Road Traffic Fine Database

Professor:

Paolo Ceravolo

Student:

MohammadHashem DehghaniFirozabadi

33814A

Introduction

This project aims to improve the management of road traffic fines using process mining techniques. We'll start by examining event logs from the police information system to understand the differences between paid and unpaid fines. Our goal is to map the current process (AS-IS), envision an improved future process (TO-BE), and compare it with unpaid cases to find gaps. Finally, we'll set goals and suggest improvements.

The Knowledge Uplift Trail

The Knowledge Uplift Trail (KUT) transforms raw data into valuable insights through several steps:

1. Data Cleaning: Standardizes data, fills in missing values, and converts categories to numbers.

2. Data Filtering: Filters data to include only specific cases, such as those ending in payment.

3. Descriptive Analysis: Uses statistics to understand data distribution and characteristics.

4. Process Mining: Applies techniques to uncover the process.

5. Conformance Checking: Identifies deviations from the usual event sequence.

6. Intervention Strategies: Develops strategies based on insights and improves them through monitoring.

Dataset

This section describes the dataset used, detailing attributes that track a fine's life cycle from creation to payment. Key attributes include:

- case:concept:name: Unique identifier for each case.

- concept:name: Describes the activity related to the fine, such as "Create Fine," "Send Fine," or "Payment."

- time:timestamp: Records the date and time of each event.

- dismissal: Indicates whether the fine was dismissed. NIL means not dismissed; # or G indicate dismissal by a judge or the prefecture.

- vehicleClass: Specifies the class of the vehicle involved.

- article: Refers to the regulation under which the fine was issued.

- points: Shows points deducted from the driver's license.

- amount: Monetary value for "Create Fine" and "Add Penalty" events.

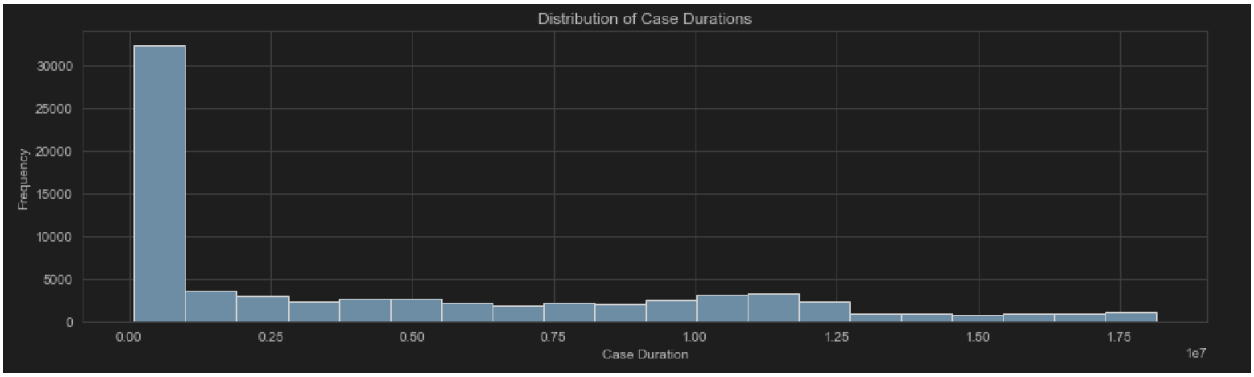
- expense: Additional expenses for the "Send Fine" event.

- paymentAmount: Amount paid in a "Payment" event.

Analysis of Case Duration Filtering and Distribution

We analyzed case durations in an event log using PM4Py. First, we filtered cases with durations between 60 seconds and 18144000 seconds (about 210 days) to focus on typical durations. We then calculated and displayed the number of these cases and plotted a histogram to show the frequency of each duration range.

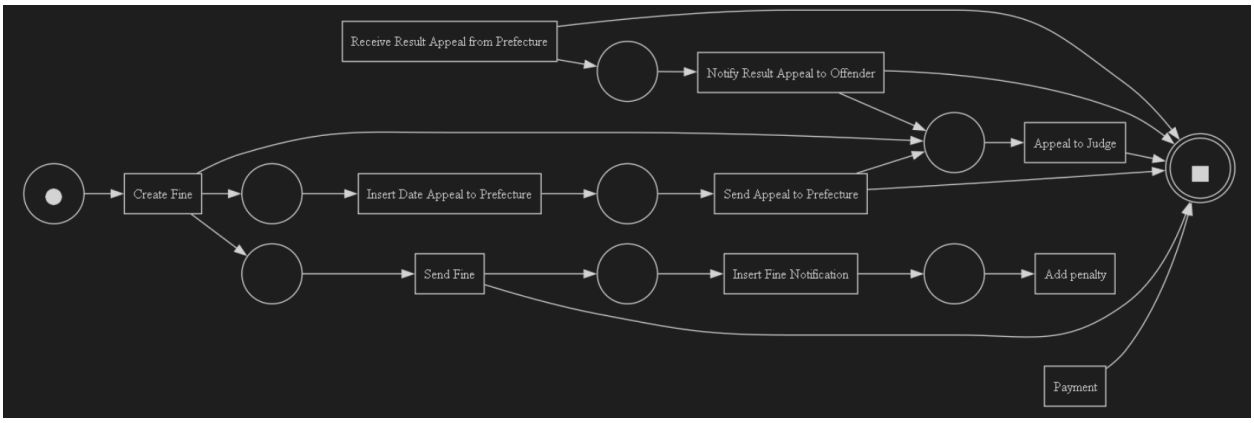
Next, we filtered the log to retain the top 50 most common variants, focusing on the most significant process patterns. We printed the number of cases remaining after this filtering. This analysis helps identify typical durations and common variants, providing insights for process optimization.



