

Chapter 4: My First Model

4.1. Introduction

We have finally reached the heart of the KNIME Analytics platform: data modeling. There are two categories of nodes in the “Node Repository” panel fully dedicated to data modeling: “Analytics” → “Statistics” and “Analytics” → “Mining”. The “Statistics” category contains nodes to calculate statistical parameters and perform statistical tests. The “Mining” category contains mainly machine learning algorithms, from Artificial Neural Networks to Bayesian Classifiers, from clustering to Support Vector Machines, and more.

Data modeling consists of two phases: training the model on a set of data (the training dataset) and applying the model to a set of new data (live data or a test dataset). Complying with these two phases, data modelling algorithms in KNIME Analytics Platform are implemented with two nodes: a “Learner” node to train the model and a “Predictor” node to apply the model. The “Predictor” node takes on another name when we are dealing with unsupervised training algorithms.

The “Learner” node reproduces the training or learning phase of the algorithm on a dedicated training dataset. The “Predictor” node classifies new unknown data by using the model produced by the “Learner” node. For example, “Mining” → “Bayes” category implements naïve Bayesian classifiers. “Naïve Bayes Learner” node builds (learns) a set of Bayes rules on the learning (or training) dataset and stores them in the model. The “Naïve Bayes Predictor” node then reads the Bayes rules from the model and applies them to the incoming data.

All data modeling algorithms need a training dataset to build the model. Usually, after building the model, it is useful to evaluate the model quality, just to make sure we are not believing predictions produced by a poor-quality model. For evaluation purposes, a new data set, named test dataset, is used. Of course, the test dataset has to contain different data from the training dataset, to allow for the evaluation of the model capability to work properly onto unknown new data. For evaluation purposes, then, all modelling algorithms need a test dataset as well.

In order to provide a training set and a test set for the algorithm, usually the original data set is partitioned in two smaller data sets: the learning/training dataset and the test dataset. To partition, reorganize, and re-unite datasets, we use nodes from the “Manipulation” → “Row” → “Transform” category.

Sometimes problems can be incurred when there are missing values in the data. Indeed, not all modeling algorithms can deal with missing data. The model might also require the dataset to have a normal distribution. To remove missing data from the data sets and to normalize values in a column, we can use more nodes from the “Manipulation” → “Column” → “Transform” category.

In this chapter, we provide an overview of machine learning nodes, i.e., Learner and Predictor nodes, and of nodes to manipulate rows and transform values in columns. We work on the adult dataset, already used in the previous chapters. Here we create a new workflow group “Chapter4”, and inside that a new workflow called “Data Preparation”. We use this workflow to prepare the data for further data modeling operations. The first step of this workflow is to read the adult dataset with a “CSV Reader” node.

4.2. Split and Combine Datasets

Since many models need training data and separated test data, these two data sets have to be set up before modeling the data. In order to extract two data sets - one for training and one for testing - from the original data set, the “Partitioning” node can be used. If only a training set is needed and not a test set or if the original data set is too big to be used wholly, we can use the “Row Sampling” node.

Row Sampling

The “Row Sampling” node extracts a sample (= a subset of rows) from the input data. The configuration window enables you to specify:

- The sample size as an absolute number of rows or as a percentage of the original data set
- The extraction mode
- “Take from the top” means the top rows of the original data set
- “Linear Sampling” takes the first and the last row and samples in between these rows at regular steps
- “Draw randomly” extracts rows at random

- “Stratified sampling” extracts rows randomly whereby the distribution of values in the selected column is approximately retained in the output table

For “Draw randomly” and “Stratified sampling” a random seed can be defined so that the random extraction is reproducible (you never know when you need to recreate the exact same random training set).

In Figure 4.1 we selected a size of 20% of the original dataset for the learning set. Rows were extracted randomly from the original dataset. The size of 20% of the original dataset is probably too small; to be sure that all classes are actually represented in the learning set we could use the stratified sampling option.

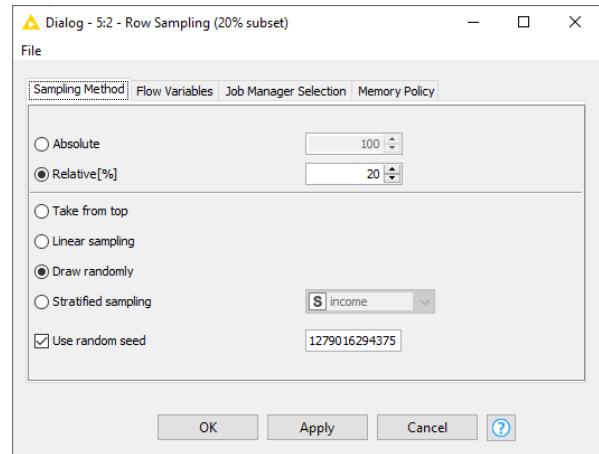


Figure 4.1. Configuration window for the Row Sampling node.

Note. The “Row Sampling” node only produces one data subset that we can use either to train or to test a model, but not both. If we want to generate two data subsets, the first one according to our specifications in the configuration window, and the second one with the remaining rows, we need to use the “Partitioning” node.

Partitioning

The “Partitioning” node performs the same task as the “Row Sampling” node: it extracts a sample (= a subset of rows) from the input data. It also builds a second dataset with the remaining rows and makes it available at the lower output port.

The configuration window enables you to specify:

- The sample size as an absolute number of rows or as a percentage of the original data set
- The extraction mode
- “Take from the top” means the first rows of the original data set
- “Linear Sampling” takes the first and the last row and samples between rows at regular steps

- “Draw randomly” extracts rows at random
- “Stratified sampling” extracts rows whereby the distribution of values in the selected column is approximately retained in the output table

For “Draw randomly” and “Stratified sampling” a random seed can be defined so that the random extraction is reproducible (you never know when you need to recreate the same learning set).

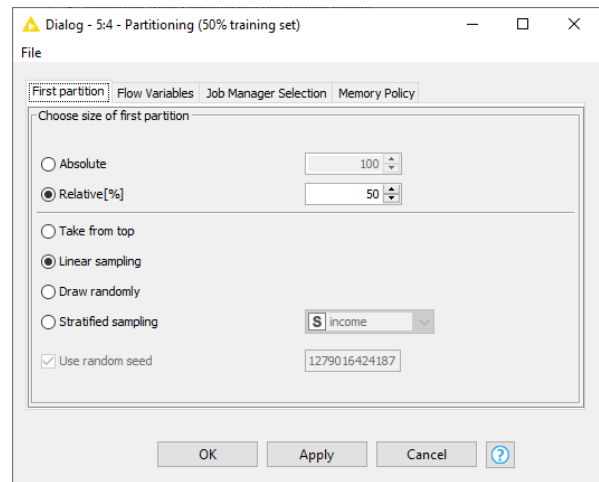


Figure 4.2. Configuration window of the Partitioning node.

Here, we selected a size of 50% of the original data set for the training set plus a linear extraction mode. The training set was made available at the upper output port; the remaining data were made available at the lower output port. In the linear sampling technique, rows adhere to the order defined in the original data set.

Sometimes it is required to present the data rows in the original order to the training algorithm, for example, when dealing with time series prediction. The row order, in this case, is a temporal order and is used by the model to represent temporal sequences. In this case, the linear sampling technique is advised. If we are dealing with time series analysis, where the past and the future have to remain separate, the “take from top” strategy is recommended.

Sometimes, however, it is not advisable to present rows to a Learner node in a specific order; otherwise, the model might learn the row order among all other underlying patterns. For example, the customer order in the database does not mean anything more than assigning a sequential identifying key to each customer. To be sure that data rows are presented to the model’s Learner node in a random order, we can extract them randomly or apply the “Shuffle” node.

Shuffle

The “Shuffle” node shuffles the rows of the input table putting them in a random order.

In general, the “Shuffle” node does not need to be configured. If we want to be able to repeat exactly the same random shuffling of the rows, we need to use a seed, as follows:

- Check the “Use seed” flag

- Click the “Draw new seed” button to create a seed for the random shuffling and recreate it at each run

We only applied the “Shuffle” node to the training set. It does not make a difference whether the data rows of the test set are presented into a pre-defined order or not.

Now we have a training dataset and a test dataset. But what if we want to recreate the original dataset by reunifying the training and the test set? KNIME has a “Concatenate” node that comes in handy for this task.

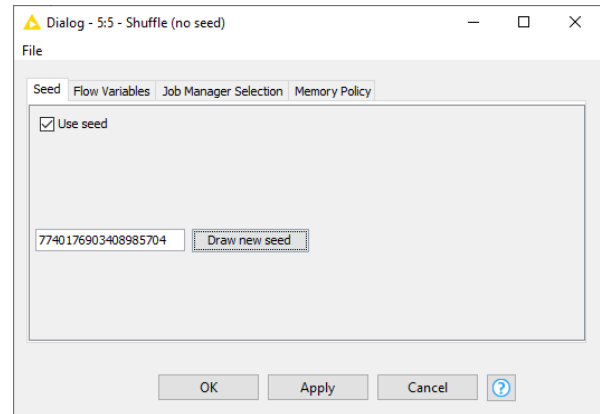


Figure 4.3. Configuration window of the Shuffle node.

Concatenate

The “Concatenate” node has two input ports, each one for a data set. The “Concatenate” node appends the data set at the lower input port to the data set at the upper input port.

The configuration window deals with the following:

- What to do with rows with the same ID:
 - skip the rows from the second data set
 - rename the RowID with an appended suffix
 - abort execution with an error (This option can be used to check for unique RowIDs)
- Which columns to keep
 - all columns from the second and first data set (union of columns)
 - only the intersection of columns in the two data sets (i.e., columns contained in both tables)
- Option “Enable hiliting” refers to the hiliting property available in the old “Data Views” nodes.

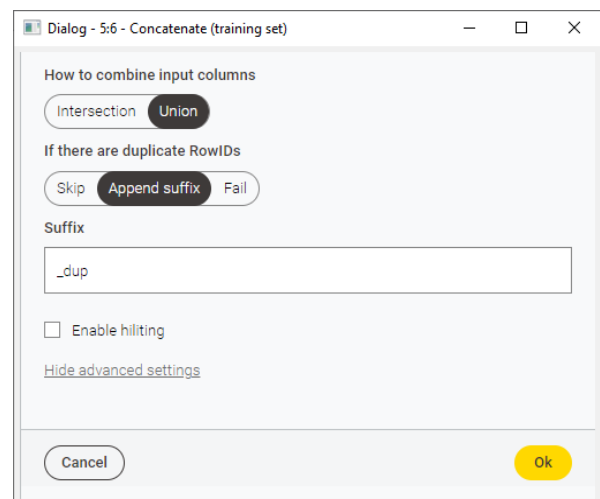


Figure 4.4. Configuration window for the Concatenate node.

Figure 4.4 shows an example of how the “Concatenate” node works, when the following options in the configuration window are enabled:

- append suffix to RowID in rows with duplicate RowID
- use union of columns
- no hiliting enabled

A similar node to the “Concatenate” node is the “Concatenate (Optional in)” node. The “Concatenate (Optional in)” node works exactly the same as the “Concatenate” node but allows to concatenate up to 4 data sets at the same time.

An example how the Concatenate node works:

First Data Table

RowID	Scores
Row1	22
Row3	14
Row4	10

Second Data Table

RowID	Name	Scores
Row1	The Black Rose	23
Row2	Cynthia	2
Row5	Tinkerbelle	4
Row6	Mother	6
Row7	Augusta	8
Row8	The Seven Seas	3

Concatenated Table

RowID	Name	Scores
Row1	?	22
Row3	?	14
Row4	?	10
Row1_dup	The Black Rose	23
Row2	Cynthia	2
Row5	Tinkerbelle	4
Row6	Mother	6
Row7	Augusta	8
Row8	The Seven Seas	3

4.3. Transform Columns

We have successfully derived a training set and a test set from the original data set. The original data set, though, contained missing values in some of its data columns and some machine learning algorithms cannot deal with missing values. KNIME data cells, indeed, can have a special “missing value” status. By default, missing values are displayed in the table view with a question mark (“?”).

Some of the Learner nodes might ignore data rows containing missing values, therefore reducing the data basis they are working on; and some of the Learner nodes will just fail when encountering a missing value. In the last case, a strategy to deal with missing values is required; but even in the first case, a strategy to deal with missing values is advisable to expand the data basis for the future model.

There are many strategies to deal with missing values and books have been written about which strategy is best to use in which context. Each strategy consists in substituting the missing value in question with another plausible value. What is the most plausible value depending on the context and often on the expert’s knowledge.

KNIME Analytics Platform implements the most common strategies to deal with missing values, such as using the data column mean value, moving average, maximum/minimum, most frequent value, linear and average interpolation, previous or next value, a fixed value, and probably by now more. Of course, the option of removing the data row containing a missing value is always available.

The node that deals with missing values is named “Missing Value”. The “Missing Value” node takes a data table as input and replaces missing values everywhere or only in selected columns with a value of your choice. The new data table with replaced missing values is then produced at the upper output port. Indeed, this node has two output ports. The lower output port is in the shape of a square blue rather than the usual black triangle. A blue square port means a PMML-compliant model.

PMML

PMML (Predictive Model Markup Language) is a standard XML-based structure that enables the storage of predictive models and data transformations. Since it is a standard structure, it enables the portability of models and transformations across platforms and applications.

KNIME Analytics platform supports PMML for models and transformations. The blue squares as input and output ports in KNIME nodes identify PMML compliant objects, be it predictive models or ETL transformations.

In KNIME it is not only possible to export models and single transformations as PMML structures, but also to modularly concatenate them so that the final PMML structure contains the sequence of transformations and the model created in the workflow and fed into the PMML structure. Two nodes are key for modular PMML: “PMML Transformation Appender” and “PMML Model Appender”.

Note. Some of the missing value strategies are marked with an asterisk in the menus of the configuration window in the “Missing Value” node. The asterisk indicates that such transformations are not PMML supported.

Missing Value

The “Missing Value” node replaces missing values in a data set everywhere or only in selected columns with a value of your choice.

In the “**Default**” tab, replacement values are defined separately for numerical and string type columns and applied to all data columns of the same type.

In the “**Column Settings**” tab, a replacement value is defined specifically for each selected data column and applied only to that column. To define the replacement value for a column:

- Double-click the column in the list on the left, OR
- Select the column from the list on the left
- Click the “Add” button under the list



missing values in:
age -> mean value
income -> remove row

Figure 4.5. The Missing Value node.

Chapter 4: My First Model

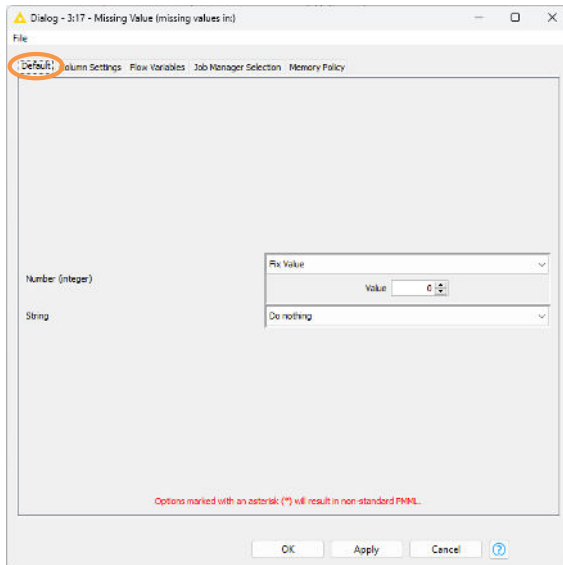


Figure 4.6. Configuration window for the Missing Value node: the "Default" tab.

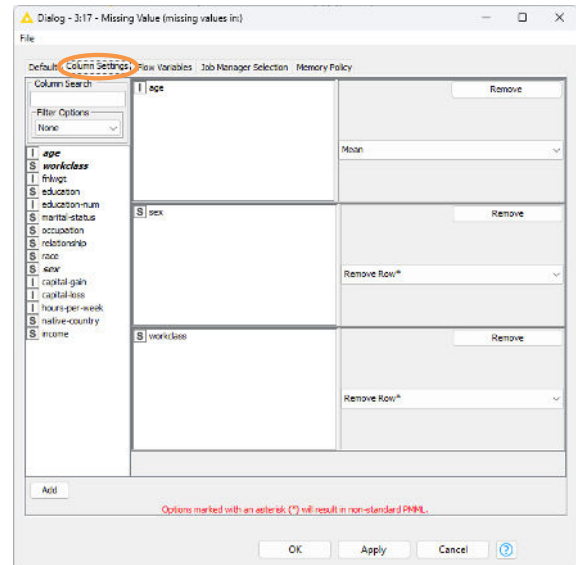


Figure 4.7. Configuration window for the Missing Value node: the "Column Settings" tab.

Then, select the desired missing value handling strategy.

A "Column Search" box is provided to help to find columns among many. A "Remove" button is also provided in the data column frame to remove the individual missing value handling strategy for the selected column. We introduced a "Missing Value" node prior the "Partitioning" node in our "Data Preparation" workflow. Here we set 0 as the fixed value to replace missing values in all numerical columns and "Do nothing" for missing values in String columns. Then, for column "age" (Integer) and "income" (String), we set individual replacement strategies for missing values. In column "age" missing values are replaced by the data column mean value; in column "income", rows with missing values are simply removed. While the missing value strategy for "age" is purely demonstrative, the missing value strategy for "income" is necessary, since we want to predict the "income" value given all other census attributes for each person.

Some data models - such as neural networks, clustering, or other distance-based models - require normalized input attribute values, for the data to be either normalized to follow the Gaussian distribution or just to fall into the [0,1] interval. In order to comply with this requirement, we use the "Normalizer" node.

Normalizer & Normalizer (Apply)

The “Normalizer” node normalizes data, i.e., it transforms the data to fall into a given interval or to follow a given statistical distribution. The “Normalizer” node is located in the “Node Repository” panel in the “Manipulation” → “Column” → “Transform” category.

The configuration window requires:

- the list of numerical data columns to be normalized
- the normalization method

The column selection is performed by means of an “Exclude”/“Include” frame, by manual selection or Wildcard/RegEx selection. For manual selection:

- The columns to be normalized are listed in the “Normalize” frame. All other columns are listed in the “Do not normalize” frame.
- To move from frame “Normalize” to frame “Do not normalize” and viceversa, use buttons “add” and “remove”. To move all columns to one frame or the other use buttons “add all” and “remove all”.

The “Normalizer” node has 2 output ports:

- At the upper port we find the normalized data
- At the lower port the transformation parameters are provided to repeat the same normalization on other data (light blue/dark blue square port)

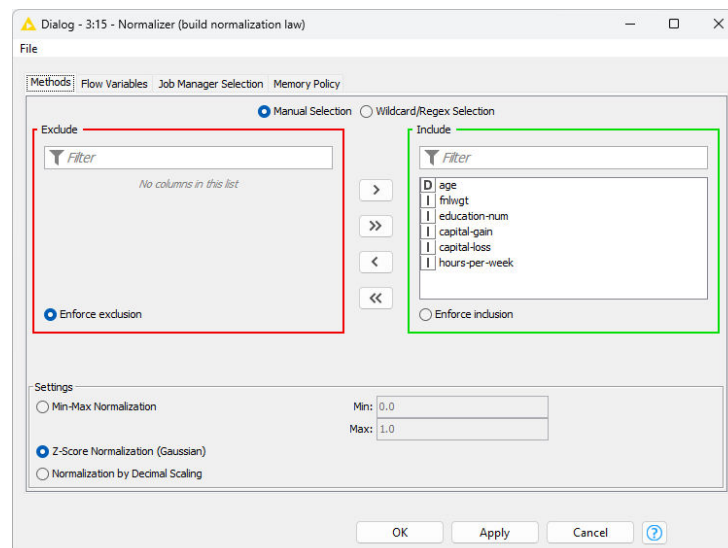


Figure 4.8. Configuration window of the Normalizer node.

Note. Triangular ports output/read data. Squared ports output/read parameters: model's parameters, normalization parameters, transformation parameters, graphics parameters, etc.

There are two normalizer nodes: "Normalizer" and "Normalizer (PMML)" node. They perform exactly the same task using the same settings. The only difference is in the exported parameter structure: KNIME proprietary structure (light blue square) or PMML compliant structure (dark blue square).

The "Normalizer (Apply)" node normalizes data. That is, it transforms data to fall into a given interval or to follow a given statistical distribution. It does not calculate the transformation parameters though; it obtains them from a "Normalizer" node previously applied to a similar data set. The "Normalizer(Apply)" node is located in the "Node Repository" panel in the "Manipulation" → "Column" → "Transform" category. The node has two input ports:

- one for the data to be normalized
- one for the normalization parameters

No additional configuration is required.

Normalization Methods

- **Min-Max Normalization:** This is a linear transformation whereby all attribute values in a column fall into the [min, max] interval and min and max are specified by the user.
- **Z-score Normalization:** This is also a linear transformation whereby the values in each column are Gaussian-(0,1)-distributed, i.e., the mean is 0.0 and the standard deviation is 1.0.
- **Normalization by Decimal Scaling:** The maximum value in a column is divided j-times by 10 until its absolute value is smaller or equal to 1. All values in the column are then divided by 10 to the power of j.

We applied the "Normalizer" node to the training set from the output port of the "Partitioning" node, in order to normalize the training set and to define the normalization parameters. We then introduced a "Normalizer (Apply)" node to read the normalization parameters and to use them to normalize the remaining data from the "Partitioning" node (2nd output port).

Let's now write the processed training dataset and test dataset into CSV files, named "training_set.csv" and "test_set.csv" respectively. We used two "CSV Writer" nodes: one to write the training set into "training_set.csv" file and one to write the test set into "test_set.csv" file. These last 2 nodes conclude the "Data Preparation" workflow.

Workflow: Data Preparation

This workflow prepares the data for the next workflow (My First Data Model) and uses some of the most common data preparations:

- subsetting (Row Sampling and Partitioning nodes)
- Strategies to deal with missing values (Missing Value node)
- Shuffling (Shuffle node)
- Concatenation of data sets (Concatenate node)
- Normalization (Normalizer and Normalizer (Apply) nodes)

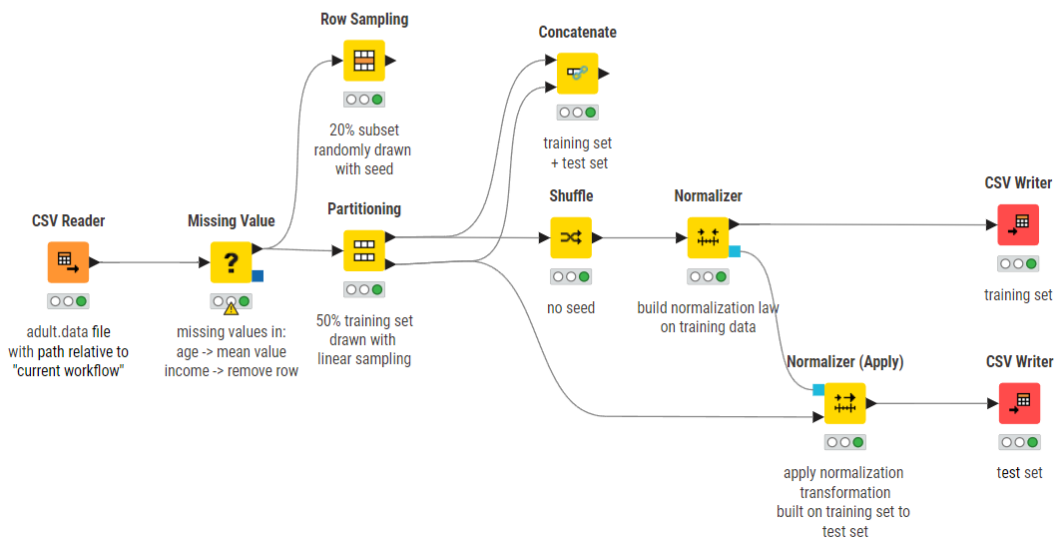


Figure 4.9. The "Data Preparation" workflow.

4.4. Machine Learning Models

Now let's create a new workflow and call it "My First Model". We will use this workflow to show how models can be trained on a set of data and then applied to new data. To give an overview, we will go through some standard data analysis method paradigms. Standard here refers to the way the paradigms are implemented in KNIME – for example with one node as the Learner and a separate node as the Predictor/Applier – rather than with regard to the quality of the algorithm itself.

The first two nodes in this new workflow are two “CSV Reader” nodes: one to read the training set and one to read the test set that was saved in two CSV files in the “Data Preparation” workflow at the end of the last section.

In this workflow “My First Model”, we want to predict the “income” label of the adult data set by using the other attributes and based on a few different models. This section does not intend to compare those models in terms of accuracy or performance. Indeed, not much work has been spent to optimize these models to become the most accurate predictors. Contrarily, the goal here is to show how to create and configure such models. How to optimize the model parameters to ensure that they will be as accurate as possible is a problem that can be explored elsewhere^{2,3,4}.

In every supervised prediction/classification problem, we need a labelled training set; that is a training set where each row has been assigned to a given class. These output classes of the data rows are contained in a column of the data set: this is the class or target column.

Most data mining and statistics paradigms consist of two nodes: a Learner and a Predictor. The Learner node defines the model’s parameters and rules that make the model suitable to perform a given classification/prediction task. The Learner node uses the input data table as the training set to define these parameters and rules. The output of this node is a set of rules and/or parameters: the model. The Predictor node uses the model built in the previous step and applies it to a set of unknown (i.e., new unclassified) data to perform the classification/prediction task for which it was built.

The Learner node requires a data table as input and provides a model as output. The output port of the Learner node is represented as a blue square, which is the symbol for a PMML-compliant model.

The Predictor node takes a data table and a model at the input ports (a black triangle for the data and a blue square for the model) and provides a data table containing the classified data at the output port.

² C.M. Bishop, “Pattern Recognition and Machine Learning”, Springer (2007)

³ M.R. Berthold, D.J. Hand, “Intelligent Data Analysis: An Introduction”, Springer Verlag, 1999

⁴ M.R. Berthold, C. Borgelt, F. Höppner, F. Klawonn, “Guide to intelligent data analysis”, Springer 2010

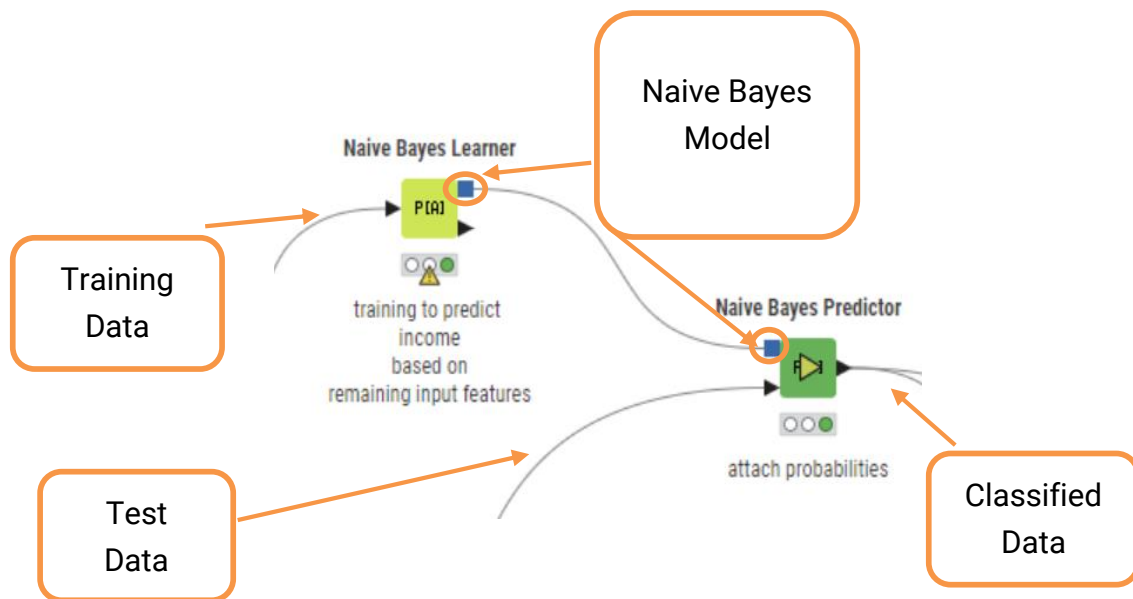


Figure 4.10. Learner and Predictor nodes.

Naïve Bayes Model

Let's start with a naïve Bayes model. A Bayesian model defines a set of rules, based on the Gaussian distributions and on the conditional probabilities of the input data, to assign a data row to an output class^{2,3,4}. In the "Node Repository" panel in the "Mining" → "Bayes" category we find two nodes: "Naïve Bayes Learner" and "Naïve Bayes Predictor".

Naïve Bayes Learner

The "Naïve Bayes Learner" node creates a Bayesian model from the input training data. It calculates the distributions and probabilities to define the Bayesian model's rules from the training data. The output ports produce the model and the model parameters respectively. In the configuration window you need to specify:

- The class column (= the column containing the classes)
- The default probability and the minimum standard deviation (almost 0)

- The maximum number of unique nominal values allowed per column. If a column contains more than this maximum number of unique nominal values, it will be excluded from the training process.
- How to deal with missing values (skip vs. keep)
- Compatibility of the output model with PMML 4.2

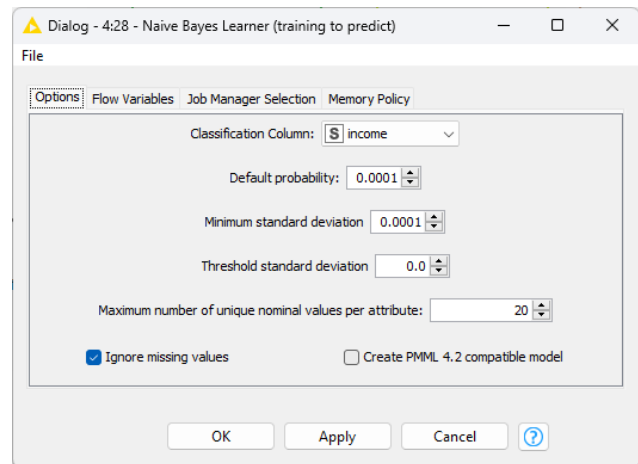


Figure 4.11. Configuration window for the Naive Bayes Learner node.

Naïve Bayes Predictor

The “Naïve Bayes Predictor” node applies an existing Bayesian model to the input data table.

All necessary configuration settings are available in the input model.

In the configuration window you can only:

- Append the normalized class distribution values for all classes to the input data table
- Customize the column name for the predicted class

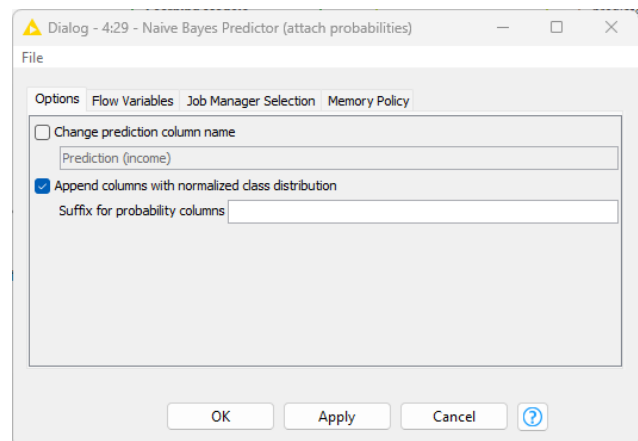


Figure 4.12. Configuration window for the Naive Bayes Predictor node.

Note. All predictor nodes expose the same configuration window: one option to append predicted class probabilities/normalized distributions and one option to change the default prediction class column name.

In the “My First Model” workflow we connected a “Naïve Bayes Learner” node to the “CSV Reader” node that reads the training data set. In the configuration window of the “Naïve Bayes Learner”, we specified “income” as the class/target column, we opted to skip rows with missing values in the model estimation and to skip a column if more than 20 nominal values were found.

Chapter 4: My First Model

After setting this configuration, a yellow triangle appears under the “Naïve Bayes Learner” to say that column “native country” in the input data set has too many (> 20 as from configuration settings) nominal values and will be ignored. We then run the “Execute” option for the “Naïve Bayes Learner” node.

The next step involves connecting a “Naïve Bayes Predictor” node to the “CSV Reader” node to read the test set through the data port; the “Naïve Bayes Predictor” node is then also connected to the output port of the “Naïve Bayes Learner” node through the model port.

After execution, the “Naïve Bayes Predictor” shows a new column appended to the output table: “Prediction (income)”. This column contains the class assignments for each row performed by the Bayesian model. How correct these assignments are, that is how good the performance of the model is, can only be evaluated by comparing them with the original labels in “income”.

If the flag to append the probability values for each output class was enabled, in the final data table there will be as many new columns as there are values in the class column; each column contains the probability for a given class value according to the trained Bayesian model.

The screenshot shows a data table with 38 columns and 37 rows. The columns include: Row ID, D age, S workclass, D fnlwgt, S occupation, S relation..., S sex, S income, D capital..., D hours..., S native..., D P (inco..., D P (S Predict... (circled in orange). The data rows show various demographic and occupational information, such as 'Self-emp-not-inc', 'Exec-managerial', 'Husband', 'White', 'Male', and 'income' values like '<=50K' or '>50K'. The 'Predict...' column contains predicted income class values, such as '<=50K' or '>50K'.

Figure 4.13. Bayes Model's classified data.

KNIME has a whole “Analytics” → “Mining” → “Scoring” category with nodes that measure the classifiers’ performances. The most straightforward of these evaluation nodes is the “Scorer”

node. We will use the “Scorer (JavaScript)” node because it also offers a nicer visualization of the results.

Scorer (JavaScript)

The “Scorer” node compares the values of two columns (target column and prediction column) in the data table; based on this comparison it shows the confusion matrix and some accuracy measures.

This node produces three output data tables: the confusion matrix, the statistics of correctly identified rows for each class, and the overall accuracy measures as set in the configuration window.

This node has a View option, where the confusion matrix and some accuracy metrics are displayed.

The configuration window has three tabs: “Scorer Options”, “Statistics Options”, “Control Options”.

Tab “Scorer Options” requires the selection of the two columns to compare (“Actual Column” and “Predicted Column”) and the sorting to be used in the evaluation strategy. The flag “Ignore missing values”, if unchecked, makes the node fail if missing values are encountered in one of the two columns to compare. All other options are related to the display of the node view.

Tab “Statistics Options” includes all accuracy measures and false/true positive/ negative numbers to calculate.

Like all JavaScript based nodes, this node produces a view with some degree of interactivity. Interactivity options are defined in the tab “Control Options”.

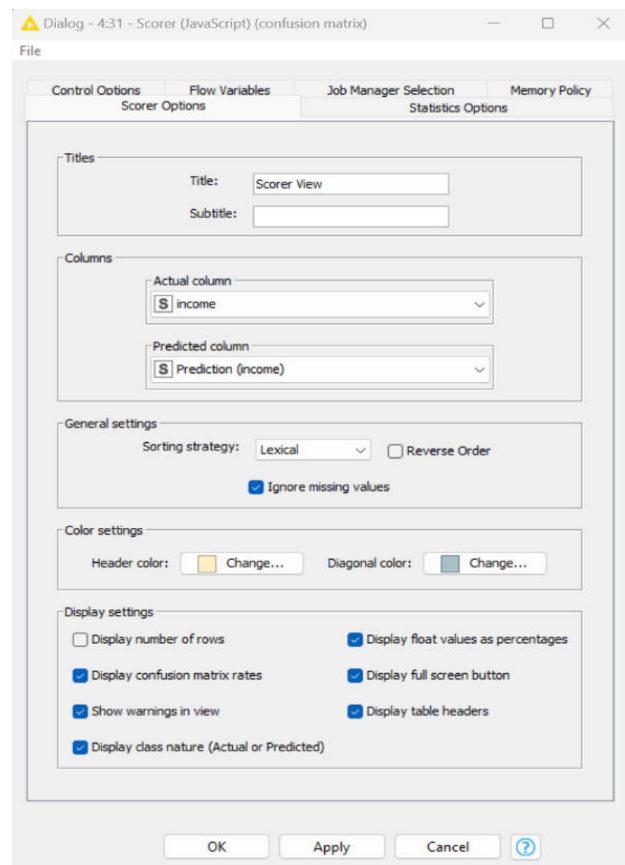


Figure 4.14. Configuration window of the Scorer (JavaScript) node.

We added a “Scorer (JavaScript)” node into the workflow “My First Model”. The node is connected to the data output port of the “Naïve Bayes Predictor”. The first column with the original reference values is “income”; the second column with the class estimation is the column called “Prediction (income)” which is produced by the “Naïve Bayes Predictor” node. During execution, values are then compared row by row and the confusion matrix and the consequent accuracy measures are calculated.

We can see the confusion matrix and the accuracy measures for the compared columns by selecting either the last three items or the item “Interactive View: Confusion Matrix” in the context-menu of the “Scorer (JavaScript)” node.

Confusion Matrix

In Figure 4.14, you can see the confusion matrix generated by the “Scorer” node. The confusion matrix shows the number of matches between the values in the target column and the values in the predicted column.

The values found in the target column are reported as Row IDs; the values found in the predicted column are reported as column headers. Since “income” has only two possible values – “>50K” and “<=50K” – the reading of the confusion matrix is quite simple.

The first cell contains the number of data rows that had an income “<=50K” and were correctly classified as having an income “<=50K”. The last cell, the one identified as (“>50K”, “>50K”), contains the number of data rows with an income “>50K” and that were correctly classified as

Row ID	<=50K	>50K
<=50K	11748	613
>50K	1950	1970

Figure 4.15. True Positives, False Negatives, True Negatives, and False Positives in the Confusion Matrix for “<=50K” as the positive class.

having an income ">50K". The other two cells represent the number of data rows with original income "<=50K" and incorrectly classified as having an income ">50K" and vice versa.

The cells along the diagonal from the top left corner to the lower right corner contain the numbers of correctly classified events. The opposite diagonal, the one from the top right corner to the lower left corner, contains the numbers of incorrectly classified events, that is the errors that we want to minimize.

The sum across one row of the confusion matrix indicates the total number of data rows in one class according to the labels in the original data set. The sum across one column indicates the number of data rows assigned to one class by the model. The sum of all columns and the sum of all rows must therefore be the same, since they represent the total number of data.

In our "Scorer" node, we selected the first column as the target classification column "income" and the second column as the output column of the Bayesian classifier. Thus, this confusion matrix says that 9554 data rows were correctly classified as having an income "<=50K"; 2902 were correctly classified as having an income ">50K"; and 876 and 2031 data rows were incorrectly classified.

Accuracy Measures

The second part of the "Scorer" node presents a number of accuracy measures^{5,6}. In a binary classification (or in any classification), we need to choose one of the classes as the positive class. Such choice is completely arbitrary and usually dictated by the data context. Once one of the classes has been assumed as the positive one, the following definitions can take place:

- **True Positives** is the number of data rows belonging to the positive class in the original data set and correctly classified as belonging to that class.
- **True Negatives** is the number of data rows that do not belong to the positive class in the original data set and are classified as not belonging to that class.
- **False Positives** is the number of data rows that do not belong to the positive class but are classified as if they do.
- **False Negatives** is the number of data rows that belong to the positive class but are assigned to a different class by the model.

⁵ D. L. Olson, D. Delen, "Advanced Data Mining Techniques" Springer; 2008

⁶ D.G. Altman, J.M. Bland, "Diagnostic tests. 1: Sensitivity and specificity" *BMJ* 308 (6943): 1552; 1994

In our case, if we arbitrarily choose “<=50K” as the positive class, the True Positives are in the first cell, identified by (“<=50K”, “<=50K”); the False Negatives are in the adjacent cell; the False Positives are below it; and the True Negatives are in the remaining diagonal cell.

On the basis of these True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) numbers, a number of correctness measures can be defined, each measure enhancing some aspect of the correctness of the classification task. These accuracy measures are provided in the lower output port of the “Scorer” node.

Let’s see how they are defined.

- $Sensitivity = \frac{True\ Positives}{(True\ Positives + False\ Negatives)}$
- $Specificity = \frac{True\ Negatives}{(True\ Negatives + False\ Positives)}$

“Sensitivity” measures the model’s capability to recognize one class correctly. If all instances of a given class are recognized correctly, the result is 0 “False Negatives” for that class; which means that no items of that class are assigned to another class. “Sensitivity” is then 1.0 for that class.

“Specificity” measures the model’s capability of recognizing what does not belong to a given class. If the model recognizes what does not belong to that class, the result is 0 “False Positives”; which means no extraneous data rows are misclassified in my class. “Specificity” is then 1.0 for that class.

In a two-class problem, “Sensitivity” and “Specificity” are used to plot the ROC Curves (see “ROC Curve” later on in this section).

- $Recall = \frac{True\ Positives}{(True\ Positives + False\ Negatives)} = Sensitivity$
- $Precision = \frac{True\ Positives}{(True\ Positives + False\ Positives)}$

“Precision” and “Recall” are two widely used statistical accuracy measures. “Precision” can be seen as a measure of exactness or fidelity, whereas “Recall” is a measure of completeness.

In a classification task, the “Precision” for a class is the number of “True Positives” (i.e. the number of items correctly labeled as belonging to that class) divided by the total number of elements labeled as belonging to that class. “Recall” is defined as the number of “True Positives” divided by the total number of elements that actually belong to that class. “Recall” has the same definition as “Sensitivity”.

- $F - measure = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$

The F-measure can be interpreted as a weighted average of “Precision” and “Recall”, where the F-measure reaches its best value at 1 and worst score at 0.

- $$Accuracy = \frac{(TP+TN)}{(TP+FP+FN+TN)}$$

being TP = True Positives, FP = False Positives, TN = True Negatives, and FN = False Negatives.

Cohen’s Kappa is a measure of inter-rater agreement as $\frac{(Accuracy-P(chance))}{1-P(chance)}$ where $P(chance)$ is the average probability of classifying events as positive or negative, weighted for the respective a priori probability of the positive and negative class. The Cohen’s kappa gives a more balanced accuracy estimation in case of strong differences in the class distributions.

Accuracy” is an overall measure and is calculated across all classes. An accuracy of 1.0 means that the classified values are exactly the same as the original class values.

All these accuracy measures are reported in the data table in the second port at the bottom of the “Scorer (JavaScript)” node and give us information about the correctness and completeness of our model.

Row ID	True Po...	False Positives	True Negatives	False Negatives	Recall	Precision	Sensitivity	Specificity	F-measure
<=50K	11748	1950	1970	613	0.95	0.858	0.95	0.503	0.902
>50K	1970	613	11748	1950	0.503	0.763	0.503	0.95	0.606

Figure 4.16. Accuracy statistics table from the Scorer (JavaScript) node with the accuracy measures for each class.

View: Confusion Matrix

The context menu of the “Scorer” node offers a possibility to visualize the confusion matrix:

- Item “Open output port” > “1: Confusion matrix” > “Table”

The context menu of the *Scorer* node has an option called “Open view”, which shows us the Scorer view, and under that, we have the confusion matrix table and the overall statistics. The tab is “Confusion Matrix”, then the next option is to view the “Class Statistics table”, and the final table is the “Overall Statistics Table”.

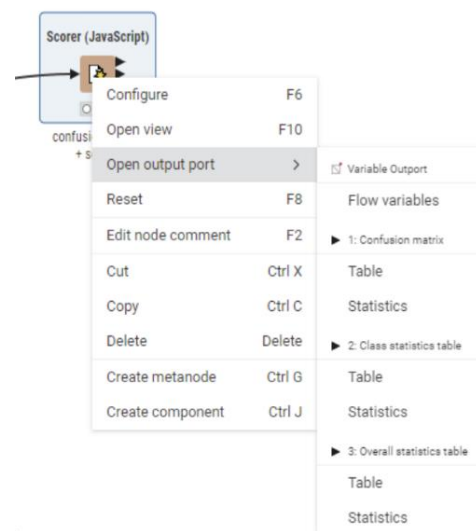


Figure 4.17. The context menu of the Scorer (JavaScript) node.

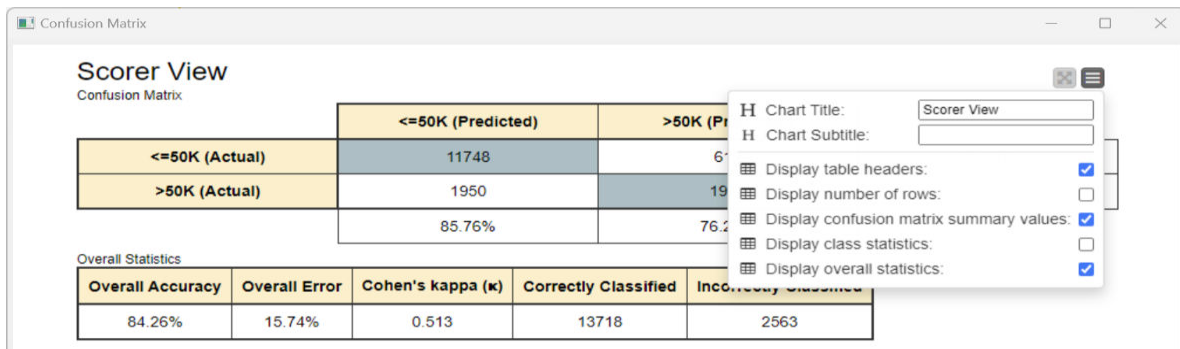


Figure 4.18. Confusion Matrix from the Interactive View of the Scorer node.

Like all JavaScript based nodes in KNIME Analytics Platform, also this node includes a menu button in the top right corner. The menu opening, after clicking this button, allows you to change the view display, such as title and subtitle, but also to include (or not) summary values, class statistics, overall statistics and so on. Finally, cell content in the confusion matrix is selectable.

Decision Tree

Using the same workflow “My First Model”, let’s now apply another quite popular classifier: a decision tree^{7,8}.

The Decision Tree algorithm is a supervised algorithm and therefore consists of two phases – training and testing - like the Naïve Bayes classifier that we have seen in the previous section. The decision tree is implemented in KNIME with two nodes: one node for training and one node for testing, i.e.:

- The “Decision Tree Learner” node
- The “Decision Tree Predictor” node

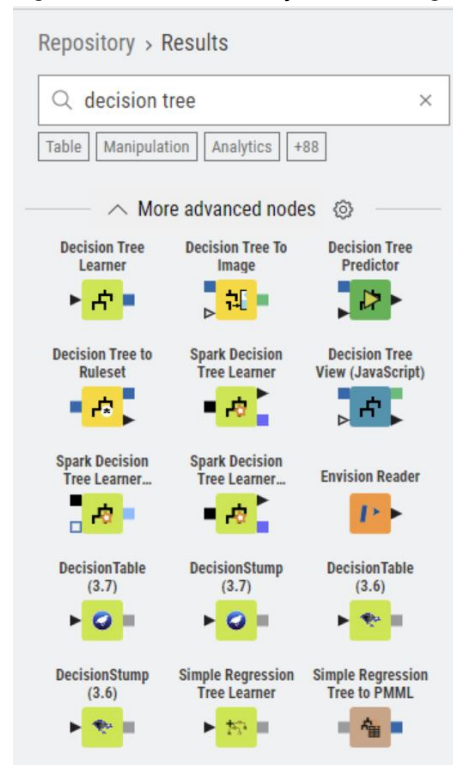


Figure 4.19. Two nodes implement a Decision Tree: the Decision Tree Learner node and the Decision Tree Predictor node.

⁷ J.R. Quinlan, "C4.5 Programs for machine learning", Morgan Kaufmann Publishers Inc. , 1993

⁸ J. Shafer, R. Agrawal, M. Mehta, "SPRINT: A Scalable Parallel Classifier for Data Mining", Proceedings of the 26th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc. ,1996 (<http://citeseer.ist.psu.edu/shafer96sprint.html>)

- The “Decision Tree Learner” node takes a data set as input (black triangle), learns the rules necessary to perform the desired task, and produces the final model at the output port (blue square). Let’s connect a “Decision Tree Learner” node to the “CSV Reader” node named “training set”.

Let’s also create a “Decision Tree Predictor” node to follow the “Decision Tree Learner” node. The “Decision Tree Predictor” node has two inputs:

- A data input (black triangle) with new data to be classified
- A model input (blue square) with the model parameters produced by a “Decision Tree Learner” node

Decision Tree Learner: “Options” Tab

The “Decision Tree Learner” node builds a decision tree from the input training data. In the configuration window you need to specify:

General

- The class column. The target attribute must be nominal (String).
- The quality measure for split calculation: “Gini Index” or “Gain Ratio”.
- The pruning method: “No Pruning” or a pruning based on the “Minimum Description Length (MDL)” principle^{7,8}. The option Reduced Error Pruning, if checked, applies a simple post-processing pruning.
- The stopping criterion: the minimum number of records in each decision tree’s node. If one node has fewer records than this minimum number, the algorithm stops further splitting of this branch. The higher the number, the shallower the tree.
- The number of records to store for view: the maximum number of rows to store for the hilite functionality. A high number slows down the algorithm execution.
- The “Average Split Point” flag. For numerical attributes, the user has to choose one of two splitting strategies:
- The split point is calculated as the mean value between the two partitions’ attributes (“Average Split Point” flag enabled)

- The split point is set to the largest value of the lower partition (“Average Split Point” flag disabled)
- The Number of threads on which to run the node (default Number threads = 2 * number of processors available to KNIME).

