I. Introduction

This project use process mining techniques to study business processes for optimizing the management of road traffic fines and establishing new targets. We are going in first place to involve examining event logs from the police to discern the statistical characteristics that are different between paid and unpaid fines. The main goal is to identify the current process from these logs, determine an ideal process, and then assess the gaps between the interpolated process and the unpaid fines segment. Finally, operational objectives will be defined, we will also talk about some problems we may have and how we can fix them.

II. Context

The Italian police have developed an information system to manage and handle road traffic fines issued by local police forces. This system records detailed data for each fine, resulting in an event log with information on over 140,000 fines. The central office intends to use this log to analyze the process and identify ways to increase the likelihood of fine payment completion, to reduce the management cost and to detect process issues early on.

The process begins with the creation of a fine, recording variables such as the fine amount (A), points deducted from the driver's license (PO), payment status (P), and dismissal status (D). Dismissal status indicates whether the fine must be paid or the reasons for its dismissal.

Key steps in the process include the fine creation, it records the fine amount, points deduction, payment status, and dismissal status. Offenders can pay the fine in full or partially at various stages, including immediately after creation, upon receiving a notification, or after notification receipt. If a notification is sent, additional postal expenses are incurred. If the fine is not paid within 180 days, a penalty equal to the fine amount is added. Offenders can appeal the fine, which, if successful, leads to the fine's dismissal (with specific dismissal values recorded). If the appeal fails, the process continues with further activities. If payment is not made, the case may be sent for credit collection.

By analyzing the event log, the central office aims to improve process efficiency, reduce costs, and enhance early identification of problematic cases, thereby optimizing the overall management of road traffic fines.

III. Examination of the event logs:

The Knowledge Uplift Trail (KUT) is an approach for transforming raw data into valuable insights. It consists of a sequence of steps designed to enhance understanding of the data's inherent properties. The methodology incorporates the following techniques:

We are going at first to do data cleaning. This will prepare the event log by standardizing data, addressing missing values, and converting categorical variables into numerical formats. It will help our analysis because it will be way easier to compare and use standardized data. Then we do the data filtering. It will establish conditions to filter data, retaining only cases that meet specified criteria, such as cases where the final activity is a payment. We will then use statistical tools to comprehend the distribution and characteristics of the data. To discover, we are going to do Process mining. We will apply process mining techniques to uncover the process. We then do Conformance Checking to detect deviations from the standard sequence of events. We will think about strategies from the insights we got with KUT to create a new model that fits better with the goals of the police.

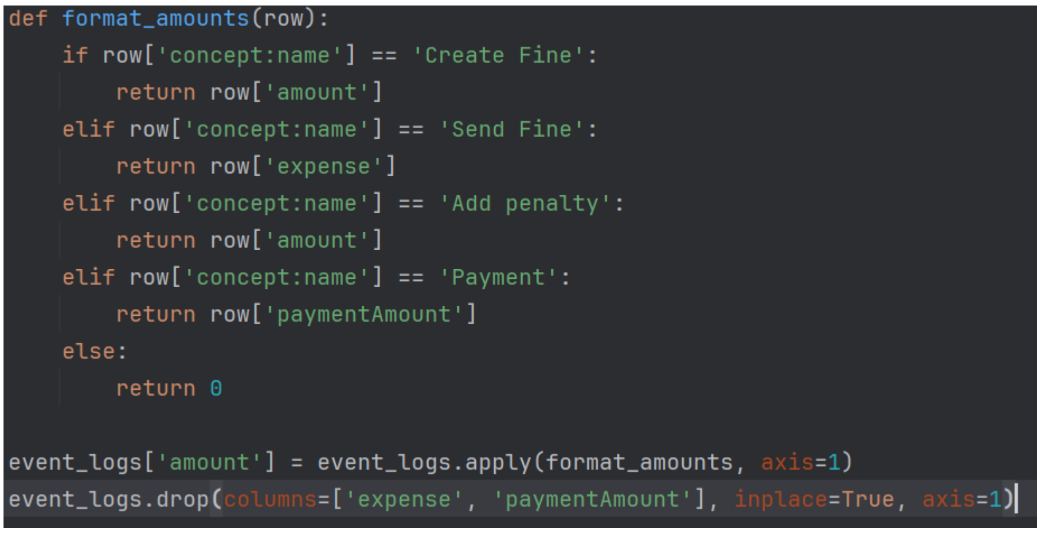
The dataset :

This section outlines the dataset employed in the study, we are going to insist on attributes that capture different aspects of the fine life cycle, from creation to the payment of the fine. Key attributes of the dataset include:

- case:concept:name: Identifies the unique case to which the events belong.
- concept:name: Describes the specific activity related to processing a fine, such as "Create Fine," "Send Fine," or "Payment."
- time:timestamp: Records the exact date and time each event occurred.
- 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 related to the event "Send Fine."
- paymentAmount: Records the amount paid in a "Payment" event.
- dismissal: Indicates whether the fine was dismissed, with NIL meaning the fine was not dismissed, and # or G indicating dismissal by a judge or the prefecture.
- vehicleClass: Specifies the class of the vehicle involved in the fine.
- article: Refers to the regulation under which the fine was issued.

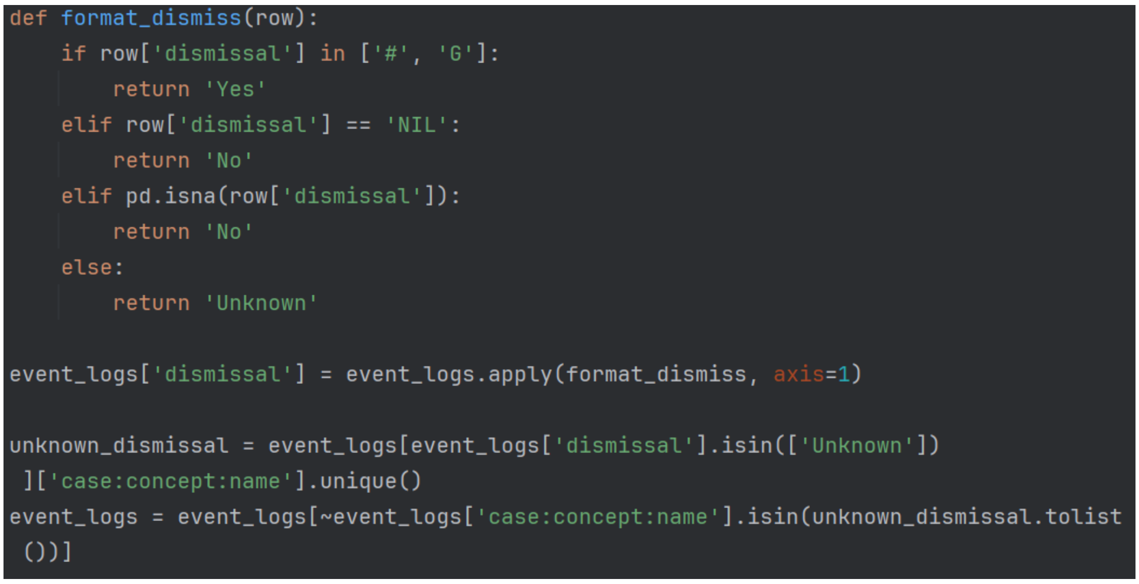
Now that we have specified each steps of the KUT, we are going to have a more detailed look on the implementation of these steps to proceed to the approach we chose.

The Data Cleaning in the code : To simplify the processing of the event logs, we need a function that merges the values into a single column. The format_amounts function is used for it. On the event logs, we will then apply our format to the amounts.



