Q1 Process Overview

We are using the *ProM* tool to provide the information required in the subtasks of this question.

a) All this information can be found in the *Log Visualizer (LogDialog)*. The event log consists of **16412 cases**, **674486 events**, and **36 unique activities**. The timeframe ranges from **Sat Apr 01 06:00:00 CEST 2023** to **Sun Mar 31 21:22:00 CEST 2024**.

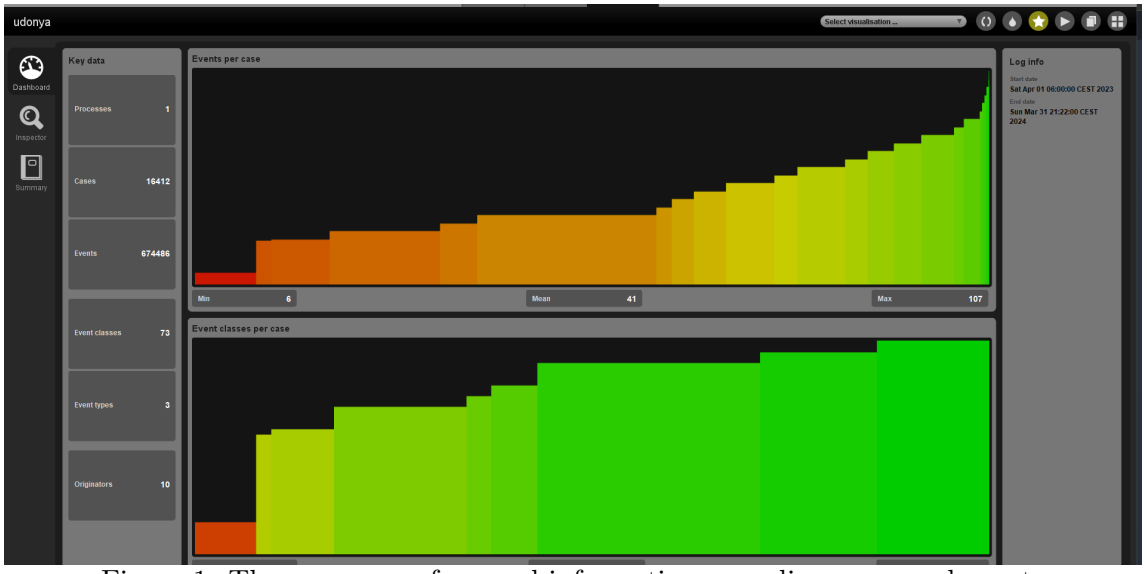


Figure 1: The summary of general information regarding cases and events

b) There are **7631 variants** in the log, and **41.097** events per trace/case in average.

c) As indicated by the plot, we are dealing with an event log whose case variants follow a *pareto distribution*. The majority of the case variants consist of only a few cases (mostly only one case), and only a small number of variants out of the *7631 variants* consist of more than 1 case. Also, the percentage of cases covered by the first top variants suggests that most of the cases are following a separate unique path. This can be observed by the fact that the most common variant accounts only for about 5.5% of the cases in the event log, which is indeed not favorable.

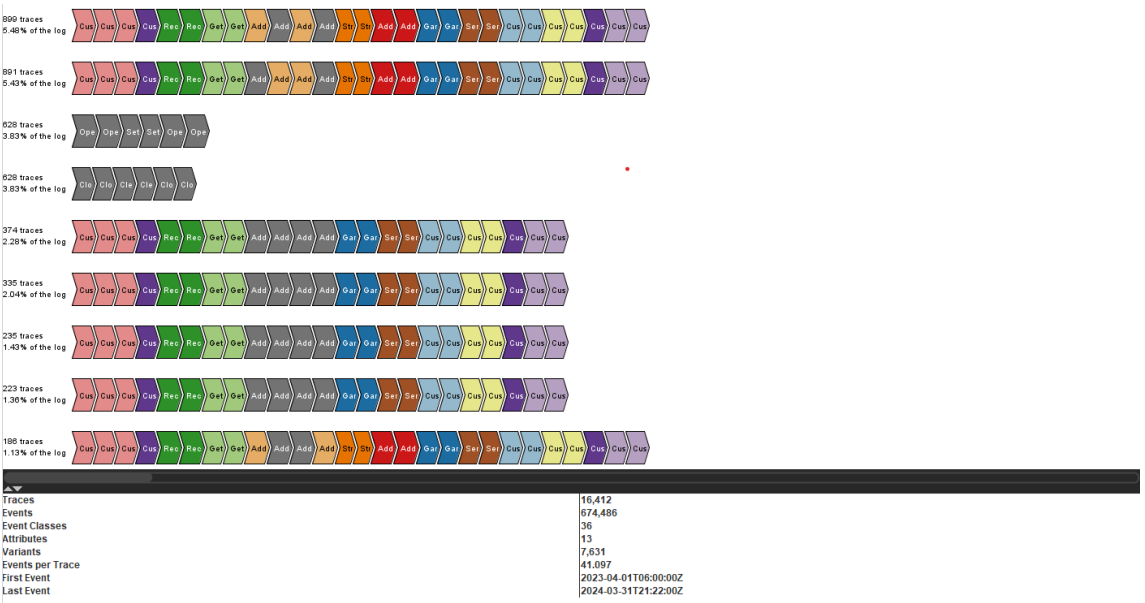


Figure 2: Information about case variants and their corresponding absolute and relative frequency in the *Explore Event Log* visualization

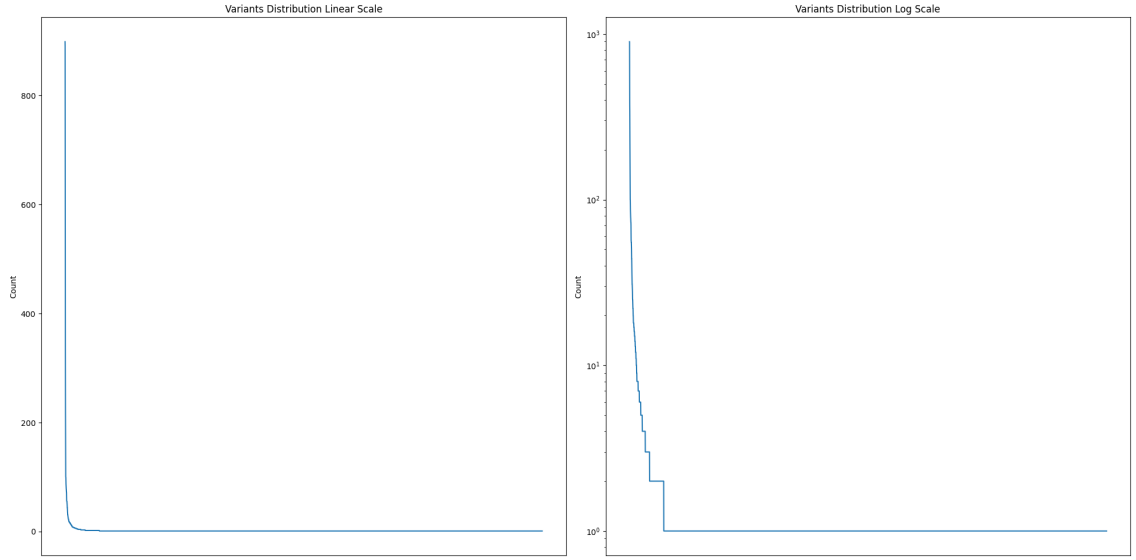


Figure 3: The frequency distribution of the variants displayed in linear and log scale

Q2 Visual Analytics

a)

i. Since the chart is sorted by the timestamp of the first event of the cases, we can get an overview of the arrival pattern of the cases. The arrival rate is almost constant; however, there's been a tangible reduction in the arrival rate of the cases from around the 20^{th} of June until the 23^{rd} of July, as displayed in figure 4.