Root Split

If you know that one attribute must be important for the classification, you can force it on the root node of the tree, by enabling “Force root split column” and selecting the “Root split column”.

Binary nominal splits

Here you can define whether Binary nominal splits apply to nominal attributes. In this case you can set the threshold Maximum # of nominal splits, up to which an accurate split is calculated instead of just a heuristic. The heuristic, though less precise, reduces the computational load.

“Filter invalid attribute values ...” inspects the tree at the end of the training procedure and removes possible duplicates and incongruences.

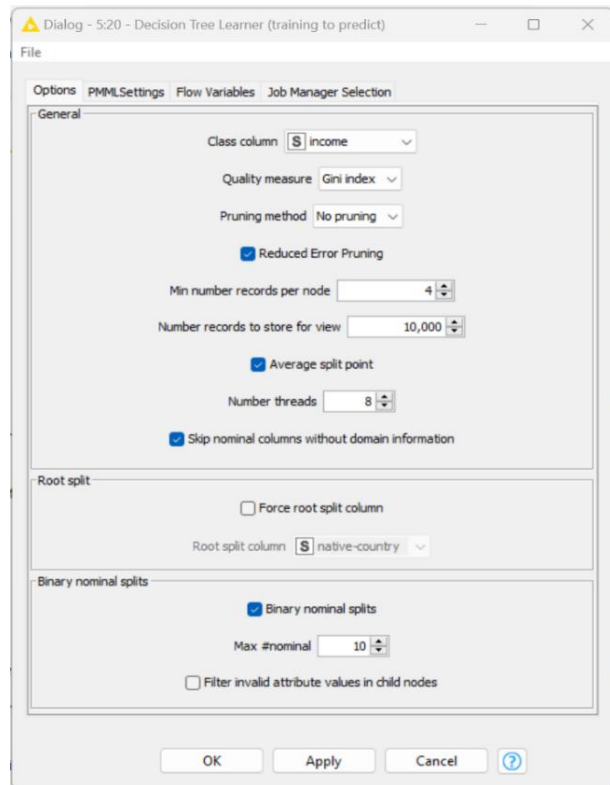


Figure 4.20. Decision Tree Learner node: the “Options” tab.

Decision Tree Learner: “PMML Settings” Tab

The configuration window of the “Decision Tree Learner” node offers two tabs: “Options” (described above) and “PMML Settings”. The “PMML Settings” tab deals with settings for the final PMML model.

How to deal with the no true child problem

Sometimes, the evaluation process reaches a node in the tree for which the required attribute shows an out of training domain value. In this case, for the predicted class you can:

- Use the majority class in the previous node (“returnLastPrediction” option)
- Return a missing value (“returnNullPrediction” option)

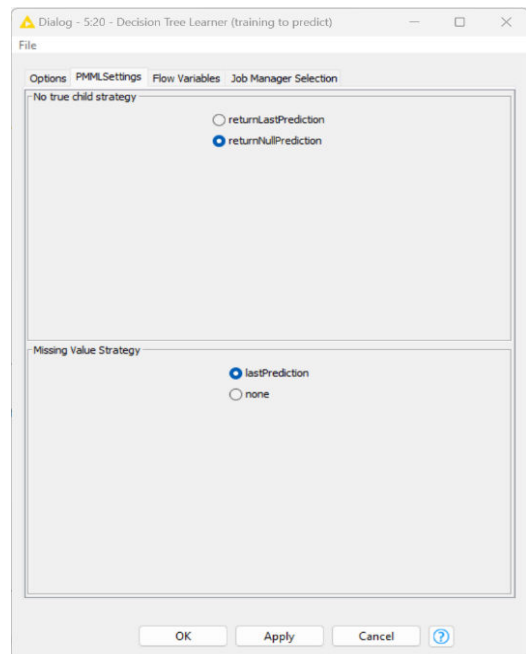


Figure 4.21. Decision Tree Learner node: the “PMML Settings” tab.

How to deal with missing values

Sometimes, the evaluation process reaches a node in the tree for which the required attribute shows a missing value. In this case, for the predicted class you can:

- Use the majority class in the previous node (“lastPrediction” option)
- Revert to the no true child strategy (“none” option)

We trained the “Decision Tree Learner” node with:

- Class column = “income”
- Gini Index as quality measure
- Pruning = No Pruning
- Stopping criterion = 4 data points per node
- Number of records for hiliting = 10000
- Split point calculated as the average point between the two partitions

- Binary splits for nominal values
- Maximum number of distinct nominal value allowed in a column = 10
- 8 as number of threads, since we are working on a 4-core machine

We can now run the “Execute” command and therefore train our decision tree model. At the end of the training phase the model is available at the output port (blue square) of the “Decision Tree Learner” node.

The “Decision Tree Predictor” node has only one output table, consisting of the original data set with the appended prediction column and optionally the columns with the probability for each class, like all other predictor nodes. The “Decision Tree Predictor” node was introduced to get the test data from the “CSV Reader” node and the model from the “Decision Tree Learner” node, with the option to append the normalized class distribution at the end of the prediction data table.

Decision Tree Predictor

The “Decision Tree Predictor” node imports a Decision Tree model from the input port and applies it to the input data table.

In the configuration window you can:

- Define the maximum number of records for highlighting (again a heritage of the old “Data Views” visualization nodes)
- Define a custom name for the output column with the predicted class
- Append the columns with the normalized distribution of each class prediction to the output data set

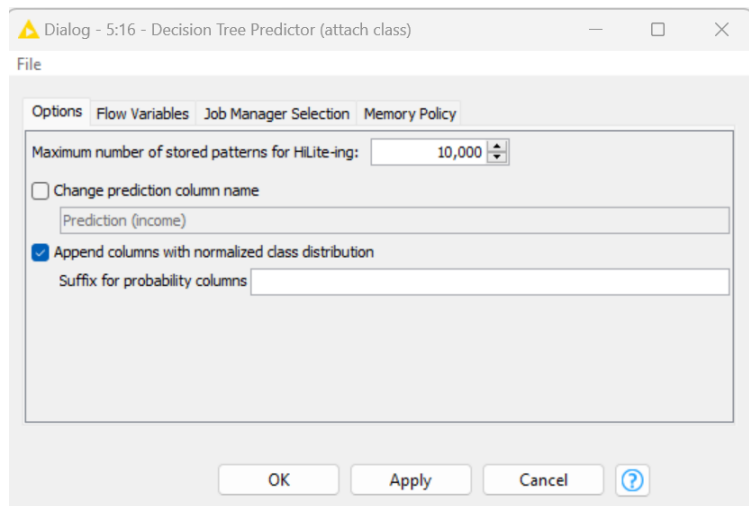


Figure 4.22. Configuration window of the Decision Tree Predictor node.

Decision Tree Views

In the context menu of both the “Decision Tree Predictor” node and the “Decision Tree Learner” node, we can see two options to visualize the decision tree rules:

- “View: Decision Tree View (simple)”
- “View: Decision Tree View ”
- Sub-category “Mining” → “Decision Tree” also includes a “Decision Tree To Image” node and a “Decision Tree to Ruleset” node. The “Decision Tree To Image” node converts the view of the decision tree model into an image. The “Decision Tree to Ruleset” node converts the decision tree splits into a set of rules.

RowID	age Number (float)	W... Str	fn... Str	edu... Str	ma... Str	oc... Str	FE... Str	ra... Str	sex... Str	ra... Str	eth... Str	inc... String	P (income<=50K) Number (float)	P (income>50K) Number (float)	Prediction (income) String			
1	Row0 0.83	Self-e...	-1.012	Bach...	1.131	Marri...	Exec...	Husb...	White	Male	-0.146	-0.211	-2.208	Unit...	<=50K	0.714	0.285	<=50K
2	Row1 1.05	Private	0.418	11th	-1.193	Marri...	Handl...	Husb...	Black	Male	-0.146	-0.211	-0.038	Unit...	<=50K	0.961	0.039	<=50K
3	Row2 -0.122	Private	0.889	Mast...	1.518	Marri...	Exec...	Wife	White	Female	-0.146	-0.211	-0.038	Unit...	<=50K	0	1	>50K
4	Row3 0.976	Self-e...	0.181	HS-gr...	-0.419	Marri...	Exec...	Husb...	White	Male	-0.146	-0.211	0.364	Unit...	>50K	0.846	0.154	<=50K
5	Row4 0.244	Private	0.298	Bach...	1.381	Marri...	Exec...	Husb...	White	Male	0.556	0.211	0.038	Unit...	>50K	0.006	0.994	>50K

Figure 4.23. Output data table from the Decision Tree Predictor node.

Let’s have a look at the decision tree views.

The more complex view (“Decision Tree View”) displays each branch of the decision tree as a rectangle. The data covered by this branch are shown inside the rectangle and the rule implementing the branch is displayed on top of the rectangle.

In the view, a branch can show a little sign “+” indicating the possibility to expand the branch with more nodes.

The decision tree always starts with a “Root” branch that contains all training data. The “Root” branch in our decision tree is labeled “<=50K”, because it covers 11490 “<=50K” records out of 153362 from the training set. A majority vote is used to label each branch of the decision tree.

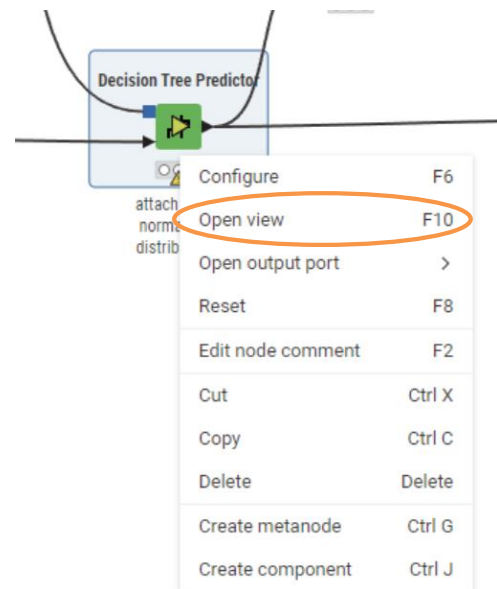


Figure 4.24. Context Menu of the Decision Tree Predictor node.

The first split happens in the “relationship” column. From the “Root” branch the data rows are separated into a number of sub-branches according to their value of the “relationship” attribute. Each branch is labeled with a predicted class coming from its cover factor. For example, the branch defined by the split condition “relationship = Wife, Husband” is labeled as “<=50K” since during the training phase it covered 3788 “<=50K” records out of its incoming 7074 training patterns. Fraction values in the total number of training patterns can occur when missing values are encountered during training. In this event only fractions of the patterns are passed down to the following branches.

Inside each branch more splits are performed, and data rows are separated into different branches and so on, deeper and deeper into the tree, until the final leaves. The final leaves produce the final prediction/class.

If you click on “open view”, a branch of the decision tree can be selected by clicking it. Selected branches are shown with a black rectangle border (simple view) or with a darker background (complex view). A selection of multiple branches is not possible.

The “Decision Tree View” window has a top menu with three items.

“**File**” has the usual options:

- “Always on top” ensures that this window is always visible
- “Export as PNG or SVG” exports this window as a picture to be used in a report for example
- “Close” closes the window

“**Hilite**” contains the hilite commands to work together with the “Data Views” nodes.

“**Tree**” offers the commands to expand and collapse the tree branches:

- “Expand Selected Branch” opens the sub-branches, if any, of a selected branch of the tree
- “Collapse Selected Branch” closes the sub-branches, if any, of a selected branch of the tree

On the right side, there is an overview of the decision tree. This is particularly useful if the decision tree is big and very bushy. In the same panel on the bottom, there is a zoom functionality to explore the tree with the most suitable resolution.

The “Scorer” node at the end measures the success performance for the decision tree as well, which amounts to 83% accuracy and 53% Cohen’s kappa. The Naive Bayes classifier produced 81% accuracy and 54% Cohen’s kappa. This means that the two model performances are

comparable, even though the decision tree performs slightly better on one of the two classes, probably the more populated one.

Another possible visualization for a decision tree consists of the interactive view produced by the “Decision Tree View” node. This is another one of the JavaScript based visualization nodes and it is dedicated to visualize the splits in a decision tree model (Figure 4.25).

Another possible evaluation of the model performance could be achieved through an ROC curve. Actually, both models, the naïve Bayes and the decision tree, could be evaluated and compared by means of an ROC curve. Of course, there is a JavaScript based node that produces an interactive visualization of a number of ROC curves. To draw an ROC curve, the target classification column has to contain two class labels only. One of them is identified as the positive class. A threshold is then incrementally applied to the column containing the probabilities for the positive class, therefore defining the true positives rate and the false positives rate for each threshold value⁴. At each step, the classification is performed as:

```
IF Probability of positive class > threshold => positive class
ELSE => negative class
```

The ROC Curve, in particular the area below the ROC curve, gives an indication of the predictor performance. It is possible to display multiple curves for different columns in the ROC Curve View if we want to compare the performances of more than one classifier. From Figure 4.28

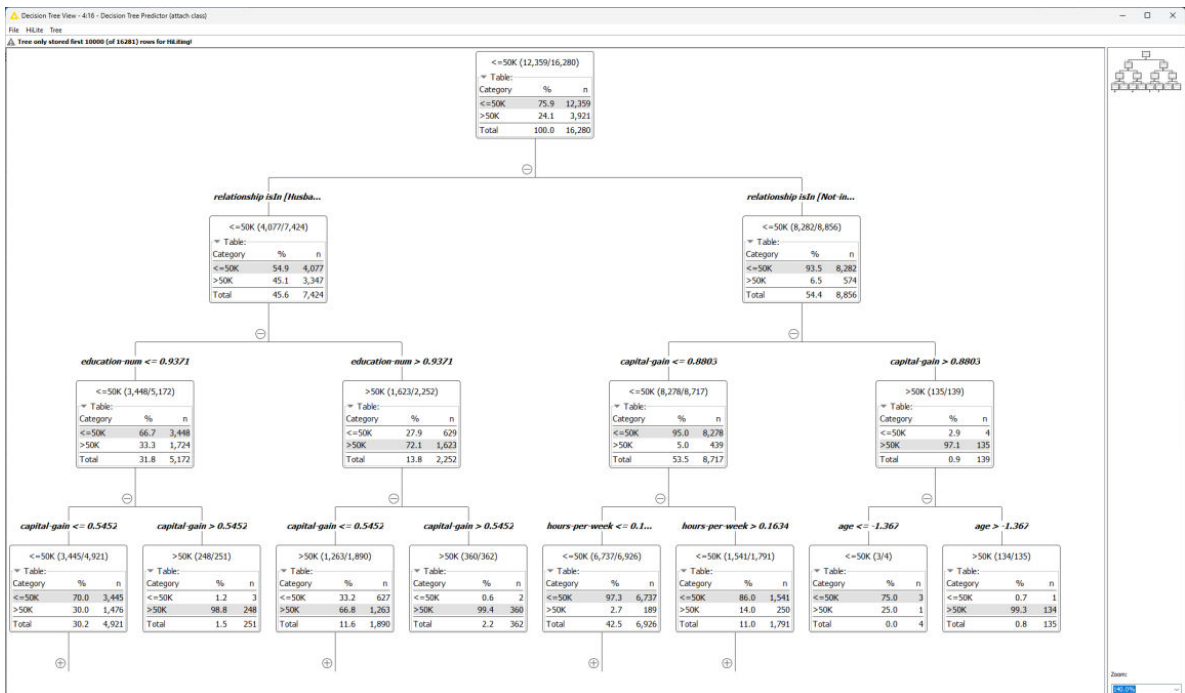


Figure 4.25. View of the decision tree model created by the “Decision Tree Learner” node.

we can see that the two classification models have very similar performances (Area under the Curve = 89% for Naïve Bayes, Area under the Curve = 85% for the decision tree).

As for all JavaScript based visualization nodes, the configuration window of this node contains three tabs: “Decision Tree Plot Options”, “General Plot Options”, and “View Controls”.

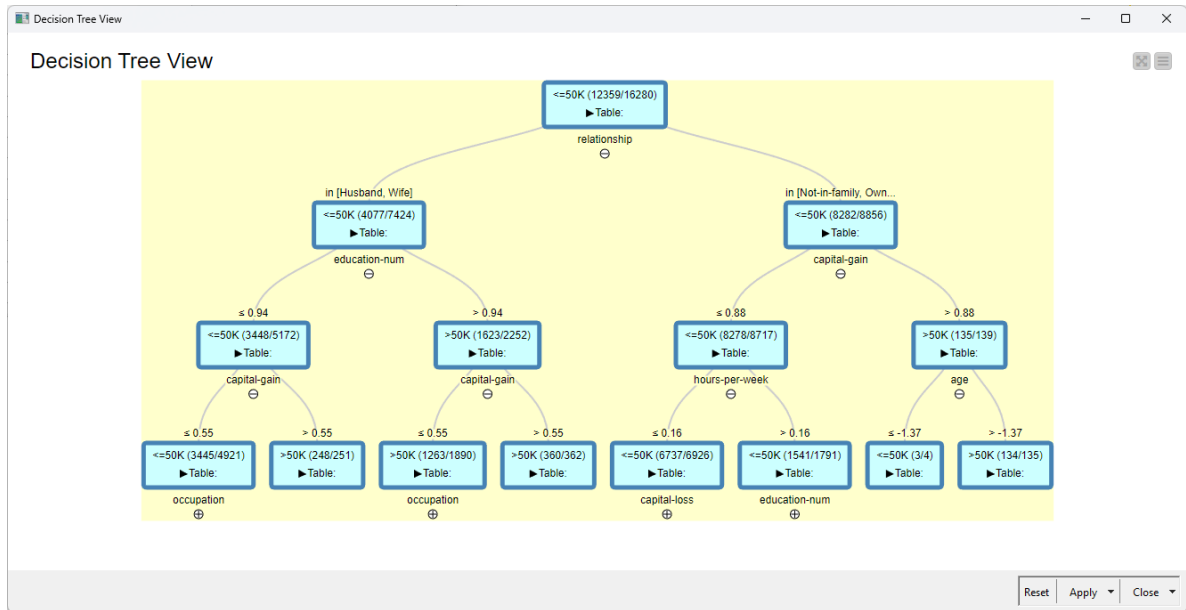


Figure 4.26. View of the decision tree model as produced by the “Decision Tree View” node.

“Decision Tree Plot Options” tab defines the content to plot, such as the number of rows and the number of levels to be expanded already at view opening. Of course, the higher the number of rows to visualize, the slower the node execution. It also contains the flag to create an image from the produced view.

“Decision Tree Plot Options” tab defines general plot properties, such as background color, tree area background color, node color, title and subtitle, and formatting.

“View Controls” tab sets the plot interactivity, like zoom and title and subtitle editing.

Figure 4.26 shows a possible final view of the decision tree generated with a “Decision Tree View” node.

In this example predictions by the decision tree and the Naïve Bayes algorithms are combined

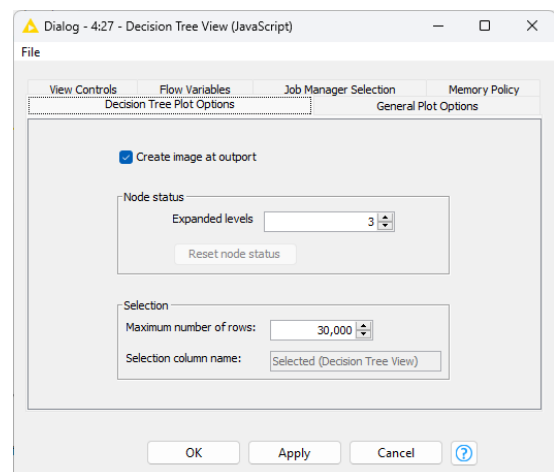


Figure 4.27. Configuration window of the Decision Tree View node: the “Decision Tree Plot Options” tab.

together via a “Joiner” node. We will omit details about the “Joiner” node in this chapter and we will explain this node more in detail in the next chapter. For now, it is enough to know that the “Joiner” node appends collates together columns from two input tables by matching key values in selected columns.

ROC Curve

The “ROC Curve” node draws a number of ROC curves for a two-class classification problem. The configuration window covers four tabs: “ROC Curve Settings”, “General Plot Options”, “Axis Configuration”, and “View Controls”.

ROC Curve Settings

- The column containing the reference class
- The positive value of the class (arbitrarily assumed as positive)
- The column(s) with the probabilities for the positive class
- The limit on the number of points to plot. Remember less points less accurate curve, more points slower execution.
- The selection of the columns with the probabilities for the positive class is performed by means of an “Excludes”/“Includes” frame.

General Plot Options: Here all plot settings are required: image size, formatting, background colors, etc.

Axis Control: Contains all settings about the plot axis.

View Controls: Define the level of interactivity of the curve view, such as label or title editing.

Chapter 4: My First Model

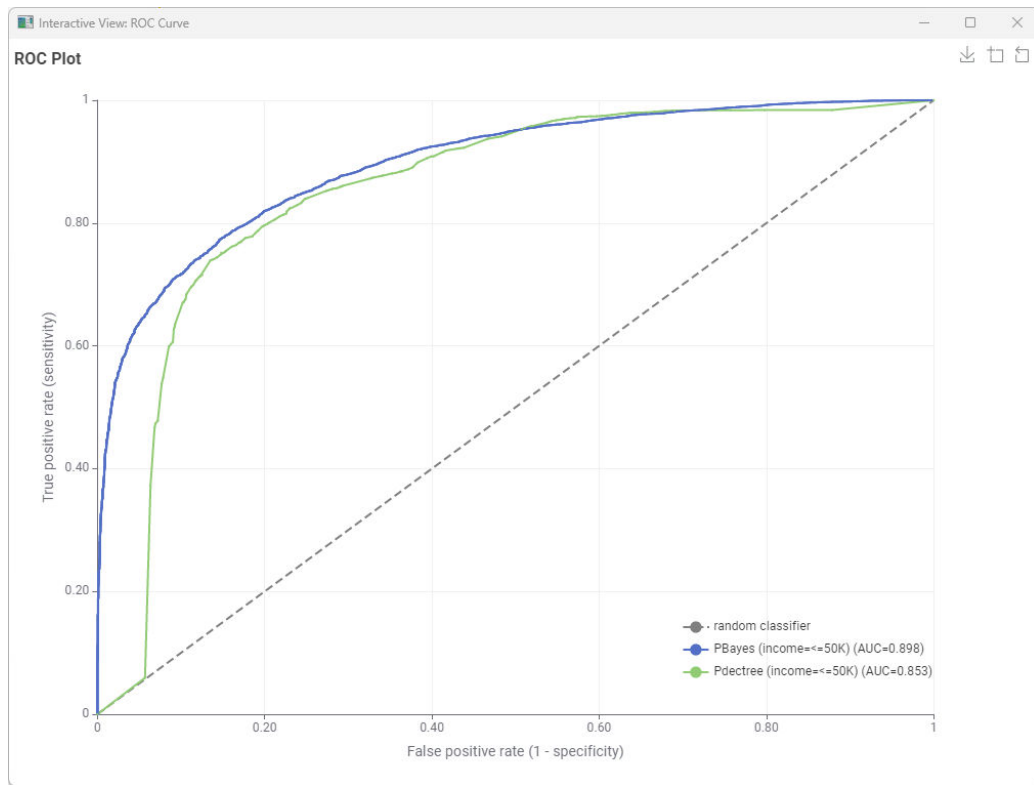
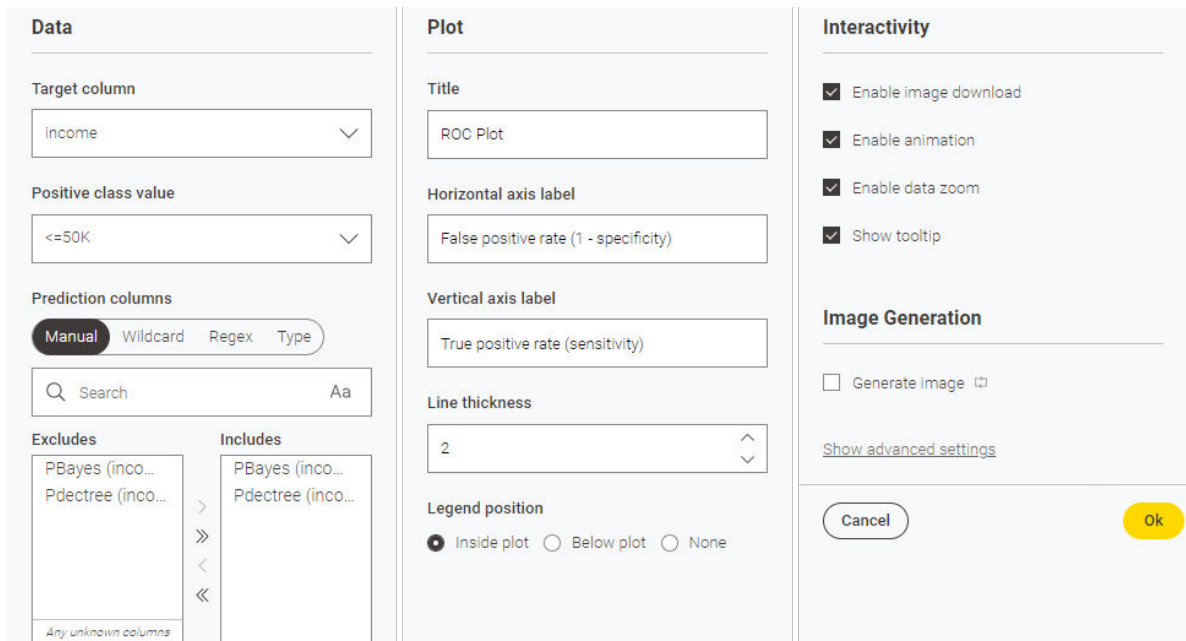


Figure 4.28. View of the ROC Curve node.

The node outputs the image (optionally) of the produced ROC curve and the Area under the Curve (AuC) for the probability columns.

Chapter 4: My First Model



The configuration window is divided into three main sections: Data, Plot, and Interactivity.

- Data:**
 - Target column:** A dropdown menu with 'income' selected.
 - Positive class value:** A dropdown menu with '<=50K' selected.
 - Prediction columns:** Includes tabs for 'Manual', 'Wildcard', 'Regex', and 'Type'. Below is a search bar with 'Aa' and a list of columns in 'Excludes' and 'Includes' panels. 'PBayes (inco...' and 'Pdectree (inco...' are listed in both.
- Plot:**
 - Title:** A text input field containing 'ROC Plot'.
 - Horizontal axis label:** A text input field containing 'False positive rate (1 - specificity)'.
 - Vertical axis label:** A text input field containing 'True positive rate (sensitivity)'.
 - Line thickness:** A dropdown menu with '2' selected.
 - Legend position:** Radio buttons for 'Inside plot' (selected), 'Below plot', and 'None'.
- Interactivity:**
 - Four checked checkboxes: 'Enable image download', 'Enable animation', 'Enable data zoom', and 'Show tooltip'.
 - Image Generation:** An unchecked checkbox for 'Generate image' with a tooltip icon.
 - A link for 'Show advanced settings'.
 - 'Cancel' and 'Ok' buttons at the bottom.

Figure 4.29. Configuration window of the ROC Curve node.

Workflow: My First Model

This workflow reads the data sets written by the Prepare Data workflow. It then trains two machine learning algorithms:

- Naive Bayes
- Decision Tree (C4.5)

to predict Income (>50k / <=50K) based on the remaining input features.

Finally, it scores both models on the test set.

The last section is about visualization: ROC Curves and decision tree visualization.

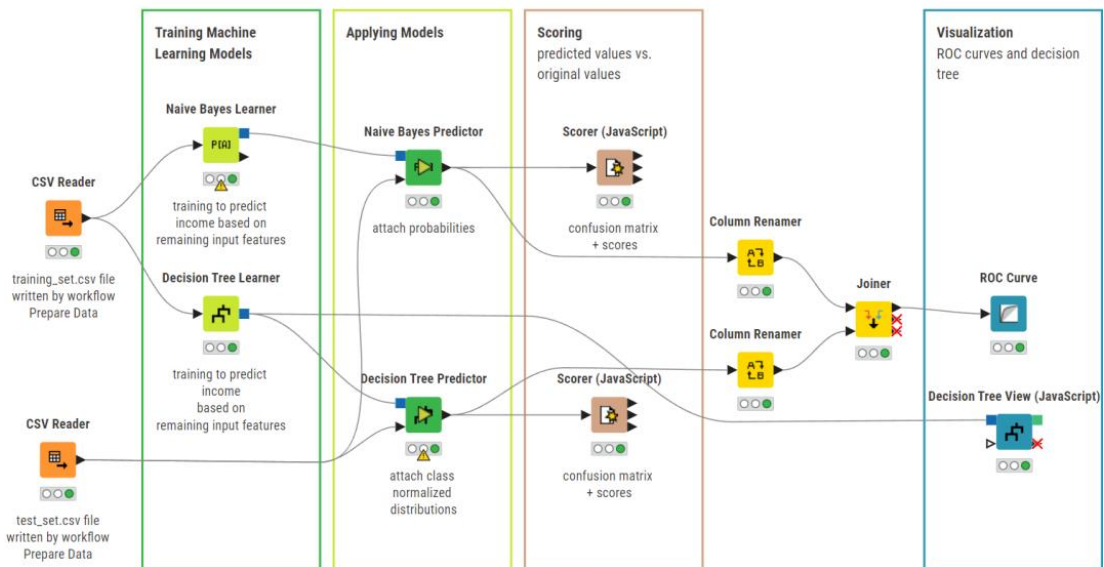


Figure 4.30. Workflow "My First Model".

Artificial Neural Network

We move on now to a neural network and specifically to a Multilayer Perceptron (MLP) architecture, with one hidden layer, and the Back Propagation learning algorithm. The neural network paradigm is available in the "Mining" category and consists of:

- A learner node ("RProp MLP Learner")
- A predictor node ("Multilayer Perceptron Predictor")

The learner node learns the rules to separate the input patterns of the training set, packages them into a model, and assigns them to the output port.

The predictor node applies the rules from the model built by the learner node to a data set with new records.

RProp MLP Learner

The “RProp MLP Learner” node builds and trains a Multilayer Perceptron with the BackPropagation algorithm. In the configuration window you need to specify:

- The maximum number of iterations for the Back Propagation algorithm
- The number of hidden layers of the neural architecture
- The number of neurons per each hidden layer
- The class column, i.e. the column containing the target classes. The class column has to be of type String (nominal values)
- You also need to specify what to do with missing values. The algorithm does not work if missing values are present. If you have missing values you need to either transform them earlier on in your workflow or ignore them during training. To ignore the missing values just mark the corresponding checkbox in the configuration window.
- Finally, you need to specify a seed to make the weight random initialization repeatable.

The “RProp MLP Learner” only accepts numerical inputs. String data columns will not be processed as input attributes.

We created a new workflow in the workflow group “Chapter4” and named it “My First ANN”. We also used the data sets “training set” and “test set” derived from the adult.data data set in the “Data Preparation” workflow. We set the classification/prediction task to predict the kind of income each person/record has. “Income” is a string column with only two values: “>50K” and “<=50K”.

First, we inserted two “CSV Reader” nodes: one to read the training set and one to read the test set prepared by the “Data Preparation” workflow earlier on in this chapter.

Of all the string attributes in the adult data set, we decided to keep only the attribute “sex”, since we think that sex is an important discriminative variable in predicting the income of a person. Of course we also kept the “Income” column to be the reference class. We removed all other string attributes.

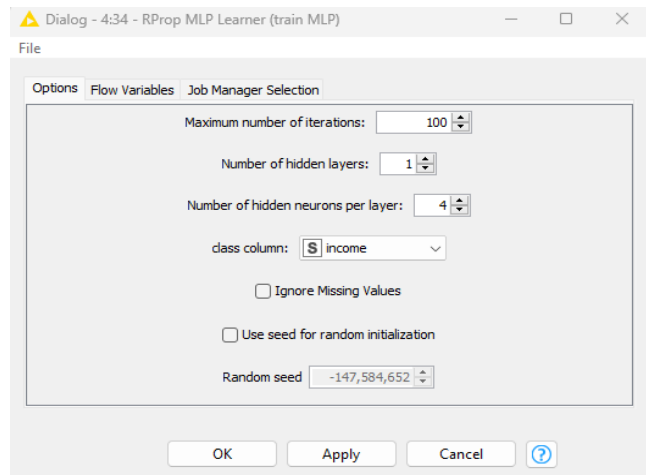


Figure 4.31. Configuration window of the RProp MLP Learner node.

The attribute “sex”, being of type String, could not be used as it was and it has been converted into a binary variable “sex_01”, according to the following rule:

IF	\$sex\$	=	“Male”	=>	\$sex_01\$	=	“-1”
IF	\$sex\$	=	“Female”	=>	\$sex_01\$	=	+1”

In order to implement this rule, we used a “Rule Engine” node. “sex_01” is the newly created Integer column containing the binary values for sex. We then used a “Column Filter” node to exclude all remaining string columns besides “Income”.

Multilayer Perceptron requires numerical data in the [0,1] range. In order to comply with that, a “Normalizer” node was placed after the “Column Filter” node to normalize all numerical data columns to fall into the range [0,1]. This sequence of transformations (“Rule Engine” on “sex”, “Column Filter” to keep only numerical attributes and the “Income” data column, and the [0,1] normalization”) was applied on both training and test set.

We then applied the “RProp MLP Learner” node to build an MLP neural network with 6 input variables (age, education-num, fnlwgt, capital-gain, capital-loss, hours-per-week, sex_01), 1 hidden layer with 4 neurons, and 2 output neurons, i.e. one output neuron for each “Income” class. We trained it on the training set data with a maximum number of iterations of 100.

After training, we applied the MLP model to the test set’s data by using a “Multilayer Perceptron Predictor” node. The neural network’s predictor node applies the rules from the model built by the learner node to a data set with new records. The predictor node has two input ports:

- A data input (black triangle) with the new data to be classified
- A model input (blue square) with the model parameters produced by a “RProp MLP Learner” node

The predictor node has one output port, where the original data set plus the predicted classes and optionally the class distributions are produced.

Multilayer Perceptron Predictor

The “Multilayer Perceptron Predictor” node takes an MLP model, generated by an “RProp MLP Learner” node, at the model input port (blue square) and applies it to the input data table at the input data port (black triangle).

The “Multilayer Perceptron Predictor” node can be found in the “Node Repository” in the “Analytics” → “Mining” → “Neural Network” → “MLP” category.

The only settings required for its configuration, like for all other predictor nodes, are a checkbox to append the normalized class distributions to the input data table and a possible customized name for the output class column.

Classified Data

Let's visualize the results of the MLP Prediction:

- Right-click the "Multilayer Perceptron Predictor" node
- Select "Classified data"

The "Classified Data" data table contains the final predicted classes in the "Prediction (income)" column and the values of the two output neurons in the columns "P (Income >50K)" and "P(Income<=50K)".

The firing value of the two output neurons is represented by a red-green bar instead of a double number. Red means a low number (< 0.5), green a high number (> 0.5). The highest firing output neuron decides the prediction class of the data row.

To change the rendering of the neuron firing values, you right-click the column header and select a new rendering under "Available Renderers", like for example "Standard Double".

We have used the Neural Network paradigm in a two-class classification problem ("Income > 50K" or "Income <=50K"). We can now apply an "ROC Curve" node to the results of the "Multilayer Perceptron Predictor" node. We identified:

- The class column as column "Income"
- The positive value "<=50K" in class column "Income"
- The column "P(Income <=50K)" as the column containing the probability/score for the positive class

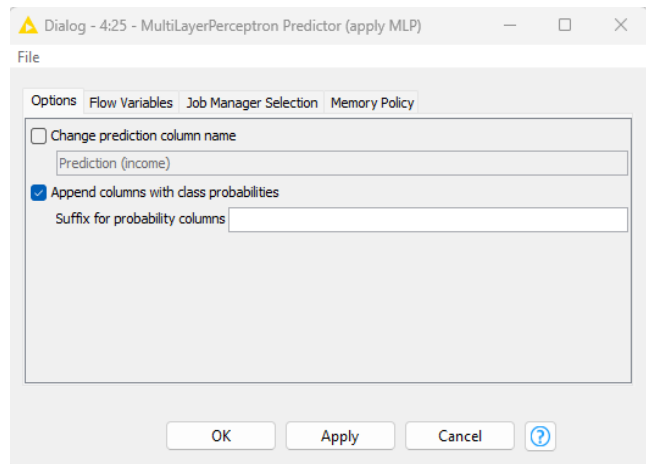


Figure 4.32. Configuration window of the Multilayer Perceptron Predictor node.

The resulting ROC Curve shows an Area under the Curve around 0.85.

Row ID	D age	D fhvgt	D educati...	D capital-...	D capital-...	D hours-p...	S income	D sex_01	D P (income=<=50K)	D P (income=>50K)	S Prediction (income)
Row0	0.452	0.049	0.8	0	0	0.122	<=50K	0			<=50K
Row1	0.493	0.154	0.4	0	0	0.398	<=50K	0			<=50K
Row2	0.274	0.189	0.867	0	0	0.398	<=50K	1			<=50K
Row3	0.479	0.137	0.533	0	0	0.449	>50K	0			<=50K
Row4	0.342	0.102	0.8	0.052	0	0.398	>50K	0			>50K
Row5	0.178	0.089	0.8	0	0	0.398	>50K	0			<=50K
Row6	0.205	0.134	0.733	0	0	0.5	<=50K	0			<=50K
Row7	0.233	0.162	0.2	0	0	0.449	<=50K	0			<=50K
Row8	0.205	0.121	0.533	0	0	0.398	<=50K	0			<=50K
Row9	0.356	0.194	0.867	0	0	0.449	>50K	1			<=50K
Row10	0.507	0.201	0.533	0	0	0.194	<=50K	1			<=50K
Row11	0.356	0.073	0.4	0	0.469	0.398	<=50K	0			<=50K
Row12	0.534	0.142	0.8	0	0	0.398	>50K	0			>50K
Row13	0.507	0.116	0.6	0	0	0.602	>50K	0			<=50K
Row14	0.438	0.125	0.533	0	0	0.398	<=50K	0			<=50K
Row15	0.041	0.176	0.6	0	0	0.439	<=50K	0			<=50K
Row16	0.178	0.033	0.6	0	0	0.398	<=50K	0			<=50K
Row17	0.425	0.159	0.4	0	0	0.398	<=50K	0			<=50K
Row18	0.027	0.369	0.533	0	0	0.245	<=50K	1			<=50K
Row19	0.425	0.175	0.733	0	0	0.398	<=50K	0			>50K
Row20	0.493	0.053	0.8	0	0	0.398	<=50K	0			>50K
Row21	0.438	0.057	0.533	0	0	0.398	<=50K	1			<=50K
Row22	0.548	0.226	0.8	0	0	0.398	>50K	0			>50K
Row23	0.37	0.08	0.867	0	0	0.398	<=50K	1			<=50K
Row24	0.164	0.18	0.667	0	0	0.429	<=50K	0			<=50K
Row25	0.014	0.149	0.533	0	0	0.296	<=50K	1			<=50K
Row26	0.452	0.166	0.8	0	0	0.551	>50K	0			>50K
Row27	0.356	0.156	0.6	0	0	0.398	>50K	0			<=50K
Row28	0.247	0.031	0.667	0	0	0.398	<=50K	0			<=50K
Row29	0.178	0.122	0.533	0.05	0	0.398	<=50K	0			<=50K
Row30	0.205	0.195	0.2	0	0	0.398	<=50K	0			<=50K
Row31	0.342	0.072	1	0	0	0.449	>50K	0			>50K
Row32	0.26	0.099	0.533	0	0	0.398	<=50K	0			<=50K
Row33	0.493	0.109	0.533	0	0	0.398	>50K	1			<=50K
Row34	0.11	0.131	0.6	0	0	0.398	<=50K	0			<=50K
Row35	0.192	0.206	0.8	0	0	0.398	<=50K	1			<=50K
Row36	0.082	0.138	0.6	0	0	0.398	<=50K	0			<=50K
Row37	0.137	0.14	0.533	0	0	0.398	<=50K	0			<=50K

Figure 4.33. Output Table of the Multilayer Perceptron Predictor node.

Write/Read Models to/from File

Once we have trained a model and ascertained that it works well enough for our expectations, it would be nice if we could reuse the same model in other similar applications on new data. This means that we should be able to recycle the model in other workflows as well.

Model Writer

The “Model Writer” node takes a model at the input port (gray square) and writes it into a file by using the KNIME internal format. The “Model Writer” node is located in the “IO” → “Write” category in the “Node Repository” panel.

The configuration window only requires:

Chapter 4: My First Model

- The path of the output file (*.zip) (knime:// protocol is also accepted)
- The flag to override the file, if the file exists

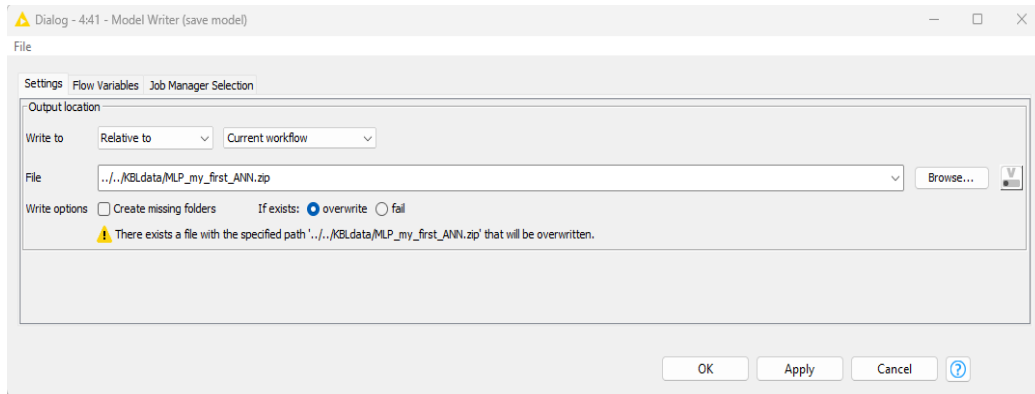


Figure 4.34. Configuration window of the Model Writer node.

The final workflow “My First ANN” is shown in the figure below.

At the same time, KNIME also provides a node to read a model from a file, the “Model Reader” node, located in the “IO” → “Read” category in the “Node Repository” panel.

Workflow: My First ANN

This workflow reads the data sets written by the Prepare Data workflow. It then trains two machine learning algorithms:

- Naive Bayes
- Decision Tree (C4.5)

to predict Income (>50k / <=50K) based on the remaining input features. Finally, it scores both models on the test set.

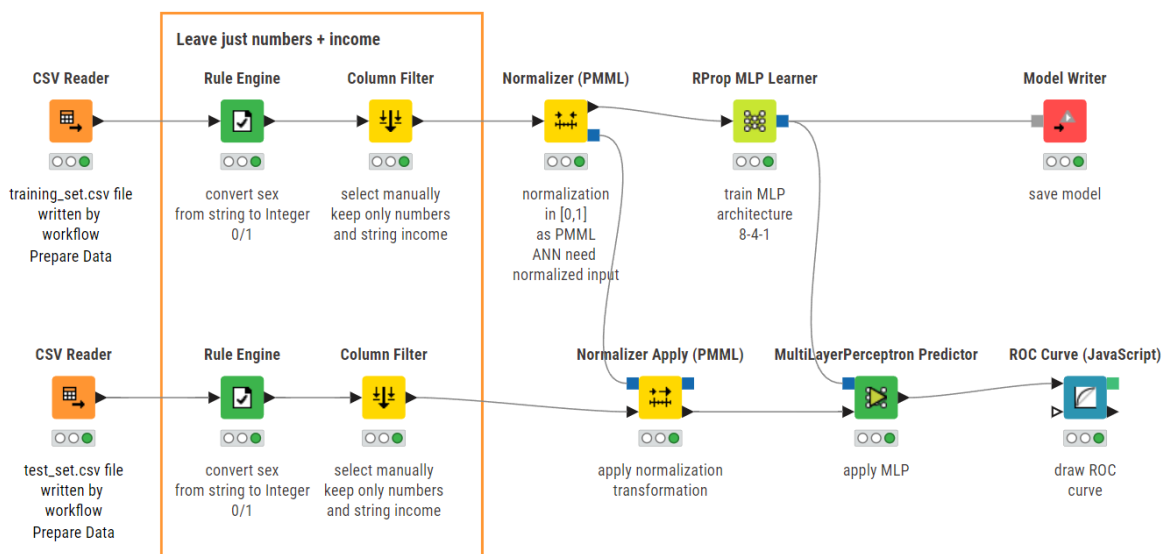


Figure 4.35. Workflow "My First ANN".

Model Reader

The "Model Reader" node reads a model from a file using the KNIME internal format and makes it available at the output port (gray square).

The configuration window only needs:

- The path of the input file (*.pmml) (knime:// protocol is also accepted)

Drag and drop of a model file from a data folder automatically creates a "Model Reader" node with the right configuration settings.

In this last part of the chapter, we would like to show a few more nodes that are commonly used in data analytics. We will build a new workflow, named "Clustering and Regression", in the workflow group "Chapter4" to explain these nodes.

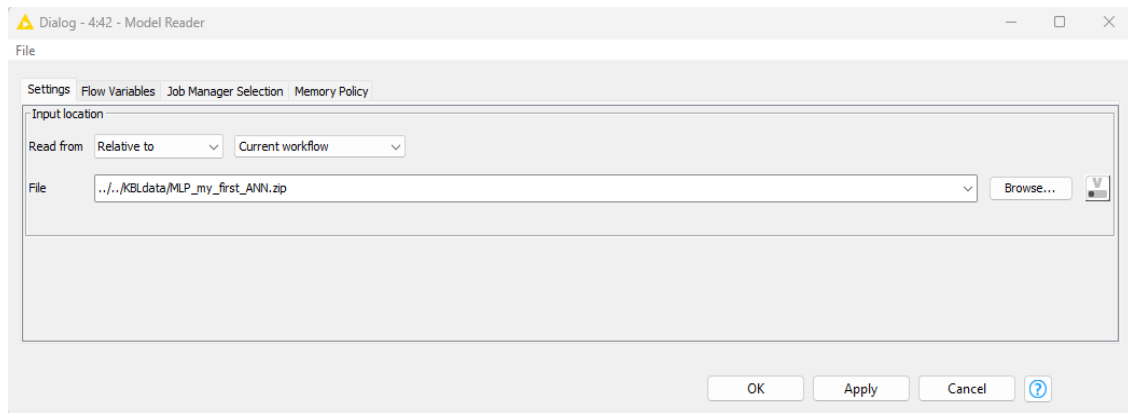


Figure 4.36. Configuration window of the Model Reader node.

We will use the same data that we used for the previous two workflows, “training set” and “test set”, created in the “Data Preparation” workflow. The first two nodes in the workflow will then be two “CSV Reader” nodes, one to read the training set and one to read the test set data, as in the previous workflows.

Statistics

The “Statistics” node calculates statistic variables on the input data, such as:

For numerical columns (available in a table on output port 0):

- minimum
- maximum
- Mean
- Standard deviation
- skewness
- Histogram
- variance
- median
- overall sum
- kurtosis
- number of NaN/missing values

For nominal columns (available in a table on output port 1 and 2):

- number of occurrences of nominal values
- Number of missing values
- Histogram of nominal values

The “Statistics” node is located in the “Node Repository” panel in the “Analytics” → “Statistics” category. The configuration window requires:

- The selection of the nominal columns on which to calculate the statistical measures (the statistical measures for numerical variables are calculated on all numerical columns by default).
- The maximum number of most frequent and infrequent values to display in the view
- The maximum number of possible values per column. This is to avoid long lists of nominal values.
- Whether to calculate the median value
- All the statistical measures, described in the table above, are available at the node output ports as well as in the node View.
- Selection of the input data columns is performed by means of the column selection framework: by manual selection with “Include/Exclude” panels; by type selection, by Wildcard/Regex expression selection.

