BSI - CHALLENGE 2020

Marco Incerti
Università degli Studi di Milano
Matricola 989428
Prof. Paolo Ceravolo

07.2022

1 Scenario

This challenge focuses on studying and transforming the real data that has been provided to us by whatever means possible. In many organizations, staff members travel for work. They travel to customers, to conferences or to project meetings and these travels are sometimes expensive. As an employee of an organization, you do not have to pay for your own travel expenses, but the company takes care of them.

At Eindhoven University of Technology (TU/e), this is no different. The TU/e staff travels a lot to conferences or to other universities for project meetings and/or to meet up with colleagues in the field. And, as many companies, they have procedures in place for arranging the travels as well as for the reimbursement of costs.

On a high level, we distinguish two types of trips, namely domestic and international.

For domestic trips, no prior permission is needed, i.e. an employee can undertake these trips and ask for reimbursement of the costs afterwards.

For international trips, permission is needed from the supervisor. This permission is obtained by filing a travel-permit and this travel permit should be approved before making any arrangements.

To get the costs for a travel reimbursed, a claim is filed. This can be done as soon as costs are actually payed (for example for flights or conference registration fees), or within two months after the trip (for example hotel and food costs which are usually paid on the spot).

1.1 The Process Flow

The various declaration documents (domestic and international declarations, prepaid travel costs and requests for payment) all follow a similar process flow. After submission by the employee, the request is sent for approval the travel administration. If approved, the request is then forwarded to the budget owner and after that to the supervisor. If the budget owner and supervisor are the same person, then only one of the these steps it is taken. In some cases, the director also needs to approve the request.

In all cases, a rejection leads to one of two outcomes. Either the employee resubmits the request, or the employee also rejects the request.

If the approval flow has a positive result, the payment is requested and made.

The travel permits follow a slightly different flow as there is no payment involved. Instead, after all approval steps a trip can take place, indicated with an estimated start and end date. These dates are not exact travel dates, but rather estimated by the employee when the permit request is submitted. The actual travel dates are not recorded in the data, but should be close to the given dates in most cases.

After the end of a trip, an employee receives several reminders to submit a travel declaration.

After a travel permit is approved, but before the trip starts, employees can ask for a reimbursement of prepaid travel costs. Several requests can be submitted independently of each other. After the trip ends, an international declaration can be submitted, although sometimes multiple declarations are seen in specific cases.

It's important to realize that the process described above is the process for 2018. For 2017, there are some differences as this was a pilot year and the process changed slightly on several occasions.

2 Data

The data is split into travel permits and several request types, namely domestic declarations, international declarations, prepaid travel costs and requests for payment, where the latter refers to expenses which should not be related to trips (think about representation costs, hardware purchased for work, etc.).

The data is anonymized in such a way that no TU/e internal IDs are visible in the final dataset, i.e. all identifiers are freshly generated. Furthermore, the amounts mentioned in the data are not exact amounts. However, adding declarations referring to the same travel permit and then comparing them to the original budget/estimate is still possible. Furthermore, on large enough samples of the data, the summed/averaged amounts should be roughly correct.

Staff members cannot be identified in the data. Instead, for all steps, the role of

the person executed the step is recorded. The resource recorded in the data is either the SYSTEM, a STAFF MEMBER or UNKNOWN, or, on occasion, the data is MISSING. Within each file we have logs. Each log consists of all the events that make up the path of that log (of that request).

The available datasets are:

- **Requests for Payment** (should not be travel related): 6,886 cases, 36,796 events.
- **Domestic Declarations**: 10,500 cases, 56,437 events.
- **Prepaid Travel Cost**: 2,099 cases, 18,246 events.
- **International Declarations**: 6,449 cases, 72151 events.
- **Travel Permits** (including all related events of relevant prepaid travel cost declarations and travel declarations): 7,065 cases, 86,581 events.

3 Organization Goal

The goal of my project is to be able to decrease organizational costs through data analysis. Using processes, libraries, and software to try to extract useful information so that we can find an answer to these questions:

- What is the throughput of a travel declaration from submission (or closing) to paying?
- Is there are difference in throughput between national and international trips?
- Where are the bottlenecks in the process of a travel declaration?
- Are there any double payments?
- Create a model to predict whether a trip will be overspent.