Process Discovery

Alpha Miner

This code snippet uses the PM4Py library to perform process discovery with the Alpha Miner algorithm. It takes a filtered event log (filtered_log) as input and discovers a Petri net, representing the process model. The discover_petri_net_alpha function outputs three components: the Petri net (net), the initial marking (initial_marking), and the final marking (final_marking). The view_petri_net function visualizes this Petri net. This process helps in understanding the actual flow of activities, identifying patterns and dependencies, and providing insights for process optimization.



This code snippet evaluates the quality of a discovered Petri net using several key metrics:

1. Replay Fitness:

- Measures how well the Petri net can reproduce the behavior observed in the event log. High fitness indicates the model accurately represents the log's traces.

2. Precision:

- Assesses how much of the behavior allowed by the Petri net is actually observed in the event log. High precision means the model avoids overgeneralizing and only allows behavior seen in the log.

3. Generalization:

- Evaluates the ability of the Petri net to generalize to new, unseen cases. A good generalization score suggests the model can handle variations not present in the log without being too restrictive.

4. Simplicity:

- Measures the complexity of the Petri net. Simpler models are preferred as they are easier to understand and analyze. High simplicity indicates the model is not overly complicated.

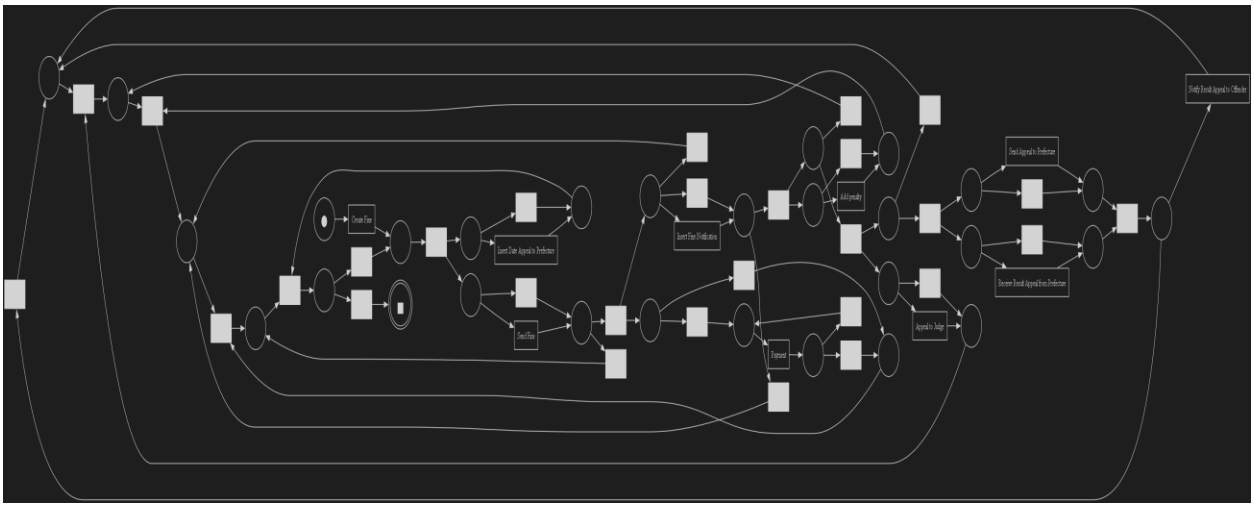
These metrics together ensure the discovered process model is accurate, precise, generalizable, and simple, providing a comprehensive assessment of its quality.

Infuctive Miner

The provided code snippet continues the process discovery by employing the Inductive Miner algorithm. This algorithm constructs a process model from an event log, focusing on creating sound models free from errors such as deadlocks or livelocks.

In this example, the Inductive Miner is used to create a Petri net from a filtered event log. The process starts by identifying the initial state (initial marking) and the final state (final marking) of the Petri net, which represent the beginning and end of the process. By visualizing this Petri net, users get a clear, graphical view of the process model, making it easier to understand how activities flow and relate to each other.

The Inductive Miner algorithm is applied to the event log to extract a process model that accurately shows the observed sequences and dependencies of activities. This model is essential for analyzing how the process actually runs and finding potential areas for improvement. This approach ensures that the discovered model is both accurate and reliable, providing a strong basis for further process analysis and optimization.