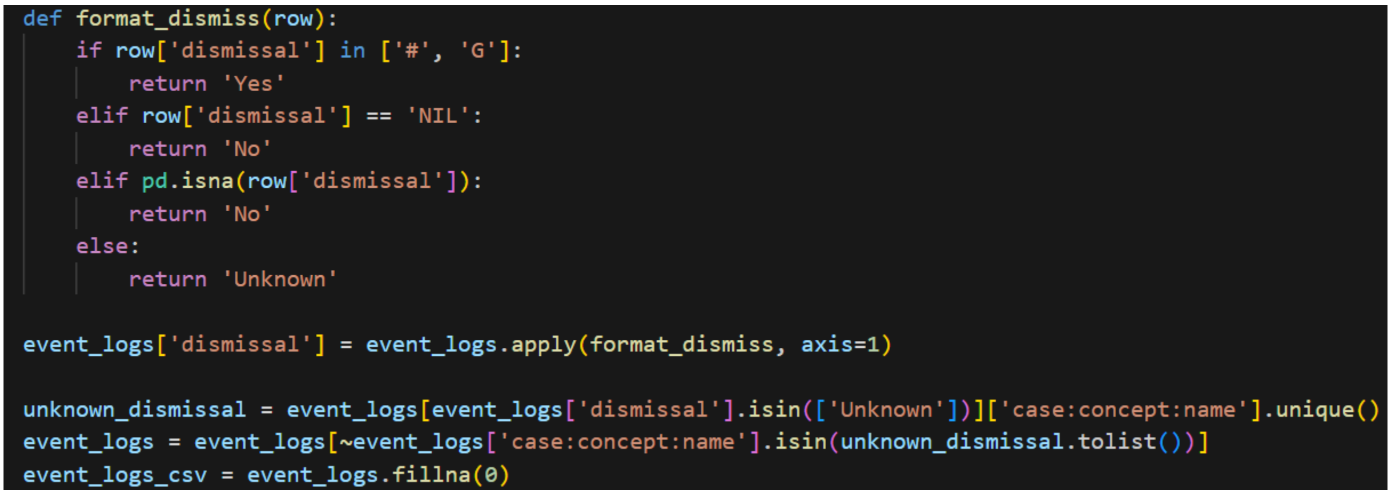
```python
def format_amounts(row):
    if row['concept:name'] == 'Create Fine':
        return row['amount']
    elif row['concept:name'] == 'Send Fine':
        return row['expense']
    elif row['concept:name'] == 'Add penalty':
        return row['amount']
    elif row['concept:name'] == 'Payment':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

event_logs['amount'] = event_logs.apply(format_amounts, axis=1)
event_logs.drop(columns=['expense', 'paymentAmount'], inplace=True, axis=1)
```

To enhance the clarity of the data, we need to standardize the values in the dismissal column to facilitate interpretation. The format_dismiss function applies specific criteria to achieve this standardization, operating as follows:



```
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)

unknown_dismissal = event_logs[event_logs['dismissal'].isin(['Unknown'])
 ]['case:concept:name'].unique()
event_logs = event_logs[~event_logs['case:concept:name'].isin(unknown_dismissal.tolist
 ())]
```

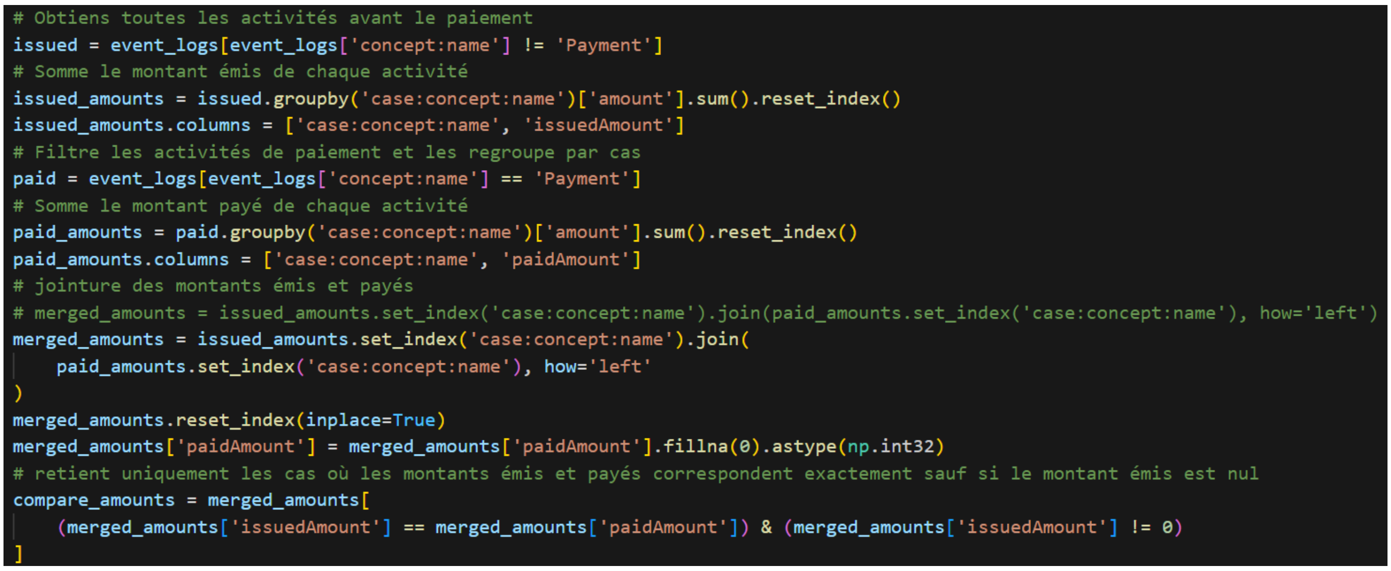
If the dismissal value is # or G, the function returns "Yes," indicating that the fine was dismissed. If the dismissal value is NIL or NaN (missing), the function returns "No," indicating that the fine was not dismissed. For all other values, the function returns "Unknown," as these values do not clearly indicate the dismissal status.



```
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)

unknown_dismissal = event_logs[event_logs['dismissal'].isin(['Unknown'])]['case:concept:name'].unique()
event_logs = event_logs[~event_logs['case:concept:name'].isin(unknown_dismissal.tolist())]
event_logs_csv = event_logs.fillna(0)
```

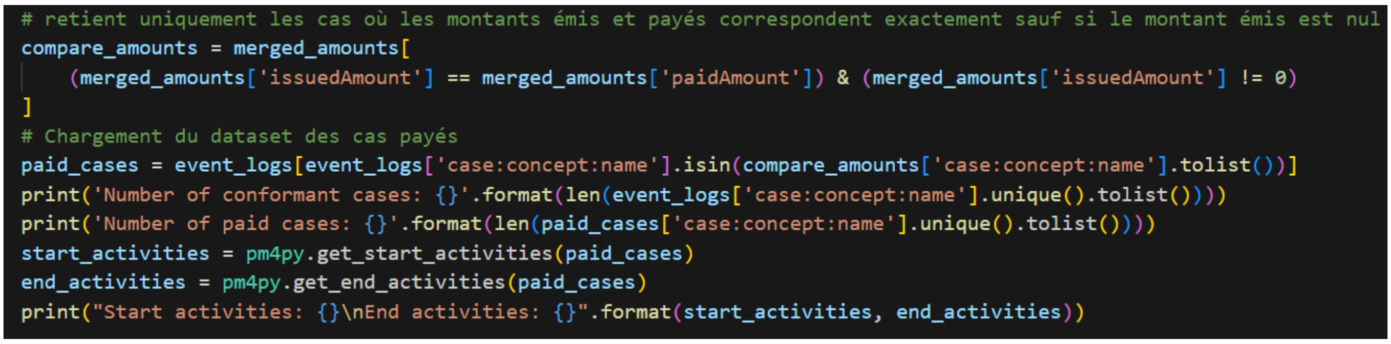
To uphold the dataset's relevance, it's vital to exclude cases labeled with an Unknown dismissal code, ensuring only those with clearly defined statuses are analyzed. When an attribute lacks a value, it's crucial to insert a placeholder for completeness. Additionally, filtering out cases where the duration is zero or less helps maintain the focus on meaningful data for analysis.

About the data filtering, we exclude cases where the last action isn't a payment to examine successful and compliant process executions.



```python
# Obtiens toutes les activités avant le paiement
issued = event_logs[event_logs['concept:name'] != 'Payment']
# Somme le montant émis de chaque activité
issued_amounts = issued.groupby('case:concept:name')['amount'].sum().reset_index()
issued_amounts.columns = ['case:concept:name', 'issuedAmount']
# Filtre les activités de paiement et les regroupe par cas
paid = event_logs[event_logs['concept:name'] == 'Payment']
# Somme le montant payé de chaque activité
paid_amounts = paid.groupby('case:concept:name')['amount'].sum().reset_index()
paid_amounts.columns = ['case:concept:name', 'paidAmount']
# jointure des montants émis et payés
# merged_amounts = issued_amounts.set_index('case:concept:name').join(paid_amounts.set_index('case:concept:name'), how='left')
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)
merged_amounts['paidAmount'] = merged_amounts['paidAmount'].fillna(0).astype(np.int32)
# retient uniquement les cas où les montants émis et payés correspondent exactement sauf si le montant émis est nul
compare_amounts = merged_amounts[
    (merged_amounts['issuedAmount'] == merged_amounts['paidAmount']) & (merged_amounts['issuedAmount'] != 0)
]
```