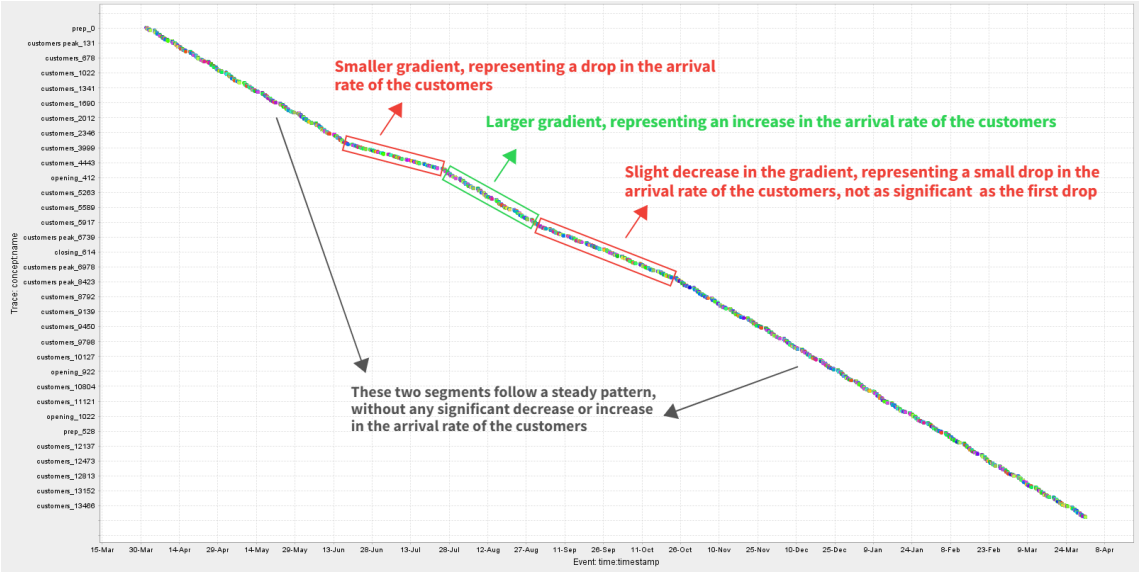


Figure 4: The *Dotted Chart* visualization of the event log, where the reduction in the arrival rate of customers has been marked as a prominent peculiarity

ii. Zooming into the dotted chart, we can observe that the weekdays when activities occur begin from Tuesday until Sunday, and no activities are recorded on Mondays, which is depicted in figure 5.

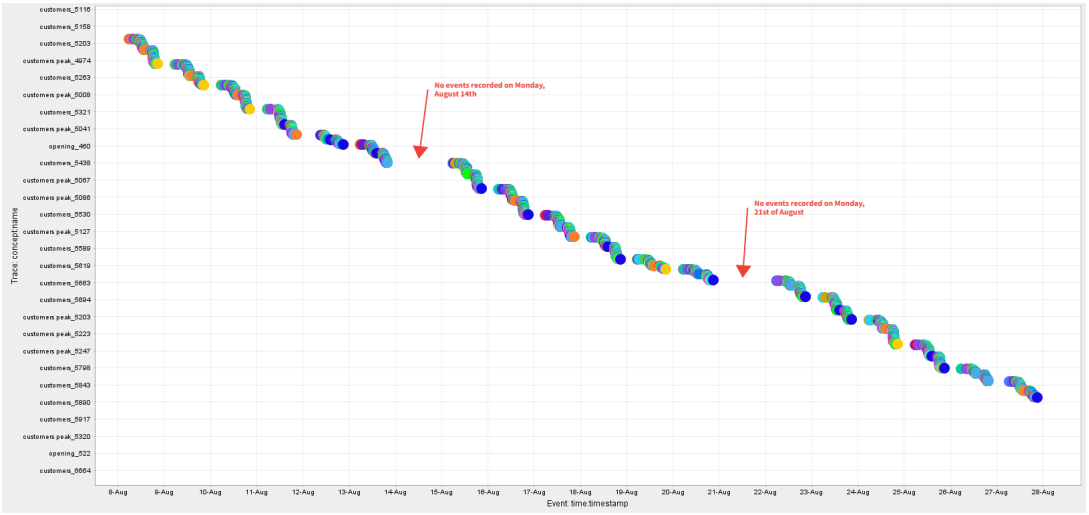


Figure 5: Overview of the weekly working pattern of the store

iii. By analyzing the dotted chart on various working days depicted in figure 6, we can see that the working hours begin from 06:00 until about 21:30, with a break starting from around 14:30 and ending at about 17:30. We can also observe that there are some periods in the morning during which no activities are recorded, which refers to time frames when an ingredient needs to be prepared before starting with a new one.

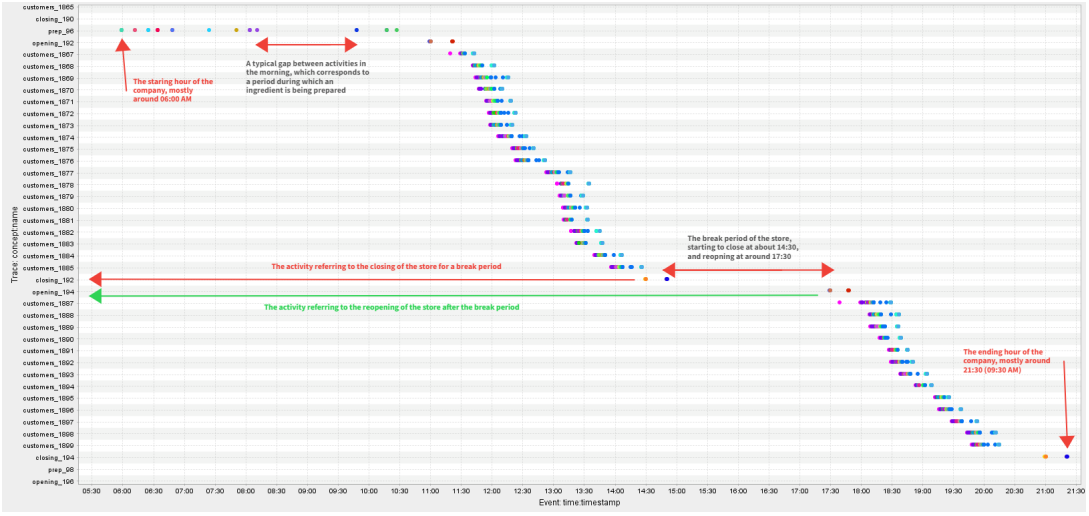


Figure 6: Overview of the daily working pattern of the store

iv. We can form a suitable visualization for this subtask by setting the X-axis attribute to *E:org:resource* which corresponds to the resource of the event, and setting

the Y-axis and color attribute to *E:concept:name* which refers to the corresponding activity of the event. The resulting chart is provided in figure 7. By observing the dotted chart obtained, we can conclude that there are some resources whose activities are in common, i.e., they execute similar activities to some extent. For instance, the resources **Baito-san A** and **Baito-san B** share certain activities. This is also the case for the reso urces **Deshi-san A**, **Deshi-san B**, **Oku-san**, and **Tenchou-san**.

