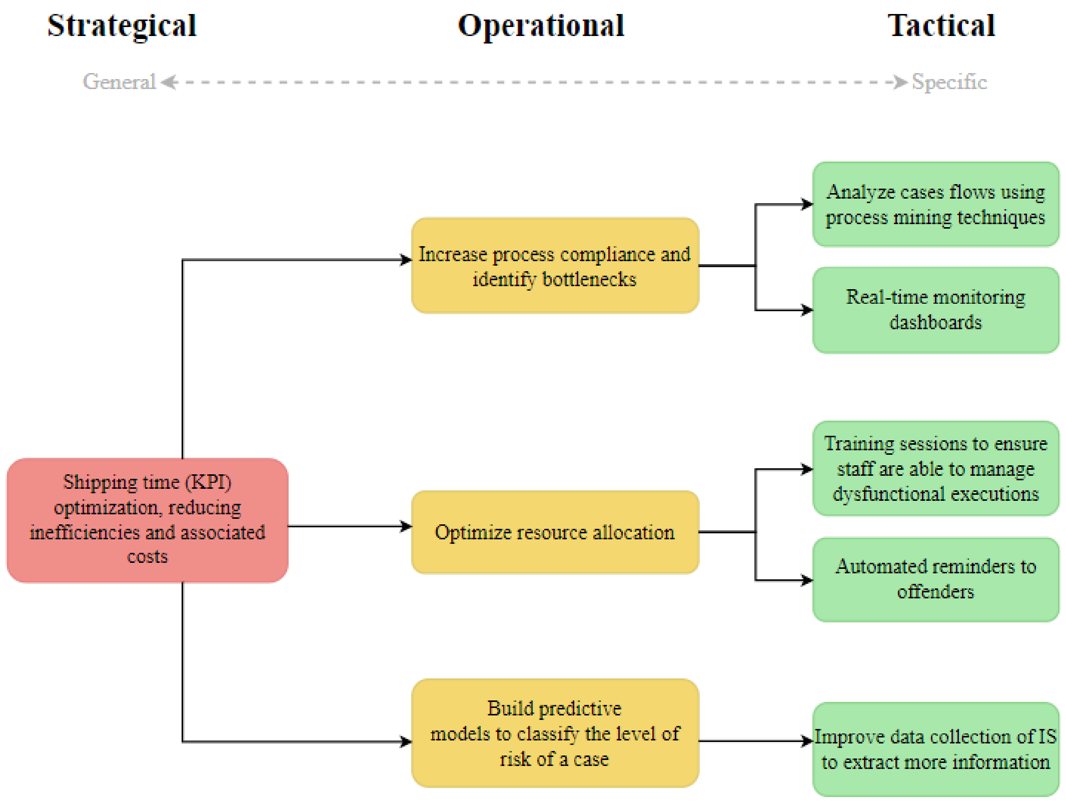

Strategical
Operational
Tactical
General
Specific
Shipping time (KPI) optimization, reducing inefficiencies and associated costs
Increase process compliance and identify bottlenecks
Optimize resource allocation
Build predictive models to classify the level of risk of a case
Analyze cases flows using process mining techniques
Real-time monitoring dashboards
Training sessions to ensure staff are able to manage dysfunctional executions
Automated reminders to offenders
Improve data collection of IS to extract more information

Figure 7: Diagram of Organisational Goals

9 Further improvements

To further improve this analysis, additional information, such as the geographic area where the fine is issued, the gender of the offender, and the classification of the area (urban city, periphery, highways) should be collected.
By incorporating this additional data, we can perform clustering analyses to identify patterns and trends within the dataset. This, in turn, would enable us to build more accurate predictive models that classify the likelihood of fine payment based on various independent variables.
In conclusion, the development of machine learning models would help staff to prioritize cases with a higher level of risk, achieving more effective outcomes.

10 Code Reference

Additional material can be found at the following link: `https://github.com/AndreaIngoglia/Business-Information-Systems`