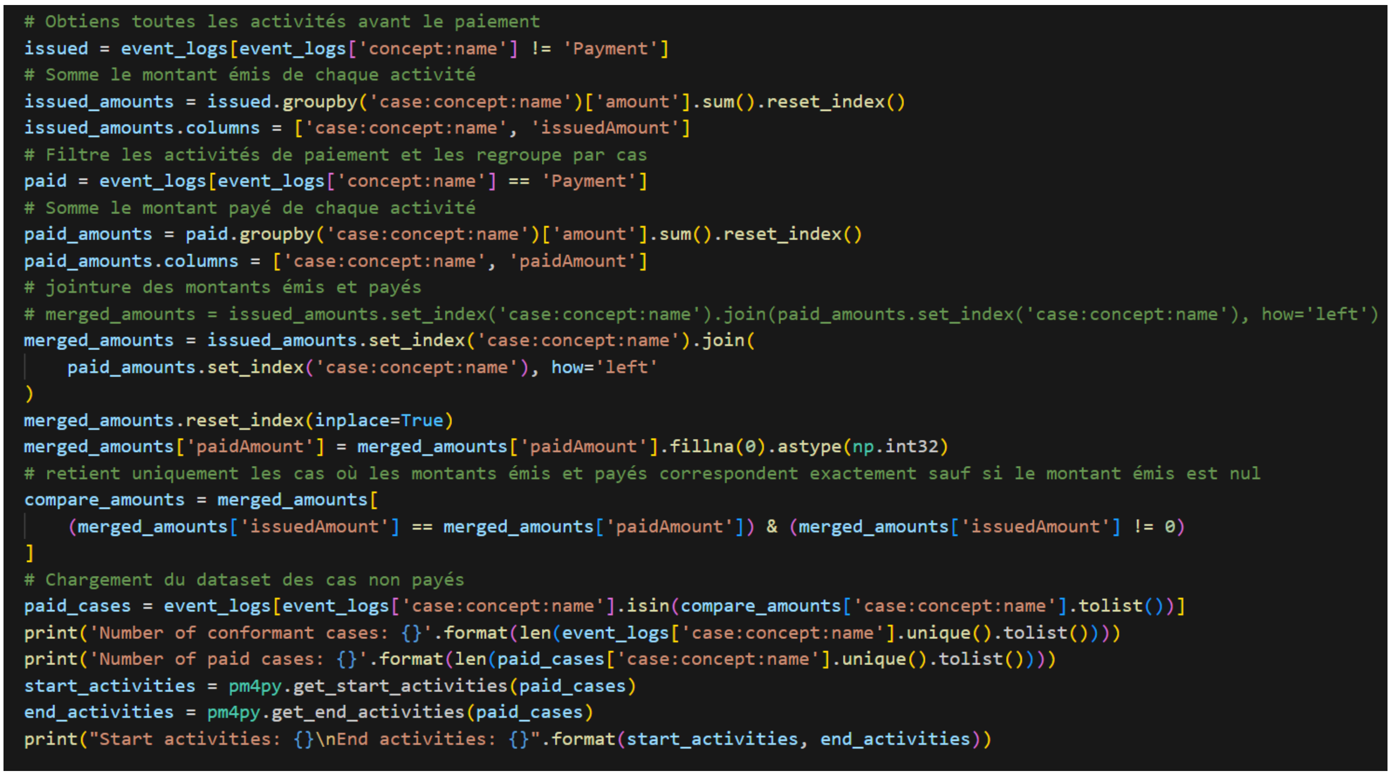
We then identify instances where the issued amount matches the paid amount, disregarding cases where both amounts are zero.



```python
# retient uniquement les cas où les montants émis et payés correspondent exactement sauf si le montant émis est nul
compare_amounts = merged_amounts[
    (merged_amounts['issuedAmount'] == merged_amounts['paidAmount']) & (merged_amounts['issuedAmount'] != 0)
]
# Chargement du dataset des cas payés
paid_cases = event_logs[event_logs['case:concept:name'].isin(compare_amounts['case:concept:name'].tolist())]
print('Number of conformant cases: {}'.format(len(event_logs['case:concept:name'].unique().tolist())))
print('Number of paid cases: {}'.format(len(paid_cases['case:concept:name'].unique().tolist())))
start_activities = pm4py.get_start_activities(paid_cases)
end_activities = pm4py.get_end_activities(paid_cases)
print("Start activities: {}\nEnd activities: {}".format(start_activities, end_activities))
```

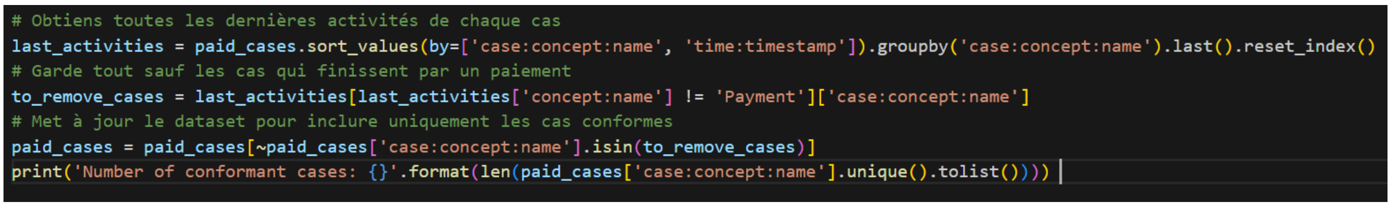
The unpaid fines case :

To concentrate on cases where fines have been issued but remain unpaid, we need to identify records where the issued amount is greater than zero and the paid amount is zero.



```python
# Obtiens toutes les activités avant le paiement
issued = event_logs[event_logs['concept:name'] != 'Payment']
# Somme le montant émis de chaque activité
issued_amounts = issued.groupby('case:concept:name')['amount'].sum().reset_index()
issued_amounts.columns = ['case:concept:name', 'issuedAmount']
# Filtre les activités de paiement et les regroupe par cas
paid = event_logs[event_logs['concept:name'] == 'Payment']
# Somme le montant payé de chaque activité
paid_amounts = paid.groupby('case:concept:name')['amount'].sum().reset_index()
paid_amounts.columns = ['case:concept:name', 'paidAmount']
# jointure des montants émis et payés
# merged_amounts = issued_amounts.set_index('case:concept:name').join(paid_amounts.set_index('case:concept:name'), how='left')
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)
merged_amounts['paidAmount'] = merged_amounts['paidAmount'].fillna(0).astype(np.int32)
# retient uniquement les cas où les montants émis et payés correspondent exactement sauf si le montant émis est nul
compare_amounts = merged_amounts[
    (merged_amounts['issuedAmount'] == merged_amounts['paidAmount']) & (merged_amounts['issuedAmount'] != 0)
]
# Chargement du dataset des cas non payés
paid_cases = event_logs[event_logs['case:concept:name'].isin(compare_amounts['case:concept:name'].tolist())]
print('Number of conformant cases: {}'.format(len(event_logs['case:concept:name'].unique().tolist())))
print('Number of paid cases: {}'.format(len(paid_cases['case:concept:name'].unique().tolist())))
start_activities = pm4py.get_start_activities(paid_cases)
end_activities = pm4py.get_end_activities(paid_cases)
print("Start activities: {}\nEnd activities: {}".format(start_activities, end_activities))
```

It is important to focus on cases that have advanced to the credit collection stage, identifying those that have reached the final stage of the process. This is accomplished with the following code:



```python
# Obtiens toutes les dernières activités de chaque cas
last_activities = paid_cases.sort_values(by=['case:concept:name', 'time:timestamp']).groupby('case:concept:name').last().reset_index()
# Garde tout sauf les cas qui finissent par un paiement
to_remove_cases = last_activities[last_activities['concept:name'] != 'Payment']['case:concept:name']
# Met à jour le dataset pour inclure uniquement les cas conformes
paid_cases = paid_cases[~paid_cases['case:concept:name'].isin(to_remove_cases)]
print('Number of conformant cases: {}'.format(len(paid_cases['case:concept:name'].unique().tolist())))
```

This section details the attributes of the dataset after data cleaning. The cleaned dataset comprises the following attributes: case:concept:name is identifying the unique case to which the events belong. concept:name is describing the specific activity related to the execution of an event, such as "Create Fine," "Send Fine," or "Payment." time:timestamp is recording the exact date and time when each event occurred. The amount will consolidate the previous "amount," "expense," and "paymentAmount" columns. The dismissal will indicates whether an event is associated with the dismissal of the fine or not.