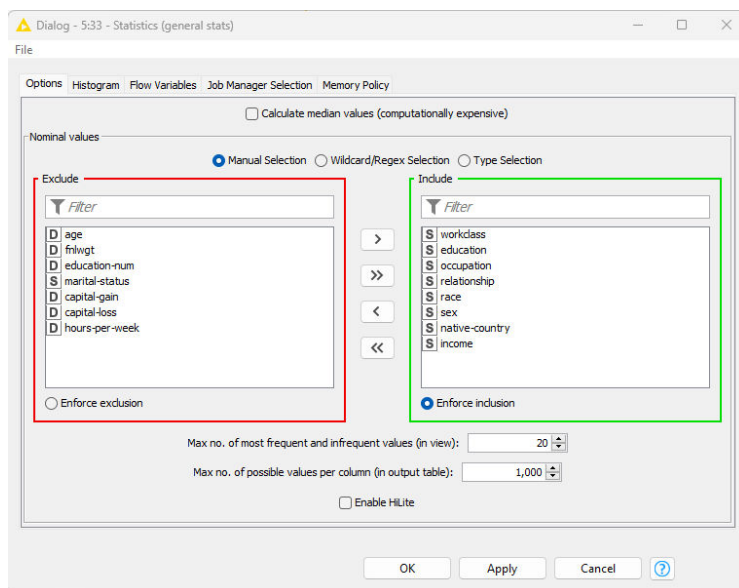


Figure 4.37. Configuration window of the Statistics node.

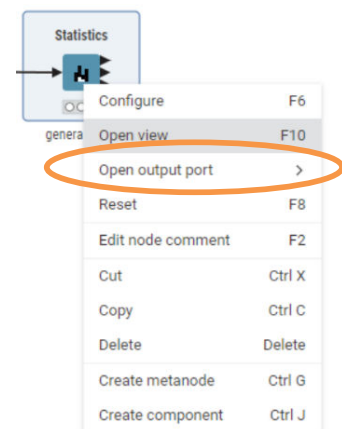


Figure 4.38. Context menu of the Statistics node.

Chapter 4: My First Model

The “Statistics” node has one visualization option: the “Open View” and the data tables on the output ports. The data tables can be accessed from the node monitor below.

The node has three output ports and the “Statistics View” has three specular tabs. Output port “Statistics Table” corresponds to tab “Numeric” in the view; output port “nominal Statistical Values” corresponds to tab “Nominal” in the view; and output port “Occurrences Table” corresponds to tab “Top/Bottom” in the view.

Tab “Numeric” contains a number of statistical measures calculated on all numerical columns, with an approximate histogram. Each row then with all the statistical measures and the rough histogram offers an idea of the statistical properties and distribution of the values in a numeric data column.

Similarly, the “Nominal” and the “Top/Bottom” tabs give an idea of the statistical properties of the values in a nominal data column.

Note. The statistics of nominal columns are calculated only for the nominal columns included in the configuration window.

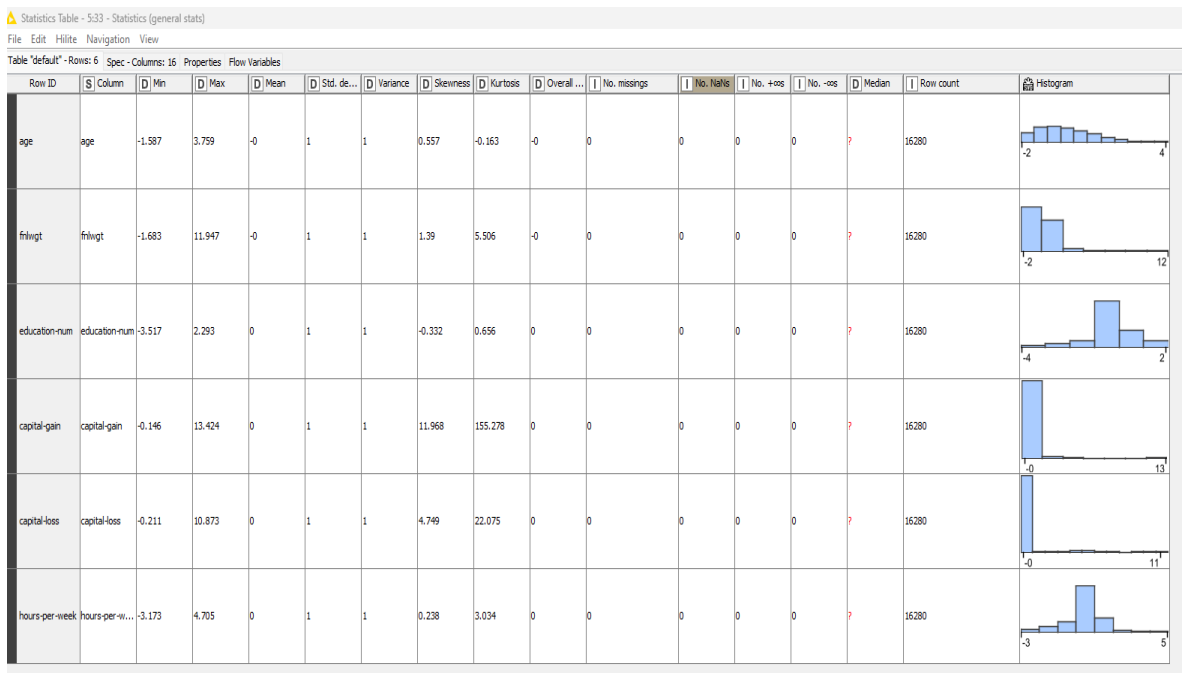


Figure 4.39. The Numeric tab of the Statistics node view displays statistical measures and histogram calculated on all numerical columns.

In our workflow’s node we excluded the column “marital-status” from the nominal columns and the corresponding column pair “marital-status” and “marital status_Count” is not in the “Occurrences Table”.

In “Statistics View” → tab “Nominal Columns” we find the same information, but the lists of nominal values are sorted by frequency. For each column we find two table cells: one at the top for the most frequent (top 20) nominal values in the column and one at the bottom for the least frequent (bottom 20) nominal values in the column.

Row ID	S workclass	I Count (workclass)	D Relativ...	S education	I Count (...)	D Relativ...	S occupa...
Row0	Private	11305	0.694	HS-grad	5278	0.324	Prof-specialty
Row1	Self-emp-not-inc	1271	0.078	Some-college	3607	0.222	Craft-repair
Row2	Local-gov	1030	0.063	Bachelors	2748	0.169	Exec-manag...
Row3	?	940	0.058	Masters	836	0.051	Adm-clerical
Row4	State-gov	657	0.04	Assoc-voc	668	0.041	Sales
Row5	Self-emp-inc	568	0.035	11th	592	0.036	Other-service
Row6	Federal-gov	499	0.031	Assoc-acdm	538	0.033	Machine-op-...
Row7	Without-pay	6	0	10th	450	0.028	?
Row8	Never-worked	4	0	7th-8th	331	0.02	Transport-m...
Row9	?	?	?	Prof-school	275	0.017	Handlers-cle...
Row10	?	?	?	9th	257	0.016	Farming-fsh...
Row11	?	?	?	Doctorate	212	0.013	Tech-support

Figure 4.40. The “Occurrence Table” contains the number of occurrences of nominal values calculated only on the selected nominal columns.

Regression

Another very common task in data analysis is the calculation of the linear regression^{2,3,4}. In the “Node Repository” panel, in the “Analytics” → “Statistics” → “Regression” category, there are two learner nodes to learn the regression parameters: one node performs a multivariate linear regression, the other node a multivariate polynomial regression. Both regression learner nodes share the predictor node. Regression Learner nodes have two input ports and two output ports. At input, the node is fed with the training data and optionally with a pre-existing model. After execution, the node produces the regression model and the statistical properties of the model in a data table. The predictor node takes the regression model, linear or polynomial, as input and applies it to new input data rows to predict their response. In this book, we will only show how to implement the linear regression.

The models we have seen so far were classifiers; that is, they were trying to predict nominal values (classes) for each data row. The linear regression is a fitting model; that is a model that tries to predict numerical values. In this case, the target data column must be a numerical column with numerical values to be approximated through the linear regression fitting.

Chapter 4: My First Model

workclass	education	occupation	relationship	race	sex	native-country	income
No. missings: 0	No. missings: 0	No. missings: 0	No. missings: 0	No. missings: 0	No. missings: 0	No. missings: 0	No. missings: 0
Top 20:	Top 20:	Top 20:	Top 20:	Top 20:	Top 20:	Top 20:	Top 20:
Private : 11305	HS-grad : 5278	Prof-specialty : 2086	Husband : 6639	White : 13918	Male : 10879	United-States : 14560	=50K : 12359
Self-emp-not-inc : 1271	Some-college : 3607	Craft-repair : 2055	Not-in-family : 4144	Black : 1538	Female : 5401	Mexico : 326	>50K : 3921
Local-gov : 1030	Bachelors : 2748	Exec-managerial : 1985	Own-child : 2510	Asian-Pac-Islander : 519	? : 298	Philippines : 94	
? : 940	Masters : 836	Adm-clerical : 1875	Unmarried : 1716	Amer-Indian-Eskimo : 153	Germany : 75	Canada : 73	
State-gov : 657	Assoc-voc : 668	Sales : 1846	Wife : 785	Other : 152	Puerto-Rico : 59	El-Salvador : 54	
Self-emp-inc : 568	11th : 592	Other-service : 1644	Other-relative : 486		India : 49	Cuba : 46	
Federal-gov : 499	Assoc-acdm : 538	Machine-op-inspct : 994			Japan : 35	Guatemala : 34	
Without-pay : 6	10th : 450	? : 944			Columbia : 33	Dominican-Republic : 33	
Never-worked : 4	7th-8th : 331	Transport-moving : 785			Haiti : 24	Taiwan : 24	
	Prof-school : 275	Handlers-cleaners : 704			Portugal : 21	Nicaragua : 19	
	9th : 257	Farming-fishing : 476			France : 19	Iran : 17	
	Doctorate : 212	Tech-support : 467			Ecuador : 17	Peru : 13	
	12th : 206	Protective-serv : 335			Hong : 12	Greece : 12	
	5th-6th : 161	Priv-house-serv : 79			Ireland : 11	Laos : 10	
	1st-4th : 88	Armed-Forces : 5			Outlying-US(Guam-USVI-etc) : 8	Thailand : 8	
	Preschool : 33				Trinidad&Tobago : 8	Cambodia : 8	
					Hungary : 6	Honduras : 6	
					Yugoslavia : 6	Scotland : 4	
Bottom 20:	Bottom 20:	Bottom 20:	Bottom 20:	Bottom 20:	Bottom 20:	Bottom 20:	Bottom 20:

Figure 4.41. "Statistics View" → Tab "Top/Bottom" with the number of occurrences of nominal values calculated only on the selected nominal columns and sorted in descending order.

Linear Regression Learner

The "Linear Regression Learner" node performs a multivariate linear regression on a target column, i.e., the response. The "Linear Regression (Learner)" node can be found in the "Node Repository" in category "Analytics" → "Mining" → "Regression".

In the configuration window you need to specify:

- The target column for which the regression is calculated
- The columns to be used as independent variables in the linear regression
- The number of the starting row and the number of rows to be visualized in the node's scatter plot view

- The missing value handling strategy
- A default offset value to use (if any)

Selection of the input data columns is performed by means of the column selection framework: by manual selection with “Include/Exclude” panels; by type selection, by Wildcard/Regex expression selection.

The node outputs the regression model as well as the model’s coefficients and statistics.

Note. The “Linear Regression Learner” node can only deal with numerical values. Nominal columns are automatically discretized using a dummy coding available for categorical variables in regression http://en.wikipedia.org/wiki/Categorical_variable#Categorical_variables_in_regression.

We connected a “Linear Regression Learner” node to the “CSV Reader” with the training data. We wanted to predict the columns “hours-per-week” by using the columns “age”, “education-

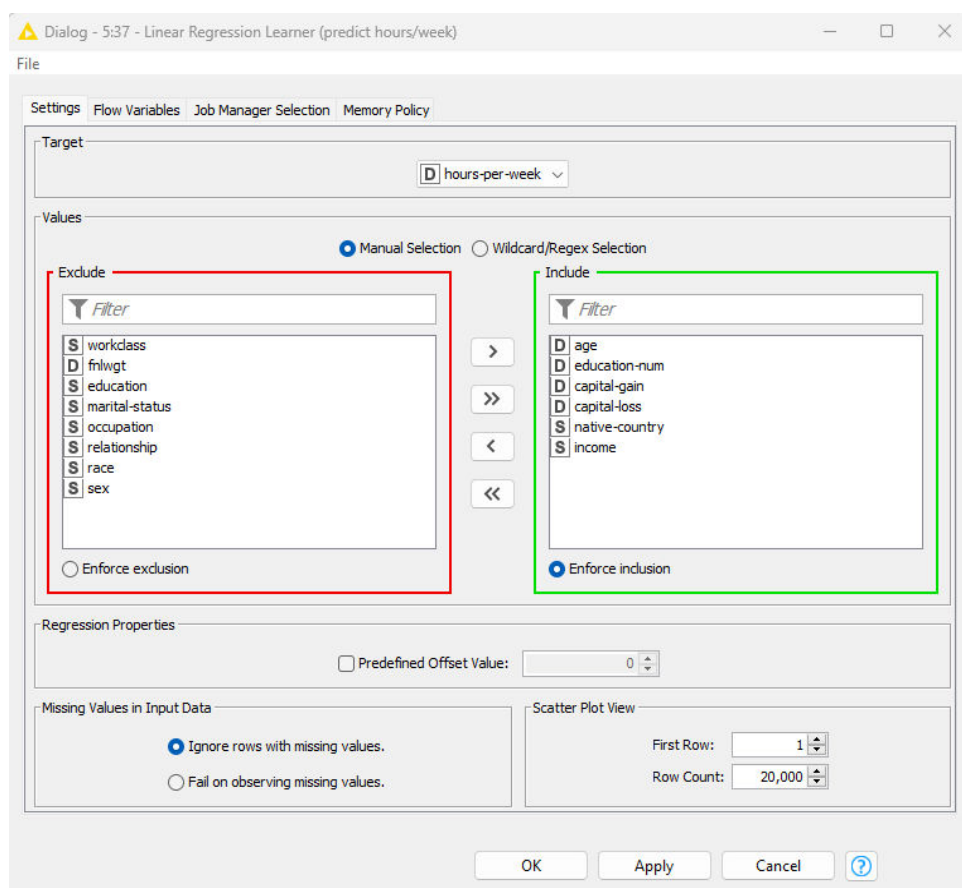


Figure 4.42. Configuration window for the Linear Regression Learner node.

num”, “capital-gain”, “capital-loss”, “native-country”, and “income” as independent variables for the linear regression. The “Linear Regression Learner” node produces the regression model at the node’s output port. The regression model is subsequently fed into a “Regression Predictor” node and used to predict new values for a different data set.

Regression Predictor

The “Regression Predictor” node obtains a regression model from one of its input ports (blue square) and data from the other input port (black triangle). It uses the model and the data to make a data-based prediction.

Since all information is already available in the model, this node only needs the minimal predictor settings: an alternative customized name for the output classification column.

The “Regression Predictor” node is located in the “Analytics” → “Mining” → “Regression” category in the “Node Repository” panel.

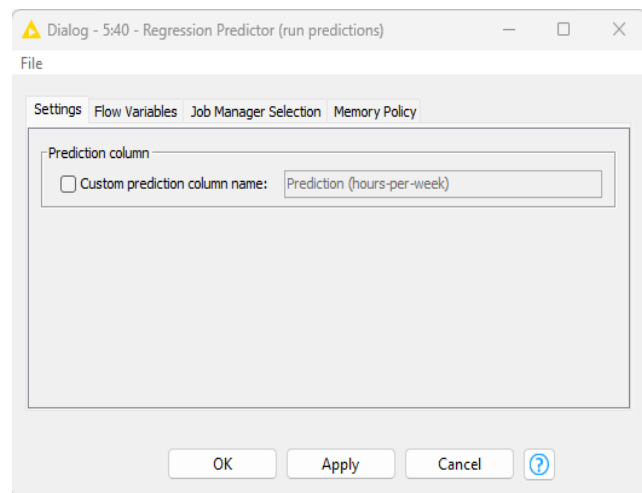


Figure 4.43. Configuration window for the Regression Predictor node.

Clustering

The last topic that we want to discuss in this chapter is clustering. There are many clustering techniques around and KNIME has implemented a number of them.

As in data models we already looked at, we have a trainer node and a predictor node for the clustering models. The learner nodes implement a clustering algorithm; that is they build a number of clusters by grouping together similar patterns and calculate their representative prototypes. The predictor then assigns a new data vector to the cluster with the nearest prototype. Such a predictor is not specific to only one clustering technique, but it works for any clustering algorithm that requires a cluster assignment on the basis of a distance function in the prediction phase. This leads to many specific clustering learner nodes (implementing different clustering procedures) but to only one clustering predictor node.

A learner node could implement the k-Means algorithm, for example. The k-Means procedure builds k clusters on the training data, where k is a predefined number^{2,3,4}. The algorithm iterates multiple times over the data and terminates when the cluster assignments no longer change. Note that the k clusters are only built on the basis of a similarity (distance) criterion. k-Means does not take into account the real class of each data row: it is an unsupervised classification algorithm. The predictor performs a crisp classification that assigns a data vector to only one of the k clusters which were built on the training data; in particular it assigns the data vector to the cluster with the nearest prototype.

We will focus on the k-Means algorithm to give you an example of how clustering can be implemented with KNIME (see the “Clustering and Regression” workflow).

k-Means Clustering

The “k-Means” node groups input patterns into k clusters on the basis of a distance criterion and calculates their prototypes. The prototypes are built as the mean value of the cluster patterns. This node takes the training data on the input port and presents the model at the blue squared output port and the training data with cluster assignment on the data output port (black triangle). The “k-Means” node can be found in the “Node Repository” in the “Analytics” → “Mining” → “Clustering” category.

In the configuration window you need to specify:

- The final *number of clusters* k
- The *maximum number of iterations* to ensure that the learning operation converges within a reasonable time
- The *columns* to be used to calculate the distance and the prototypes
- Flag “Always include all columns” is alternative to the column selection frame.

Column selection is performed by means of an “Exclude”/“Include” frame.

- The columns to be used for the distance calculation are listed in frame “Include”. All other columns are listed in frame “Exclude”.
- To move from frame “Include” to frame “Exclude” and vice versa, use buttons “add” and “remove”. To move all columns to one frame or the other use buttons “add all” and “remove all”.

The “k-Means” node has a “Cluster View” option in the context-menu: “View: Cluster View”. The Cluster View shows the prototypes of the k clusters.

Note. Since clustering algorithms are based on distance, a normalization is usually required to make all feature ranges comparable. In the “Clustering and Regression” workflow, we normalized the input features all into [0,1] by using a “Normalizer” node.

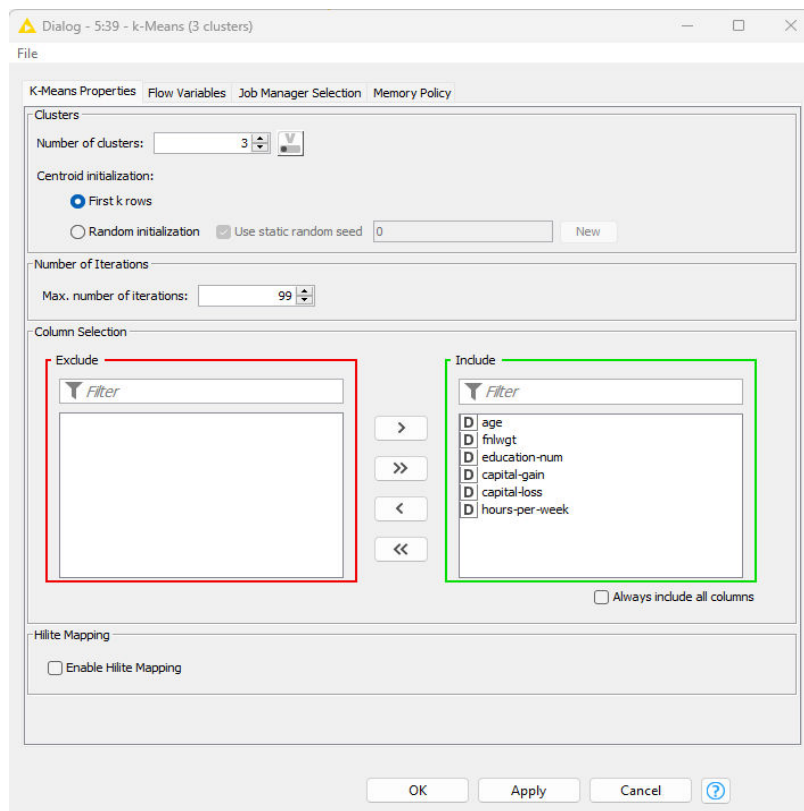


Figure 4.44. Configuration window of the k-Means node.

The k-Means algorithm just defines the clusters in the input space on the basis of a representative subset of the same input space. Once the set of clusters is defined, new data rows need to be scored against it to find the cluster they belong to. To do that, we use the “Cluster Assigner” node.

Cluster Assigner

The “Cluster Assigner” node assigns test data to an existing set of prototypes that have been calculated by a clustering node such as the “k-Means” node. Each data row is assigned to its nearest prototype.

The node takes a clustering model and a data set as inputs and produces a copy of the data set with an additional column containing the cluster assignments.

The “Cluster Assigner” node is located in the “Analytics” → “Mining” → “Clustering” category in the “Node Repository” panel.

It does not need any configuration settings specific to its cluster assignment task.

Workflow: Clustering and Regression

This workflow shows a few more data analysis algorithms:

- a linear regression to predict the number of hours/week given all other attribute values
- a k-Means clustering to group together the most similar data rows.

Remember that k-Means, like Neural Networks, needs normalized data.

The Statistics node mainly produces general statistical measures about one data column, including a roughly drawn histogram.

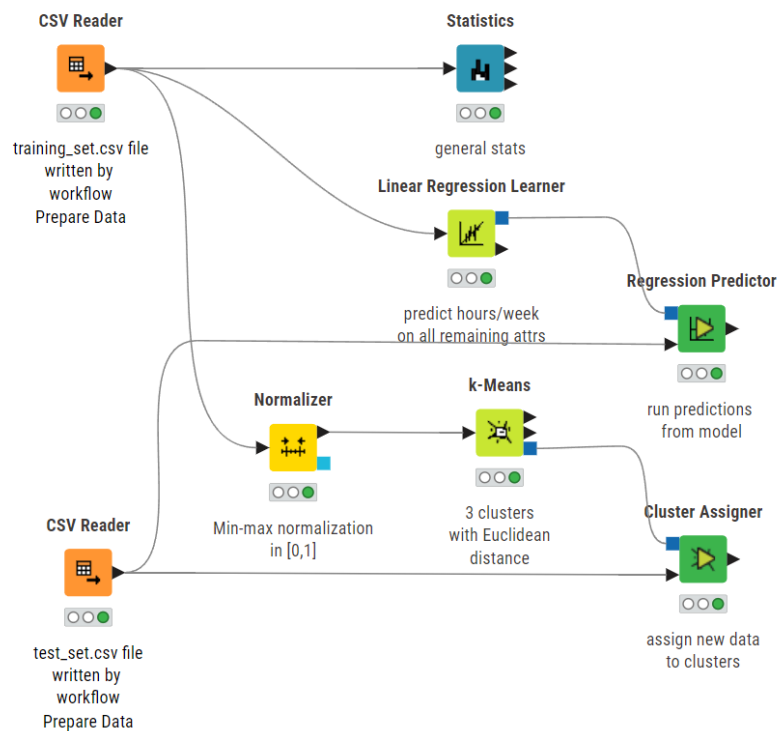


Figure 4.45. Workflow “Clustering and Regression”.

Note. The “Cluster Assigner” node is not specific for the “k-Means” node. It performs the cluster assignment task from a cluster set based with any of the available clustering algorithms.

Hypothesis Testing

A few nodes are available in KNIME to perform classical statistical hypothesis testing. Most of them can be found in “Analytics” → “Statistics” → “Hypothesis Testing”: the single sample t-test, the paired t-test, the one way ANOVA, and the independent group t-test. Only the node performing the chi-square test is located outside of the “Hypothesis Testing” sub-category into the “Crosstab” node.

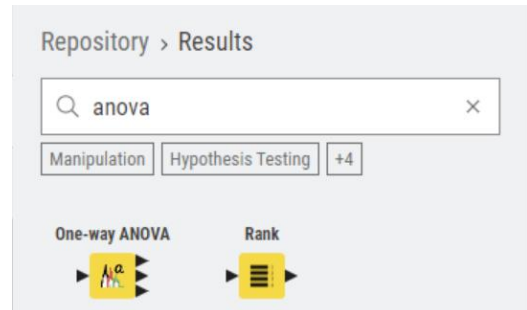


Figure 4.46. The “Hypothesis Testing” sub-

Additional newer nodes for statistical hypothesis testing are available under “KNIME Labs” → “Statistics” in the “Node Repository” panel.

4.5. Exercises

Exercise 1

Using the *wine.data* file (training set = 80% and test set = 20%), train a decision tree to recognize the class to which each wine belongs. Run the decision tree on the wine test set and measure the decision tree performance. In particular, we are interested in finding out how many false negatives for class 2 there are.

Solution to Exercise 1

In the “Decision Tree Learner” node we used column “class” as the class column. By default, the “CSV Reader” node reads the wine data class as Integer, since the classes are “1”, “2”, and “3”.

If you use a decision tree, as we did, for the final classification, you need the column “Class” to be of nominal values, i.e., to be of String type. You have two options for that:

- You read “Class” as String. In the “CSV Reader” configuration window, right-click column “Class” and change type from “Integer” to “String”

Chapter 4: My First Model

- You leave the default settings in the “CSV Reader” and then you use a “Number To String” node for the conversion

Workflow: Chapter 4/Exercise 1

This workflow:

- reads the wine data from the KBLdata folder
- partitions the data: 80% to training set, 20% to test set
- trains a decision tree to predict the wine class based on all other attributes (i.e. data columns)
- applies the model to the test set
- scores the right guess of Prediction(Class) vs. the original Class values.

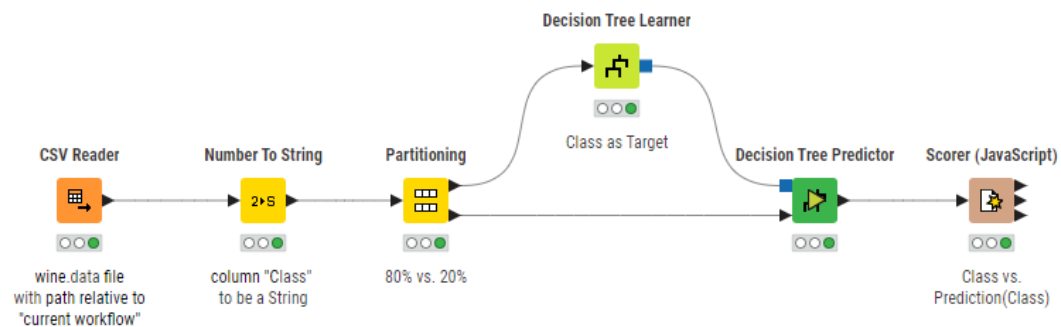


Figure 4.47. Exercise 1: Workflow.

We then used a “Scorer (JavaScript)” node to see how many False Negatives were produced in the accuracy statistics and/or in the confusion matrix.

The screenshot shows the 'Scorer View' window with the following data:

		1 (Predicted)	2 (Predicted)	3 (Predicted)	
1 (Actual)		11	1	0	91.67%
2 (Actual)		1	12	1	85.71%
3 (Actual)		0	0	10	100.00%
		91.67%	92.31%	90.91%	

Overall Statistics				
Overall Accuracy	Overall Error	Cohen's Kappa (κ)	Correctly Classified	Incorrectly Classified
91.67%	8.33%	0.874	33	3

Figure 4.48. Exercise 1: Confusion Matrix window of the Scorer (JavaScript) node.

We open the view in the context menu and look for the number of records belonging to class 2 (Y-axis) that are misclassified as class 3 (X-axis).

There is only one record that has been misclassified to class 3 from class 2.

Note. "Decision Tree Learner" node needs at least one nominal value to be used as classification column.

Exercise 2

Build a training set (80%) and a test set (20%) from the wine.data. Train a Multilayer Perceptron (MLP) on the training set to classify the data according to the values in column "Class". Next, apply the MLP to the test set and measure the model performance.

Solution to Exercise 2

We use a "Normalizer" node to scale the data before feeding them into the MLP. Since the wine dataset is very small, we used the whole data set to define the normalization parameters.

The next step involved using a neural network with only one output neuron to model the three class values: "1", "2", and "3".

Workflow: Chapter 4/Exercise 2

This is an example of how to change the classification thresholds. In the previous exercise some patterns were not correctly classified. By applying a different set of thresholds to the output probabilities, the final wine classification can be improved.
The key node of this exercise is the Rule Engine at the end, defining a set of thresholds for a more exact classification.

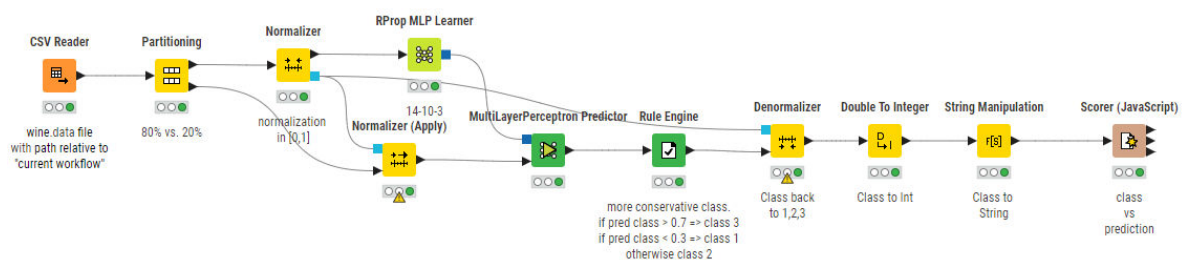


Figure 4.49. Exercise 2: Workflow.

As a neuron has a continuous output value, its output has to be post-processed to assign a class in the form of "1", "2", and "3" to each data row. To do this we used a "Rule Engine" node that implements the following rule:

```
IF          $neuron output$ <= 0.3                => class 1
ELSE IF    $neuron output$ >0.3 AND $neuron output$ <0.6  => class 2
ELSE IF    $neuron output$ >= 0.6                => class 3
```

The model performance is measured with a “Scorer (JavaScript)” node. Alternatively, you can explore the “Numerical Scorer” node to measure performances with numerical distances.

Exercise 3

Read the data *web site 1.txt* with a CSV Reader node. This data set describes the number of visitors to a web site for the year 2013. Compute a few statistical parameters on the number of visitors, such as the mean and the standard deviation. Train a Naïve Bayes model on the number of visitors to discover whether a specific data row refers to a weekend or to a business day. Finally, draw the ROC curve to visualize the Naïve Bayesian Classifier performance.

Solution to Exercise 3

We used a “Rule Engine” node to translate the column called “Day of Week” into a “weekend/not weekend” binary class. We filtered out the “Day of Week” column so as not to make the classification task too easy for the Naïve Bayesian Classifier. We trained the Bayesian Network on the binary class “weekend/not weekend”, and we built the ROC curve on the “weekend” class probability.

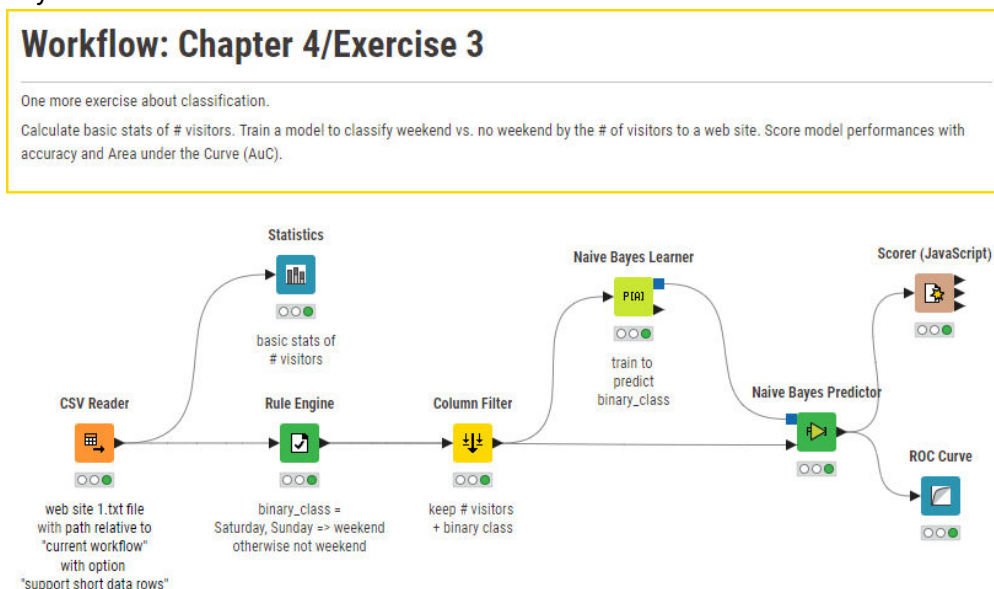


Figure 4.50. Exercise 3: Workflow.

Chapter 4: My First Model

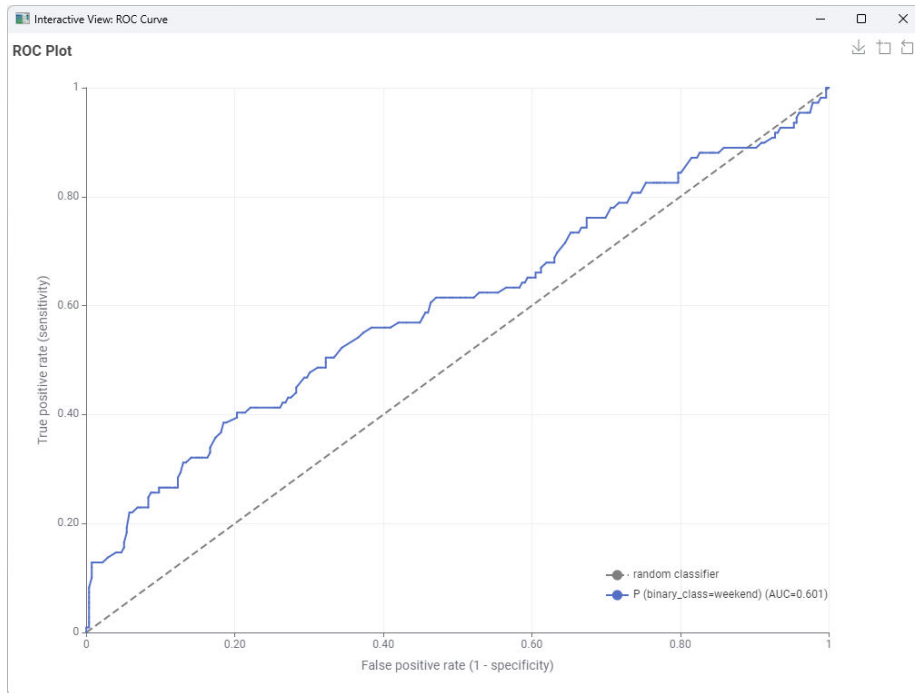


Figure 4.51. Exercise 3: ROC Curve on the results of the Bayesian Classifier.

Chapter 5: Preparing Data for Reporting

5.1. Introduction

One piece of a Data Science project is reporting. For example, it can be used to show the model scores to the management board or to quantify performances for your boss. In this case, it is convenient to save the intermediate data into some history files, in order to be able to easily replicate the reports or to proceed with further data analysis later on.

While KNIME Analytics Platform has some reporting capabilities - via integration with other reporting tools (BIRT, Tableau, Spotfire, QlikView, and more) or via the data app deployed via the KNIME Business Hub – we will focus here

on some summarization features to prepare the data for reporting or for storage in some intermediate Data Warehousing tables or files. A number of KNIME nodes are available to help us in this data manipulation task.

Before continuing, let's create a new workflow group "Chapter5" and open a new workflow with the name "Projects".

5.2. Transform Rows

Usually, data needs to reach the report in a predefined form. In this section we explore a few KNIME nodes that can help us to reach the desired data set structure.

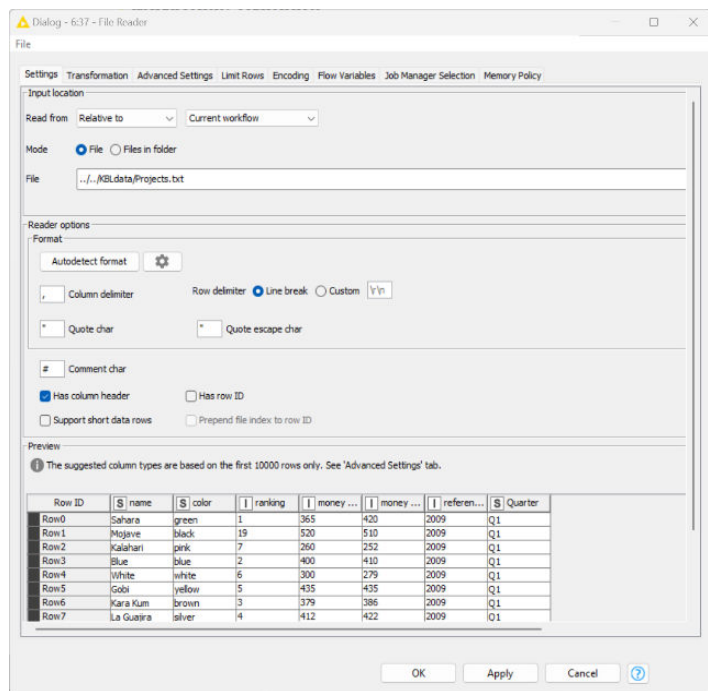


Figure 5.1. Data Structure of the Projects.txt file.

Chapter 5: Preparing Data for Reporting

The data for the report comes from the file "Projects.txt", which contains a list of projects and details how much money has been assigned to or used by each project during the years 2007, 2008, and 2009. In the report, we want to show 3 tables, with a structure as the one shown in Table 5.1., and 2 charts, from a data table as the one reported in Table 5.2.

The first table should show the project names in the row headers, the years in the column headers, and how much money has been assigned in total to each project for each year in the table cells.

The second table has the same structure, but it shows how much money has been used in total by each project for each year in the table cells.

The third table has the same structure as the two tables described above and shows the remaining amount of money (= money assigned - money used) for each project for each year.

The first chart should show the total amount of money assigned to each project (y-axis) over the three years (x-axis). The chart is fed with a data set where the values for x-axis and the values for y-axis are listed in two different columns; that is a data set where the year and the corresponding sum of money belong to the same row.

The second chart has the same structure as the first chart but shows the total amount of money used instead of the total amount of money assigned. That is, the chart must show the total amount of money used by each project (y-axis) over the three years (x-axis). For this reason, it needs a data set with year and total money used by each project separated into different columns.

Project Name/ Year	2007	2008	2009
Project 1	Sum (money) for project 1 in year 2007	Sum (money) for project 1 in year 2008	Sum (money) for project 1 in year 2009
Project 2	Sum (money) for project 2 in year 2008
Project 3

Table 5.1. Data structure required for the tables in the report (Pivoting node).

Project Name	Year	Sum (money)
Project 1	2009	Sum (money) for project 1 in year 2009
Project 2	2009	Sum (money) for project 2 in year 2009
Project 3	2009	Sum (money) for project 3 in year 2009
...

Table 5.2. Data structure required for the charts in the report (GroupBy node).

Chapter 5: Preparing Data for Reporting

For both table and chart structures, we need to calculate the sum of money (assigned, used, or remaining), but we need to report it on a different data layout. For example, in one case we want the years to be the column headers and in the other case we want the years to be the column values.

The first data table (Table 5.1) could be obtained with a “Pivoting” node, while the second data table (Table 5.2) with a “GroupBy” node. We then introduced a “GroupBy” node and two “Pivoting” nodes in the “Projects” workflow.

In the “GroupBy” node, we calculated the sum (= aggregation method) of the values in the column “money assigned (1000)” and in the column “money used (1000)” (= multiple aggregation columns) for each group of rows defined by the combination of distinct values in the columns “reference year” and “name” (= Group Columns).

In the resulting data table, the first two columns contained all combinations of distinct values in the “name” and “reference year” columns. The aggregations were then run over the groups of data rows defined by each (“name”, “reference year”) pair. The aggregated values are displayed in two new columns “Sum(money assigned (1000))” and “Sum(money used (1000))”.

We named the new “GroupBy” node “money by project by year”.

Group table - 6:3 - GroupBy (money)

File Edit Hilitte Navigation View

Table "default" - Rows: 33 Spec - Columns: 4 Properties Flow Variables

Row ID	name	referen...	Sum(money assigned (1000))	Sum(money used (1000))
Row0	Blue	2007	1360	1300
Row1	Blue	2008	1277	1124
Row2	Blue	2009	1565	1650
Row3	Gobi	2007	1203	1220
Row4	Gobi	2008	1424	1308
Row5	Gobi	2009	1740	1740
Row6	Kalahari	2007	630	876
Row7	Kalahari	2008	800	768
Row8	Kalahari	2009	1192	1178
Row9	Kara Kum	2007	800	800
Row10	Kara Kum	2008	888	992
Row11	Kara Kum	2009	1516	1544
Row12	La Guajira	2007	1020	1200
Row13	La Guajira	2008	1404	1648
Row14	La Guajira	2009	1496	1518
Row15	Mojave	2007	1800	2000
Row16	Mojave	2008	1819	1820
Row17	Mojave	2009	1860	1809
Row18	Patagonia	2007	864	1332
Row19	Patagonia	2008	2098	2139
Row20	Patagonia	2009	1359	1364

Figure 5.2. Output data table of the GroupBy node.

In one “Pivoting” node we calculated the sum (= aggregation method) of the values in column “money assigned(1000)” (= aggregation column) for each combination of values in the “reference year” (= pivot column) and “name” (= group column) columns.

In the other “Pivoting” node we calculated again the sum (= aggregation method) of the values in column “money used(1000)” (= aggregation column) for each combination of values in the “reference year” (= pivot column) and “name” (= group column) columns.

In both “Pivoting” nodes, we chose to keep the original names in the “Column naming” box.

The aggregated values are then displayed in a pivot table with <year + aggregation variable> as column headers, the project names in the first column, and the sum of “money assigned(1000)” or “money used(1000)” for each project and for each year as the cell content. We named the new Pivoting nodes “money assigned to project each year” and “money used by project each year”.

In order to make the pivot table easier to read, we moved the values of the project name column to become the RowIDs of the data table and we renamed the pivot column headers with only the reference year value. In order to do that, we used a “RowID” node and a “Column Rename” node respectively.

#	RowID	name <small>String</small>	2007+Sum(money used (1000)) <small>Number (integer)</small>	2008+Sum(money used (1000)) <small>Number (integer)</small>	2009+Sum(money used (1000)) <small>Number (integer)</small>
1	Row0	Blue	1300	1124	1650
2	Row1	Gobi	1220	1308	1740
3	Row2	Kalahari	876	768	1178
4	Row3	Kara Kum	800	992	1544
5	Row4	La Guajira	1200	1648	1518
6	Row5	Mojave	2000	1820	1809
7	Row6	Patagonia	1332	2139	1364
8	Row7	Sahara	905	1460	1670
9	Row8	Sechura	3600	3113	4000
10	Row9	Tanami	591	0	468
11	Row10	White	860	948	1347

Figure 5.3. Output pivot table of the Pivoting node.

RowID

The RowID node can be found in the “Node Repository” panel in the “Manipulation” → “Row” → “Other” category. The RowID node allows the user to:

- Replace the current RowIDs with the values of another column (top half of the configuration window)

- Copy the current RowIDs into a new column (bottom half of the configuration window)

When replacing the current RowIDs a few additional options are supported.

- “Remove selected column” removes the column that has been used to replace the RowIDs.
- “Ensure uniqueness” adds an extension “(1)” to duplicate RowIDs. Extension becomes “(2)” or “(3)” etc... depending on how many duplicate values are encountered for this RowID.

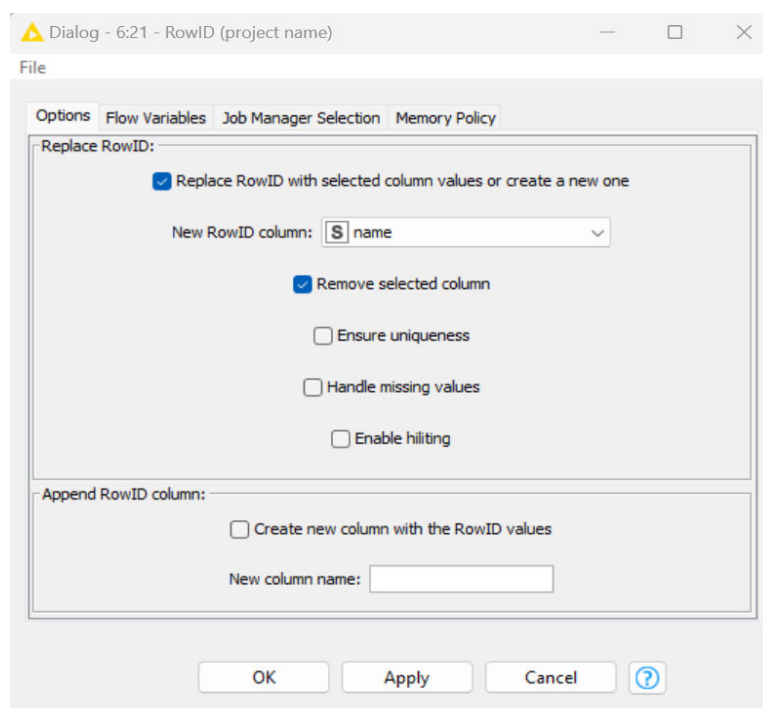


Figure 5.4. Configuration window for the RowID node.

- “Handle missing values” replaces missing values in RowIDs with default values.
- “Enable hiliting” keeps a map between the old and the new RowIDs to keep hiliting working in other nodes.
- In the “Node Repository” panel, close to the “Pivoting” node you can find the “Unpivoting” node. Although we are not going to use the “Unpivoting” node in our example workflow, it is worth it having a look at it.

Unpivoting

The *Unpivoting* node rotates the content of the input data table. This kind of rotation is shown in Table 5.3 and Table 5.4. Basically, it produces a “GroupBy”-style output data table from the “pivoted” input data table. The “Unpivoting” node is located in the “Manipulation” → “Row” → “Transform” category.

In the settings you need to define:

- Which columns should be used for the cells redistribution
- Which columns should be retained from the original data set

The unpivoting process produces 3 new columns in the data table:

- One column called “RowIDs”, which contains the RowIDs of the input data table
- One column called “ColumnNames”, which contains the column headers of the input data table
- One column called “ColumnValues”, which reconnects the original cell values to their RowID and column header

The column selection follows the already seen an “Exclude”/“Include” frame:

- The still available columns for grouping are listed in the frame “Available column(s)”. The selected columns are listed in the frame “Group column(s)”.
- To move from frame “Available column(s)” to frame “Group column(s)” and vice versa, use the “add” and “remove” buttons. To move all columns to one frame or the other use the “add all” and “remove all” buttons.
- “Excludes and Includes” keeps the included/excluded columns as fixed and adds possible new columns to the other column set.

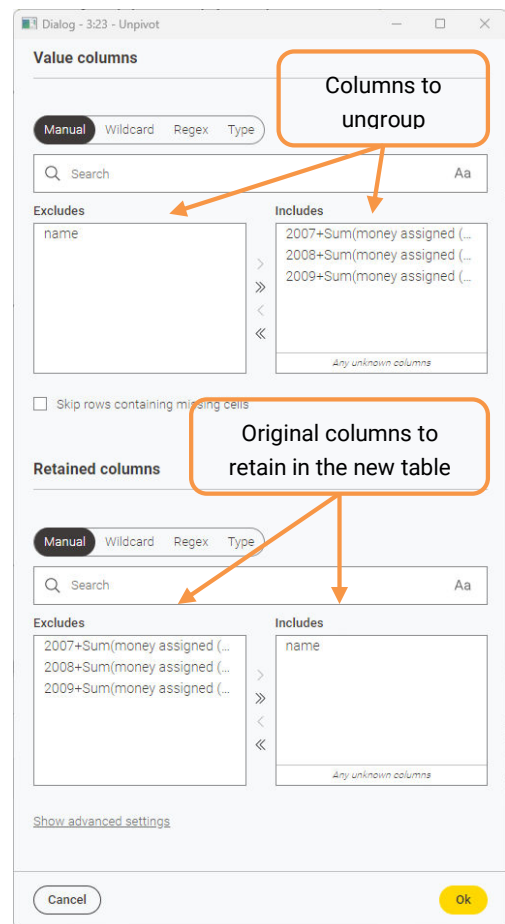


Figure 5.5. Configuration window for the Unpivoting node.

Input Table

	Col1	Col2	Col3
ID1	1	3	5
ID2	2	4	6

Table 5.3. Input Data Table.

Unpivoted Data Table

	RowIDs	Column Names	Column Values
Row1	ID1	Col1	1
Row2	ID1	Col2	3
Row3	ID1	Col3	5
Row4	ID2	Col1	2
Row5	ID2	Col2	4
Row6	ID2	Col3	6

Table 5.4. Unpivoted Data Table.

Note. Pivoting + Unpivoting = GroupBy

The output data tables of the “GroupBy” node and of the “Pivoting” node are sorted by the group columns’ values.

