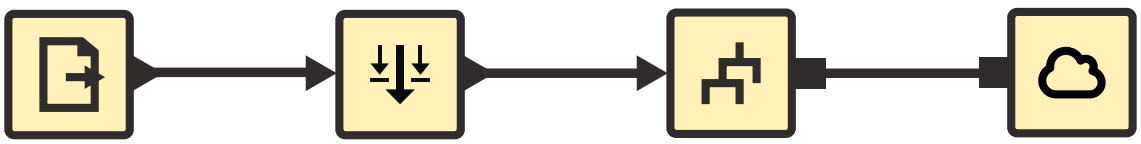
KNIME Textbooks

ROSARIA **SILIPO**

SANKET **JOSHI**

# KNIME Beginner's Luck

KNIME v5.2



Open for Innovation
KNIME

This book has been updated for **KNIME 5.2**.

For information regarding permissions and sales, write to:

KNIME Press
Talacker 50
8001 Zurich
Switzerland

knimepress@knime.com

ISBN: 978-3-9523926-5-2

www.knime.com

# Acknowledgements

First of all, I would like to thank the whole KNIME Team for their patience in dealing with me and my infinite questions.

Among all others in the KNIME Team I would like to specifically thank Peter Ohl for having reviewed this book in order to find any possible aspects that were not compatible with KNIME best practice.

I would also like to thank Casiana Rimbu for the help in providing the most beautiful, clear, and artistic screenshots I could ever imagine and Meta Brown for encouraging me in the first steps of developing the embryonic idea of writing this book.

Many thanks finally go to Heather Fyson for reviewing the book's English.

# Preface

This is the first book I wrote in 2010 for the KNIME Press on how to use KNIME Analytics Platform. Since then, we (the KNIME Press Team and I) have been constantly updating the book twice a year every year, following each new release of KNIE Analytics Platform; not immediately after – but close enough.

That is right! KNIME Beginner's Luck, like all other e-books from KNIME Press, is a live e-book, constantly changing to fit the newest version of the software – which is also true for the new and improved UX/UI that came with KNIME Analytics Platform Version 5 in summer 2023. This liveness of the e-book is also the reason why it has only rarely been printed. Updating printed pages is undoubtedly harder than updating a pdf file!

As this is the first book, it is inevitably about the basics: the basics of KNIME Analytics Platform of course and the basics of a data science project. This book guides you through the most important access functions, data transformation operations, and of course machine learning nodes available in KNIME Analytics Platform. Supplemented with many example workflows, exercises, and screenshots, it will quickly familiarize you with the basic functions of the software. If you are looking for more advanced topics, you won't find them here, instead….

If you want to learn more about advanced machine learning algorithms, flow variables, or loops, check the sequel to this book: "KNIME Advanced Luck". If you want to learn more about text processing, have a look at the book, "From Words To Wisdom". If you come from that school of thoughts where reading manuals or instructions is overrated, you can start directly with reading about solutions to case studies in various application fields in our collection "Practicing Data Science". If your job is more about integrating and blending different data sources and data types, then the book for you is the "Will they blend?" collection. More useful booklets are available on the KNIME Press page, if you are transitioning from Excel, Alteryx, SPSS Modeler, or SAS.

All this is to say that the KNIME Press team and I have been working hard to provide you with the learning material, books, and tutorials, to become progressively more and more productive with KNIME Software and data science concepts.

*Rosaria Silipo*

# Table of Contents

# Chapter 1: Introduction

## 1.1. Purpose and Structure of this Book

We live in the age of data! Every purchase we make is dutifully recorded; every money transaction is carefully registered; every web click ends up in a web click archive. Nowadays everything carries an RFID chip and can record data. We have data available like never before. What can we do with all this data? Can we make some sense out of it? Can we use it to learn something useful and profitable? We need a tool, a surgical knife that can empower us to cut deeper and deeper into our data, to look at it from many different perspectives, to represent its underlying structure.

Let's suppose then that we have this huge amount of data already available, waiting to be dissected. What are the options for a professional to enter the world of Business Intelligence (BI) and Data Science (DS)? The options available are of course multiple and growing rapidly. If our professional does not control an excessive budget, he or she could turn to the world of open-source software. Open-source software, however, is more than a money driven choice. In many cases it represents a software philosophy for resource sharing and control that many professionals support.

Inside the open-source software world, we can find a few Data Science and BI tools. KNIME Analytics Platform represents an easy choice for the non-initiated professional. It does not require learning a specific script and it offers a Graphical User Interface (GUI) to implement and document analysis procedures. In addition - and this is not a secondary advantage - KNIME Analytics Platform can work as an integration platform into which many other BI and Data Science tools can be plugged. It is then not only possible but even easy to analyze data with KNIME Analytics Platform and then to build dashboards on the same processed data with a different BI tool.

Even though KNIME Analytics Platform is very simple and intuitive to use, any beginner would profit from an accelerated orientation through all of the nodes, categories, and settings. This book represents the beginner's luck, because it is aimed to help any beginner to gear up his/her learning process. This book is not meant to be an exhaustive guide to the whole KNIME software. It does not cover implementations under the KNIME Business Hub, which is not open-source, or topics which are considered advanced. Flow Variables, for example, and

implementations of database SQL queries are discussed in the sequel book "KNIME Advanced Luck".

This book is divided into six chapters. The first chapter covers the basic concepts of KNIME Analytics Platform, while chapter two takes the reader by the hand into the implementation of the very first KNIME application. From the third chapter, we start the exploration of data science concepts in a more in-depth manner. The third chapter indeed explains how to perform some basic data exploration and visualization, in terms of nodes and processing flow. Chapter four is dedicated to data modeling. It covers a few demonstrative approaches to machine learning, Naïve Bayes, decision trees, and artificial neural networks. Finally, chapters five, six, and seven are dedicated to reporting. Usually, the results of an investigation based on data visualization or, in a later phase, on data modeling must be shown at some point to colleagues, management, directors, customers, or external workers. Thus, reporting is a very important phase at the end of the data analysis process. Chapter five shows how to prepare the data to export into a report, while chapter six shows how to build the report itself.

Each chapter guides the reader through an ETL or a machine learning (ML) process step by step. Each step is explained in detail and offers some explanations about alternative employments of the current nodes. At the end of each chapter several exercises are proposed to the reader to test and perfect what he/she has learned so far.

Examples and exercises in this book have been implemented using KNIME 5.2. They should also work under subsequent KNIME versions, although there might be slight differences in their appearance.

## 1.2. The KNIME Community

Being an open-source software, KNIME Analytics Platform benefits a number of forums and groups of KNIME users all around the world. This is a good safety net for advice, hints, and learning material. We report below the most popular sites and groups.

### Useful Web Pages

- *KNIME Website:* The root page on the KNIME website.

  https://www.knime.com

- *Software Overview:* The first place to look for an overview of all KNIME products. The open source KNIME Analytics Platform can be downloaded here.

  https://www.knime.com/software-overview

- *Learning Hub:* A central spot to access education material to get you started with KNIME.

  https://www.knime.com/learning

- *KNIME Community Hub:* The perfect place to search for nodes or example workflows when you're not quite sure what you need yet.

  https://hub.knime.com

- *KNIME Forum:* Come here to engage in community discussion, submit feature requests, ask for help, or help others yourself!

  https://forum.knime.com

- *Events and Courses:* Information on all our upcoming events including courses, webinars, learnathons, and summits.

  https://www.knime.com/events

- *Blogs:* A collection of blog posts covering data science with KNIME, a great space to learn what KNIME can really do.

  https://www.knime.com/blog

- *FAQ:* A collection of some of our most commonly asked questions, check out the forum if your answer isn't here!

  https://www.knime.com/faq

- *KNIME Press:* Information on all our available books, like this one!

  https://www.knime.com/knimepress

## Courses, Events, and Videos

- *Courses for KNIME Analytic Platform:* KNIME periodically offers onsite and online courses for the KNIME software. This includes basic and advanced elements. To check for the next available date and to register, just go to the KNIME Events web page (https://www.knime.com/events) and select the tab "Online Course".

- *KNIME Webinars:* A number of webinars are also frequently available on specific topics, like chemistry nodes, text mining, integration with other analytics tools, automated machine learning, deep learning, time series analysis, best practices, and so on. To know about the next scheduled webinars, check the KNIME Events web page (https://www.knime.com/events) and select the tab "Webinars".
- *KNIME Data Connects, KNIME Data Talks, and KNIME Summits:* KNIME Data Connects, KNIME Data Talks and KNIME Summits are held periodically all over the world. These are always good chances to learn more about the KNIME software, to get inspired about new data science projects, and to get to know other people from the KNIME Community. To check for the next upcoming events, just go to the KNIME Events web page (https://www.knime.com/events) and select the tabs "Data Talks", "Summit", or "Data Connect".
- *KNIME TV Channel on YouTube:* KNIME has its own video channel on YouTube, named KNIMETV. There, a number of videos are available to learn more about many different topics and specially to get updated about the new features in the new KNIME releases (http://www.youtube.com/user/KNIMETV).

## Books

- *Advanced Features in KNIME Analytics Platform:*

  For the advanced use: Rosaria Silipo & Victor Palacios, "KNIME Advanced Luck", KNIME Press

  https://www.knime.com/knimepress/advanced-luck

- *Data Science and KNIME*:

  For an overview of data science, data mining, and data analytics, please check: Berthold, M.R., Borgelt, C., Höppner, F., Klawonn, F., Silipo, R., "Guide to Intelligence Data Science", Springer 2020

  https://www.datascienceguide.org/

- *Codeless Deep Learning with KNIME*:

  It is possible to implement deep learning solutions also within KNIME Analytics Platform:

  Kathrin Melcher & Rosaria Silipo, "Codeless Deep Learning with KNIME", Packt, 2020

- *Codeless Time Series Analysis with KNIME*:

  A book explaining the main steps for time series analysis using the KNIME time series components.

  Corey Weisinger, Maarit Widmann, Daniele Tonini, "Codeless Time Series Analysis with KNIME", Packt, 2022

## KNIME Community Hub

However, there is a privileged place where to find information about KNIME nodes and example workflows for your next projects: the KNIME Community Hub (https://hub.knime.com/).

The KNIME Community Hub is a repository of applications, components, and nodes to recycle, reuse, and assemble on KNIME Analytics Platform. Or as it says on the home page: The KNIME Community Hub is "the place to find and collaborate on KNIME workflows and nodes. Here you can find solutions for your data science questions."