The Paid and Unpaid Cases Knowledge uplift trail :

For the paid cases, we have 7 steps in our process, the first step normalizes the amounts. It fills the values that aren't numbers and translate the dismissal codes. The Knowledge it acquired is description and it outputs the event log we gave in input but after cleaning. The second step takes this output to identify the paid cases, it outputs them. The third step this output to filter the data. The descriptive knowledge we acquire from it is the cases with duration that are higher than 0 days and their final activity is the payement. Once the filtering is done, the 4th step benchmarks the payment timelines by averaging the shipping time from the creation of the fine. From this, we get the parameters of the data distribution. These parameters helps us to draw plots and to have indications on the data distribution. Now that the descriptive analysis is done, we need to use the results. These indications are used by the next step which will give us confidence intervals on the average shipping time from the creation of the fine. We obtain from it the statistical analysis. This will give us statistical confidence that the data we observe is representative of the performance. Then we do the process mining from the filtered paid cases that last activity is payement. The process discovery will gives us Petri nets that are describing the process of the most frequent variants (we will ignore the variants that occurs the less. With these petri nets, we do the process optimization and we also simplify it. This prescriptive knowledge will design a more simplified process that will allow us to have a better understanding of the correct sequences.

For the unpaid cases, the process is much the same. We normalize the amounts of the event logs, we fill the incorrect values and translate the dismissal codes. The resulting cleaned event logs is then filtered by the identification of the unpaid cases (i.e. the cases where the amount is bigger than zero but the paid amount is equal to zero). The unpaid cases are then filtered so we can have only the unpaid cases where the last activity is the payement. We then average the shipping time since creation to have to data distribution parameters. These will give us plots of the data distribution we are going to use of a statistical approach where we obtain confidence intervals on the average shipping time. This statistical approach gives confidence that the data we observe is representative of the performance. The process mining is then applied so we obtain a process discovery approach that will create petri nets which describe the most frequents variants in our process. We these petri nets, we can begin to optimize the process and simplify the process by designing a more simplified version to have a better approach on the correct sequences of the activities.

We can know do Conformance Checking :

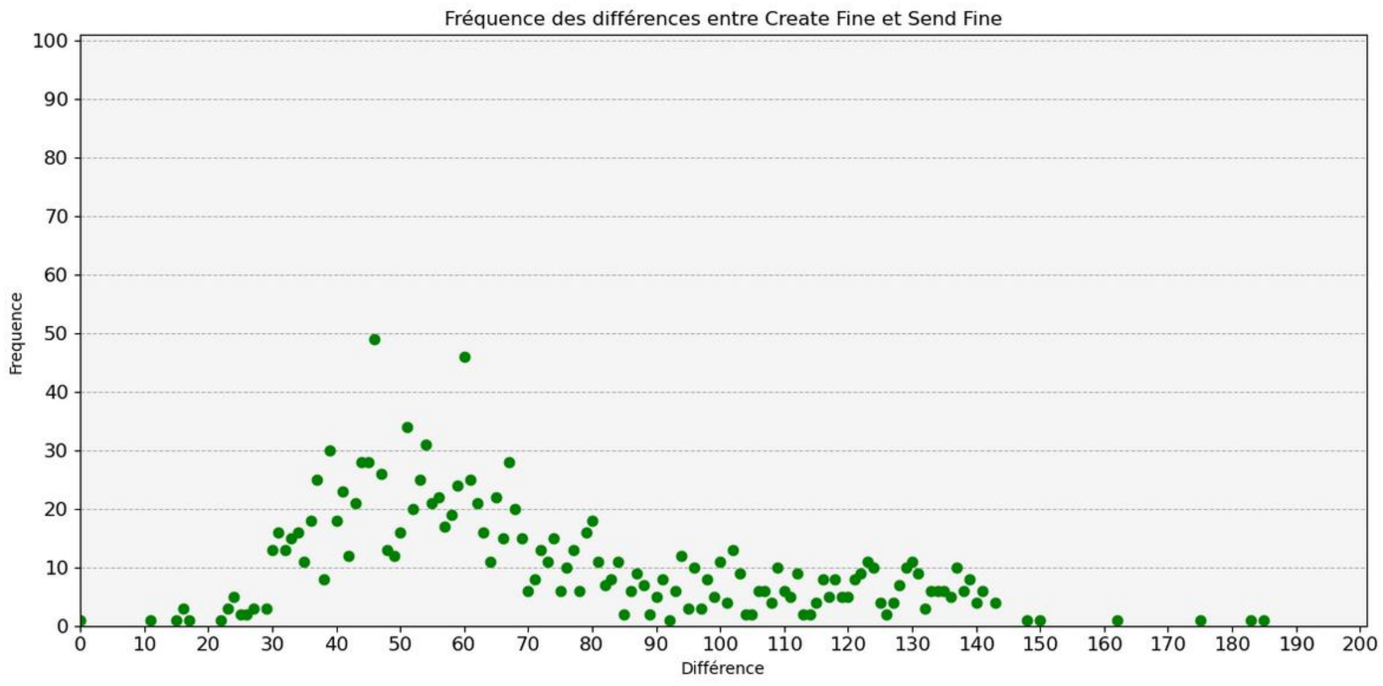
Our conformance checking is pretty simple. Since we obtain petri nets in the previous processes, we can construct a simplified model with a prescriptive approach and design. This simpler model will represent the process. This allows us to get specifications on the optimized version of the process. These specifications are not useless, because we are going to use them to calculate the standard deviation of unpaid cases from the objective process, to achieve that we also need to take the event logs in account. This is where conformance checking is achieved because that approach will allow us to obtain fitness on our process. This fitness will be important. The prescriptive analysis using the fitness is giving us the main goal we want to achieve by designing and mining : get possible improvements and implementations on our process to optimize and simplify. We then obtain conformance to the process and identification of the possible weaknesses of our process.

We also do an analysis on Data Segments :

We want to run descriptive statistics on the distribution of data for paid and unpaid cases of the event logs. The goal is to extract the timing reference level KPI that represents the process.

For the paid cases, we want to asses the objective, we are gonna assume that the time frame from the creation of the fine to its shipping is an important factor to determine the success of a given case. To check this assumption, we will measure the key performance indicators (that we are gonna call KPIs) from the paid cases to establish benchmarks. Then, we will compare these benchmarks to the timeline of unpaid fines to show that update fines have a longer time frame. We will then support the business hypothesis that reducing the send time can increase the success rate of fine payements.

In the following chart, we are going to study analytically the frequency of the difference between the Create Fine and the Send Fine activities. To have better understanding on these frequencies, we are also going to compute the mean, the median and the standard deviation of the frequencies of the differences. The difference between Create Find and Send Fine represents the difference between the two time frames of the activities.



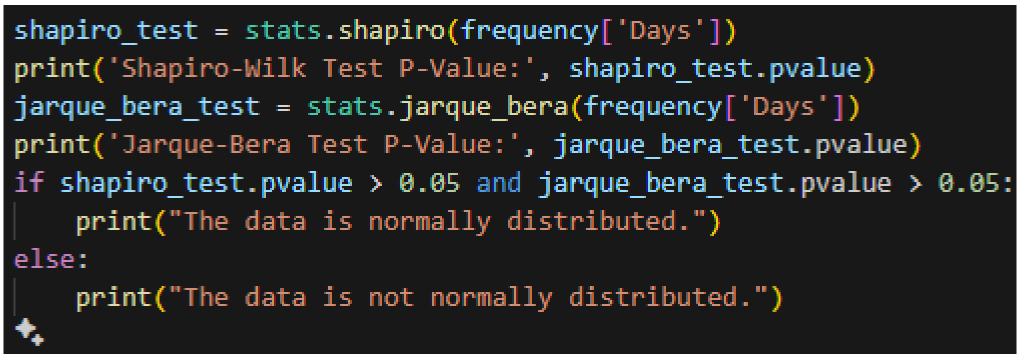
```
Médiane: 61.0
Moyenne: 69.91117478510029
Ecart-type: 31.142049224053515
```

The central tendency analysis reveals that half of the fines are dispatched within a timeframe of 60-70 days, although there is considerable variability, evidenced by a standard deviation of 31 days. To evaluate the normality of the data distribution, statistical tests such as the Shapiro-Wilk and Jarque-

Bera tests were conducted. Both tests produced p-values exceeding 0.05, which indicates that the data can be considered normally distributed.

Furthermore, the observation that the mean is higher than the median suggests a slight positive skew in the data. However, this skewness is not pronounced enough to undermine the assumption of normality. This finding supports the appropriateness of using parametric statistical analyses for this dataset.