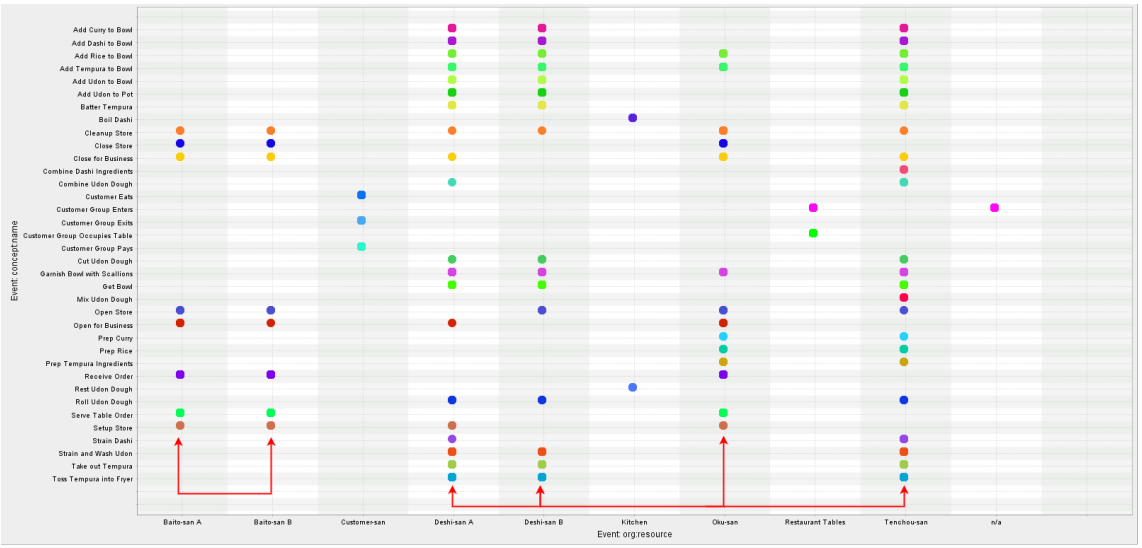


Figure 7: Overview of the activities executed by different resources. The arrows indicate resources that share certain activities.

b)

i. The first top 5 variants differ in the activities they contain, their number of events, and the related case attributes in the following way:

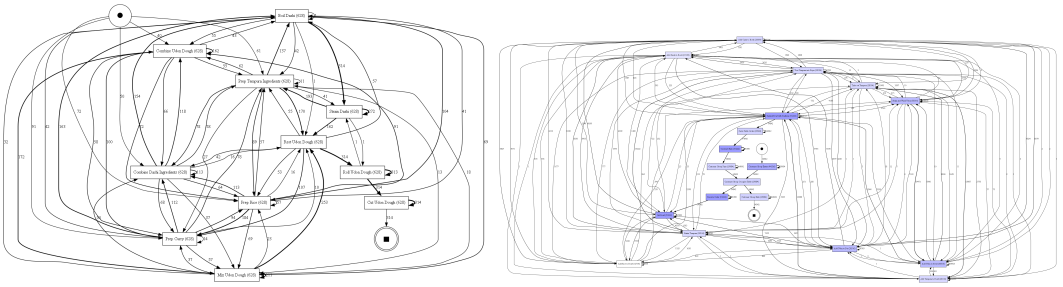
- **Activities**: The first two variants are common in the type of activities they contain; however, the ordering of activities differs to some extent. The third and fourth variants are common in their activities but are totally different from the first two variants, Finally, the fifth variant shares some activities with the first two variants but is completely different from the third and fourth variants. More precisely, the main difference between the activities related to the first, second, and fifth variants with the activities executed in the third and fourth variants is that the former indicates the procedures for the arrival of a customer and their orders, whereas the latter represents the activities for closing and opening the store.
- **Number of events**: The first two variants consist of 27 events, the third

and the fourth variants consist of 6 events, and the fifth variant consists of 23 events. Note that some events may correspond to the same activity; however, they have been separated due to the consideration of lifecycle transition. Besides, the first variant accounts for 899 traces, the second one for 891 traces, the third and forth for 628 traces, and the fifth for 374 traces in the event log.

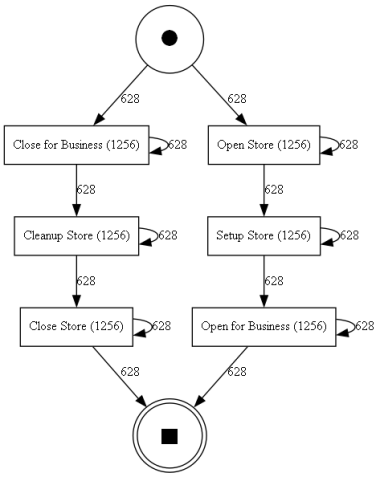
- **Case attributes**: The cases related to the first, second, and fifth trace variants have the case-level attributes *case_type, concept:name, customer_count, customer_group_id*, whereas the cases in the third and fourth variants have the case-level attributes *case_type, concept:name, description*. Note that the *case_type* of the cases in the first group is *customers*, whereas the *case_type* of the cases in the latter group is *store*.

Figure 8: Overview of the 5 most frequent variants in the event log *udonya.xes*

ii. The DFGs obtained from each of the logs filtered on different values of the attribute *case_type*, namely, *prep*, *store*, and *customers* are displayed in figure 9. Accordingly, all the DFGs derived contain self-loops, which is the result of having multiple events for the same activity in the event log (due to the existence of lifecycle transitions). Besides, the DFGs for the case types of *prep* and *customers* are quite complex and follow a so-called spaghetti model; therefore, they are not suitable for understanding the underlying flow of the processes. However, despite the simplicity of the DFG derived for the case type *store*, it's still not useful as the self-loops are not intended to exist and they only appear because of the duplicate events for the same activity representing the different stages (e.g. start and end) of the corresponding activity. This can be supported by the DFGs provided in figure 9.



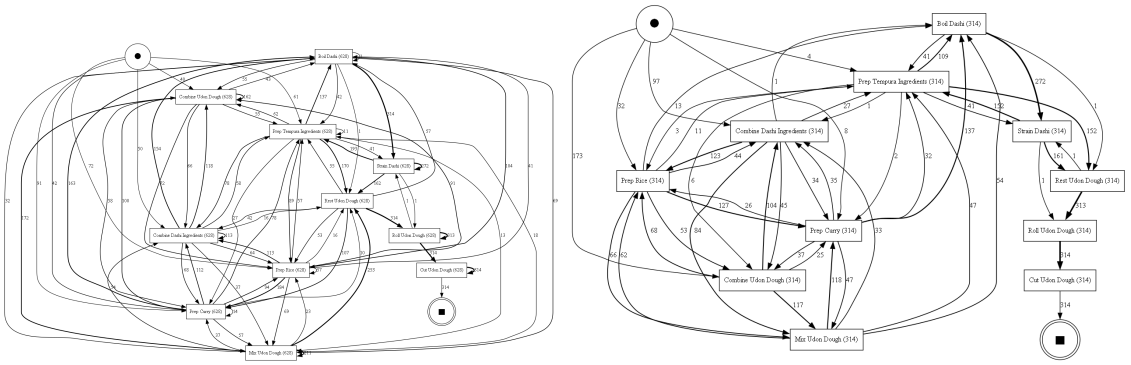
(a) (b)



(c)

Figure 9: (a) The DFGs obtained from the log filtered by the case type (a) *prep*, (b) *customers* (c) *store*. All the DFGs contain self-loops, and the DFGs displayed in (a) and (b) are complex and not useful for understanding the process.

iii. By comparing the DFGs obtained from the two event logs, we can see that the numbers displayed on the nodes (representing the number of times an activity has occurred) and the edges (representing the number of times two activities follow each other) differ. Diving more deeply into the log, we can find out that this difference is caused by the fact that more than one event is recorded for some activities in the event log *ingredient_prep_A.xes* indicating the start and end of the activity, i.e., lifecycle transition is taken into account; whereas in the second event log (*ingredient_prep_B.xes*), there's only one event per activity representing the end timestamp of the activity. Another relevant slight difference is the existence of self-loops in the DFG obtained for the event log *ingredient_prep_A.xes*, which indicates that the start and end events of most activities are often recorded consecutively in the log; thereby similar activities follow each other in the log.