When you access the KNIME Community Hub the first time, you end up with the page in Figure 1.1. This page offers a few links to the starting guide documentation, the KNIME Forum, and

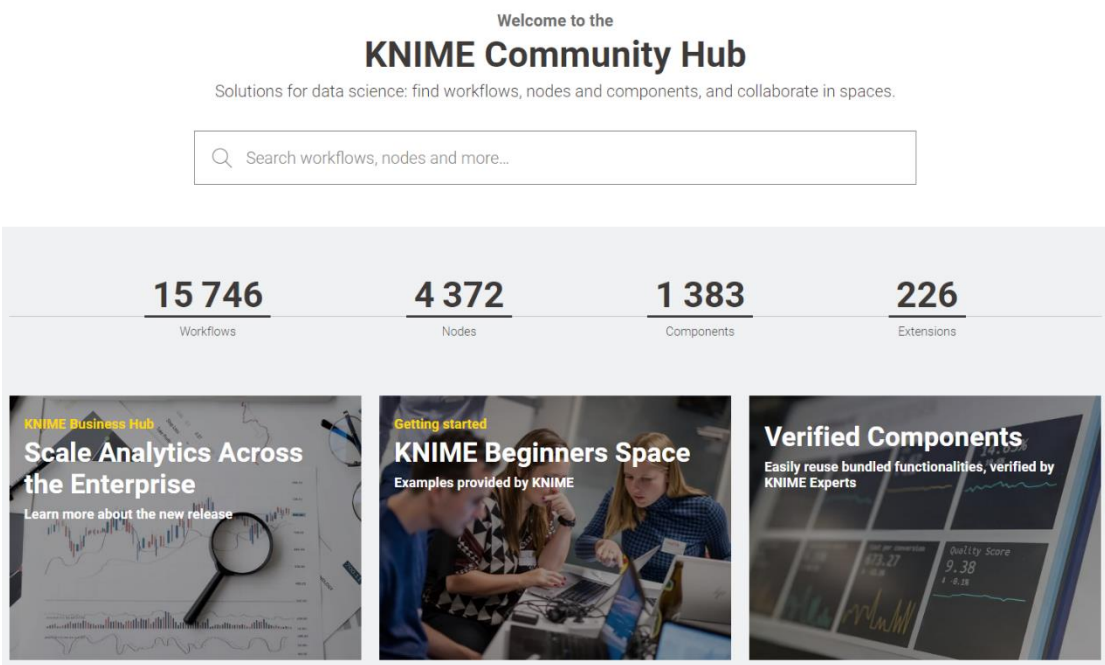



*Figure 1.1. The KNIME Community Hub home page at https://hub.knime.com.*

the KNIME blog. Most importantly at the top it offers a search box to find applications, nodes, and components uploaded by KNIME users in this shared place of the KNIME community.

If we type “Customer Intelligence” in the search box, we end up with a list of nodes and workflows related to Customer Intelligence. If you then select just “Workflows” in the top menu, you will see a list of applications (workflows) implementing some aspects of Customer Intelligence - and appropriately tagged - as uploaded by users of the KNIME community. Indeed, you can upload your own developed applications on the KNIME Community Hub. All you need is an account with the KNIME Forum.

On the KNIME Community Hub you can also find official dedicated spaces with their own description and landing page. Try for example to type “Marketing” and select the tag “Marketing Analytics”. You will get 31 marketing analytics related solutions hosted in the marketing analytics space and described on the landing page on the right, including relevant links.

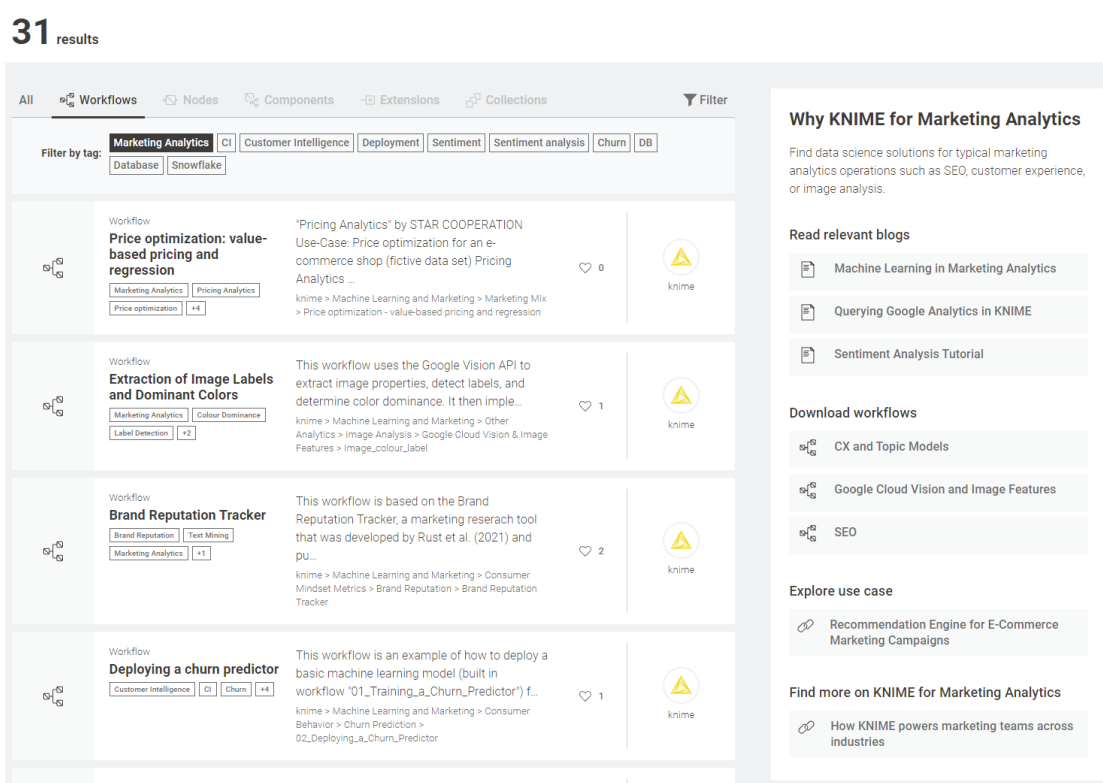



*Figure 1.2. The list of applications (workflows) related (and tagged) with “Marketing Analytics” on the KNIME Community Hub.*

Clicking one of the applications in the list opens the corresponding page (Figure 1.3), with a nice explanatory picture of the implemented workflow.

On the top right corner, you can see the button to log in with your KNIME account. Being logged in allows you to upload, download, comment, like, and update the spaces and workflows for which you have permissions. Below that, you can find the author picture and below that a number of utility buttons: to download the workflow, to like it, to drag & drop it into KNIME Analytics Platform, and to copy the short permanent link for this workflow to share.

If you hover over the author image, and if you have editing permissions for this Hub space, a pen will appear. Clicking on it will allow you to give other KNIME users permission to upload and change this space.

Notice that the KNIME Community Hub is a repository for workflows, but also for nodes, components, and extensions.

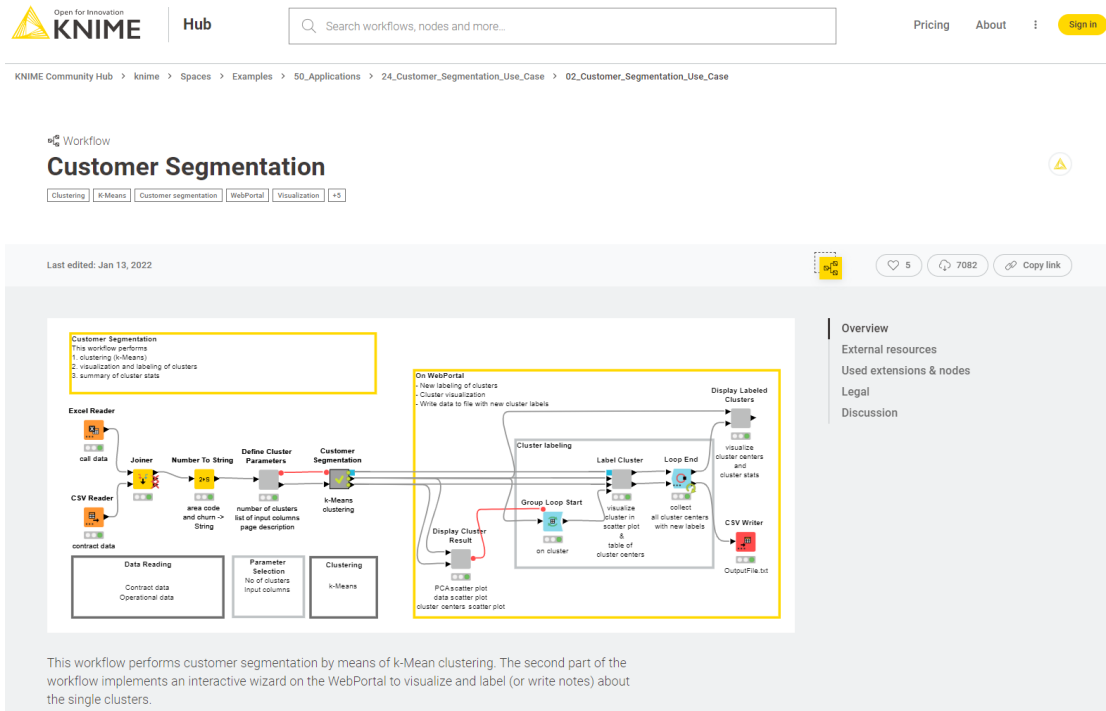



*Figure 1.3. The page dedicated to the application named "Customer Segmentation" on the KNIME Community Hub, with short link [https://kni.me/w/37cHxqru6dbllUeP](https://kni.me/w/37cHxqru6dbllUeP).*

# 1.3. Download and Install KNIME Analytics Platform

There are two available KNIME products:

- the open source KNIME Analytics Platform, which can be downloaded free of charge at https://www.knime.com/software-overview under the GPL version 3 license
- the KNIME Business Hub, which is described at https://www.knime.com/knime-business-hub

Analytically speaking, the functionalities of the two products are the same. The KNIME Business Hub in addition includes a number of useful IT features for team collaboration, enterprise workflow deployment and management, data warehousing, integration, and scalability for the data science lab. In this book, however, we will work with KNIME Analytics Platform (open source). To start playing with KNIME Analytics Platform, first, you need to download it to your machine.

## Download KNIME Analytics Platform

- Go to www.knime.com
- In the upper right corner of the main page, click “Download”
- Provide a little information about yourself (that is appreciated), then proceed to step 2 “Download KNIME”
- Choose the version that suits your environment (Windows/Mac/Linux, 32 bit/64 bit, with or without Installer for Windows) optionally including all free extensions
- Accept the terms and conditions
- Start downloading. You will end up with a zipped (*.zip), a self-extracting archive file (*.exe), or an Installer application
- For .zip and .exe files, just unpack it in the destination folder. If you selected the installer version, just run it and follow the installer instructions.

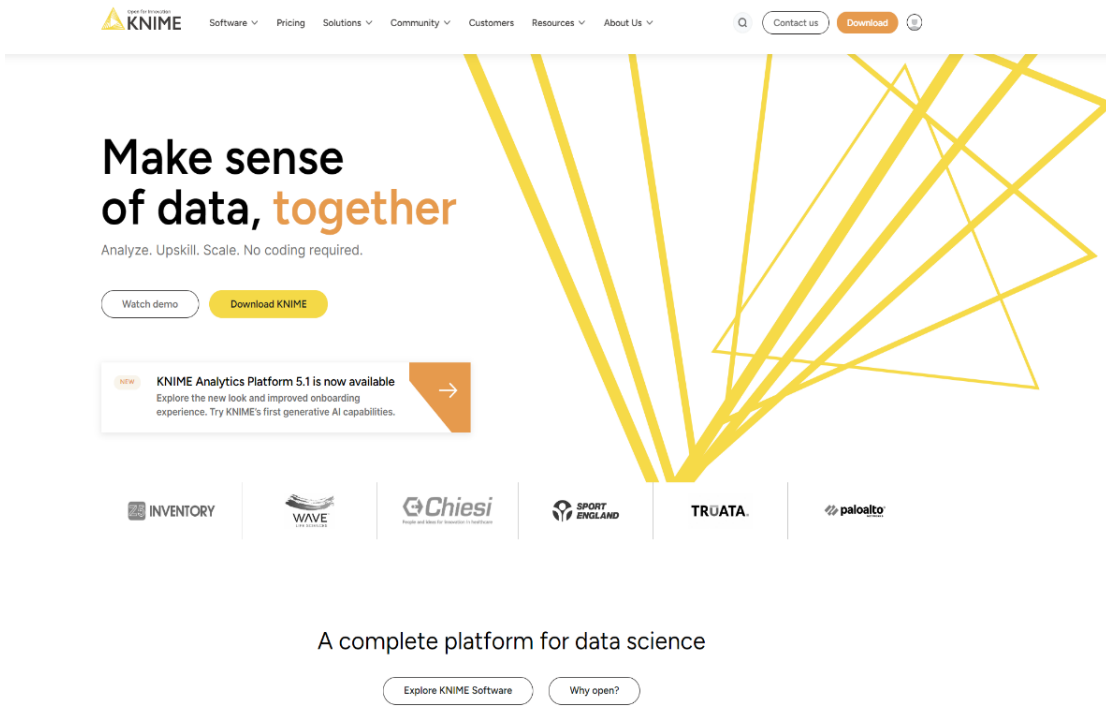



*Figure 1.4. The KNIME web page.*

## 1.4. Workspace

To start KNIME Analytics Platform, open the folder where KNIME has been installed and run knime.exe (or knime on a Linux/Mac machine). If you have installed KNIME using the Installer, then you can just click the icon on your desktop or on your Windows main menu.