4 Knowledge Uplift Model

	Input	Analytics/Model	Type	Output
Step 1	5 event Logs	First analysis on process map with Disco and pm4py	Descriptive	Fields and events explanation
Step 2	5 event Logs	Temporal filtering	Prescriptive	Filtered event logs
Step 3	Step 2	Filtering logs on top 5 variants	Prescriptive	Filtered event logs
Step 4	Step 2	Throughput for payed request	Descriptive	Statistics
Step 5	Step 2 Step 4	Calculated the time lost in each individual activity	Descriptive	Statistics
Step 6	Step 2 Step 4 Step 5	Bottlenecks research	Descriptive	Statistics
Step 7	Step 1 Step 2	Checking double payments	Descriptive	Statistics
Step 8	Step 2 Step 3	Find best models for each log	Prescriptive	Petri net and and frequencies for each log
Step 9	Step 8	Conformance Checking for each model	Descriptive	Anomalous cases for each log
Step 10	Step 2 Step 3	Predicting overspent declarations	Predictive	Predictive model

Figure 1: Knowledge Uplift Model

5 Project Solution

5.1 Process Model and Process Map

First, I calculated the Process Model and Process Map so that I would have an idea of the flow that follows a request. (In the images we see only those from national trips)

Figure 2: Process Model

Through these maps and thanks to the Disco software, I was able to better understand the overall process. I was able to better analyze the different variations and analyze the various statistics. I understood that the process for travel reimbursement starts with "Declaration SUBMITTED by EMPLOYEE" event and most successful refund processes end with "Request Payment" followed by "Payment Handled." Also I noticed that if "Amount" is 0 then the request was rejected.

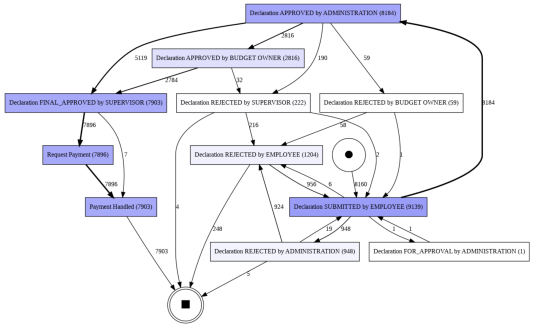


Figure 3: Process Map

5.2 Temporal filtering

Reading the project description, it is noted that the full data is only from 2018, so at this stage I went to apply a time filter so that we only have the full data. You can see in the following tables a decrease in the number of data.

```
data_log_from_2017 = {k:len(data_log[k]) for k in data_log.keys()}
```

```
{'DomesticDeclarations': 10500,
 'InternationalDeclarations': 6449,
 'PermitLog': 7065,
 'PrepaidTravelCost': 2099,
 'RequestForPayment': 6886}
```

```
from pm4py.algo.filtering.log.timestamp import timestamp_filter

data_log_from_2018 = {
    k:timestamp_filter.filter_traces_contained(data_df[k], "2018-01-01 00:00:00", "2022-07-01 23:59:59",
                parameters={timestamp_filter.Parameters.TIMESTAMP_KEY: "time:timestamp"}) for k in data_df.keys()
                }
data_log_from_2018_num = {k:len(data_log_from_2018[k]) for k in data_log_from_2018.keys()}
```

```
{'DomesticDeclarations': 8260,
 'InternationalDeclarations': 4952,
 'PermitLog': 5598,
 'PrepaidTravelCost': 1776,
 'RequestForPayment': 5778}
```