The “Sorter” node, like the “Pivoting” and the “GroupBy” node, is another node that is frequently used for reporting. For demonstrative purposes we briefly show here the “Sorter” node.

Sorter

The “Sorter” node sorts the rows of a data table by sorting the values of one of its columns. In the settings you need to select:

- The column(s) to be sorted (the RowIDs column is also accepted)
- Whether to sort in ascending or descending order

It is possible to sort the table by multiple columns. To add a new sorting column:

- Click the “add sorting criterion” button
- Select the new column

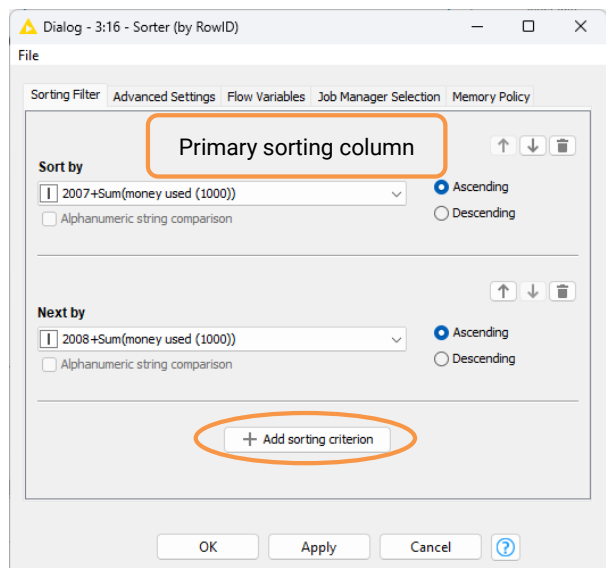


Figure 5.6. Configuration window of the Sorter node.

The first column (in the top part of the configuration window) gives the primary sorting; the second column gives the secondary sorting, and so on.

5.3. Joining Columns

After applying the two “Sorter” nodes, we now have two data tables with the same column structure:

- RowID containing the project names
- “2009” column with the used/assigned money for year 2009
- “2008” column with the used/assigned money for year 2008
- “2007” column with the used/assigned money for year 2007

Now, it would be useful to

- have all values for used and assigned money over the years together for each project in the same row, for example:

RowID	assigned 2009	assigned 2008	assigned 2007	used 2006	used 2005	used 2008
<project name>

- Calculate the remaining money for each year for each project, as: `remain <year> = assigned <year> -used<year>`

Basically, we want to join the two data tables, the table with the values for the assigned money and the table with the values for the used money, into one single table. After that, we want to calculate the remaining money values.

First of all, in order to be able to perform the table join without confusion, we need different columns to bear different names. We will see that actions need to be taken in case of a join of tables with columns with the same name. Let’s connect a “Column Renamer” node to each “RowID” node.

In the table resulting from the node “money used by project each year”, let’s rename the column called “2009 + money used(1000)” as just “used 2009”, the column called “2008 + money used(1000)” to “used 2008”, and the column called “2007 + money used(1000)” to “used 2007”.

The data tables we want to join now have the structure reported below.

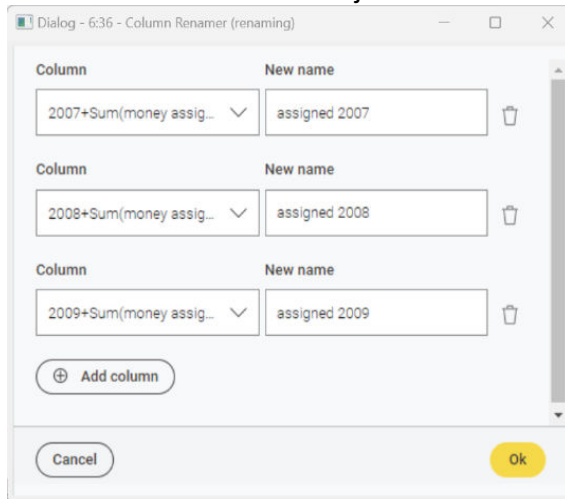


Figure 5.7. Money assigned to each project each year.

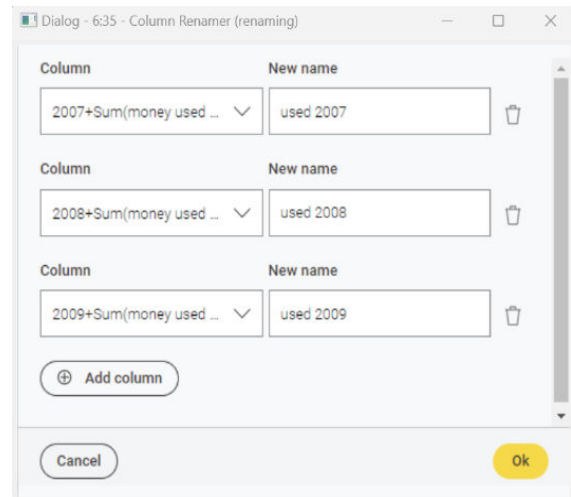


Figure 5.8. Money used by each project each year.

Now that we have the right data structure, we need to perform a table join. We want to join the cells to be in the same row based on the project name, i.e. in this case this is the RowID. In fact, we want the row of used money for project “Blue” to be appended at the end of the corresponding row of the table with the assigned money. KNIME has a very powerful node that can be used to join tables, known as the “Joiner” node.

Joiner

The “Joiner” node is located in the “Node Repository” panel in “Manipulation” → “Column” √ “Split & Combine”. The “Joiner” node takes two data tables on the input ports and matches a column of the table on the upper port (left table) with a column of the table on the bottom port (right table). These columns can also be the RowID columns. This node has three output ports: one for the matched rows, one for the unmatched rows from the top (left) table (if any), and one for the unmatched rows from the lower (right) table (if any).

There are two tabs to be filled in, in the “Joiner” node’s configuration window.

- The “Joiner Settings” tab contains all the settings pertaining to:
 - The join mode

- The columns to be matched
- Other secondary parameters
- The “Column Selection” tab contains all settings pertaining to:
 - Which columns from the two tables to be included in the joined table
 - How to handle duplicate columns (i.e., columns with the same name)
 - Whether to filter out the joining column from the left and/or from the right data table or none at all

Joiner Node: The “Joiner Settings” Tab

The “Joiner Settings” tab sets the basic joining properties, like the “join mode”, the “joining columns”, the “matching criterion”, and so on.

The first setting is about the columns with values to match from the top (left) table and from the lower (right) table. Joining on multiple columns is supported. To add a new pair of joining columns:

- Click the “+”button;

Values in key columns can be matched by value and type, just as Strings (types can differ), or just as numbers.

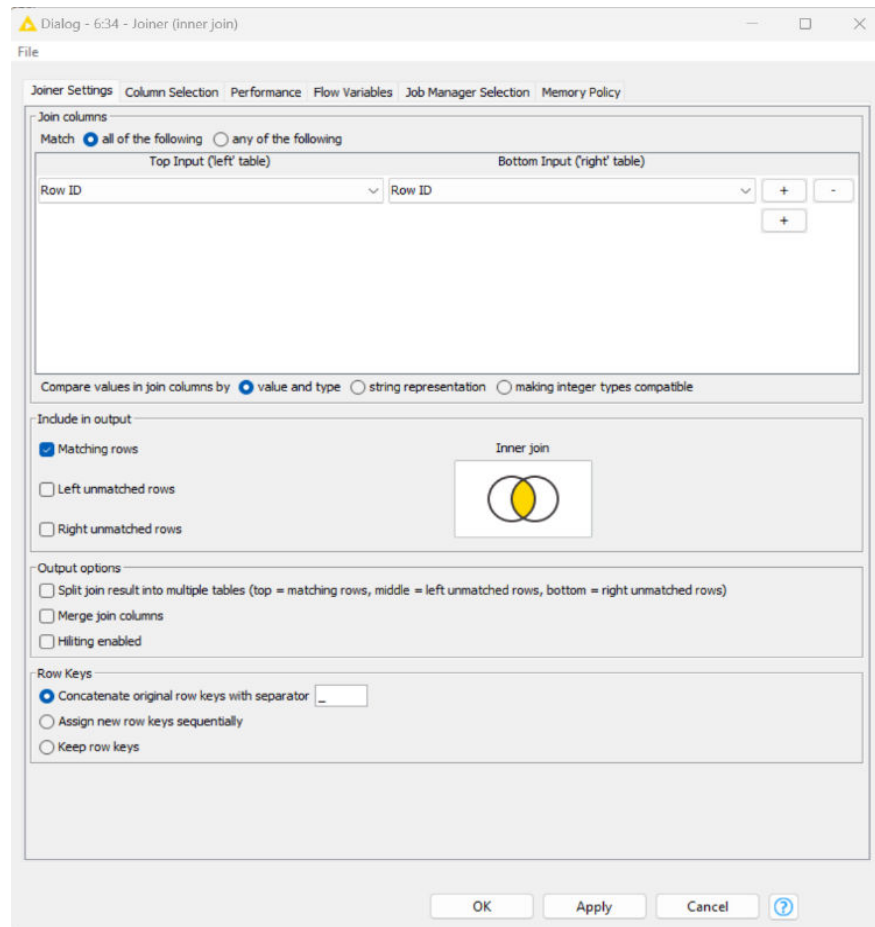


Figure 5.9. Configuration window for the Joiner node which includes two tabs to be filled: "Joiner Settings" and "Column Selection".

The second setting is the join mode.

- Matching rows is the equivalent of an Inner join, that is it keeps only those rows where the values in the two joining columns match;
- Left unmatched rows in addition keeps all rows from the left (top) table, even if unmatched. Enabling the first and this checkbox is equivalent to a left join.
- Right unmatched rows in addition keeps all rows from the right (bottom) table, even if unmatched. Enabling the first and this checkbox is equivalent to a right join.
- Enabling all three checkboxes is equivalent to a full outer join.

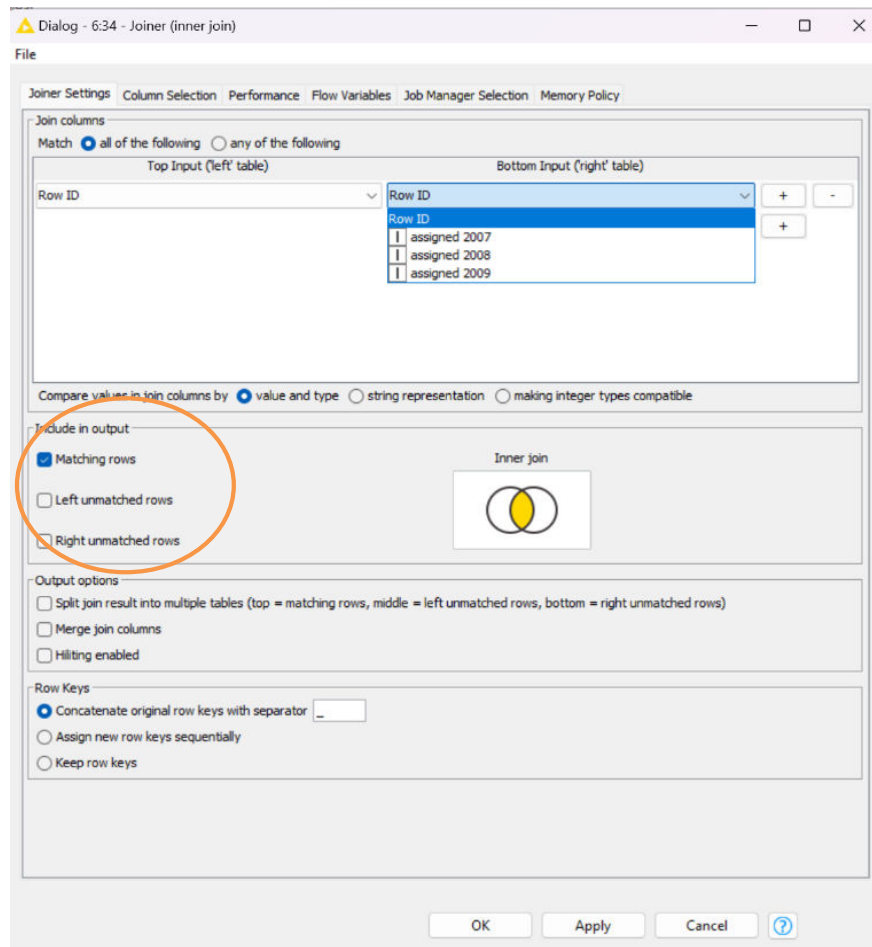


Figure 5.10. Configuration window for the Joiner node: the "Joiner Settings" tab.

Other settings:

- Output options sets the table(s) to export at the output ports.
- Row Keys sets the format for the new row keys.

Joiner Node: The "Column Selection" Tab

Tab "Column Selection" defines how to handle the columns that are not involved in the match process. Once that two key values match, the other columns from the left and the right table could be retained or removed. A classic "Exclude"/"Include" frame sets the columns to keep or remove from both input tables.

- The columns to be kept in the new joined table are listed in frame "Include". All other columns are listed in frame "Exclude".

- To move from frame “Include” to frame “Exclude” and vice versa, use buttons “add” and “remove”. To move all columns to one frame or the other use buttons “add all” and “remove all”.

The “**Duplicate column names**” panel offers a few options to deal with the problem of columns with the same header (= duplicate columns) in the two tables.

- “**Do not execute**” produces an error
- “**Append suffix**” appends a suffix, default or customized, to the name of the duplicate columns in the right table

We joined the two tables (money assigned and money used) using the RowIDs as the joining column for both; we chose to append a suffix “(right)” for the columns from the right table with the same name as the columns from the left table; and we chose the inner join as join mode.

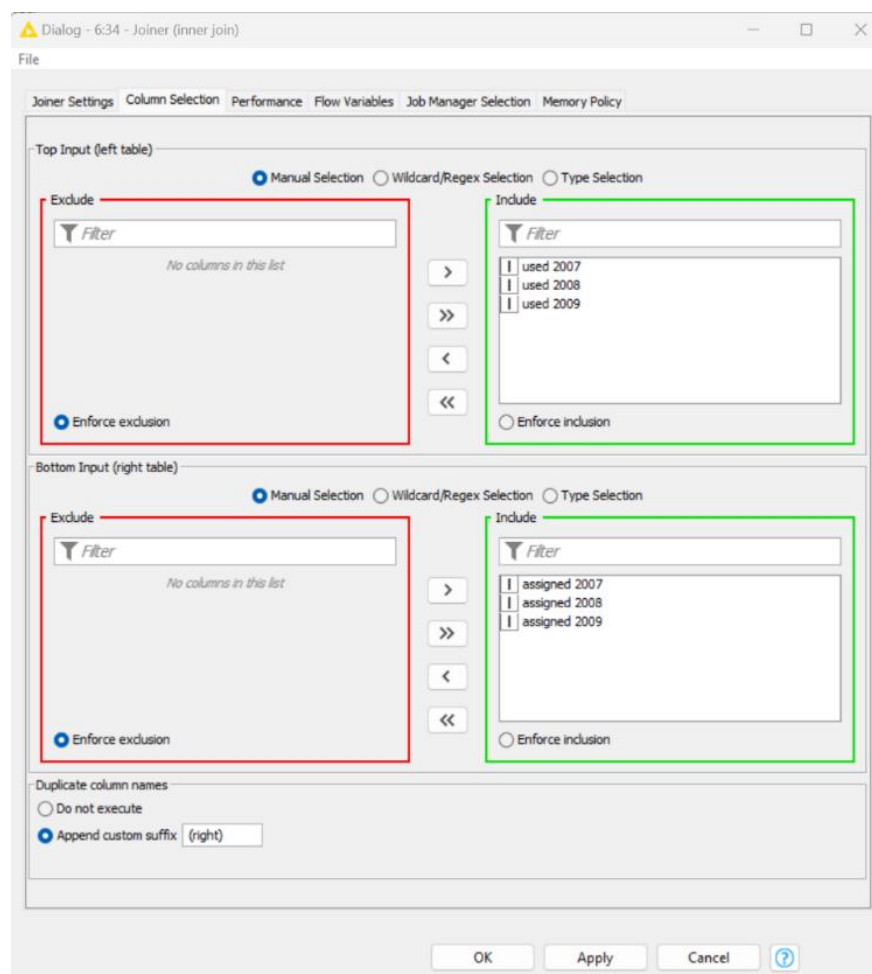


Figure 5.11. Configuration window for the Joiner node: the “Column Selection” tab.

Chapter 5: Preparing Data for Reporting

The resulting data is shown in figure 5.12. You can see that now the “assigned money” values and the “used money” values are on the same row for each project.

It is of course possible to make the join on different columns than the RowIDs columns. However, the joining on RowID allows the user to keep the original RowID values which might be important for some subsequent data analysis or data manipulation. In this case we need the RowIDs to contain the joining keys. To manipulate the RowID values, KNIME has a “RowID” node.

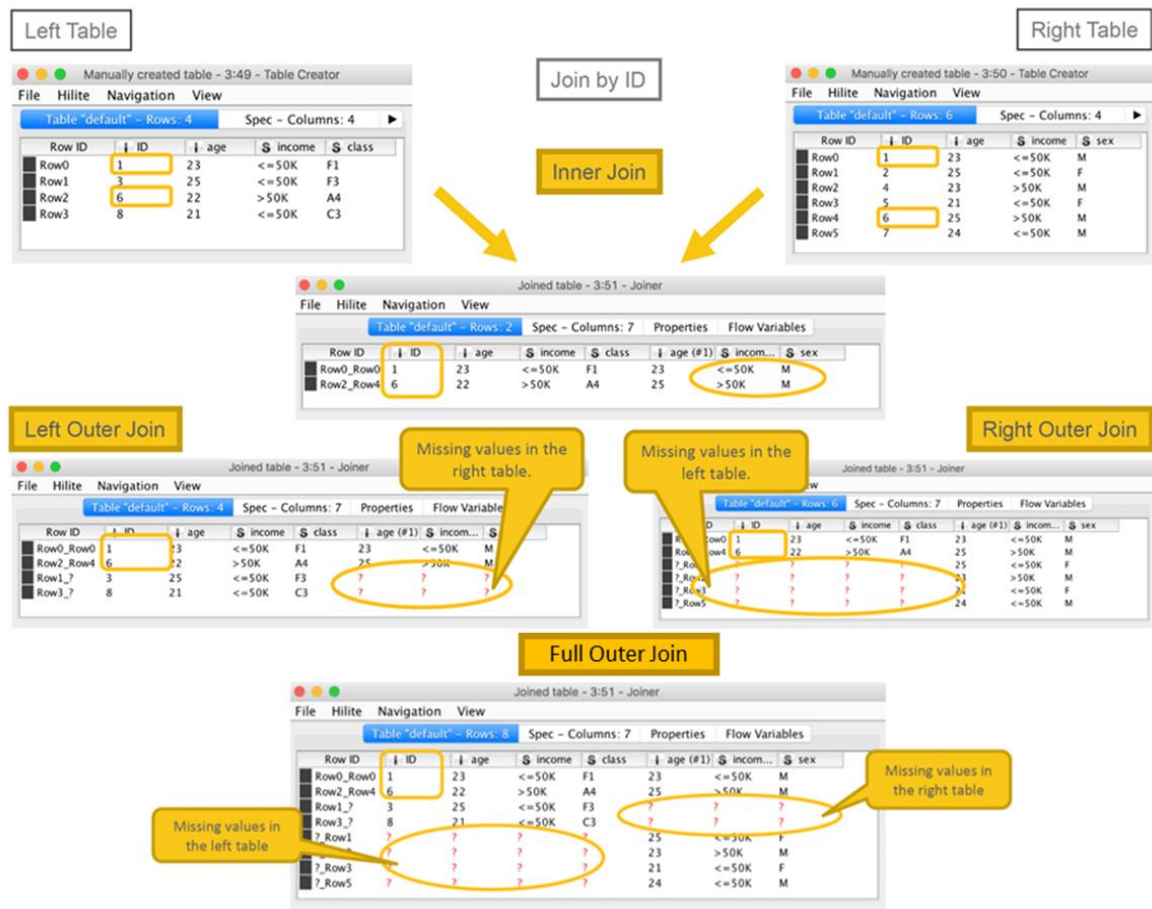


Figure 5.12. Join modes with the “Filter duplicates” option enabled.

Join result - 6:34 - Joiner (inner join)

File Edit Hilite Navigation View

Table "default" - Rows: 11 Spec - Columns: 6 Properties Flow Variables

Row ID	used 2007	used 2008	used 2009	assigned 2007	assigned 2008	assigned 2009
Blue_Blue	1300	1124	1650	1360	1277	1565
Gobi_Gobi	1220	1308	1740	1203	1424	1740
Kalahari_Kalahari	876	768	1178	630	800	1192
Kara Kum_Kara Kum	800	992	1544	800	888	1516
La Guajira_La Guajira	1200	1648	1518	1020	1404	1496
Mojave_Mojave	2000	1820	1809	1800	1819	1860
Patagonia_Patagonia	1332	2139	1364	864	2098	1359
Sahara_Sahara	905	1460	1670	806	1457	1495
Sechura_Sechura	3600	3113	4000	3200	2966	3940
Tanami_Tanami	591	0	468	453	0	453
White_White	860	948	1347	860	1087	1420

Figure 5.13. Output data table of the Joiner node with "Inner Join" as join mode.

5.4. Misc. Nodes

In our report we want to include the remaining money for each year, calculated as: <remaining value> = <assigned value> - <used value>. There are two ways to calculate this value: the "Math Formula" node and the "Java Snippet" nodes. All of these nodes are located in the "Misc" category.

The "Java Snippet" nodes allow the user to execute pieces of Java code. We can then use a "Java Snippet" node to calculate the amount <remaining value>. Actually, we will use three "Java Snippet" nodes: one to calculate the <remaining value 2009>, a second one to calculate the <remaining value 2008>, and a third one to calculate the <remaining value 2007>. We name the three "Java Snippet" nodes "remain 2009", "remain 2008", and "remain 2007".

There are two types of "Java Snippet" nodes: the "Java Snippet" node and the "Java Snippet (simple)" node. The functionality is the same: run a piece of Java code. However, the "Java Snippet" node has a more complex and more flexible GUI, while the "Java Snippet (simple)" node offers a more simplified GUI. That is, the "Java Snippet" node is for more expert users and more complex pieces of code, while the "Java Snippet (simple)" node is for medium expert users and more simple pieces of Java code.

Java Snippet (simple)

A "Java Snippet (simple)" node allows the execution of a piece of Java code and places the result value into a new or existing data column. The code has to end with the keyword "return"

followed by the name of a variable or expression. The “Java Snippet (simple)” node is in the “Node Repository” panel in the “Misc” → “Java Snippet” category.

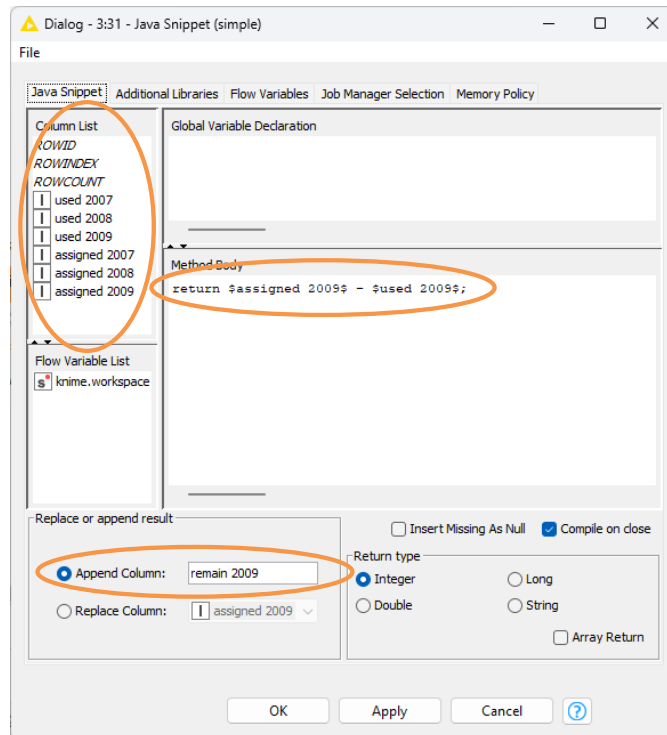


Figure 5.14. Configuration window for the Java Snippet node.

When opening the node’s dialog for configuration, a window appears including a number of panels:

- The **Java editor** is the central part of the configuration window. This is where you write your code. Please remember that the code has to return some value and therefore it has to finish with the keyword “return” followed by a variable name or expression. Multiple “return” statements are not permitted in the code.
- The **list of column names** is on the top left-hand side. The column names can be used as variables inside the Java code. After double-clicking the column name, the corresponding variable appears in the Java editor. Variables carry the type of their original column into the java code, i.e.: Double, Integer, String and Arrays. Array variables come from columns of the type Collection Type.
- The **name and type of the column** to append or to replace. The column type can be “Integer”, “Double”, or “String”. If the column type does not match the type of the variable that is being returned, the Java snippet code will not compile.

- It is also possible to **return arrays** instead of single variables (see the checkbox on the bottom right). In this case a number of columns (as many as the array length) will be appended to the output data table.
- In the “**Global Variable Declaration**” panel global variables can be created, to be used recursively in the code across the table data rows.
- The “**Additional Libraries**” tab allows the inclusion of non-standard Java Libraries.
- In the “**Global Variable Declaration**” panel global variables can be created, to be used recursively in the code across the table data rows.

For node “remain 2009” we used the Java code: `return $assigned 2009$ - $used 2009$`

The same code could be used with the other two Java snippet nodes to calculate “remain 2008” and “remain 2009”. The same task could have been accomplished with a “Java Snippet” node.

Java Snippet

Like the “Java Snippet (simple)” node, the “Java Snippet” node allows the execution of a piece of Java code and places the result value into a new or existing data column. The node is located in the “Node Repository” panel in the “Misc” → “Java Snippet” category.

The configuration window of the “Java Snippet” node also contains:

- The **Java editor**. This is the central part of the configuration window and it is the main difference with the “Java Snippet (simple)” node. The editor has sections reserved for: variable declaration, imports, code, and cleaning operations at the end.
 - The “**expression_start**” section contains the code.
 - The “**system variables**” section contains the global variables, those whose value has to be carried on row by row. Variables declared inside the “expression_start” section will reset their value at each row processing.
 - The “**system imports**” section is for the library import declaration.

Self-completion is also enabled, allowing for an easier search of methods and variables. One or more output variables can be exported in one or more new or existing output data columns.

- The **table named "Input"**. This table contains all the variables derived from the input data columns. Use the "Add" and "Remove" buttons to add input data columns to the list of variables to be used in the Java code.
- The **list of column names** on the top left-hand side. The column names can be used as variables inside the Java code. After double-clicking the column name, the corresponding variable appears in the Java editor and in the list of input variables on the bottom. Variables carry the type of their original column into the java code, i.e.: Double, Integer, String and Arrays. Their type though can be changed (where possible) by changing the field "Java Type" in the table named "Input".
- The **table named "Output"**. This table contains the output data columns to be created as new or to be replaced by the new values. To add a new output data column, click the "Add" button. Use the "Add" and "Remove" button to add and remove output data columns. Enable the flag "Replace" if the data column is to override an existing column. The data column type can be "Integer", "Double", or "String". If the column type does not match the type of the variable that is being returned, the Java snippet code will not compile. It is also possible to **return arrays** instead of single variables, just by enabling the flag "Array". Remember to assign a value to the output variables in the Java code zone.

Both "Java Snippet" nodes are very powerful nodes, since they allow the user to deploy the power of Java inside KNIME. However, most of the times, such powerful nodes are not necessary. All mathematical operations, for example, can be performed by the "Math Formula" node. The "Math Formula" node is optimized for mathematical operations and therefore tends to be faster than the "Java Snippet" nodes.

Math Formula

The "Math Formula" node enables mathematical formulas to be implemented and works similarly to the "String Manipulation" node (see section 3.5).

The "Math Formula" node is not part of the basic standalone KNIME. It has to be downloaded with the extension package (see par. 1.5) "*KNIME Math Expression Extension (JEP)*". Once the extension package has been installed, the Math Formula node is located in the "Node Repository" panel in the "Misc" category.

In the configuration window there is:

- The list of column names from the input data table

- The list of variables (to see in the “KNIME Advanced Luck”)
- A list of mathematical functions, e.g. $\log(x)$.
- The expression editor

Double-clicking data columns from the list on the left automatically inserts them in the expression editor. You can complete the math expression by typing in what’s missing. Here, like in the “Java Snippet” nodes, $\$<column_name>\$$ indicates the usage of a data column.

A number of functions to build a mathematical formula are available in the central list. At the

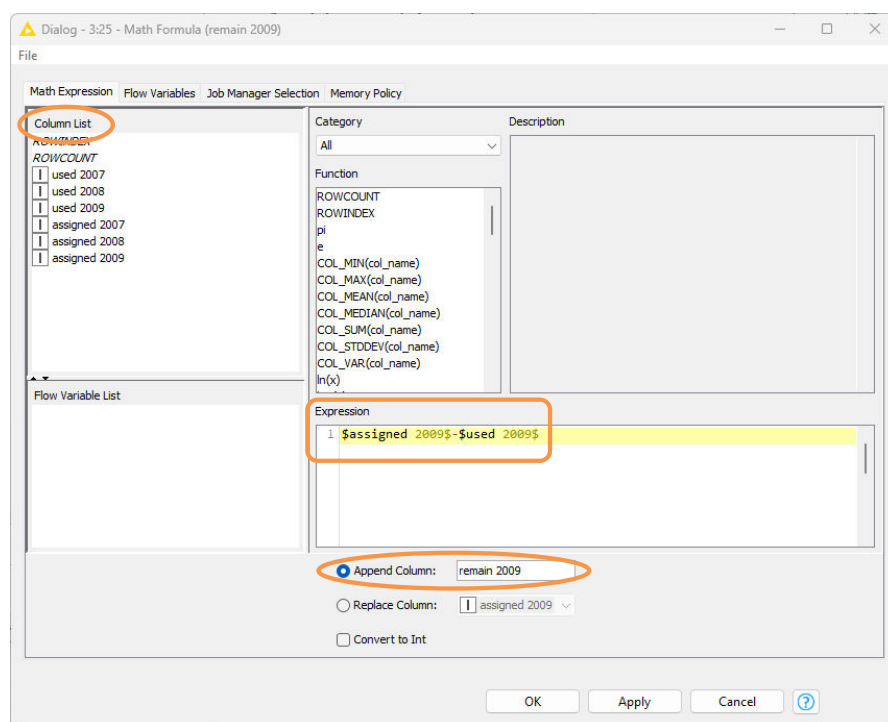


Figure 5.15. Configuration window of the Math Formula node.

bottom you can insert the name of the column to be appended or replaced. The node exports double type data, but integer type data can also be exported by enabling the “Convert to Int” option.

In the configuration window of the Math Formula node introduced in the “Projects” workflow, we implemented the same calculation of values $\langle \text{remain 2009} \rangle$ mentioned in the “Java Snippet” node earlier in this chapter. We simply needed to:

- Double-click the two columns $\$used\ 2009\$$ and $\$assigned\ 2009\$$ in the column list
- Type a character “-” between the two data column names in the expression editor.

In the “Projects” workflow we decided to use this implementation of the remaining values with the “Math Formula” nodes. That is, we used 3 “Math Formula” nodes, one after the other, to calculate the remaining values for 2009, the remaining values for 2008, and the remaining values for 2007 respectively. We also kept the implementation with the Java Snippet nodes for demonstration purposes. However, only the sequence of “Math Formula” nodes has been connected to the next node. We gave the “Math Formula” nodes the same names that we used for the Java Snippet nodes, to indicate that they are performing exactly the same task.

Another type of “Math Formula” node is the “Math Formula (Multi Column)” node.

Math Formula (Multi Column)

The “Math Formula (Multi Column)” node allows to implement the same mathematical formula on a list of columns, as specified in the configuration window.

In the upper part of the configuration window, we choose the list of columns on which to implement the formula, through an Exclude/Include frame. After that, we have the same configuration as for the simpler “Math Formula” node.

- The list of column names from the input data table
- The list of variables (to see in the “KNIME Cookbook”)
- A list of mathematical functions, e.g. $\log(x)$ and their descriptions
- The mathematics expression editor

Chapter 5: Preparing Data for Reporting

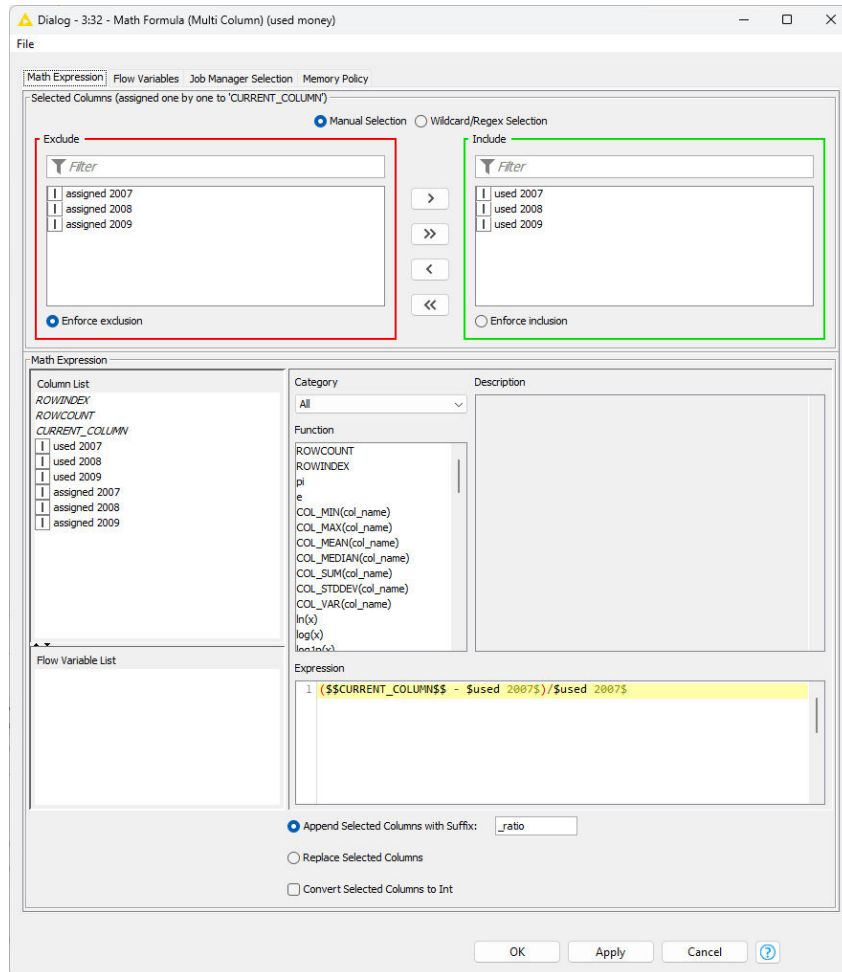


Figure 5.16. Configuration window of the Math Formula (Multi Column) node.

Double-clicking data columns from the list on the left automatically inserts them in the expression editor. You can complete the math expression by typing in what's missing.

The last three options include:

- A suffix to add to the original column names to form the output column names, if we want to create new columns;
- The option of overwriting the values in the original columns
- The flag to convert all results into Integer numbers

Let's finish the workflow with a "Constant Value Column" node to add an additional temporary column with the goal of this workflow "Preparing Data Workflow". The Constant Value Column was then connected to the last Math Formula node.

Workflow: Projects

In this workflow we prepare data to be displayed in a report.

Starting from the accounting of a list of projects, on one branch we calculate the money in budget for each year for each one of the project and on the other branch we calculate the money actually spent for each project each year.

We then report the money allocated, the money spent and the money remaining for each project each year.

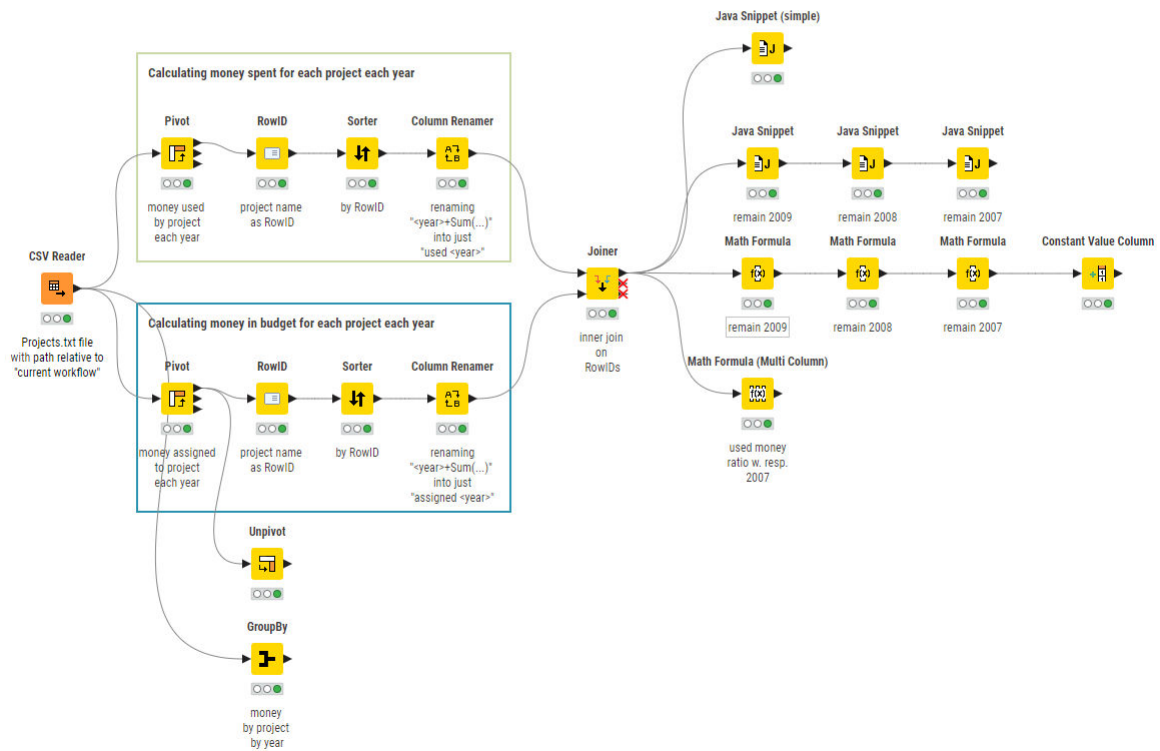


Figure 5.17. Workflow "Projects".

5.5. Cleaning Up the Final Workflow

The "Projects" workflow is now finished. However, we can see that it is very crowded with nodes, especially if we want to keep all the "Java Snippet" nodes and the "Math Formula" nodes. To make the workflow more readable, we can group all nodes that belong to the same task in a "Meta-node". For example, we can create meta-node "remaining money" that groups together all the nodes for the remaining values calculations.

Note. A metanode is a (gray) node containing other nodes.

There are two ways to build a metanode. The easiest way collapses pre-existing nodes into a metanode; the other way creates a metanode from scratch.

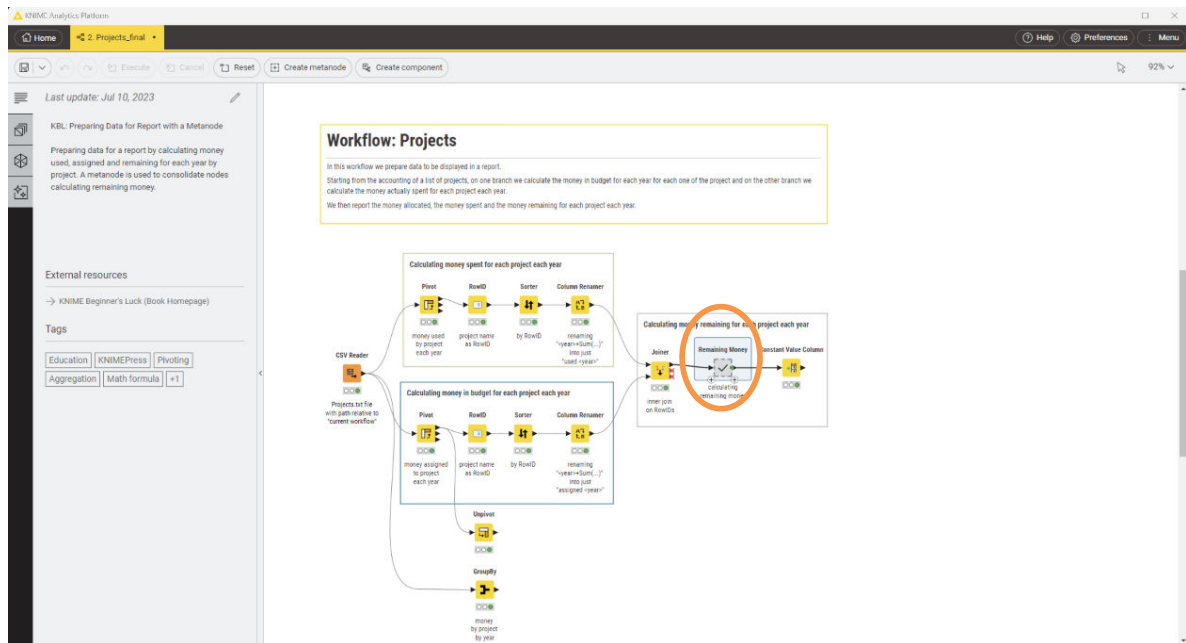


Figure 5.19. Metanode.

Collapse Pre-existing Nodes into a Metanode

- In the workflow editor, select the nodes that will be part of the meta-nodes (to select multiple nodes in Windows use the Shift and Ctrl keys)
- Right-click any of the selected node
- Select “Create Metanode...”
- A new meta-node is created with the sub-workflow of selected nodes
- The number of input and output ports is automatically defined based on the selected nodes.

Execute	F7
Reset	F8
Cut	Ctrl X
Copy	Ctrl C
Delete	Delete
Arrange annotations	>
Create metanode	Ctrl G
Create component	Ctrl J

Figure 5.18. “Create a Metanode...” option in the context menu of the selected nodes.

In workflow “Projects” we have selected all Java Snippet and Math Formula nodes to become part of a metanode named “Remaining Money” with one input port and one output port. The resulting workflow has been saved as “Projects_final”.

Create a Metanode from Scratch

A meta-node is a node that contains a sub-workflow of nodes. A meta-node does not perform a specific task; it is just a container of nodes.

Create a “Meta-node”:

- Select all the nodes you want to include it under the metanode
- In the Tool Bar click the “Meta-node” icon

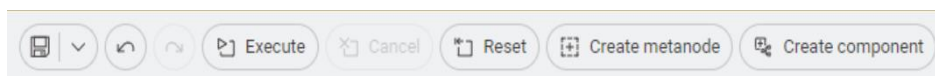


Figure 5.20. Metanode icon in the tool bar.

Open a “Meta-node”:

- Double-click the metanode, OR
- Right-click the metanode and select “Metanode” > “Open metanode”

A new editor window opens for you to edit the associated sub-workflow contained in the metanode.

Fill a metanode with nodes:

- Drag and drop nodes from the “Node Repository” panel as you would do with a normal workflow, OR
- Cut nodes that already exist in your workflow and paste them into the sub-workflow editor window

Note. The “Configure” option is enabled in a metanode but not really usable, as there is nothing to configure. All the other node commands, such as “Execute”, “Reset”, “Node name and description”, etc., are applied in the familiar manner, as for every other node. In particular, the “Execute” and “Reset” respectively run and reset all nodes inside the meta-node. You can add an input or output type port to the metanode as per your requirement by clicking on + symbol

Expand and Reconfigure a Metanode

Once the metanode exists, it is possible to interact with it (reconfigure, expand, etc.) through its context menu.

The context menu of a metanode is completely similar to the context menu of any other node, besides the “metanode” option. The metanode option opens a sub-menu with commands applicable only to a metanode, such as:

- “Open metanode”, to open the metanode content in the workflow editor
- “Expand metanode”, to reintroduce the meta-node content into the main workflow and get rid of the meta-node container
- “Rename metanode”, to change the name of the meta-node

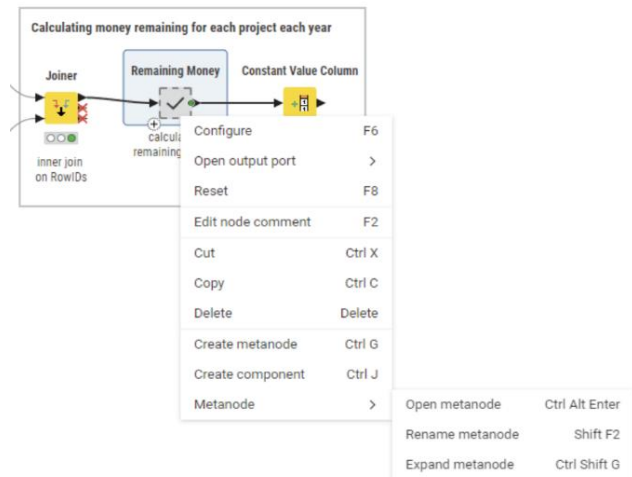


Figure 5.21. Metanode context menu.

You might have noticed the presence of items involving “Components” in the context menu. A component is a special kind of self-contained meta-node. Components are described in the sequel of this book, the “KNIME Advanced Luck”.

Note. The main difference between meta-nodes and components involves the usage of composite views in a data app on the KNIME Business Hub, as inherited from the contained nodes.

We created a new “Metanode” from all Math Formula and Java Snippet nodes and named it “Remaining Money”. We connected its input port to the output port of the “Joiner” node and its output port to the input port of the “Constant Value Column” node.

There are a number of pre-packaged meta-nodes in the KNIME “Node Repository” panel in some sub-categories “Meta Nodes” under some main categories, like “Workflow Control”, “Mining”, “R”, “Time Series”, and others. The “Meta Nodes” categories contain useful pre-packaged meta-node implementations for the parent category.

Workflow: Projects

In this workflow we prepare data to be displayed in a report.

Starting from the accounting of a list of projects, on one branch we calculate the money in budget for each year for each one of the project and on the other branch we calculate the money actually spent for each project each year.

We then report the money allocated, the money spent and the money remaining for each project each year.

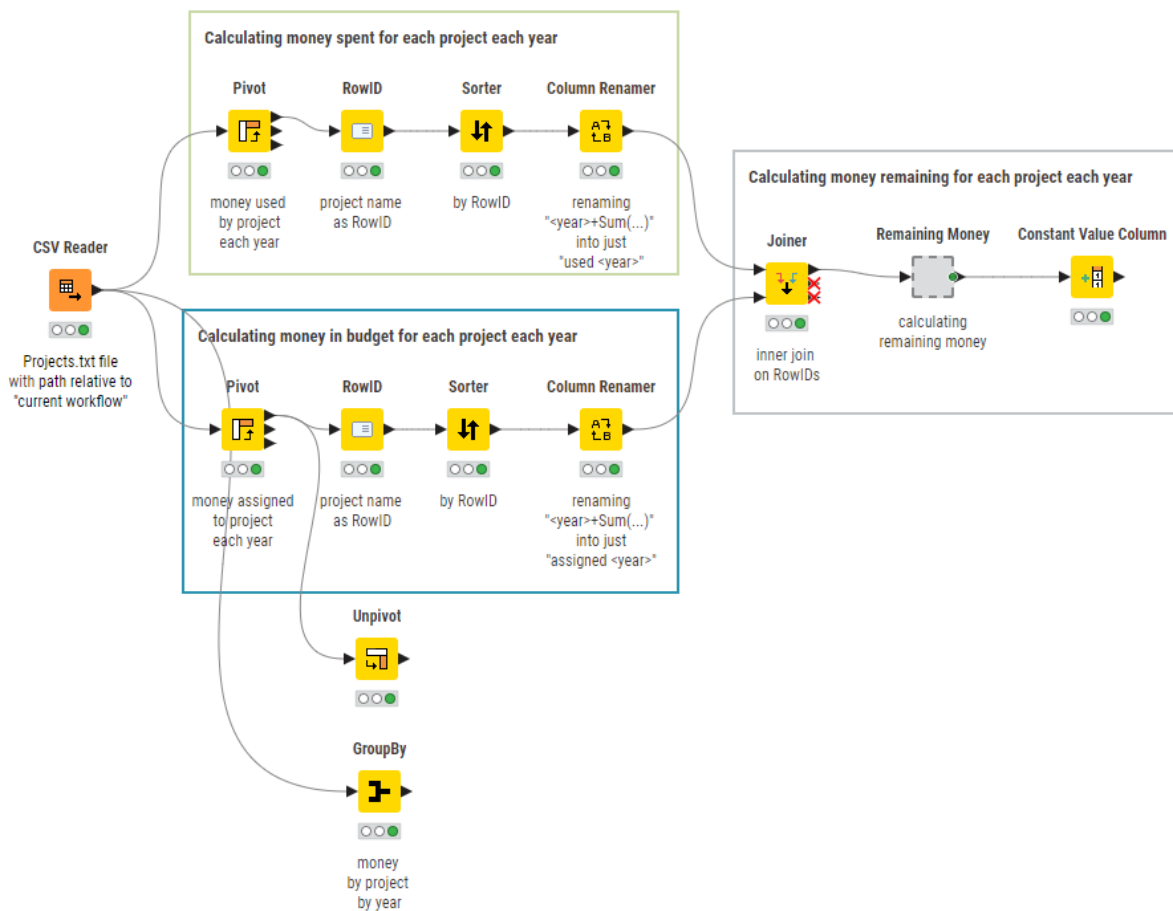


Figure 5.22. Workflow "Projects_final".

