The KNIME Model Factory

Scaling Modeling Processes for the Enterprise

Iris Adä
Iris.Adae@KNIME.com

Phil Winters
Phil.Winters@KNIME.com

Michael Berthold

Michael.Berthold@KNIME.com

Summary

In this white paper, we present and full document the KNIME Model Factory, designed to provide you with a flexible, extensible and scalable application for running very large numbers of model processes in an efficient way. The KNIME Model Factory is composed of an overall workflow, tables that manage all activates and a series of workflows and data for learning.

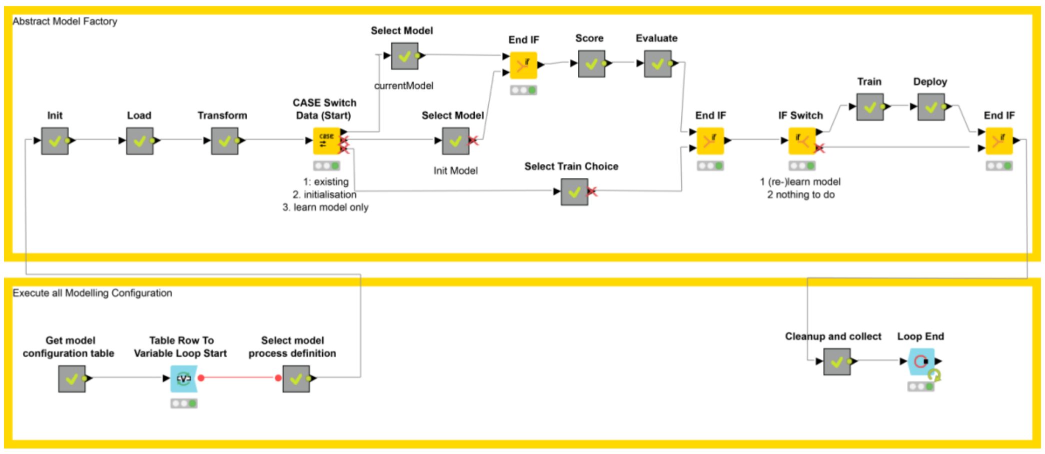


Figure 1.

Workflow for model process management

The KNIME Model Factory is available via the KNIME examples server and can run on the KNIME Analytics Platform, which means it is open source and free. Major benefits can be realized in terms of automation and interfacing by using the KNIME Model Factory with KNIME Server.

All workflows, tables and sample datasets are available for download from the EXAMPLES Server under **50_Applications/ 26_Model_Process_Management**

Table of Contents

Background and the Business Challenge

The benefits of using predictive analytics is now a given. In addition, the Data Scientist who does that is highly regarded but our daily work is full of contrasts. On the one hand, you can work with data, tools and techniques to really dive in and understand data and what it can do for you. On the other hand, there is usually quite a bit of administrative work around accessing data, massaging data and then putting that new insight into production - and keeping it there.

In fact, many surveys say that at least 80% of any data science project is associated with those administrative tasks. One popular urban legend says that, within a commercial organization trying to leverage analytics, the full time job of one data scientist can be described as building and maintaining a maximum of four (yes 4) models in production - regardless of the brilliance of the toolset used. There is a desperate need to automate and scale the modelling process, not just because it would be good for business (after all, if you could use 29000 models instead of just 4, you would want to!) but also because otherwise we data scientists are in for a tedious life.

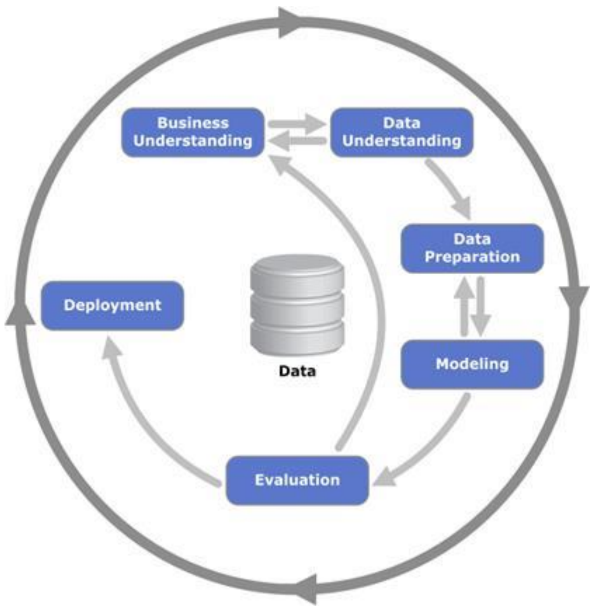


Figure 2.

CRISP diagram for explaining the modelling process.

A good description of the modelling process comes from CRISP. That describes very well the process we manually walk through in doing data science to apply to a particular problem or focus area. If you think of CRISP as a process, it would look like this:

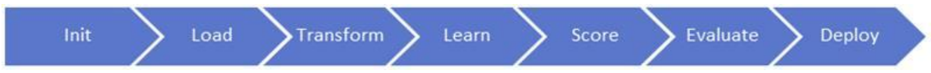


Figure 3.

CRISP diagram as a straightline process

If we are going to scale to have many processes, it is these steps we must concentrate on automating and making extremely efficient. In the market today are a number of attempts at "model management" that fall into three categories:

In KNIME, the concept of a process is inherent to its design: everything is a flow of tasks and the model process in its simplest form in KNIME might look like this:

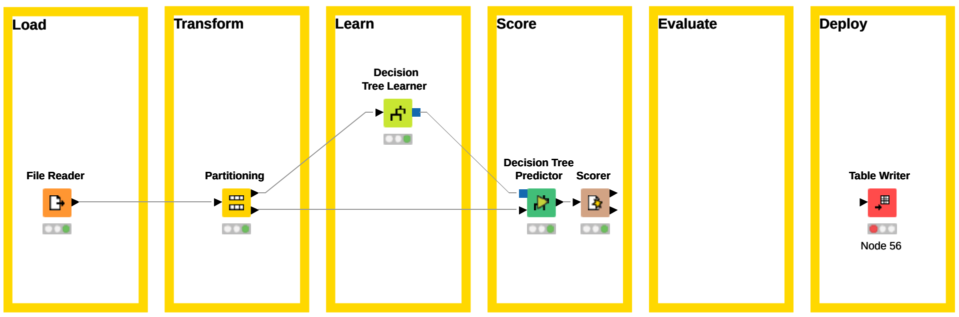


Figure 4.

Simplest example of a model process in KNIME

And as any KNIME user knows, we can make this as robust as we like, either by coding sequences of nodes or by collapsing that complexness/robustness into Metanodes:

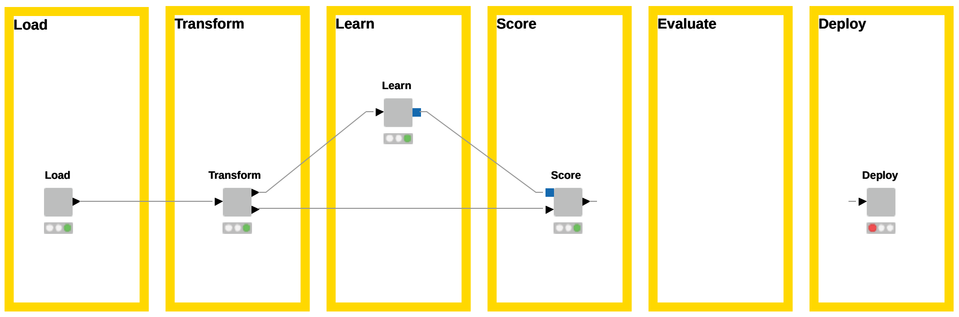


Figure 5.

Robust example of a model process in KNIME

But one of the major challenges of ANY package (including KNIME) is that step where we evaluate then decide our modeling process is good then deploy that model in some way so that it can be used.

As a first step, the evaluation process is always a manual one as we carefully evaluate our work and assumptions before coming to a conclusion. When we move our model into production by deploying it, we have the additional challenge that a model may deteriate over time. By default, we then set up to manually recheck the model occasionally and - if it falls below some threshold - we can retrain a new model to take its place.

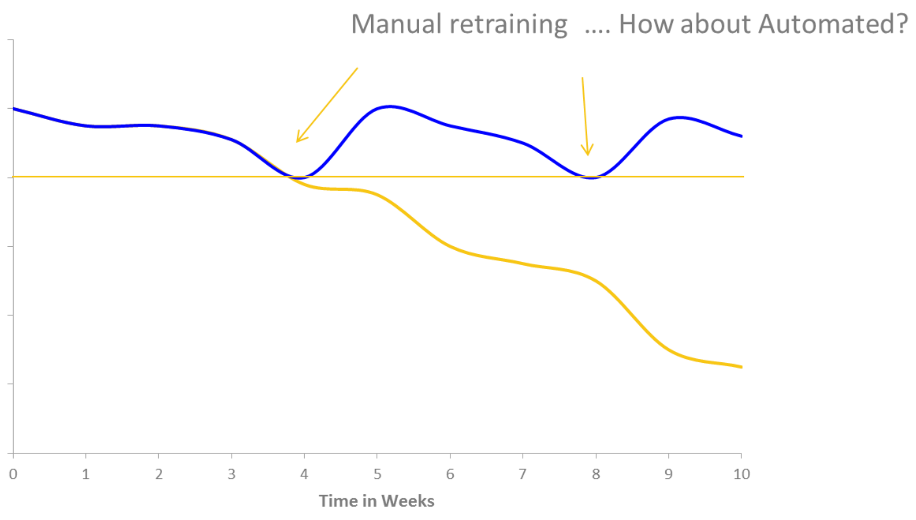


Figure 6.

