Pietro Masolini - 2023/2024 - University of Milan - Computer Science

Road Traffic Fine Analysis

Introduction:

The aim of this project is to leverage process mining techniques to enhance the management of road traffic fines (RTF). We start by analyzing the event logs collected by the police information system to understand the statistical properties that distinguish cases. The primary objective is to map out the process from the event logs and identify an optimal process.

Data collection:

The project begins with importing the XES file and converting it into a pandas DataFrame using the pm4py library. This transformation allows for more straightforward analysis and manipulation.

```python
original_event_log = pm4py.read_xes("../Assets/Road_Traffic_Fine_Management_Process.xes")
orginal_data_frame = pm4py.convert_to_dataframe(original_event_log)
```

Then I decided to take a look into the dataframe, obtaining the start and the end activities:

```python
start_activities = pm4py.get_start_activities(original_event_log)
end_activities = pm4py.get_end_activities(original_event_log)
```

As expected, an RTF process typically starts with the `Create Fine` activity and can conclude in several ways. Although many cases end with `Payment`, a significant number also end with activities like `Send for Credit Collection` or `Send Appeal to Prefecture`.

Description	Activities
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) *Notify Result Appeal to Offender* (86) *Receive Result Appeal from Prefecture* (53)

Initial considerations:

Using python:

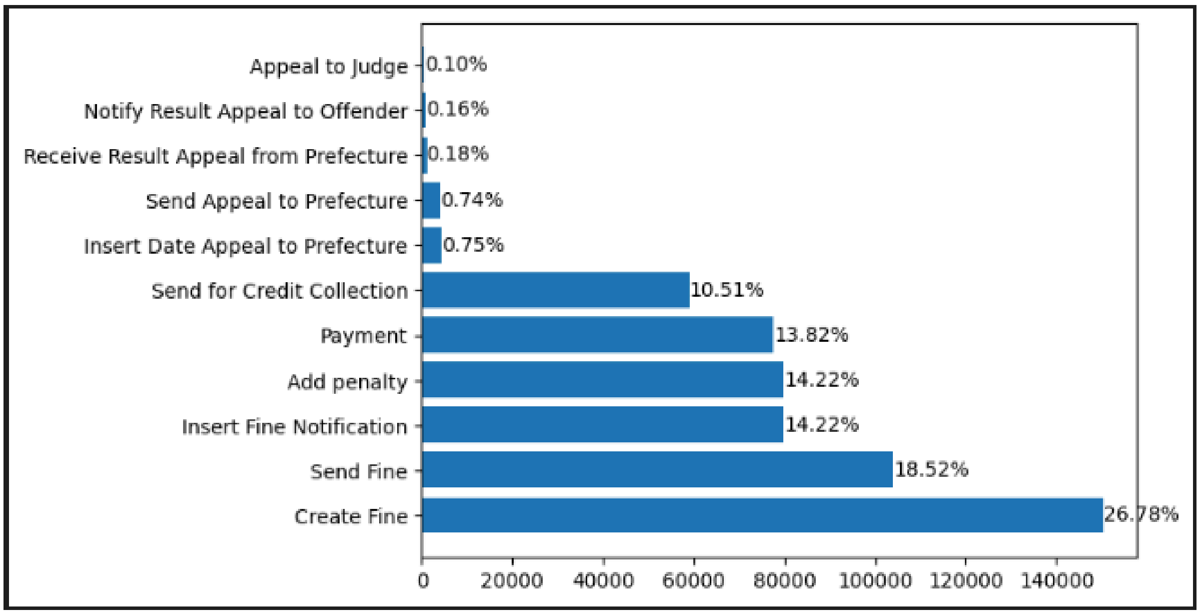
It's crucial to recognize that activities like `Send Fine` and `Send for Credit Collection` are checkpoints rather than true endpoints. For example, `Send Fine` often precedes `Payment` or `Send for Credit Collection`.

Exploring activity percentages further confirms that `Send Fine` is a transitional activity. Conversely, `Send for Credit Collection` often marks the end of a process, as evidenced by its frequency in the logs.

Based on my analysis of the percentage of all activities without considering them as Start or End points, I confirmed my previous thought that `Send Fine` is primarily a checkpoint in the traces. This activity is recorded approximately $20'000$ times as an end activity and $100'000$ times in total.

However, I cannot say the same for the `Send for Credit Collection` activity, as the numbers indicate that it is indeed a final activity.

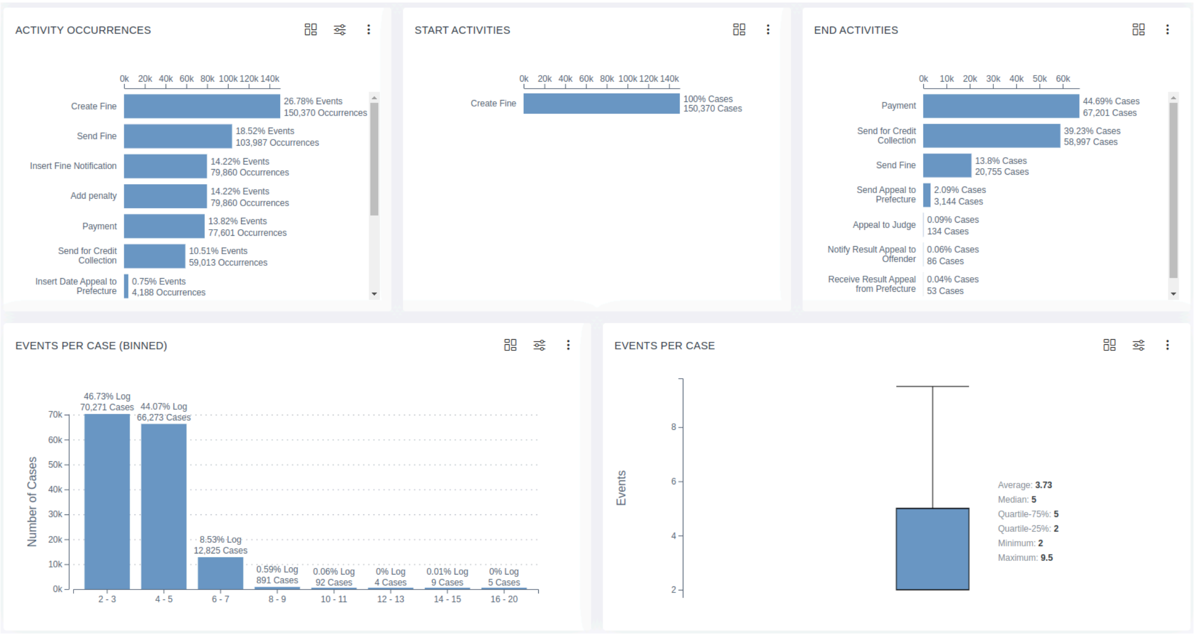
Another discovery from this initial analysis is that the activities `Add Penalty` and `Insert Fine Notification` appear to be part of most traces of an RTF.



Using PMTK:

The dataset was explored to gain insights into the number of events per case and the most common cases. This helped in identifying useful information and filtering out outliers that did not contribute significant insights.