If you are starting KNIME Analytics Platform for the first time, you will be asked if you want to share your usage statistics with KNIME. These numbers will be used to fuel the best practice recommendation engine provided within KNIME Analytics Platform workbench: the Workflow Coach. No personal information will ever reach KNIME and your anonymous statistics will never be shared with anybody.

After the splash screen, the “Workspace Launcher” window requires you to enter the path of the workspace.

## The Workspace Launcher

The *workspace* is a folder where all preferences and applications (workflows), both developed and currently under development, are saved for the next KNIME session.

The workspace folder can be located anywhere on the hard disk.

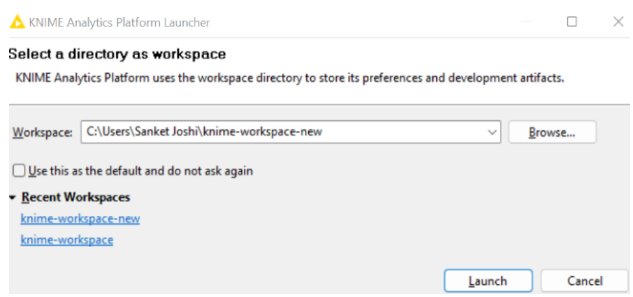



*Figure 1.5. The "Workspace Launcher" window.*

By default, the workspace folder is `"..\knime-workspace"`. However, you can easily change that, by changing the path proposed in the "Workspace Launcher" window, before starting the KNIME working session.

Once KNIME Analytics Platform has been opened, from within the KNIME workbench you can switch to another workspace folder, by selecting "File" in the top menu and then "Switch Workspace". After selecting the new workspace, KNIME Analytics Platform restarts, showing the workflow list from the newly selected workspace. Notice that if the workspace folder does not exist, it will be automatically created.

When having a large number of customers for example, different workspaces can be used for each one of them. This keeps each workspace clean and tidy and protects from mixing up information by mistake.

For this project I used the workspace "Knime-workspace-new".

# 1.5. KNIME Workflow

KNIME Analytics Platform does not work with scripts, it works with graphical workflows.

Small little icons, called nodes, are dedicated each to implement and execute a given task.

A sequence of nodes makes a workflow to process the data to reach the desired result.

## What is a Workflow?

A workflow is an *analysis flow*, i.e., the *sequence of analysis steps* necessary to reach a given result. It is the pipeline of the analysis process, something like:

*Step 1: Read data*
*Step 2: Clean data*
*Step 3: Filter data*
*Step 4: Train a model*

KNIME Analytics Platform implements its workflows *graphically*. Each step of the data analysis is implemented and executed through a little box, called *node*. A sequence of nodes makes a workflow.

In the KNIME whitepaper[1] a workflow is defined as follows: "Workflows in KNIME are graphs connecting nodes, or more formally, direct acyclic graphs (DAG)."

On the right is an example of a KNIME workflow, with:

- a node to read data from a file
- a node to exclude some data columns
- a node to filter out some data rows
- a node to write the processed data into a file

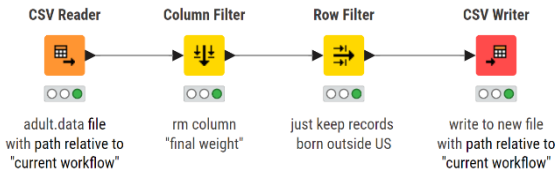



*Figure 1.6. Example of a KNIME Workflow.*

> ***Note.*** A workflow is a data analysis sequence, which in a traditional programming language would be implemented by a series of instructions and calls to functions. KNIME Analytics Platform implements it graphically. This graphical representation is more intuitive to use, lets you keep an overview of the analysis process, and makes for the documentation as well.

## What is a Node?

A node is the *single processing unit* of a workflow.

---

[1] *M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Koetter, T. Meinl, P. Ohl, C. Sieb, and B. Wiswedel, "KNIME: The Konstanz Information Miner". KDD 2006 (http://www.kdd2006.com/docs/KDD06_Demo_13_Knime.pdf).*

A node takes a data set as input, processes it, and makes it available at its output port. The "processing" action of a node ranges from modeling - like an Artificial Neural Network Learner node - to data manipulation - like transposing the input data matrix - from graphical tools - like a scatter plot, to reading/writing operations.

Every node in KNIME has 4 states:

- Inactive and not yet configured → **red** light
- Configured but not yet executed → **yellow** light
- Executed successfully → **green** light
- Executed with errors → **red with cross** light

Nodes containing other nodes are called *metanodes* or *components*.

On the right are four examples of the same node (a *File Reader* node) in each one of the four states.

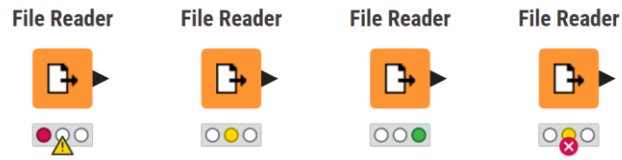



*Figure 1.7. The File Reader node in different states.*

## 1.6. File Extensions: *.knwf* and *.knar*

KNIME workflows can be packaged and exported in *.knwf* or *.knar* files. A *.knwf* file contains only one workflow, while a *.knar* file contains a group of workflows. Such extensions are associated with KNIME Analytics Platform. A double-click opens the workflow inside KNIME Analytics Platform.

| | | | |
|---|---|---|---|
| 01_From_Strings_to_Documents.knwf | 10/4/2017 9:45 AM | KNIME Workflow ... | 18,619 KB |
| 04_Interaction_Graph.knwf | 9/29/2017 8:20 AM | KNIME Workflow ... | 9,465 KB |
| 06_REST_Examples_Google_Geocode.knwf | 7/29/2017 7:09 PM | KNIME Workflow ... | 62 KB |
| 06_Semantic_Web_updated.knar | 11/3/2016 2:24 PM | KNIME Archive File | 178 KB |
| AzureDemoWorkflowArchive.knar | 5/5/2017 11:24 AM | KNIME Archive File | 24,104 KB |
| Building a Simple Classifier_.knwf | 2/18/2017 5:46 PM | KNIME Workflow ... | 43 KB |
| Cookbook_Ch5.knar | 11/24/2017 10:03 ... | KNIME Archive File | 477 KB |
| Cookbook_Ch6.knar | 11/24/2017 10:26 ... | KNIME Archive File | 155 KB |
| Corsair.knwf | 7/10/2017 4:20 PM | KNIME Workflow ... | 106 KB |

*Figure 1.8. .knwf and .knar files are associated with KNIME Analytics Platform. A double-click opens the workflow(s) directly inside the platform.*

# 1.7. KNIME User Interface

After accepting the workspace path, the KNIME UI opens on a “Getting started with KNIME Analytics Platform 5” entry page. This page provides access to the local space or create a new workflow. This page also provides the mount points. You can connect to them for usage by clicking the sign-in button. The “KNIME User Interface” consists of a top menu, a tool bar, and a few panels. Panels can be closed, re-opened.

- ***Home button:*** This panel shows the list of workflow projects available in the selected workspace (LOCAL), on the EXAMPLES, on the My-KNIME-Hub (your own space on the KNIME Community Hub), or on other connected KNIME Hub spaces.
- ***Quick node adding panel:*** This is a node recommendation engine. It will provide the list of the compatible nodes to follow the currently selected node. You will also be able to search for the other available nodes by typing in keywords in the search field. Double-click the node you want to add to the workflow, and it will be added and connected automatically.
- ***Node Repository:*** This panel contains all the nodes that are available in your KNIME installation. It is something similar to a palette of tools when working in a report or with a web designer software. There we use graphical tools, while in KNIME we use data analytics tools.

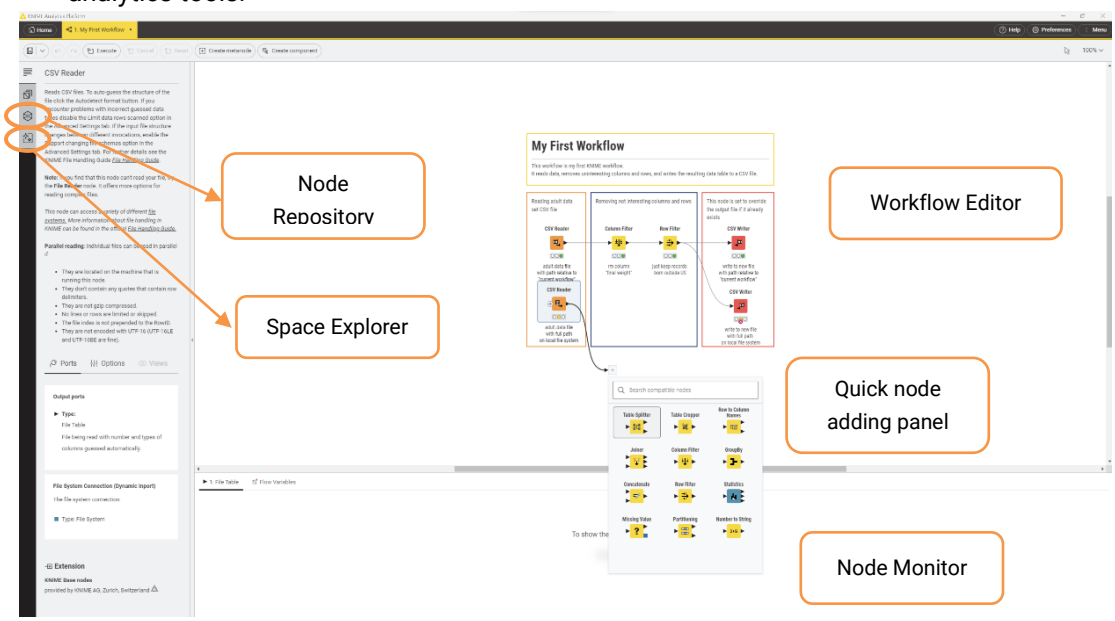



*Figure 1.9. The KNIME user Interface.*

- ***Workflow Editor:*** Workflow Editor is the central part of the KNIME UI. A node can be selected from the "Node Repository" panel and dragged and dropped here, in the "Workflow Editor" panel. Nodes can be connected by clicking the output port of one node and releasing the mouse either at the input port of the next node or at the next node itself.
- ***Space Explorer:*** To navigate to local or KNIME Hub spaces and access the workflows, components, and files, you can switch to the "Space Explorer" panel.
- ***Node Description:*** If a node or a workflow is selected, the "Node Description" panel on the left displays a summary description of the node's functionalities or the workflow's meta information.

## 1.8. Help

The "Help" button is one of the newer options added to the KNIME User Interface in version 5.2.

It includes a few more buttons; if you click on those buttons, you will be redirected to the respective website.