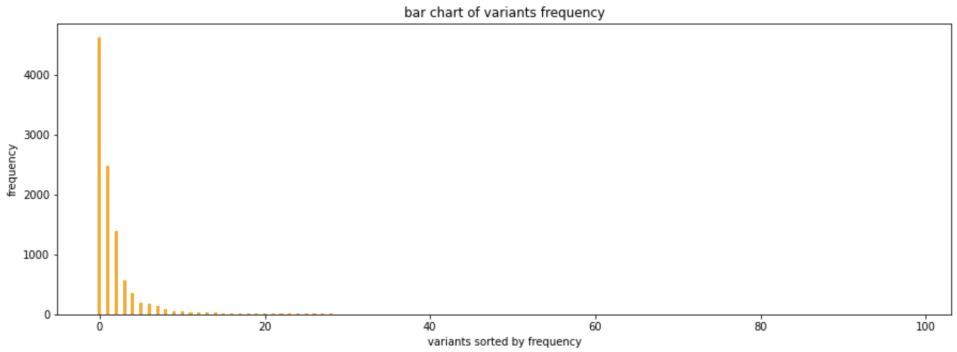
5.3 Filtering logs on top k variants

In this section I have filtered the log files to get the top-k popular variants. Below there is shown the code to perform the filter and the relative coverage percentages with respect to the total traces and the number of traces kept.

```
import pm4py
log_top = {k:pm4py.filter_variants_top_k(data_log[k], 5) for k in data_log.keys()}

for k,v in log_top.items():
    print (f"{k} --> {round(100*(len(v)/len(data_log[k])))}% - {len(v)}/{len(data_log[k])}")
```

DomesticDeclarations –> 90% - 9403/10500
InternationalDeclarations –> 45% - 2919/6449
PermitLog –> 32% - 2271/7065
PrepaidTravelCost –> 57% - 1199/2099
RequestForPayment –> 88% - 6039/6886

In addition, we can see a graphic representation on the distribution of variants. (Only the one of domestic trips so as not to create redundancy)



5.4 Throughtput

At this stage, I calculated the average time over the entire procedure by comparing the duration between domestic and international trips, showing a significant difference between the two results.

```
import pm4py

all_case_durations = {
    k:pm4py.get_all_case_durations(data_log_from_2018[k]) for k in data_df.keys()
                }
max_case_durations = {
    k:sum(all_case_durations[k])/len(all_case_durations[k]) for k in data_df.keys()
                }
```

```
{'DomesticDeclarations': 11.5606353,
 'InternationalDeclarations': 76.35622649,
 'PermitLog': 84.151760905,
 'PrepaidTravelCost': 36.604646802,
 'RequestForPayment': 12.354546698}
```

5.5 More analysis on Throughput

I noticed a big difference between domestic and international flights so I went to do more in-depth analysis. For each case I want to isolate the events between "Declaration SUBMITTED by EMPLOYEE" and "Payment Handled" via this pm4py function (I only show the functions applied to domestic travel to avoid redundancy):

```
data_df = log_to_df(data_log_from_2018["DomesticDeclarations"])
filtered_log = pm4py.filter_between(data_df, "Declaration SUBMITTED by EMPLOYEE", "Payment Handled")
```

Now I have calculated the average duration of each payment request within the two states described above. So we can see that the average duration from request to payment for domestic trips is much shorter than for domestic trips.

```
{'DomesticDeclarations': 11.624,
 'InternationalDeclarations': 15.130}
```

The difference is not much so I decided to go deeper. So I calculated the average lead time for each individual activity that is present between the two phases listed earlier. This allows me to better analyze where there is a large waste of time by increasing the fineness of my analysis.

```
from pm4py.objects.log.util import interval_lifecycle

enriched_log = interval_lifecycle.assign_lead_cycle_time(df_to_log(filtered_log))
enriched_log
data_df = log_to_df(enriched_log)
df_new = data_df[['concept:name', '@@approx_bh_overall_wasted_time']]
df_new.groupby('concept:name', as_index=False)['@@approx_bh_overall_wasted_time'].mean()
```

Result:

concept:name	@@approx_bh_overall_wasted_time
Declaration APPROVED by ADMINISTRATION	0.705
Declaration APPROVED by BUDGET OWNER	1.225
Declaration FINAL_APPROVED by SUPERVISOR	1.507
Declaration FOR_APPROVAL by ADMINISTRATION	5.961
Declaration REJECTED by BUDGET OWNER	1.209
Declaration REJECTED by EMPLOYEE	2.372
Declaration REJECTED by SUPERVISOR	1.591
Declaration SUBMITTED by EMPLOYEE	0.402
Payment Handled	3.559
Request Payment	2.386