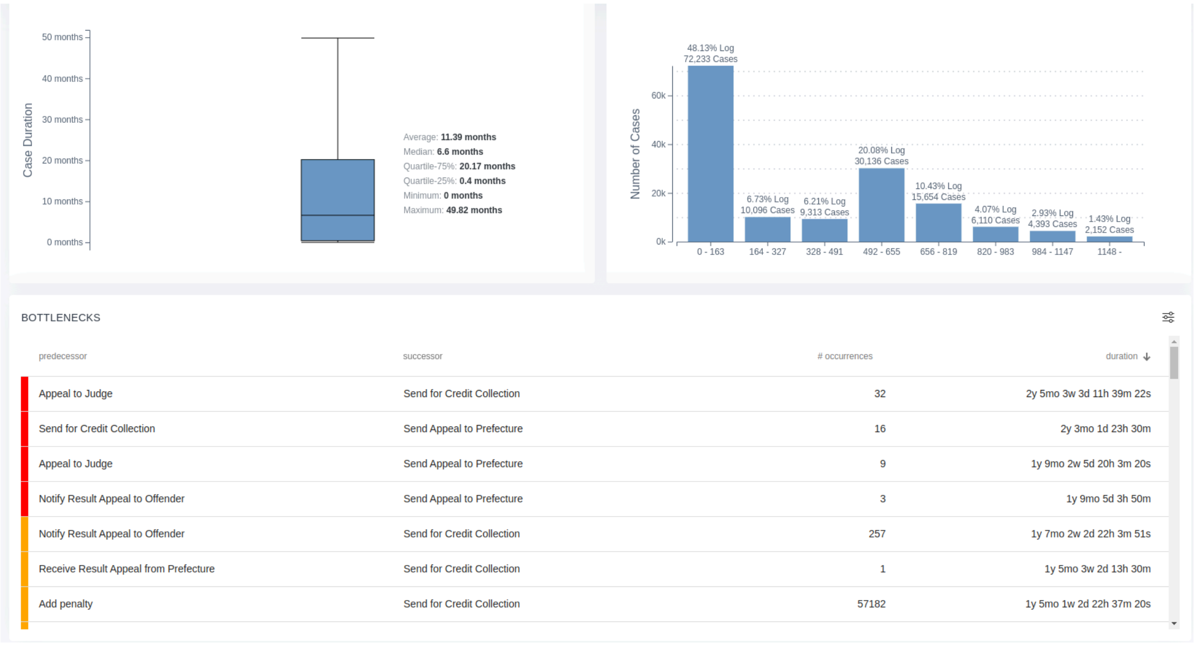
Log Summary



ACTIVITY OCCURRENCES
0k 20k 40k 60k 80k 100k 120k 140k
Create Fine
26.78% Events 150,370 Occurrences
Send Fine
18.52% Events 103,987 Occurrences
Insert Fine Notification
14.22% Events 79,860 Occurrences
Add penalty
14.22% Events 79,860 Occurrences
Payment
13.82% Events 77,601 Occurrences
Send for Credit Collection
10.51% Events 59,013 Occurrences
Insert Date Appeal to Prefecture
0.75% Events 4,188 Occurrences
START ACTIVITIES
0k 20k 40k 60k 80k 100k 120k 140k
Create Fine
100% Cases 150,370 Cases
END ACTIVITIES
0k 10k 20k 30k 40k 50k 60k
Payment
44.69% Cases 67,201 Cases
Send for Credit Collection
39.23% Cases 58,997 Cases
Send Fine
13.8% Cases 20,755 Cases
Send Appeal to Prefecture
2.09% Cases 3,144 Cases
Appeal to Judge
0.09% Cases 134 Cases
Notify Result Appeal to Offender
0.06% Cases 86 Cases
Receive Result Appeal from Prefecture
0.04% Cases 53 Cases
EVENTS PER CASE (BINNED)
Number of Cases
0k 10k 20k 30k 40k 50k 60k 70k
2 - 3: 46.73% Log 70,271 Cases
4 - 5: 44.07% Log 66,273 Cases
6 - 7: 8.53% Log 12,825 Cases
8 - 9: 0.59% Log 891 Cases
10 - 11: 0.06% Log 92 Cases
12 - 13: 0% Log 4 Cases
14 - 15: 0.01% Log 9 Cases
16 - 20: 0% Log 5 Cases
EVENTS PER CASE
Events
2 4 6 8
Average: 3.73
Median: 5
Quartile-75%: 5
Quartile-25%: 2
Minimum: 2
Maximum: 9.5

The summary adds information regarding the number of events per case, revealing the most common cases and helping me to understand which information are useful and which are not.

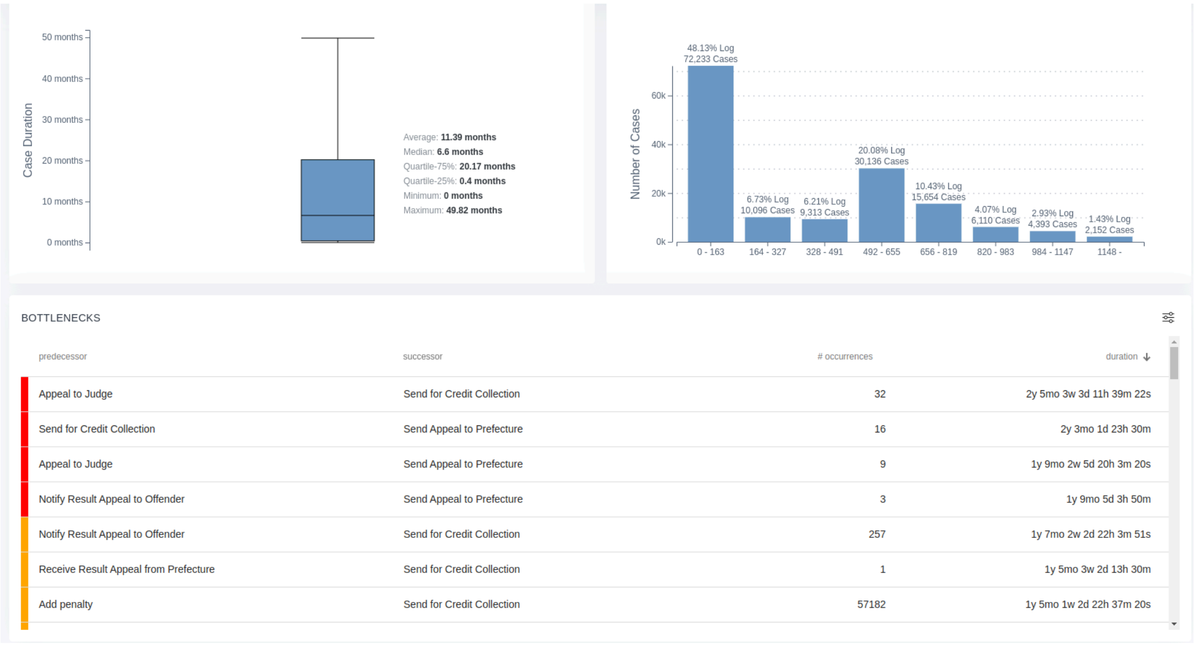
Then I had a look at the outliers, which doesn't provided any useful information.



BOTTLENECKS

	predecessor	successor	# occurrences	duration
🟥	Appeal to Judge	Send for Credit Collection	32	2y 5mo 3w 3d 11h 39m 22s
🟥	Send for Credit Collection	Send Appeal to Prefecture	16	2y 3mo 1d 23h 30m
🟥	Appeal to Judge	Send Appeal to Prefecture	9	1y 9mo 2w 5d 20h 3m 20s
🟥	Notify Result Appeal to Offender	Send Appeal to Prefecture	3	1y 9mo 5d 3h 50m
🟧	Notify Result Appeal to Offender	Send for Credit Collection	257	1y 7mo 2w 2d 22h 3m 51s
🟧	Receive Result Appeal from Prefecture	Send for Credit Collection	1	1y 5mo 3w 2d 13h 30m
🟧	Add penalty	Send for Credit Collection	57182	1y 5mo 1w 2d 22h 37m 20s

Throughput Time:

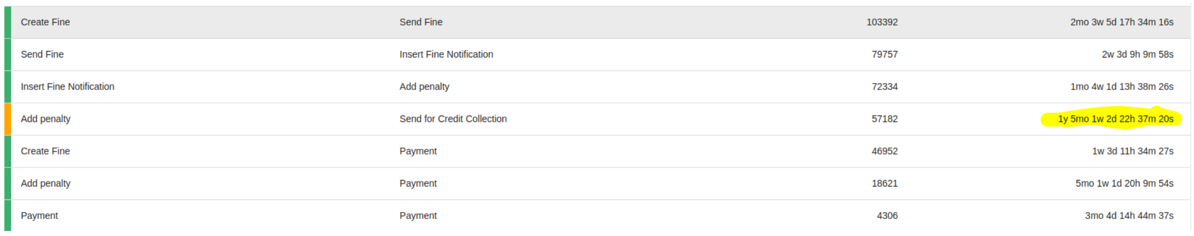


BOTTLENECKS

	predecessor	successor	# occurrences	duration
🟥	Appeal to Judge	Send for Credit Collection	32	2y 5mo 3w 3d 11h 39m 22s
🟥	Send for Credit Collection	Send Appeal to Prefecture	16	2y 3mo 1d 23h 30m
🟥	Appeal to Judge	Send Appeal to Prefecture	9	1y 9mo 2w 5d 20h 3m 20s
🟥	Notify Result Appeal to Offender	Send Appeal to Prefecture	3	1y 9mo 5d 3h 50m
🟧	Notify Result Appeal to Offender	Send for Credit Collection	257	1y 7mo 2w 2d 22h 3m 51s
🟧	Receive Result Appeal from Prefecture	Send for Credit Collection	1	1y 5mo 3w 2d 13h 30m
🟧	Add penalty	Send for Credit Collection	57182	1y 5mo 1w 2d 22h 37m 20s

A substantial percentage of events conclude quickly, but many take longer, even excluding outliers. This suggests a need to optimize certain parts of the process. For instance, initiating credit collection promptly after adding a penalty could significantly reduce resolution times.

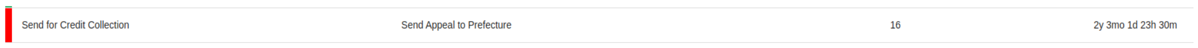
I then decided to sort the *Bottlenecks* section by occurrences:



🟩	Create Fine	Send Fine	103392	2mo 3w 5d 17h 34m 16s
🟩	Send Fine	Insert Fine Notification	79757	2w 3d 9h 9m 58s
🟩	Insert Fine Notification	Add penalty	72334	1mo 4w 1d 13h 38m 26s
🟧	Add penalty	Send for Credit Collection	57182	1y 5mo 1w 2d 22h 37m 20s
🟩	Create Fine	Payment	46952	1w 3d 11h 34m 27s
🟩	Add penalty	Payment	18621	5mo 1w 1d 20h 9m 54s
🟩	Payment	Payment	4306	3mo 4d 14h 44m 37s

Even if I consider a problem that between the creation of a fine and the actual fine, this bottlenecks doesn't seem too huge in terms of time. Instead, the highlighted one is really important for my analysis, it is suggesting me that *if credit collection was done right after adding a penalty we can save up so much time*.