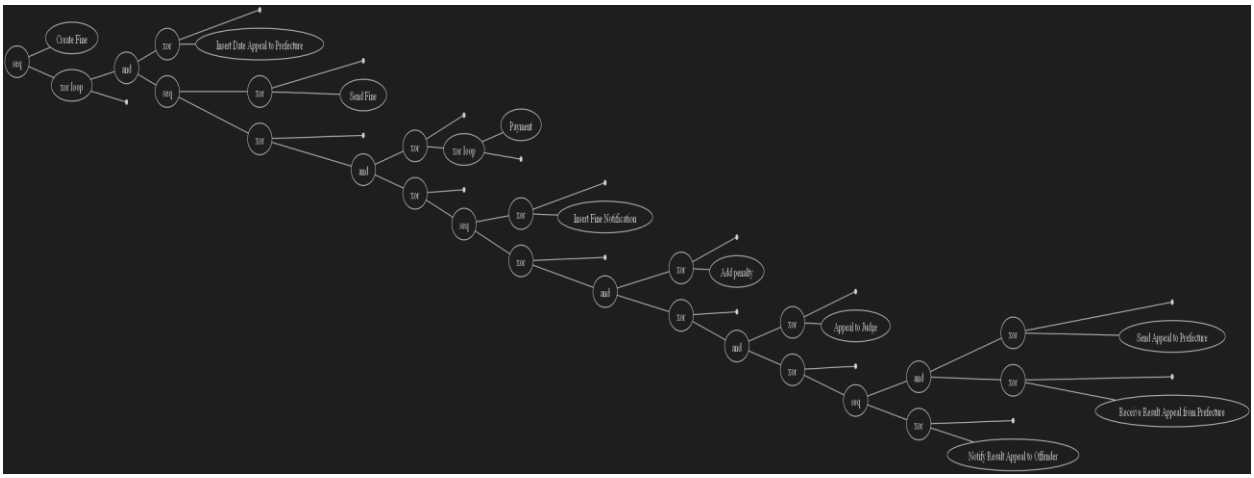


This code snippet extends the process discovery by applying the Inductive Miner algorithm and visualizing the discovered process model as a process tree.

The Inductive Miner algorithm is used on the filtered event log to discover a process model, which is represented as a process tree. Process trees are hierarchical representations of process models, making them easier to interpret and analyze, especially for complex processes.

After discovering the process model, the code uses the PM4Py visualization module to convert the process tree into a graphical visualization. This involves applying the visualizer to the model with specific parameters to generate a PNG format visualization. The visualization is then displayed, providing a clear, graphical representation of the process tree.

This approach offers an alternative to Petri nets, focusing on hierarchical structures that can simplify understanding of the process flow and dependencies, especially in complex processes. It allows for an intuitive visual representation, aiding in the analysis and communication of the discovered process model.



This code snippet demonstrates the conversion of a discovered process model from a process tree to a Petri net and subsequently visualizes it using a frequency-based approach.

1. Conversion of Process Tree to Petri Net:

- The discovered process model, initially represented as a process tree, is converted into a Petri net using the process tree converter (pt_converter).

- This conversion results in a Petri net (net), along with its initial state (initial_marking) and final state (final_marking).

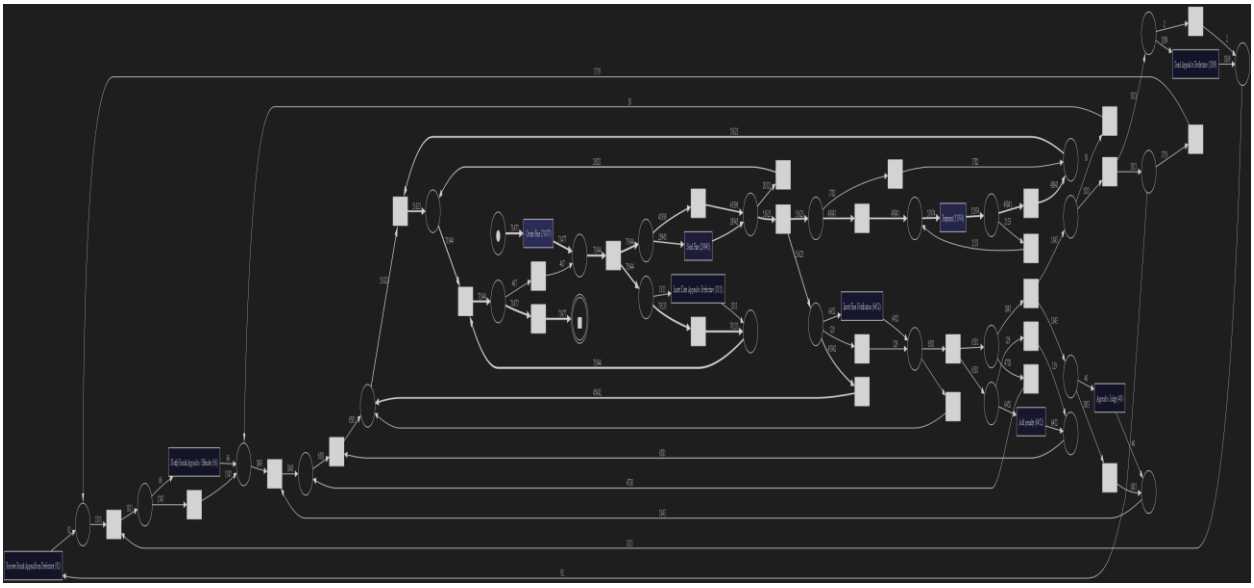
2. Visualization of Petri Net:

- The code uses `pn_visualizer` to display the converted Petri net.

- The visualization includes frequency data, showing how often each transition or place is used based on the event log (`filtered_log`). This helps identify the most and least frequent parts of the process.

- The final visualized Petri net is shown, providing a clear view of the process model with frequency details.