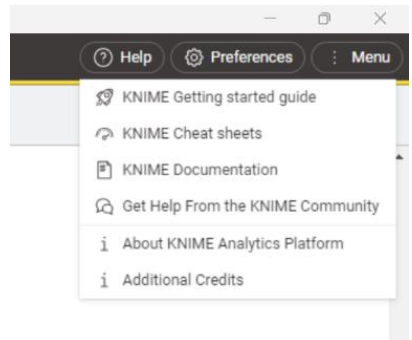



*Figure 1.10. The "Help" button in KNIME.*

## 1.9. Preferences

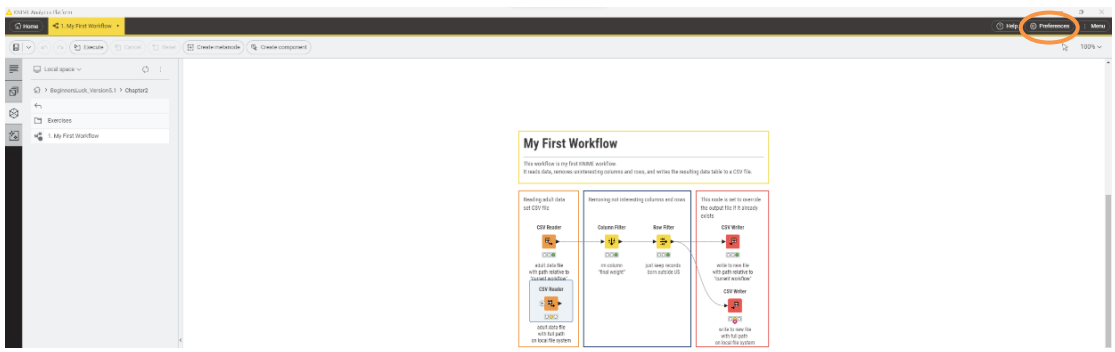



*Figure 1.11. The "Preferences" option in the KNIME User Interface.*

Preferences brings you to the window where all KNIME settings can be customized. Under the item "KNIME" can be found:

- ***Chemistry*** – has settings related to the KNIME Renderers in the chemistry packages.

- ***Databases*** – specifies the location of specific database drivers, not already available within KNIME. Indeed, the most common and most recent database drivers are already available in the driver menu of Database nodes. However, if you need some specific driver file, you can set its path here.
- ***Space Explorer*** – contains the list of the shared repositories via KNIME Hub spaces.
- ***KNIME GUI*** – allows the customization of the KNIME workbench options and layout via a number of settings.
- ***Master Key*** – contains the master key to be used in nodes with an encryption option, like database connection nodes. Since KNIME 2.3 database passwords are passed via the "Credentials" workflow variables and the Master Key preference has been deprecated. You can still find it in the Preferences menu for backward compatibility.
- In ***Meta Info Preferences*** you can upload meta-info template for nodes and workflows.
- Here you can also find the preference settings for the ***external packages***, like ***H2O, R, Report Designer, Perl, Perl, Open Street Map,*** and others if you have them installed. In particular, for the external scripts, this page offers the option to set the path to the reference script installation.
- Finally, ***Workflow Coach*** contains the dataset to be used for the node recommendation engine: the community, a Hub workspace, or your own local workspace.

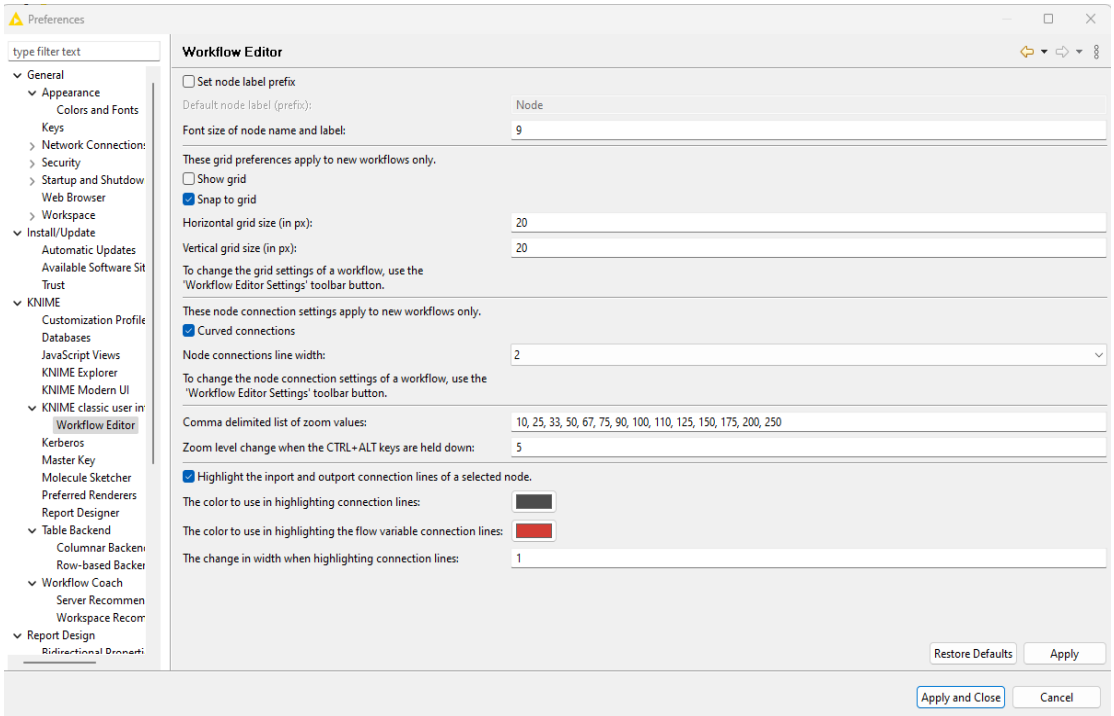



*Figure 1.12. The "Preferences" window.*

## Tool Bar

The tool bar is another important piece of the KNIME User Interface.

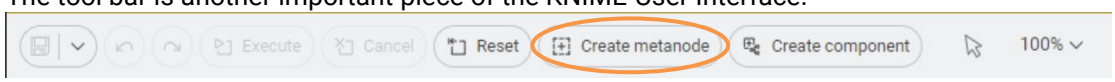



*Figure 1.13. The "Create Metanode" button in the tool bar.*

From the left, we see

- an icon to *save* or *save as* the current workflow. To the left, we find the icon to save the selected workflow
- *undo* and *redo* the changes that were made to the workflow
- *execute* or *execute all* (which executes all the selected nodes)
- *cancel* the executing node(s)
- *reset* all the node(s) to the original state
- *create Metanode*

- *create Component*,
- the arrow icon which provides the option of selecting, using the annotating mode or using the panning mode
- and on the right is the zoom (in %) level selection.

For now, let's have a look at the ***Create Metanode*** button. The *Create Metanode* button creates a metanode for all the selected nodes. It creates one node inside which you can find all the selected nodes. This is particularly useful to create a clean and easy-to-use workflow, for example, by compressing all the nodes used for data cleaning within a *Pre-processing* metanode.

## Hotkeys

For all keyboard lovers, most KNIME commands can also run via hotkeys. All hotkeys are listed in the KNIME menus on the side of the corresponding commands or in the tooltip messages of the icons in the Tool Bar under the Top Menu. Here are the most frequently used hotkeys.

**Node Configuration**

- `F6` opens the configuration window of the selected node

**Node Execution**

- `F7` executes selected configured nodes
- `Shift + F7` executes all configured nodes

**Stop Node Execution**

- `F9` cancels selected running nodes
- `Shift + F9` cancels all running nodes

**Node Resetting**

- `F8` resets selected nodes

**Save Workflows**

- `Ctrl + S` saves the workflow
- `Ctrl + Shift + S` saves all open workflows

- `Ctrl + W` closes all open workflows

**Metanode**

- `Shift + F12` opens Meta Node Wizard

**To move Annotations**

- `Ctrl + Shift + PgUp/PgDown` moves the selected annotation in the front or in the back of all the overlapping annotations

# 1.10. Menu

The “Menu” option button is another option added to the KNIME User Interface in KNIME Analytics Platform v5.2.

Once you click this button, you will see multiple options, and we will see what each option does.

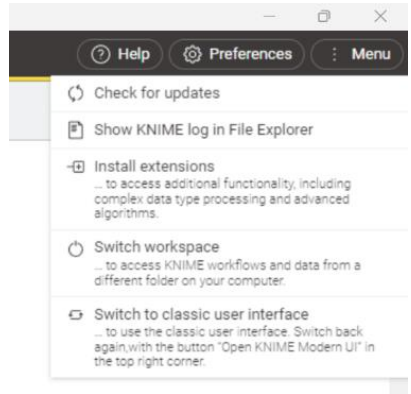



*Figure 1.14. The "Menu" button in the tool bar.*

- ***Check for Updates:*** On clicking this option, you can see if any updates are available for your KNIME Analytics Platform.
- ***Show KNIME log in File Explorer:*** This option will open a window in your local system where the KNIME log file is stored.
- ***Install Extensions:*** On clicking this option, you will be able to see a list of extensions that you could install.
- ***Switch Workspace:*** This will allow you to change the working directory.
- ***Switch to Classic User Interface:*** This option will first ask you to save or not save the current workflow and then switch to the classic UI. You can also switch back to the Modern UI from the classic UI by clicking on the “Open KNIME Modern UI” option in the top right corner of the Analytics Platform.

# 1.11. Node Repository

In the lower left corner, we find the Node Repository, containing all installed nodes organized in categories and subcategories. KNIME Analytics Platform has accumulated by now more than 1500 nodes. It has become hard to remember the location of each node in the Node Repository. To solve this problem, two search options are available: by exact match and by fuzzy match, both in the search box placed at the top of the Node Repository panel.

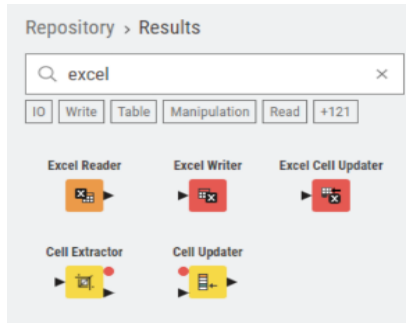



*Figure 1.15. Word search in the Node Repository panel: exact match mode.*

### Search Box

At the top of the "Node Repository" panel there is a search box. If you type a keyword in the search box and hit "Enter", you obtain the list of nodes containing an exact match of that keyword. Press the "Esc" key to see all nodes again.

# 1.12. Space Explorer

We find the Space Explorer panel below the Home button, one of the four panels on the KNIME User Interface. This panel contains:

- Under LOCAL the workflows that have been developed in the selected workspace
- The mount points to a number of KNIME Hub spaces.
- The workflows contained in the reference workspace of such Hub spaces.
- The access to the My-KNIME-Hub, that is to your space on the KNIME Community Hub. Remember that you need an account with the KNIME Forum to access this space.

At the beginning, the Space Explorer panel only contains LOCAL, My-KNIME-Hub, and EXAMPLES. As we already stated, LOCAL shows the content of the selected workspace. EXAMPLES points to a read-only public repository, accessible via anonymous login. This repository hosts a number of example workflows that you can use to jump start a new project. My-KNIME-Hub allows to access your space on the KNIME Community Hub.

When you open KNIME Analytics Platform for the first time, you will find a folder named "Example Workflows" containing the solutions to a few common data science use cases, comprehensive of data.

Folders in "Space Explorer", containing workflows, are also called "Workflow Groups".

***Note.*** Space Explorer panel can also host data. Just create a folder under the workspace folder, fill it with data files on the machine, and select "Refresh" in the context-menu (right-click) of the "Space Explorer" panel.

## My-KNIME-Hub

From the Space Explorer panel, you can access your spaces on the KNIME Community Hub and upload and update new or existing content in there.

By default, an authenticated KNIME user has a public space, for material to share publicly, and a private space to park his/her own material for further usage. However, new private or public spaces can be created with a right-click on My-KNIME-Hub in the Space Explorer panel and then a selection of the option "Create new Space…".