A model can deteriate over time. Normally, one would manually retrain. But we can use KNIME to automatically retrain.

However, we can use the power of KNIME to automate the checking and retraining of the model. That might look like this:

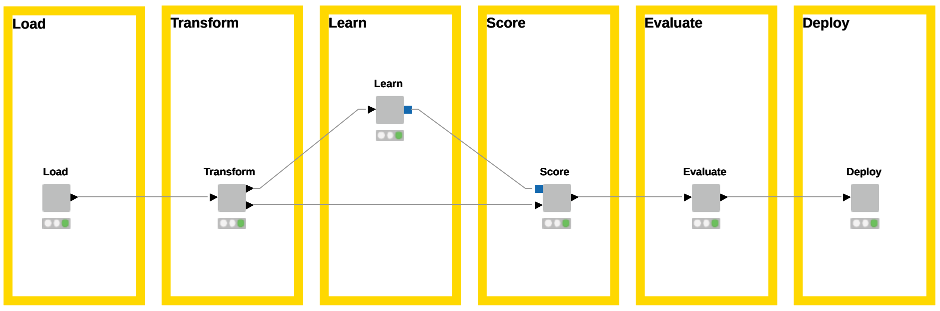


Figure 7.

Example of automated retraining of a model in KNIME.

KNIME as Metadata and KNIME Model Factory Concepts

Up until now, we have looked at one workflow that represents one modeling process. The KNIME platform itself has something far more important to offer for model management: the fact that KNIME is script-free means that everything in KNIME is defined and driven by metadata. From nodes and metanodes through workflow control and on to decisions about workflow execution and notification, absolutely everything in KNIME can be represented and controlled via its metadata.

That means we can extrapolate our simple model process workflow away from the GUI visual of the workflow and instead represent it as a table of metadata that describes that exact same workflow. And, of course, if we can do it once we can do it for multiple model processes, which give us the ability to manage and control everything via a workflow that manages such a metadata table. That is the basis for the KNIME Model Factory.

Before diving in to the KNIME model factory itself, there are some key KNIME Model Factory concepts to understand:

PROCESS STEPS: The process steps are the smallest logical pieces that need to be defined for a complete abstract modelling process to occur. In KNIME, we define each process step as a specific standalone KNIME Workflow. Each of those well-defined KNIME workflows knows how to work with each of the other process steps. The seven process steps are:

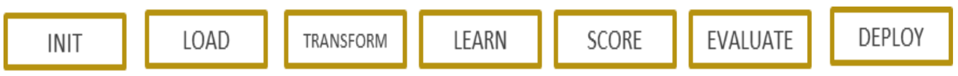


Figure 8.

The process steps of an abstract modelling process. In KNIME, these would each be a standalone KNIME workflow.

These PROCESS STEPS are associated with specific tasks in the abstract model process. A brief description of each PROCESS STEP is provided here:

INIT - defines the variable parameters that would be required for the other process steps
LOAD - takes information from INIT and know where and how to load data for further processing.
TRANSFORM - Not only manipulate, transform and blend data but would also set up training and testing data.
LEARN - Does the actual training of one (or more) models
SCORE - Scores the models created in LEARN on the testing data
EVAULATE- Makes the decision of whether a particular model satisfies tolerance criteria
DEPLOY - in its simplest form, does nothing since the process of storing the model has already occurred within the Model Factory. But in this step, we can also setup or change the deployment method.

PROCESS DEFINITION: is the formal stored definition of the 7 PROCESS STEPS (and the associated KNIME Workflows) that would define a complete model process. Every PROCESS DEFINITION will always have all 7 PROCESS STEPS (even if for some modelling processes some of the steps are in theory empty workflows that perform nothing).

There would be one PROCESS DEFINITION for each type of analytic problem. For example, if you have three business topics, there would be three PROCESS DEFINITIONS:

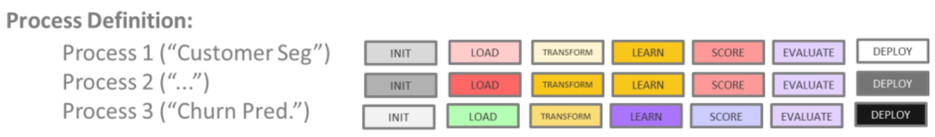
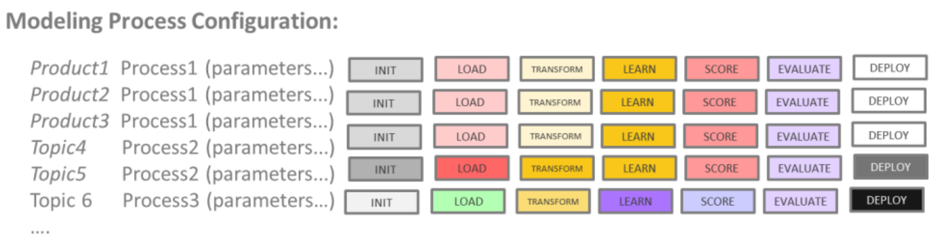


Figure . 9

There will be one PROCESS DEFINITION for each type of analytic problem.

Note that a very important concept is that a PROCESS DEFINITION can reuse a PROCESS STEP. In the example above, all processes use the same EVALUATE step, however they differ in the other steps. We have now defined processes and the workflows associated with them but they are still not attached to the data and the task to be done.

MODEL PROCESS CONFIGURATION: this is where you associate PROCESS DEFINITIONs with concrete business topics, specific data and any other necessary parameters. You might have many products that used one PROCESS DEFINITION. There would then be one model Process Configuration for each product/process combination:


Modeling Process Configuration:

Product1	Process1 (parameters...)	INIT	LOAD	TRANSFORM	LEARN	SCORE	EVALUATE	DEPLOY
Product2	Process1 (parameters...)	INIT	LOAD	TRANSFORM	LEARN	SCORE	EVALUATE	DEPLOY
Product3	Process1 (parameters...)	INIT	LOAD	TRANSFORM	LEARN	SCORE	EVALUATE	DEPLOY
Topic4	Process2 (parameters...)	INIT	LOAD	TRANSFORM	LEARN	SCORE	EVALUATE	DEPLOY
Topic5	Process2 (parameters...)	INIT	LOAD	TRANSFORM	LEARN	SCORE	EVALUATE	DEPLOY
Topic 6	Process3 (parameters...)	INIT	LOAD	TRANSFORM	LEARN	SCORE	EVALUATE	DEPLOY
....								

Figure 10.

A Process Configuration will associate a concrete business topic to a Model Process Definition.

MODEL: A model is the structure of the analysis result created and then store from within our complete model process. Within KNIME, a model is just another type of data that can also be defined by its metadata. A model is stored and available to other relevant PROCESS STEPS.

With these basic definitions, we can now describe in detail the KNIME Model Factory.

The KNIME Model Factory

Overall description

The KNIME Model Factory is a package of workflows, tables, templates and definitions that, when combined with KNIME and in particular KNIME Server, provides an extremely robust and flexible yet highly scalable environment for model process management. It assumes that in a constantly running steady state:

- data will constantly be changing
- many existing models will need to be tested against more up-to-date data,

- new models processes will need to be added possibly even for focus areas where there is not enough data to model
- Processed will need to be automatic, but also have the ability to be manually started and
- Existing model processes will need to be occasionally challenged by a new model process

The KNIME Model Factory takes care of all these cases. The KNIME Model Factory will work with KNIME Analytic Platform on the desktop. The examples we have prepared will all work on the desktop. This give the KNIME user the capability to learn, test and expand on the concepts.

But the real power of the KNIME Model Factory comes when it is combined with the KNIME Server, which provides the sharing, scheduling, interaction (via the WebPortal) and automation capabilities necessary for a robust and scalable production environment.

Storing and Editing Metadata information

At the heart of the model factory are two tables that store the metadata information described earlier. In addition, two tables are automatically generated for capturing all models changes over time as well as all evaluation statistics over time.