This method combines the hierarchical structure of process trees with the detailed visualization of Petri nets. By adding frequency information to the visualization, it offers a clear and detailed view of the process, highlighting areas with different activity levels and helping to spot potential bottlenecks or inefficiencies.



Continuing from before, the code evaluates the Petri net model's quality using key metrics for a thorough assessment of the process model.

First, it calculates the replay fitness metric with a token-based approach. This checks how well the Petri net can mimic the behavior in the event log. A high replay fitness score means the model accurately represents the event sequences in the log, showing that the discovered model closely matches the actual process.

Then, it measures the precision metric using the ETConformance token-based method. Precision evaluates how much of the behavior allowed by the Petri net is seen in the event log. A high precision score indicates that the model avoids overgeneralizing and only includes behaviors present in the log, ensuring a more accurate fit to the recorded data.

Generalization is a key measure that checks how well the Petri net can manage new, unseen cases that were not in the event log. A high score here means the model can adapt to different situations, showing it is robust and flexible without being too specific.

The simplicity metric evaluates how complex the Petri net is. This measure ensures the model isn't too complicated, making it easier to understand and analyze. A high simplicity score means the model is detailed enough to be useful but still easy to comprehend.

Together, these evaluations give a thorough assessment of the Petri net's quality, making sure the process model is accurate, adaptable, and simple. This complete validation is crucial for the reliability and usefulness of the model in analyzing and improving business processes.

Heuristic Miner Algorithm

The Heuristic Miner algorithm helps discover process models from event logs, focusing on capturing the main behavior while allowing for some noise and exceptions. This is particularly useful for dealing with real-life event logs that often have irregularities.

1. Heuristic Miner Algorithm:

- This algorithm analyzes the filtered event log to create a process model.

- It uses rules to find the most important connections between activities, balancing common patterns with some noise tolerance.

- A dependency threshold (set at 0.9 here) filters out weaker connections, highlighting the stronger and more relevant ones.

2. Heuristic Net Discovery:

1. Heuristic Miner Algorithm:

- Use the `apply_heu` function from the Heuristic Miner to generate a heuristic net from the event log.

- A heuristic net is a graphical representation that shows the activities and their dependencies in the discovered process model.

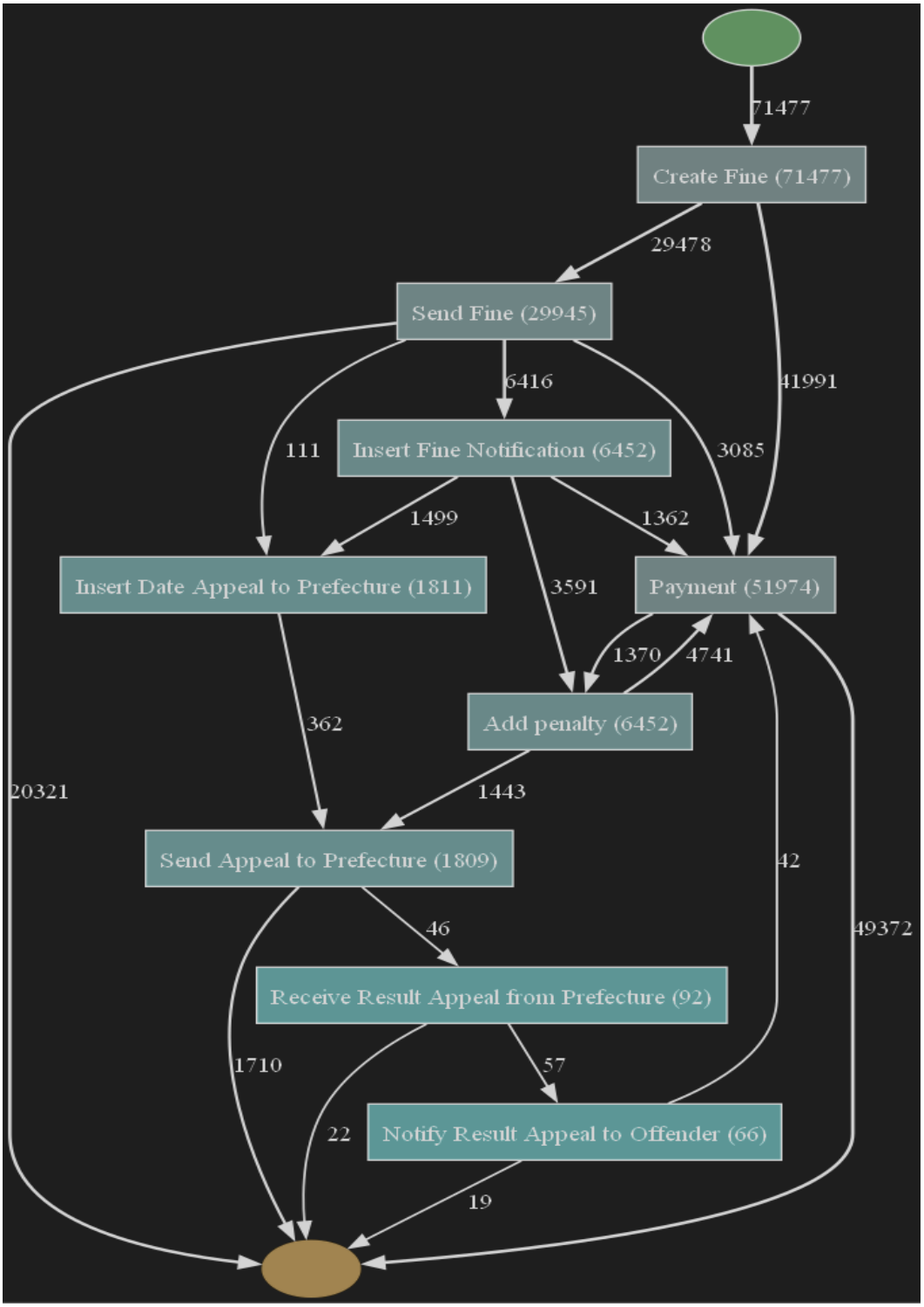
3. Visualization of Heuristic Net:

- Visualize the heuristic net using the `hn_visualizer`.

- This visualization helps understand the process model by showing the flow of activities and their dependencies.

- Stakeholders can see the main paths of the process and the frequency of activities, making it easier to analyze and interpret.

This method offers a flexible and realistic view of the process model, handling the imperfections often present in real event logs. The Heuristic Miner algorithm highlights essential process behaviors and filters out noise, while the visualization provides a clear understanding of the process dynamics.



Heuristic Miner Algorithm: Process Filtering and Visualization Report

This report outlines the steps taken to filter event logs according to specific criteria and visualize the outcomes. The process involves categorizing the log file, applying filters based on performance and activities, and then visualizing the filtered data.

1. Log File Categorization:

- The `categorizeLogFile` function divides the log file into categories based on the `vehicleClass` attribute: "A", "C", "M", and "R".

- This categorization is achieved using the `pm4py.filter_event_attribute_values` function, which filters the log by the specified attribute values.

2. Log Filtering:

- The `filter` function refines the log through several filtering steps:

  - Performance Filtering: Keeps cases with durations between 60 seconds and 210 days (18,144,000 seconds).

  - Top Variants Filtering: Retains the top `k` log variants.

  - Start and End Activities Filtering: Keeps cases that start with "Create Fine" and end with specified activities.

  - Rework Filtering: Excludes cases where certain activities ("Send Fine", "Insert Fine Notification", "Add penalty") occur more than twice.

- The outcome is a refined log with fewer, more relevant cases.

3. Filtering by Category:

- The `filter` function is applied to each categorized data set.

- The number of cases before and after filtering is displayed for each category.

4. Results Visualization:

- A bar chart visualizes the number of cases in each category before and after filtering.

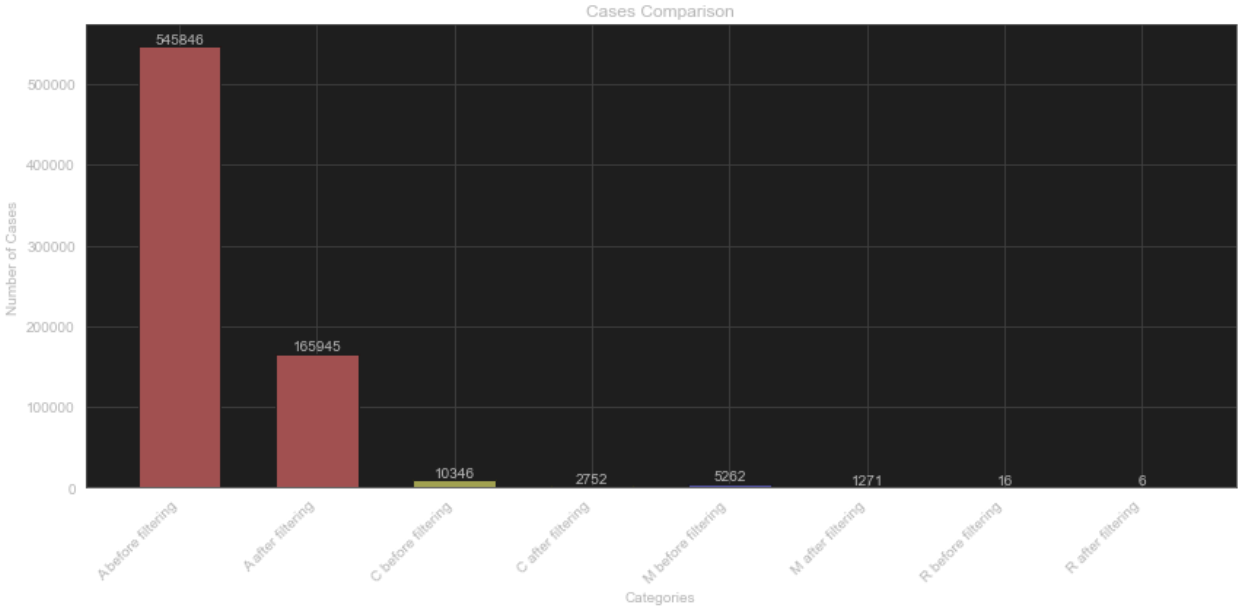
- The bar chart features:

  - Labels such as 'A before filtering', 'A after filtering', 'C before filtering', 'C after filtering', 'M before filtering', 'M after filtering', 'R before filtering', and 'R after filtering'.

  - Corresponding values for each category.

  - Different colors for each category to improve visual clarity.