For International Declarations (in between there are other phases, but they were not taken to make the comparison with national trips):

concept:name	@@approx_bh_overall_wasted_time
Declaration APPROVED by ADMINISTRATION	1.353
Declaration APPROVED by BUDGET OWNER	2.222
Declaration FINAL_APPROVED by SUPERVISOR	2.520
Declaration FOR_APPROVAL by ADMINISTRATION	NaN
Declaration REJECTED by BUDGET OWNER	3.777
Declaration REJECTED by EMPLOYEE	2.478
Declaration REJECTED by SUPERVISOR	2.405
Declaration SUBMITTED by EMPLOYEE	0.402
Payment Handled	4.615
Request Payment	3.435

5.6 Bottlenecks

I noticed a very large expenditure of time in the final parts of the process. So I decided to isolate other cases, specifically from "Request Payment" to "Payment Handled" and from "Declaration FINAL_APPROVED by SUPERVISOR" to "Request Payment".

"Request Payment" to "Payment Handled"

```
{'DomesticDeclarations': 3.450,
 'InternationalDeclarations': 3.415}
```

"Declaration FINAL_APPROVED by SUPERVISOR" to "Request Payment"

```
{'DomesticDeclarations': 2.842,
 'InternationalDeclarations': 2.856}
```

In order to speed up the process it is interesting to analyze in detail if there are any bottlenecks during the various stages of the process. The code snippet shows the code to identify the 25th percentile, 50th percentile, 75th percentile and the maximum time, expressed in days.

```
d_log = {n:get_logs(f) for n,f in zip(list_name, lis\t_xes)}
for k in d_log.keys():
    list_res = []
    for trace in d\_log[k]:
        start = None
        end = None
        for event in trace:
```

```
            if event['concept:name'] == 'Declaration SUBMITTED by EMPLOYEE':
                start = event['time:timestamp']
            elif event['concept:name'] == 'Payment Handled':
                end = event['time:timestamp']
                break
        if start is not None and end is not None:
            list\_res.append((start, end, trace.attributes['concept:name']))

    print(k,"\n", pd.Series([(end-start).total_seconds()/60/60/24 for (start, end, _) in list_res]).describe()
    [["25%","50%", "75%"]])
```

We can see that the general slowest sequence is Request Payment to Payment Handled. Ignoring the outliers present in the maximums, we can see that the DomesticDeclarations have the same timing as InternationalDeclaration despite the latter, we have a more complicated approval process that also requires the passage of Permit.

```
DomesticDeclarations
 25%      5.919103
50%      7.268623
75%     11.257731
```

```
InternationalDeclarations
 25%      6.182905
50%      9.096863
75%     14.118640
```

```
PermitLog
 25%      6.151730
50%      9.073310
75%     14.109491
```

"Request Payment" to "Payment Handled"

```
DomesticDeclarations
 25%     2.318148
50%     3.240475
75%     4.280729
```

```
InternationalDeclarations
 25%     2.307937
50%     3.218652
75%     4.259711
```

```
PermitLog
 25%     2.310336
50%     3.223032
75%     4.264769
```

5.7 Checking double payments

Now we approach the economic part. We focus on finding solutions that save us money. The fastest and most effective solution is to check that a trip has been paid for only once. So I went to analyze the PermitLogs and found that the organization lost a lot of money in double payments amounting to 5,035,828. In the following image we can see which requests were paid more than once. Reaching up to 17 times!

```
dict_decl = set()
double_time = []
for trace in data_log["PermitLog"]:
  for event in trace:
    if event['concept:name'] == 'Payment Handled':
      dec = trace.attributes['concept:name']
      if dec in dict_decl:
        double_time.append(dec)
      else:
        dict_decl.add(dec)