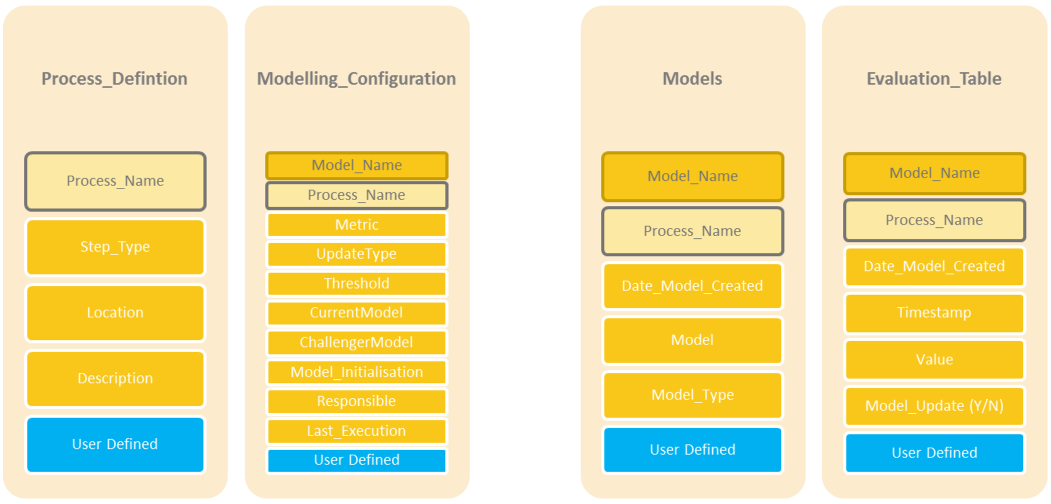


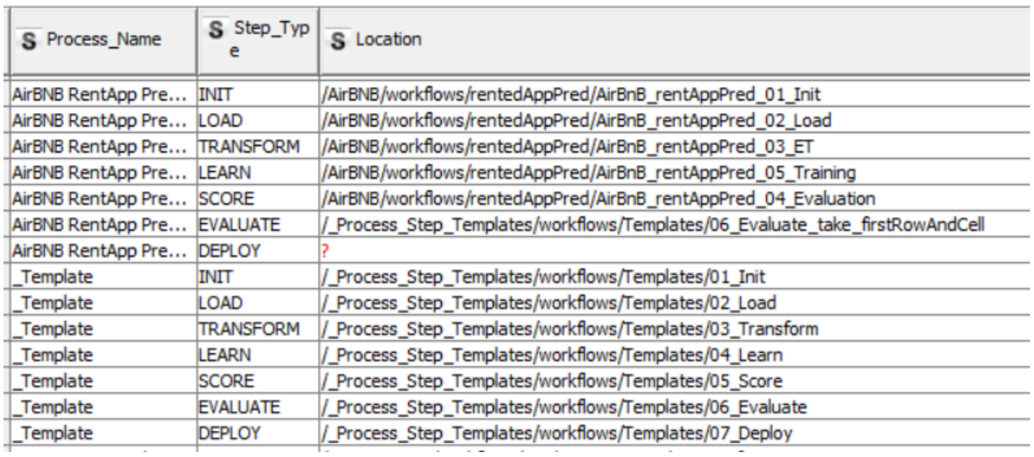
Figure 11.

The four tables at the heart of the model factory.

In the example model factory, all 4 tables are saved in KNIME Tables within the repository. You could alternately save these tables to a database. In that case, you would simply modify the Model Factory workflow to refer to those alternate tables.

Process_Definition Table

The first table is the Process_Definition table, which contains all information necessary to define a process definition. As already discussed, a process is defined by its steps, which are workflows identified with a specific process step. To define a process, you first define a Process name. You then need to define the 7 process steps that make up that process definition. Each process step will be one line in you process_defintion table. Below you see two processes definitions that are defined in the process_definition table:



S Process_Name	S Step_Type	S Location
AirBNB RentApp Pre...	INIT	/AirBNB/workflows/rentedAppPred/AirBnB_rentAppPred_01_Init
AirBNB RentApp Pre...	LOAD	/AirBNB/workflows/rentedAppPred/AirBnB_rentAppPred_02_Load
AirBNB RentApp Pre...	TRANSFORM	/AirBNB/workflows/rentedAppPred/AirBnB_rentAppPred_03_ET
AirBNB RentApp Pre...	LEARN	/AirBNB/workflows/rentedAppPred/AirBnB_rentAppPred_05_Training
AirBNB RentApp Pre...	SCORE	/AirBNB/workflows/rentedAppPred/AirBnB_rentAppPred_04_Evaluation
AirBNB RentApp Pre...	EVALUATE	/_Process_Step_Templates/workflows/Templates/06_Evaluate_take_firstRowAndCell
AirBNB RentApp Pre...	DEPLOY	?
_Template	INIT	/_Process_Step_Templates/workflows/Templates/01_Init
_Template	LOAD	/_Process_Step_Templates/workflows/Templates/02_Load
_Template	TRANSFORM	/_Process_Step_Templates/workflows/Templates/03_Transform
_Template	LEARN	/_Process_Step_Templates/workflows/Templates/04_Learn
_Template	SCORE	/_Process_Step_Templates/workflows/Templates/05_Score
_Template	EVALUATE	/_Process_Step_Templates/workflows/Templates/06_Evaluate
_Template	DEPLOY	/_Process_Step_Templates/workflows/Templates/07_Deploy

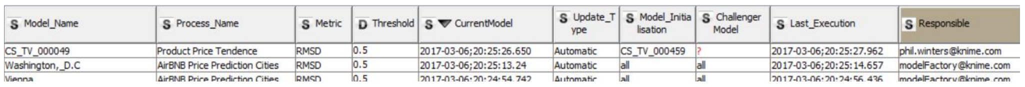
Figure 12.

Example process definition table

The seven steps are: INIT, LOAD, TRANSFORM, LEARN, SCORE, EVALUATE and DEPLOY. We will later explain how the steps are used and can be created using standard templates. For each step, we need a location where we find the workflows that will be executed. For a locally running Model Factory, this can simply point to workflow locations. If you are using the Model Factory in production, you will want to use workflow related references so that the workflows are always found irrespective of the physical location. This is how our model factory is already built, everything is done in relation to the workflows. Hence, you can freely shift this around. The table allows for a Description as well as User Defined Parameters. These are all optional.

Modelling_Configuration Table

The modelling processes we have defined now need to be configured for execution on specific data and topics. We do this in the Modelling_Configuration Table. This is where we define the parameters surrounding the actual modelling process. Below you see entries for three model configurations. Notice that two of the configurations use the same model process:



S Model_Name	S Process_Name	S Metric	D Threshold	S CurrentModel	S Update_Type	S Model_Initialisation	S Challenger Model	S Last_Execution	S Responsible
CS_TV_000049	Product Price Tendence	RMSD	0.5	2017-03-06;20:25:26.650	Automatic	CS_TV_000459	?	2017-03-06;20:25:27.962	phil.winters@knime.com
Washington,_D.C	AirBNB Price Prediction Cities	RMSD	0.5	2017-03-06;20:25:13.24	Automatic	all	all	2017-03-06;20:25:14.657	modelFactory@knime.com
Vienna	AirBNB Price Prediction Cities	RMSD	0.5	2017-03-06;20:24:54.742	Automatic	all	all	2017-03-06;20:24:56.436	modelFactory@knime.com

Figure 13.

Example model configuration table.

Model_NAME defines the specific modelling configuration. This will always point to the specific item, product, topic or area of focus for this modelling process sequence. A best practice is to use this name to point to something specific within your data such as the specific target value or segment you will be focusing on. In this way, you can write your INIT workflows to automatically find prepare for selecting the correct data for this specific topic.

Process_Name points to a Model Process that has already been defined in the first table.

Metric tells you which metric is used for evaluation – this is for documentation only, please make sure this is represented by your EVALUATE step.

Threshold specifies the accuracy under which the model factory will do a model retraining (by default, our example is set up for thresholds below this value).

Current Model is a link to a model in the models table. The name of the current model is always a timestamp, so you can directly see when the model was learned. The current model as well as the Last Execution is updated within the model factory.

Update_Type can be Automatic, Manual, Challenger, Single or missing.

- Automatic means that the model configuration will run automatically every time the model factory is executed.
- If Manual is chosen, only the evaluation of the model will be done, but the model will not be retrained. You can schedule a single retrain by changing the state to Manual_Retrain. You want to choose manual if you only want to monitor the performance of the model.
- If Manual_Retrain is chosen, we will do exactly one retrain of the data and afterward set the state back to Manual. This retrain will be done independently of the evaluation.

- The Challenger type is similar to Manual in that it is not done automatically and must be manually executed. However, if the challenger execution of the process creates a new model, it will be saved but will not replace a Current Model automatically by the model factory. We will return to this special case later.
- If there is no Update_Type defined, hence, it is a missing cell; this modelling configuration will be skipped by the model factory. This allows you to build model configurations and save them within the model factory, even if they are not being executed

Model_Initialisation can be used to point to another Model_Name for initialization (i.e.: deciding what data to use, what to focus on, what parameters to set, etc.) if it is not yet possible to create a Model - this can happen when a new product or item appears but there is not yet enough data for training/testing. In this case, the referenced Model_Name modelling configuration will be executed. That modelling configuration must exist already, as the model will be taken from it.