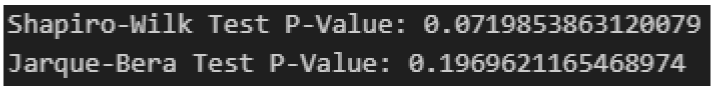
The significant variability in shipping times suggests that while some fines are dispatched promptly, others experience considerable delays. This insight is critical for identifying potential inefficiencies or bottlenecks within the shipping process. Understanding these delays can help in devising strategies to improve the overall efficiency and reliability of the system. By pinpointing where the delays occur, steps can be taken to streamline operations and reduce the time taken to ship fines, thereby enhancing the overall process and ensuring more consistent shipping times. The Shapiro-Wilk and Jarque-Bera tests are statistical methods employed to determine whether a dataset follows a normal distribution. Specifically, the Jarque-Bera test operates by calculating the skewness and kurtosis of the dataset, which are metrics that describe the shape of the distribution. These calculated values are then compared to the expected values under a normal distribution. If the dataset significantly deviates from a normal distribution, the Jarque-Bera test will identify and flag this discrepancy.



```python
shapiro_test = stats.shapiro(frequency['Days'])
print('Shapiro-Wilk Test P-Value:', shapiro_test.pvalue)
jarque_bera_test = stats.jarque_bera(frequency['Days'])
print('Jarque-Bera Test P-Value:', jarque_bera_test.pvalue)
if shapiro_test.pvalue > 0.05 and jarque_bera_test.pvalue > 0.05:
    print("The data is normally distributed.")
else:
    print("The data is not normally distributed.")
```

This code snippet first performs the Shapiro-Wilk and Jarque-Bera tests to assess the normality of the frequency['Days'] data. It prints the p-values of these tests and then determines whether the data can be considered normally distributed based on the threshold of 0.05 for both tests. This approach helps in deciding whether parametric statistical analyses are appropriate for the dataset.

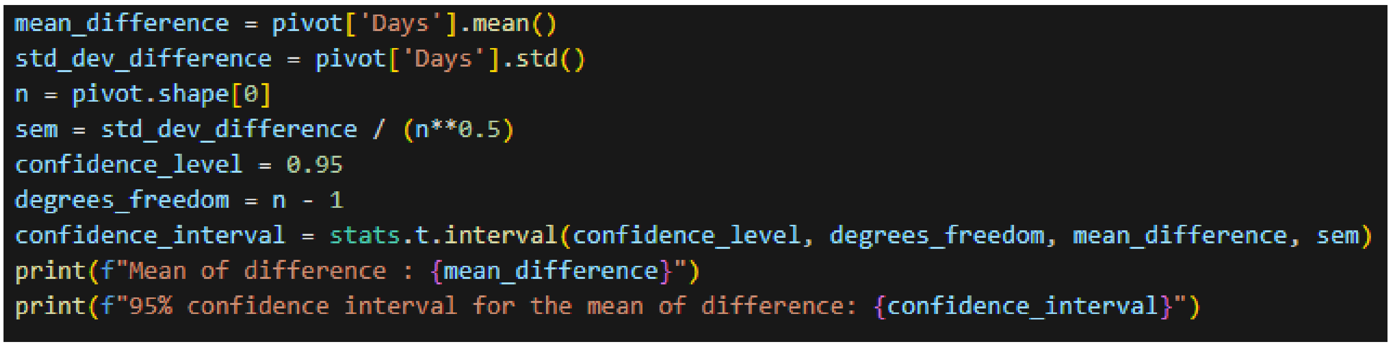
We obtain the following results :



```
Shapiro-Wilk Test P-Value: 0.0719853863120079
Jarque-Bera Test P-Value: 0.1969621165468974
```

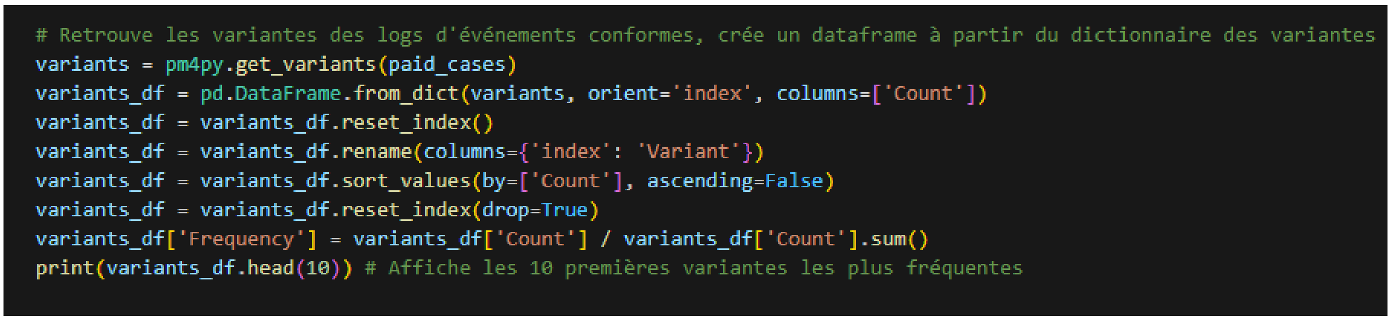
For both tests, we find a P-Value higher than 0.05 so we can find that the data is indeed normally distributed.

Given this assumption, we need to compute and find the normal distribution confidence interval. The confidence level represents the percentage of times an estimate is expected to fall between the upper and lower bounds of the confidence interval, and it is determined by the alpha value. In this context, the focus is on testing the average shipping time from the point of creation.



```python
mean_difference = pivot['Days'].mean()
std_dev_difference = pivot['Days'].std()
n = pivot.shape[0]
sem = std_dev_difference / (n**0.5)
confidence_level = 0.95
degrees_freedom = n - 1
confidence_interval = stats.t.interval(confidence_level, degrees_freedom, mean_difference, sem)
print(f"Mean of difference : {mean_difference}")
print(f"95% confidence interval for the mean of difference: {confidence_interval}")
```

The Process Discovery for the paid cases can now be achieved. The first step would be to find the most frequent variants we observe. To do that, we use the following code :



```python
# Retrouve les variantes des logs d'événements conformes, crée un dataframe à partir du dictionnaire des variantes
variants = pm4py.get_variants(paid_cases)
variants_df = pd.DataFrame.from_dict(variants, orient='index', columns=['Count'])
variants_df = variants_df.reset_index()
variants_df = variants_df.rename(columns={'index': 'Variant'})
variants_df = variants_df.sort_values(by=['Count'], ascending=False)
variants_df = variants_df.reset_index(drop=True)
variants_df['Frequency'] = variants_df['Count'] / variants_df['Count'].sum()
print(variants_df.head(10)) # Affiche les 10 premières variantes les plus fréquentes
```

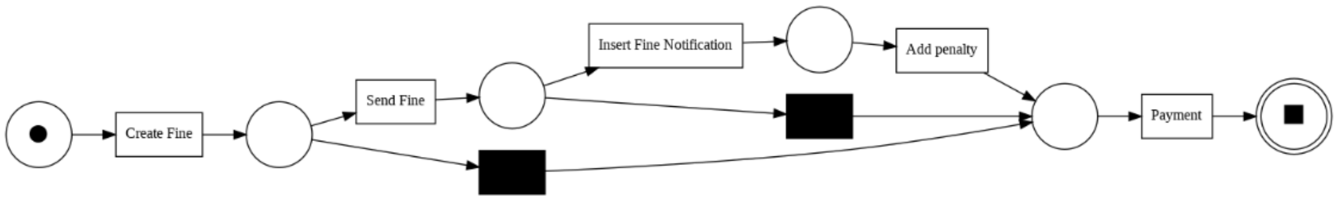
Which give us this table :

```
                                           Variant  Count  Frequency
0                            (Create Fine, Payment)  30436   0.956115
1                 (Create Fine, Send Fine, Payment)   1360   0.042723
2        (Create Fine, Payment, Send Fine, Payment)     18   0.000565
3        (Create Fine, Send Fine, Payment, Payment)     12   0.000377
4  (Create Fine, Send Fine, Insert Fine Notificat...      3   0.000094
5  (Create Fine, Send Fine, Insert Date Appeal to...      2   0.000063
6                   (Create Fine, Payment, Payment)      1   0.000031
7  (Create Fine, Send Fine, Insert Fine Notificat...      1   0.000031
```

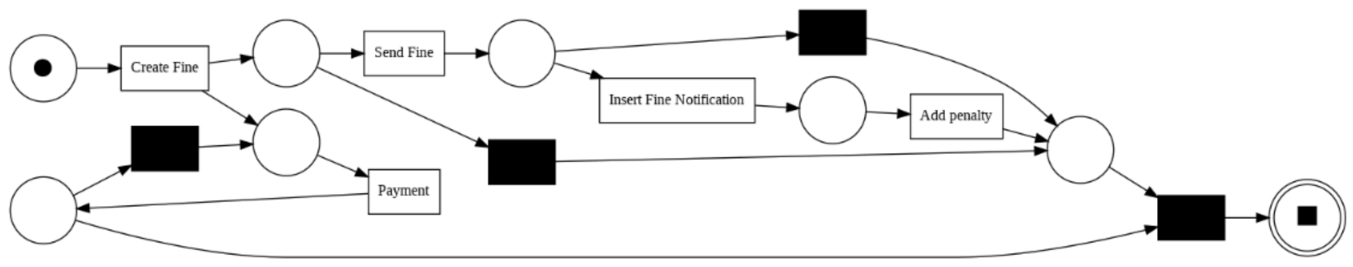
So we can see that one variant is dominating the table with a frequency of 95.6%.