```

```
#print(data_log["PermitLog"][0])
set_double = list(set(double_time))
res_dict = {k:[] for k in set_double}
for trace in data_log["PermitLog"]:
    if (trace.attributes["concept:name"]) in set_double:
        for event in trace:
            if event['concept:name'] == 'Payment Handled':
                res_dict[trace.attributes["concept:name"]].append(trace.attributes["TotalDeclared"])
                #ogni quando veniva rilasciato un altro pagamento per poi fare una media - poco utile
                #res_dict[trace.attributes["concept:name"]].append(trace.attributes["time:timestamp"])
tmp = []
#print(res_dict)
for k, v in res_dict.items():
    tmp.append([k, round(v[0]), len(v)])
#print(tmp)
df_res = pd.DataFrame(tmp, columns=["ID", "AMOUNT", "TIMES"])
df_res["LOSS"] = df_res["AMOUNT"].to_numpy()*(df_res["TIMES"].to_numpy()-1)
df_res.sort_values("LOSS", ascending=False).head(10)
```

	ID	AMOUNT	TIMES	LOSS
378	travel permit 54518	1458579	2	1458579
1183	travel permit 22776	12232	14	159016
788	travel permit 24054	9246	12	101706
206	travel permit 53343	4811	14	62543
1207	travel permit 73562	3375	17	54000
644	travel permit 57181	4652	9	37216
15	travel permit 33319	6177	6	30885
167	travel permit 45672	6626	5	26504
677	travel permit 66595	5043	6	25215
150	travel permit 28756	4133	7	24798

5.8 Best model for each log

At this stage I tried to find the best templates for each log file. Mainly I used the Process Discovery functions of PM4PY. Process Discovery algorithms want to find a suitable process model that describes the order of events/activities that are executed during a process execution.

- Alpha Miner: The alpha miner is one of the most known Process Discovery algorithm and is able to find:
  - A Petri net model where all the transitions are visible and unique and correspond to classified events (for example, to activities).
  - An initial marking that describes the status of the Petri net model when a execution starts.
  - A final marking that describes the status of the Petri net model when a execution ends.

- Inductive Miner: In PM4Py, we offer an implementation of the inductive miner (IM), of the inductive miner infrequent (IMf), and of the inductive miner directly-follows (IMd) algorithm. The basic idea of Inductive Miner is about detecting a 'cut' in the log (e.g. sequential cut, parallel cut, concurrent cut and loop cut) and then recur on sublogs, which were found applying the cut, until a base case is found. The Directly-Follows variant avoids the recursion on the sublogs but uses the Directly Follows graph. Inductive miner models usually make extensive use of hidden transitions, especially for skipping/looping on a portion of the model. Furthermore, each visible transition has a unique label (there are no transitions in the model that share the same label).

- Heuristic Miner: Heuristics Miner is an algorithm that acts on the Directly-Follows Graph, providing way to handle with noise and to find common constructs (dependency between two activities, AND). The output of the Heuristics Miner is an Heuristics Net, so an object that contains the activities and the relationships between them. The Heuristics Net can be then converted into a Petri net.

I created a function that creates a model for me by type with various parameters, and to figure out the best performing I use 4 metrics:

- Replay Fitness
- Precision
- Generalization
- Simplicity

I did a comparison with the logs before and after 2018 getting better results using the events from 2018 onwards in when they were not compliant before. In the report I show the excellent results.