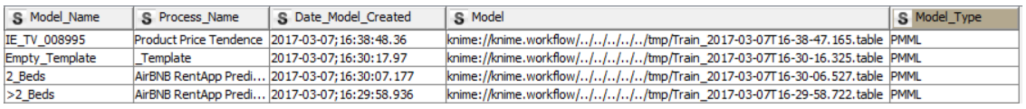
Challenger_Model points to another existing model process configuration of type CHALLENGER, which can be compared using the same model_name to the current process.

Last_Execution documents when the process configuration was last executed.

Responsible should be an email address of the owner of the particular process configuration.

Model and Evaluation Tables

The last two tables are fully generated during the execution of the model factory. The first one is the models table.



S Model_Name	S Process_Name	S Date_Model_Created	S Model	S Model_Type
IE_TV_008995	Product Price Tendence	2017-03-07;16:38:48.36	knime://knime.workflow/../../../../../tmp/Train_2017-03-07T16-38-47.165.table	PMML
Empty_Template	_Template	2017-03-07;16:30:17.97	knime://knime.workflow/../../../../../tmp/Train_2017-03-07T16-30-16.325.table	PMML
2_Beds	AirBNB RentApp Predi...	2017-03-07;16:30:07.177	knime://knime.workflow/../../../../../tmp/Train_2017-03-07T16-30-06.527.table	PMML
>2_Beds	AirBNB RentApp Predi...	2017-03-07;16:29:58.936	knime://knime.workflow/../../../../../tmp/Train_2017-03-07T16-29-58.722.table	PMML

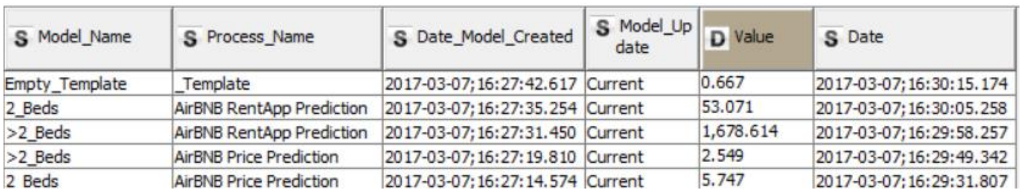
Figure 14.

Example model table.

It contains a list of all models - both current and historical - which were saved during the execution of the model factory. The model is marked by the model_name / process_name configuration from the modelling_configuration table together with the date it was created

(Date_Model_Created). In addition, we save a link to the actual model (Model) and the Model_Type (KNIME or PMML).

The Last Table is the Evaluation Table.



S Model_Name	S Process_Name	S Date_Model_Created	S Model_Update	D Value	S Date
Empty_Template	_Template	2017-03-07;16:27:42.617	Current	0.667	2017-03-07;16:30:15.174
2_Beds	AirBNB RentApp Prediction	2017-03-07;16:27:35.254	Current	53.071	2017-03-07;16:30:05.258
>2_Beds	AirBNB RentApp Prediction	2017-03-07;16:27:31.450	Current	1,678.614	2017-03-07;16:29:58.257
>2_Beds	AirBNB Price Prediction	2017-03-07;16:27:19.810	Current	2.549	2017-03-07;16:29:49.342
2_Beds	AirBNB Price Prediction	2017-03-07;16:27:14.574	Current	5.747	2017-03-07;16:29:31.807

Figure 15.

Example evaluation table

It is fully maintained and filled by the model factory. It is filled each time the evaluate node is executed. It will tell you to which modelling configuration it belongs. And you can find the model by using the Date_Model_Created parameter. There is also a timestamp, when this value was evaluated and of course the value itself.

All tables can be expanded with your own user defined fields but the fields that are present should be left as-is to ensure correct execution of the Model Factory.

The KNIME Model Factory

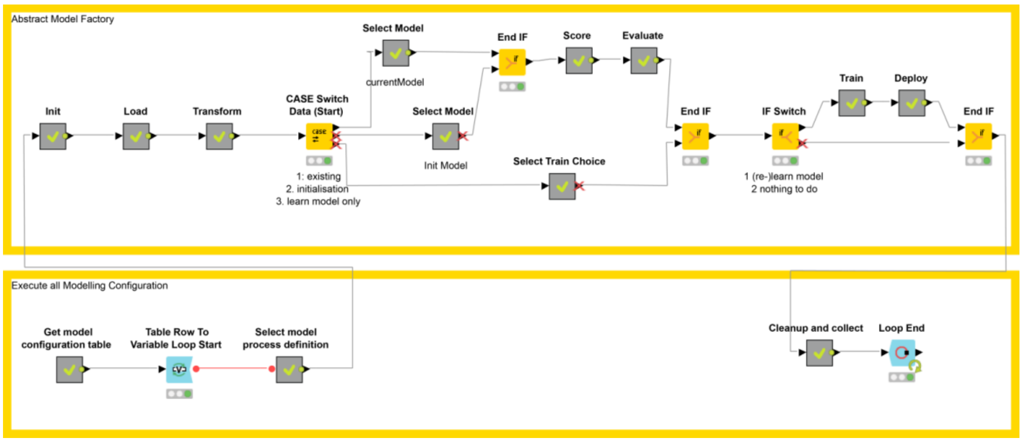


Figure 16.

The KNIME Model Factory workflow

The model factory is comprised of a KNIME workflow that provides two specific activities. One part is a workflow that models a generic modeling process with its 7 steps. The second part is a loop that reads the model configuration table and loops over the abstract modelling workflow part to execute a complete modelling process for each row in the model configuration table.

In the screenshot above, the looping is done in the lower part of the workflow. We read the model configuration table and use the Table

Row to Variable Loop Start node to start the loop. The loop will process each row separately.

The Abstract Model Factory will only be run for those modelling configuration with a non-missing status.

The loop is closed on the right side of the workflow. Before closing the loop, we clean up some temporary tables produces by the Abstract Model Factory.

Between the loop start and end is contained the Abstract Model Factory portion of the workflow, which is contained in the upper part of the workflow. The Abstract Model Factory is the heart of our modelling process. We call it Abstract because it in and of itself cannot run a modeling process- it simply defines the process. But when fed with the information from one row of the model configuration table, it loads and executes once across the 7 process steps we previously defined. In addition to executing the modeling process, it captures and maintains met information in the additional MODEL and EVALUATION tables we previously described.

Overview of the Abstract Modeling Process Workflow

We will describe the KNIME process step workflows in detail in a moment. At the moment, it is important only to remember that each process step loads a physical workflow via the model configuration and by extension the model process table.

When the Abstract Model Factory is started, at first the INIT node is executed. The INIT is used to initialize parameters. For example, you can set up the INIT workflow to initialize a pointer to the file you want to read. You than might have three different INIT workflows, pointing to three different file versions. Alternatively, you could point an INIT at a specific target variable and a second INIT at another target variable. The LOAD can then always be the same.

The LOAD step now gets as input all parameters provided by the INIT Step. The LOAD is used to load and select the data. The result is written into a temporary file, the file names are sent to the output and it is called filePath.

The TRANSFORM workflow is the third in row. It takes the data from the LOAD workflow, applies filters or does any other kind of preprocessing required to prepare the data for scoring and/or modelling. As a last and required step, the data is split into training and test set. This information is send back to the model factory.

Now we reach the first automatic decision. As a natural next step, we would like to SCORE the possibly updated test data on our model, which was learned in previous iterations. If this is the case, we take the top port of the Case Switch node.

However, if there is currently no model in the modelling configuration for us to use, we can use an initialization model, for example from a similar product or a generic model where we DO have enough data. This will be selected in the second path of the Case Switch. This model initialization is defined in the modelling configuration table. Please note that this have to have a modelling configuration with the same Process_Name, which has a currentModel assigned.

For both paths, we have then selected an existing model. This model is then sent together with the test data set path to the SCORE step. The SCORE steps applies the model to the test data set and sends the scored data back to the factory.

After scoring we use the EVALUATE step to evaluate the performance of our model THRESHOLD defined in the Model Configuration table for this model process. The EVALUATE will return a value for the chosen metric. This is the key for deciding if the model will be retrained. It is also within this metanode were we make the decision whether, based on the METRIC value and the THRESHOLD, we retrain the model or not. In this abstract model factory, we are retraining if the value generated by the EVALUATE step is smaller than the threshold saved in the modeling configuration. The evaluate step also saves the new evaluated value into the Evaluation table.

Based on that decision, we either take the top portion of the IF node when we retrain. The LEARN steps gets the training data. If the learning is successful, the DEPLOY metanode will take care to write the model into the models table. And it will also update the current model in the modelling configuration table. If we do NOT decide to retrain, the LEARN and DEPLOY metanodes are not executed.

Finally, as a last step all temporary tables are deleted and we update the last execution timestamp of the modelling configuration. We have now completed one actual execution of a complete model process.