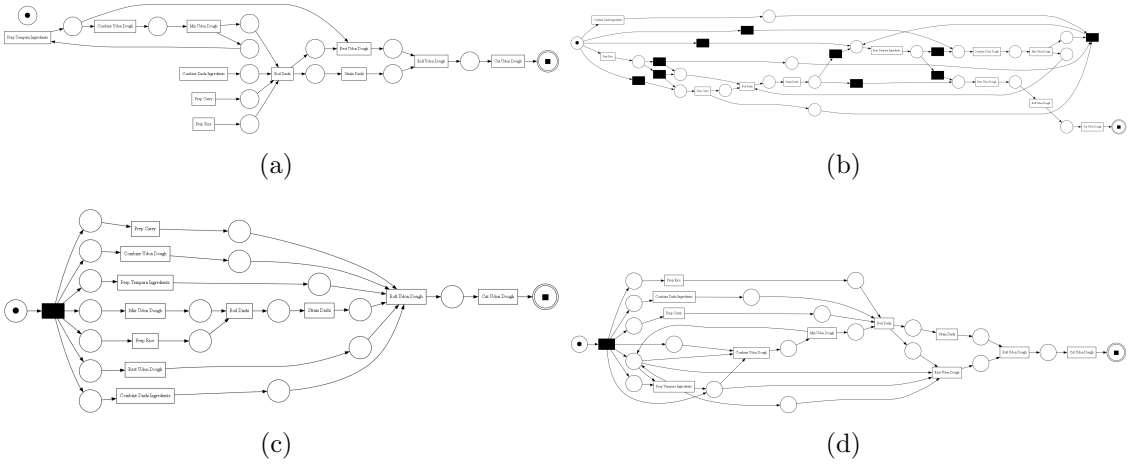
(a) (b)

Figure 10: (a) The DFG derived from the event log *ingredient_prep_A.xes* (b) The DFG derived from the event log *ingredient_prep_B.xes*

Q3 Process Discovery

a)

i. The models discovered by applying the different mining algorithms are depicted in figure 11:



(a) (b)

(c) (d)

Figure 11: The models discovered by applying the (a) *Alpha Miner* (b) *Heuristics Miner* (c) *Inductive Miner* (d) *ILP Miner*

Note, however, if we enhance the log by inserting artificial start and end events, the model discovered by the Alpha Miner will differ, as depicted in figure 12:

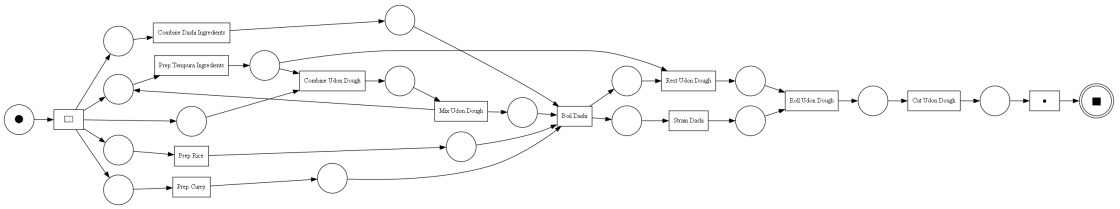


Figure 12: The models discovered by the *Alpha Miner* by inserting artificial start and end events into the log

ii. As represented in the figure 11, The model discovered by the **Alpha Miner** is not a workflow net; however, it's **simple but lacks proper readability due to its unconnected places and transitions without any input places**. On the other hand, the model discovered by the **Alpha Miner** displayed in figure 12, which uses an **enhanced version of the log**, is a workflow net, it's fairly simple and has decent readability. The model discovered by the **Inductive Miner** is sound, simple, and more readable than the two models discovered using the Alpha Miner. The models discovered by the **Heuristics Miner** and **ILP Miner** are workflow nets, but they are more complex and consist of more places, transitions, and arcs; therefore sacrificing readability. The fitness, precision, and F1 score values are computed for each process discovery algorithm and provided in figure 13.

Note: Along with the fitness, precision, and soundness, you can find a simplicity score computed for each of the discovered models in the *Jupyter Notebook*. This score is based on the method discussed in Appendix B.

iii. According to the model, we can perform the activities *Mix Udon Dough* and *Combine Udon Dough* in any order; i.e., it's possible to perform the activity *Mix Udon Dough* before the activity *Combine Udon Dough*, which is not reasonable. However, there's no trace in the log that depicts this behavior. (Please refer to the *Jupyter Notebook* to find the proof regarding the non-existence of this behavior in the event log.)

iv. Considering the fitness and precision criteria, the model discovered by the *ILP Miner* outperforms the other models with a fitness score of about 1 and a precision score of about 0.94. Besides, it prevents some of the unexpected behaviors we saw in the model discovered by the *Inductive Miner* in *part (iii)*. It's also a sound workflow net; however, it has the downside of being rather complex. The model discovered by the *Alpha Miner* using the enhanced version of the log is also a sound workflow net and performs very well in terms of fitness (0.96) and precision (0.93), and is simpler than the model discovered by the *ILP Miner*. Next comes the *Inductive Miner*. The model discovered by the *Indcutive Miner* is a sound workflow and also performs very

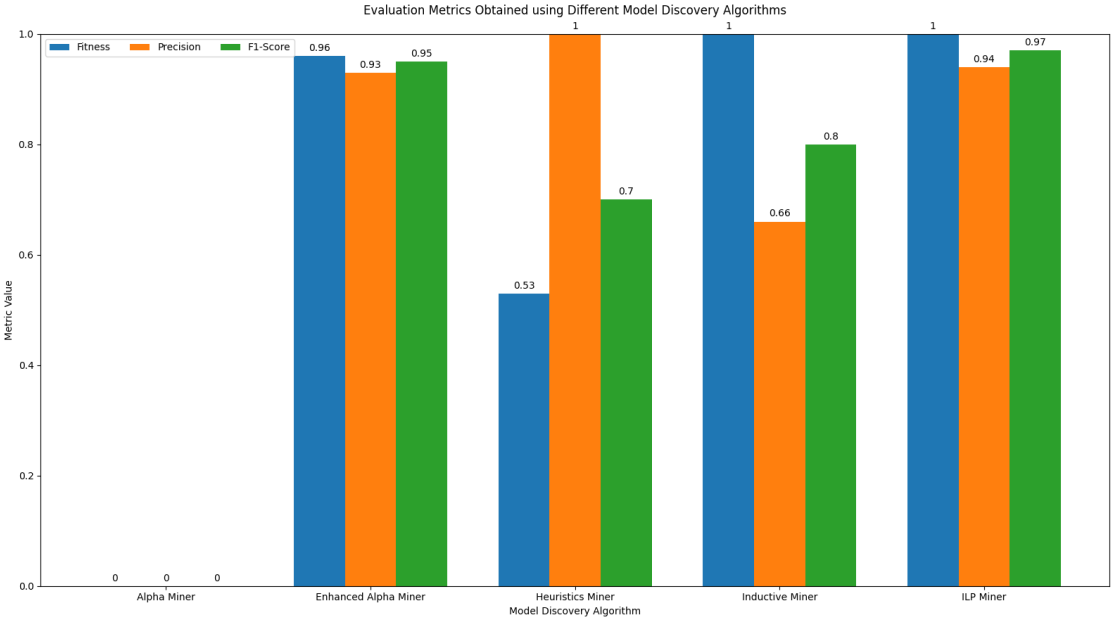


Figure 13: The evaluation metrics computed for the different model discovery algorithms