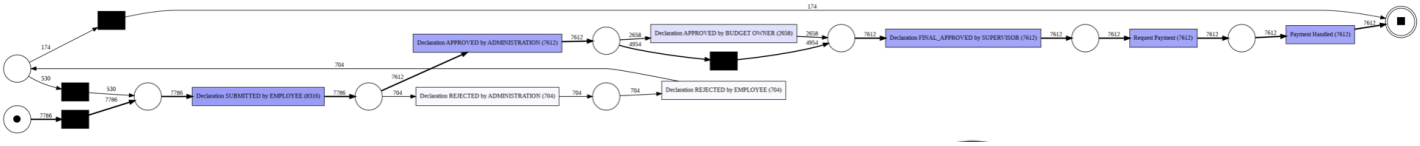
5.9 Domestic Declarations

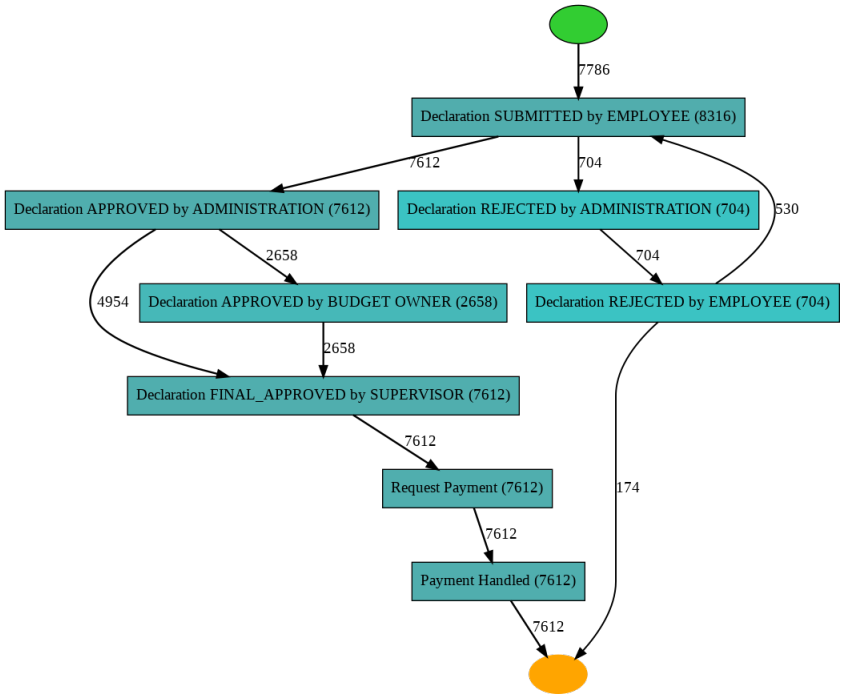
I used my function to find the best model by comparing the metrics used.In this case the best model is provided to us by the Heuristic Miner. In this model we can see employee payment requests can go through or be rejected. Also, we can see how the map is less complex and more linear overall. In this process it is also evident that the approval of the budget owner is optional and for the cases analyzed, about two thirds of the cases do not need this approval.

```
import pm4py
data_log = {
    k:timestamp_filter.filter_traces_contained(data_df[k], "2018-01-01 00:00:00", "2022-07-01 23:59:59",
```

```
    parameters={timestamp_filter.Parameters.TIMESTAMP_KEY: "time:timestamp"}) for k in data_df.keys()
}
d_log_var = {k:pm4py.filter_variants_top_k(data_log[k], 5) for k in data_log.keys()}
res = model_stats(d_log_var['DomesticDeclarations'])
res['SUM'] = res['FITNESS'] + res['PRECISION']
res.sort_values(['SUM',"SEMPLICITY", "GENERALIZATION"], ascending=False).head(5)
```

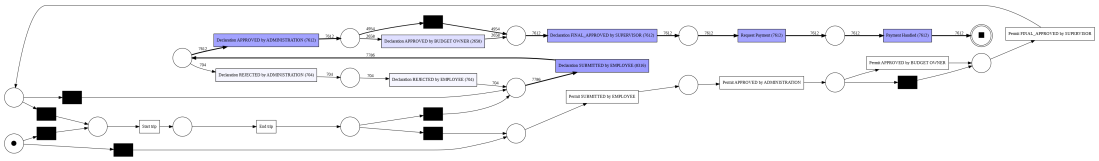
	MODEL	PARAMETERS	FITNESS	PRECISION	GENERALIZATION	SEMPLICITY	SUM
14	HEU	0.1	1.0	0.99	0.98	0.85	1.99
15	HEU	0.2	1.0	0.99	0.98	0.85	1.99
16	HEU	0.3	1.0	0.99	0.98	0.85	1.99
17	HEU	0.4	1.0	0.99	0.98	0.85	1.99
18	HEU	0.5	1.0	0.99	0.98	0.85	1.99





5.10 International Declarations

For all other logs the process is very similar so I will not report all the results in this report. For international trips in this process, we can see that users can request reimbursement before the start of the trip or afterwards. The frequency is also spread out so as to burden only certain steps in the process. Even in this log only one-third of requests go through the budget owner.



5.11 Permit Log

For Permitted logs we use a Heuristic net. The net is a bit more complicated than the previous ones, but still quite simple, very similar to the behavior in InternationalDeclarations with the addition of the Send Reminder event, which loops itself and then ends. Here, too, the Budget Owner in the approvals remains optional.