The KNIME Process Step Workflows

The 7 process step workflows follow a specific protocol aimed at keeping them extremely flexible but at the same time working within the Model Factory. To ensure the workflow communicate with each other, we use JSON as the interface. That means every process step workflow gets an input JSON and will produce an output JSON. This JSON will contain all parameters for the workflow to run.

As each workflow needs an input, please note how the inputs are generated for each of the 7 Workflows:

Step_Type	**Input**	**Output**
INIT	All values from the respective modelling configuration	Values the LOAD Workflow needs to read the file (e.g. a file path or a filter criteria)
LOAD	Everything provided by INIT	*filePath* (to the loaded file)
TRANSFORM	filePath (from LOAD)	filePath_Train (can be missing) filePath_Test
LEARN	filePath_Train (from LOAD)	modelPath
SCORE	filePath_Test (from TRANSFORM) modelPath (from models table or LEARN)	filePath
EVALUATE	filePath (from SCORE)	value
DEPLOY	modelPath (from LEARN)	-

Figure 17.

The 7 process step workflows and their input and output definitions.

Each of the 7 process step metanodes described earlier knows how to call workflows designed with the JSON protocol. Each metanode sets up the INPUT parameters for each process step and then calls the workflow via a Call Local Workflow node. At the end, the input and output of each process steps is updated in the table. The output will be reused in consequent steps, as described in the table above.

Note that while the sample Model Factory uses Call Local Workflows (which will also work well on KNIME Server), it would be possible to also swap these out for Call Remote Workflows.

Making Process Steps (workflows)

To make your own process steps to create your own model process definitions, we would highly recommend that you use the templates provided as they set up the appropriate JSON connections for you and the other connections and basic logic to the model factory so you can fully focus on your analysis. In the following description, we introduce each template, show how you can edit them and show how they can be effectively "skipped" if they are not required.

In all of the following graphics, the part in gray is where you should be adding your own logic, analysis and workflow sequences to achieve what you need to achieve within that step. Leave the yellow pieces as-is to ensure your new process step will work within the model factory.

1. INIT

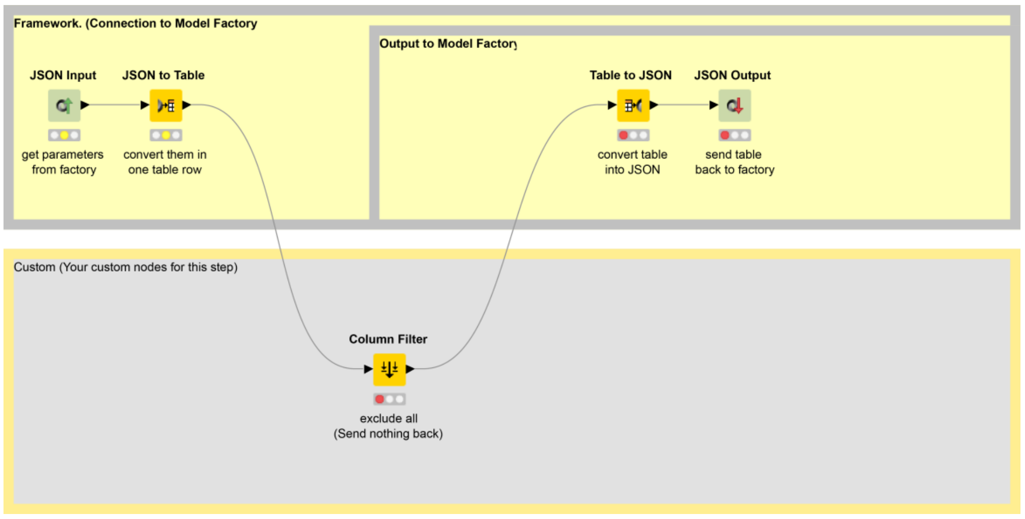

Framework. (Connection to Model Factory
Output to Model Factory
JSON Input
JSON to Table
get parameters from factory
convert them in one table row
Table to JSON
JSON Output
convert table into JSON
send table back to factory
Custom (Your custom nodes for this step)
Column Filter
exclude all (Send nothing back)

Figure 18.

INIT workflow template

The INIT workflow is always the first in row. It is the only one which will get parameters from the modelling configuration table. You would typically use it to initialize parameters for the LOAD workflow. For example, imagine you have two different data sets; both can be loaded using the same methods. Than you could have two INIT workflow, providing the LOAD workflow with the path of the file only. The LOAD will take care of reading and processing the file. So you can use in your process the same LOAD and only different INIT workflows. The INIT is not a mandatory workflow. You can skip it. However, then your LOAD Step does need to know what you are loading.

2. LOAD

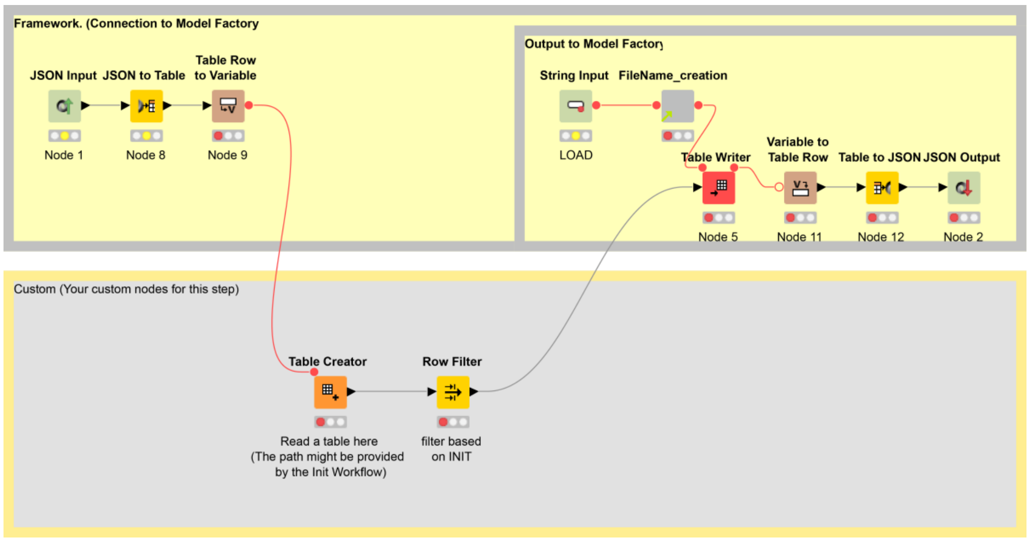


Figure 19.

LOAD workflow template

The LOAD step is good when new data is available, a separate security system is necessary for accessing data or there are multiple data sources. It is not a mandatory workflow but in most cases, it will be very useful. You can just return an empty table and do everything in the TRANSFORM workflow but this will possibly decrease reuse later.

3. TRANSFORM

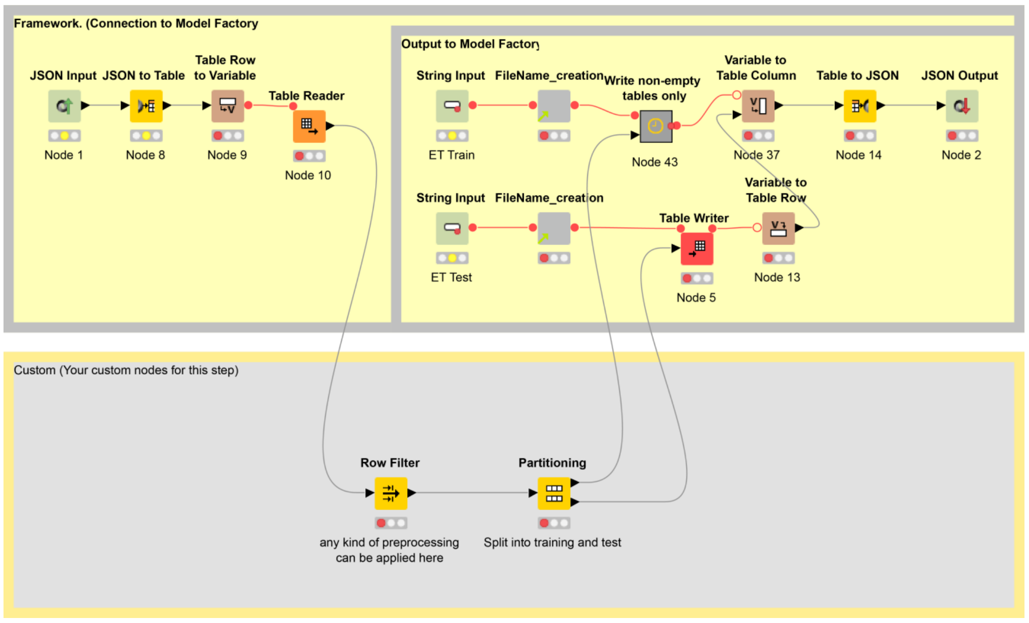


Figure 20.