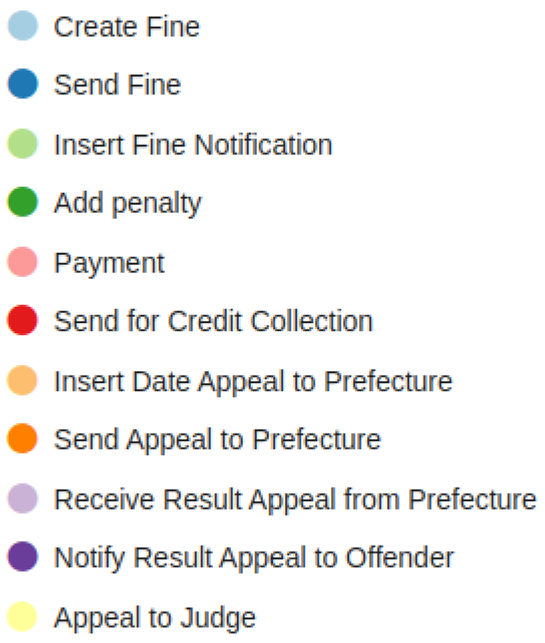
Then I discovered that `Send for Credit Collection` is mostly a final state with only 16 occurrences that involves it as a transition activity:



Send for Credit Collection	Send Appeal to Prefecture	16	2y 3mo 1d 23h 30m

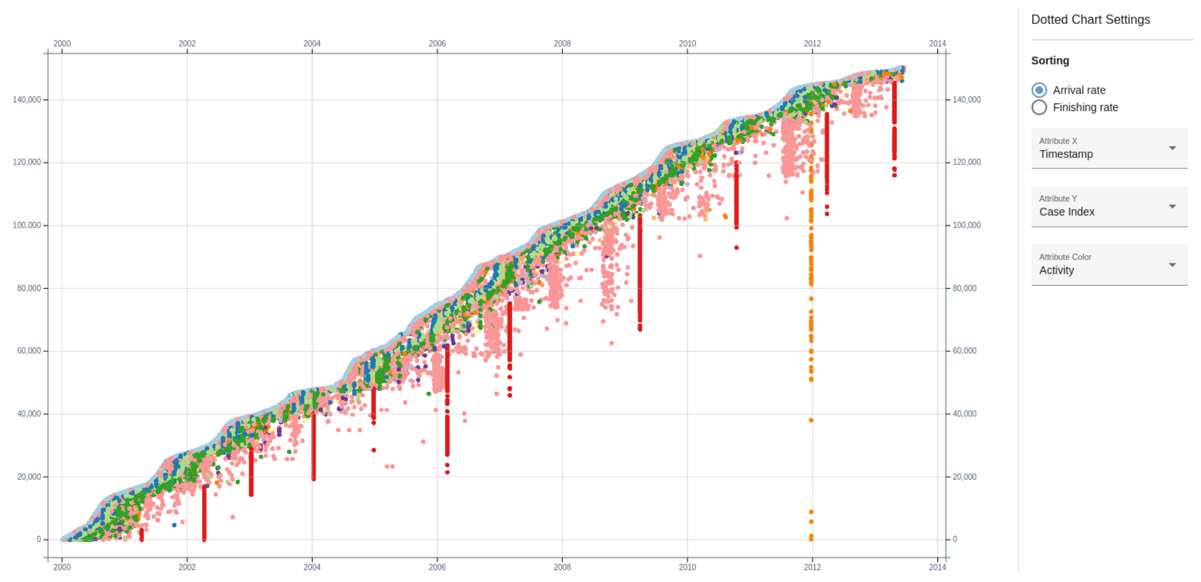
Dotted Chart:

First of all I provide the legend:



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

Then the chart:



Dotted Chart Settings

Sorting

[✓ Arrival rate]
Finishing rate

Attribute X
Timestamp

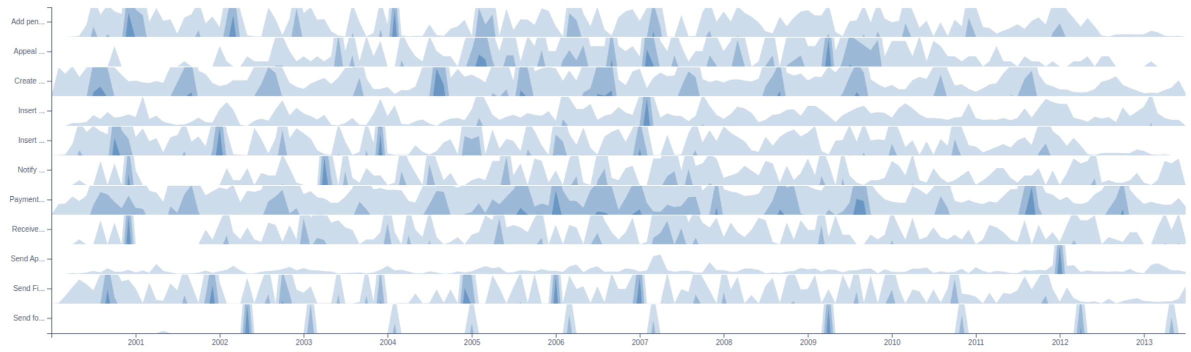
Attribute Y
Case Index

Attribute Color
Activity

This chart reveals periodic activities, such as `Send for Credit Collection` and `Send Appeal to Prefecture'`

The bottlenecks identified, such as delays between `Create Fine` and `Send Fine`, are less critical than those involving credit collection.

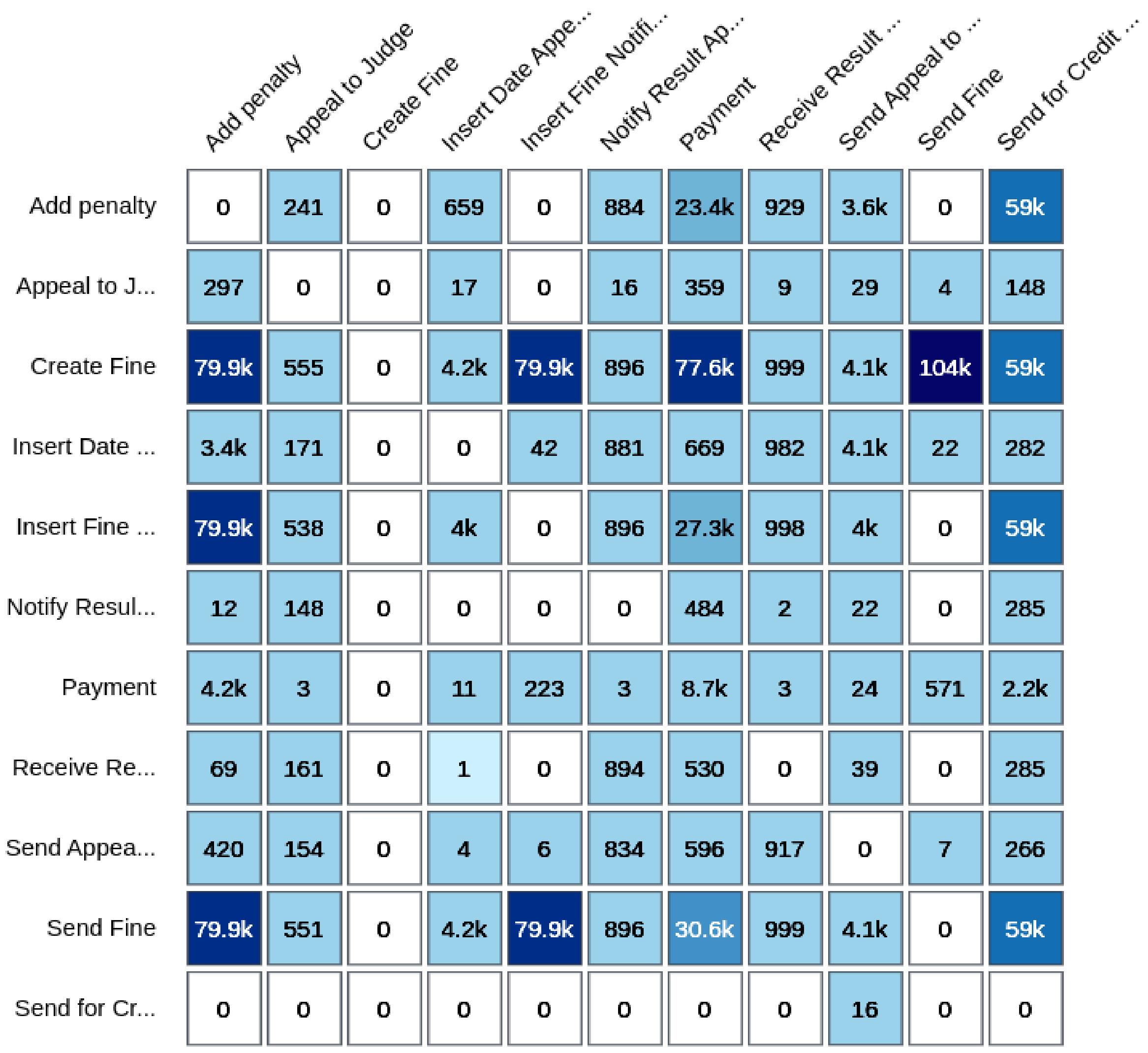
Horizon Chart:



Confirms the periodicity of certain activities but indicates peaks rather than a consistent periodic pattern for others.