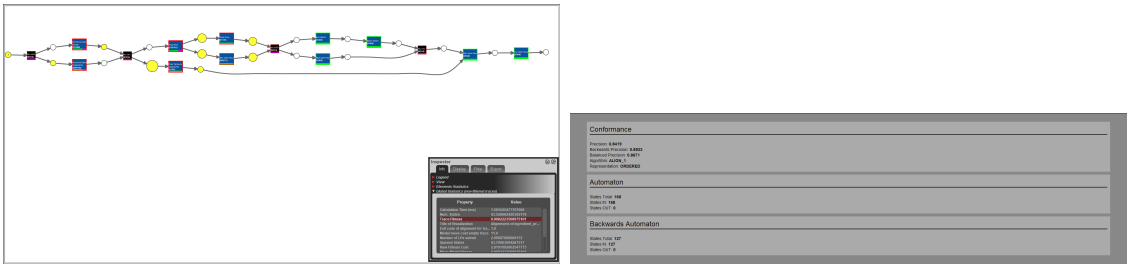
well in terms of fitness (almost equal to 1), but it lacks precision as it receives a score of about 0.66. Considering the model, we can attribute this impreciseness to the fact that the model allows for the parallel/concurrent execution of a number of activities at the beginning, which might not be reasonable. The *Heuristics Miner*, however, does not discover a sound model and performs poorly with a score of about 0.53 in terms of fitness, but it's quite precise as it achieves a precision score almost equal to 1. Finally, the model discovered by the *Alpha Miner* using the original log is not even a workflow net as it has unconnected parts and allows for the infinite firing of some of the transitions because they have no input places; Therefore, we cannot explicitly calculate the fitness and precision for the model, but the model is indeed imprecise and lacks fitness. We rank the models in the same order as we discussed their attributes; however, if simplicity matters significantly and inserting artificial start and end events to the log is not considered undesirable, we could change the ranking of the models discovered by the *ILP Miner* and *Alpha Miner with enhanced log*.

b)

i. In order to get a model with the prescribed constraints, we need to lower the *paths* slider to around 0.2, which results in the model represented in figure 14. With this adjustment, we obtain a model that has a fitness score of about 0.9 and a precision of around 0.84 on the *ingredient_prep_B* event log, as displayed in figure 15. Note, however, setting the *paths* slider to any value between around 0.13 and 0.23 satisfies the constraints, i.e., they exclude clearly out-of-order behaviors and achieve a fitness and precision score above 0.8.

(a) (b)

Figure 14: (a) The model discovered using the *Inductive Visual Miner* and setting the *paths* slider to 0.2 (b) The equivalent Petri net



(a) (b)

Figure 15: (a) Fitness score for the model discovered by the *Inductive Visual Miner* (b) Precision of the model discovered by the *Inductive Visual Miner*

ii. Since we have adjusted the *paths* slider in the previous subtask, there is almost no out-of-order execution of activities possible to be replayed on the model. Regardless of this, and although the fitness and precision of the model computed on the *ingredient_prep_B* event log are high, the resulting model does not account for some of the concurrencies that seem reasonable in the process. For instance, based on the discovered model, the activity *Prep Tempura Ingredients* and *Prep Rice* come only after having completed the activities *Combine Dashi Ingredients* and *Combine Udon Dough*, which seems to be an unnecessary constraint since there exist traces with the opposite order or with the activities being executed within the same interval.

c)

i. Please refer to the *Jupyter Notebook* provided alongside this report.

ii. Table 1 represents the results of running the *Inductive Miner* with different noise thresholds:

Noise Threshold	Fitness	Precision	F1 Score
0.0	1.0	0.45	0.62
0.25	0.94	0.67	0.78
0.5	0.88	0.61	0.72
0.75	0.77	1.0	0.87
1.0	0.28	0.5	0.36

Table 1: The result of evaluating *Inductive Miner* using different noise thresholds

iii. Based on the results provided in table 1, the model obtained by setting the noise threshold to 0.75 achieves the highest F1 score among the other models with different noise configurations. Displayed in figure 16, We can see that the resulting model has a sequential structure; however, due to the existence of lifecycle transitions, multiple activities can start while the previous ones have not ended yet, i.e., the model has concurrent semantics despite its sequential structure. Nevertheless, the model is too precise, i.e., it does not allow for any trace with different concurrency levels, which leads to the model not being well-generalized. Relevantly, the model lacks a high fitness value as it only allows for traces with a certain order of events.

Figure 16: The model discovered using the *Inductive Miner* with a noise threshold of 0.75, resulting in the highest F1 score.

The models discovered in the first two parts tend to be more complex and allow for more concurrent and potentially out-of-order behavior, whereas this model is simple and totally precise and can only replay a certain trace variant with a fixed order of events. In addition, the F1 score achieved by this model (i.e., 0.87) is placed in between the minimum and maximum F1 scores achieved by the models in the first two parts.

iv. Considering the 4 quality criteria used for evaluation, the model performs well in terms of precision and simplicity. However, it does not perform very well in terms of fitness due to its sequential structure, i.e., the alignment consists of *Move on log only* and *Move on model only* transitions. Also, the model is not very well-generalized to fully capture the behavior of common traces in the log. As a result, the model is not recommended despite its simplicity and preciseness.

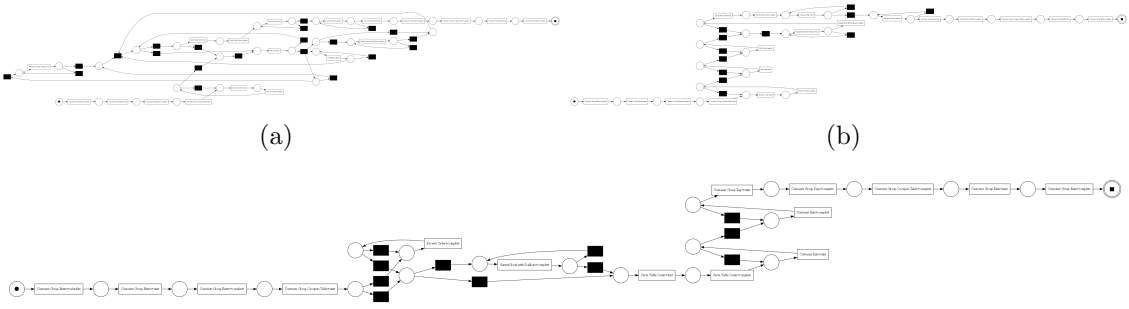
Noise Threshold	Fitness	Precision	F1 Score
0.0	1.0	0.67	0.8
0.5	1.0	0.88	0.93
1.0	0.83	0.9	0.86

Table 2: The result of evaluating *Inductive Miner* using different noise thresholds on the *customer_perspective.xes* event log

Q4 Conformance Checking

a)

i. Running the *Inductive Miner* using different noise thresholds, we obtain the evaluation results for each discovered model displayed in table 2. We can observe that using a noise threshold of 0.5 results in the highest F1 score. The discovered models are depicted in figure 17.



(a) (b)

(c)

Figure 17: Model discovered by running the *Inductive Miner* with a noise threshold of (a) 0.0 (b) 0.5 (c) 1.0

ii. First, we claim that both models are **workflow nets** since they have a well-defined source place and a well-defined sink place, and all other nodes are on the path from the source place to the sink place.

Both models are exactly similar in terms of whether each soundness property holds or not. Therefore, we summarize the arguments for both models as below:

- **Safeness**: The resulting Petri-net is **not safe** because there exist places that can hold more than one token.
- **Proper Completion**: The proper completion property **does not hold** since we can mark the sink place while there are still tokens remaining in other places.

- **Option to Complete**: Since the proper completion does not hold, the option to complete property **does not hold** either.
- **Absence of Dead Parts**: This property **holds** since for each transition there exists at least one possible firing sequence enabling that transition.

The potentially problematic places and transitions are indicated in the annotated figure 18. For a more transparent visualization, please refer to the figures 32 and 33 provided in the Appendix.

iii. The overall behavior of both models is almost similar as both go through the same steps to describe the process. The process starts with the arrival of a customer group, and continues with the customers occupying a table. Then, the orders are received, customers are provided with a bowl (Garnish bowl with scallions), and the table order is served. The process continues with the customers eating, paying, and finally ending their occupation of the table and exiting. However; there are nuances in the behavior of the two models. Firstly, the model provided in *customer_perspective.pnml* allows for the concurrent execution of garnishing the bowl with the scallions and serving the table order; whereas the model described by *customer_perspective_seq.pnml* indicates a sequential process, i.e., it only allows for serving the table order after the bowl is garnished. Secondly, the first model allows for receiving orders and any other activity that follows this activity to occur infinitely many times, whereas the second model prevents this behavior. Finally, the first model allows for the customers to pay before they have finished eating; whereas this behavior is not possible according to the second (sequential) model. These could be the underlying reasons the first model has achieved a lower precision.