By default, you are the only owner of your own spaces. However, when accessing this space from a web browser, after hovering on your image in the top right corner, a pen appears. This will allow us to add colleagues and teammates as contributors to the space.

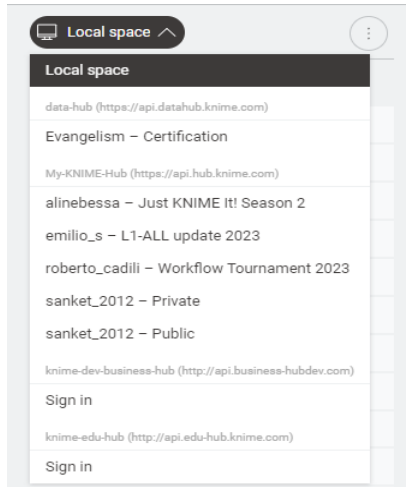



*Figure 1.16. Space Explorer panel. At the top the content of the EXAMPLES; below the content of the LOCAL workspace.*

## EXAMPLES

A link to EXAMPLES is available in the "Space Explorer" panel. This is a repository provided by KNIME to all users for tutorials and demos. There you can find a number of useful examples on how to implement specific tasks with KNIME. To connect to the EXAMPLES:

- double click "EXAMPLES" in the "Space Explorer" panel
- double click "Double click to connect…"

You should be automatically logged in as a guest.

To transfer example workflows from the EXAMPLES to your LOCAL workspace, just drag and drop or copy and paste (Ctrl-C, Ctrl-V in Windows) them from “EXAMPLES” to “LOCAL”.

You can also open the EXAMPLES workflows in the workflow editor, however only temporarily and in read-only mode. A yellow warning box on top warns that this workflow copy will not be saved.

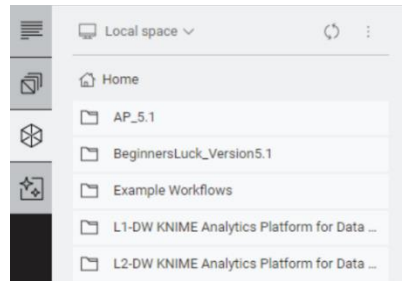



*Figure 1.17. Space Explorer panel.*

The Space Explorer panel can of course host more than one KNIME Hub space. It is enough to add mountpoints to the list of the available KNIME Hub spaces.

## Mounting KNIME Business Hub in Space Explorer

To add KNIME Business Hubs to the “Space Explorer” panel, in the modern UI, click on “Preferences” -> KNIME -> KNIME Explorer and:

- The “Preferences (Filtered)” window opens on the “KNIME Explorer” page and lists all KNIME spaces already mounted in this KNIME instance. The three KNIME spaces available by default on every KNIME instance are the local workspace “LOCAL”, the KNIME “EXAMPLES”, and the My-KNIME-Hub located on the KNIME Community Hub (hub.knime.com).
- Use the “New” and the “Remove” button to add /remove connections to remote Hub spaces.
- After clicking the “New” button, fill in the required information about the KNIME Business Hub in the “Select New Content” window (Figure 1.18).

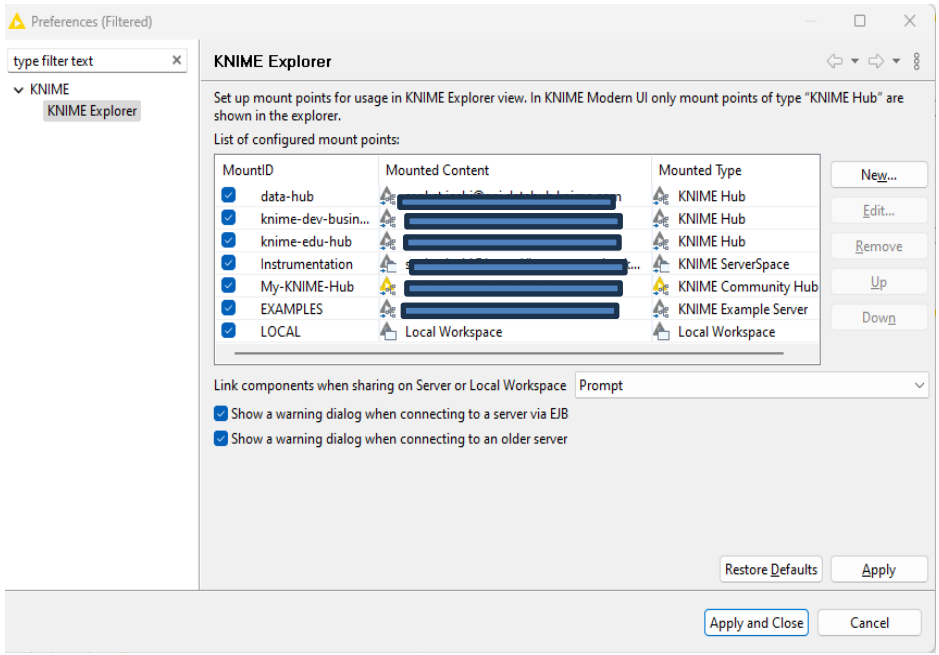



*Figure 1.18. The "Preferences (Filtered)" window.*

The same KNIME Explorer "Preferences" page can be reached via *File > Preferences > KNIME Explorer*.

To login into any of the available KNIME Business Hubs in the "Space Explorer" panel:

- right-click or double-click the Hub space name
- provide the credentials

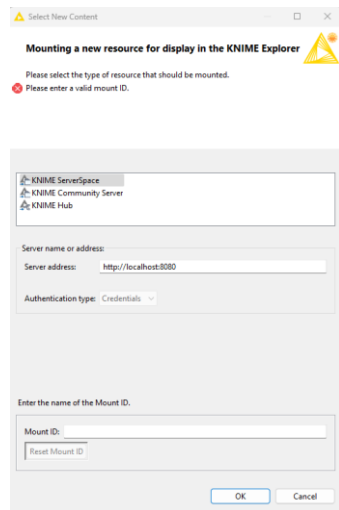



*Figure 1.19. The "Select New Content" window.*

## Workflow Editor

The central piece of the KNIME User Interface consists of the workflow editor itself. This is the place where a workflow is built by adding one node after the other. Nodes are inserted in the workflow editor by drag and drop or double-click from the Node Repository or the Quick node adding panel. The workflow building process will be described widely in the next sections of this book. Here, we will describe how to customize and probably improve the canvas role of the workflow editor space. We will describe two options:

- change the canvas appearance with grids and different visualizations for the connections;
- introducing annotations to comment the work.

### Adding annotations to the canvas

It is also possible to include annotations in the workflow editor. Annotations can help to explain the task of the workflow and the function of each node or group of nodes. The result is an improved documentation-like overview of the workflow general task and of the single sub-tasks.

**Workflow Annotations**

To insert a new annotation:

- right-click anywhere in the workflow editor and select “New Workflow Annotation”
- a gray small frame appears; this is the default annotation frame
- double-click the frame to edit its content
- Notice the tool bar appearing at the top to edit text style, text size, background color, text alignment, and border properties (color, thickness)
- To reopen an annotation, just double-click anywhere on the annotation

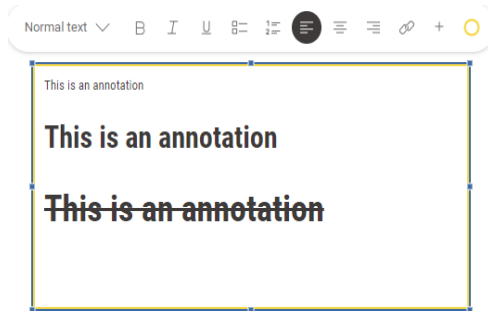



*Figure 1.20. The Annotation Editor.*

## 1.13. Download the KNIME Extensions

KNIME Analytics Platform is an open-source product. As every open-source product, it benefits from the feedback and the functionalities that the community develops. A number of extensions are available for KNIME Analytics Platform. If you have downloaded and installed KNIME Analytics Platform including all its free extensions, you will see the corresponding categories in the Node Repository panel, such as KNIME Labs, Text Processing, R Integration, and many others. However, if at installation time, you have chosen to install the bare KNIME Analytics Platform without the free extensions, you might need to install them separately at some point on a running instance.

## Installing KNIME Extensions

To install a new KNIME extension from within KNIME Analytics Platform, there are three options.

1. From the Top right options, select Menu → "Install KNIME Extensions", select the desired extension, click the "Next" button and follow the wizard instructions.
2. IF you want install extension from the classic UI, From the Top Menu, select "Help" → "Install New Software". In the "Available Software" window, in the "Work with" textbox, select the URL with the KNIME update site (usually named "KNIME Analytics Platform 5.x Update Site" - `http://update.knime.com/analytics-platform/4.x`). Then select the extension, click the "Next" button and follow the wizard instructions.
3. Search the KNIME Community Hub, on a web browser or from the KNIME Community Hub panel. When the desired extension is found, drag and drop the extension icon from the browser to the Workflow Editor.

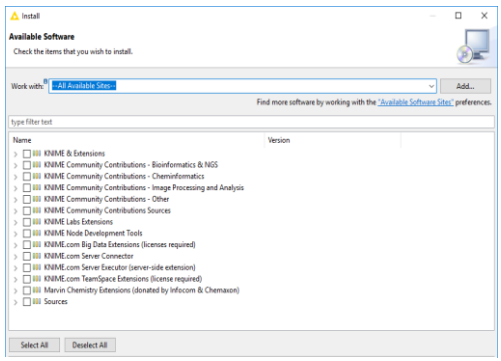



*Figure 1.21. The "Available Software" window.*

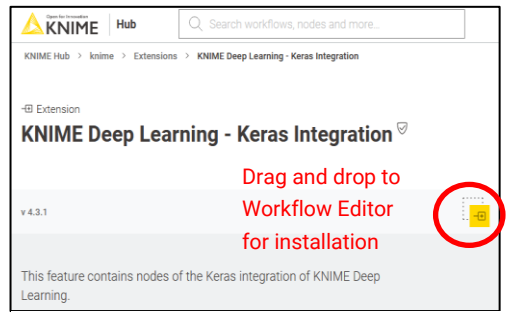



*Figure 1.22. An extension from the KNIME Community Hub.*

Once the selected KNIME extension(s) has/have been installed and KNIME has been restarted, you should see the new category, corresponding to the installed extension, in the "Node Repository".

In the "Available Software" window you can find some extension groups: KNIME & Extensions, KNIME Labs Extensions, KNIME Node Development Tools, Sources, and more. "KNIME & Extensions" contains all extensions provided for the current release; "KNIME Labs Extensions" contains a number of extensions ready to use, but not yet of x.1 release quality; "KNIME Node Development Tools" contains packages with some useful tools for Java programmers to develop nodes; "Sources" contains the KNIME source code. Specific packages donated by third parties or community entities might also be available in the list of extensions. These are usually

grouped under “Community” categories. My advice is to install all extensions, even the cheminformatics ones. Many of them contain several useful nodes not necessarily restricted to a particular domain.

## 1.14. Data and Workflows for this Book

This book builds a few examples and provides the solutions to the exercises. The workflows are accessible via the KNIME Community Hub and are stored in the KNIME Press space – look for the respective book and KNIME version. To download material from the KNIME Community Hub, you need to be logged in with your KNIME account (see how to create a KNIME account). After entering the KNIME Community Hub, in order to download the workflows, just click on the cloud icon. Download the whole folder onto your machine from the KNIME Beginner’s Luck space, which will result in a *.knar* file. Then double click it OR import it into the KNIME Explorer via Select “File” → “Import KNIME Workflow…”.