The objective of this section is to provide a comprehensive overview of these variants and illustrate how the activities are interrelated. The following process discovery techniques from the PM4Py library are employed:

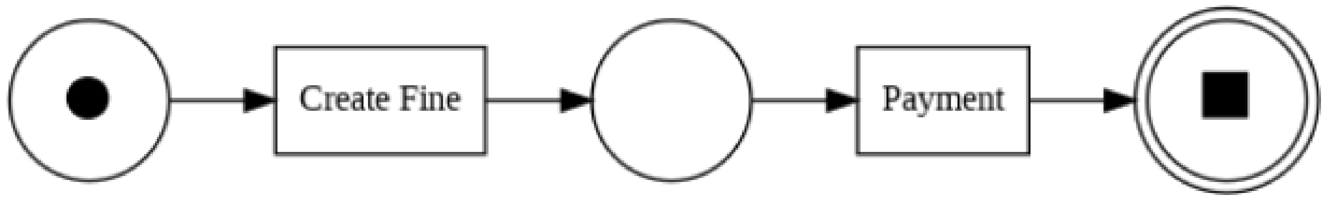
- Alpha Miner Algorithm: This algorithm detects the fundamental ordering relationships between activities and creates a Petri net to represent these relationships.
- Heuristic Miner: This technique enhances the alpha miner by integrating frequency and dependency metrics, allowing it to better handle data noise. It identifies the most frequent and significant patterns within the event log.

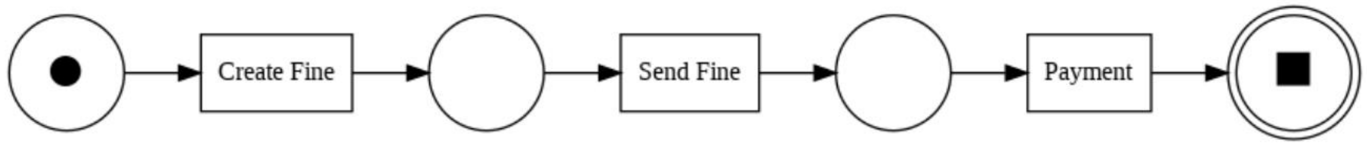


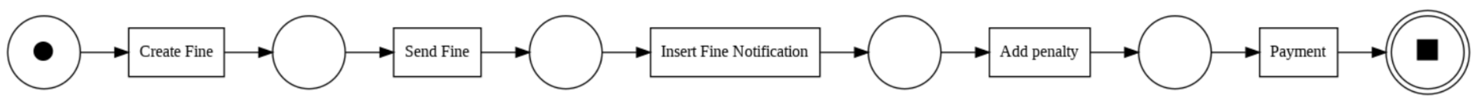
The inductive miner operates by recursively breaking down the event log into smaller segments until straightforward models can be identified, which are then combined to form the final model.

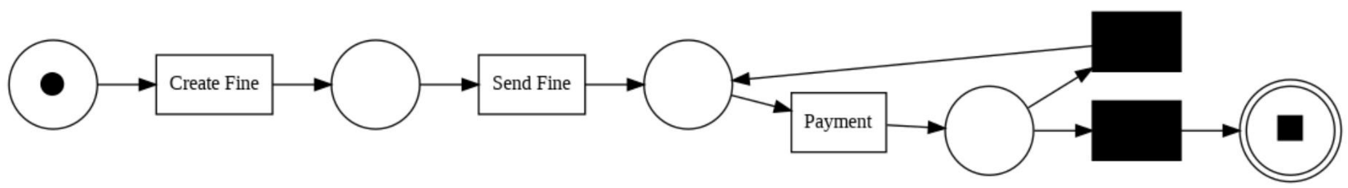


Given the complexity of the previous process representations, simpler and more effective representations of the underlying workflows are considered. By deconstructing the process into manageable segments, we can gain a clearer understanding of the primary mechanisms in the most frequently occurring scenarios.

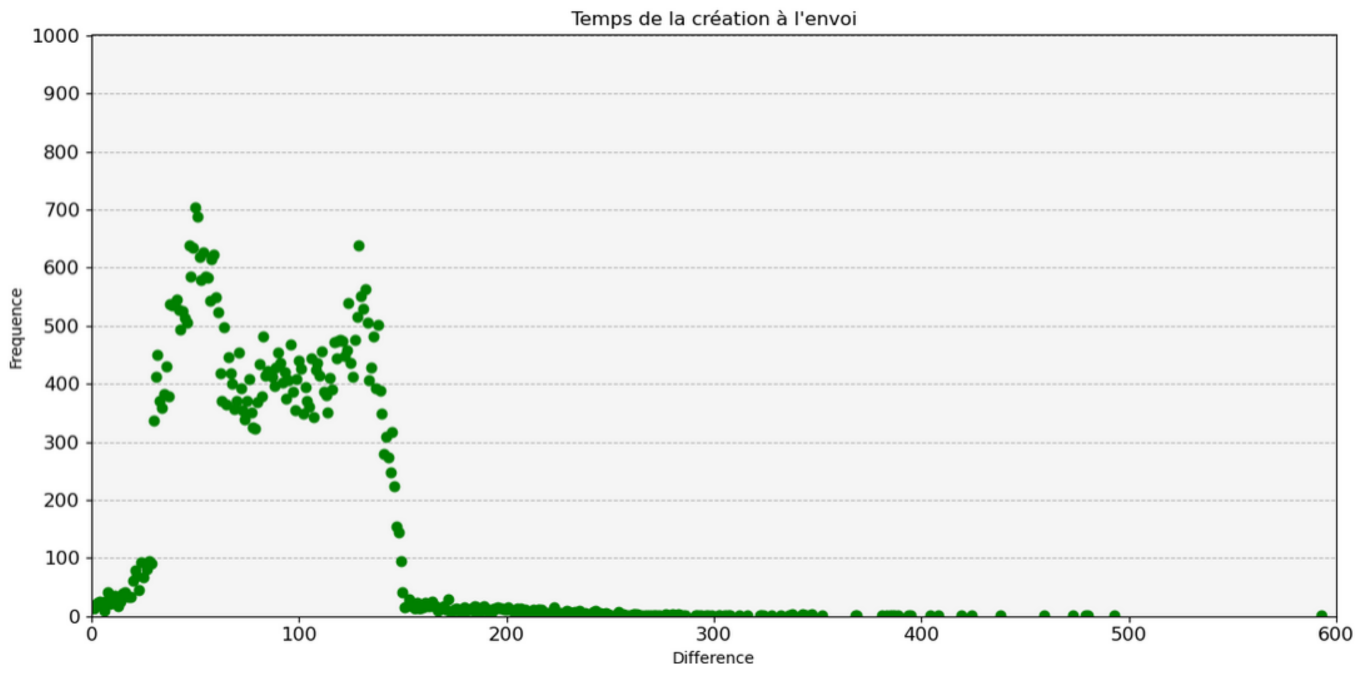






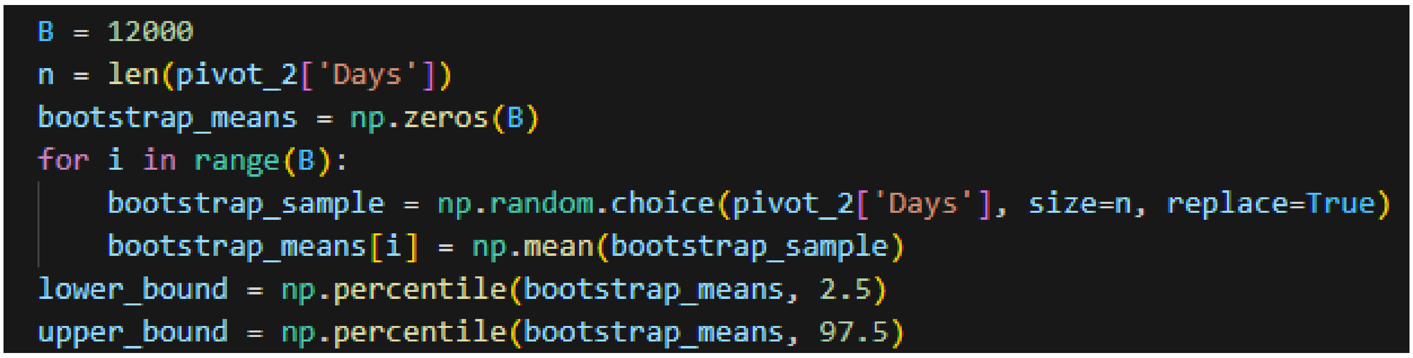


For the unpaid cases, the timeline between creation and the sending of the fine is important to be studied. We examine unpaid cases to identify the characteristics of the variant distribution and validate our primary hypothesis: the shipping timeframe from fine creation is a crucial factor in the success of a case. This analysis aims to show that unpaid fines are associated with longer shipping timeframes, reinforcing the idea that minimizing delays can improve the success rate of fine payments.