5.12 Request For Payment

For Request For Payment we use a Heuristic net. Process very similar to Domestic Declaration, it seems to differ only in the prefix of the events that become “Request For Payment”. Also here, the Budget Owner in the approvals remains optional.

5.13 Prepaid Travel Cost

For Prepaid Travel Cost we use an Inductive net. We can see how the map is very linear and in some cases the intervention of some roles can be skipped saving time for sure.

6 Conformance Checking

Conformance checking is a techniques to compare a process model with an event log of the same process. The goal is to check if the event log conforms to the model, and, vice versa. In PM4Py, two fundamental techniques are implemented: token-based replay and alignments.

6.1 Token-based replay

Token-based replay matches a trace and a Petri net model, starting from the initial place, in order to discover which transitions are executed and in which places we have remaining or missing tokens for the given process instance. Token-based replay is useful for Conformance Checking: indeed, a trace is fitting according to the model if, during its execution, the transitions can be fired without the need to insert any missing token. If the reaching of the final marking is imposed, then a trace is fitting if it reaches the final marking without any missing or remaining tokens.

6.2 Alignments

Alignment-based replay aims to find one of the best alignment between the trace and the model. For each trace, the output of an alignment is a list of couples where the first element is an event (of the trace) or » and the second element is a transition (of the model) or. For each couple, the following classification could be provided:

- Sync move: the classification of the event corresponds to the transition label; in this case, both the trace and the model advance in the same way during the replay.
- Move on log: for couples where the second element is », it corresponds to a replay move in the trace that is not mimicked in the model. This kind of move is unfit and signal a deviation between the trace and the model.
- Move on model: for couples where the first element is », it corresponds to a replay move in the model that is not mimicked in the trace. For moves on model, we can have the following distinction:
  - Moves on model involving hidden transitions: in this case, even if it is not a sync move, the move is fit.
  - Moves on model not involving hidden transitions: in this case, the move is unfit and signals a deviation between the trace and the model.

To calculate Conformance values, I created a simple function:

```
def check(log, net, initial_marking, final_marking):
  replayed_traces_token = pm4py.conformance_diagnostics_token_based_replay(log, net, initial_marking, final_marking)
  aligned_traces = pm4py.conformance_diagnostics_alignments(log, net, initial_marking, final_marking)

  df_token = pd.DataFrame(replayed_traces_token)
  df_align = pd.DataFrame(aligned_traces)

  print("REPLAY TRACES TOKEN")
```

```
print("Numeber of traces: " , df_token['trace_is_fit'].count())
print("Numeber of traces fit: ", df_token['trace_is_fit'].value_counts()[True])
print("Numeber of anomalous traces: ",
      (df_token['trace_is_fit'].count() - df_token['trace_is_fit'].value_counts()[True]))
print("Percentage of anomalous traces: ",
      ((df_token['trace_is_fit'].count() - df_token['trace_is_fit'].value_counts()[True]) /
       df_token['trace_is_fit'].count()) * 100)
print("ALIGNED")
print("Numeber of trace: " , df_align['fitness'].count())
print("Numeber of traces fit: ", df_align['fitness'].value_counts()[1])
print("Numeber of anomalous traces: ",
      (df_align['fitness'].count() - df_align['fitness'].value_counts()[1]))
print("Percentage of anomalous traces: ",
      ((df_align['fitness'].count() - df_align['fitness'].value_counts()[1]) /df_align['fitness'].count()) * 100)
```

Some results to avoid redundancy:

```
REPLAY TRACES TOKEN
Numeber of traces:  7786
Numeber of traces fit:  7786
Numeber of anomalous traces:  0
Percentage of anomalous traces:  0.0
ALIGNED
Numeber of trace:  7786
Numeber of traces fit:  7786
Numeber of anomalous traces:  0
Percentage of anomalous traces:  0.0
```

Figure 4: Domestic Declaration

```
REPLAY TRACES TOKEN
Numeber of traces:  2632
Numeber of traces fit:  2632
Numeber of anomalous traces:  0
Percentage of anomalous traces:  0.0
ALIGNED
Numeber of trace:  2632
Numeber of traces fit:  2632
Numeber of anomalous traces:  0
Percentage of anomalous traces:  0.0
```