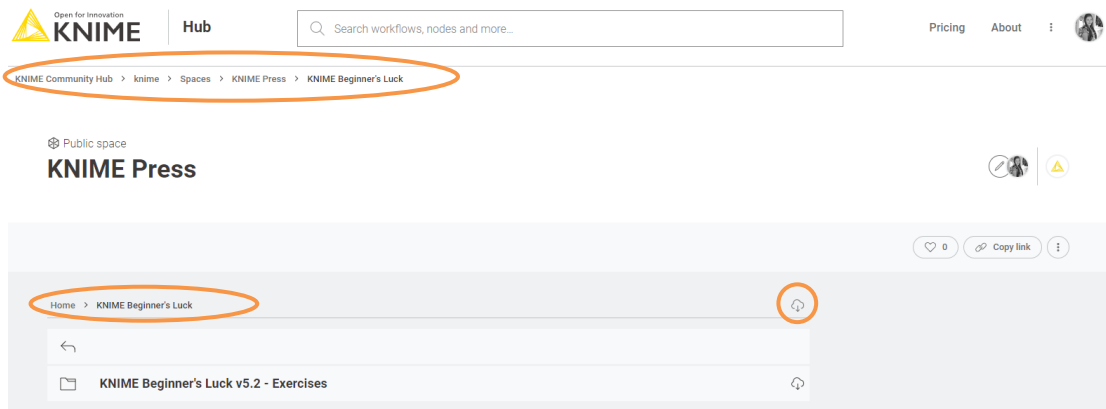



*Figure 1.23. Beginners Luck space on the KNIME Community Hub.*

At the end of the import operation, in the Space Explorer panel you should find a *KNIME Beginner’s Luck v5.2 - Exercises* folder containing Chapter2, Chapter3, Chapter4, Chapter5, Chapter6, and Chapter7 subfolders, each one with workflows and exercises to be implemented in the next chapters. You should also find a KBLdata folder containing the required data.

The data used for the exercises and for the demonstrative workflows of this book were either generated by the author or downloaded from the UCI Machine Learning Repository, a public data repository (http://archive.ics.uci.edu/ml/datasets). If the data set belongs to the UCI Repository, a full link is provided here for download. Data generated by the author, that is not public data, are located in the KBLdata folder.

Data from the UCI Machine Learning Repository:

- *Adult.data:* http://archive.ics.uci.edu/ml/datasets/Adult
- *Iris data:* http://archive.ics.uci.edu/ml/datasets/Iris
- *Yellow-small.data (Balloons):* http://archive.ics.uci.edu/ml/datasets/Balloons
- *Wine data:* http://archive.ics.uci.edu/ml/datasets/Wine

# 1.15. Exercises

## Exercise 1

Create your own workspace and name it “book_workspace”. You will use this workspace for the next workflows and exercises.

### Solution to Exercise 1

- Launch KNIME
- In Workspace Launcher window, click “Browse”
- Select the path for your new workspace
- Click “Launch”

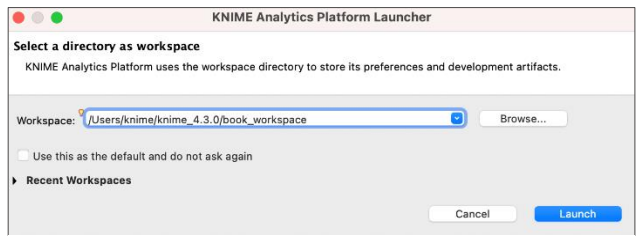



*Figure 1.24. Exercise 1: Create workspace “book_workspace”.*

To keep this as your default workspace, enable the option on the lower left corner.

## Exercise 2

Install the following extensions:

- KNIME Database
- KNIME Javascript Views
- KNIME Report Designer

## Solution to Exercise 2

From the Top right corner options, select Menu → “Install Extensions”. Search and select the required Extensions. Click “Next” and follow the instructions.

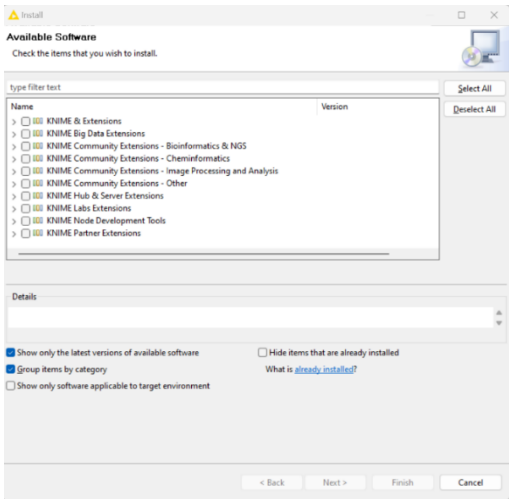



*Figure 1.25. Exercise 2: List of KNIME Extensions.*

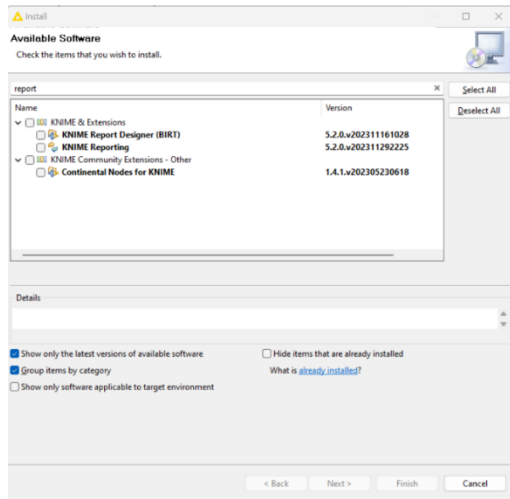



*Figure 1.26. Exercise 2: Reporting Extension.*

# Exercise 3

Search all “Row Filter” nodes in the Node Repository. From the “Node Description” panel, can you explain what the difference is between a “Row Filter”, a “Reference Row Filter”, and a “Nominal Value Row Filter”? Show the node effects by using the following data tables:

**Original Table**

| Position | Name | Team |
|---|---|---|
| 1 | The Black Rose | 4 |
| 2 | Cynthia | 4 |
| 3 | Tinkerbell | 4 |
| 4 | Mother | 4 |
| 5 | Augusta | 3 |
| 6 | The Seven Seas | 3 |

**Reference Table**

| Ranking | Scores |
|---|---|
| 1 | 22 |
| 3 | 14 |
| 4 | 10 |

## Solution to Exercise 3

### Row Filter

The node allows for row filtering according to certain criteria. It can include or exclude certain ranges (by row number), rows with a certain row ID, and rows with a certain value in a selectable column (attribute). In the example below, we used the following filter criterion: `team > 3`

**Original Table**

| Position | Name | Team |
|---|---|---|
| 1 | The Black Rose | 4 |
| 2 | Cynthia | 4 |
| 3 | Tinkerbell | 4 |
| 4 | Mother | 4 |
| 5 | Augusta | 3 |
| 6 | The Seven Seas | 3 |

**Filtered Table**

| Position | Name | Team |
|---|---|---|
| 1 | The Black Rose | 4 |
| 2 | Cynthia | 4 |
| 3 | Tinkerbell | 4 |
| 4 | Mother | 4 |

### Reference Row Filter

This node has two input tables. The first input table, connected to the bottom port, is taken as the reference table; the second input table, connected to the top port, is the table to be filtered. You have to choose the reference column in the reference table and the filtering column in the second table. All rows with a value in the filtering column that also exists in the reference column are kept, if the option “include” is selected; they are removed if the option “exclude” is selected.

**Reference Table**

| Ranking | Scores |
|---|---|
| 1 | 22 |
| 3 | 14 |
| 4 | 10 |

**Filtering Table**

| Position | Name | Team |
|---|---|---|
| 1 | The Black Rose | 4 |
| 2 | Cynthia | 4 |
| 3 | Tinkerbell | 4 |
| 4 | Mother | 4 |
| 5 | Augusta | 3 |
| 6 | The Seven Seas | 3 |

**Resulting Table**

| Position | Name | Team |
|---|---|---|
| 1 | The Black Rose | 4 |
| 3 | Tinkerbell | 4 |
| 4 | Mother | 4 |

In the example above, we use “Ranking” as the reference column in the reference table and “Position” as the filtering column in the filtering table. We have chosen to include the common rows.

**Nominal Value Row Filter**

Filters the rows based on the selected value of a nominal attribute. A nominal column and one or more nominal values of this attribute can be selected as the filter criterion. Rows that have these nominal values in the selected column are included in the output data. Basically, it is a Row Filter applied to a column with nominal values. Nominal columns are string columns and nominal values are the values in it.

In the example below, we use “name” as the nominal column and `name = Cynthia` as the filtering criterion.

**Original Table**

| Position | Name | Team |
|---|---|---|
| 1 | The Black Rose | 4 |
| 2 | Cynthia | 4 |
| 3 | Tinkerbell | 4 |
| 4 | Mother | 4 |
| 5 | Augusta | 3 |
| 6 | The Seven Seas | 3 |

**Filtered Table**

| Position | Name | Team |
|---|---|---|
| 2 | Cynthia | 4 |

# Chapter 2: My First Workflow

## 2.1. Workflow Operations

If you have started KNIME for the first time, your "Space Explorer" panel on the top left corner of the KNIME User Interface contains only one workflow group (folder) named "**Example Workflows**". This "Example Workflows" folder contains a number of sub-folders, each with basic workflows for very common use cases:

- ***Basic Examples.*** Workflows in "Basic Examples" sub-folder show basic general operations, like import data, data blending, ETL, train and evaluate a model, and finally display results in a simple report.
- ***Customer Intelligence.*** Basic workflows for churn prediction, credit scoring, and customer segmentation are available inside sub-folder "Customer Intelligence".
- ***Retail.*** A recommendation engine is built in sub-folder "Retail".
- ***Social Media.*** An example of social media analysis is available in "Social Media".

These example workflows can be reused and readapted for your own application. However, in this chapter we want to build our own very first workflow, to perform the following basic operations:

- Read data from a text file
- Filter out undesired rows
- Filter out undesired columns
- Write resulting data to a CSV file

We will use this first workflow to explore data structures and data types, node and workflow commands, debugging and data inspection possibilities, commenting options, configuration windows and execution commands, and other features available inside the KNIME User Interface.

In order to keep our space clean, we use workflow groups to organize workflows by chapter or topic. Let's create now a new workflow group and call it "Chapter2". Once this has been done, we need to populate the newly created workflow group with a new workflow, let's call it "My

First Workflow". Eventually in the "Space Explorer" panel, you should see workflow group "Chapter2" with a workflow named "My First Workflow" in it. For now, "My First Workflow" is an empty workflow. Indeed, if you double-click it, the workflow editor opens to an empty page.

Let's see now how to perform some workflow operations, including creating, saving, and deleting a workflow.

## Create a New Workflow Group

If you click on the Home Button:

- Click on the LOCAL workspace
- Select "Create Folder"

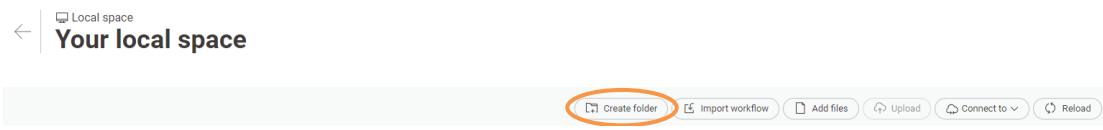



*Figure 2.1. Create new workflow group.*

In the "Create Folder" dialog:

- Enter the name of the workflow group

> ***Note.*** If you select an existing workflow group in KNIME Explorer, right-click, and start the "New KNIME Workflow Group Wizard", the default destination will be under the selected workflow group.