TRANSFORM workflow template

The TRANSFORM workflow is mandatory. Normally the data will be provided by the LOAD step. Alternately, you could access it here. In the end, you need to generate a filePath_test and a filePath_Train so that the data is available for testing and scoring. Those point to files containing the respective parts of your data. If you do not have training data available, maybe because there is not enough data, you can simply set filePath_Train to missing. The model factory will not execute the learning, but will wait until a filepath is provided in one of the next executions of the model factory either as you manually set it or as your workflow logic provide it.

4. LEARN

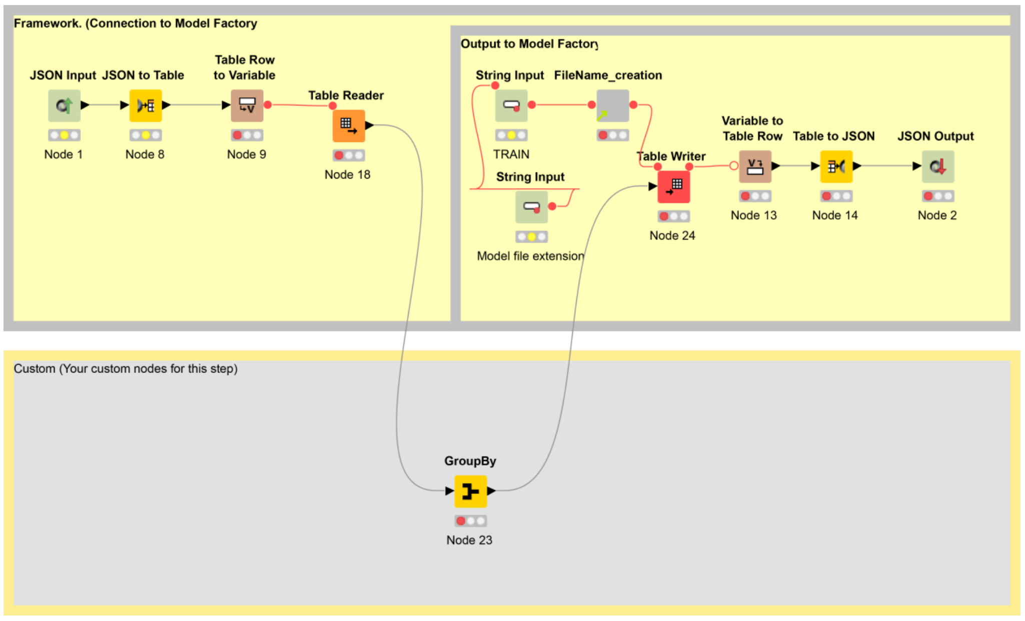


Figure 21.

LEARN workflow template

The LEARN step is mandatory. It will get the filePath_Train containing the training data and will output the path to the model. This model can be written in any format, as we will save the full file for deployment later.

5. SCORE

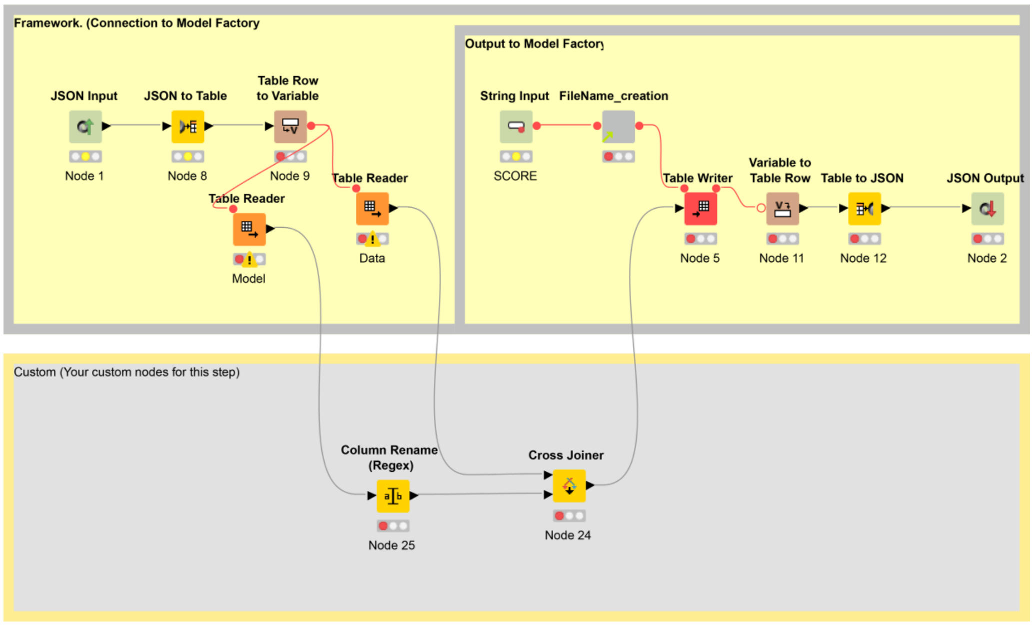


Figure 22.

SCORE workflow template

The SCORE step is mandatory. It will get the filePath_Test and a modelPath. Afterwards it should apply the model on the test data for prediction.

6. EVALUATE

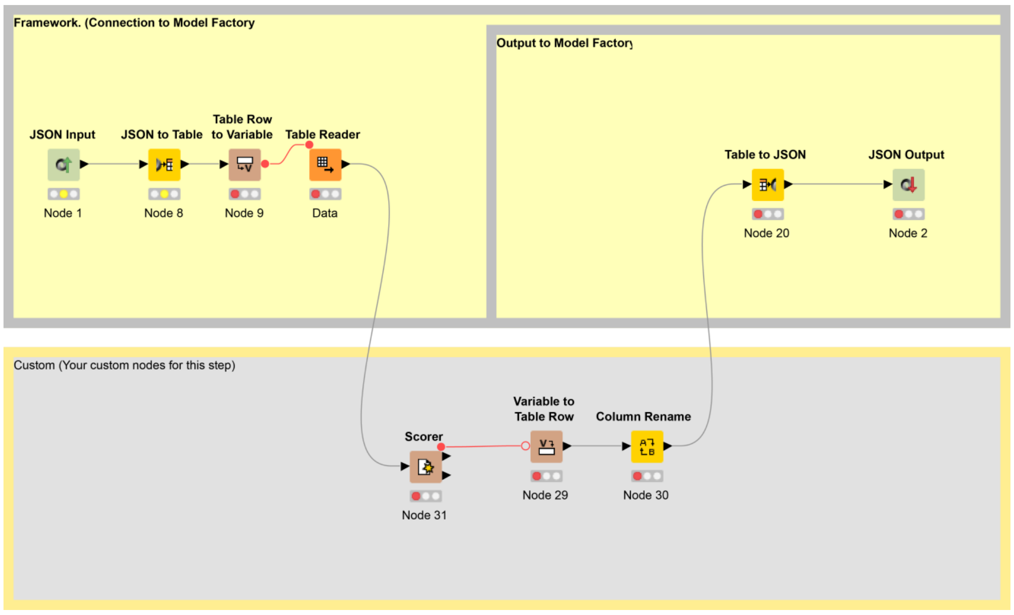


Figure 24.

EVALUATE workflow template

The EVALUATE step is not mandatory. It can be skipped if you e.g. already calculate the value in the score step. However, as the value is expected from the EVALUATE step, you need to provide it with a simple table (one line one row) containing only the value. The EVALUATE step will then send this value as the evaluated value back to the model factory for further processing.

7. DEPLOY

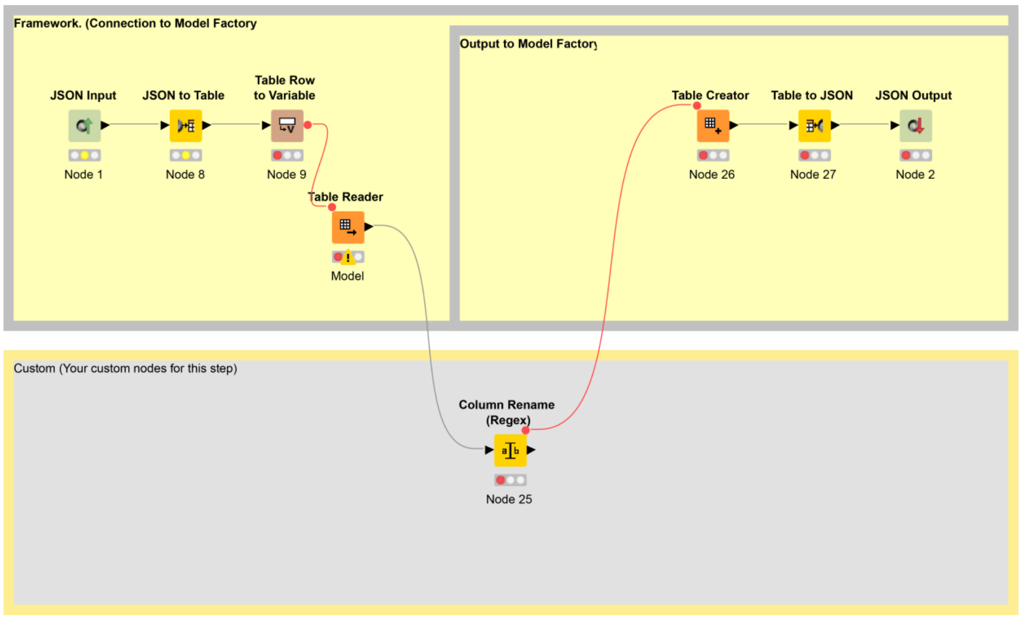


Figure 25.