This process involves sorting, filtering, and visualizing event logs to see how many cases meet certain conditions. Filtering keeps only the relevant cases, and visualization shows a clear comparison of case numbers before and after filtering for each category. This method helps refine and analyze process data to gain useful insights.




```
import numpy as np
import matplotlib.pyplot as plt

x_labels = ["fitness", "precision", "generalization", "simplicity"]
y_labels = ["alpha miner", "inductive miner", "heuristic miner"]

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(25, 12))

for i, key in enumerate(alpha_miner_heatmap.keys()):
    row = i // 2
    col = i % 2
```

```
    values = np.array([
        alpha_miner_heatmap[key],
        inductive_miner_heatmap[key],
        heuristic_miner_heatmap[key]
    ])
    values = np.round(values, decimals=2)
    ax = axes[row, col]
    im = ax.imshow(values, cmap='coolwarm', vmin=0, vmax=1)  # Change the colormap
to 'coolwarm'
    ax.set_title(key)
    ax.set_xticks(np.arange(len(x_labels)))
    ax.set_xticklabels(x_labels)
    ax.set_yticks(np.arange(len(y_labels)))
    ax.set_yticklabels(y_labels)
    for i in range(len(y_labels)):
        for j in range(len(x_labels)):
            text = ax.text(j, i, values[i, j], ha="center", va="center", color="black")

cbar = fig.colorbar(im, ax=axes.ravel().tolist(), shrink=0.6)
cbar.ax.set_ylabel('Values')

plt.show()
```

Here's a simplified version of the provided text:

The code snippet shows how to visualize the evaluation metrics of process models found by different mining algorithms (Alpha Miner, Inductive Miner, and Heuristic Miner) for each segment of event log data. Here's a breakdown of the process:

1. Setup and Initialization:

- Imports `numpy` for numerical operations and `matplotlib.pyplot` for plotting.

- Defines labels for the x-axis (evaluation metrics) and y-axis (mining algorithms).

2. Creating Subplots:

- Creates a figure with a 2x2 grid of subplots to display heatmaps for each segment of event log data.

- Sets the figure size to 25x12 inches for clarity.

3. Plotting Heatmaps:

- For each segment in the `alpha_miner_heatmap` dictionary:

- Identifies the subplot using row and column indices.

- Gathers and rounds evaluation metrics for Alpha Miner, Inductive Miner, and Heuristic Miner to two decimal places.

- Generates a heatmap using `imshow`, with a 'coolwarm' colormap and value range between 0 and 1.

- Titles the subplot with the segment name and labels the x and y ticks with the evaluation metrics and mining algorithms.

- Annotates each heatmap cell with its value for clarity.

4. Colorbar:

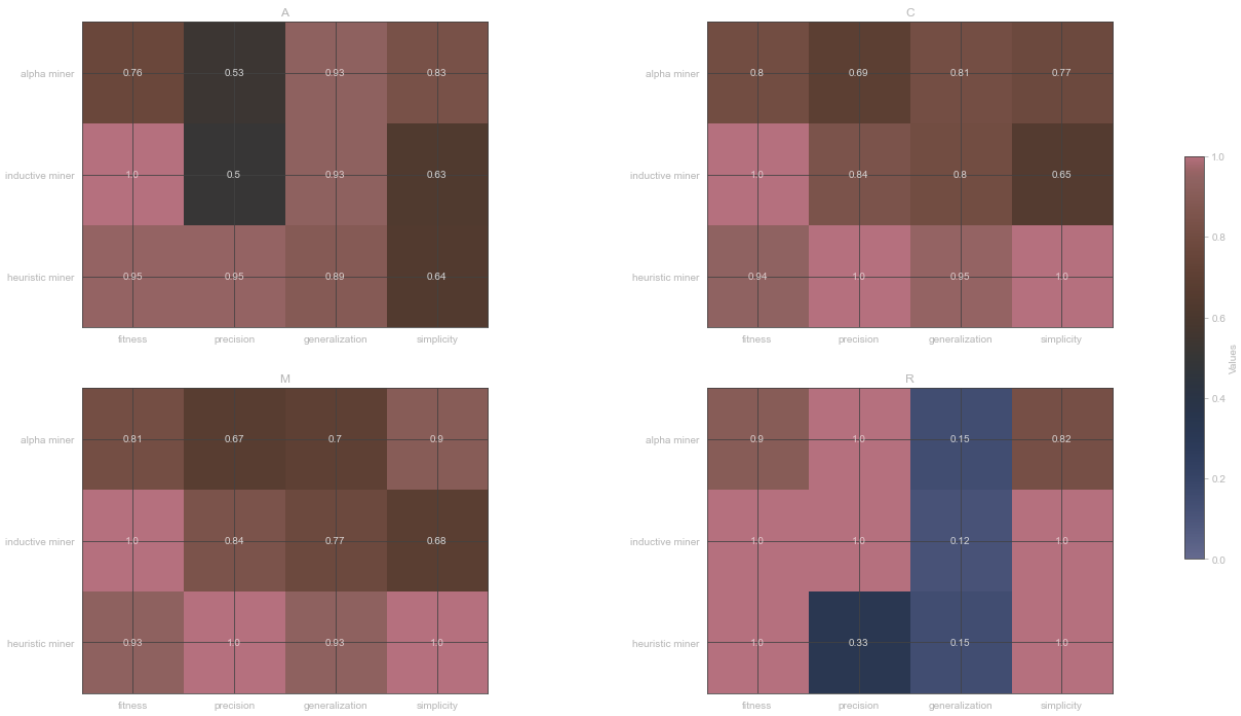
- Adds a colorbar to the figure for reference.