(a) (b)

Figure 18: The hand-made Petri-net model represented by (a) *customer_perspective.pnml* (b) *customer_perspective_seq.pnml*

iv. The models discovered by the *Inductive Miner* using different noise thresholds are all sound; however, they tend to be more complex compared to hand-made models and yet achieve lower F1 scores. By comparing the two hand-made models and visualizing the trace variants in *ProM*, we can observe that the sequential model captures the behavior more accurately since the activities *Get Bowl*, *Garnish Bowl with Scallions*, and *Serve Table Order* appear sequentially in the respective order,

according to the most common variants depicted in figure 19. Therefore, it's not reasonable to include a concurrent execution of these activities in the model, which occurs in the non-sequential model and is a potential cause of the model's low precision. The sequential model is rather simple, has a high F1 score, and captures the behavior seen in the event log reasonably well. The most accurate model (in terms of the F1 score) obtained using the *Inductive Miner* is the one with a noise threshold of 0.5, which is rather simple and achieves a F1 score of 0.93, which is very close to the F1 score of the hand-made sequential model (0.95). If the soundness of the model is of high priority, this model is recommended over the sequential hand-made model. However, if the other quality criteria such as simplicity, fitness, precision, F1 score, and therefore, the ability to capture the behavior more accurately are more important to us, the sequential hand-made model is recommended.

Note: Please refer to Appendix B to see how the simplicity score is calculated for the models.

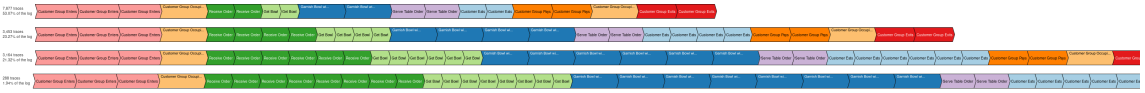


Figure 19: The most common variants in the event log *customer_perspective.xes*, which account for more than 99% of the cases

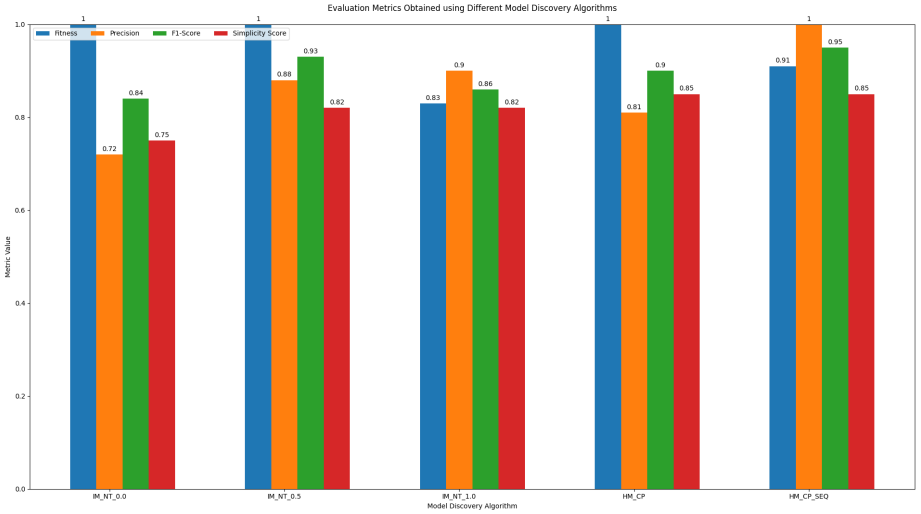


Figure 20: Fitness, precision, F1 score, and simplicity score computed for each of the discovered and hand-made models

b)

i. The activities *Garnish Bowl with Scallions* and *Customer Group Pays* are problematic since they seem to be missing in certain events in the event log, as indicated in figure 21.

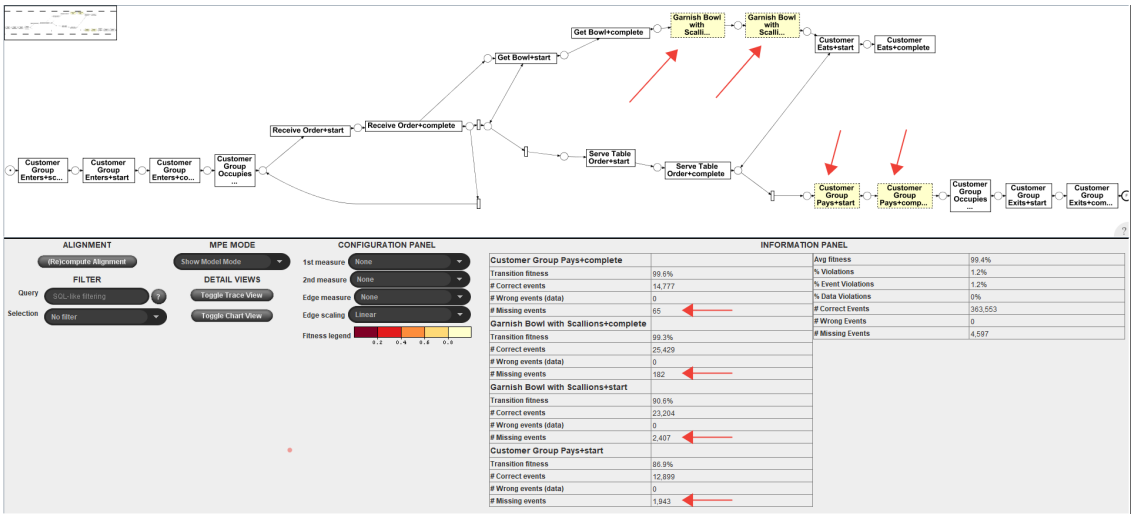


Figure 21: Deviations appearing in form of missing events discovered by the *Multi-perspective Process Explorer* plugin using the *customer_persepective_dev.xes* event log and *customer_perspective.pnml* model.

ii. In certain cases, there's no event recorded that corresponds to the payment of the customers after they finish eating, which is an unacceptable deviation from the perspective of the store owner.

iii. There are certain cases in which no event corresponding to the activity *Garnish Bowl with Scallions* is recorded. This could imply that the bowl is provided ungarnished for the customers, which is indeed annoying from the customers' perspective.

Q5 Diving into some Details

a)

i. Using the *get_variants* function implemented in *PM4Py*, we can observe that there are 7358 unique variants in the log.

ii. Iterating through the variants obtained from the previous subtask, we keep track of every unique trace variant in the form of a set and only consider adding a new variant if its set of activities, independent of their order, has not already been seen in the log. It means that we ignore a trace variant once we have visited another variant with the exact set of activities regardless of their order of appearing within the trace. With this approach, we can see that there exist **8 variations** of dish preparations. Inspecting the 8 variants further, we can see the different combinations of adding ingredients as below:

- *Dashi, Tempura, Udon*
- *Udon, Curry*
- *Tempura, Rice, Curry*
- *Dashi, Tempura, Udon, Curry, Rice*
- *Dashi, Tempura, Udon, Curry*
- *Udon, Rice, Curry*
- *Tempura, Udon, Curry, Rice*

iii. By investigating one of the cases corresponding to a variant, we can see that the following ingredients were added to the dish: *Curry, Rice, Udon.*

iv. Although we find only **8 dish preparation variants** and a maximum of **5 ingredients** for one meal, this might not be reasonable for the menu of a tiny restaurant. By inspecting the log, we can observe that a case ID refers to an instance of a customer group rather than a single customer, or even one customer but with different orders. Therefore, multiple orders might interleave within one case in the event log. This leads to the dish preparation variants to be misleading since the dish ingredients could be related to different orders. To account for this problem, we can consider the *global_order_id* as the Case ID in our event log, which makes it possible to easily distinguish the different orders and their corresponding dish ingredients. Following this alternative approach, the number of **dish preparation variants** is reduced to **4** with a maximum of **3 ingredients** for one meal, indicated below:

- *Dashi, Udon, Tempura*
- *Curry, Udon*
- *Curry, Tempura, Rice*
- *Curry, Rice*

b)

i. The values for different revenue aggregations are extracted from the *Jupyter Notebook* and displayed in table 3:

Revenue Aggregation	Value
Total Revenue	21,535,080.0
Daily Average Revenue	68,583.0
Average per Case Revenue	1,451.0

Table 3: Total, daily average, and average per case revenue. All values are rounded up to one decimal precision

ii. There are two obvious peculiarities in the process within considering the complete timeframe. First, at certain points in time, the total daily revenue reaches zero. This points out the non-working days, i.e., every Monday per week. Besides, we can observe a dramatic drop in daily revenue from around mid-June to almost the end of July. These peculiarities are marked in figure 22.

Figure 22: Overview of the daily revenue of the store, with the red window indicating a drop in the revenue from mid-June to the end of July, and the bottom red dots indicating non-working days

The result we obtain in this subtask is consolidated by what we observed in the *Visual Analytics* section in the subsections Q2.a.i and Q2.a.ii, as the dotted chart

displayed in figure 4 clearly indicates a reduction in the customer arrival rate within the specified time window, i.e., from mid-June to almost the end of July. Besides, we observed that no activities are recorded on Mondays, and that explains why the daily revenue in figure 22 touches the bottom line on Mondays.

c) Based on the *resource utilization* plot in figure 23, we can observe that there's a significant drop in the weekly working hours of the resources (marked by red arrows) in exactly the same time frame as there's a reduction in total revenue, which again consolidates our findings regarding the customers' arrival rate and the daily revenue of the store in that time frame. Additionally, we can see a significant increase in the working hours of *Deshi-san A*, *Deshi-san B*, and *Oku-san*, and a drop in the working hours of *Tenchou-san* in the time interval from 15^{th} of October to 12^{th} of November (marked by blue arrows). Altogether, on November 5^{th}, we observe the highest weekly sum of working hours; nevertheless, this week doesn't account for the highest revenue.

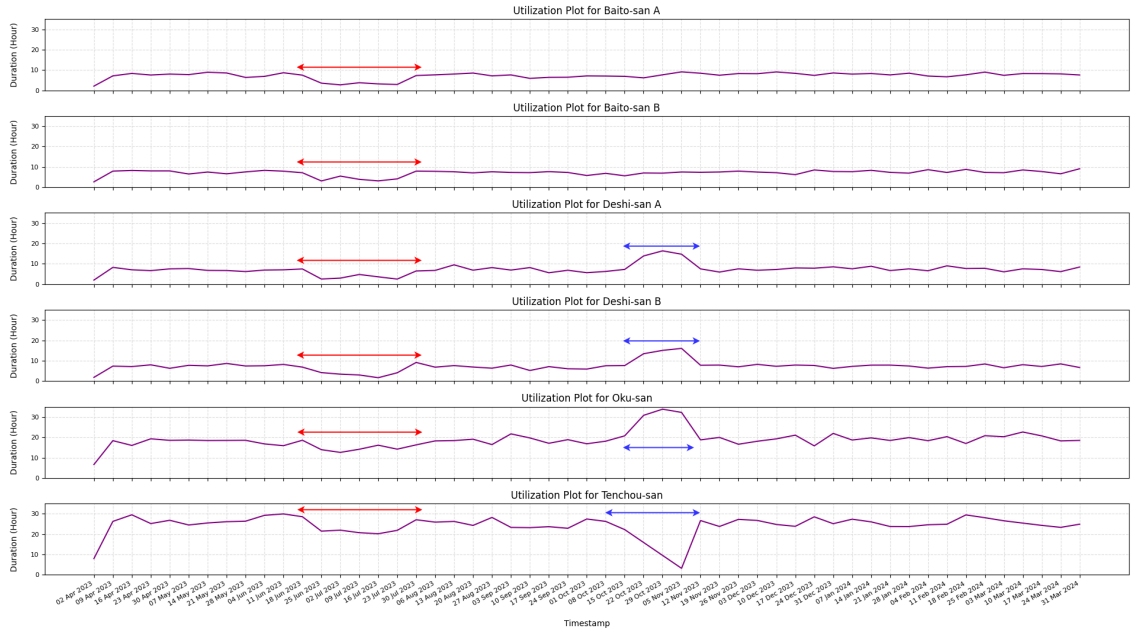


Figure 23: Weekly working duration of the resources

Figure 24: Weekly working duration of the resources based on different activities

However, we can relate our findings in this timeframe to the earlier findings we obtained in figure 7, which displays the activities different resources are responsible for. Accordingly, we can observe that the other resources that share the same activities as *Oku-san* have slightly increased their utilization for the activities to account for his low utilization in this period. This pattern is depicted in figure 24

Q6 Technical Questions

a) Consider the event log $L = [< a, c, d, b, c, d >, < c, a, a, d >, < c, d, b, c, d, a >]$. There's no trivial cut within the known cuts (*choice, sequence, parallel, loop*) at the beginning; therefore, the algorithm applies the *activity concurrent* fall-through by removing the activity a from every trace and performing a cut detection on the projected log denoted as L'. By doing so, we get $L|_a = [< a >^2, < a, a >]$ and $L' = [< c, d, b, c, d >^2, < c, d >]$. We see that there's a redo loop cut in the form of **Redo**(a, tau) on L' and a redo loop cut of the form **Redo**(c,d, b). The algorithm finds a *sequence cut* between the activities c and d in the next step. The Petri net and the corresponding process tree of the event log L are represented in figure 25:

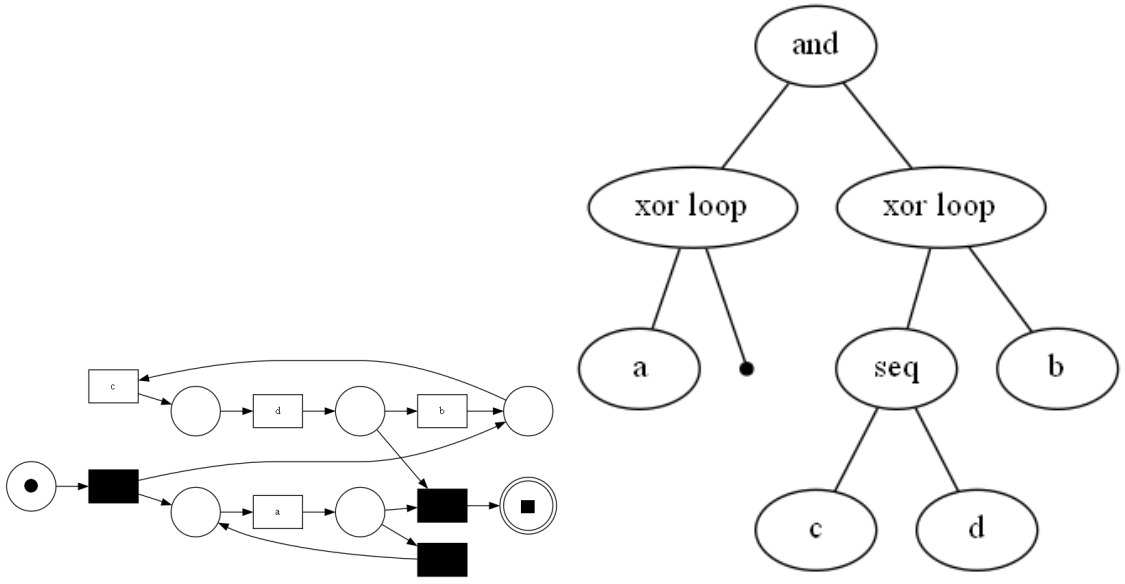


Figure 25: (a) Petri net discovered by running the *Inductive Miner* on event log L (b) The equivalent process tree

b)

i. Please refer to the *Jupyter Notebook* provided alongside this report.