5.6. Next Step: Create a Report

At this point the data is ready. We need to build the report. There are many options to do that, via KNIME native nodes as well as via integrations with external reporting tools. Here, we list a few possible options, though more options are surely possible.

- Build dashboards via KNIME components and their composite views, to be visualized on a web browser when deployed on the KNIME Business Hub
- Export data into the BIRT reporting solution, opensource and integrated within KNIME Analytics Platform
- Export data into Tableau, PowerBI, or Spotfire reporting solutions via dedicated nodes. For such solutions a paid license is needed. Dedicated nodes exist to export the data into the solutions. While example workflows are provided under the Chapter7 folder, we will not describe the construction of such reports in detail in this book due to the requirement of a paid license.
- Export data into a CSV file, or another compatible file, and import into your preferred reporting tool.

5.7. Exercises

Exercise 1

Use the input data *adult.data* to do the following:

- Calculate the total number of people with income > 50K and the total number of people with income <= 50K for each work class
- Sort rows on the “work class” column in alphabetical order
- Create a data table structured as follows:

Work Class	Nr. of people with income >50K	Nr. of people with income < 50K
Work Class 1		
...		
Work Class n		

- Mark this dataset for reporting

Solution to Exercise 1

1. Read the data.
2. Use a “Pivoting” node to build the data table in the requested format. The “workclass” column should be the group column and the “Income” column should be the pivot column.
3. Optionally rename the column headers to make the table easier to read.

Workflow: Chapter 5/Exercise 1

This is a simple reporting exercise.

After generating a pivoting table of # of occurrences for each group (Income class, workclass), the table is exported to build a bar chart and to show the pivoting table.

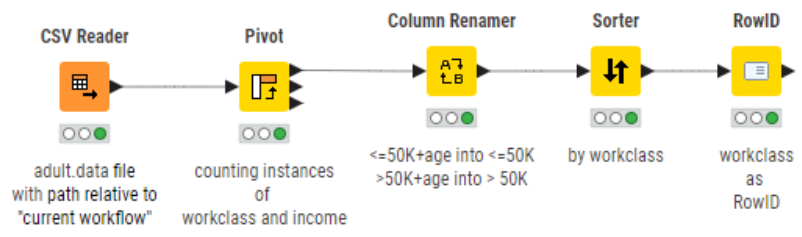


Figure 5.23. Exercise 1: The workflow.

Exercise 2

Extend Exercise 1:

- Calculate the total number of people with income > 50K and with income <= 50K
- Calculate the total number of people for each work class
- Calculate the total number of people
- Extend the data table produced for exercise 1 as follows:

Work Class	Nr. of people with income >50K	Nr. of people with income < 50K	Nr. of people
Work Class 1			

...			
Total	Sum(Nr. of people with income >50K)	Sum(Nr. of people with income <50K)	Sum(Nr. of people)

Solution to Exercise 2

1. To calculate the number of people for each “workclass” and each income class, we use the “Pivoting” node built in Exercise 1. The “Pivoting” node has three outputs: the pivot table, the totals by row, and the totals by column. Remember to enable “Append overall totals” in the “Pivots” tab.
2. We then inner join on the “workclass” values the pivot table with the totals by row using a “Joiner” node.
3. We then concatenate the data table resulting from the “Joiner” node with the totals by column of the “Pivoting” node.

Workflow: Chapter 5/Exercise 2

This is another simple reporting exercise.

After generating a pivoting table of # of occurrences for each group (Income class, workclass), the table is exported to build a simple line plot, an aggregated line plot, and to show the pivoting table.

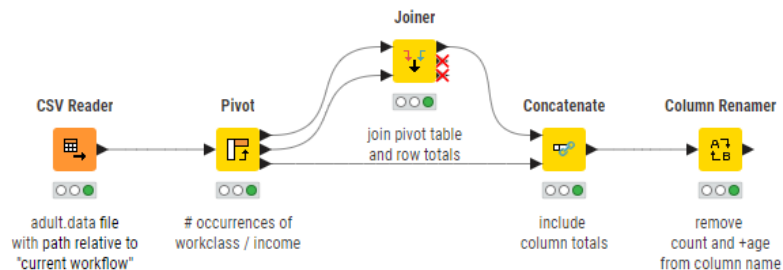


Figure 5.24. Exercise 2: The workflow.

Exercise 3

Read the csv file *SoccerWorldCup2006.txt* from the KBLdata folder. This file describes the results of soccer games during the soccer world cup 2006 (www.fifa.com). The second semifinal game for the third and the fourth placement is not reported.

For each team calculate:

- The total number of played games
- The total number of scored goals
- The total number of taken goals
- The average number of scored goal per game
- The average number of taken goal per game
- A fit measure as: $(\text{total number of scored goals} - \text{total number of taken goals}) / \text{number of played games}$

Document each step with the appropriate node's name and description. Make the workflow readable by using metanodes.

Solution to Exercise 3

In the “# scored goals” meta-node, first we sum team 1’s scores over all team 1, then the sum of team 2’s score over all team 2’s, and finally we sum the total scores of team 1 and team 2 when team 1 = team 2.

Meta-node “# taken goals” has the same structure as Meta-node “# scored goals”. The only difference lies in the aggregation variable of the first two “GroupBy nodes. In the meta-node “# scored goals” the first “GroupBy” node sums the “score of team 1” for all “team 1” values and

Workflow: Chapter 5/Exercise 3

This reporting exercise reads the results of 2006 World Cup soccer games and counts the total and average number of scored and taken goals during the whole tournament for each team.

The report then reports such numbers in a table, two bar charts (total and average numbers) and a scatter plot.

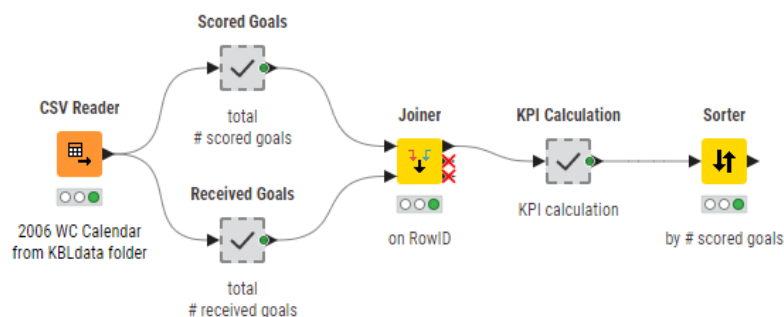


Figure 5.25. Exercise 3: The workflow.

the second “GroupBy” node sums the “score of team 2” for all “team 2” values. In meta-node “# taken goals” the first “GroupBy” node sums the “score of team 2” for all “team 1” values and the second “GroupBy” node sums the “score of team 1” for all “team 2” values.

In the “KPI calculation” meta-node we used 2 “Math Formula” nodes and one “Java Snippet” node. It could have been any other combination of “Java Snippet” and “Math Formula” nodes.

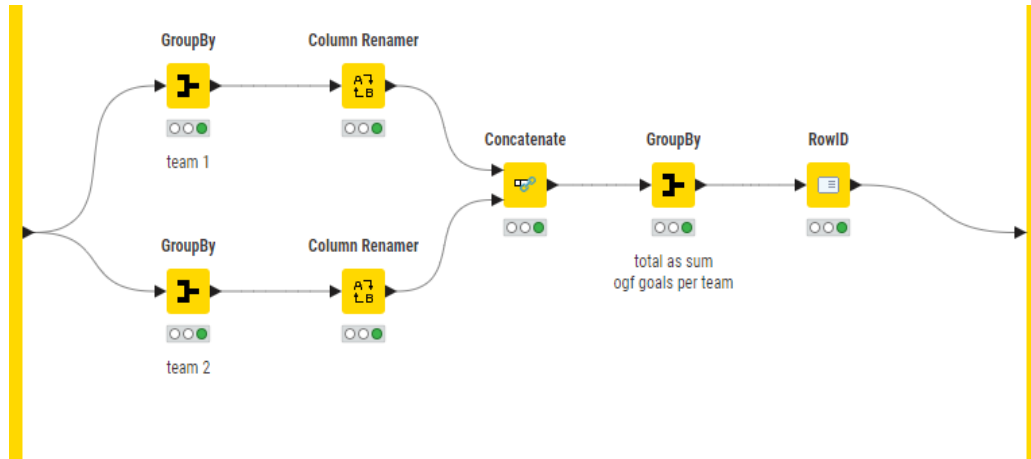


Figure 5.26. Metanode “Score Goals” and “Received Goals” respectively.

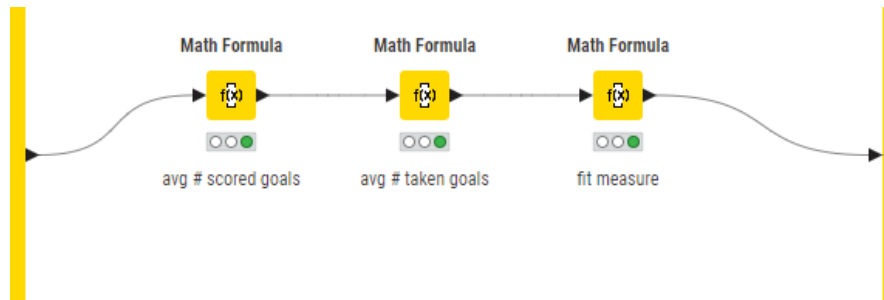


Figure 5.27. Metanode “KPI Calculation”.

Chapter 6: Dashboards with Composite View

6.1. The Dashboard

From the data tables produced in the previous chapter, we build here a simple dashboard, including a table for assigned money, a table for used money, and a table for remaining money across all projects across the three years of observation and the corresponding three bar charts.



Figure 6.1. The final dashboard visualizing the tables and bar charts for assigned, used, and remaining money across all 11 projects and the three years of observation.

The dashboard of course must also be interactive, that is it must offer as many customization options as possible to the end user while displaying it on a web browser. Customization options must include:

- Changing titles, subtitles, and other labels for each table and bar chart

- Selecting a project and visualizing all data pertaining to that project across all tables and charts
- Table pagination, if needed
- Zoom in/zoom out of single dashboard items

6.2. The Nodes

In chapter 3, dedicated to data visualization, we have seen already the Table View node and the Bar Chart node. As the name says, the Table View node produces a table-based visualization, and the Bar Chart node produces a bar chart visualization of the input data. As discussed above, all such visualization nodes have a few tabs in their configuration window: three the Table Viewer node and four the Bar Chart node.

Tabs in Table View's configuration window

- **Options.** This tab sets the input columns to display in the table together with some other minor settings, like the title and subtitle, and the display of colors, keys, and headers. We decided to include the project names in column "Project" as first column from the left in the table.
- **Interactivity.** This tab includes checkboxes for the interactivity options to be enabled, such as pagination options, selection options for the data rows, as well as searching and sorting options for tables with a very high number of rows. We have 11 projects, and we would like to be able to see all of them at once. So, here we set Initial Page Size 11.
- **Formatters.** This tab defines how to represent dates, numbers, and missing values in the table cells.

Tabs in Bar Chart's configuration window

- **Options.** Again, this tab sets the input columns to be involved in the visualization, as just an occurrence count, a sum, or an average. Since our numbers are unique and are supposed to be displayed as they are, we can choose the option Average or Sum. Category column is "Project" containing the project names. Optionally, the category names could be sorted alphabetically. However, our data rows are already sorted by project name in

alphabetical order. The checkbox “Generate image” enables the creation of the chart as an image at the output port. While useful for some tasks, this can be time and resource consuming.

- **General Plot Options.** This tab includes all plot settings: title, subtitle, axis labels, item display, and size of the optionally created image.
- **Control Options.** This tab includes checkboxes for the plot customization, such as editing of title, subtitle, and axis label, switching bar orientation and bar grouping vs. stacking, and other view settings.
- **Interactivity.** This tab completes the list of checkboxes for interactivity, covering the options for selecting bars in the chart.

The last node missing to complete this simple dashboard is a title. For that we use the “Text Output Widget” node.

The Output Widget

This node just outputs text in a graphical view.

- “Text” contains the text to be output.
- “Text format” contains the type of text and how to interpret it. Three types of text format are possible: simple text, preformatted text, and HTML text.

HTML text interprets and visualizes HTML instructions. So, the line:

```
<h1>Project Report: Money Flow</h1>
```

Is visualized as:

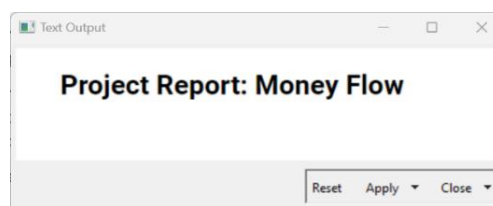


Figure 6.2. The output of the Text Output Widget node.

This node has an optional Flow Variable input port. Flow Variables though are not described in this book, and we do not need them for this particular example.

In conclusion, to build the dashboard displayed above, we need:

- Three Table View nodes
- Three Bar Chart nodes
- One Text Output Widget node

The workflow developed in the previous chapter, named Projects, has been imported in folder Chapter6 and renamed “Dashboard”. Then the seven required nodes have been created. Notice that we added a RowID node to copy the project names from the RowIDs into a data column named “Project”, and that we added a Column Resorter node to place the “Project” column at the very left of the input data table. Notice also that the Text Output Widget node does not need any input and therefore was left floating freely around.

Now, let’s assemble all these nodes together into a component to create the dashboard.

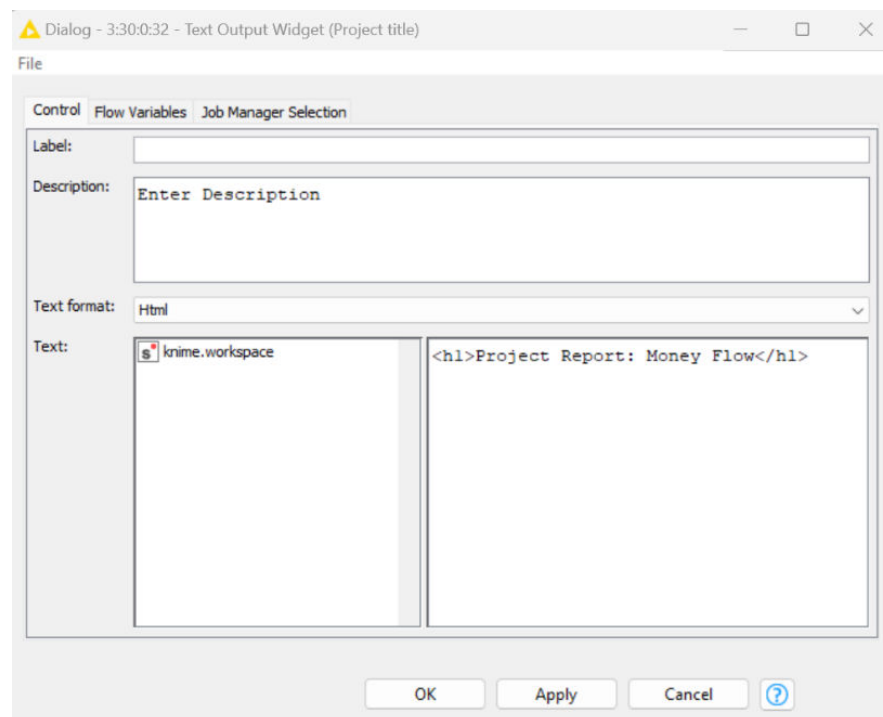


Figure 6.3. Configuration window for the Text Output Widget node.

6.3. The Component

Remember the metanode described in chapter 5? Components represent the natural evolution of metanodes. Components are like metanodes in the sense that they collect nodes inside. Like metanodes, to create a component:

- select all nodes of interest by clicking and drawing a rectangle around them or by shift-clicking/ctrl-clicking each one of them
- right-click and select “Create Component...”
- give it a name

and your new component is created. We named it “Dashboard”.

Workflow: Dashboard

In this workflow we prepare and visualize the data in a dashboard.

Starting from the accounting of a list of projects, on one branch we calculate the money in budget for each year for each one of the project and on the other branch we calculate the money actually spent for each project each year.

We then report the money allocated, the money spent and the money remaining for each project each year within ‘the view of the component “Dashboard”

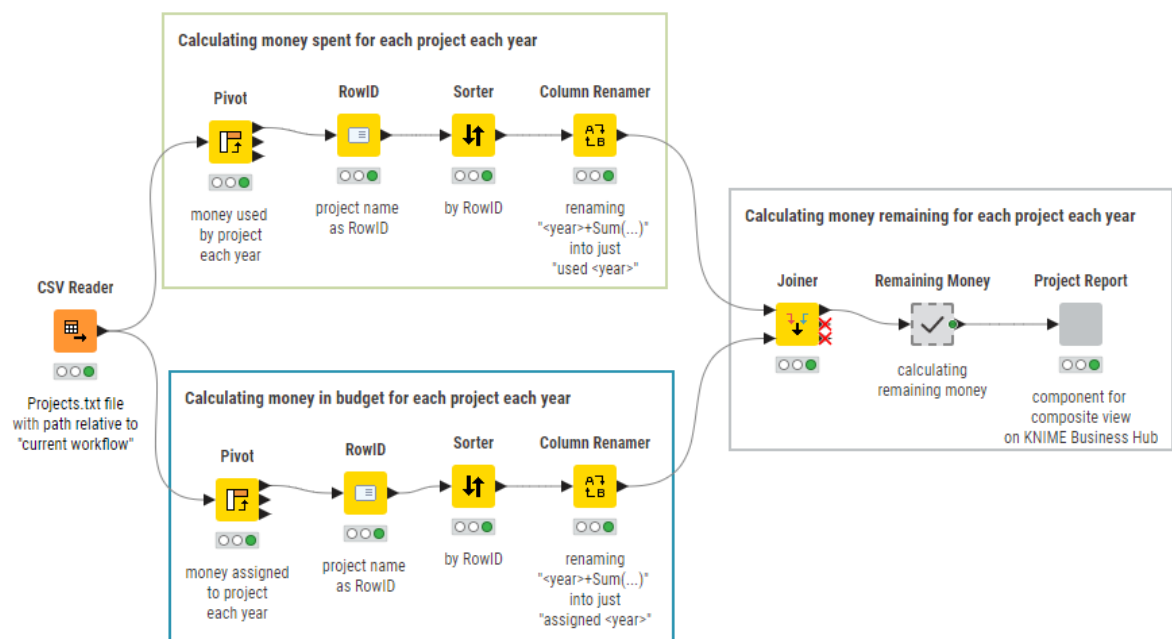


Figure 6.4. New workflow with component “Dashboard” at the end of the data flow for visualization.

Like metanodes, if you right-click the component and then select “Component”, you can open the component submenu. Here you can find the following options:

- “Open” to open and inspect the content of the component
- “Expand” to remove the component and reinstate the nodes back into the original workflow

- “Setup” to change some of the component settings, like for example the name or the input and output ports in number and type. “Setup” is equivalent to the option “Reconfigure...” for metanodes.
- “Convert to Metanode” to fall back on the familiar structure of a metanode
- “Share” to create a template on a local or a remote workspace that can be later reused to create linked instances of this same component.

So far, a component looks very similar to a metanode. What can a component do that a metanode cannot?

- **A component is an encapsulated environment.** We have not talked about Flow Variables yet. However, we can describe a component as a vacuum environment that only lets data in and out and nothing else.
- **A component can have a configuration window.** Components can have a configuration window, metanodes cannot. Inserting one or more nodes from the folder “Workflow Abstraction/Configuration” provides one or more items for the configuration window of the component. A component is a way to create a new node without coding! All of the node templates in “EXAMPLES/00_Components” in the KNIME Explorer panel (or here on the KNIME Community Hub) are actually components that have a configuration window.
- **A component can be given a view.** Components can get a view, metanodes cannot. Inserting one or more nodes from the folder “Workflow Abstraction/Widgets” provides one or more items for your component view. The interactive views of these nodes are passed into the interactive view of the component. Views with many items from many corresponding widget nodes are called composite views. In addition, Widget node views inside the same composite view subscribe to the selection and visualization of the same data. This means that what is selected in the view of a plot, for example, is also selected

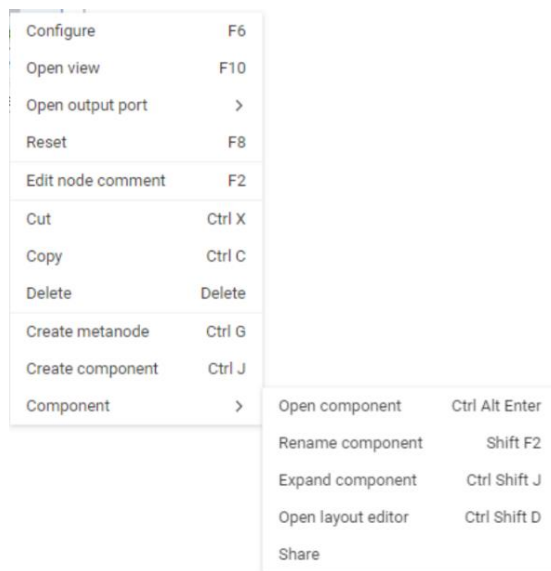


Figure 6.5. Context menu of a component.

(and can be visualized exclusively) in the view of another plot within the view of the same component. This is the part of components we are interested in.

All blue nodes within the “Dashboard” component produce a graphical view. Thus, the new component “Dashboard” has a view composed of three tables, three bar charts, and a text to work as title. This is the composite view of the component.

6.4. Adding Colors

What we have built so far is still in black and white. This last part is dedicated on how to introduce colors in KNIME plots and charts. We will use colors to identify projects in the tables and to identify years in the bar charts.

Project	remain 2009	remain 2008	remain 2007
Blue_Blue	-85	153	60
Gobi_Gobi	0	116	-17
Kalahari_Kalahari	14	32	-246
Kara Kum_Kara Kum	-28	-104	0
La Guajira_La Guajira	-22	-244	-180
Mojave_Mojave	51	-1	-200
Patagonia_Patagonia	-5	-41	-468
Sahara_Sahara	-175	-3	-99
Sechura_Sechura	-60	-147	-400
Tanami_Tanami	-15	0	-138
White_White	73	139	0

Figure 6.6. Table with assigned colors by project name.

To assign a color property to each project we just pass the original data through the Color Manager node. This automatically assigns a color to each data row. Colors can also be manually customized within the node configuration window. The data rows with colors reach the Table View node and get represented each with its own color on the left.

To assign a color to each year we need to make the column headers (“assigned 2009”, “used 2009”, ...) pass through the Color Manager node as data rows. To do that, we use the “Extract Table Specs” node. This node extracts the column properties – column headers, maximum and minimum values, etc. - and puts such information, including the column header names, on data rows. This table can be used to assign colors to the yearly columns through the Color Manager

node. Here, we became a bit more graphically creative, and we assigned dark green to all 2009 columns, mid green to all 2008 columns, and light green to all 2007 columns. This map is then sent to the second input port of the Bar Chart node to color the bars in the chart. Indeed, the second input port of the Bar Chart node, and of many other visualization nodes, requires a map (<column header>, <column color>) to transfer into the chart.

The final content of the component “Dashboard” is reported in Figure 6.1. Notice the two Color Manager nodes, one for the project names in the tables and one for the yearly amounts in the bar charts. Also notice the free-floating Text Output Widget node.

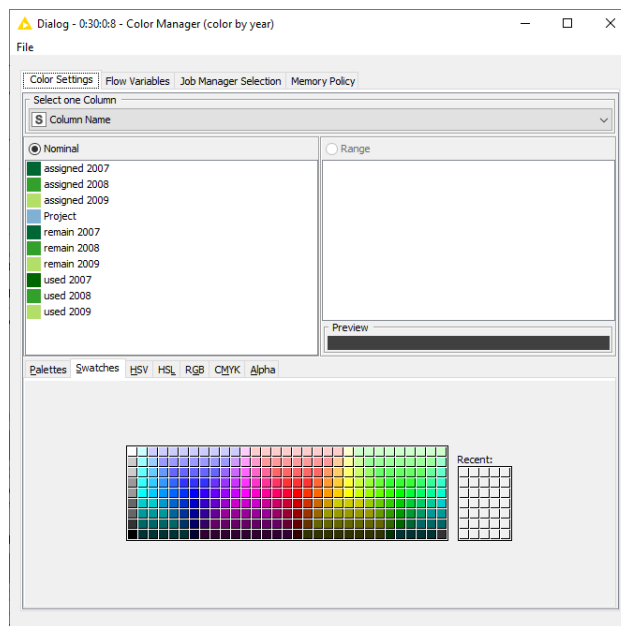


Figure 6.7. Shades of green assigned to the yearly columns in the Color Manager node.

Component for composite view

To create a component, select nodes to be included in the combined view. Then right-click and select "Create Component".

To arrange various views, open the component and click on the component configuration button on the toolbar (looks like stacked rectangles). Then you can adjust the placement and size of individual pieces on the report. The resulting composite view becomes a dashboard when executed on the KNIME Business Hub.

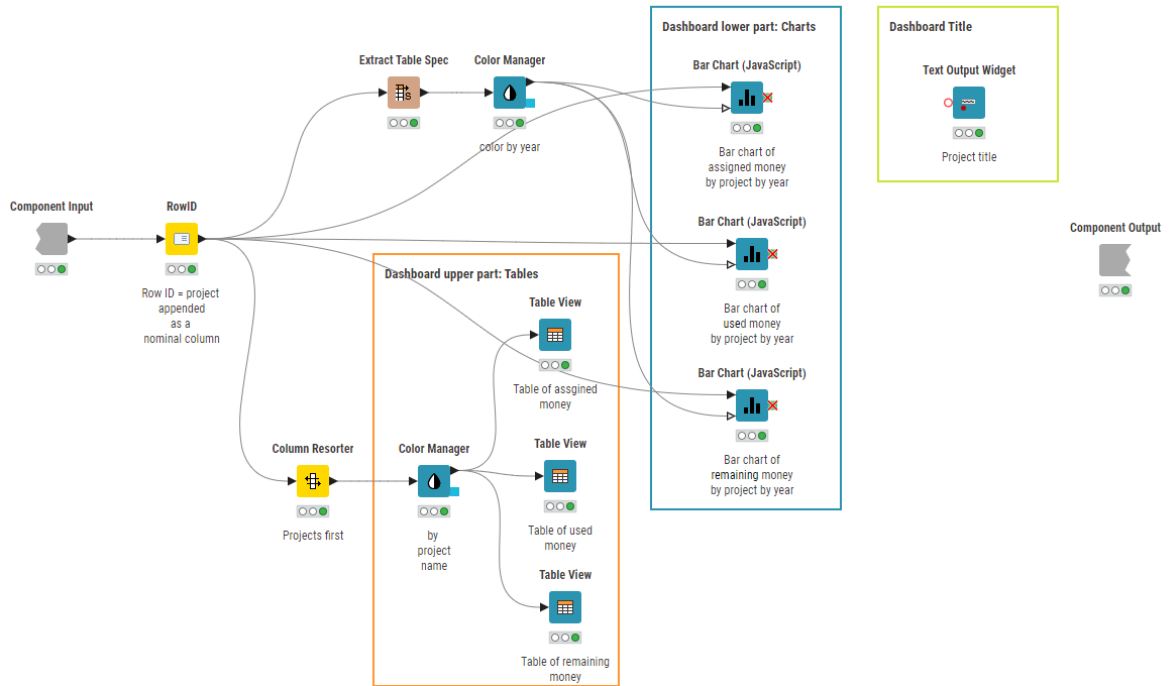


Figure 6.8. Content of component "Dashboard".

6.5. The Composite View

Let's inspect now the composite interactive view of the final component.

To open the interactive view of a component, right-click the component and select "Interactive View: <name of component>". If you open the interactive view of the component "Dashboard" now, you will see just a long list of tables and bar charts with no organized layout. A good layout could be:

- A title
- A row with the three tables
- A row with the three bar charts

The layout of a composite view is decided via the Layout button in the toolbar at the top of the KNIME workbench or if you right-click on the component -> component -> layout editor is the other way open the layout editor. After opening the content of the component, click the layout button to arrange the view items in the composite view. The layout editor opens.

In the layout editor you will see on the left all view items pertaining to the visualization nodes within the component and a set of possible row grids. All these items can be moved into the final view by drag&drop. We would like to have:

- a full row dedicated to the title, that is the view produced by the Text Output Widget node
- a row with three cells for the three tables
- a row with three cells for the three bar charts

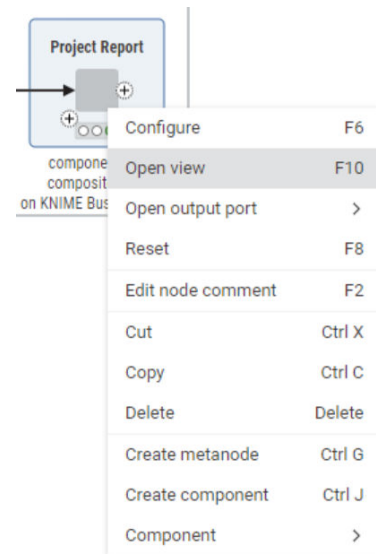


Figure 6.9. How to open the composite view of a component.

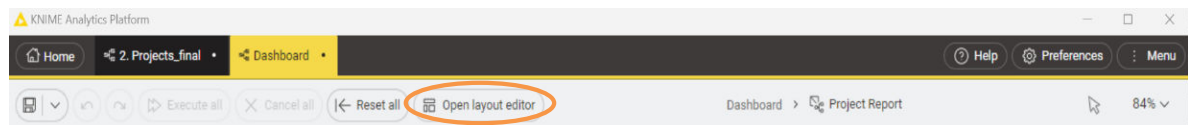


Figure 6.10. The layout button in the tool bar in the KNIME Workbench.

We drag&drop the row with one cell only at the top; then the row with three cells; then again, the row with three cells. Then, we drag&drop the “Text Output Widget” item into the first row, the “Table View” of node “Table of Assigned Money” in the first cell of the second row, and so on till all cells are populated as desired.

Notice the trash basket, the plus sign, and the setting wheel in the top right corner of each cell and row, respectively to add one more cell/row, remove the current cell/row, and customize them.

The view now with the new layout should resemble the dashboard shown in Fig. 6.1, with one important addition: interactivity. Let’s have a look at the interactivity options derived from the “Interactivity” settings in the configuration windows.

To open the composite view of a component, again, right-click the component and select “Interactive view: <name of component>”. The view opens and it contains the single views from each one of the visualization nodes within the component.

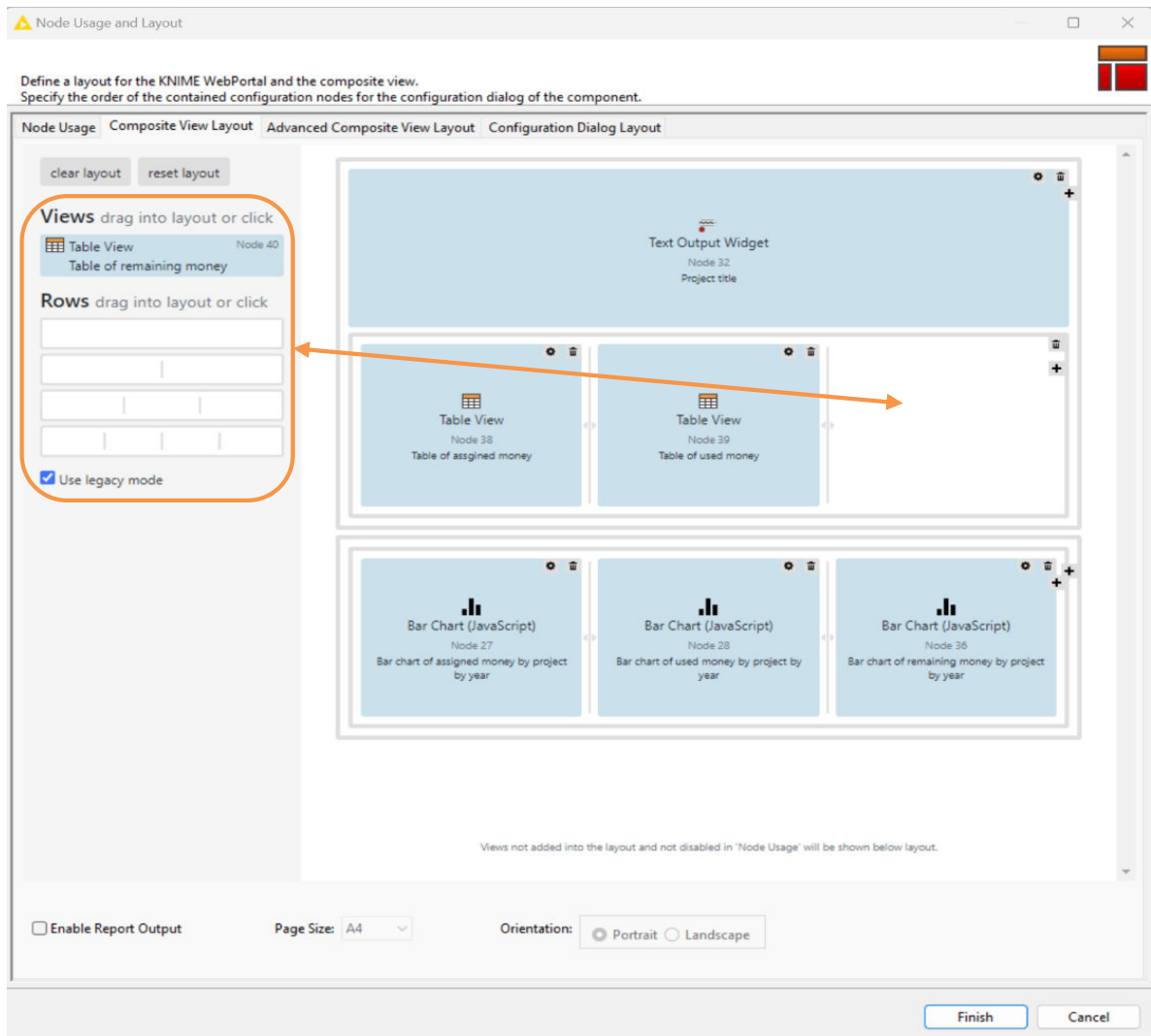


Figure 6.11. The Layout Editor.

Customize each item in the composite view

If we focus on the single view item in the composite view, we notice three buttons in the top right corner.

From the right, the first button allows you to change all settings in the plot/table/chart, like titles and labels, visualization mode, etc. ... In some plots it even allows you to change the columns reported on the x and y axis.

The second button is to zoom the current chart into full screen.

The third button clears all previous settings.

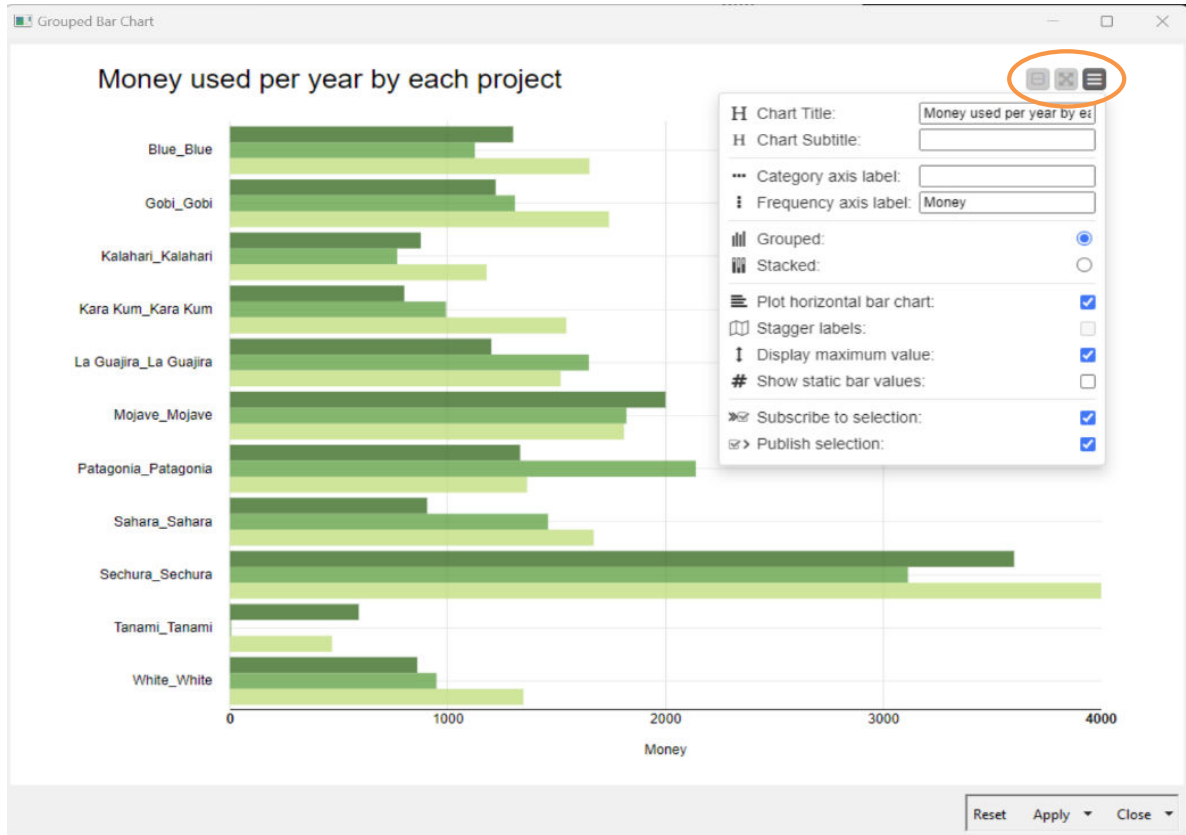


Figure 6.12. One of the bar charts in the composite view of component "Dashboard".

Selecting and visualizing data rows

Many plots and charts offer tooltips when hovering over the chart/plot area. In the case of the bar chart, hovering over the bars provides a tooltip with the exact number represented by the bar.

Also, bars/points/rows in a chart/plot/table can be selected and the same selection appears in all other view items if subscription and publishing was enabled in their configuration settings. For example, I have selected project Kalahari and Kara Kum in the central table. The same selection automatically appears in all other tables as well as in all bar charts. In addition, many simple views, like the table view in this composite view, offer the option of visualizing only the

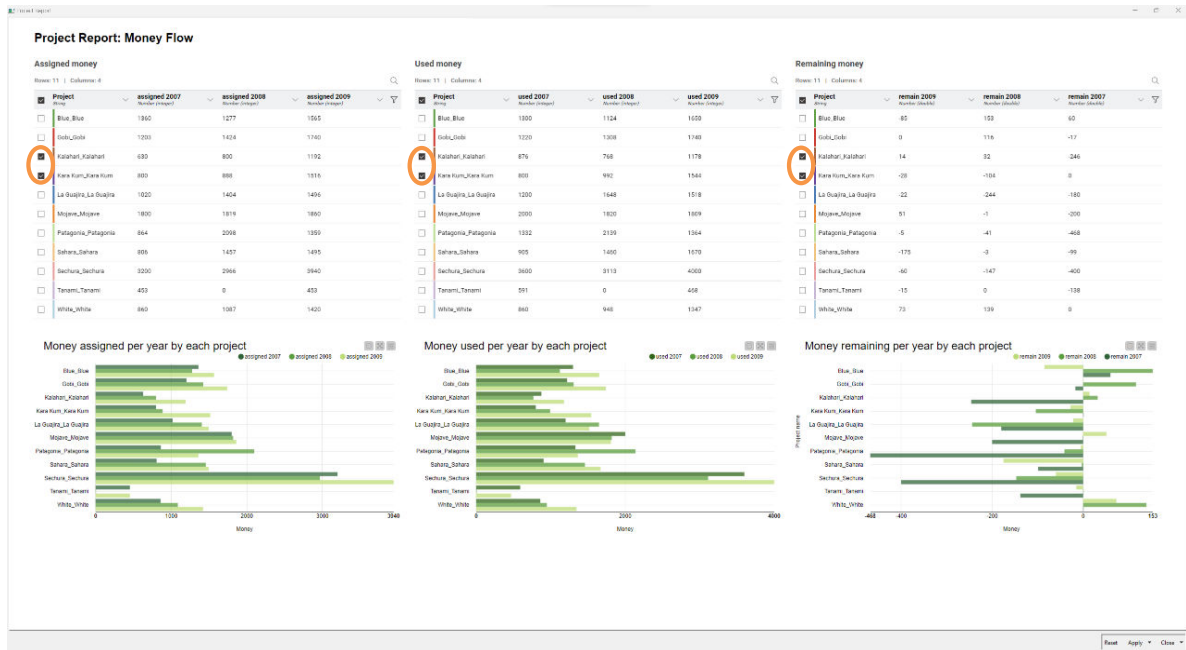


Figure 6.13. Selection of points across all view items.

selected rows in the settings menu from the top right button.

6.6. Executed as a Data App on the KNIME Business Hub

This workflow, like all other workflows, can then be executed on the KNIME Business Hub for production. There, composite views of components become a data app on the KNIME Business Hub.

After logging in into a KNIME Business Hub from a web browser and after starting the execution of the workflow, the composite view of the first component in the workflow appears on the web browser in the shape of a web page; after interacting with page if needed and pressing "Next", the composite view of the next component in the workflow appears, and so on.

In our case, our workflow (Fig. 6.3) has just one component with a composite view. That is, starting the workflow on the KNIME Business Hub will land us onto the one and only web page

generated by the composite view of the component “Dashboard”. Here we can interact with all items of the view exactly as we did with the local view of the composite view.

Notice that more complex Widget and data visualization nodes could be introduced into the component for an even more interactive experience. Since this book is only about introducing the reader to the basic features of KNIME Analytics Platform, we will stop here. However, keep in mind that more advanced features - like slide bars for filtering, refresh buttons, and more - could be also implemented within the component.

6.7. Exercises

The exercises for this chapter follow on from the exercises in Chapter 5. In particular, they require shaping a report layout for the data sets built in Chapter 5 exercises.

Exercise 1

Using the workflow built in Chapter 5\Exercise 1, build a report with:

- A title “income by work class”
- A table on the left side like:

Work Class	Income <= 50K	Income > 50K
[workclass]	[no <= 50K]	[no > 50K]

With colors assigned to each workclass.

- A bar chart on the right with:
 - Work class on the x-axis
 - “Income <= 50K” and “Income > 50K” on the y-axis
 - Legend available
 - No title
 - No axis titles
 - Color blue for bars <=50K and color green for bars >50K
- Select “Local-gov” in both the table and the bar chart

Solution to Exercise 1

We built the component “Dashboard2” with

- A Table View node
- A Bar Chart node
- A Text Output Widget node

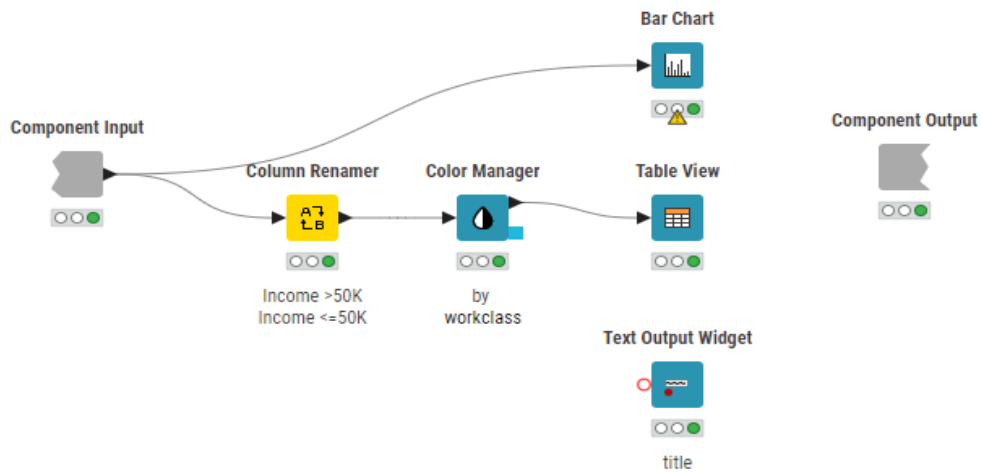


Figure 6.14. Component “Dashboard2” to implement required composite view.

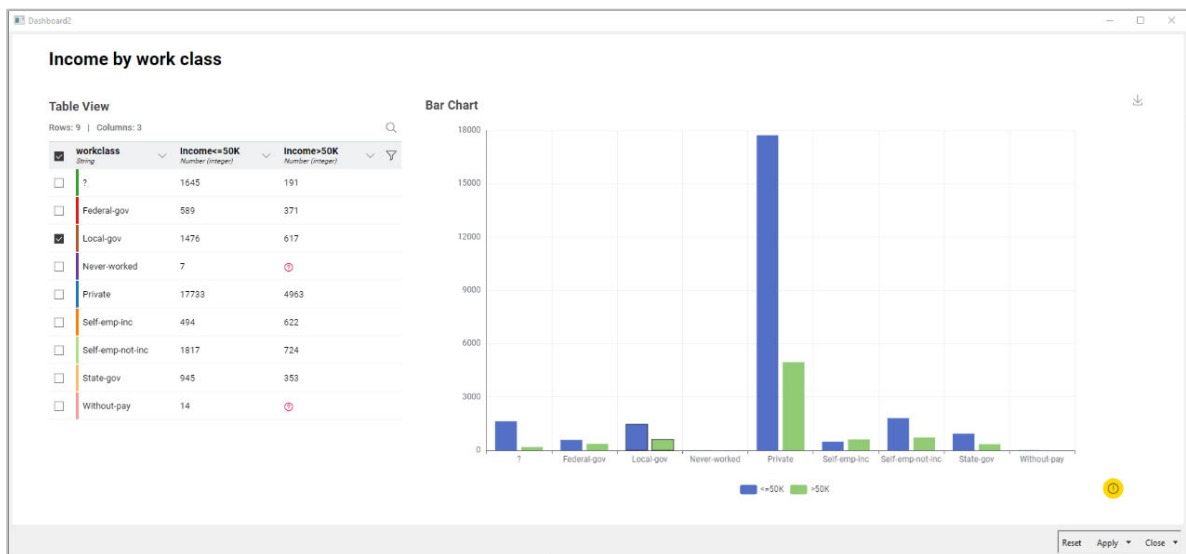


Figure 6.15. The final composite view with class “Local-gov” selected in both the table and the bar chart.

Chapter 7: Reporting in KNIME

7.1. KNIME Reporting (Labs)

KNIME Analytics Platform already includes powerful visualization features. The [KNIME Views](#) extension, for example, provides several types of interactive charts which can be assembled into [composite views](#) as dashboards or [Data Apps](#).