- Shrinks the colorbar to 60% of its original size and labels it 'Values'.

5. Displaying the Plot:

- Calls `plt.show()` to display the heatmaps.

This visualization compares how well Alpha Miner, Inductive Miner, and Heuristic Miner perform in terms of fitness, precision, generalization, and simplicity. Each part of the event log data is shown in separate subplots, with heatmaps displaying the performance of each mining algorithm. The colorbar helps to easily see the values, highlighting the strengths and weaknesses of each algorithm for different segments. This visual comparison helps understand how effective each process mining technique is on various parts of the event log data.



Clustering

This code performs clustering analysis on a dataset of event logs to identify patterns and group similar cases. The overall process includes feature extraction, normalization, and clustering using the K-means algorithm, followed by an evaluation and examination of the clusters.

1. Feature Extraction and Encoding:

- The code extracts features from an event log dataframe (`log_df`) and encodes them into a one-hot encoded vector, creating a feature dataframe (`features_df`). This process converts categorical data into a format suitable for machine learning algorithms.

2. Normalization:

- The extracted features are normalized using MinMaxScaler, which scales the data to a range between 0 and 1. This step ensures that all features contribute equally to the clustering process.

3. Elbow Method for Optimal Clusters:

- The code uses the Elbow method to determine the optimal number of clusters by plotting the sum of squared errors (SSE) for different numbers of clusters. The goal is to find the "elbow point" where the rate of decrease in SSE slows down, indicating a suitable number of clusters.

4. Clustering with K-means:

- After determining the optimal number of clusters (in this case, 3), the K-means algorithm is instantiated and fitted to the normalized data. The algorithm assigns each observation to one of the clusters based on feature similarity.

5. Cluster Assignment:

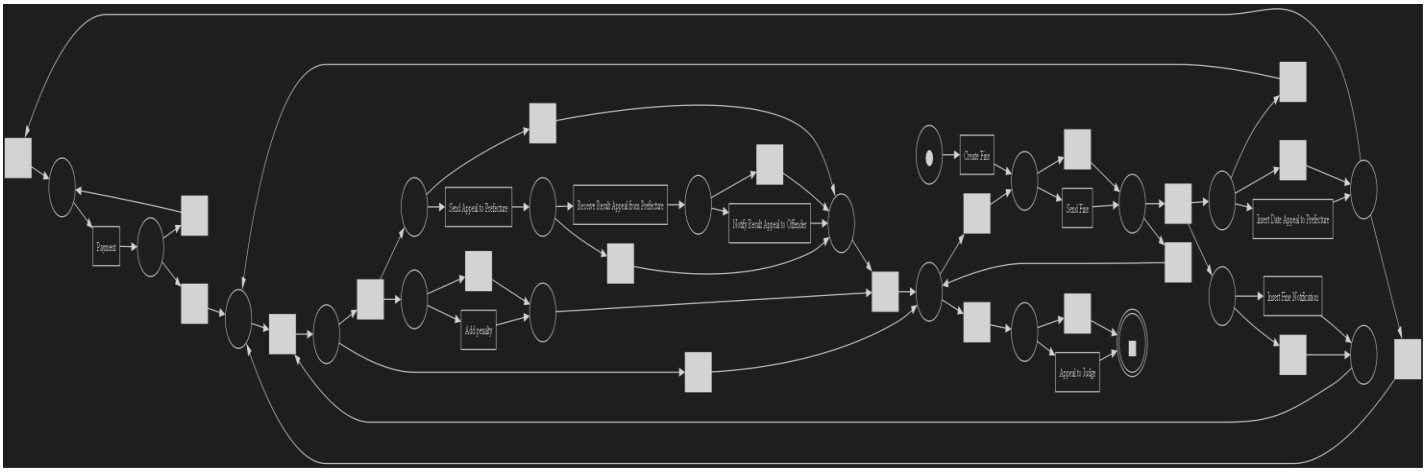
- The cluster assignments are appended to the original feature dataframe (`features_df`), allowing for the identification of which cases belong to which clusters.