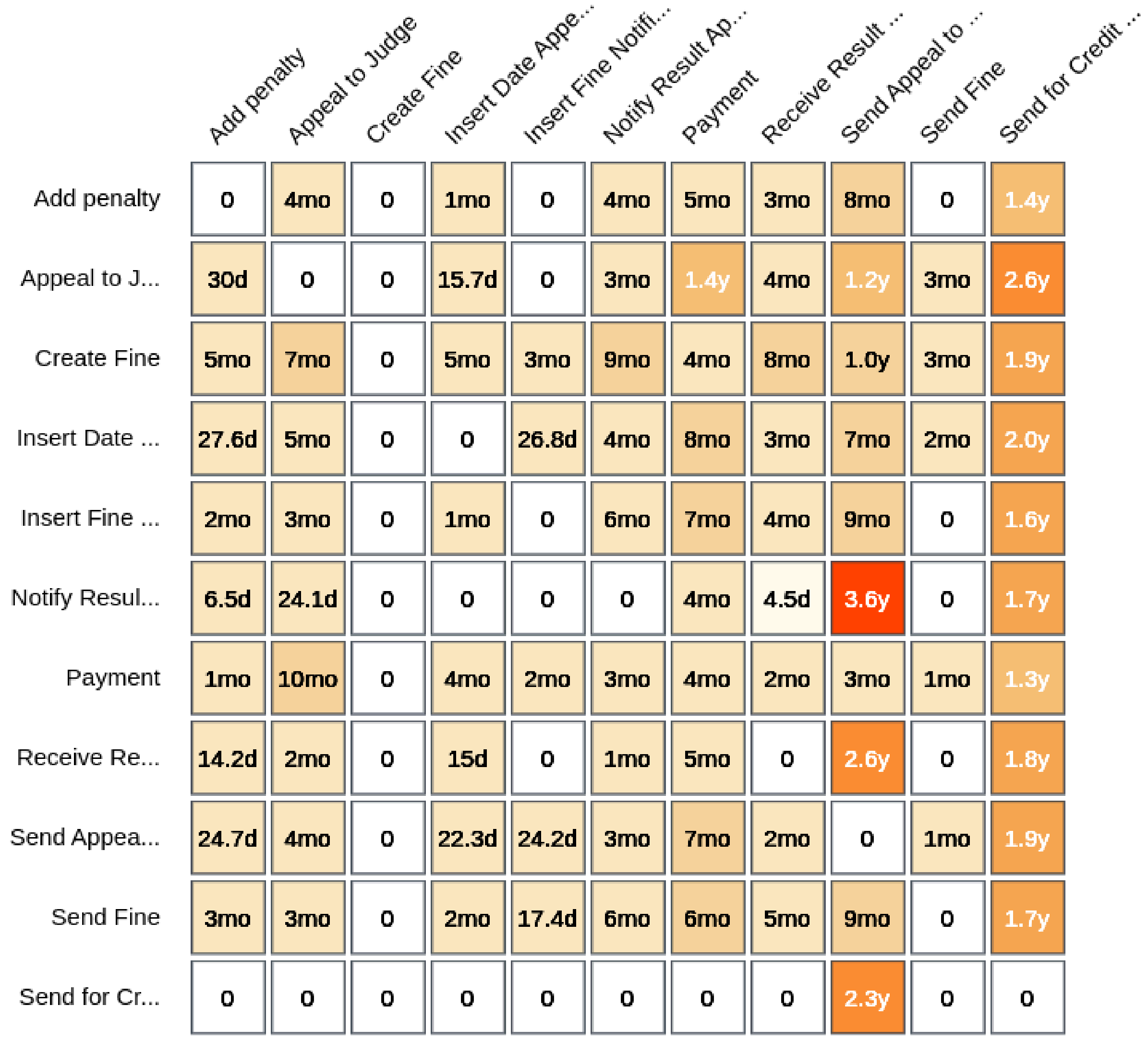
It confirms the periodicity of `Send for Credit Collection` but not the periodicity of `Send Appeal to Prefecture`, that just have a peak in 2012.

Transition Matrix:



	Add penalty	Appeal to Judge	Create Fine	Insert Date Appe...	Insert Fine Notifi...	Notify Result Ap...	Payment	Receive Result ...	Send Appeal to ...	Send Fine	Send for Credit ...
Add penalty	0	241	0	659	0	884	23.4k	929	3.6k	0	59k
Appeal to J...	297	0	0	17	0	16	359	9	29	4	148
Create Fine	79.9k	555	0	4.2k	79.9k	896	77.6k	999	4.1k	104k	59k
Insert Date ...	3.4k	171	0	0	42	881	669	982	4.1k	22	282
Insert Fine ...	79.9k	538	0	4k	0	896	27.3k	998	4k	0	59k
Notify Resul...	12	148	0	0	0	0	484	2	22	0	285
Payment	4.2k	3	0	11	223	3	8.7k	3	24	571	2.2k
Receive Re...	69	161	0	1	0	894	530	0	39	0	285
Send Appea...	420	154	0	4	6	834	596	917	0	7	266
Send Fine	79.9k	551	0	4.2k	79.9k	896	30.6k	999	4.1k	0	59k
Send for Cr...	0	0	0	0	0	0	0	0	16	0	0

Highlights transitions, showing frequent transitions from `Payment` to `Payment`, the significant role of `Create Fine` as a starting point and `Send for Credit Collection` is not utilized for a transition.



	Add penalty	Appeal to Judge	Create Fine	Insert Date Appe...	Insert Fine Notifi...	Notify Result Ap...	Payment	Receive Result ...	Send Appeal to ...	Send Fine	Send for Credit ...
Add penalty	0	4mo	0	1mo	0	4mo	5mo	3mo	8mo	0	1.4y
Appeal to J...	30d	0	0	15.7d	0	3mo	1.4y	4mo	1.2y	3mo	2.6y
Create Fine	5mo	7mo	0	5mo	3mo	9mo	4mo	8mo	1.0y	3mo	1.9y
Insert Date ...	27.6d	5mo	0	0	26.8d	4mo	8mo	3mo	7mo	2mo	2.0y
Insert Fine ...	2mo	3mo	0	1mo	0	6mo	7mo	4mo	9mo	0	1.6y
Notify Resul...	6.5d	24.1d	0	0	0	0	4mo	4.5d	3.6y	0	1.7y
Payment	1mo	10mo	0	4mo	2mo	3mo	4mo	2mo	3mo	1mo	1.3y
Receive Re...	14.2d	2mo	0	15d	0	1mo	5mo	0	2.6y	0	1.8y
Send Appea...	24.7d	4mo	0	22.3d	24.2d	3mo	7mo	2mo	0	1mo	1.9y
Send Fine	3mo	3mo	0	2mo	17.4d	6mo	6mo	5mo	9mo	0	1.7y
Send for Cr...	0	0	0	0	0	0	0	0	2.3y	0	0

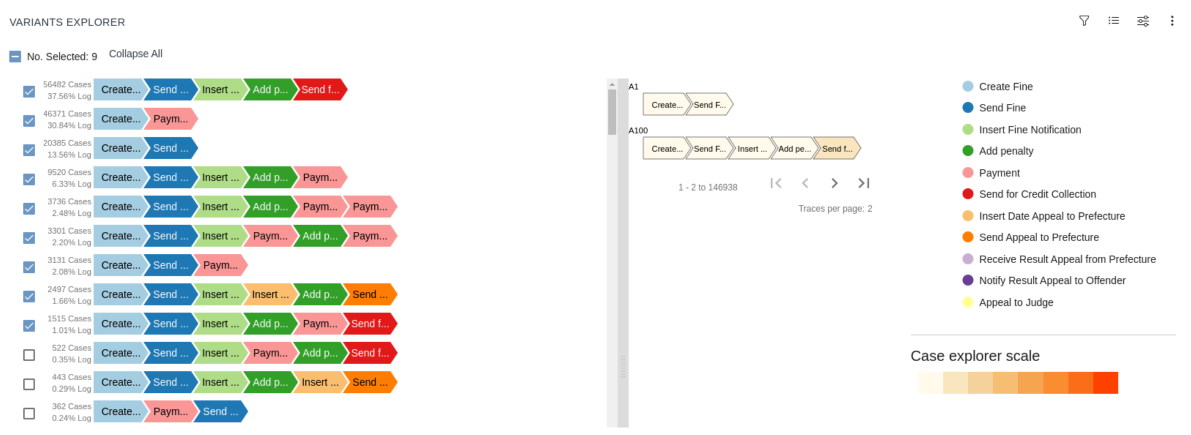
I have done the same consideration even with the performance filter, but here we can also observe bottlenecks (that often involves `Send for Credit Collection`).

By crossing the two matrix we can see the real bottlenecks:

- `Add penalty` -> `Send for Credit Collection`
- `Create Fine` -> `Send for Credit Collection`
- `Insert Fine Notification` -> `Send for Credit Collection`
- `Send Fine` -> `Send for Credit Collection`

There are others transition that can be valuable, like `Create Fine` -> `Insert Fine` but again, it seems that there is some bureaucracy in the middle of it and the time that pass between is not that much (~5m).

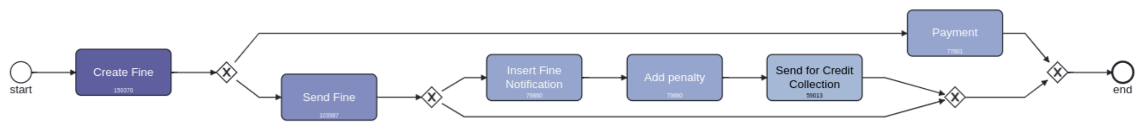
Variant Explorer:



In the image above I have selected the variants that I am gonna consider in my analysis, so the threshold in this case is about ~1%, so I will explore only 4 of the End Activities.

Process Model:

Considering 75% of the cases:



This model is not a surprise if we consider that the first three variants cover ~75%, but using a larger percentage make the model unreadable.