BIRT (Business Intelligence Reporting Tool) is already integrated into KNIME for static reports and is described in detail in section 7.2. Although BIRT is a great reporting tool, it can be difficult to master due to its complexity. With the release of the KNIME Modern UI, which simultaneously has simplicity, functionality and beauty, could we also have a new easy-to-use reporting

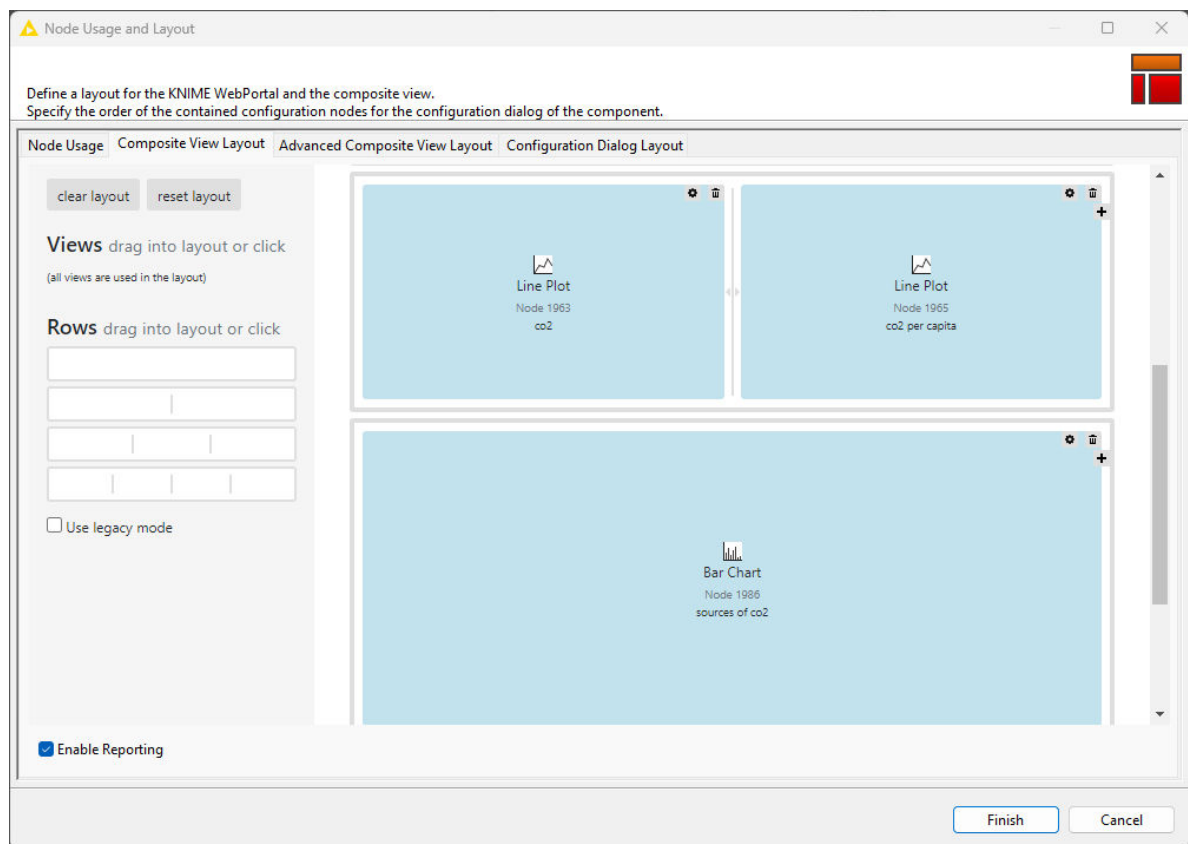


Figure 7.1. The Node usage and layout window

feature? Such a feature could have a proper harmony with other existing KNIME capabilities and make the generation of hard-to-create static reports a piece of cake.

Well, the feature we just described is here! The recent release of 5.1 and 5.2 includes a new [KNIME Reporting](#) extension that allows you to take the composite visualizations you build and render them into static PDF reports quickly and easily. The best part of this new feature is you don't need to learn anything new; all you need is what you are already familiar with, plus enabling report output in the layout settings of a component with a view. Then, you will have a new report output port, which can be connected to the [Report PDF Writer](#) node to generate a PDF file.

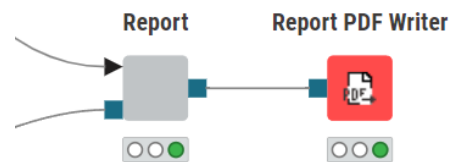


Figure 7.2. The Report PDF Writer node.

Let's look at a use case regarding the CO2 Emissions report, focusing on the highest producers of CO2 emissions in 2020. We are going to illustrate this use case with the help of the new [KNIME Reporting](#) extension. We will use the open-source data from the [World in Data](#) project. This data encompasses information on countries, years, CO2 emissions, and emission sources.

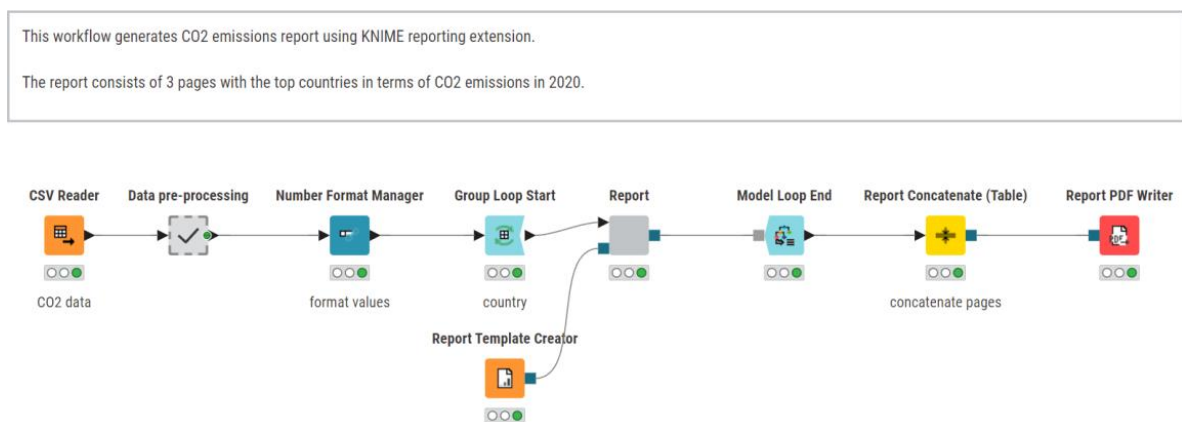


Figure 7.3. Workflow generating CO2 emissions report using the KNIME Reporting extension.

Inside the data pre-processing metanode, we will read and clean up the CO2 data. We limit our data to the period from 2000 to 2020 and select only the countries with the highest emissions in 2020 (China, the United States, and India).

Since we would like to create a report where each page represents the statistics for one country, we use the [Group Loop Start](#) node to iterate over countries.

Chapter 7: Reporting in KNIME

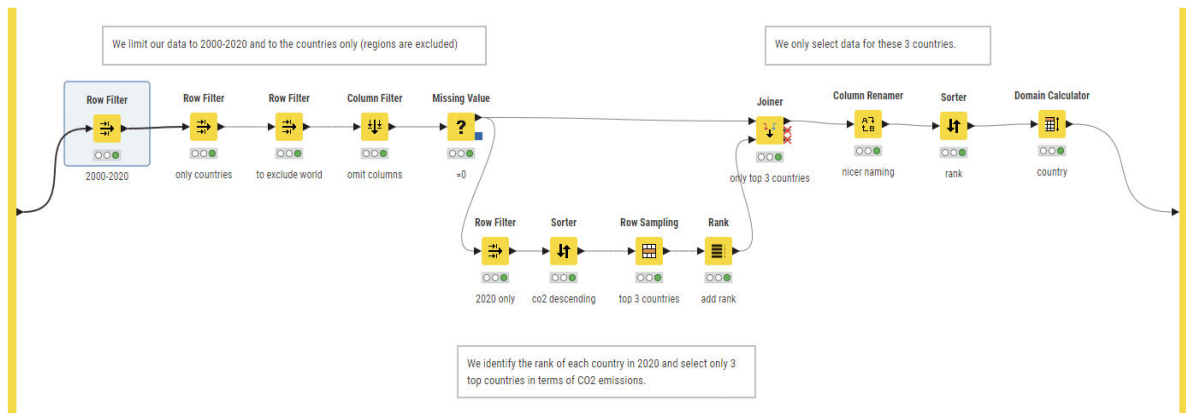


Figure 7.4. Data Pre-processing steps inside the metanode.

After the data preprocessing is done, we are ready to continue with the next step: building the report. In our report, we will display a few visualizations, a table view, and text views. Let's look inside our component to see how it's done.

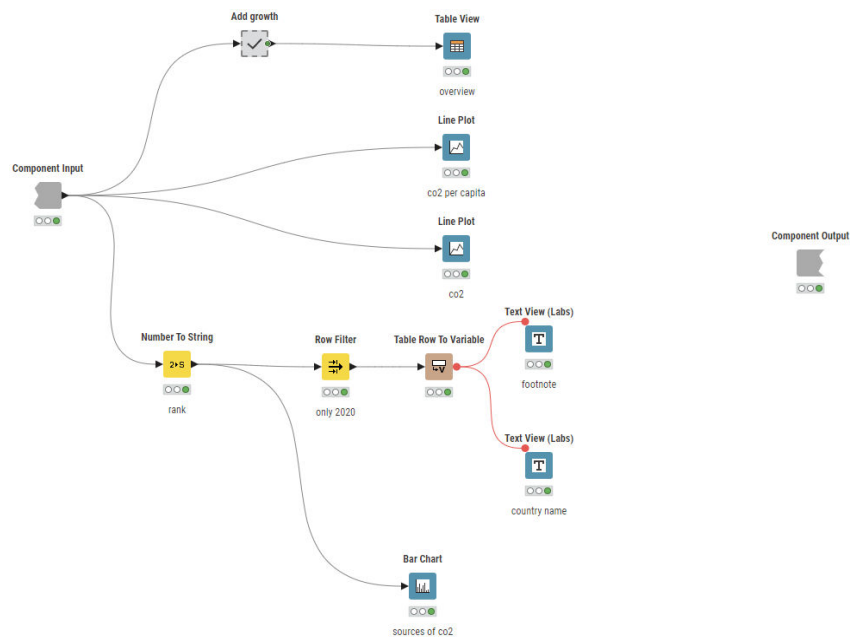


Figure 7.5. Visualization nodes inside the "Report" component.

With the [release of 5.1](#), the new HTML column type was introduced, enabling us to apply styling to our text and render it as HTML. Inside the "Add growth" metanode, we use the [Column Expressions](#) node to change the text color of CO2 values. If the growth rate of CO2 emissions is positive ("+"), then the value will be red. If the growth rate is negative ("-"), then the CO2 emissions in the country are decreasing; thus, these values are displayed as green. Another

interesting feature released with version 5.1 is the [Text View \(Labs\)](#) node. Using this node, we add the country's name as the title and the footnote at the bottom of each page.

After finishing all our tables and views, we wrap our visualizations and table inside the component and "Enable Report Output" in the Layout Editor. If we preview the component's output, we will see the first page of our generated report.

To close the loop, we will use the [Model Loop End](#) node. This node converts the model into data cells and collects the results into a data table. To concatenate all pages that we have generated, we use the [Report Concatenate \(Table\)](#) node.

The last step of the workflow is to export the pdf report. To do so, we use the [Report PDF Writer](#) node to specify the file's output location (path and file name).

And that's it! with the new [KNIME Reporting](#) extension, it's very easy to generate static PDF reports in KNIME. You can download the extension and give it a try to see how it works.

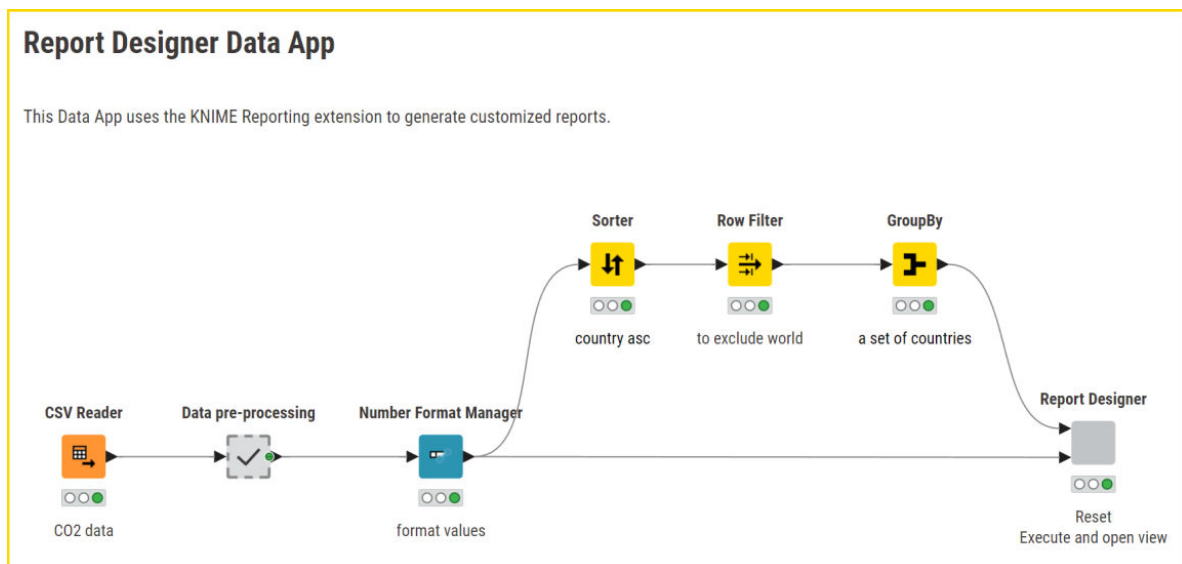


Figure 7.6. Preprocessed data to be sent for report generation.

How to build the report designer

Now that you are familiar with how to generate a PDF/HTML report out of your composite views in KNIME, we are going to see how to build an online report designer. The purpose of this report is to provide accessibility to the entire team via a browser-based data app. It will enable any of the team members to design and generate their own custom reports without needing to be familiar with the underlying process.

At its core, the data app generates a customized report and sends it via email. Our example demonstrates a case where we have three individuals involved in the process: The Manager, who receives the report, a Data Analyst, who can customize and generate the report, and a Data Scientist, who builds the data app with KNIME.

The data app will enable users to:

- Select the main countries to be included in the report
- Select other countries for comparison with the main countries
- Select the sources of CO2 emissions to be included in report
- Select between multiple report designs
- Switch between different charts
- Select to show or hide a chart
- Configure the charts in the report
- See a quick preview of the report without the need to switch windows
- Send the report via email

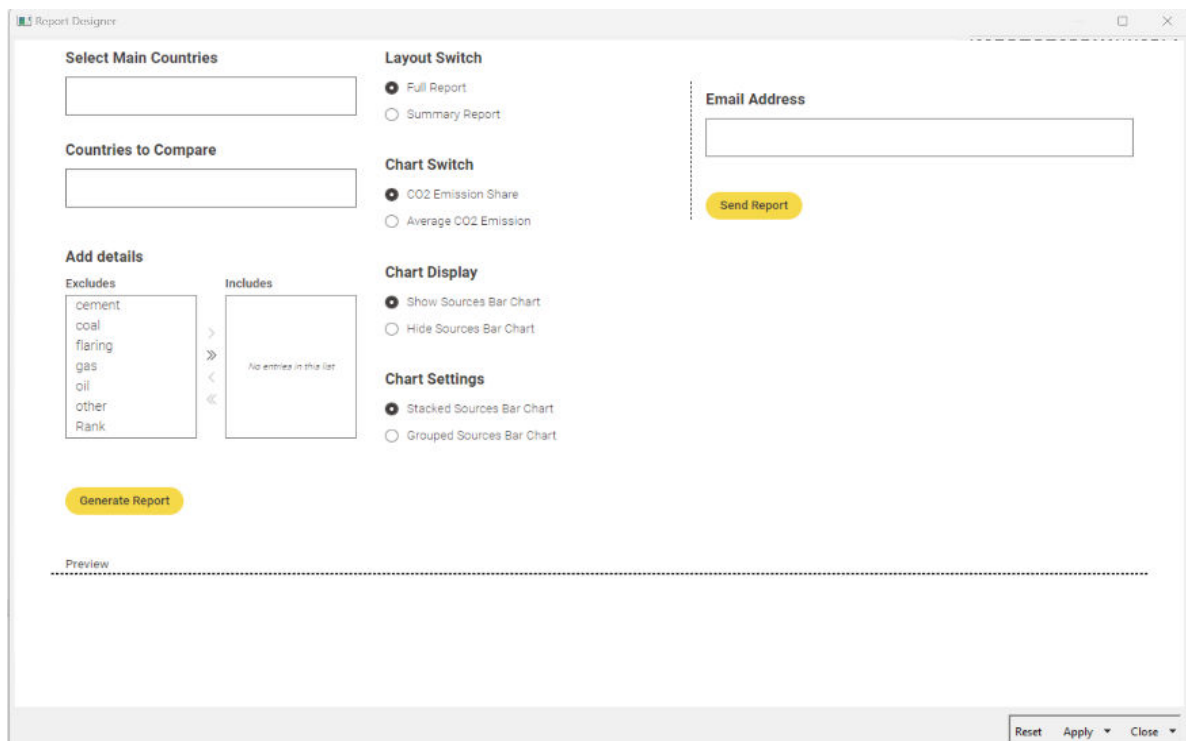


Figure 7.7. DataApp showing different selection options.

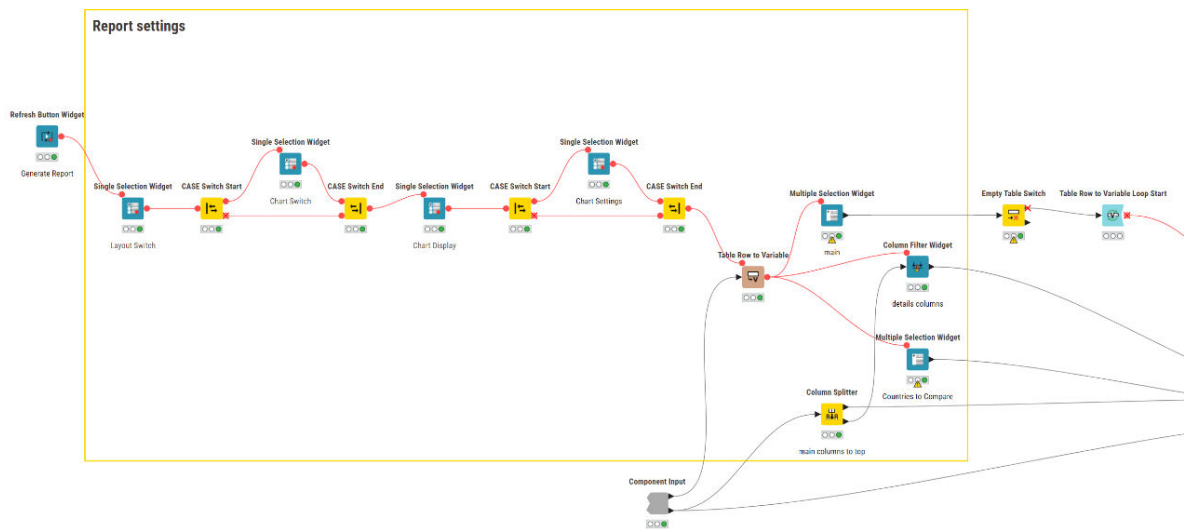


Figure 7.8. Preprocessed data to be sent for report generation.

Taking our [workflow example](#) that we built in the previous section as the starting point, let's now modify the preprocessing metanode by including the "World" value (in the countries) and passing the top 15 countries to the next node in the flow (instead of just 3). This will give the Data Analyst more options to choose from in the app.

Also, using the GroupBy node, we create a set of countries to provide the Data Analyst with country selection options. Later, this set will be passed to the Multiple Selection Widget nodes (inside the Report Designer component) as a flow variable.

Let's open the "Report Designer" component and check inside.

In this part of the component, we have nodes to let the Analyst choose which kind of charts he wants to include in the report: widgets to switch between pie chart and a bar chart to visualize CO2 emissions, an option to display or hide the bar chart that shows the sources of emissions, and the ability to choose between a stacked or grouped bar chart.

The first Single Selection Widget node lets the Analyst choose from two different report layouts. In the "Report layouts" section, you can see the two components with the report output ports.

Each component produces a different layout. The Summary Report component generates 1 page for each main country, while the Full Report component creates a report with 2 pages for each selected country. The Summary Report layout is a simplified report: it includes fewer charts, and

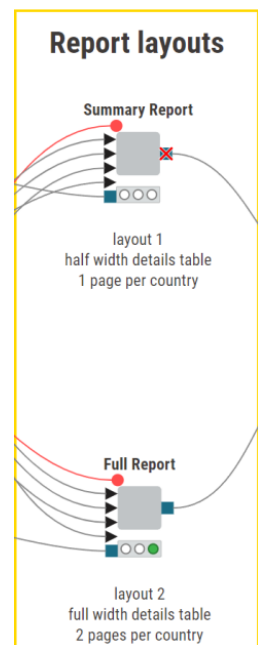


Figure 7.9. Components to select different layout options.

displays the Table View in half-width. By using components, the Analyst can add as many layouts as she needs in a single workflow without having to build a new workflow for each layout.

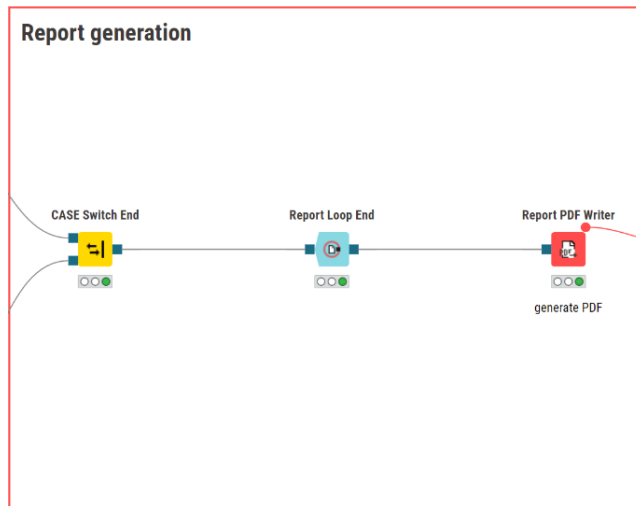


Figure 7.10. Section of the workflow that generates report.

In the “Report generation” section, the PDF file is written in the data area of the workflow so we can access it by the File Download Widget and Send Email nodes.

Finally, a Multiple Selection Widget is used to get the Email address(es), and the Send Email node is used to send the report.

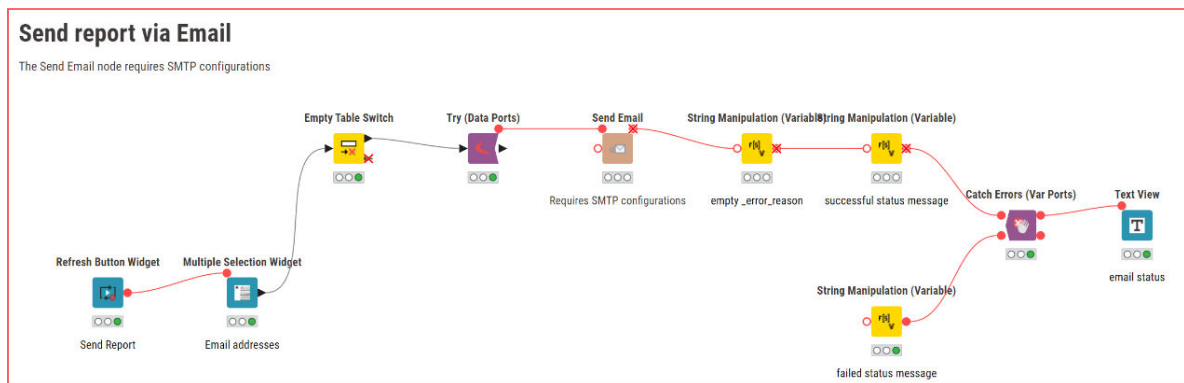


Figure 7.11. Section of the workflow that gathers email addresses and sends the report using the “send email” node.

7.2. Reporting with BIRT

KNIME analytics Platform also offers an integration with the open-source version of the BIRT (Business Intelligence and Reporting Tool) BI tool. The BIRT integration is contained in a KNIME

extension named "Report Designer". This extension installs the interface between KNIME Analytics Platform and BIRT as well as the open-source version of BIRT. Thus, you do not need to have a pre-installed version of BIRT working on your machine to use the extension, as it is the case for the other BI tools. You also do not need to buy a license, which makes the integration between BIRT and KNIME Analytics Platform much easier to handle. Because of all of that, in this chapter we will focus on how to build a report using the Report Designer

Workflow: Reporting_w_BIRT

In this workflow we prepare data to be displayed in a report.

Starting from the accounting of a list of projects, on one branch we calculate the money in budget for each year for each one of the project and on the other branch we calculate the money actually spent for each project each year.

We then export the money allocated, the money spent and the money remaining for each project each year and build a report with the BIRT integration within KNIME.

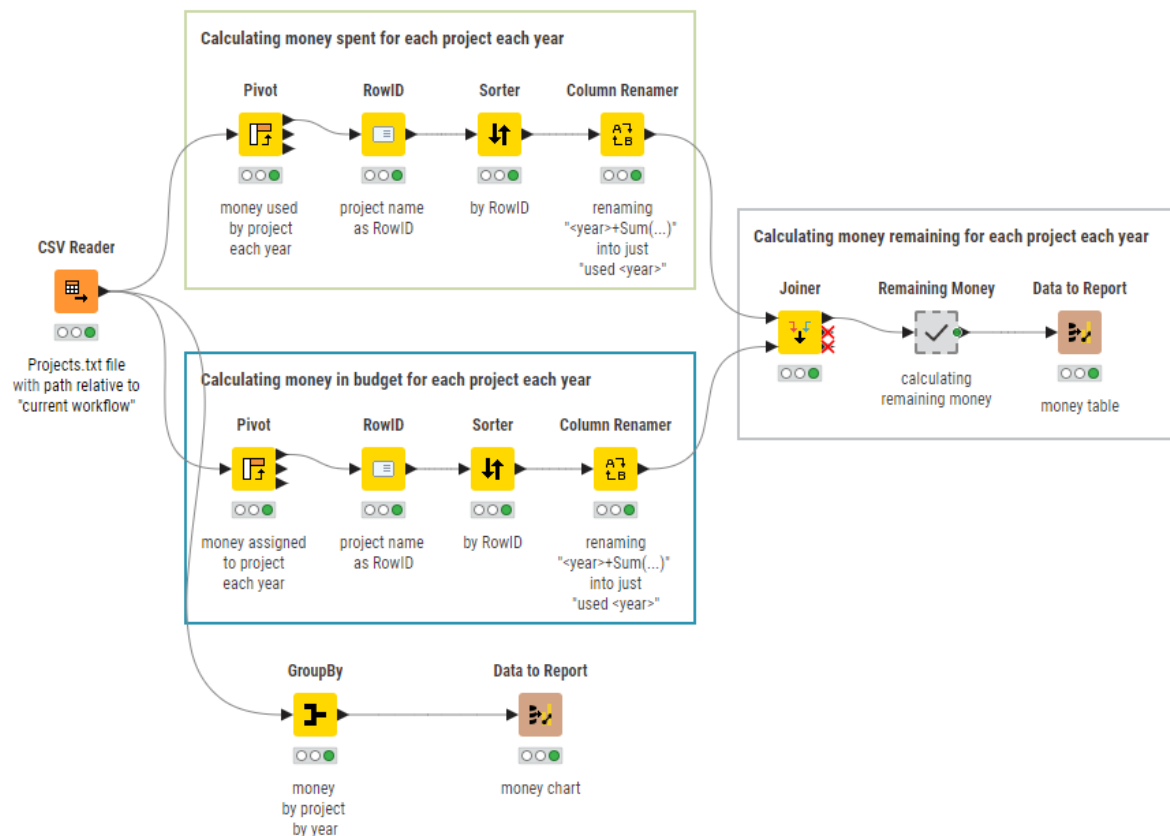


Figure 7.12. The "Reporting_w_BIRT" workflow.

Extension (BIRT integration). In this chapter we will use the BIRT integration to build a similar report to the dashboard built in the previous chapter.

In folder Chapter 7 of the downloaded material, you can find a number of workflows – originating from the workflow “Projects_final” in the previous chapter – to export the data to different BI tools. In particular, workflow #1 and #2 also produce a similar dashboard for BIRT and the data app deployed on the KNIME Business Hub respectively. All these workflows work on the “Projects.txt” file available in the book’s data folder “KBLData”. The “Projects.txt” file

Row ID	name	used 2007	used 2008	used 2009	assigned 2007	assigned 2008	assigned 2009	remain 2007	remain 2008	remain 2009
Row0_Row0	Blue	1300	1124	1650	1360	1277	1565	60	153	-85
Row1_Row1	Gobi	1220	1308	1740	1203	1424	1740	-17	116	0
Row2_Row2	Kalahari	876	768	1178	630	800	1192	-246	32	14
Row3_Row3	Kara Kum	800	992	1544	800	888	1516	0	-104	-28
Row4_Row4	La Guajira	1200	1648	1518	1020	1404	1496	-180	-244	-22
Row5_Row5	Mojave	2000	1820	1809	1800	1819	1860	-200	-1	51
Row6_Row6	Patagonia	1332	2139	1364	864	2098	1359	-468	-41	-5
Row7_Row7	Sahara	905	1460	1670	806	1457	1495	-99	-3	-175
Row8_Row8	Sechura	3600	3113	4000	3200	2966	3940	-400	-147	-60
Row9_Row9	Tanami	591	0	468	453	0	453	-138	0	-15
Row10_Row10	White	860	948	1347	860	1087	1420	0	139	73

Figure 7.13. The data table from the “money table” node of the “Reporting_w_BIRT” workflow.

contains a list of project names with the corresponding amount of money assigned and used for each quarter of each year between 2007 and 2009. All workflows build a pivot table with the project names and the sum of the money assigned, the sum of the money used, and the sum of the money remaining (= assigned - used) for each project and for each year between 2007 and 2009. The resulting data fills a few tables and two bar charts in the associated report.

Installing the Report Designer Extension (BIRT)

The “KNIME Report Designer” suite is not included in the basic standalone version of KNIME Analytics Platform. It can be downloaded as a separate extension package from the “KNIME & Extensions” link in “File” → “Install KNIME Extensions”.

To install the “KNIME Report Designer” package:

- Start KNIME Analytics Platform
- In the Top Menu click “File” → “Install KNIME Extensions...”
- In the “Available Software” window, expand “KNIME & Extensions” and scroll down to “KNIME Report

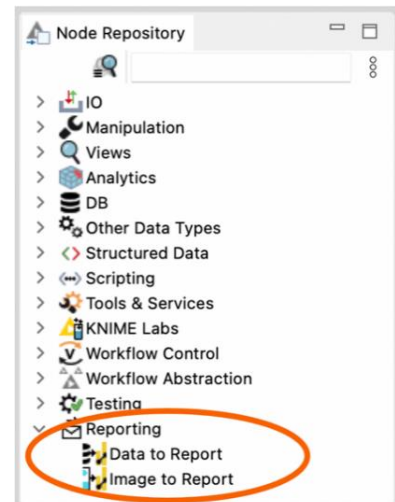


Figure 7.14. The Reporting category in the Node Repository.

Designer”. Alternatively, search for “KNIME Report Designer” in the search box at the top.

- Select extension “KNIME Report Designer”
- Click the button “Next” on the bottom and follow the installation instructions

If the installation runs correctly, after KNIME Analytics Platform has been restarted, you should have a new category “Reporting” in the “Node Repository” panel with two nodes: “Data To Report” and “Image To Report”.

Marking Data in the Workflow

The KNIME reporting tool is a different application (BIRT) from KNIME Analytics Platform. The idea is that the KNIME workflow prepares the data for the KNIME Report Designer, while the KNIME Report Designer displays this data in a graphical layout.

The two applications, the workflow editor and the reporting tool, need to communicate with each other; in particular, the workflow needs to pass the data to the reporting tool. This data communication between workflow and reporting tool happens via the “Data to Report” node.

Data to Report

The “Data to Report” node can be found in the “Node Repository” panel in the “Reporting” category. The “Data to Report” node marks the KNIME data table at the input port as a data set for the KNIME Report Designer.

When switching from the workflow editor to the reporting tool, all data tables marked by a “Data to Report” node are automatically imported as data sets in the reporting tool. Each data set carries the name as the text below the originating “Data to Report” node. Therefore, the text under the “Data to Report” node is important! It has to be a meaningful text to facilitate the identification of the data set in the report environment.

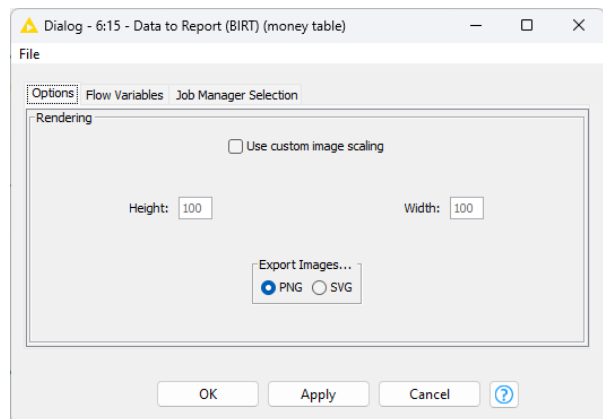


Figure 7.15. Configuration window of the Data to Report node.

Since the “Data to Report” node is only a marker for a data table, it does not need much configuration. The configuration window contains just a flag “use custom image scaling” to scale images in the data to a custom size. The default image size is the renderer size.

We used two “Data to Report” nodes in our workflow. One is connected to the sequence of “Math Formula” nodes – in the metanode named “Remaining Money” - and exports the data for the tables in the report. The second one is connected to the “GroupBy” node texted as “money by project by year” and exports the data for the charts in the report. We added a text “money table” under the first Data to Report node and a text “money chart” under the second Data to Report node. Thus, when switching to the reporting tool, we will find there two data sets called “money table” and “money chart” respectively. We will know immediately which data to use for the tables and which data for the charts.

In the category “Reporting” there is also the “Image to Report” node. The “Image to Report” node works similarly to the “Data to Report” node, it only applies specifically to images.

From KNIME to BIRT and Back

In KNIME Analytics Platform we develop workflows for data manipulation and modeling. In BIRT we create and shape the report to represent the workflows’ data. Only one report is associated to one workflow and vice versa. It is not possible to associate more than one report to one workflow. When we move into the BIRT environment, we open the report associated with the active workflow. From a KNIME workflow open in the KNIME workbench, you can switch to the BIRT environment and open the associated report by:

- Opening the workflow from the “KNIME Explorer” panel into the workflow editor
- Clicking the “Report” icon in the tool bar.

The BIRT report editor then opens the report associated with the selected workflow. The report editor creates a new tab in the KNIME Workflow Editor window.

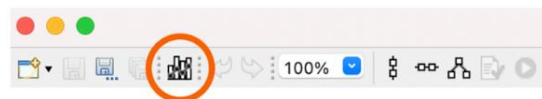


Figure 7.16. The report icon in the tool bar.



Figure 7.17. The new tab in the KNIME Workflow Editor for the selected report.

To go back from the report to the workflow editor, you can either select the workflow tab or click the KNIME icon in the tool bar. This will take you back to the more familiar KNIME environment.

If it is the first time you open the report associated with the workflow, it will be empty.

Note. If the workflow has no Data to Report node, the “Report” icon is not present in the tool bar.

Double-click the “Reporting_w_BIRT” workflow in the “KNIME Explorer” panel to open it; then select the report icon in the tool bar. This takes you to the BIRT environment, to the associated report.

The BIRT Environment

BIRT is developed as an Eclipse Plug-In, as KNIME Analytics Platform is. This means that they both inherit a few properties and tools from the Eclipse platform. As a consequence, the BIRT

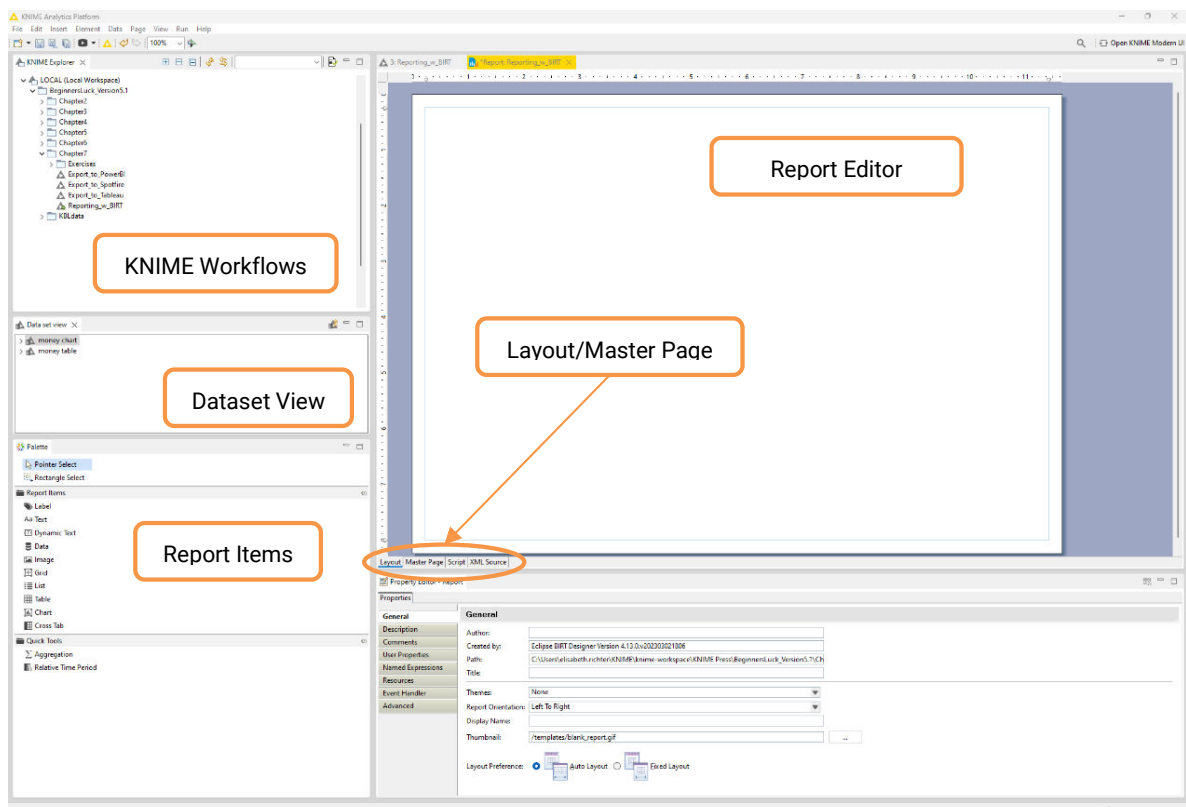


Figure 7.18. The Report Editor in the BIRT environment.

report editor and the KNIME workflow editor are very similar, which makes our learning process easier for the reporting tool. In this section we provide a quick overview of the BIRT report editor. For more information on the BIRT software, the book listed in⁹ gives a detailed overview of BIRT potentials. Let's have a look at the different windows in the BIRT environment with an empty report.

The "**KNIME Explorer**" panel is still in the top left corner, and it still contains the list of available KNIME workflows.

Under the "KNIME Explorer" panel, we find the "**Data Set View**" panel. This panel contains all data sets that are available for the report.

Under the "Data Set View" panel, we find the list of all available "**Report Items**" to create our report, like Table, Label, Chart, and so on.

In the center, as for the KNIME workflow editor, we find the **report editor**. Like in KNIME, where we built workflows by "dragging and dropping" the nodes into the workflow editor, here we can compose the report by "dragging and dropping" the report items into the report editor.

Finally, in the center bottom of the window there are a few tabs, of which only two are interesting for our work: Layout and Master Page.

Layout is the page editor, where the single report page is processed.

Master Page, as in PowerPoint Master Page, defines a template for every page of the report. This is where the page header and footer are designed.

Master Page

Right below the report editor, there are a few tabs: "Layout", "Master Page", and others. Let's select tab "Master Page".

Now the report editor in the center has become the Master Page editor and, below the tabs, you can see the Master Page's Properties Editor. There are 6 property groups: "General", "Border", "Margin", "Header/Footer", "Comments", and "Advanced".

We would like to prepare a report to be exported into slides in PowerPoint format. We also want to have a running title with a logo on all the slides.

⁹ D. Peh, N. Hague, J. Tatchell, "BIRT. A field Guide to Reporting", Addison-Wesley, 2008

Chapter 7: Reporting in KNIME

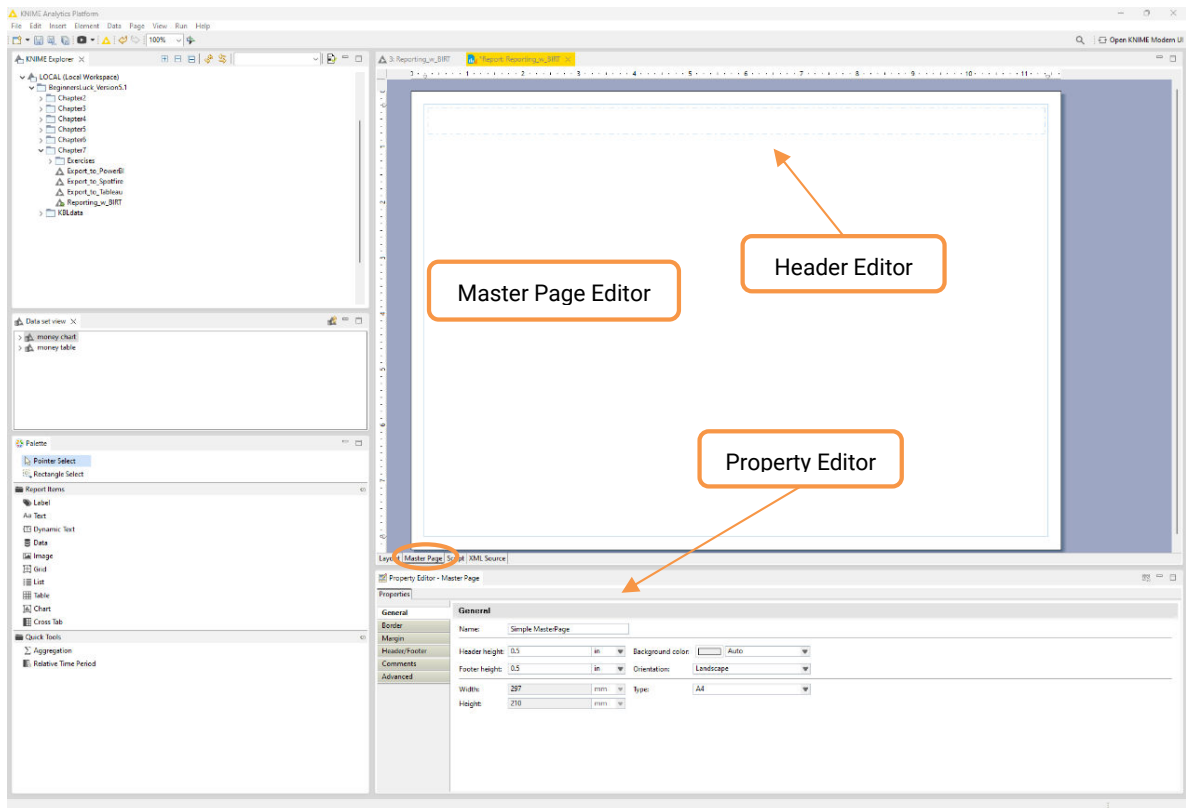


Figure 7.19 The Header Editor inside the Master Page Editor.

Usually, PowerPoint slides have a landscape orientation. To change the paper orientation, we go to the “Orientation” field under the property “General”. We change it to “Landscape”.

Header/Footer” offers only check boxes about showing or not showing the header and the footer. In order to actually change the header and the footer, we need to work in the Master Page editor itself. In the top part of the Master Page editor there is a dashed rectangle. This is the header editor.

To insert a logo in the header of each slide go to the Master Page editor:

- Right-click the header editor
- Select “Insert”
- Select “Image”
- In the “Edit Image Item” window upload your image, for example as an embedded file

The logo image will appear in the top left corner of the header editor.

Instead of an image you can insert a “Label” in the header editor to have a running title in your slides. You can also combine both, a running title and a logo, in the header editor. However, you can only combine more report items side by side by using the “Grid” report item.

To see how the report will look like, you need to select “Run” > “View Report” in the top menu and then your output format. This generates the real report. For a quick preview you can choose “In Web Viewer” for a quick creation of the HTML report page. For the moment it is just the logo we have introduced in the top left corner and the footer with the KNIME logo.

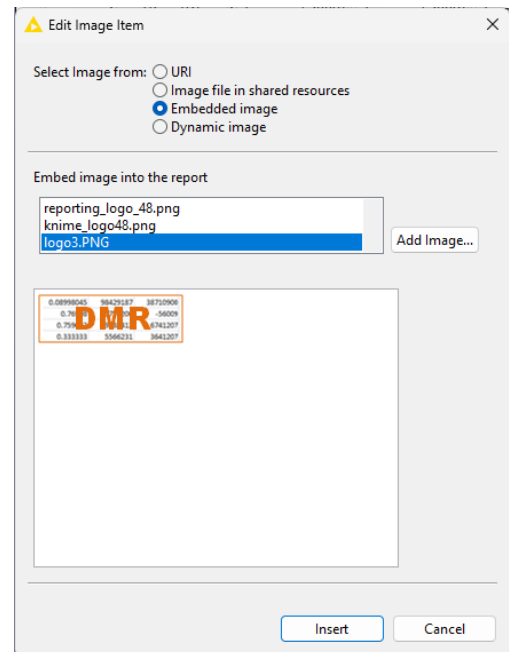


Figure 7.20. The “Edit Image Item” window.

The Data Sets

The panel named “Data set view” contains the data available for the report. Each report is linked to one and only one workflow. In the integration of BIRT inside KNIME, data sets are automatically imported from the data tables marked by a “Data to Report” node in the underlying workflow. In the integrated version, there is no other way to generate data sets in the reporting environment.

Let’s have a look at the data sets available for the report of workflow “Reporting_w_BIRT”.

In the “Data Set View” panel you should see two data sets, named “money chart” and “money table”. These were the names of the two “Data to Report” nodes in the “Reporting_w_BIRT” workflow. Indeed, when switching from the KNIME workflow editor to the BIRT report editor, the data of the “Data to Report” nodes are automatically exported as data sets into the report environment.

If you have used obscure dataset names and cannot remember which “Data to Report” node the data set has been generated from or to check that the data set got exported correctly, you might need to preview the data in the data set. In order to do that:

- Double-click the data set OR Right-click the data set and select “Edit”
- In the “Edit Data Set” window select “Preview Results”

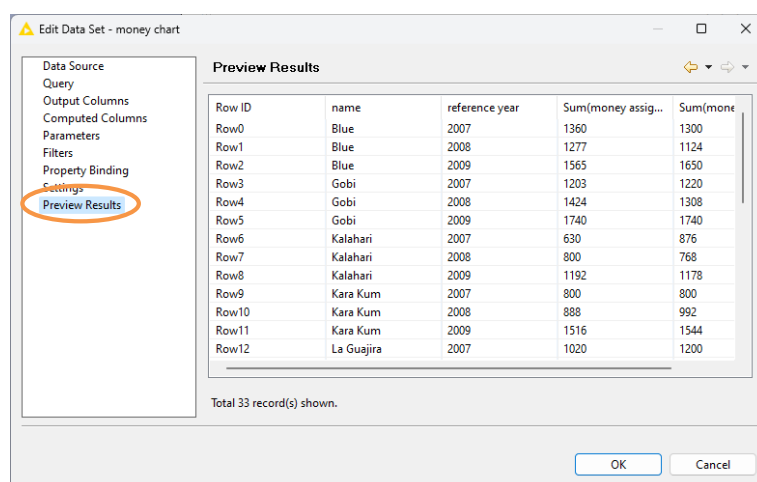


Figure 7.21. “Preview Results” shows the content of the dataset.

The Layout

Let’s now start assembling the report. Click the “Layout” tab to move away from the Master Page editor and back to the Report editor. What we see now is an empty page. First of all, we would like to have a title for our report, something like “Project Report: Money Flow” for example. We are going to place tables, charts, and more explicative labels under the main title.

The Title

To build a title:

- Drag and drop the “Label” report item from the “Report Items” panel in the bottom left corner into the Report editor
- Double click the label and enter the title: “Project Report: Money Flow”
- Select the whole label by clicking its external contour
- In the “Property” editor under the Report editor, go to the tab called “General” and select the properties for your title: font, font size, font style, font color, background color, and so on.

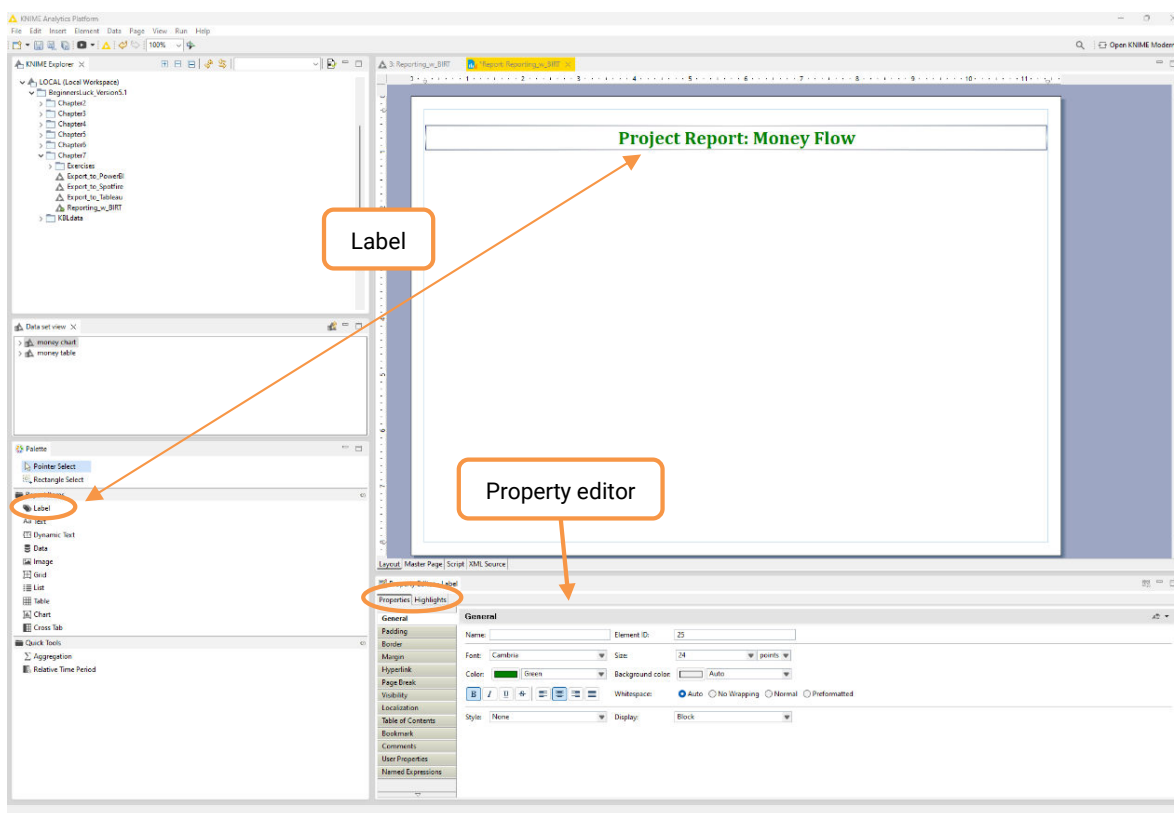


Figure 7.22. Drag and drop a "Label" item into the Report Editor to create the report title.