DEPLOY workflow template

Deployment is the last step in the data analytics process and so it is as well the last step in our process. Independent of the DEPLOY step, any model created by the Model Factory will be written to the appropriate tables. This means that the “current model” will always be available by another workflow or application. That is already one type of “deploy”. But since there are so many other things you might want to do from WITHIN the Model factory - such as scoring any new data, transforming a model into another form for use, copying the model to the Cloud etc. - then the DEPLOY step would be where you could do that. The DEPLOY step is not mandatory, if you do not need it, just don’t add it to your process definition.

One note: based on the Status of your modelling configuration, the Deploy is not always executed. The DEPLOY step is only executed for AUTOMATIC that are RETRAINED and MANUAL_RETRAIN.

One last word on the steps, the power of this whole step approach only becomes clear when you start to to mix and match the process steps. Reusing steps from other processes will simplify your life and will make it easier to maintain. You will also see this in the examples.

Best practice for generating steps

If you are starting from scratch, it can be extremely helpful to have a complete working workflow to use as the basis for creating the process steps for your model process. Use workflow annotation to mark out the seven process steps similar to what has been done in Figure 7. You would use this completed workflow to then split into the seven steps.

We are using the capbility in KNIME for capturing and passing information between JSON steps. The mechanism for doing that requires that each step is created in order, as the input for a follow-on step is dependent on the executed output of a previous step.

The INIT step will be created first. Copy and rename the INIT Template and use it as a basis for creating your own INIT step. You should be able to do this in the gray area in the example. When you are finished, save and execute the INIT step. Output of your INIT step is needed the first time for creating the LOAD step. Copy and rename the LOAD Template. Now go back to your executed INIT step and open the output from the Table to JSON node. In there, you will see one row where one of the columns is a JSON column. Copy the contents of that JSON column. Go back to your new LOAD workflow and configure the JSON Input node by removing the brackets that are there and pasting the contents of the JSON field you previously copied. Save the node. You can now put your workflow components into the LOAD step (again within the gray areay), save and execute the node. This second step now has all the JSON components it needs to be used and reused later.

The procedure for each additional step is exactly the same. For the next step in the sequence (TRANSFORM in our case), copy the JSON field from the LOAD step and use that in the JSON INPUT node.

Some step Templates contain FileNameCreation wrapped nodes. These generate unique temporary file names. When debugging, you should never reset these nodes. Instead only mark and reset the components in the gray area.

If you are done with your changes, reset all workflows and save them. They can now be be used from the model factory.

Model Factory Use Cases

It is important to think of the ongoing steady-state of the model factory. At any time, you may have hundreds or thousands of model configurations in the system for processing. We have selected a number of use cases that all can be handled by the Model Factory.

Automated Threshold Checking

This is by far the most important use case because it saves so much time, resources and saved false positives and negatives because of a degraded model. If you have checked in many model configurations, you want to check that the models have not deteriorated below their threshold and the performance is therefore sub-optimal. This is handled by the Model Configuration type AUTOMATIC with the specified METRIC and a THRESHOLD.

By default, AUTOMATIC will be evaluated every time the Model Factory runs. We will talk later about extending the model factory to include both recency/frequency and scheduling.

Manual Threshold Checking

Occasionally you will have a model configuration that only needs to be retrained on an “as needed” basis. But you would like to benefit from the infrastructure provided by the Model Factory in capturing the Evaluation over time. In this case, if the type is set to MANUAL then evaluation always occurs but not TRAIN and DEPLOY. When you set the type to MANUAL_RETRAIN, the model factory will run the Retrain and - on end - reset the Type back to MANUAL.

New Product Model Generation

In many cases, you MUST provide a model (because a production system requires it, for example) and a “missing” is not an option. In this case, you can assign either a generic model or a “similar” model so that a model is available. This is done using the model initialization parameter in the modelling configuration. Please be aware that this can only work inside your current process definition. Hence, the initialization needs to be a modelling configuration itself and has the same process name.

As data appears and that alternate model degenerates below the THRESHOLD, a new model will be trained based on the real data without intervention.

Champion Model Testing

For many organizations, they will have an existing model that may occasionally be retrained. However, for some topics, the real data science may continue and someone may come up with a totally different model process (either preprocessing, algorithm changes or whatever) that they think are better than the current model. Within the model factory, you can have an existing model configuration point to a champion model. You can then set up a separate champion model configuration that can be manually run. The model is NOT automatically replaced but instead a manual comparison can be performed. In an ideal world, the evaluation process step will be the same between the production and the champion model.

The Public Example

We have already described the KNIME Model Factory. That Model Factory workflow along with the required tables are available on the public examples server. To give you a better feel for how this whole concept can be used inside KNIME, we have put two working use cases online as well.

Use Case: AirBNB

The first one is based on data from AirBNB. The data is available for free from Inside AirBNB (http://insideairbnb.com/get-the-data.html). There is tons of data available, you can see the individuals' apartment information, information about the property owners and much more. We here use only the calendar information. We have extracted calendar information for all apartments for each day, which shows if an apartment was booked, or not. The records also contain the price of each apartment as well as apartment size information (2 Beds and > 2 Beds). This data is available in 6 datasets representing 6 specific cities (The workflow that generated this data is available in the public example if you want to extract and change the data).

For our use cases, we want to build two predictors. The first predictor should model the prices of apartments and the second predictor should model how many apartments are available. In addition, we want to do this for both small and large apartments so that we can compare.

For price prediction, we define a Model_Process with its seven steps in Seattle that can be reused for either small or the large apartments.

- **INIT** As recommended earlier, the init takes the model_name from the model_configuration table (either "2 Beds" or ">2 Beds") and translates it into the Bin Number. It also selects the file containing the price data for Seattle. Note there is no data being transferred here, just the setup and initialization of fields.
- **LOAD** The load now loads the file, and, using the Bin Number, filters it down to the selected data.
- **TRANSFORM** Here we calculate a lag of the previous 7 days for each day. We also split out the 10 most recent days as test set and use the rest as training data.
- **LEARN** In the learn step we used this information to train a linear and a polynomial regression model and automatically return the one with a better fit on the training data.
- **SCORE** In the scoring we applied the model to our test data. As we did not want to have a dedicated evaluate workflow, the result table contained the RMSD of the predicted prices against the true prices.
- **EVALUATE** We did not use a dedicated evaluation workflow here, but just returned the value from the score workflow.
- **DEPLOY** We did not include deployment here since the Model factory saves the model automatically and we are not taking further steps to deploy.

The Model Process steps have been created and we have entered them in our Model_Process table. To execute, we create 2 model configurations entries, one called "2 Beds" or ">2 Beds", which then will run automatically in the Model Factory creating two price prediction models for Seattle.

Up next, we create the seven steps for the prediction of the number of small and large free apartments.

These are the seven steps for the prediction apartments in Seattle.

- **INIT** The init takes the model name (either 2 Beds or >2Beds) and translates it into the Bin Number. In addition it selects the file, containing the free apartment data for Seattle.
- **LOAD** same as price prediction
- **TRANSFORM** No preprocessing was necessary, so we only split the 10 most recent days as test set, while the rest was the training data.
- **LEARN** The model was a simple mean of the training data.
- **SCORE** The score was as well the mean, and we directly output the RMSD
- **EVALUATE** same as price prediction
- **DEPLOY** We did not include deployment here.

After saving these KNIME process steps, a Model Process is defined and two model process configurations, again with the names 2 Beds and >2Beds, which this time point to the Model process we just created.

Up until now, we are focusing only on the Seattle Data. However, of course, it would be nice to have that information for the other cities as well. That is why we extended the AirBNB Dataset to include other cities as well (overall and not broken down by small/large).

We simply need to either copy and modify a minimum number of steps or reuse process steps we already have. In our case, two change:

- **INIT** The init takes the Model_Name, which now should refer to the City and returns the location of the calendar data file. It does not need the binning/filter part used previously, since we are not processing on small and large apartments.
- **LOAD** Read the file, as provided by INIT.

And that's it, the other five steps can be reused from our previous process definition. We would then define the Model Process, pointing to the 2 new Process Steps and simply pointing to the other 5 existing process steps.

We have 5 additional Model Configurations, named for each city, pointing to the new process model.

A possible extension to these examples could involve catering for both City AND Apartment Size. This could be done either by have a combined Model_Configuration name containing both the City and the Size (for example: Seatle_2Beds) or by expanding the model configuration table with an extra field. Either way, you take care of the assignments in a special INIT model process workflow.

Use case: Product Data

The next use case is for predicting prices of different products. The provided dataset has anonymized product names for 6 products, one price per product per day. We here do not want to predict the real price but only the price tendency indicating if the price will go up, down or is stable currently. The following process steps were defined.