The Knowledge Uplift Trail:

The KUT is used to turn raw data into insights full of meanings; it involves various steps that let us better understand the informations that those data provide. Some of the techniques of this methodology are listed below:

1. **Cleaning Data**, this step standardize data and fills or drops its missing values
2. **Filtering Data**, filters out some data in order to keep only the cases that meet specific conditions
3. **Descriptive Analysis**, it uses some tools to help understanding distribution and data characteristics
4. **Process Mining**, it uses process mining techniques to discover the process
5. **Strategies** to improve and monitor the results of the previous steps

These are the steps that I have followed during the realization of this analysis:

Step	Input	Analytics	Acquired Knowledge	Type of Knowledge	Output
1	Event Logs	Data Cleaning	Standardization of `dismissal` and `amounts`, filling of `na` values	Descriptive	Cleaned Event Logs

Step	Input	Analytics	Acquired Knowledge	Type of Knowledge	Output
2	Cleaned Event Logs	Descriptive Analysis	Data Filtering	Descriptive	Traces with a maximum of 9 cases, variants number reduced to 9, cases with a duration less than 1494.6 days
3	Filtered Cases	Descriptive Analysis		Descriptive	
4		Process Mining	Process Discovery	Descriptive	
5		Prescriptive	Process optimization and simplification	Prescriptive	Design of a simplified process to better understand the correct sequences of the activities

Cleaning Data:

I have standardized the values of the `dismissal` column to facilitate its interpretation:

```python
def classify_dismissal(row):
    if row['dismissal'] in ['#','G']:
        return 'Y'
    elif row['dismissal'] == 'NIL' or pd.isna(row['dismissal']):
        return 'N'
    else:
        return '?'
original_event_log['dismissal'] = original_event_log.apply(classify_dismissal,
axis=1)
```

I have also standardized various financial values into a unified column called `amount`:

```python
def classify_amounts(row):
    if row['concept:name'] == 'Create Fine' or row['concept:name'] == 'Add penalty':
        return row ['amount']
    elif row ['concept:name'] == 'Send Fine':
        return row['expense']
    elif row ['concept:name'] == 'Payment':
        return row['paymentAmount']
    else:
        return 0
original_event_log['amount'] = original_event_log.apply(classify_amounts, axis=1)
```

Then I filled the `na` values with `None` to avoid dropping rows:

```python
original_event_log=original_event_log.fillna(0)
```

Filtering Data:

Then I have explored the dataset to take a look *on the activity number for trace, the variants that distinguish the dataset and the duration of the traces in term of days*. In this step we will shorten the dataset with:

1. A maximum number of activities for trace as 9 (The box plot Maximum is 9.5, even if the point n.3 will cut up even more)
2. We will consider only the variants that cover at least 0.24% of the cases in the event log (we will cover ~98.6% of all of the event logs this way)
3. A maximum duration of the traces of 1494,6 days (The box plot Maximum)

```python
# Filtering by the number of activities per trace
filtered_log = pm4py.filter_case_size(original_event_log, 1, 9)
# Filtering for the variants (we will cover ~98.6% of all of the event logs this way)
filtered_log = pm4py.filter_variants_top_k(filtered_log, 9)
# Filtering for duration on the remaining traces
filtered_log = pm4py.filter_case_performance(filtered_log, min_duration=0, max_duration=1494.6*24)
```

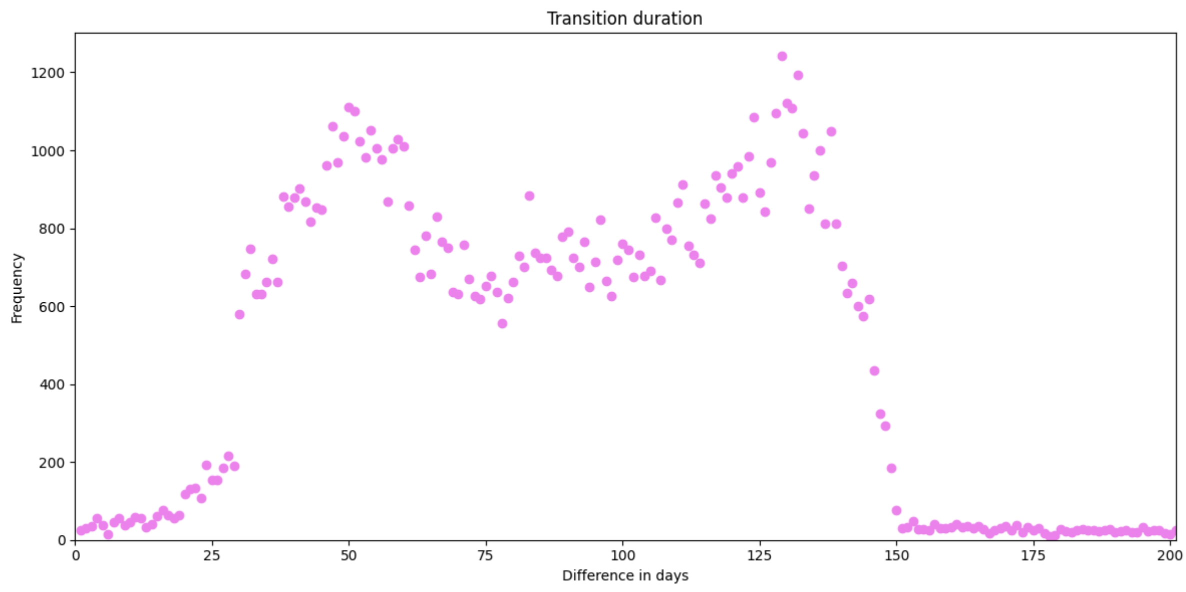
Statistical Analysis:

The main goal of this paragraph is to obtain descriptive statistics on the data distributions, in order to extract a time reference level (KPI) that distinguish the process.

`Create Fine` - `Send Fine`:

One of the critical factor that determines if a case is a success is the period starting from the creation of a fine and its shipping.

The scatter plot below shows the relation between the difference in days (that pass between creating a fine and sending it) and the frequency of them.



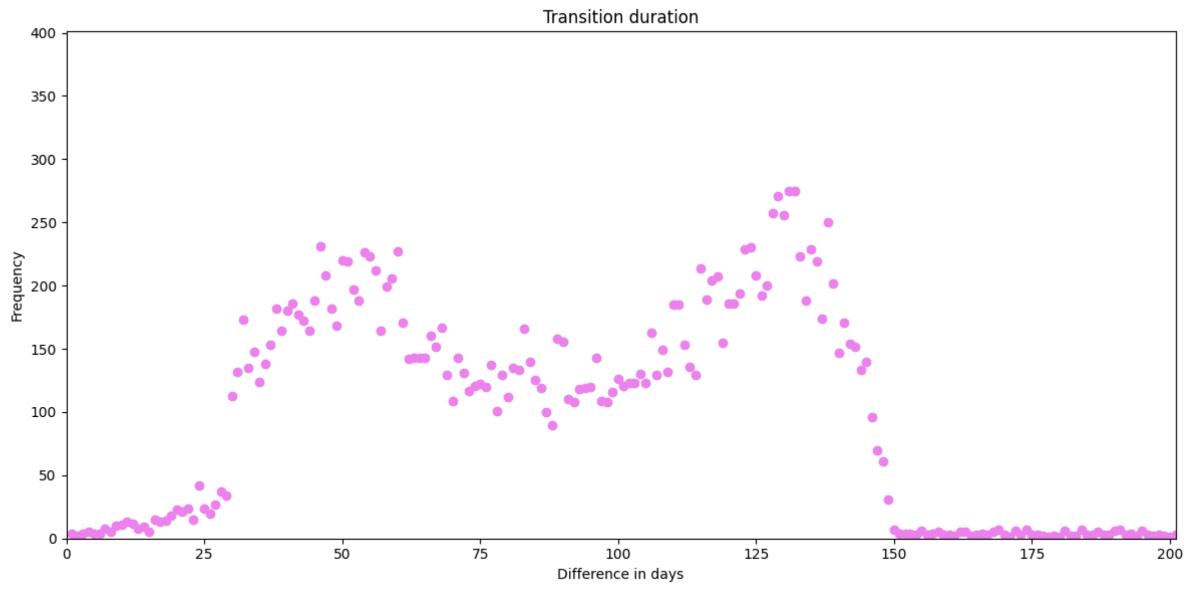
I also calculated this values (in days):

- Mean of the differences: 87.46225904371532
- Median of the differences: 87.0
- Sample Standard Deviation: 42.452027538816836

The *central tendency* suggests that fines are shipped within ~87 days with a variability states by the *standard deviation* of ~42 days.

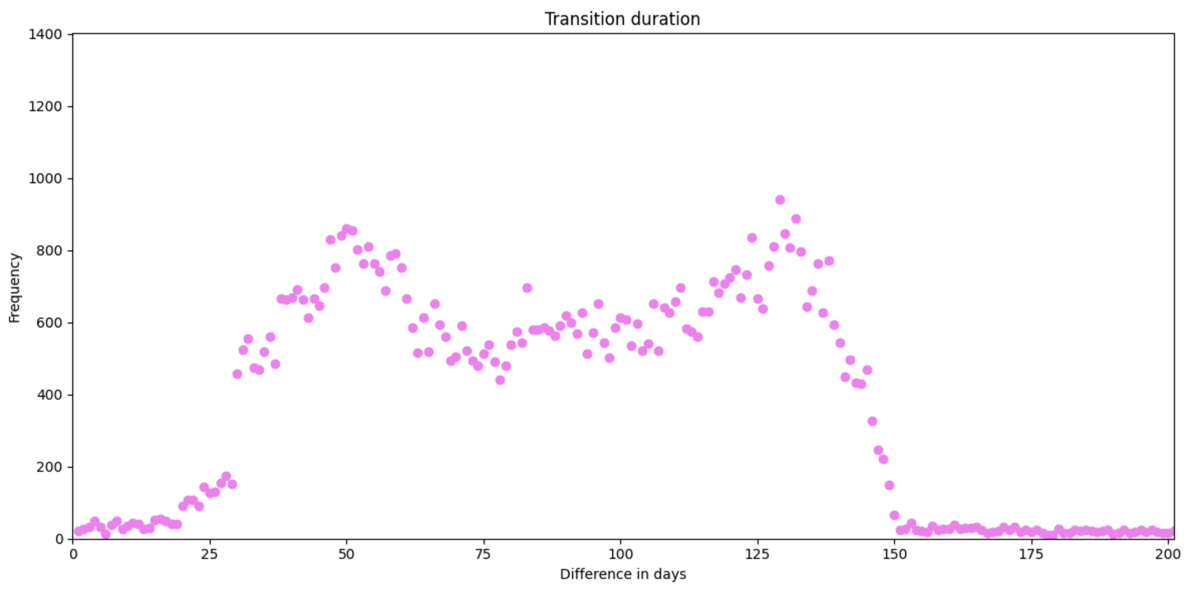
Before trying to understand if the distribution is of type *normal*, I have splitted the cases into **Paid** and **Unpaid**, to see if those two groups follow the same:

Paid `Create&Send Fine`:



- Mean of the differences: 87.8281702374059
- Median of the differences: 89.0
- Sample Standard Deviation: 42.167763832413875

Unpaid `Create&Send Fine`:



- Mean of the differences: 87.47244938665602
- Median of the differences: 87.0
- Sample Standard Deviation: 42.64077368893452

There aren't significant differences between the two distributions showed above, so I performed two different test for determine if this is a *normal distribution*:

```
#perform Shapiro-Wilk test for normality
S_TestResult = shapiro(duration_frequency["Count"])
# Shapiro-Wilk pvalue: 1.1639578788109431e-25

#perform Kolmogorov-Smirnov test for normality
K_S_TestResult= kstest(duration_frequency["Count"], 'norm')
# Kolmogorov-Smirnov pvalue: 2.704684863512192e-297

if K_S_TestResult.pvalue > 0.05 and S_TestResult.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.")
```

Both the test, Shapiro-Wilk and Kolmogorov-Smirnov, returned p-values less than the threshold of 0.05, that states that the data *cannot be considered normally distributed*.