ii. Based on the heatmap depicted in figure 26, which is obtained by performing conformance alignments between the model and different instances of the log with different percentages of deviations, we can observe that the more the log is mutated from its original state, the alignments tend to take longer. This is indeed reasonable as making more changes to the log in the form of swapping and dropping events corresponds to introducing more deviations between the log and model; consequently, while aligning the model with the log, there will be more *Moves in log only* or *Moves in model only* instead of going through the *synchronous net* (which corresponds to better alignment). Therefore, the alignment will take longer.

iii. Analyzing the obtained plot more accurately, we can see that the alignment takes longer when the fraction of swapped events is higher than the fraction of dropped events, i.e., swapping events imposes a higher impact on the alignment times compared to dropping events. This is also reasonable considering the low-level implementation because dropping the events makes the state space smaller; therefore, it would be easier for algorithms such as A* search to find the shortest

path to the final marking; whereas swapping events introduces more deviations but does not make the state space smaller, thereby increasing the alignment duration.

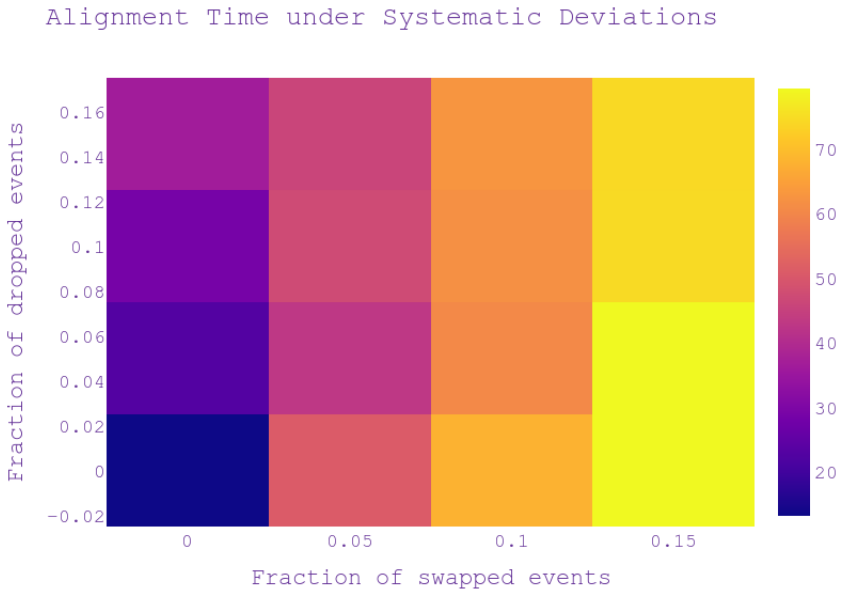


Figure 26: Alignment durations with different levels of deviation

Appendix A: More Transparent Visualizations

Q3 Process Discovery

Figure 27: Model discovered by settings the *paths* slider to 0.2

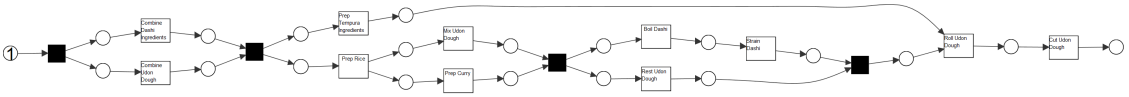


Figure 28: The equivalent Petri net for the model displayed in figure 27

Q4 Conformance Checking

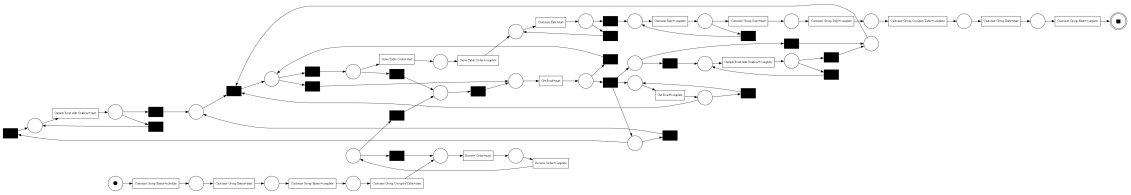


Figure 29: Model discovered by running the *Inductive Miner* with a noise threshold of 0.0

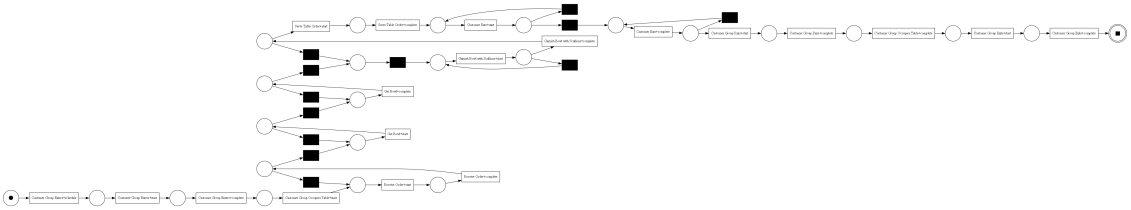


Figure 30: Model discovered by running the *Inductive Miner* with a noise threshold of 0.5

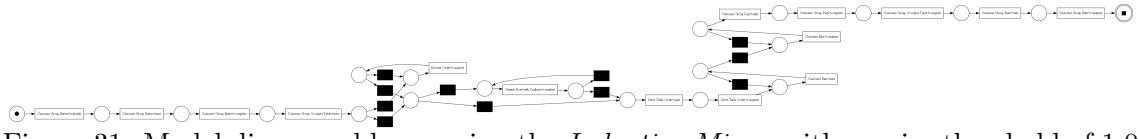


Figure 31: Model discovered by running the *Inductive Miner* with a noise threshold of 1.0

Figure 32: The hand-made Petri-net model represented by *customer_perspective.pnml*

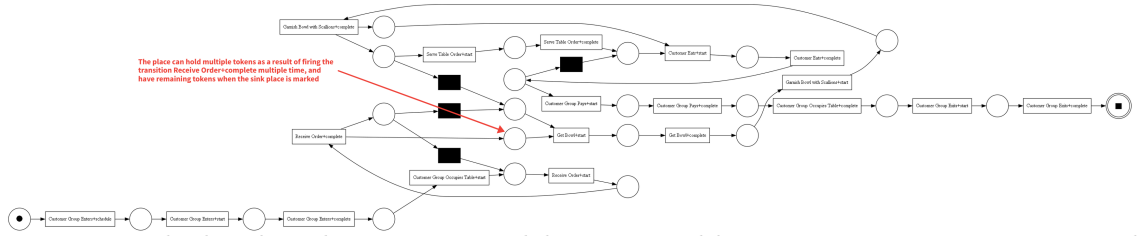


Figure 33: The hand-made Petri-net model represented by *customer_perspective_seq.pnml*

Appendix B: Simplicity Score

Simplicity is the fourth quality criterion to analyze a process model. In this case, the criteria that we use for simplicity is the inverse arc degree.

First of all, we consider the average degree for a place/transition of the Petri net, which is defined as the sum of the number of input arcs and output arcs. If all the places have at least one input arc and output arc, the number is at least 2. Choosing a number k between 0 and infinity, the simplicity based on the inverse arc degree is then defined as:

$$inv\_arc\_deg = \frac{1.0}{1.0 + max(mean\_degree - k, 0)} \tag{1}$$

For more information, you can refer to *PM4PY Documentation*.