- **INIT** The init takes the product name and forwards this to the LOAD workflow
- **LOAD** Read the product data and filter it down to the selected product only.
- **TRANSFORM** The Transform calculates the difference to the 1.1.2010, to change the date into a number. Afterwards it outputs the 10 most recent days as test and the remaining days as training data.
- **LEARN** The model is based on a linear regression. Based on the rise of the curve and the error of the linear regression, we than predict if the curve is going up or down, or is currently stable.
- **SCORE** The scoring applies the linear regression model on the test data and appends the trend as calculated in the learn step.
- **EVALUATE** For being able to evaluate if the model is still in the same trend, we calculate the trend for the test set as well. The evaluation is 1 if the trend is the same and 0 if the trend changes.
- **DEPLOY** We did not include deployment here.

A model process is defined and then one model configuration for reach product.

A possible extension to this example could be identifying new products and automatically creating model_configuration entries for those new products without the need to manually create them. Since a new product would not have enough data to train, in the INITIALIZATION workflow we could point to an existing model_configuration that would be used for that new product.

KNIME Model Factory Directory and examples

Everything you need to run the KNIME Model Factory is available on the public examples server of KNIME:

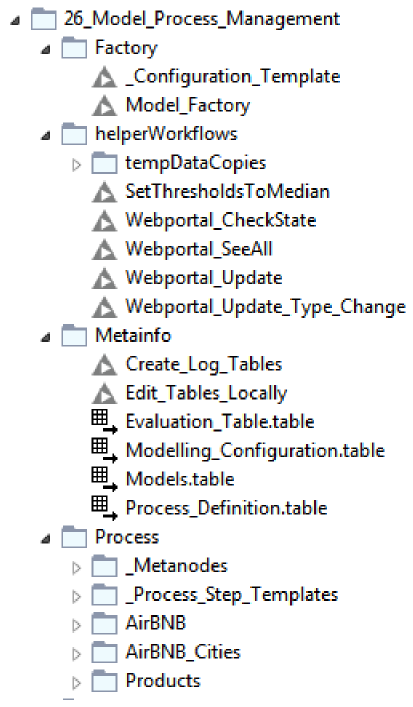

```
26_Model_Process_Management
  Factory
    _Configuration_Template
    Model_Factory
  helperWorkflows
    tempDataCopies
    SetThresholdsToMedian
    Webportal_CheckState
    Webportal_SeeAll
    Webportal_Update
    Webportal_Update_Type_Change
  Metainfo
    Create_Log_Tables
    Edit_Tables_Locally
    Evaluation_Table.table
    Modelling_Configuration.table
    Models.table
    Process_Definition.table
  Process
    _Metanodes
    _Process_Step_Templates
    AirBNB
    AirBNB_Cities
    Products
```

Figure 26.

Directory Listing of the Model Factory components

Under the high level **Model Process Management** directory you will find:

Factory

Model_Factory: the Model Factory Workflow. This workflow can be used locally or with the KNIME Server

_Configuration_Template: a workflow used by the Model_Factory. Do NOT change or remove this workflow

HelperWorkflows

tempDataCopies: A directory containing temporary copies of data tables used in the examples above

SetThresholdsToMedian: an example workflow that can be used to do a learning of the threshold to automatically determine when a model should be retrained.

Webportal_CheckState,Webportal_SeeAll,Webportal_Update,
Webportal_Update_Type_Change: Sample Webportal workflows. You will need to have access to KNIME Server to use these.

Metadata

Process_Definition.Table
Modelling_Configuration.Table
Evaluate_Table.Table
Models.Table

The KNIME tables as described here in the white paper

Create_Log_Tables: a workflow for reinitializing the 4 Model Process Tables.

Edit_Tables_Locally: a workflow for editing the tables locally if you are not using KNIME Server and the KNIME Webportal

Process

_Metanodes: shared metanodes used by the various workflows

_Process_Step_Templates: the 7 process step templates described above.

_AirBnB: all KNIME Process step workflows and data needed for both the price prediction and occupancy prediction for Seattle

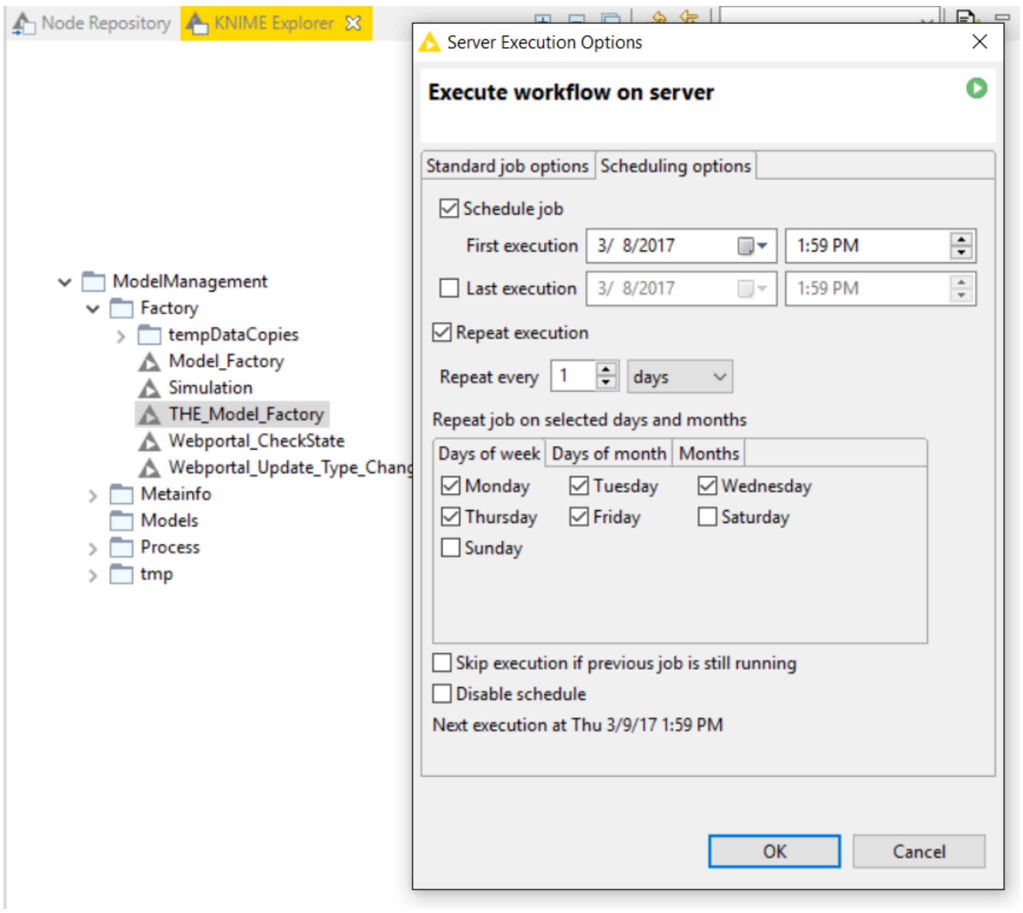
_AirBnBCities: all KNIME Process step workflows and data needed for the cities example

_Products: all KNIME process step workflows and data needed for the product price prediction

Deploying the Model Factory in Production

The Public Example workflows we are providing here can be run free of charge using the KNIME Analytics Platform, with at least version 3.2. For running the call local workflow node, you will need a personal productivity license. This is also free of charge; you only need to register for it.

However if you want to fully automate this as well as have a convenient interactive Web-based environment for maintaining the KNIME Model Factory, you will want to use the KNIME Server.



☑ Schedule job
First execution 3/ 8/2017 1:59 PM
☐ Last execution 3/ 8/2017 1:59 PM
☑ Repeat execution
Repeat every 1 days
Repeat job on selected days and months
☑ Monday ☑ Tuesday ☑ Wednesday
☑ Thursday ☑ Friday ☐ Saturday
☐ Sunday
☐ Skip execution if previous job is still running
☐ Disable schedule
Next execution at Thu 3/9/17 1:59 PM

Figure 27.

Scheduling the model factory to run on the KNIME Server.

The KNIME server allows you to

- Schedule
- Check Status
- Shared Repository
  - Workflows
  - Metanodes

  - Tables
  - Data

- Versioning
- Security

Through the KNIME Webportal you can interactively create, update and change as well as delete entries in the Model_Process as well as the Model_Configuration Tables. Sample KNIME workflows are provided for this.

It is the automation and collaboration features of the KNIME Server that allow the KNIME Model Factory to be used to its fullest.

Next Steps and Statement of Support

With the flexibility of the KNIME Model Factory, there are many things that can be done. Because it has at its heart the Call Local Workflow node, which can call any KNIME workflow, it provides an easy way to mix and match with other tools such as Spark, other Databases or even other REST based interfaces.

The KNIME Model Factory is provided as-is. Since we know it will be very popular with users for building and extending, we suggest questions, suggestions and ideas be shared in the KNIME Forum.