Moreover, the data *are not normally distributed* for both the **paid** and the **unpaid** cases.

Process Discovery:

To summarizes the most frequent observed variants I realized the following table:

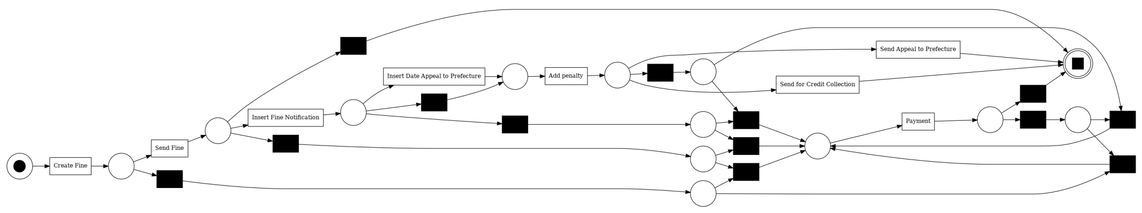
Variant	Count
'Create Fine', 'Send Fine', 'Insert Fine Notification', 'Add penalty', 'Send for Credit Collection'	56359
Create Fine', 'Payment'	46371

Variant	Count
'Create Fine', 'Send Fine'	20385
'Create Fine', 'Send Fine', 'Insert Fine Notification', 'Add penalty', 'Payment'	9519
'Create Fine', 'Send Fine', 'Insert Fine Notification', 'Add penalty', 'Payment', 'Payment'	3736
'Create Fine', 'Send Fine', 'Insert Fine Notification', 'Payment', 'Add penalty', 'Payment'	3301
'Create Fine', 'Send Fine', 'Payment'	3127
'Create Fine', 'Send Fine', 'Insert Fine Notification', 'Insert Date Appeal to Prefecture', 'Add penalty', 'Send Appeal to Prefecture'	2273
Create Fine', 'Send Fine', 'Insert Fine Notification', 'Add penalty', 'Payment', 'Send for Credit Collection'	1508

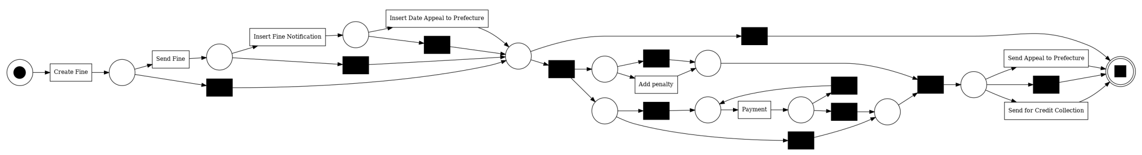
How previously said, I have choose to maintain in the data only the top 9 variants. The majority of these variants terminates into the `Payment` activity, to be precise while considering the total number of cases the 45.06% terminate on a `Payment`, while 39.48% ends on `Send for Credit Collection`.

I used the following techniques, from PM4PY process discovery, to provide a gloabal view of these variants:

- **Heuristic miner**, that extends the **alpha miner** by including frequency and dependency metrics to improve the noise handling, by considering the most significant patterns in the Event Log:

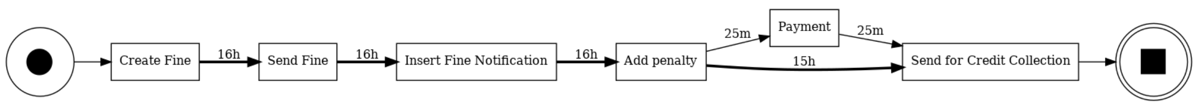

Create Fine
Send Fine
Insert Fine Notification
Insert Date Appeal to Prefecture
Add penalty
Send Appeal to Prefecture
Send for Credit Collection
Payment

- **Inductive miner**, that works recursively, dividing the Event Log into smaller pieces until simple models can be discovered and combined:

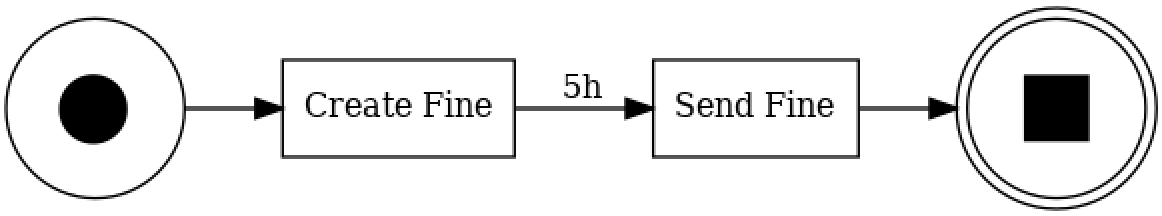

Create Fine
Send Fine
Insert Fine Notification
Insert Date Appeal to Prefecture
Add penalty
Payment
Send Appeal to Prefecture
Send for Credit Collection

These Petri Nets reflect what we have seen in the table above in a graphical way, moreover it can be great to combine the variants into smaller groups that shares the same end activity, to have a better look at the flow of the process, with the direct follow graph:

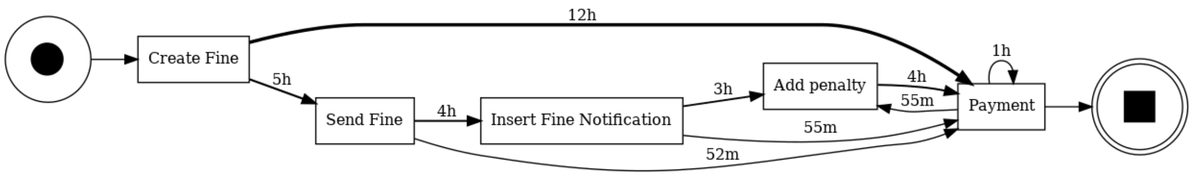
`Send for Credit Collection`



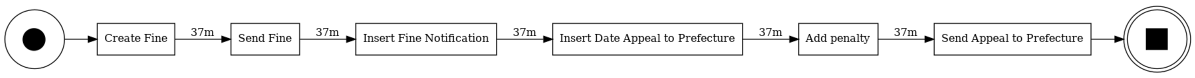
`Send Fine`



`Payment`



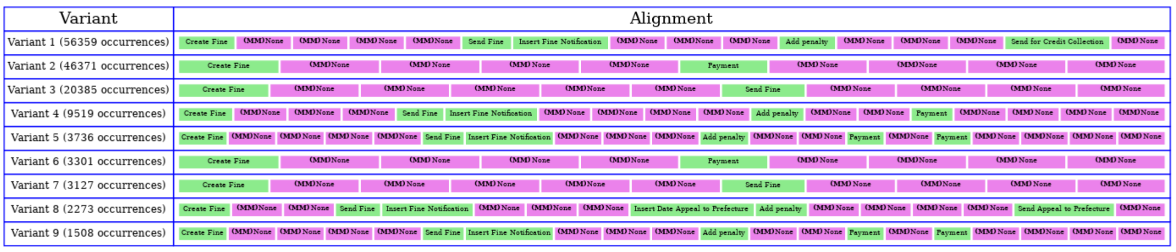
`Send Appeal to Prefecture`



Conformance Checking:

By comparing the filtered event logs (AS-IS) to the theoretical model (TO-BE), we assess process compliance.

```python
#Find diagnostic table
net, im, fm = pm4py.discover_petri_net_inductive(original_event_log,
activity_key='concept:name', case_id_key='case:concept:name',
timestamp_key='time:timestamp')
alignments_diagnostics =
pm4py.conformance_diagnostics_alignments(original_event_log, net, im, fm,
activity_key='concept:name', case_id_key='case:concept:name',
timestamp_key='time:timestamp')
pm4py.view_alignments(filtered_log, alignments_diagnostics, format='png')
```



Variant	Alignment																				
Variant 1 (56359 occurrences)	Create Fine	(MM)None	(MM)None	(MM)None	(MM)None	Send Fine	Insert Fine Notification	(MM)None	(MM)None	(MM)None	Add penalty	(MM)None	(MM)None	(MM)None	Send for Credit Collection	(MM)None					
Variant 2 (46371 occurrences)	Create Fine	(MM)None	(MM)None	(MM)None	(MM)None	Payment	(MM)None	(MM)None	(MM)None	(MM)None											
Variant 3 (20385 occurrences)	Create Fine	(MM)None	(MM)None	(MM)None	(MM)None	(MM)None	Send Fine	(MM)None	(MM)None	(MM)None	(MM)None										
Variant 4 (9519 occurrences)	Create Fine	(MM)None	(MM)None	(MM)None	Send Fine	Insert Fine Notification	(MM)None	(MM)None	(MM)None	(MM)None	Add penalty	(MM)None	(MM)None	Payment	(MM)None	(MM)None	(MM)None	(MM)None			
Variant 5 (3736 occurrences)	Create Fine	(MM)None	(MM)None	(MM)None	(MM)None	Send Fine	Insert Fine Notification	(MM)None	(MM)None	(MM)None	Add penalty	(MM)None	(MM)None	Payment	(MM)None	Payment	(MM)None	(MM)None	(MM)None	(MM)None	
Variant 6 (3301 occurrences)	Create Fine	(MM)None	(MM)None	(MM)None	(MM)None	Payment	(MM)None	(MM)None	(MM)None	(MM)None											
Variant 7 (3127 occurrences)	Create Fine	(MM)None	(MM)None	(MM)None	(MM)None	(MM)None	Send Fine	(MM)None	(MM)None	(MM)None	(MM)None										
Variant 8 (2273 occurrences)	Create Fine	(MM)None	(MM)None	Send Fine	Insert Fine Notification	(MM)None	(MM)None	(MM)None	Insert Date Appeal to Prefecture	Add penalty	(MM)None	(MM)None	(MM)None	(MM)None	Send Appeal to Prefecture	(MM)None					
Variant 9 (1508 occurrences)	Create Fine	(MM)None	(MM)None	(MM)None	(MM)None	Send Fine	Insert Fine Notification	(MM)None	(MM)None	(MM)None	Add penalty	(MM)None	(MM)None	Payment	(MM)None	Payment	(MM)None	(MM)None	(MM)None	(MM)None	

Using the Alpha Miner algorithm, we discovered the process model and evaluated its fitness against the event logs.

