Inductive Miner.

PM4Py offer an implementation of the inductive miner (IM), of the inductive miner infrequent (IMf), and of the inductive miner directly-follows (IMd) algorithm. The papers describing the approaches are the following:

- Inductive Miner: Discovering block-structured process models from event logs-a constructive approach
- Inductive Miner infrequent: Discovering block-structured process models from event logs containing infrequent behaviour
- Inductive Miner directly-follows Scalable process discovery with guarantees

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 on the model. Furthermore, each visible transition has a unique label (there are no transitions in the model that share the same label).
Two process models can be derived: Petri Net and Process Tree.

```
import os
from pm4py.objects.log.importer.xes import importer as xes_importer
from pm4py.algo.discovery.inductive import algorithm as inductive_miner

log = xes_importer.apply(os.path.join("tests","input_data","running-exam
net, initial_marking, final_marking = inductive_miner.apply(log)
```

Figure 1

```
from pm4py.algo.discovery.inductive import algorithm as inductive_miner
from pm4py.visualization.process_tree import visualizer as pt_visualizer

tree = inductive_miner.apply_tree(log)

gviz = pt_visualizer.apply(tree)
pt_visualizer.view(gviz)
```

Figure 2

To mine a Petri Net, we provide an example. A log is read, the inductive miner is applied and the Petri net along with the initial and the final marking are found. The log we take as input is the running-example.xes. First, the log is read, then the inductive miner algorithm is applied.

```
from pm4py.objects.conversion.process_tree import converter as pt_conver
net, initial_marking, final_marking = pt_converter.apply(tree, variant=p
```

Figure 3

It is also possible to convert a process tree into a petri net.

Tee available variants inside PM4Py are three

Variant	Description
variant.IM	Produces a model with perfect replay fitness.
variant.IMf	Produces a more precise model, without fitness guarantees, by eliminating some behavior.
variant.IMd	A variant of inductive miner that considers only the directly-follows graph, for maximum performance. However, replay fitness guarantees are lost.