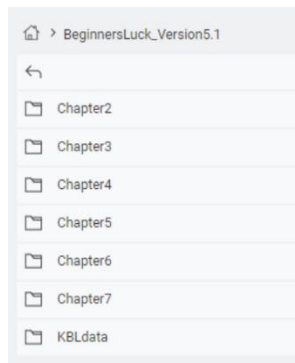



*Figure 2.2. Create a new workflow group named for every chapter.*

## Create a New Workflow

In the "Space Explorer" panel:

- Right-click anywhere in the LOCAL workspace (or in a Hub space)
- Select "New KNIME Workflow"

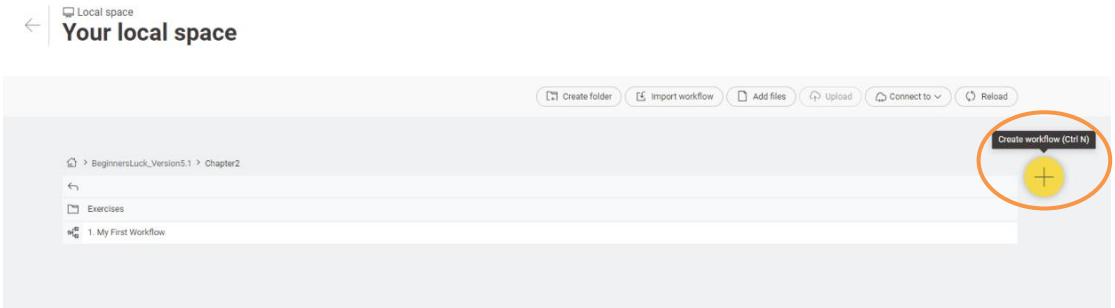



*Figure 2.3. Create new workflow.*

In the “New KNIME Workflow Wizard” dialog

- Enter the name of the new workflow
- Click “Create”

***Note.*** If you select an existing workflow group in KNIME Explorer, right-click, and start the “New KNIME Workflow Wizard”, the default destination will be in the selected workflow group.

To create a new workflow from the Space Explorer, click on the black button with three dots, and a window will pop up with multiple options like Create workflow, Create folder, Import workflow, and Connect to Hub.

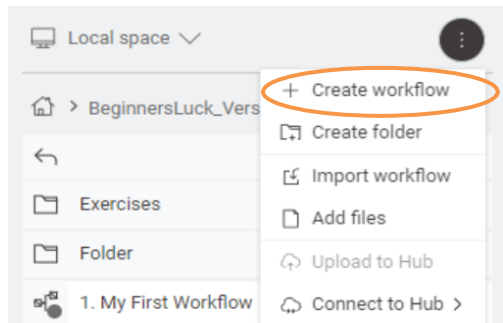



*Figure 2.4. Create a new workflow in Space Explorer named “My First Workflow” under “Chapter2”.*

## Save a Workflow

To save a workflow, click the disk icon in the toolbar. This either only saves the selected workflow open in the workflow editor or you can ***Save as*** the workflow with a different name as well. Saving the workflow saves the workflow architecture, the nodes’ settings, and the data produced at the output of each node.

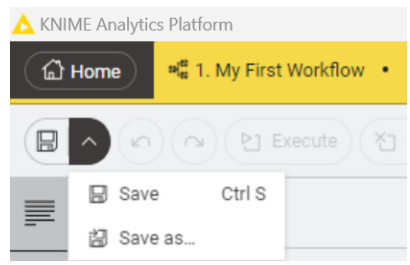



*Figure 2.5. “Save” options to save workflow.*

## Delete a Workflow

To delete a workflow

- Right-click the workflow in the "Space Explorer" panel
- Select "Delete"
- In the "Confirm Deletion" dialog, you will be asked if you really want to delete the workflow.

Beware! The "Delete" command removes the workflow project physically from the hard disk. Once it is deleted, there is no way to get it back.

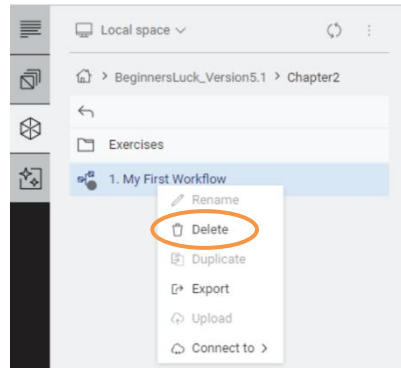



*Figure 2.6. Delete a workflow.*

# 2.2. Node Operations

In Chapter 1, we have seen that a node is the basic computational unit in a KNIME workflow. We have also seen that nodes are available, organized by categories, in the ***Node Repository*** panel in the side panel of the KNIME User Interface. And we have seen that every node has four states: not yet configured (red), configured (yellow), successfully executed (green), and executed with error (red with cross).

In this section we are going to explore:

- how to add a new node to a workflow (final status = inactive, not configured; red light),
- how to configure the node (final status = configured, not executed; yellow light), and
- how to execute the node (final status = successfully executed; green light).

## Create a New Node

To create a new node, you have two options:

- drag and drop the node from the ***Node Repository*** panel into the workflow editor
- double-click the node in the ***Node Repository*** panel

The node is usually imported with red traffic light status.

To connect a node with existing nodes, there are two more options:

- click the output port of the first node and release the mouse at the input port of the second node
- select a node in the workflow and double-click a node in the Node Repository: this creates a new node and automatically connects its first input port to the first output port of the existing node. Shift + double clicking the new node moves the connection to the next input port.
- In the Modern UI, clicking the node's output port gives the + in a dotted square symbol and shows all the compatible nodes to the current node. If you click one of the nodes from the panel, a connection is established between the two nodes.

Once the node has been created, we need to configure it, i.e. to set the parameters needed for the node task to be executed.

Let's then open the node configuration window and configure the node.

Finally, we need to associate a meaningful description to this node for documentation purposes, to easily recognize which task it is performing inside the workflow. Each node is created with a default text underneath as "Node n", where "n" is a progressive number. This node text can be customized. This, together with the workflow annotations described in chapter 1, keeps the overview of the workflow clear and fulfills the purpose of workflow documentation.

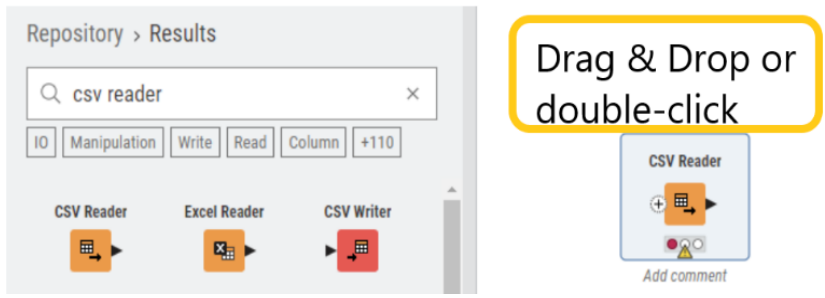



*Figure 2.7.Drag and drop or double-click the node to create a new node in the workflow editor.*

## Configure a Node

To configure an existing node:

- Double-click the node, OR
- Right-click the node and select "Configure"

If all input ports are connected, the configuration dialog appears for you to fill in the configuration settings. Every node has a different configuration dialog since every node performs a different task.

After a successful configuration, the node switches its traffic light to yellow.

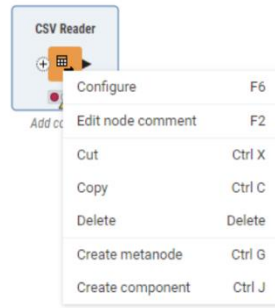



*Figure 2.8. Right-click the node and select "Configure" or double-click the node to configure.*

## Execute a Node

The node is now configured, which means it knows what to do. In order to actually make it perform its task, we need to execute it. To execute a node and let it run its task:

- Right-click the node & select "Execute", OR
- Select the node and click the single green arrow in the tool bar

If execution is successful, the node switches its traffic light to green.

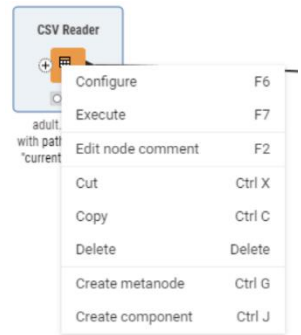



*Figure 2.9. Right-click the node and select "Execute" to run the node*

## Node Text

In order to change the text located under the node:

- Double-click the node text, so that it becomes editable
- Write the new text. The text can span more lines, separated by "Enter"
- Click outside the node to commit the text change


*Figure 2.10. Double-click the node name to edit it.*

## View the Processed Data

If the execution was successful (green light), you can inspect the processed data. You can either right-click on the node and select the open output port and in that, you can select the table which will open a new window with the output table or you will find the output table below the "workflow editor.

The output table shows the processed output and statistics of all the columns in the “Statistics” tab.

Some nodes might produce more than one output data set.

There is a tab beside the “Output Table” tab “Flow Variable”, which lists the flow variables. The concept of “Flow Variable” is explained in detail in the book KNIME Advanced Luck.

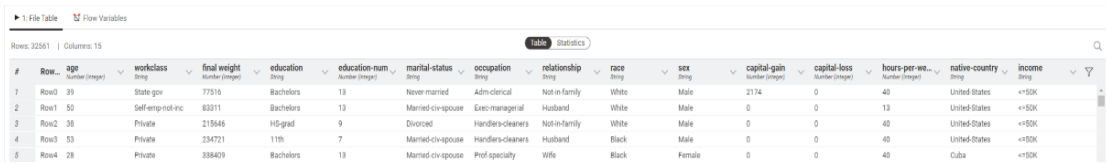



*Figure 2.11. Right-click the node and select the last option in the context menu to visualize the processed data.*

# 2.3. Read Data from a File

The first step in all data analytics projects consists of reading data. Local data is usually read from a file or from a database. In this chapter we describe how to read and write data from and to a text file. Reading and writing data from and to a database is described in Chapter 3 in section “Database Operations”.

Most common file format is the Comma Separated Values (CSV) format, that covers all text files where fields are separated by a special character. To read this kind of files of data, the CSV Reader node is the most versatile node in KNIME Analytics Platform.

## Create a *CSV Reader* Node

In the “Node Repository” panel in the bottom left corner:

- Expand the “IO” category and then the “Read” sub-category OR alternatively type “CSV Reader” in the search box
- Drag and drop the “CSV Reader” node into the workflow editor (or double-click it)
- If the “Description” panel on the right is enabled, it shows all you need to know about the “CSV Reader” node: task, output port, and required settings.
- To activate the “Description” panel, go to the Top Menu, open “View” and select “Description”.

***Note.*** Under the newly created "CSV Reader" node you might notice a little yellow warning triangle. If you hover over it with the mouse, the following tooltip appears: "No Settings available". This is because the CSV Reader node has not yet been configured (it needs at least the file path!). At the moment, the node is in the red traffic light state: not even configured.