Fitness for Original Event Log:

```python
# Discover a process model using the Alpha Miner algorithm
net, initial_marking, final_marking =
pm4py.algo.discovery.alpha.algorithm.apply(original_event_log)
# Calculate the fitness of the event log against the discovered process model
fitness =
pm4py.algo.evaluation.replay_fitness.algorithm.apply(original_event_log, net,
initial_marking, final_marking)

# Print the results
print(f"average_trace_fitness: {fitness['average_trace_fitness']}")
print(f"log fitness: {fitness['log_fitness']}")
```

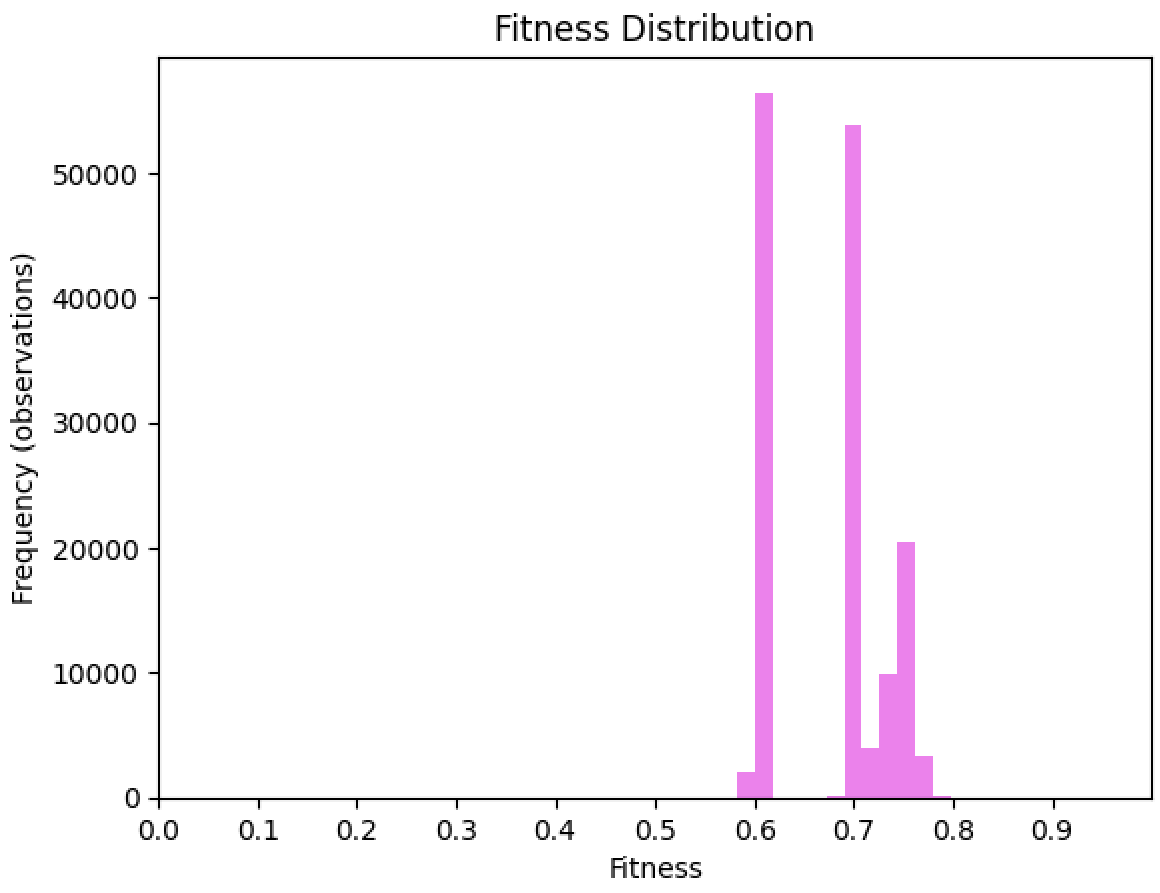
- **Average Trace Fitness**: 0.6741431381488104
- **Log Fitness**: 0.637766272135165

These scores are not so emotional as we wished, but it can be good if we refined the event log on which we compute the theoretical model.

Then I used the following code to extract an histogram with token based replay technique:

```python
net, im, fm = pm4py.discover_petri_net_alpha(original_event_log)
diagnostics =
pm4py.conformance_diagnostics_token_based_replay(original_event_log, net, im,
fm, return_diagnostics_dataframe=True)
frequency =
diagnostics.groupby('trace_fitness').size().reset_index(name='count')

# Define histogram
plt.hist(diagnostics['trace_fitness'], bins=20, color='violet')
plt.xlabel('Fitness')
plt.ylabel('Frequency (observations)')
plt.title('Fitness Distribution')
plt.xlim(0, 1)
plt.xticks(np.arange(0, 1, 0.1))
plt.show()

# Calculate mean, median, and standard deviation of fitness
mean_difference = diagnostics['trace_fitness'].mean()
print(f"Mean of the fitness values: {mean_difference}")

median_difference = diagnostics['trace_fitness'].median()
print(f"Median of the fitness values: {median_difference}")

std_deviation = np.std(diagnostics['trace_fitness'], ddof=1)
print("Sample Standard Deviation of fitness values:", std_deviation)
```

That generated:



- **Mean of the fitness values**: 0.6741431381487306
- **Median of the fitness values**: 0.7
- **Sample Standard Deviation of fitness values**: 0.05799257077111415

Fitness for Filtered Event Log:

```python
# Discover a process model using the Alpha Miner algorithm
net, initial_marking, final_marking =
pm4py.algo.discovery.alpha.algorithm.apply(filtered_log)
# Calculate the fitness of the event log against the discovered process model
fitness = pm4py.algo.evaluation.replay_fitness.algorithm.apply(filtered_log,
net, initial_marking, final_marking)

# Print the results
print(f"average_trace_fitness: {fitness['average_trace_fitness']}")
print(f"log fitness: {fitness['log_fitness']}")
```

- **Average Trace Fitness**: 0.8145291945051658
- **Log Fitness**: 0.8009576224898953

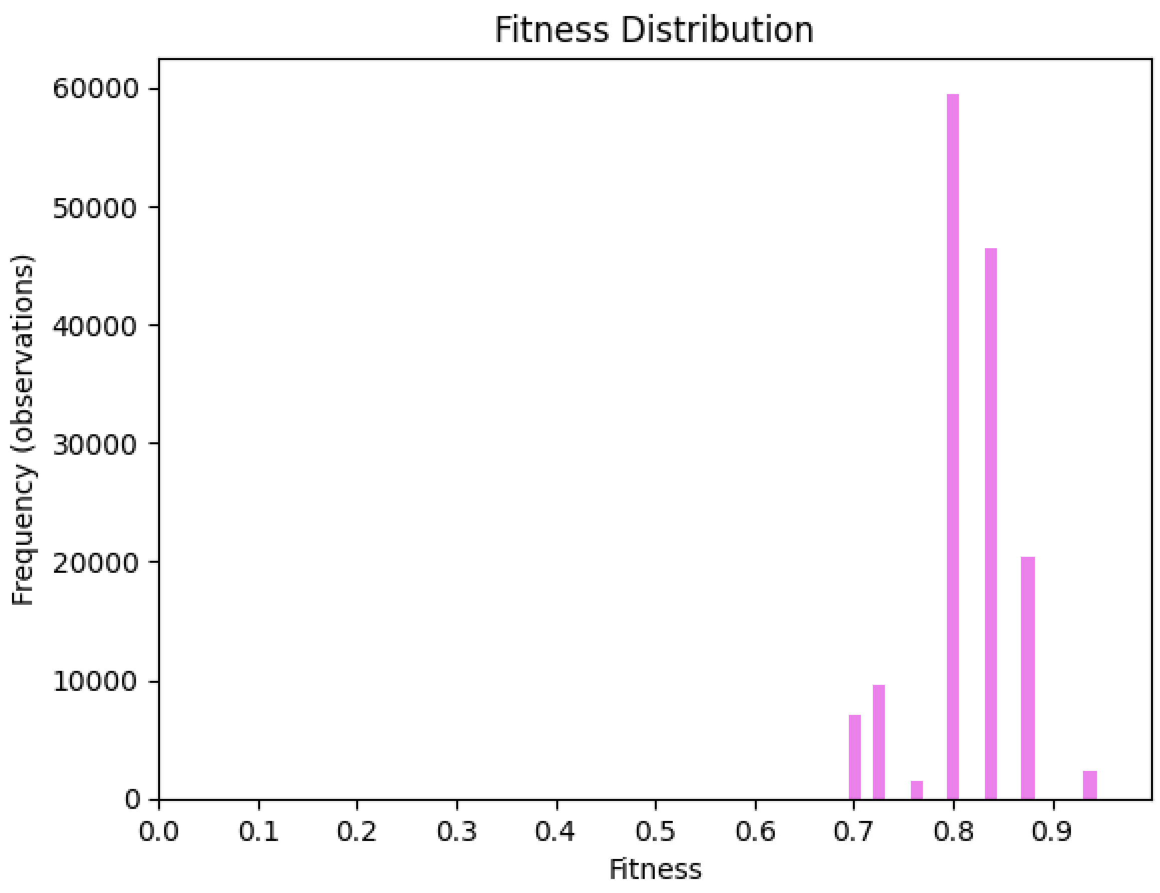
These values represent a good result for a conformity analysis, the event logs seem generally compliant with the theoretical model represented by the *alpha* algorithm.