We chose font "Cambria", color "green", size "24 points", style "bold", and adjustment "centered".

Note. The font size settings consist of 2 parameters: the number and the measure unit (% , cm, in, points, etc.). Be sure to set both of them consistently! If you set the number to 24 and the unit to "%" you will not see your title label anymore and will wonder what happened to it.

I am sure you have noticed that the title label has been automatically placed at the top of the page and that it spans the complete width of the page. You cannot move it around to place it anywhere else nor shrink it to occupy only a part of the page width. This automatic adjustment (full page width and first available spot in the page from the top) will affect all report items that are dragged from the "Report Items" panel and dropped directly into the Report editor. For the title item this is not so bad, since the title usually spans the whole page width and is placed at the page top. It is, however, undesirable for most other report items.

The Grid

In our report we would like to have three tables: two tables at the top describing the amount of money assigned and used for each project each year, and one table in the middle of the page under the two previous tables to show the remaining money. It would also be nice if all tables had the same size; i.e. something less than the half of the page width. Under the tables we would like to place two bar charts side by side to show respectively how the money has been assigned and used. In order to have the freedom to place report items anywhere in the report page and to give them an arbitrary size, we need to place them inside a “Grid”.

A “Grid” is a report item, something like a table that creates cells in the report page with customizable location and size to contain other report items.

For our report, we need:

- one row with two cells: one for the assigned money table and one for the used money table
- one row with only one cell for the remaining money table
- one row with two cells again for the 2 bar charts

We therefore want to create a “Grid” with 3 rows and 2 columns and merge the two cells of the second row into one cell only.

To create the “Grid”:

- Drag and drop the “Grid” report item from the “Report Items List” panel into the Report editor under the title label
- Enter 2 for the number of columns and accept 3 for the number of rows
- Select both cells in the second row by clicking the external left border of the row
- Right-click the two-cells selection
- Select the “Merge cells” option

Note. Sometimes we use over-detailed grids. That means we define grids with more columns and rows than necessary. This gives us more freedom in adjusting distances between report items and other margins.

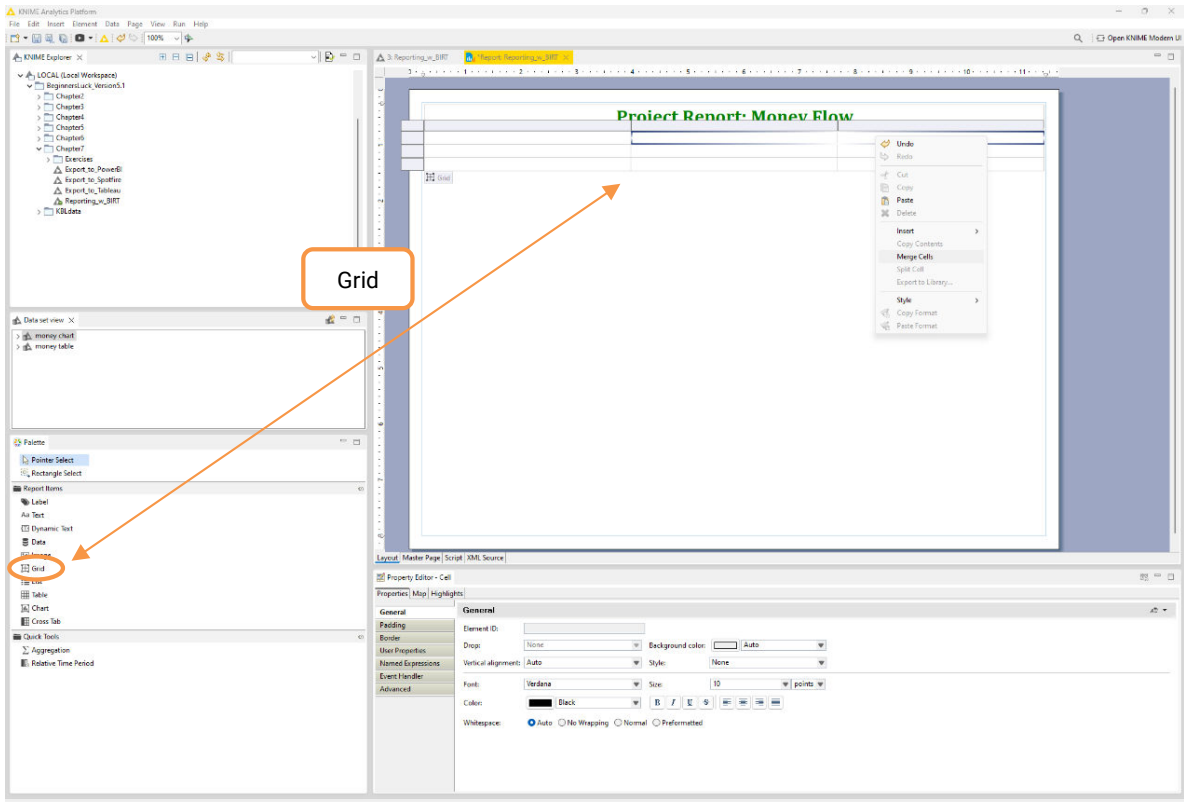


Figure 7.23. Drag and drop the “Grid” report item into the Report editor, select 3 rows and 3 columns, and merge the two cells in the middle row.

The Tables

To create a table we can follow the standard procedure:

- Drag and drop the “Table” report item into the report editor
- Bind the “Table” to a data set
- Bind each data cell to a data set field

OR we can:

- Drag and drop the data set into the report editor

- In the next window, select the data columns you want to appear in the final report

The second method is easier, especially for big tables.

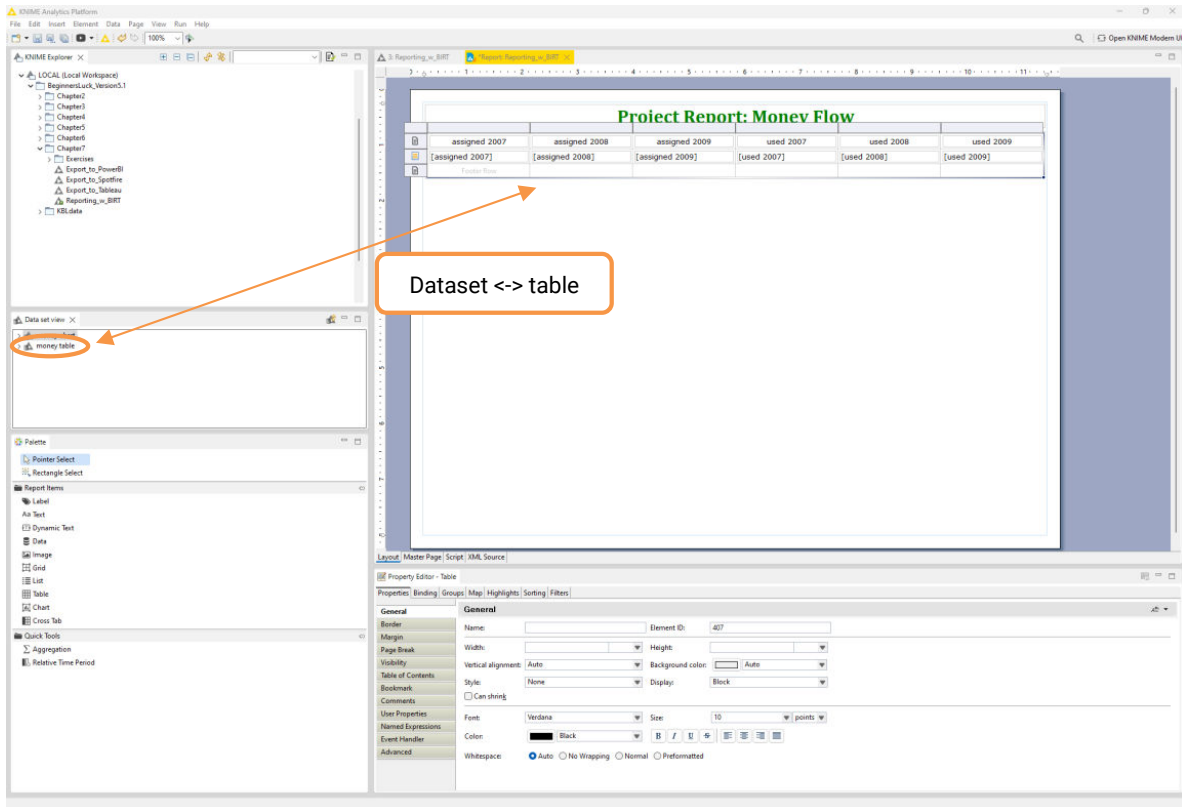


Figure 7.24. Drag and drop a dataset from the "Data Set View" panel to produce a table with as many columns as many dataset's fields.

In the report layout a table is composed of three rows:

- a header row
- a data cell row
- a footer row

The header row and the footer row contain only labels or other static report items and appear in the final report only once at the beginning and end of the table respectively. The data cell row contains the data set fields. In the real report, the data cell row multiplies into as many rows as there are in the data set.

After dragging and dropping the Data Set into the report editor, we see a table with as many columns as there are fields in the data set. The column headers are automatically set as labels

with the data set field's name. The footer row is empty. The data cell row contains the data set fields. Let's now adjust the look of the table.

Remove unwanted Columns

- Select the whole table. If you hover over the left bottom corner of the table with the mouse, a small gray rectangle with the word "Table" appears. To select the whole table, click that rectangle.
- Select the unwanted column. To select a whole column click the gray rectangle above the column's header.
- Right-click the top of the unwanted column
- Select "Delete"

Change Column Header

The header of each column is an editable label

- Double click the header label
- Change the text

Change Column Position

- Select the whole table
- Right-click the top of the column (the gray rectangle) that you want to move
- Select "Cut"
- Select the column to be positioned on the left; do this right-clicking the gray rectangle at the top of the column
- Select "Insert Copied Column"

Change Font Properties

- As for “Labels”, in the “Properties” window (“General” tab) you can change font, font size, alignment, style, etc.

Format Number

- Select a cell containing a number
- In the “Properties” editor, select the “Format Number” tab
- Choose the format for the number in your cell

Define Width and Height

- Select a row or a column
- In the “Properties” window, go to the “General” tab and change the height and width

Set Borders

- Select the item that needs borders (full table, row, or single cell).
- In the “Properties” editor, select the “Border” tab
- Choose the desired border

Note. The property “Border” is not available for columns.

Set Table Size

- Select the whole table
- In the “Properties” window, select the “General” tab
- Choose the desired width and height

Note. For the font, cell, and table size, the height and width can be expressed in different measure units. Verify that the unit you are using is a meaningful one. BIRT performs some kind of automatic adjustment on the width and height of the cells. You must define a suitable height and width for the full table first for the height and width of the single cells to become effective.

We dragged and dropped the “money table” data set into each one of the two cells in the first row and into the only cell in the second row of the “Grid”. The table on the left of the first row will show the assigned money. We then deleted all “*used*” and “*remain*” columns. The table on the right of the first row will show the used money. We then deleted all “*assigned*” and “*remain*” columns. The table in the second row will show the remaining values. Here we deleted all “*used*” and “*assigned*” columns.

In each table, the “RowID” column contains the project name. We therefore changed the header label to “Name”. The data and header cell for the “Name” column were left aligned while the last 3 cells were all right aligned. The tables had a green border running around it and also a green border between the header row and the data row.

The size of the first two tables was set to 80% (= 80% of the grid cell) and the size of the third table, which in a grid cell is double the size of the previous two, was set to 40% (= 40% of the grid cell). The alignment property of the three grid cells was set to “Center”.

In the first table, we then set the font to “Cambria” and font size to “10 points” in both header and data cells. The header’s font style was also set to “bold” and the color to “green”. Finally, the data cells containing numbers were formatted with “Format Number” set to “Fixed” with 2 decimal places and 1000s separator. All these operations should be repeated for the second and the third table as well.

Toggle Breadcrumb

In the top bar you can find the “Toggle Breadcrumb” button.

This button displays the hierarchy of a report item over the layout, for example the hierarchy of the “assigned 2008” data cell as:

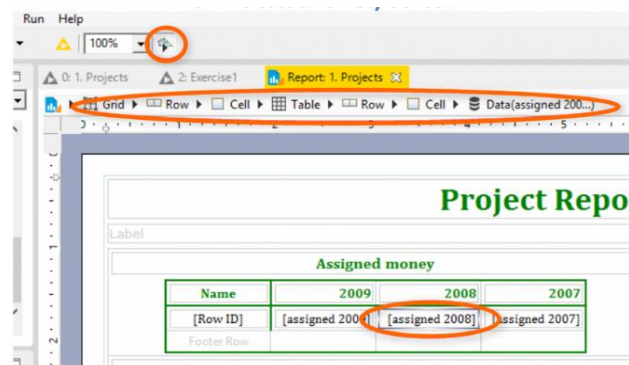


Figure 7.25. Toggle Breadcrumb button.

Grid → Row → Cell → Table → Row → Cell → <data set field name>

Report Preview

Let's now create the report (from top menu "Run" > "View Report" > "In Web Viewer") to have a rough idea of what the report will look like. Probably the "large" font size we have chosen for the data cells and header cells will be too big for the tables to nicely fit into one page. We can easily reduce the font size by setting it to "small" in one or both Style Sheets. This will automatically apply to all those table cells that have been formatted by these Style Sheets. This is one of the big advantages of using Style Sheets.

Let's put a label on top of each table to say what the table represents: "assigned money", "used money", and "remaining money". We can then change the column headers from "<assigned/used/remain> <year>" to just "<year>", for example "assigned 2009" to just "2009" and so on. Let's also add a few empty labels after each table to make the report layout more spacious. If we run a preview now, the report will look similar to the one shown below.

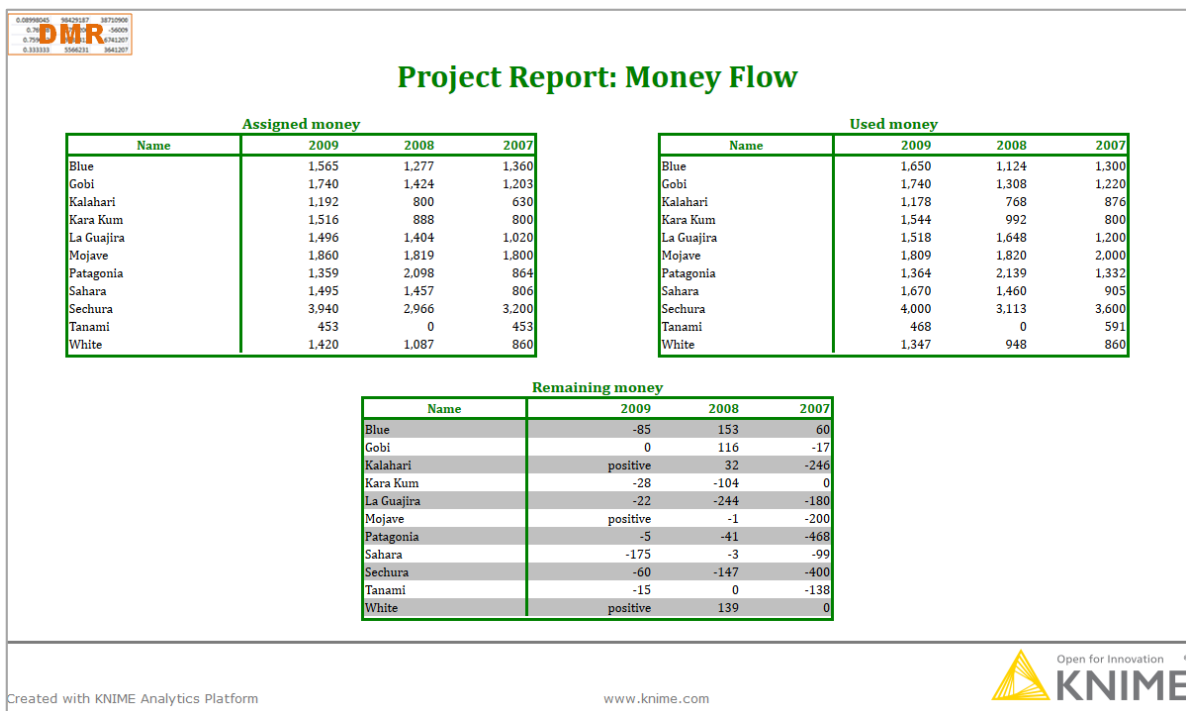


Figure 7.26. Report View on a web browser after creating and formatting the three tables.

Maps

Sometimes, we might want to map numeric values to descriptive values. For example, in a financial report, we can map one column with numeric values as:

Values < 0 to "negative"

Values = 0 to "zero"

Values > 0 to "positive"

The mapping functionality is found in the "Maps" tab in the "Properties" editor of table report items; that is cells, rows, columns, and even the whole table.

- Select the data cell, row, column, or table to which you want to apply your mapping
- Select the "Maps" tab in the "Properties" editor
- Click the "Add" button to add a new mapping rule
- The "New Map Rule" editor opens.
- Build your condition in the "Map Rule Editor", for example:

row["remain 2009"] Greater than 0 → "positive"

- Click "OK"

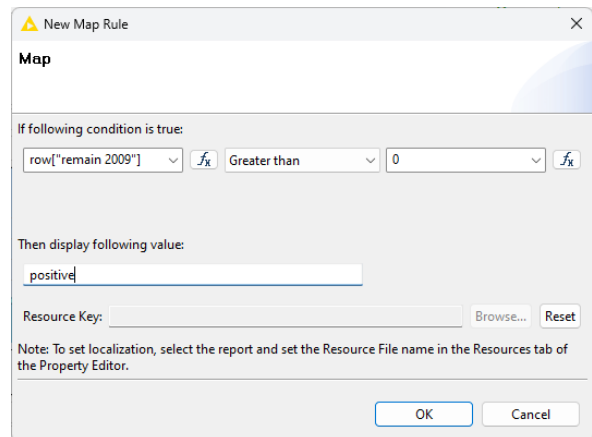


Figure 7.27. The "New Map Rule" editor.

Chapter 7: Reporting in KNIME

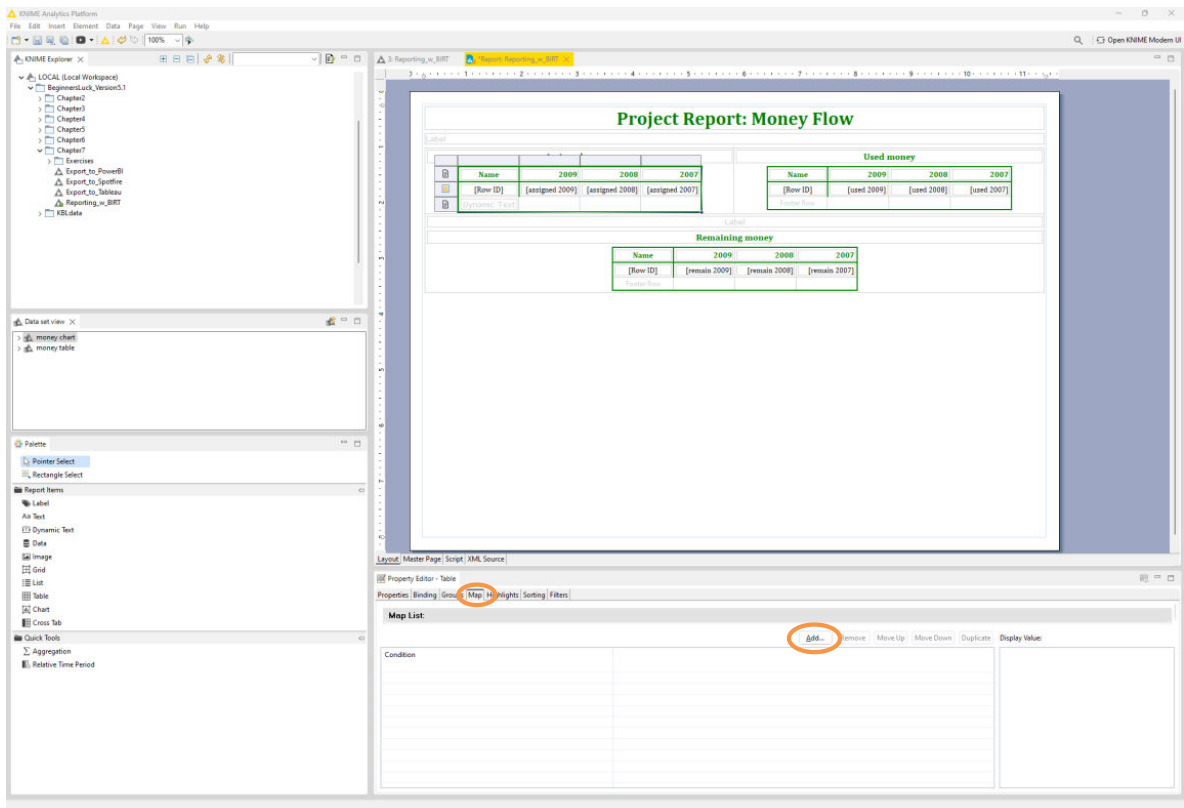


Figure 7.28. The tab "Maps" in the Table Properties Editor defines text mapping for a table item.

Highlights

The "Highlights" property works similarly to the "Maps" property, only that it affects the cell and row layout rather than cell text content.

The "Highlights" property is located in the "Highlights" tab in the Property editor of the "Table" report items: cells, rows, columns, and the whole table.

For example, we would like to mark all the cells with a "remain 2009" value smaller than 0 in red.

- Select the data cell [remain 2009] (or another cell, a row, or a column where the highlighting should occur)
- Click the "Highlights" tab in the Property editor
- Click the "Add" button

The “New Highlight” editor opens. In the “Condition” section:

- Enter the rule for the highlight, for example:

Row[remain 2009] smaller than 0

To build the rule you can also use the “Expression Editor” which is explained later in this chapter.

- In the “Format” section, enter the formatting you want to be applied, when the condition is true.

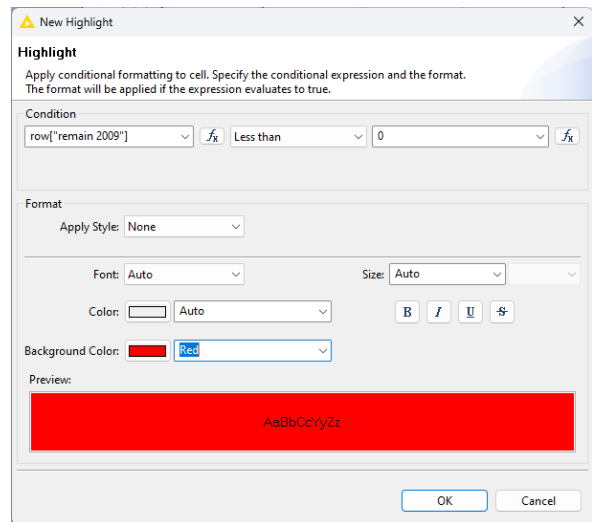


Figure 7.29. The “Highlights Rule Editor”.

- Do this by clicking the button next to “Background Color” and then select red in the color dialog.
- Click “OK”

After closing the Highlight dialog, run a view of the document to see the new red highlighted cells.

Chapter 7: Reporting in KNIME

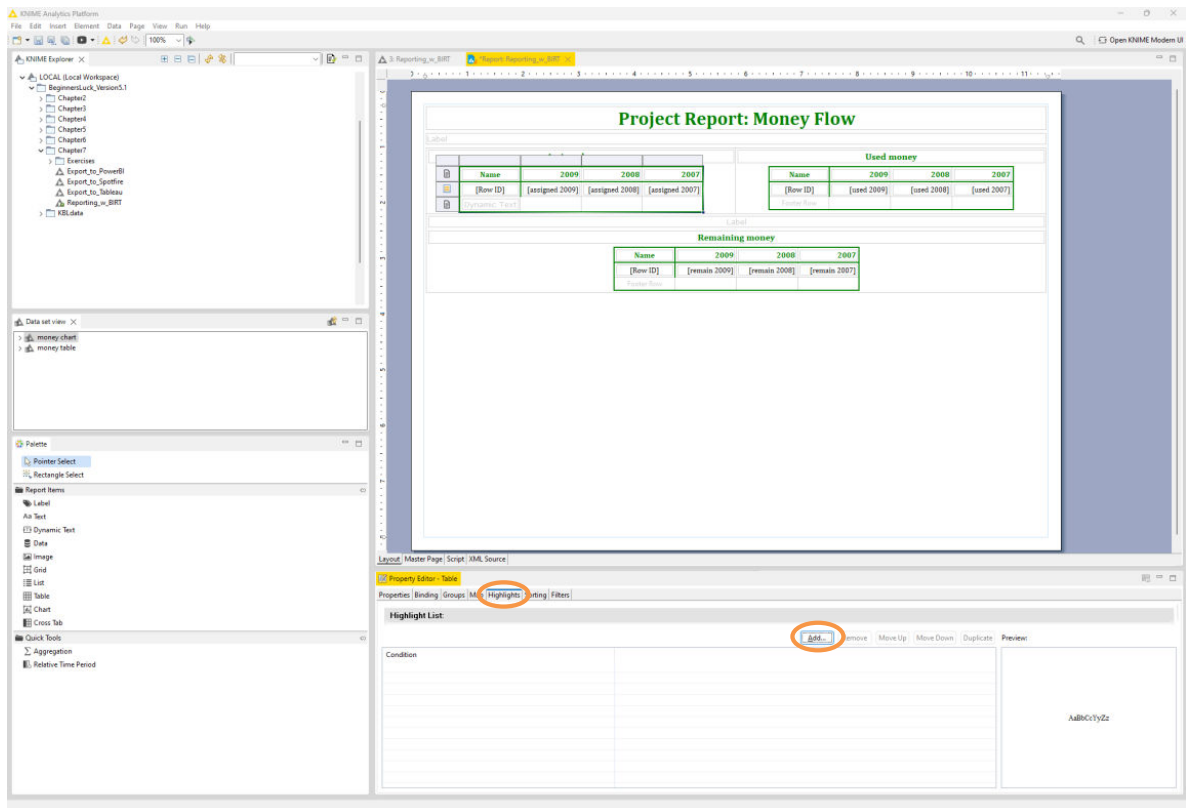


Figure 7.30. The “Highlights” tab in the Properties Editor defines conditional properties for a table item.

The Zebra Style

The zebra style is very popular for tables in reports. This is where the table’s rows have alternating colors. To produce a zebra style table, you need to add the following condition in the “New Highlight” editor:

- Select the whole data row in the table, by selecting the gray rectangle on the left of the table row
- Select the “Highlights” tab in the Property editor
- Select the “Expression Builder” icon. This is the icon with “fx” close to the “Condition” input box
- In the “Expression Builder” dialog, select “Available Column Bindings” and then “Table”
- Double-click “RowNum” in the right column of the “Expression Builder” table
- row.__rownum appears in the “Expression Builder Editor”

- Write “row._rownum % 2” in the “Expression Builder ” dialog and click “OK”
- Select “Equal to” and enter “0” in the “New Highlights” editor
- In the “Format” section, select the background color “gray” or “silver” in the consequent field of the rule
- Click “OK”

Run a preview of the document to see the zebra style table.

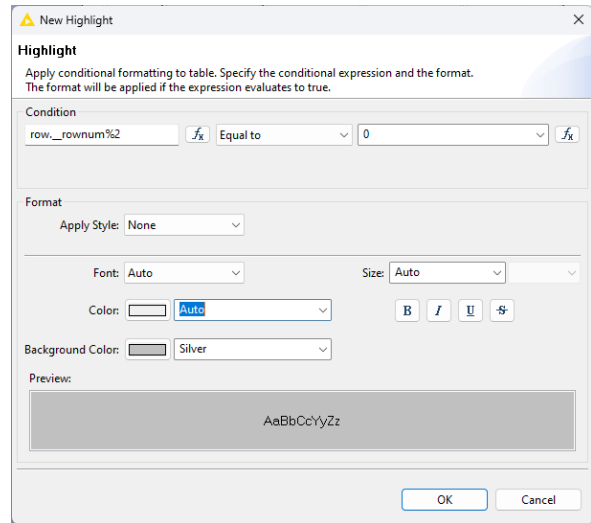


Figure 7.31. The icon to open the “Expression Builder Editor”.

Net money			
Name	2009	2008	2007
Blue	-85.00	153.00	60.00
Gobi	0.00	116.00	-17.00
Kalahari	14.00	32.00	-246.00
Kara Kum	-28.00	-104.00	0.00
La Guajira	-22.00	-244.00	-180.00
Mojave	51.00	-1.00	-200.00
Patagonia	-5.00	-41.00	-468.00
Sahara	-175.00	-3.00	-99.00
Sechura	-60.00	-147.00	-400.00
Tanami	-15.00	0.00	-138.00
White	73.00	139.00	0.00

Figure 7.32. The Zebra style table.

Page Break

We want to export the final report to PowerPoint. This first part of our report fits nicely into a PowerPoint slide. A page break at this point would be very useful to prevent undesired page format effects in the final document.

To insert a page break after a report item:

- Select the report item
- In the Property editor, select the “Page Break” tab
- Set your page break by changing the page break option from “Auto” to “Always”

In the example workflow, the page break was set after the “remaining money” table.

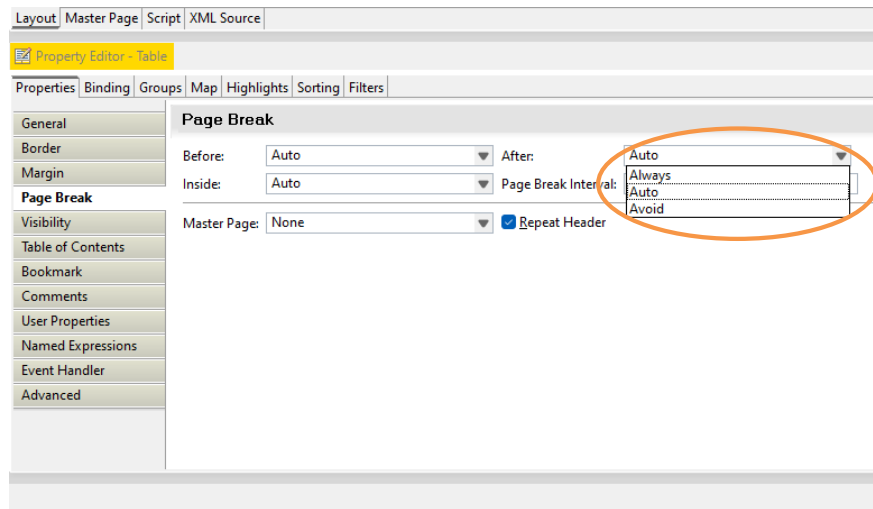


Figure 7.33. "Page Break" tab in the Property editor.

The Charts

The final part of this report consists of two charts to be placed side by side in the last row of the grid. One chart shows assigned money over the years and the other chart shows used money over the years. The two charts should have an identical look.

To create a chart, drag and drop the "Chart" report item from the "Report Item List" into the report editor. After the chart has been dropped, the "Chart Wizard" opens to guide you in setting the right properties for the chart.

The "Chart Wizard" covers three main steps for all types of charts:

- Select the Chart Type
- Select the Data
- Format the Chart



Figure 7.34. "Chart" report item in the "Report Item" panel.

The "Chart Wizard" can be reopened at any moment by double-clicking the chart.

Select Chart Type

The first step of the “Chart Wizard” consists of selecting the chart type. There are many chart types available, and each chart type has a number of chart subtypes. In addition, each chart can be rendered in 2D, in 2D with depth or in full 3D. Flip Axis will change the orientation of the chart. The X-axis will then be vertical and Y-axis horizontal.

- Select your chart type
- Click “Next” to proceed to the next chart wizard’s step.

We chose a “Tube” chart type in a simple 2D dimension.

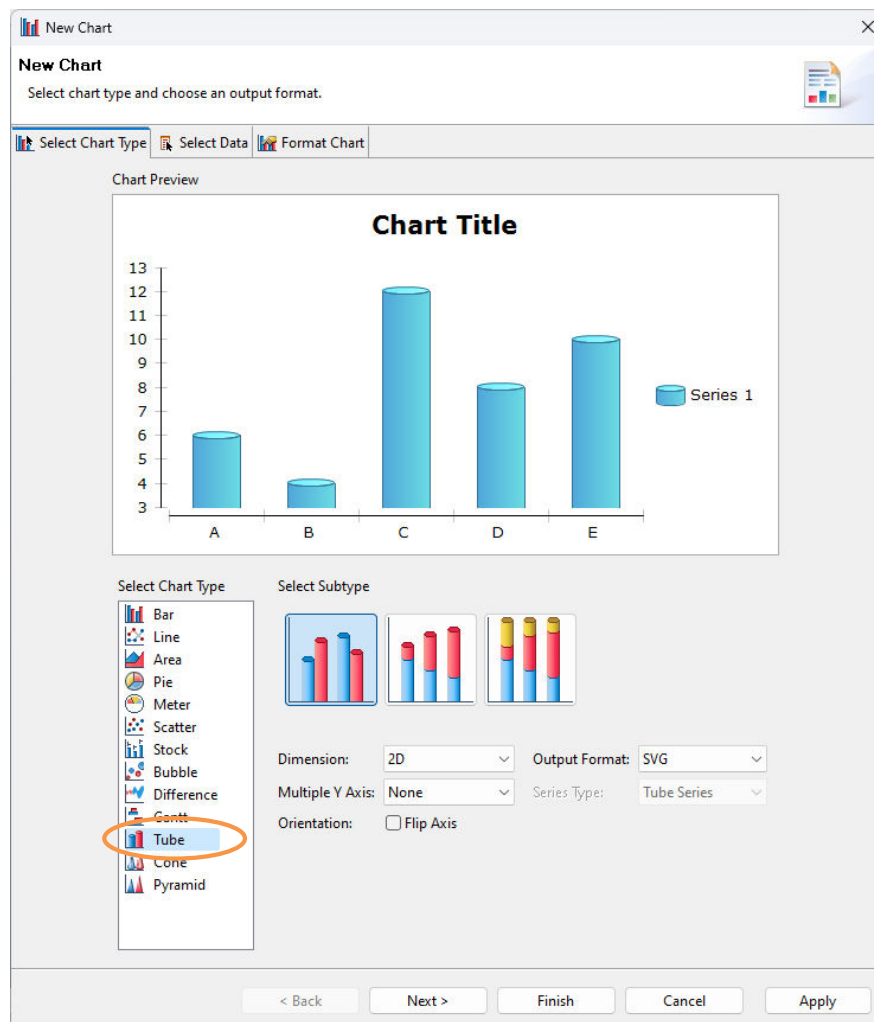


Figure 7.35. First Step of the “Chart Wizard”: Select Chart Type.

Select Data

To connect the chart to a Data Set is the second step.

- Bind the chart with a Data Set with option “Use Data from”.
- In the data preview table select the column data to be on the X-axis or on the Y-axis or to work as group data. Right-click on the column header and select one of those options:
- Use as Category (X) axis
- Plot as Value (Y) series
- Use to group Y-series
- If you need additional Y-series, select “<New Series ...>” in the menu called “Value (Y) Series”.
- Category data are sorted on the X-axis in descending order by default. If you do not want any sorting, click the sorting icon (the one with the down arrow on the side of the “Category (X) Series:” text box) and disable “Grouping”.
- Sometimes not all data rows from the data set need to be shown in a chart. To filter out rows from the data set, click the “Filters” button on the bottom right and add rules to include or exclude rows from the data set (see below).

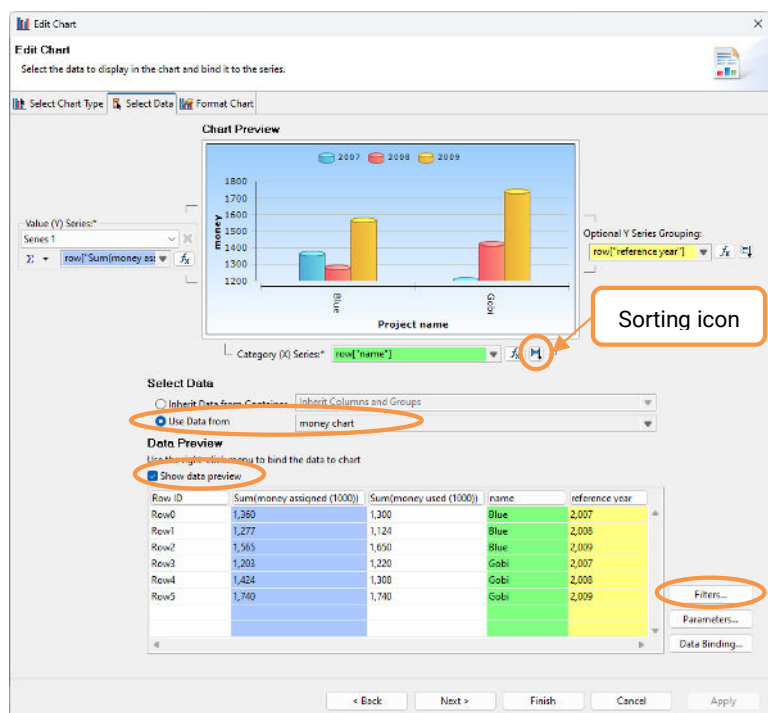


Figure 7.36. Second Step of the “Chart Wizard”: Select Data.

- Click “Next” to move to the next wizard’s step.

To filter rows in the data set:

- Click the “Filters” button
- In the “Filters” window, click the “Add” button

The “New Filter Condition” window appears. Insert your filtering rule in the “New Filter

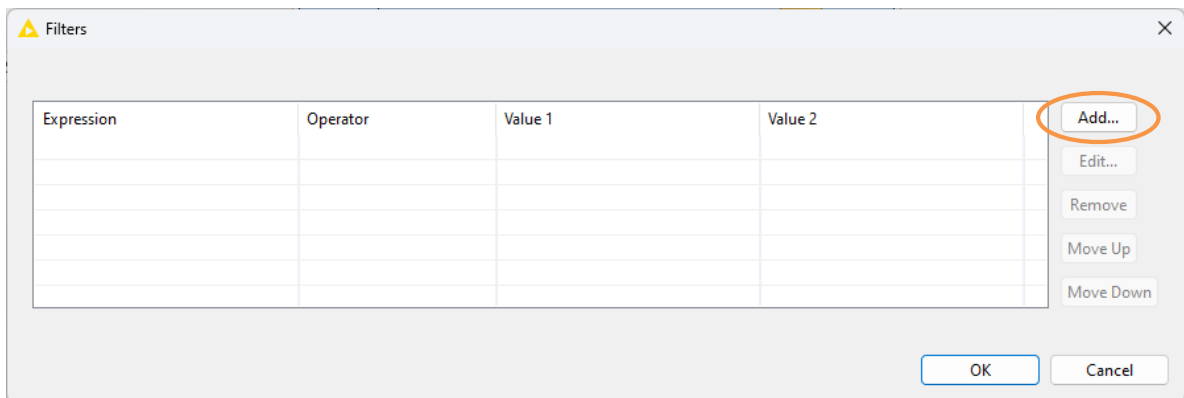


Figure 7.37. The “Filters” window.

Condition” window.

Here on the right is an example of a filtering rule that excludes all rows where column “name” = “total”. Notice that “total” is inside quotation marks. Do not forget the quotation marks in a string comparison, since BIRT needs quotation marks to recognize strings.

The first chart is supposed to show the assigned money over the years. We selected:

- Data set “money chart”
- Column “name” as Category Series (X-axis) unsorted
- Column “Sum(assigned money(1000))” as Y-Series
- Column “reference year” to group the Y-series

We have only represented one Y-series in this chart and no filter was applied to the data set rows.

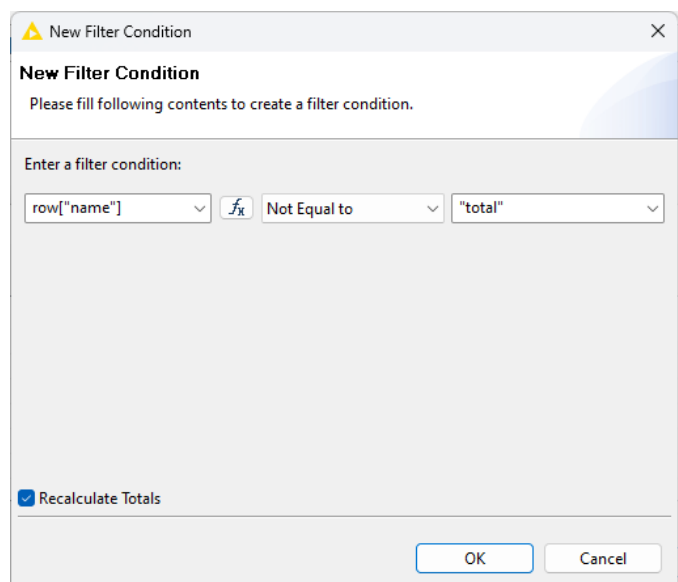


Figure 7.38. The “Filter Condition Editor”.

Format Chart

The last Wizard step guides you through the chart layout configuration. On the left, a tree shows the formatting options for the chart. In **“Series”** you can change the name of the Y-series. The default names are just **“Series 1”**, **“Series 2”**, etc. In **“Value (Y) Series”** you can add and format labels on top of each point of the chart. Under **“Chart Area”** you can define the background color and style. Under **“Axis”**, you can define labels, scale, gridlines and everything else related to the chart axis (X-axis or Y-axis). **“Title”** has options for the title text, layout, and font. **“Plot”** is similar to **“Chart Area”** but refers only to the plotting space. **“Legend”** helps you with the position, layout, font properties and everything else related to the chart legend.

Series

In **“Series”** you can change the name (labeled as **“Title”**) of each Y-series. The default names are just **“Series 1”**, **“Series 2”**, etc.... which are not very meaningful. The Y-series can be hidden by disabling the

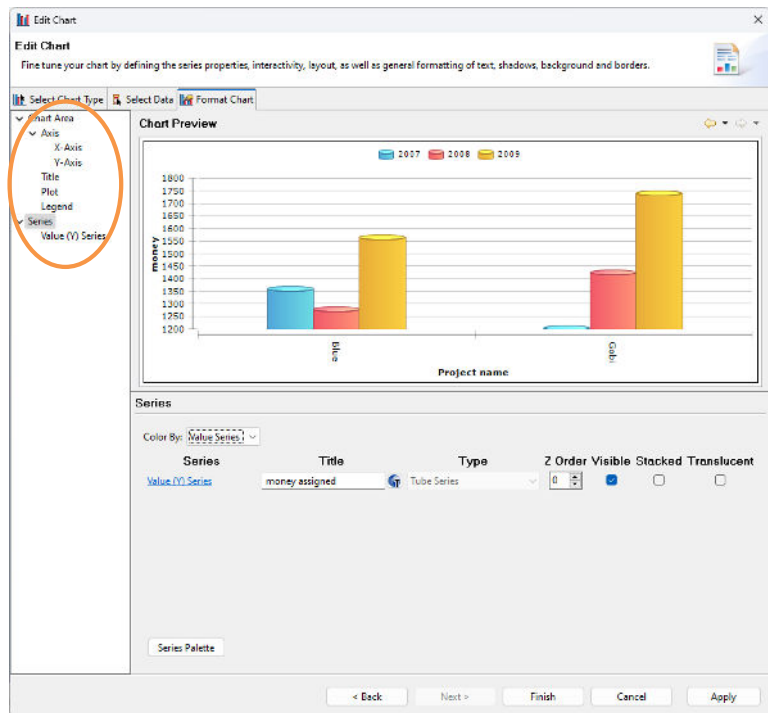


Figure 7.39. Third step of the "Chart Wizard": Format Chart.

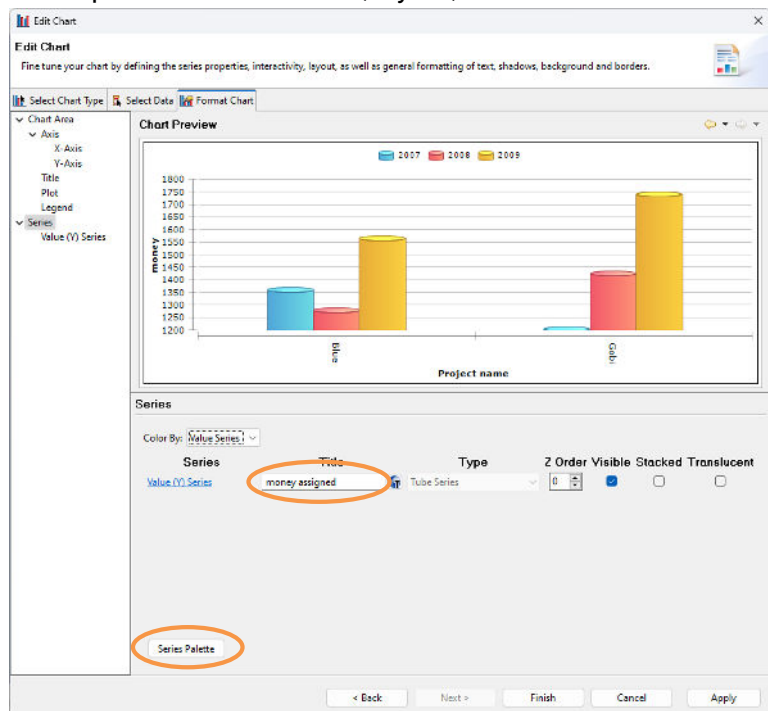


Figure 7.40. Chart Format: Series.

checkbox “Visible” on the right of the “Title” textbox.

The “Series Palette” button leads to a choice of colors for the Y-series. You can select a different color for each one of the Y-series values.

We changed the name of the Y-series from “Series 1” to “money assigned”. This name will appear in the legend. We kept the default series palette.

Value (Y) Series

In “Value (Y) Series” you can add labels on top of each point of the plot, by enabling the option “Show Series Labels”.

The “Labels” button opens the “Series Labels” window to format the series labels.

Series Label window

The “Series Labels” window helps us format the labels on top of each point in the plot, providing we choose to make them visible.

Here you can define the label position, font, background, shadow, outline, and even inset points.

You can also define which values you want shown on top of each point: current Y-value, percent Y-value, X-value, or series name. The label can also be built around the shown value with a prefix, a suffix, and a separator.

The small button with an “A” inside leads to the “Font Editor”.

The “Format” button leads to the “Format Editor” (you need to select an item in the “Values” list to enable this button).

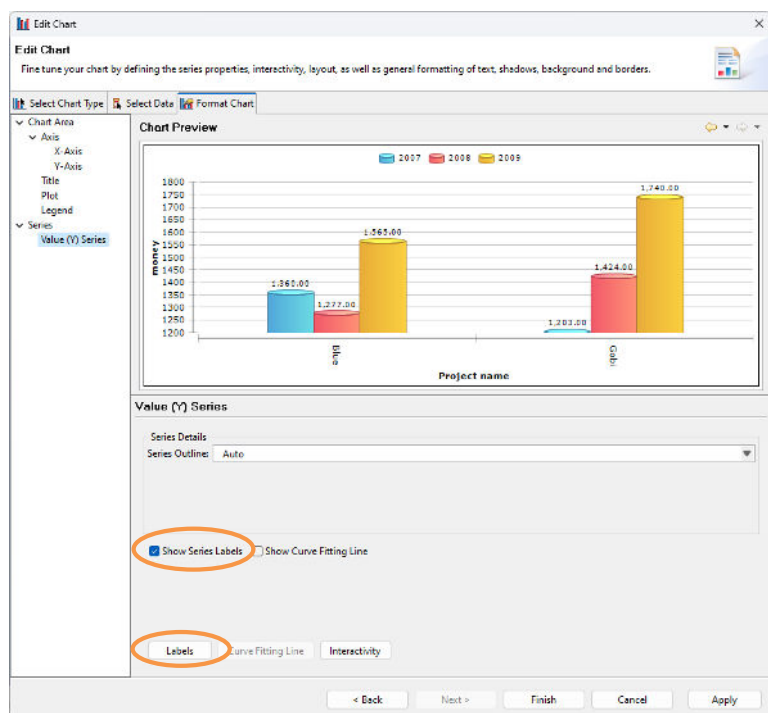


Figure 7.41. Chart Format: Value (Y) Series.