6. Analysis of Clusters:

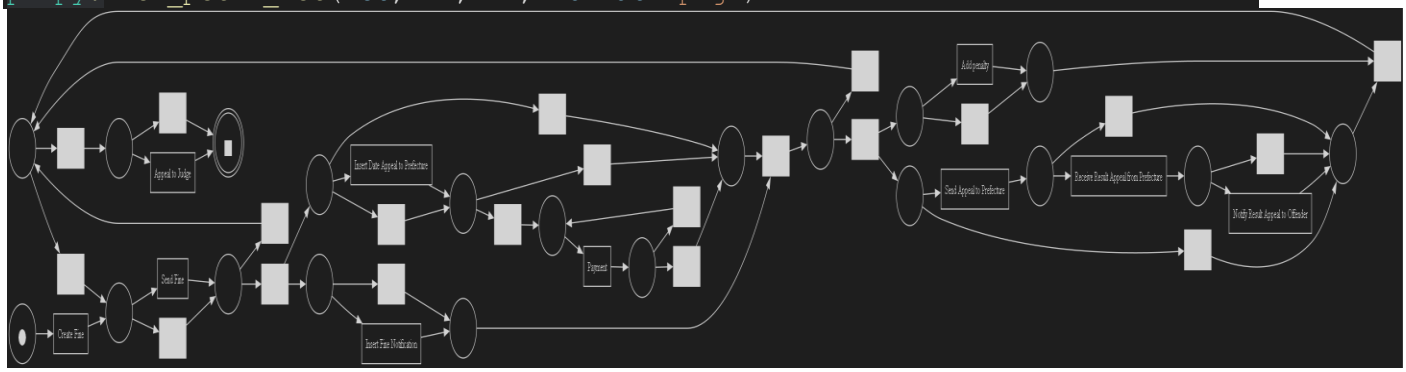
- For each cluster, the code filters the original event log to retrieve cases belonging to that cluster. It then calculates and prints the number of events and unique cases in each cluster, providing insight into the distribution and characteristics of the clustered data.

This approach helps in understanding the underlying structure of the event log data and identifying distinct groups of cases with similar attributes. This information can be useful for various analyses, including process improvement, anomaly detection, and resource allocation.

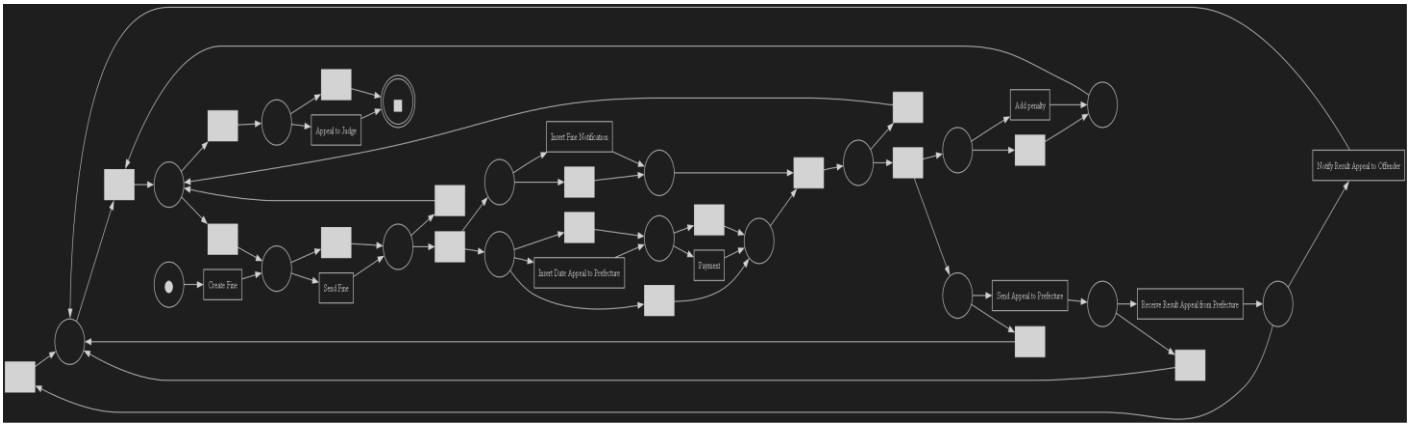
```
# create and visualize the petri net on clsuter1 using inductive miner
net, im, fm = pm4py.discover_petri_net_inductive(cluster1_log)
pm4py.view_petri_net(net, im, fm, format='png')
```




```
# create and visualize the petri net on clsuter2 using inductive miner
net, im, fm = pm4py.discover_petri_net_inductive(cluster2_log)
pm4py.view_petri_net(net, im, fm, format='png')
```




```
# create and visualize the petri net on clsuter3 using inductive miner
net, im, fm = pm4py.discover_petri_net_inductive(cluster3_log)
pm4py.view_petri_net(net, im, fm, format='png')
```



Decision Tree

A Decision Tree is an easy-to-understand classification method used in data analysis. It splits data into smaller groups based on feature values, forming a tree structure of decisions. Each node is a feature, and each branch is a decision rule, making the decision-making process simple to follow.

When analyzing logs, a Decision Tree can classify events and predict outcomes, like payment completion. By learning from past data, it identifies patterns and relationships between features and the target variable. In this project, the Decision Tree reached an accuracy of 98%, proving its effectiveness in predicting fine payments based on event log attributes. This helps improve management and operational efficiency.

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

Executed at 2024.06.17 23:51:25 in 15ms

Accuracy: 0.9598989160071822
```

Organizational Goals

This plan focuses on improving business processes.

Strategic Goal:

- Reduce shipping times by stopping fines after a certain period to avoid extra costs.

Operational Goals:

- Improve process compliance.

- Identify and fix bottlenecks.

- Optimize resource usage.

- Develop predictive models to assess risk levels.

To achieve these goals, the organization should:

1. Real-time Monitoring:

  - Use dashboards with process mining techniques for effective case management and to detect overdue cases.

2. Automated Reminders:

  - Implement systems to send reminders about due dates and benefits of timely payments, reducing postal costs.

3. Staff Training:

  - Conduct regular training sessions for staff to effectively manage problematic cases.

4. Improved Data Collection:

  - Enhance data collection methods and adapt the information system to extract more data for building predictive risk models.