Then I used the following code to extract an histogram with token based replay technique:

```python
net, im, fm = pm4py.discover_petri_net_alpha(filtered_log)
diagnostics = pm4py.conformance_diagnostics_token_based_replay(filtered_log,
net, im, fm, return_diagnostics_dataframe=True)
```

```python
frequency =
diagnostics.groupby('trace_fitness').size().reset_index(name='count')

# Define histogram
plt.hist(diagnostics['trace_fitness'], bins=20, color='violet')
plt.xlabel('Fitness')
plt.ylabel('Frequency (observations)')
plt.title('Fitness Distribution')
plt.xlim(0, 1)
plt.xticks(np.arange(0, 1, 0.1))
plt.show()

# Calculate mean, median, and standard deviation of fitness
mean_difference = diagnostics['trace_fitness'].mean()
print(f"Mean of the fitness values: {mean_difference}")

median_difference = diagnostics['trace_fitness'].median()
print(f"Median of the fitness values: {median_difference}")

std_deviation = np.std(diagnostics['trace_fitness'], ddof=1)
print("Sample Standard Deviation of fitness values:", std_deviation)
```

That generated:



- **Mean of the fitness values**: 0.8145291945066301
- **Median of the fitness values**: 0.8035714285714286
- **Sample Standard Deviation of fitness values**: 0.0461174205750374 34

Organizational goals:

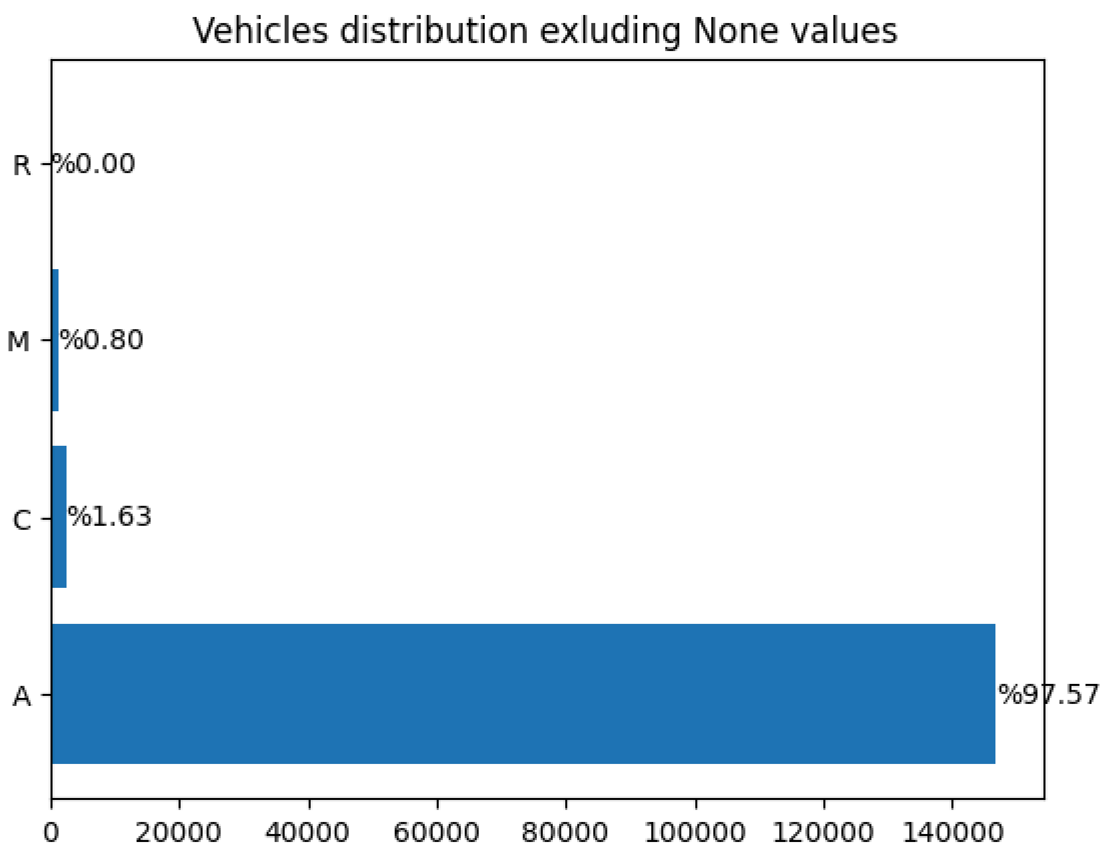
Organizational Goals

The goals are divided into three layers: Strategical, Operational, and Tactical.

- **Strategical**: *Reduce shipping time* for fines, *inefficiencies*, and *costs*.
- **Operational**: *Increase process compliance* and *identify bottlenecks*.
- **Tactical**: Utilize process mining techniques for flow analysis and real-time monitoring, *aiming to optimize the entire process*.

Further Improvement:

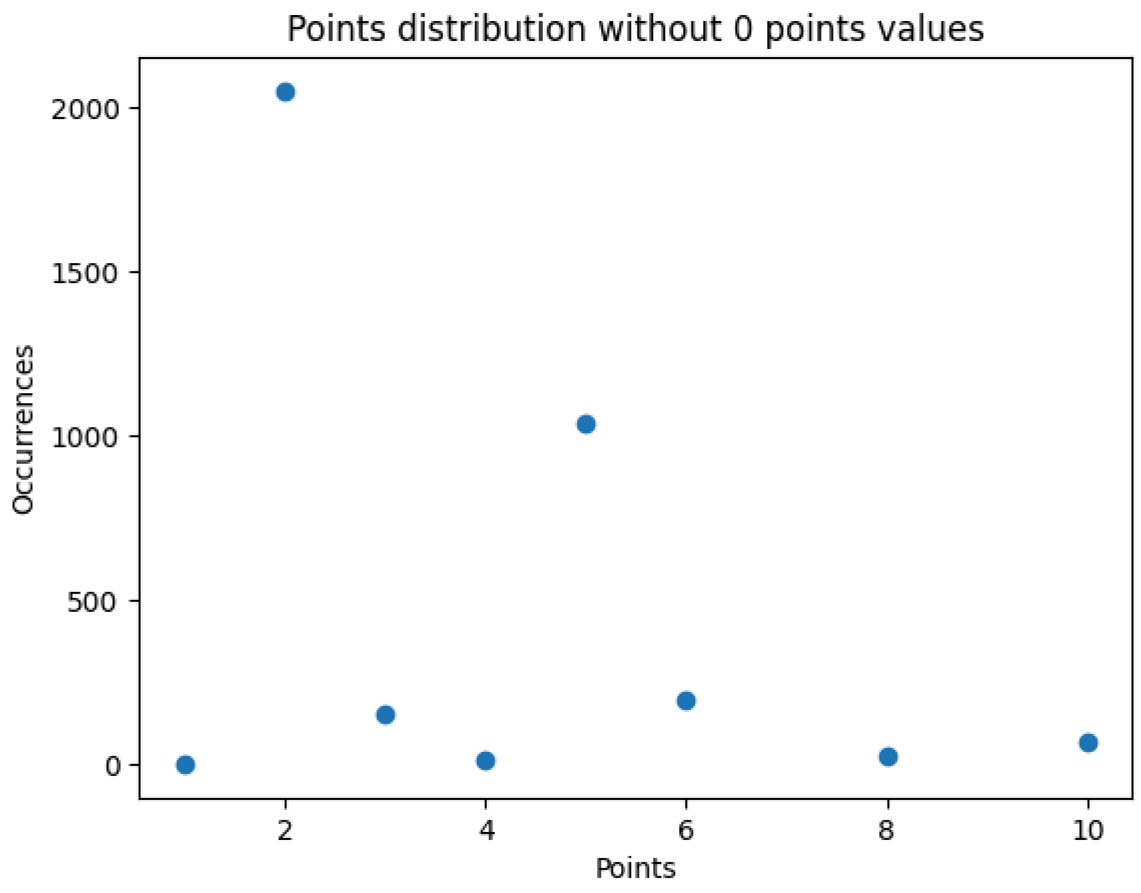
To improve this analysis we can include an analysis based on the type of vehicle:



But we must consider to extend the classification because at the moment only the type `A` can provide meaningful information.

Also, we can classify the Fines for geographic area (cities, rural areas, peripheral areas, etc) and for gender of the offender and so on.

We can also try to identify patterns to reduce the variants number or try to categorize the type of Fines by better analyzing the Points:



Conclusion:

This comprehensive analysis of the road traffic fines management process highlights several areas for improvement. By addressing identified bottlenecks and optimizing the process flow, significant efficiency gains can be achieved. The next steps involve refining the event logs and continuing the analysis to develop a more streamlined and effective process model.