There is no “OK” or “Cancel” button in this “Series Labels” dialog. The new settings are applied immediately. For the “Projects” report we decided to make the series labels visible.

Font Editor

The “Font Editor” is a standard window that you will find in the “Format Chart” step anywhere, where it is possible to change a font format. It contains the usual font format options: font name, size, style, color. A new option is “Rotation”.

“Rotation” rotates the label by the required number of degrees. “0 degrees” (= the red dot in the tachymeter) corresponds to horizontally written labels. “-90 degrees” writes labels vertically from top to bottom. “+90 degrees” writes labels still vertically but from bottom to top. “-45

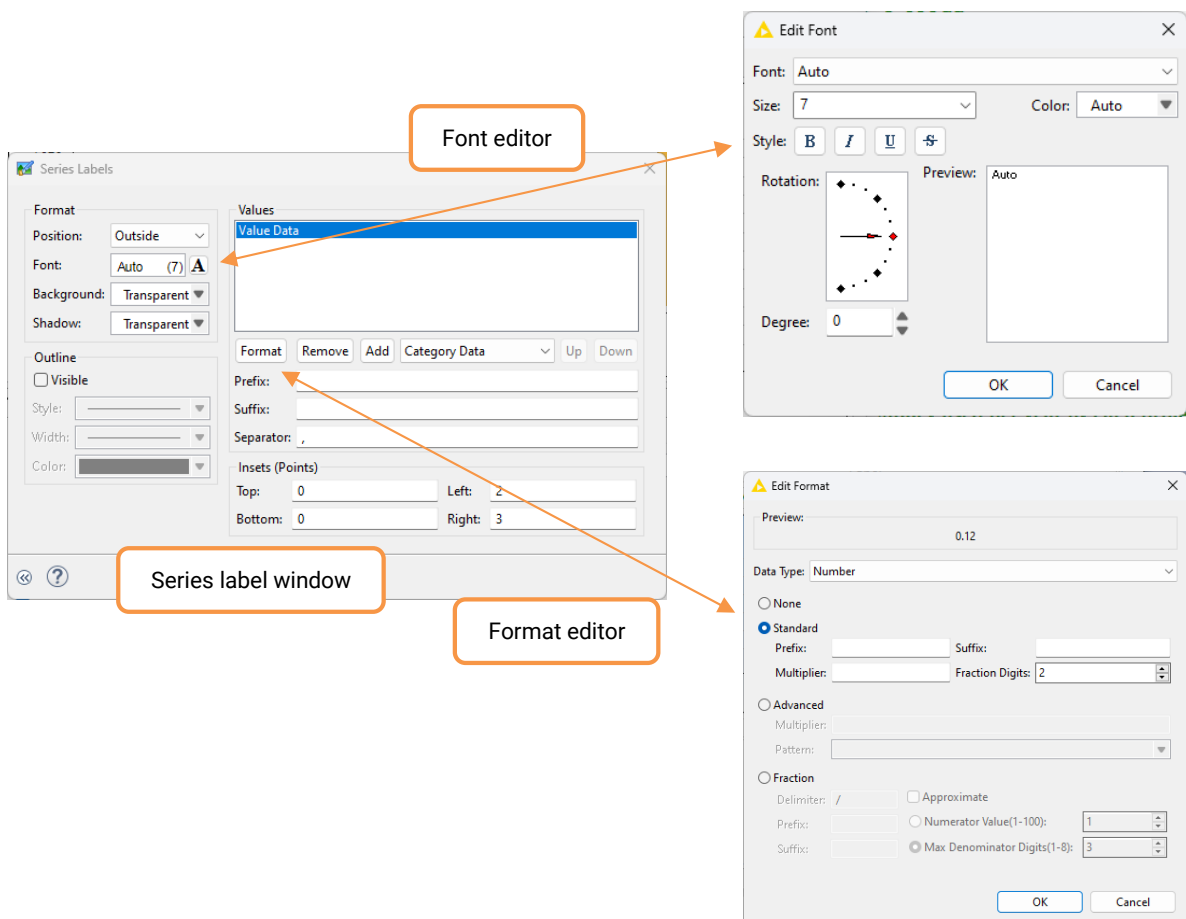


Figure 7.42. The “Series Labels” window: The “A” button opens the “Font Editor”. The “Format” button opens the “Format Editor”.

degrees” writes labels on a 45 degrees pending line from top to bottom. And so on... The option “Rotation” is very useful for crowded charts or for very long labels.

For the charts in the report “Projects” the only setting we made was to specify the series labels font size as 7.

Format Editor

The “Format Editor” is used to format numeric values, dates, and even strings. The most common usage is however to format numbers.

There are 4 possible number formats: none, standard, advanced, fraction. A multiplier is used to represent numbers with smaller strings, for example money in million units rather than in real currency. The fraction digits are the digits after the comma. Prefix and suffix are also available to format strings and are used to build a label around the basic value.

In our chart we formatted the series labels on “Value data” (that is the data of the series) using 2 decimal digits.

Chart Area

In the “Chart Area” you can define the background color and style of the chart.

If you click the “Background” menu, you are shown a number of options you can use to set the background:

- A simple color
- “Transparent” which means no background color
- A gradient between two colors
- A custom color

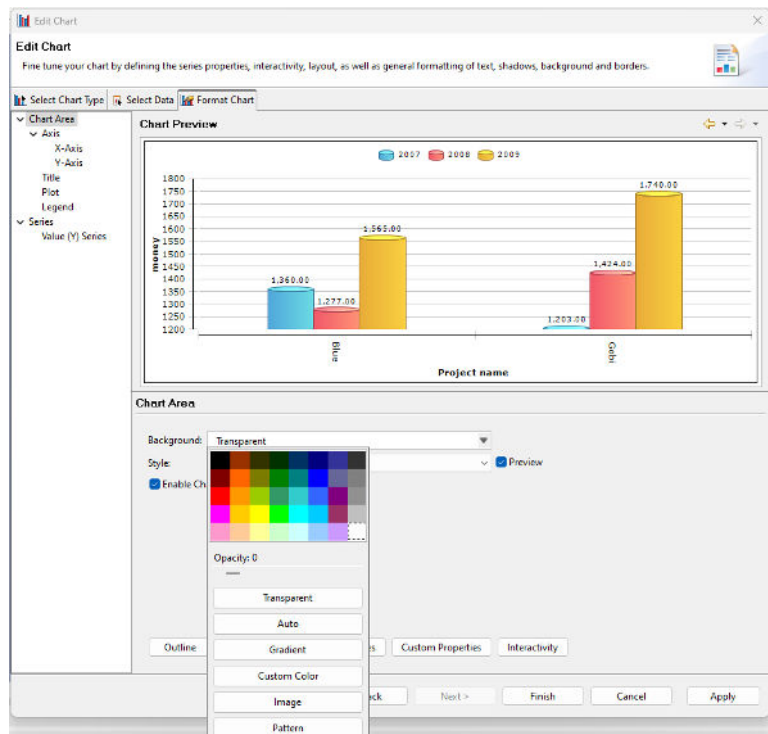


Figure 7.43. Chart Format: Chart Area.

- An image

We selected the “Gradient” option. The “Gradient Editor” needs the start and end color and the gradient direction expressed in degrees. Finally, we made the chart outline visible by clicking the “Outline” button and enabling the option “Visible” in the “Outline Editor”.

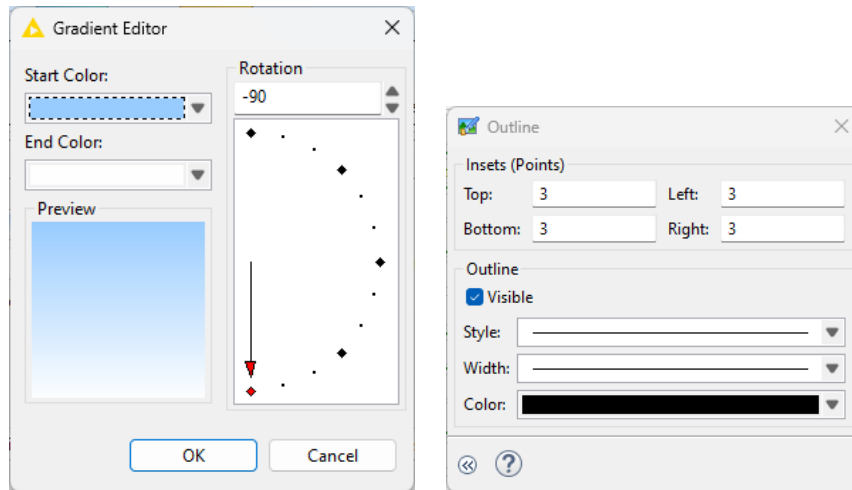


Figure 7.44. The “Gradient Editor” and the “Outline Editor”.

Axis

Under “Axis”, you can define the type and color of both X-axis and Y-axis. There are a few axis types available depending on the value types displayed on the axis (Text, Number, or Datetime). Linear and logarithmic axes apply only to numerical values.

Let’s leave the default linear scale for the value(Y) axis.

All other axis settings, like fonts, gridlines, and scale can be defined for each axis separately. The two windows

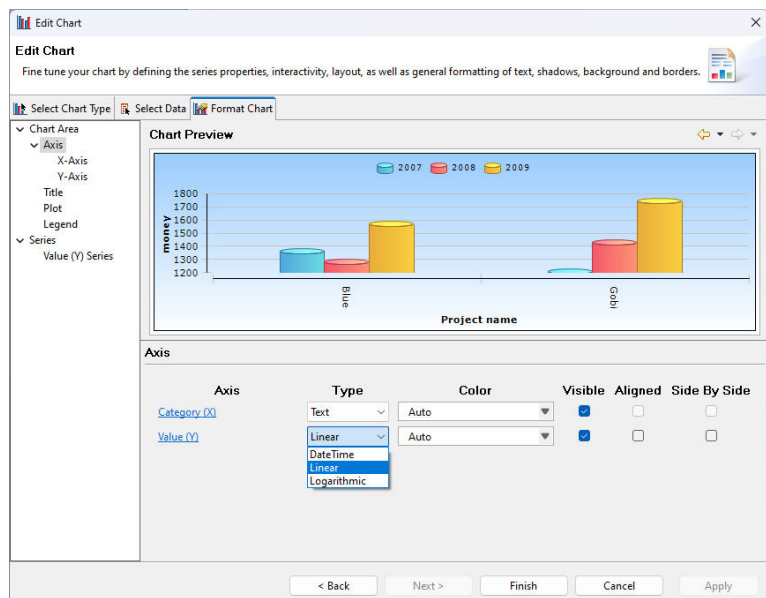


Figure 7.45. Chart Format: Axis.

for X-axis settings and Y-axis settings are almost identical, besides the two category options in the X-axis frame.

X-Axis / Y-Axis

Here the user can set an appropriate title and make it visible.

The most important part is to define the axis labels: format, font, and layout. The usual “A” button leads the user to the “Font Editor”.

The button with the Format icon leads to the “Format Editor”.

The “Label” button leads to the “Label Layout Editor”, where we can define the label position, background, outline, etc.

The “Scale” button defines the step size for numerical values on the axis. It is disabled for text values.

The “Title” button defines font and layout of the axis title if the checkbox was enabled to make the title visible.

The “Markers” button introduces lines to mark areas of the plot.

The “Interactivity” button opens the “Axis Interactivity” window where you can set an action to follow an event. This is used for dashboards or html reports. For example a mouse-click can start a Java script. Many events, such as the mouse-click, and many actions, such as a hyperlink or a script, are available.

The “Gridlines” button opens the “Axis Gridlines” window to enable gridlines for this axis; that is horizontal gridlines for the Y-axis and vertical gridlines for the X-axis.

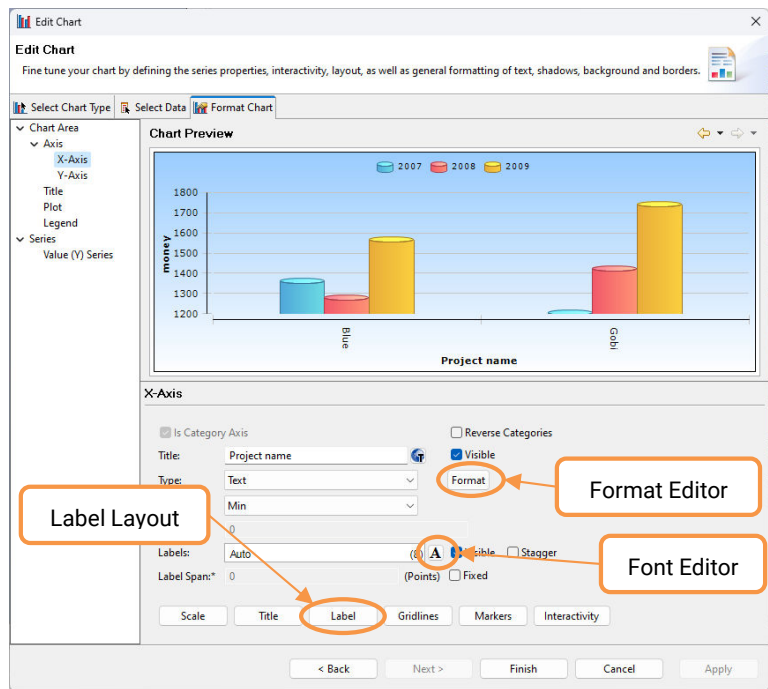


Figure 7.46. Chart Format: X-Axis.

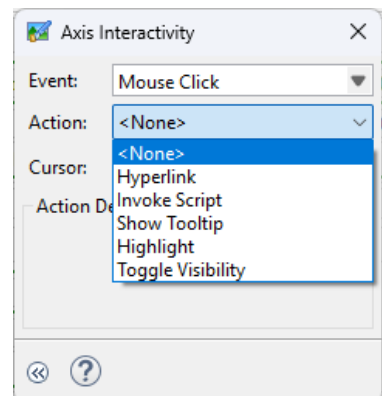


Figure 7.47. The “Axis Interactivity” window.

There are major and minor grids on the plot as well as ticks on the axis.

For the “Projects” report we enabled the following:

- Gridlines on the Y-axis, major grid and major grid ticks only. We overlooked the minor grid not to make the chart too crowded.
- Labels with font size 7 and rotated to -90 degrees on the X-axis
- Title visible on both axis with “Project name” as text for the X-axis and “money” for the Y-axis, font size is set to 8 and rotated to -90 degrees on the Y-axis

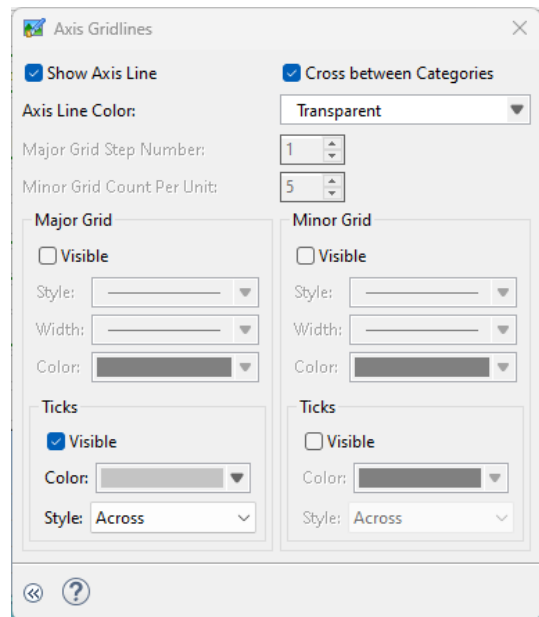


Figure 7.48. The “Axis Gridline” window.

Title

“Title” sets a title in the chart. If you enable the title to be visible, the “Title” frame has options for the title layout, font, and interactivity. I usually do not set the title to be visible, because it takes space from the chart. I use a label on top of the chart in the report layout to act as the chart title.

In the “Projects” report we have disabled the title.

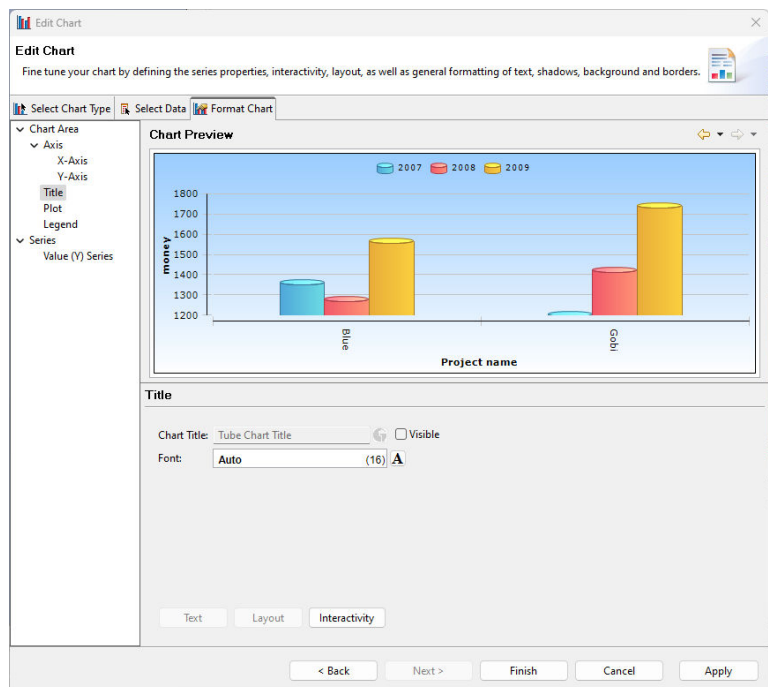


Figure 7.49. Chart Format: Title.

Plot

“Plot” is similar to “Chart Area” but refers only to the plotting space.

Legend

“Legend” helps you with the position, the layout, the font properties and everything else related to the chart legend.

If you decide to include a legend in the chart, first of all you need to make the legend visible in the legend frame (“Visible” checkbox at the very beginning of the “Legend” frame).

After that, you need to define the legend layout (“Layout” button) and font properties (“Entries” button).

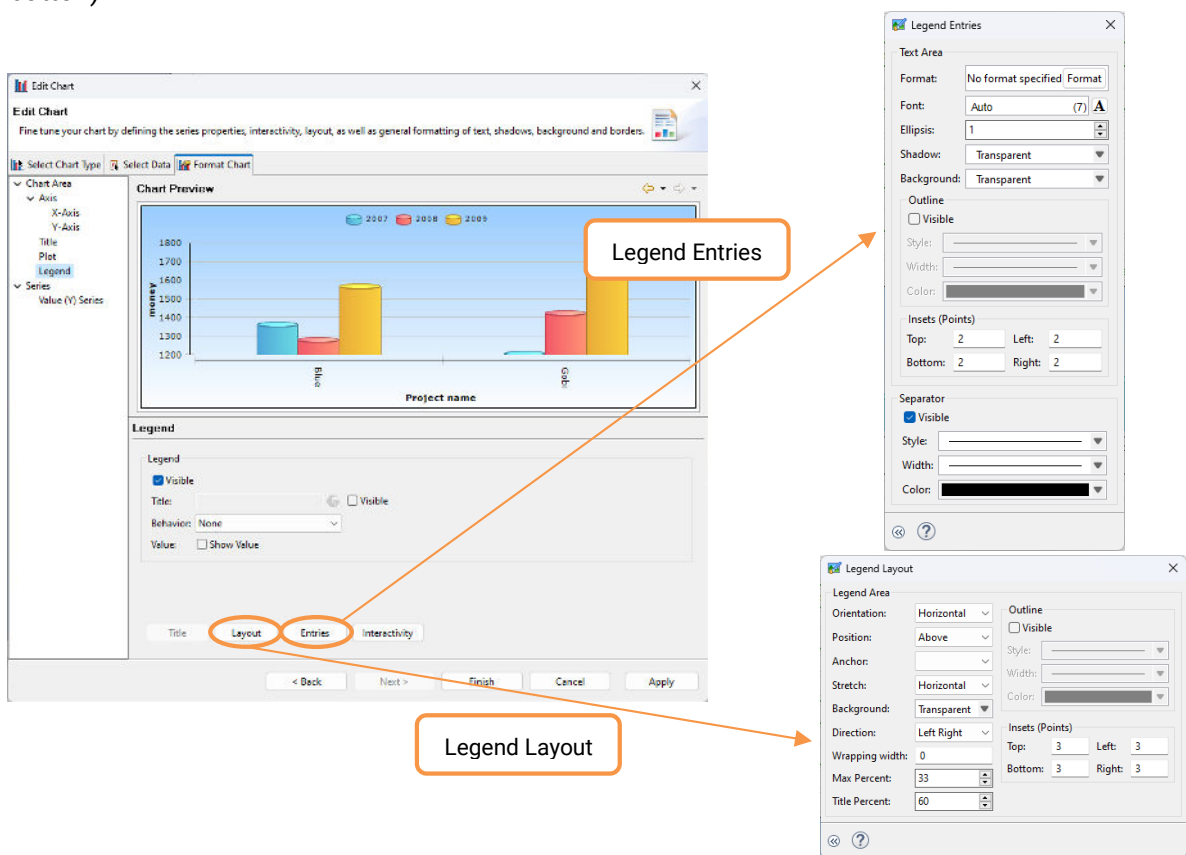


Figure 7.50. Chart Format: Legend. Button “Layout” leads to the “Legend Layout” window. Button “Entries” leads to the “Legend Entries” window.

In the “Projects” report we set the following properties for the legend:

- Font size: 7
- Orientation: horizontal
- Direction: left to right

- Position: above

When you are finished formatting the chart, click “Finish”. The chart wizard takes you back to the report.

Resize the chart to fit the grid cell. Insert a label above the chart to make the chart title, for example where the text is “money assigned per year to each project”.

Change a format property

Run a preview of the document. If you do not like what the chart looks like, just go back to the “Layout” tab, double-click the chart and change the settings that you did not like. In the “Projects” report, for example, the “Series Labels” look a bit too crowded. To disable the “Series Labels”:

- Double-click the chart
- At the top, select the “Format Chart” tab
- Select “Value (Y) Series”
- Disable the “Show Series Labels” checkbox
- Click the “Finish” button

Change data assignment

We need to create an identical chart on the right cell of the grid, but with reference to the money used instead of the money assigned.

- Copy and paste the chart and its title label from the cell on the left to the cell on the right
- Double-click the chart on the right
- Select the “Select Data” tab
- In “Chart Preview”, right-click the header of column “Sum(money used (1000))”
- Select “Plot as Value Y Series”
- Click the “Finish” button

Style Sheets

Sometimes it can be tedious to format all single elements of a report item, especially if many of these report items have to be formatted with the same style. For example, in the previous section we were supposed to format all data cells and header cells of three tables in the same way. To avoid having to repeat such tedious operations, we can use the style sheets.

Style sheets are widely used in web programming to share style specifications across the many elements of web pages. Similarly, the KNIME reporting tool supports style sheets which can be used to apply style attributes to multiple report items.

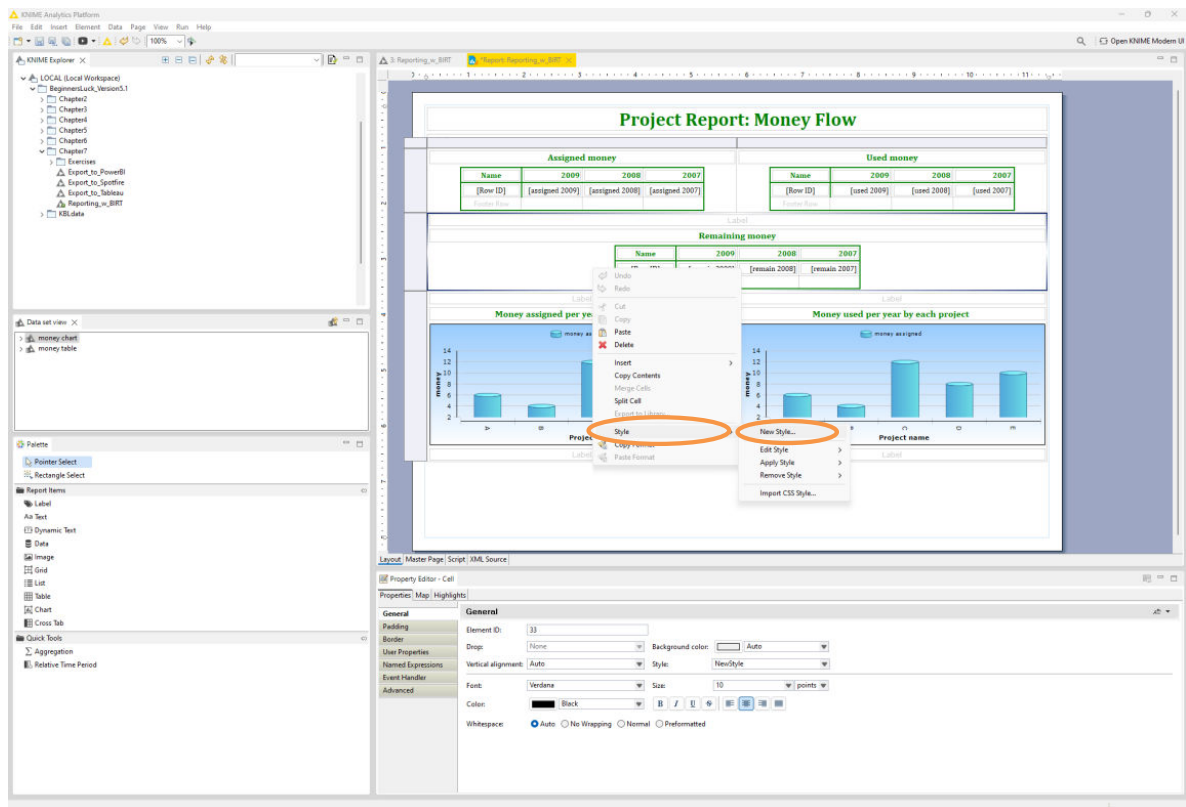


Figure 7.51. Create a new Style Sheet.

Create a New Style Sheet

- Right-click anywhere on the report editor
- Select “Style”
- Select “New Style”

The “New Style” window opens.

In the “New Style” window, you need to define:

- The name of the style sheet in the “General” tab
- The font properties in the “Font” tab
- The number properties in the “Format Number” tab

And so on with more properties in other tabs

Taking the tables in the previous section as an example, it is easy to see that there are two groups of cells for each table:

- Header cells with font “Cambria”, font size “10 points”, font style “bold”, and font color “green”
- Data cells with font “Cambria”, font size “10 points”, and number format with 2 decimal places and 1000s separator

We then built two style sheets, one for the data cells and one for the header cells with the properties listed above. We chose “large” font size for both Style Sheets, named them “data cell” and “header cell” and applied them to each header cell and each data cell of the three tables.

Note. Not all font sizes are available in the Style Sheet editor a

Apply a Style Sheet

- Right-click the report item (table cell, label, etc...)
- Select “Style”
- Select “Apply Style”
- Select the name of the Style Sheet you want to apply

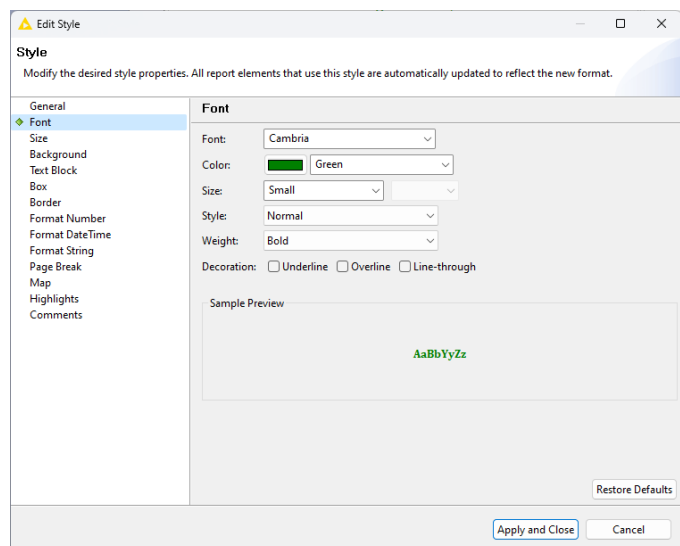


Figure 7.52. The “Edit Style” window.

Chapter 7: Reporting in KNIME

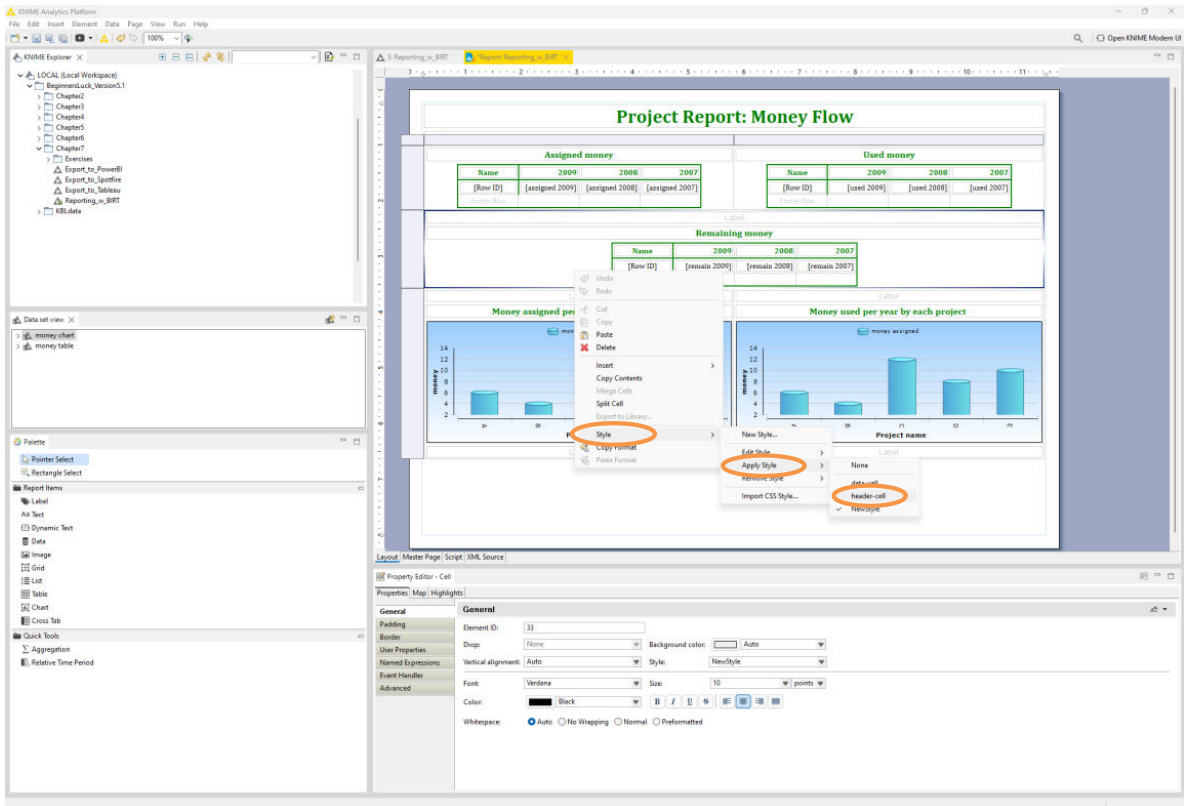


Figure 7.53. Apply a Style Sheet to a report item, for example, a data cell.

Generate the Final Document

In order to generate the final document, go to the Top Menu:

- Select “Run”
- Select “View Report”
- Select the format for your report, for example “PPT” for Powerpoint
- BIRT generates your document in the desired format.

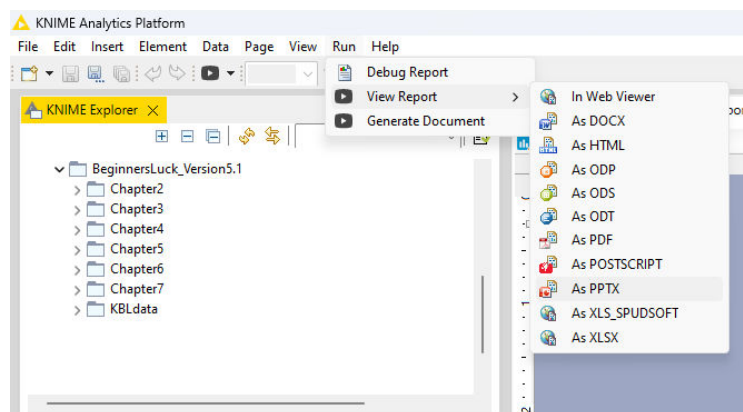


Figure 7.54. Generate the final document.

Alternatively, “Run” → “Generate Document” directly generates the file in the preferred format.

Dynamic Text

A dynamic report item is the “Dynamic Text”, which can be found in the “Report Items” list in the bottom left panel. The “Dynamic Text” item displays a small text, built with the “Expression Builder”. The “Expression Builder” window offers the list of operators and BIRT and Java Script functions. If the dynamic text is included in a table, a few additional options appear in the Expression Builder, like for example the “Available data Sets”.

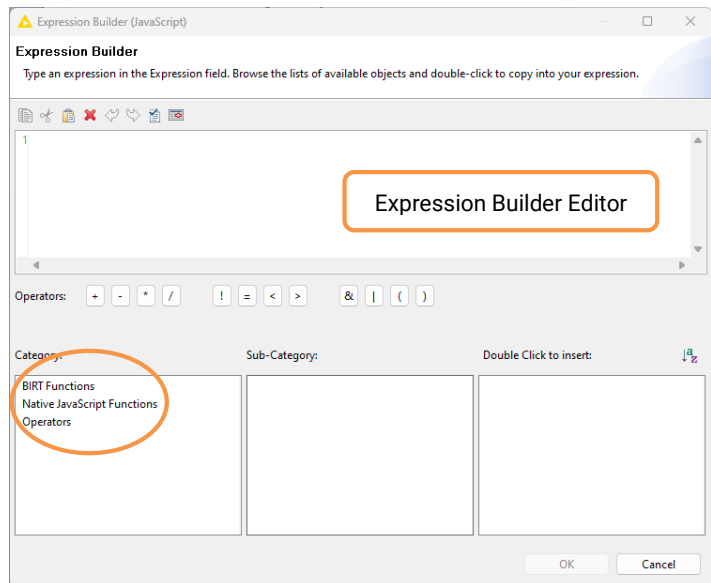


Figure 7.55. The “Expression Builder” window.

The “**Operators**” category offers a number of mathematical/logical operators. An extract with the most frequently used operators is shown in the middle panel.

The “**BIRT Functions**” category includes a number of BIRT specifically designed functions in the fields of finance, date/time manipulation, duration, mathematics, string manipulation, and string or number comparison.

The “**Native JavaScript Functions**” category includes a number of JavaScript functions. These turn out to be particularly useful when the report is created using the HTML format.

Double-clicking an item in a sub-panel, such as a column name, a BIRT function, an operator, or a Java Script function, automatically inserts the item into the Expression Builder editor above.

The “BIRT Functions” and “JavaScript Functions” categories host a few useful functions, for example the date and time functions and the mathematical functions. We could insert the current date as a running title in the report, to the right of the logo.

In the reporting environment, in the “Master Page” tab, in the top header frame, we inserted a grid with two columns and one row. The left cell already included the logo. In the right cell we wanted to insert a “Dynamic Text” item displaying the current date at report creation. Let’s drag and drop a “Dynamic Text” item from the “Report Items” list panel on the bottom left to the right grid cell in the top header frame of the “Master Page” tab. The “Expression Builder” window opens. In order to display the current date, we have many options.

We can choose for example a straightforward “BIRT Functions” → “BirtDateTime” → “Today()” function. “Today()” returns a timestamp date which is midnight of the current date.

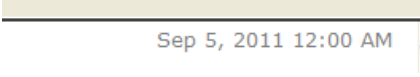


Figure 7.56. Current Date from “Today()” BIRT Function.

The “Today()” function offers no formatting options. If we want to have the current date in a customized format we must build that ourselves. “BIRT Functions” → “BirtDateTime” offers a number of functions to extract components from a DateTime object, like day(DateTime), month(DateTime), year(DateTime) and so on. We could extract the date components and combine them with a BirtStr.concat() function to get the desired date format. After extracting date parts from the result of the Today() function and combining them with a concat() function, we get, for example, the following formula in the “Expression Builder” window:

```
BirtStr.concat(      "Report created on: ",
BirtDateTime.month(BirtDateTime.today(), 2), " ",
BirtDateTime.day(BirtDateTime.today()), " ",
BirtDateTime.year(BirtDateTime.today()))
```

From that we get the current date format in the report running title as shown in the figure on the right.



Figure 7.57. Customized Current Date from “Today()” BIRT Function.

The “BIRT Functions” → “BirtDateTime” sub-category also offers a number of functions to add and subtract time from a DateTime object. For example, the running title could use the following formula with the addQuarter() function:

```
BirtStr.concat("Report valid from: ",
BirtDateTime.today(),
" to: ",
BirtDateTime.addQuarter(BirtDateTime.today(),1)
)
```

which produces a date on the running title as follows.



Figure 7.58. Use of the “addQuarter()” function in the Running Title.

Note. The change in the date locale is due to the introduction of the concat() function, which automatically sets the locale for the DateTime values as well.

Note. In the Expression Builder editor text strings must be typed in quotation marks (Java-style). Text items in a concat() function have to be separated by a comma.

In this section we have only shown the BIRT functions related to the DateTime object, because they are the most commonly used. However, the “BIRT Functions” category offers many built-in functions for mathematical expressions, finance quantities, string manipulation, etc ...

If the report output document is HTML, we could also take advantage of the built-in JavaScript functions, which are more articulated and varied than the built-in BIRT functions.

7.3. Reporting with Other Tools

Similar to using BIRT to build your report, you can also use any other reporting tool available out there. Most of these tools require a paid license and therefore they will not be described

Workflow: Export_to_Tableau

In this workflow we prepare data to be displayed in a report.

Starting from the accounting of a list of projects, on one branch we calculate the money in budget for each year for each one of the project and on the other branch we calculate the money actually spent for each project each year.

We then export the money allocated, the money spent and the money remaining for each project each year to be displayed in a Tableau report.

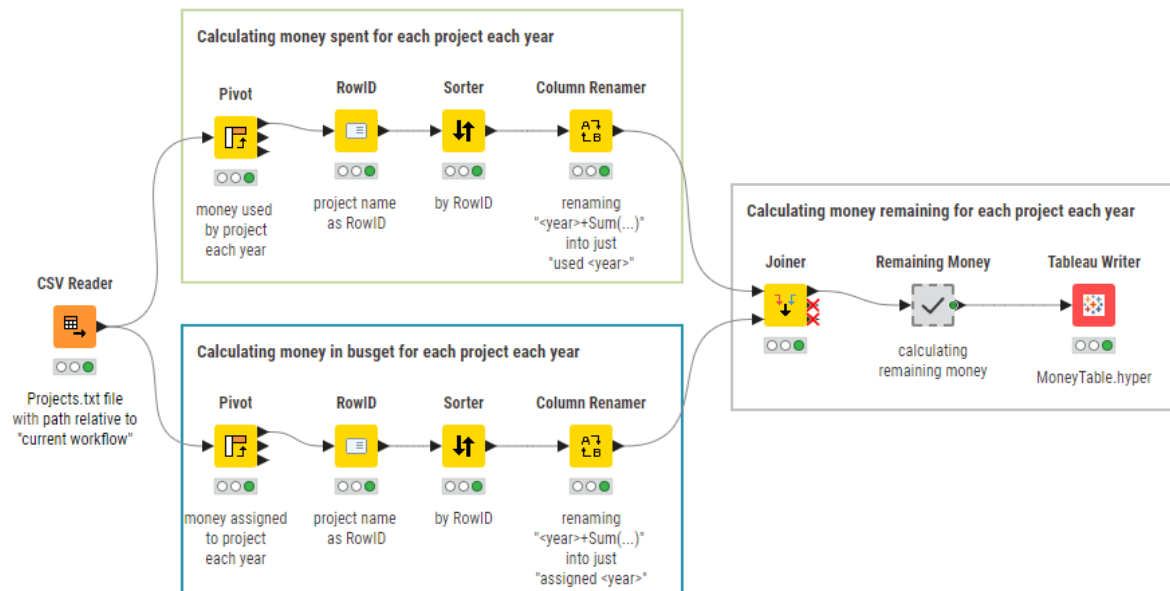


Figure 7.59. The “Export_to_Tableau” workflow exports the data into a Tableau formatted file to be later imported into the Tableau reporting platform.

here in detail. In the material that you have downloaded, you will find in folder Chapter7 a few workflows showing the KNIME nodes dedicated to export the data into the reporting tool of choice.

Workflow “Export_to_Tableau” includes the node “Tableau Writer” to write the data into a Tableau formatted file to be later imported in the Tableau platform to build the report. Another node named “Send to Tableau Server” allows for the direct transfer of data from KNIME Analytics Platform into Tableau.

Workflow “Export_to_PowerBI” uses the node “Send to Power BI”, after previous Microsoft authentication, to transfer the data directly into a PowerBI server.

7.4. Exercises

Exercise 1

The exercises for this chapter follow on from the exercises in Chapter 5. In particular, they require shaping a report layout for the data sets built in Chapter 5 exercises.

Using the workflow built in Chapter 5\Exercise 1, build a BIRT report with:

- A title “income by work class”
- A table on the left side like:

Work Class	Income <= 50K	Income > 50K
[work class]	[nr <= 50K]	[nr > 50K]

- A bar chart with:
 - Work class on the X-axis
 - “Income <= 50K” and “Income > 50K” on the Y-axis
 - Background gradient style
 - Font size 7 on the axis
 - Font size 8 in the legend
 - Legend placed above the plot and running horizontally

- No title
- No axis titles
- Export as Word document

Solution to Exercise 1

Income by work class

work class	Income <=50K	Income >50K
?	1645	191
Federal-gov	589	371
Local-gov	1476	617
Never-worked	7	
Private	17733	4963
Self-emp-inc	494	622
Self-emp-not-inc	1817	724
State-gov	945	353
Without-pay	14	

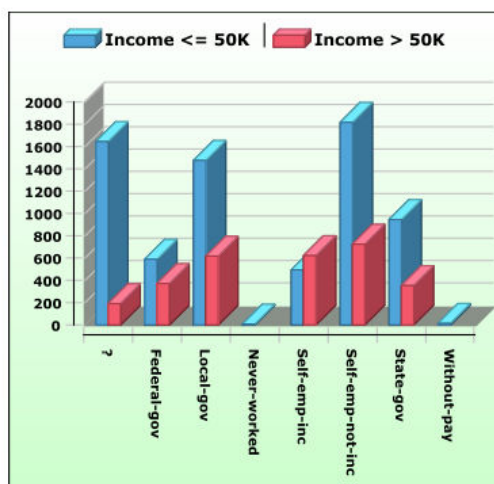


Figure 7.60. Exercise 1: The Final Report.

Node & Topic Index

A

Accuracy	142
Accuracy Measures	140
Aggregations	91
Annotations.....	23
Artificial Neural Network	155

B

Bar Chart	109, 110
Binning	91
BIRT Functions	268
Books	4

C

Case Converter	75
Cell Splitter.....	70
Cell Splitter by Position	70
Chart.....	252
Chart Format Axis.....	260
Chart Format Chart.....	256
Chart Format Chart Area	259
Chart Format Font Editor.....	258
Chart Format Format Editor	259
Chart Format Legend.....	263
Chart Format Plot	262
Chart Format Title.....	262
Chart Select Chart Type.....	253
Chart Select Data.....	254
Cluster Assigner	170
Clustering.....	168
Cohen's Kappa.....	142
Color Manager	104

Column.....	45, 64, 128, 184
Column Combiner.....	77
Column Filter.....	45, 46
Column Rename	64
Column Resorter	78
Combine.....	123
Comments	35
Community.....	2
Concatenate.....	126
Configure.....	34
Confusion Matrix	139
Connector.....	84, 85, 86
Courses	3
Create Metanode	200
CSV Reader	36, 37
CSV Writer.....	53, 54
Column Properties.....	38

D

Data	25, 43, 44
Data Models.....	133
Data to Report.....	232
Data Visualization.....	99
Database	82
Database Connector.....	84, 85, 86
Database Driver	87
Database Reader	89, 90
DB Reader	90
DB Table Selector	89
DB Writer	86
Decision Tree	143
Decision Tree Learner	144, 146
Decision Tree Predictor.....	147
Decision Tree View	148

Node & Topic Index

Delete Workflow 33
Double to Int..... 82

E

Events..... 3
EXAMPLES Server 20
Execute..... 35
Extensions..... 23

F

File 36, 53, 159
Final Document..... 267
F-measure 141

G

Graphical Properties 103
GroupBy..... 93

H

Histogram 109, 110
Hotkeys 17
Hypothesis Testing..... 172

I

Install..... 7
Interactive View 102
Iris Dataset..... 62

J

Java Snippet 193
Java Snippet (simple)..... 191
JavaScript 99
JavaScript Functions..... 268
Joiner 185
Joiner Settings..... 186

K

k-Means..... 169
knar File Type..... 12
knime
 // Protocol..... 41
KNIME Community 2
KNIME Community Hub 5
KNIME Explorer 19, 21
KNIME Extensions 23
KNIME Public Server..... 20
KNIME Servers 21
knime:..... 41
knwf File Type..... 12

L

Launcher 10
Learner node 134
Line Plot 105, 106
Linear Regression (Learner) 166

M

Math Formula..... 194, 196
Math Formula (Multi Column) 196
Metanode 198
Metanode Collapse Method..... 199
Metanode Context Menu..... 201
Mining 122
Misc..... 191
Missing Value 129
Model Reader..... 161
Model Writer 159
Multilayer Perceptron Predictor 157
MySQL Connector..... 85

N

Naïve Bayes 135
Naïve Bayes Learner..... 135
Naïve Bayes Predictor 136

Node & Topic Index

Neural Network 155
 New Node 33
 New Workflow..... 31
 New Workflow Group..... 31
 Node..... 11, 33
 Node Repository 19
 Normalization Methods..... 132
 Normalizer 131
 Normalizer (Apply)..... 131
 Number to String 80
 Numeric Binner 92

P

Page Break..... 251
 Parallel Coordinates Plot..... 105, 108
 Partitioning..... 124
 Pivoting 96
 PMML..... 129
 PMML Reader 161
 PMML Writer..... 159
 Precision 141
 Predictor node 134

R

Recall..... 141
 RegEx Split 71
 Regression 165
 Regression (Predictor) 168
 Rename 64
 Report Borders..... 244
 Report Columns 243
 Report Fonts 244
 Report Numbers 244
 Report Tables..... 244
 Reporting..... 208, 223
 Resources 2
 ROC Curve 152
 Row..... 49, 177
 Row Filter 49

Row Filter Criteria 51
 Row Sampling 123
 RowID 180
 RProp MLP Learner 156
 Rule Engine 66

S

Save Workflow 32
 Scatter Plot 100
 Scorer 138
 Search 19
 Sensitivity..... 141
 Shuffle 125
 Sorter..... 183
 Specificity..... 141
 Split 68, 123
 SQLite Connector..... 84
 Statistics 122, 162
 String Manipulation 73
 String Replacer..... 76
 String to Number 81
 Style Sheets 245, 265

T

Table View..... 112
 Tables..... 241
 Title..... 238, 268
 Tool Bar..... 16
 Type Conversion 79

U

UCI Machine Learning Repository..... 25, 62, 115
 Unpivoting 182

V

Videos 3
 View..... 35, 99, 102
 Views Properties..... 103

Node & Topic Index

Visualization 99

Workflow Editor 22

Workspace 9

W

Workbench 13

Workflow 10, 30

Workflow Annotations 23

Z

Zebra Style 250

KNIME Beginner's Luck

This book gives a detailed overview of the main tools and philosophy of KNIME Analytics Platform. The goal is to empower new KNIME users with the necessary knowledge to start analyzing, manipulating, and reporting even complex data. No previous knowledge is required.

Dr. Rosaria Silipo has been mining data since her master's degree in 1992. She kept mining data throughout all her doctoral program, her postdoctoral program, and most of her following job positions. She has many years of experience in data analysis, reporting, business intelligence, training, and writing. In the last few years she has been using KNIME for all her data science work, becoming a KNIME trainer and evangelist.

Sanket Joshi works as a Data Analyst in the Evangelism team. After completing a bachelor's degree in Computer Science, he moved to Germany to pursue his master's Degree in Data and Knowledge Engineering at OVGU, Magdeburg where he worked with different data tools. Sanket handles the internal reporting of the team and loves building workflows.