Figure 5: International Declarations

```
REPLAY TRACES TOKEN
Numeber of traces:  5432
Numeber of traces fit:  5089
Numeber of anomalous traces:  343
Percentage of anomalous traces:  6.314432989690721
ALIGNED
Numeber of trace:  5432
Numeber of traces fit:  5089
Numeber of anomalous traces:  343
Percentage of anomalous traces:  6.314432989690721
```

Figure 6: Request For Payment

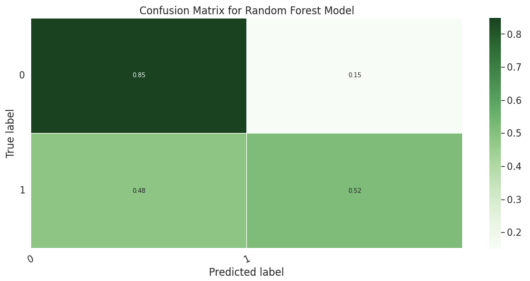
7 Predicting overspent declarations

In the last part of my project, I focused on trying to get a model that could predict to me whether an overspend trip.

I focused on the InternationalDeclarations and PermitLog logs by merging them into one file where I take the data to train the model , with very good results in terms of precision and accuracy. Initially I tested which type of model performed best among GaussianNB and RandomForestClassifier. The results were all very similar but the one with best results was the RandomForestClassifier with a number of estimators equal to 50. Below we can see the ConfusionMatrix with the results.
As we can see from the graphs and the report on the model's evaluation metrics, we can call it a model that can help us in decreasing losses. In the future with more data available the model could improve or perhaps a more complex model could be used to achieve far better solutions.

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

rf = RandomForestClassifier(class_weight="balanced")

#pca = PCA(n_components=5)
#x_train_pca = pca.fit_transform(x_train)
#x_test_pca = pca.fit_transform(x_test)

rf.fit(x_train, y_train)
y_pred = rf.predict(x_test)

print("Test accuracy: ",accuracy_score(y_test, rf.predict(x_test)))
# View the classification report for test data and predictions
print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay(confusion_matrix(y_test, rf.predict(x_test))).plot()

# Get and reshape confusion matrix data
matrix = confusion_matrix(y_test, y_pred)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Greens, linewidths=0.2)

# Add labels to the plot
class_names = ['0', '1']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
```

```
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```




```
Test accuracy:  0.7461139896373057
              precision    recall  f1-score   support

       False       0.80      0.86      0.83       270
        True       0.59      0.49      0.54       116

    accuracy                           0.75       386
   macro avg       0.70      0.67      0.68       386
weighted avg       0.74      0.75      0.74       386
```

As described in the confusion map we can see that if the true negatives are very reliable and on the other hand the true positives are not and therefore there may be more complicated considerations to be made.

8 Technologies used

The technologies I used for my project include:

- Google Colab: used to develop my analysis and solutions.
- PM4PY: python library that contains the functions used for analysis. All the functions used can be found in its documentation.
- Disco: Disco is a great process mining tool that simply works: it is able to deal with large event logs and complex models and conversion and filtering are made easy. Performance metrics are shown in a direct and intuitive manner and the history can be animated on the model.

Conclusion

In this report, I conducted analyses on several aspects. First, I tried to figure out what the most common variants were (also helping me with Disco) and patterns to help me and to analyze the log workflow. Here I found that some requests take a long time to complete. Also, the average time is not high and there is not a big difference between different trip types. One solution I have come up with to decrease the wasted time may be to decrease the steps within an event, so as to standardize the process making it smoother and faster. Also alerts are essential to ensure that the process goes on.
The second analysis I did is on the money lost. As you can see in the report, I found that many claims are paid multiple times, causing the organization to lose a lot of money. So you need software that keeps this information up to date so that it is no longer allowed.
Last Analysis is about creating a predictive model to estimate whether a trip will need new logs. It is not super accurate, but with more data and perhaps a more complex learning method for regression we can achieve good results.