The central tendency of this distribution indicates that fines are, on average, shipped within the first 83 days, with a significant variability reflected by a standard deviation of 42 days. This considerable variability in shipping times suggests notable inconsistencies. Statistical tests, including the Shapiro-Wilk and Jarque-Bera tests, were conducted to evaluate the normality of the data distribution. Both tests returned p-values less than 0.05, indicating that the data cannot be considered normally distributed. Since the normality assumption is not met, non-parametric statistical analyses, such as bootstrap methods, should be utilized.

The bootstrap method for the confidence interval involves creating numerous resamples (with replacement) from the original dataset and computing the effect size for each of these resamples. By analyzing these bootstrap resamples, the 95% confidence interval (CI) for the effect size can be determined.

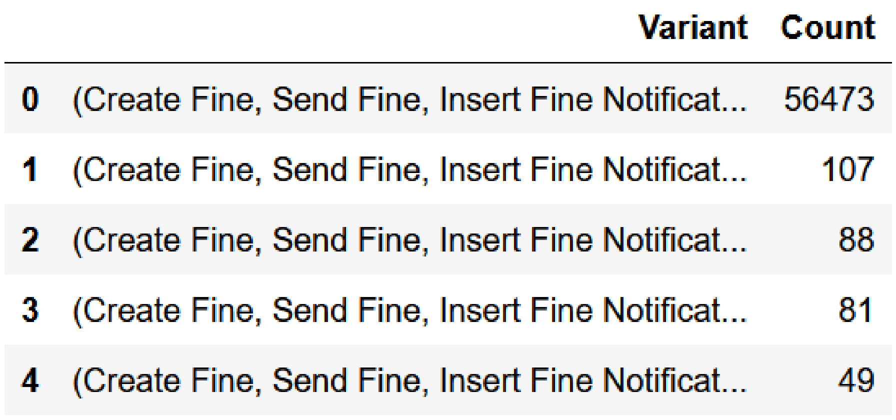


```
B = 12000
n = len(pivot_2['Days'])
bootstrap_means = np.zeros(B)
for i in range(B):
    bootstrap_sample = np.random.choice(pivot_2['Days'], size=n, replace=True)
    bootstrap_means[i] = np.mean(bootstrap_sample)
lower_bound = np.percentile(bootstrap_means, 2.5)
upper_bound = np.percentile(bootstrap_means, 97.5)
```

This code snippet conducts bootstrap resampling to estimate the confidence interval for the mean of a dataset (pivot['Days']). It initializes by setting the number of bootstrap samples (B) to 10,000 and determines the length of the dataset (n). Through a loop, it generates bootstrap samples by randomly sampling with replacement from pivot['Days'], computes the mean for each sample, and stores these means in bootstrap_means. Finally, it calculates the 95% confidence interval bounds (lower_bound and upper_bound) using percentiles of bootstrap_means (2.5th and 97.5th percentiles). This approach allows for robust estimation of uncertainty around the dataset's mean, useful particularly when the data distribution is unknown or non-normal.

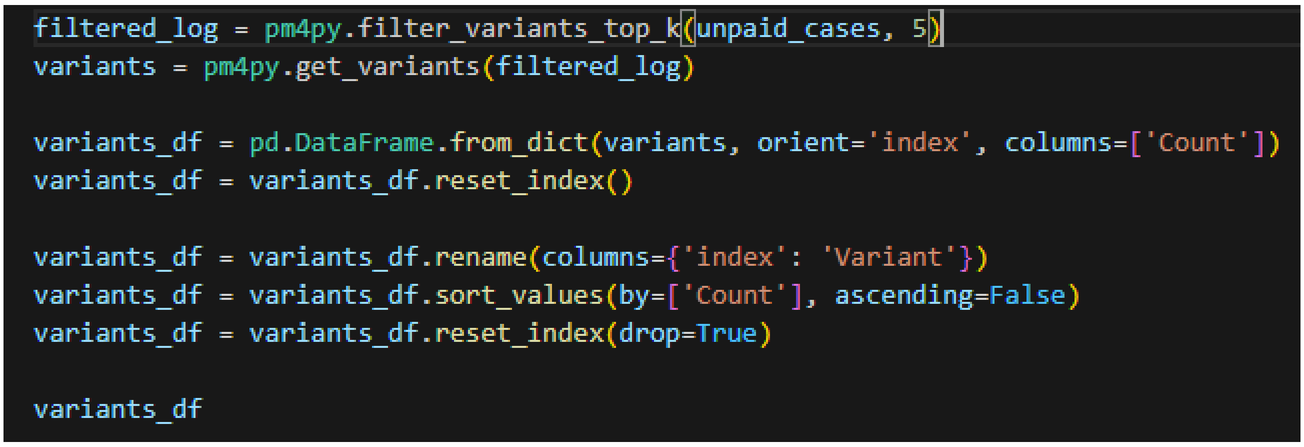
With this method we can find a 95% confidence interval of [15.51;16.37].

Now we can make a Process Discovery for the unpaid cases. We found the first five most frequent variants in the unpaid cases :

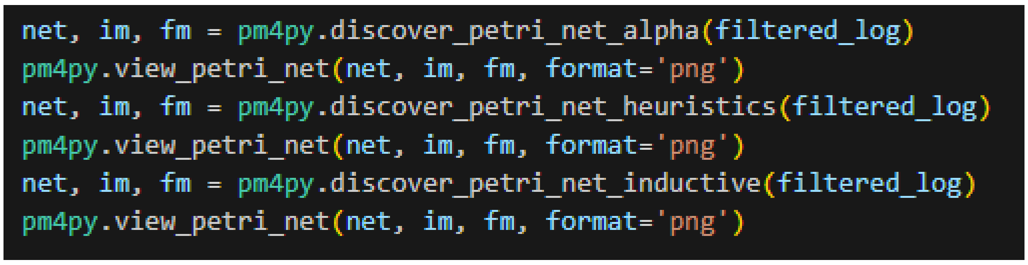


	Variant	Count
0	(Create Fine, Send Fine, Insert Fine Notificat...	56473
1	(Create Fine, Send Fine, Insert Fine Notificat...	107
2	(Create Fine, Send Fine, Insert Fine Notificat...	88
3	(Create Fine, Send Fine, Insert Fine Notificat...	81
4	(Create Fine, Send Fine, Insert Fine Notificat...	49

To understand the most common patterns in unpaid cases, we look at the five most frequent variants. The heuristic miner algorithm is then used to create a clear graphical representation of the process with Petri nets.



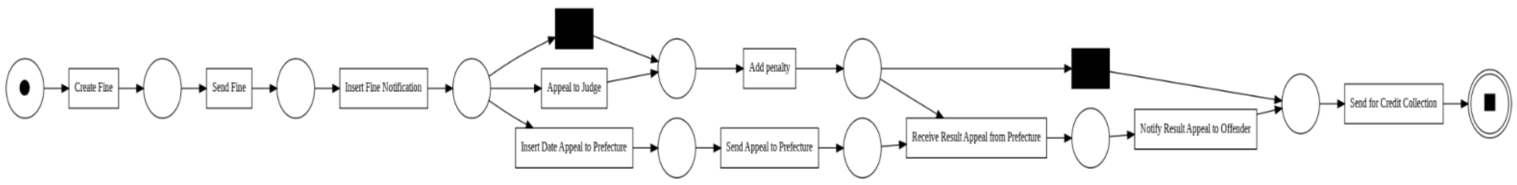
```python
filtered_log = pm4py.filter_variants_top_k(unpaid_cases, 5)
variants = pm4py.get_variants(filtered_log)

variants_df = pd.DataFrame.from_dict(variants, orient='index', columns=['Count'])
variants_df = variants_df.reset_index()

variants_df = variants_df.rename(columns={'index': 'Variant'})
variants_df = variants_df.sort_values(by=['Count'], ascending=False)
variants_df = variants_df.reset_index(drop=True)

variants_df
```



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

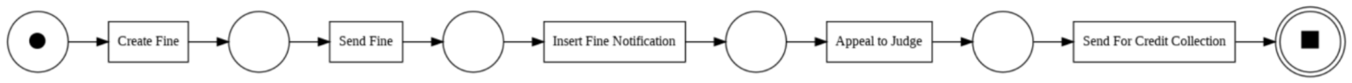
This code snippet begins by filtering unpaid cases to focus on the top 5 most frequent process variants. It then converts these variants into a DataFrame (`variants_df`), organizing them by frequency. Subsequently, it applies three process mining algorithms (`alpha miner`, `heuristic miner`, `inductive miner`) to discover Petri nets that model the workflow patterns within the filtered dataset. Finally, it visualizes each Petri net as a PNG image, providing insights into process execution dynamics and facilitating workflow optimization based on observed patterns.

So we get :

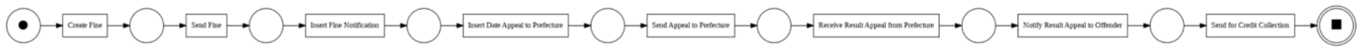

Create Fine
Send Fine
Insert Fine Notification
Appeal to Judge
Insert Date Appeal to Prefecture
Add penalty
Send Appeal to Prefecture
Receive Result Appeal from Prefecture
Notify Result Appeal to Offender
Send for Credit Collection

The problem is that the petri net is too big, so we make two petri nets instead, one to judge and one to prefecture.

To judge :

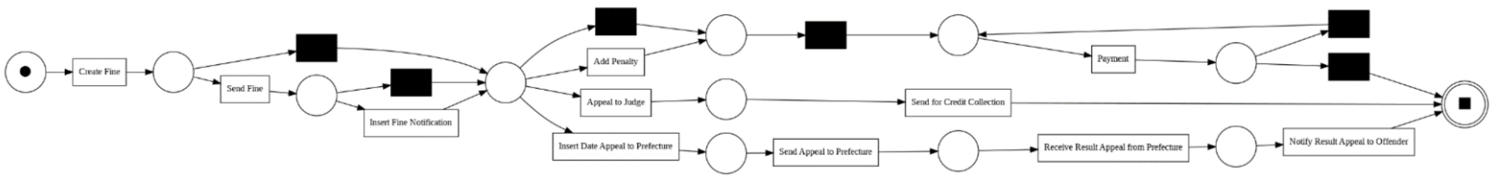


To prefecture :



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

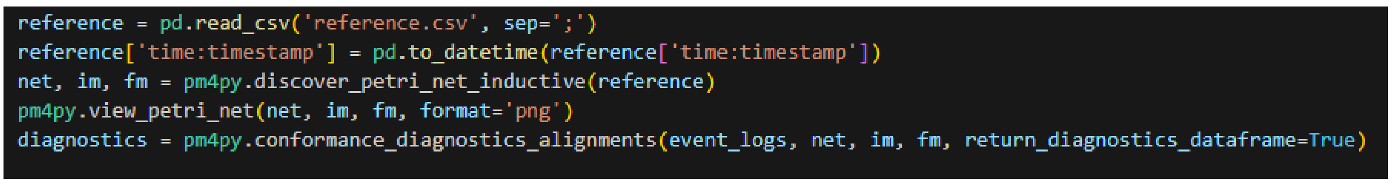
To reduce the variability of the execution instances, an additional requirement can be set: the "Add Penalty" activity should be positioned immediately after the "Insert Fine Notification" activity in the traces. Now that the process discovery is made, we can establish the specifications of our process and do the conformance checking. The inductive miner algorithm has been employed to create a comprehensive and integrated process that incorporates the previous workflows. This model provides a



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

broad and generalized perspective on the various scenarios that may arise during process execution. It is designed to encompass a diverse range of potential situations, offering a robust overview of the entire process.

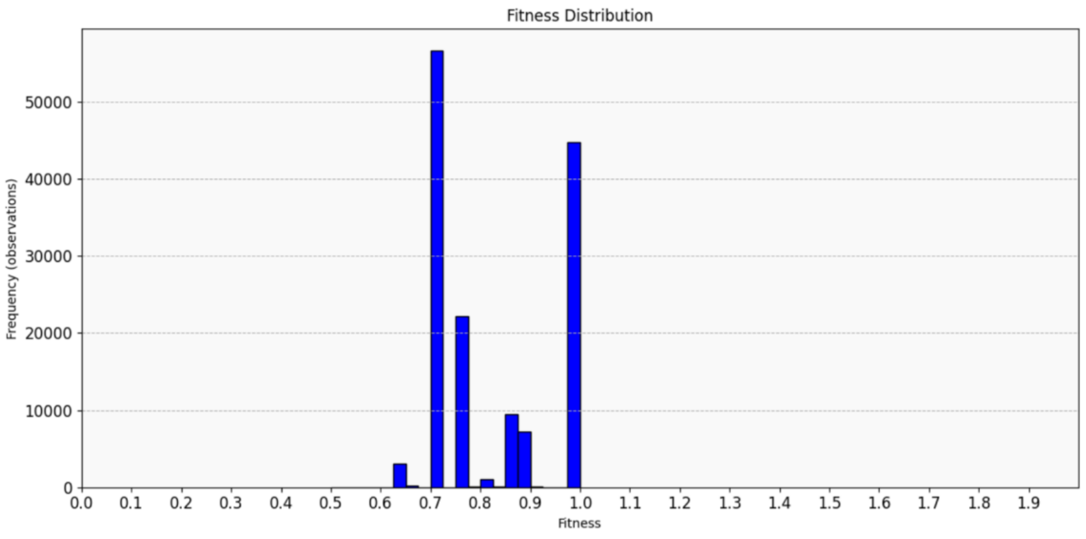
In the following lines of code, all current unpaid fines cases are compared to the prescribed process (TO-BE analysis).



```
reference = pd.read_csv('reference.csv', sep=';')
reference['time:timestamp'] = pd.to_datetime(reference['time:timestamp'])
net, im, fm = pm4py.discover_petri_net_inductive(reference)
pm4py.view_petri_net(net, im, fm, format='png')
diagnostics = pm4py.conformance_diagnostics_alignments(event_logs, net, im, fm, return_diagnostics_dataframe=True)
```

This code snippet is performing process mining tasks using the pm4py library in Python. Initially, it loads event data from a CSV file (reference.csv) into a Pandas DataFrame and converts timestamps to datetime format for consistency. Then, it utilizes pm4py to discover a Petri net structure from this event data. The discovered Petri net is subsequently visualized as a PNG image. Finally, the code conducts conformance checking between the event log data and the Petri net, generating detailed diagnostic information to assess how well the Petri net model aligns with the actual recorded behavior. Overall, the code aims to analyze and visualize process behavior based on event data, providing insights into process conformance and deviations.

This gives us the following chart :



With a general fitness score exceeding 60%, the conformity analysis indicates that the logs generally align with the theoretical model. The primary area for improvement lies in the initial hypothesis: unpaid cases experience longer shipping times. Therefore, defining a strategic objective to reduce shipping times becomes crucial, influencing operational and tactical levels to minimize event variability.

We will now discuss of the organizational goals. The organizational goals diagram outlines a strategy for optimizing the current business process. Strategically, the primary aim is to reduce shipping times, which includes decisions such as halting fines after a specific period to mitigate unnecessary expenses. Operationally, the goals focus on enhancing process compliance, identifying bottlenecks, optimizing resource allocation, and developing predictive models to assess case risk levels. To achieve these objectives, real-time monitoring dashboards utilizing process mining techniques will be deployed to ensure cases are handled appropriately and to flag any cases progressing excessively. Additionally, automated systems will be implemented to send reminders on due dates and emphasize the benefits of timely payments, thereby reducing postal costs. Regular training sessions will also be conducted to empower staff in managing irregular executions effectively. Furthermore, efforts will be made to enhance data collection methods and adapt the information system to extract more detailed insights, supporting the development of predictive models for classifying case risk levels. To enhance this analysis, it is recommended to gather additional information such as the geographic location of fine issuance, offender gender, and the classification of the area (urban, suburban, highways). Integrating this supplementary data would facilitate clustering analyses to uncover patterns and trends within the dataset. Subsequently, this would enable the construction of more precise predictive models that assess the probability of fine payment based on various independent variables. Ultimately, implementing machine learning models would empower staff to prioritize cases with higher risk levels, thereby improving overall operational effectiveness.