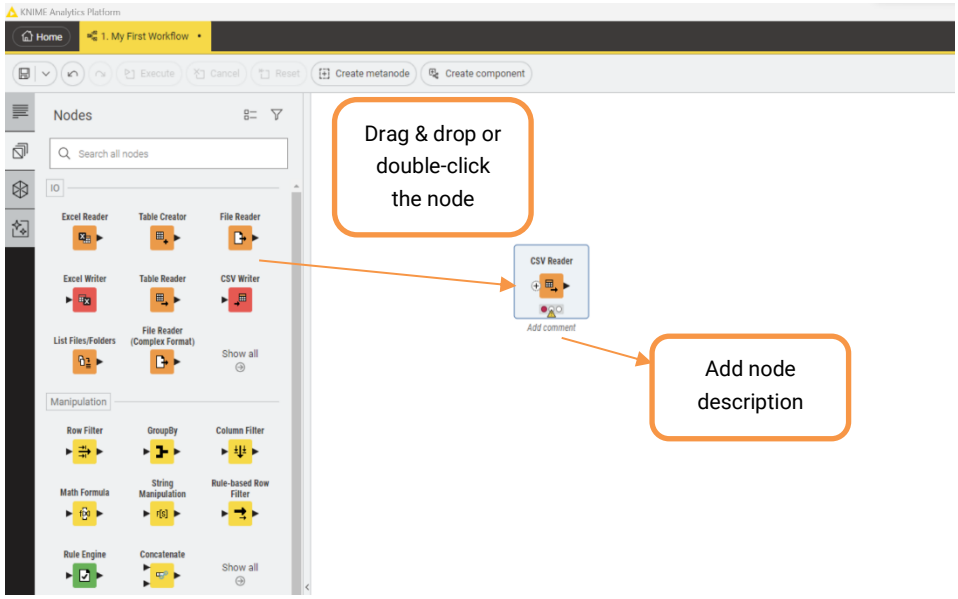



*Figure 2.12. Create a CSV Reader node.*

## Configure the *CSV Reader* Node

To configure the *CSV Reader* node,

- double-click the node, OR
- Right-click the node and select "Configure"

In the configuration dialog:

- Specify the file path, by typing or by using the "Browse" button. For this example, we used the adult.csv file, downloadable from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets/Adult) or available in KBLdata/adult data set/adult.csv.
- In most cases, the "CSV Reader" node automatically detects the file structure.

- If this is not one of most cases and the CSV Reader node has not guessed the file structure exactly, then tray with the button “Autodetect format” and/or enable/disable all required checkboxes in the tabs at the top, according to the file data structure.
- A preview of the read data is available in the lower part of the configuration window and shows possible reading errors.
- On the right of the OK/Cancel buttons in the lower part of the window, there is a small button carrying a question mark icon. This is the help button and leads to a new window containing the node description.

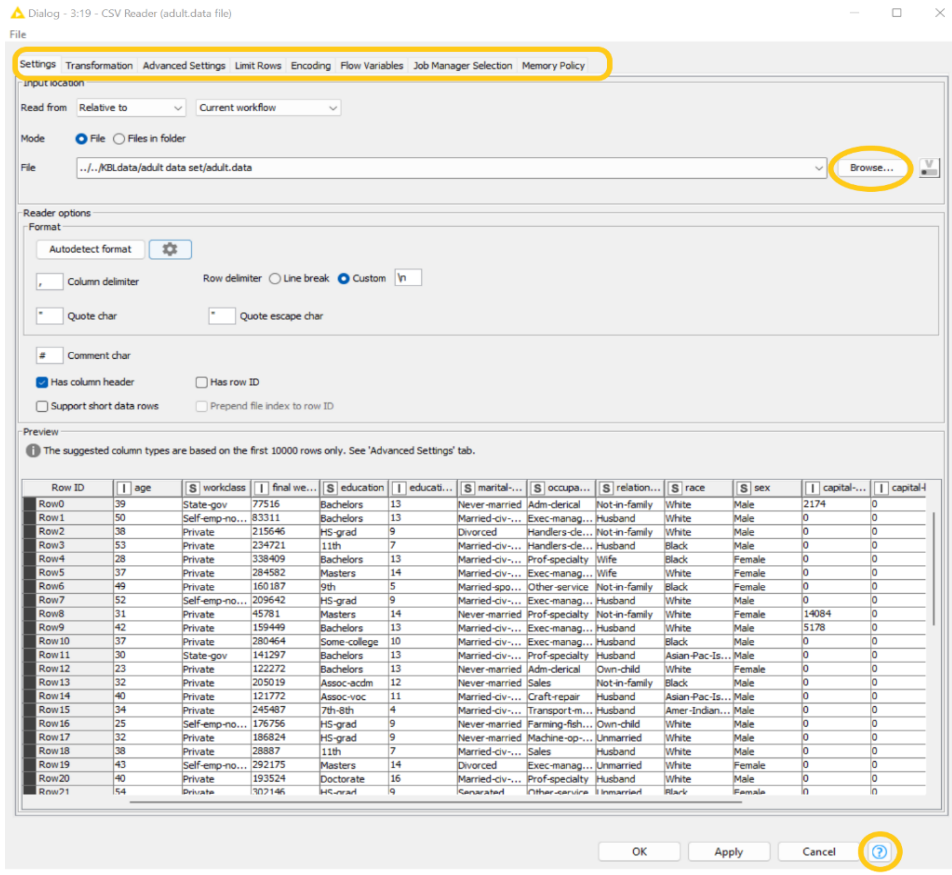



*Figure 2.13. Configuration window of the CSV Reader node.*

## Customizing Column Properties

It is possible to customize the way that each column data is read.

For example, the *adult.csv* file contains a field unclearly named “fnlwgt”. We can change this column header to a more meaningful “final weight” in the ***Transformation tab*** (Figure 2.14), just by writing the new name in the corresponding New Name field.

In the ***Settings tab*** (Figure 2.13), we can also set: the delimiter character that separates the column, whether to use the first data row for the column headers, whether to use the first column as RowIDs, and if we can tolerate shorter data rows.

The ***Limit Rows tab*** allows to skip the first lines in the text file. This is usually very useful when dealing with files with a header text.

The ***Advanced Settings tab*** offers a few additional settings, and the ***Encoding tab*** allows to set the right encoding for the text.

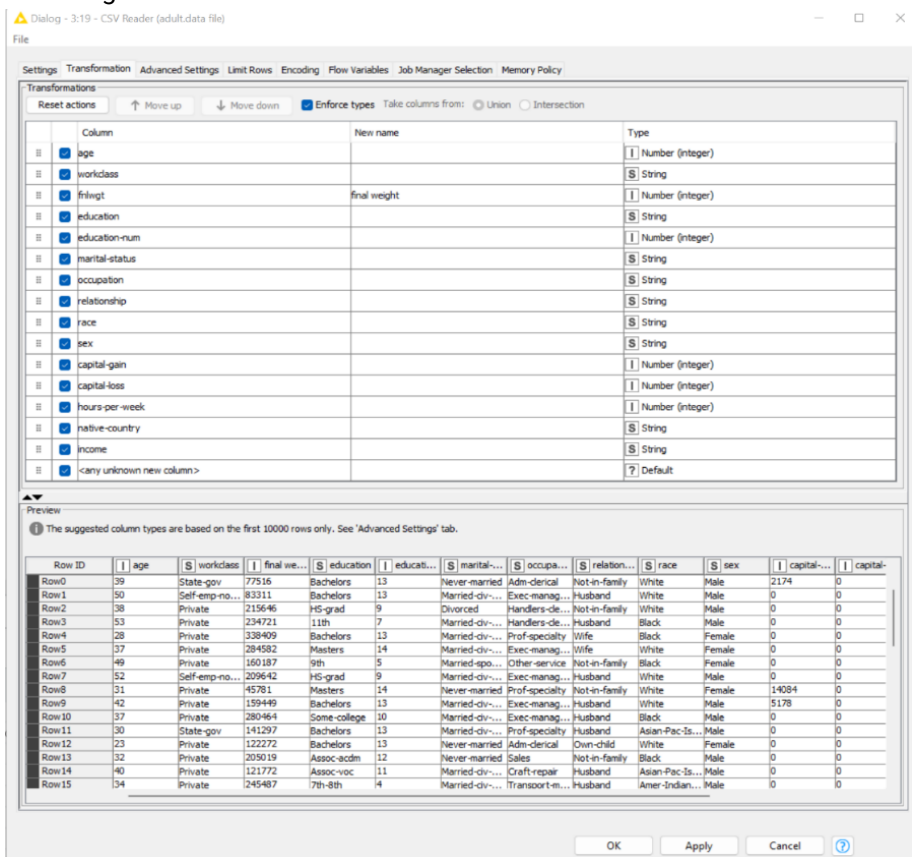



*Figure 2.14. Customize how to read columns.*

Notice the three dots in the lower left corner of the CSV Reader icon. These dots indicate dynamic port. By clicking on the three dots, you can add one or more input ports to this node

to connect to an external file system, like Amazon S3, HDFS, Databricks, Micrsoft Azure, and so on …

The three dots in a node always mean dynamic ports.

> ***Note.*** After configuring the *CSV Reader* node, its state moves to the yellow traffic light.

What we have described here is the long way to get a *CSV Reader* node configured! Since this (node creation and configuration) is a way that works for all nodes, we could not avoid going through it. However, if the file has a known extension, such as *.csv* or *.txt*, there might be a faster way.

> ***Note.*** Instead of manually writing the full file path in the Input Location/File field in the CSV Reader configuration window, we could just drag and drop the file from the Space Explorer panel into the workflow editor. If the file has a known extension (like .csv for example), this automatically creates the appropriate reader node and configures it.

Notice that when you create a *CSV Reader* this way, you will get a different protocol in the URL box. By browsing to a file, you get a *file://* protocol; by drag and drop you get a *knime://* protocol (Figure 2.15). This special protocol is paired with the option “Custom/KNIME URL” in the Read From field.


*Figure 2.15. Automatic configuration of the Input Location part in the Settings tab of a CSV Reader node created via drag & drop of a file from the KNIME Explorer panel into the workflow editor panel.*

Other options available to locate a file in the Input Location part of the configuration window of a CSV Reader node are from *Local File System*, from *a specific Mountpoint*, from *a relative location*, and *the Custom/KNIME URL* that we have already seen. The first three options define the starting point for the file path.

- “Local File System” treats the file path as an absolute path.
- “Mountpoint” defines your path as a relative path to the Mountpoint selected in the secondary menu on the right, that is: LOCAL as the current workspace folder, My KNIME Hub (you need to be logged in to see the option), another workspace, or the EXAMPLES.

- "Relative to" defines the path of the file relatively to a given location, as specified in the secondary menu on the right: the current workflow folder, the data folder within the current workflow folder, current workspace, and current mountpoint.
- "Custom/ KNIME URL" refers to the usage of the *knime://* protocol

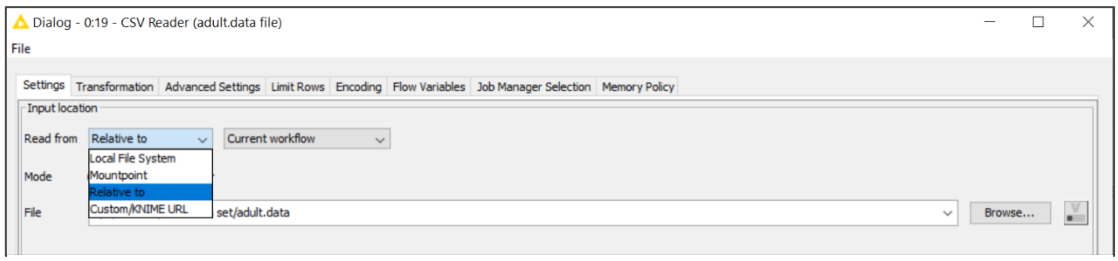



*Figure 2. 16. Options available in the "Read from" field in the Input Location part in the Settings tab of a CSV Reader*

The Browse button can help you find the file while the "Read from" option will write it in the right format.

Notice that in Reader and Writer nodes that still do not have the "read from" field, the *knime://* protocol still works. Let's have a look at the different *knime://* protocol options.

## The *knime://* Protocol

The *knime://* protocol is a special protocol that allows you to reference the local workspace or the local workflow in a path. This then permits the creation of relative paths making you independent of the workspace folder absolute URL, but just dependent on the workspace folder structure.

This feature is particularly useful when moving workflows around to other workspaces or even to other machines. As long as the folder structure of data and workflow is preserved, the *CSV Reader* will keep finding the file and reading it.

The *knime://* protocol works on all reader and writer nodes.

We also need to assign this node a meaningful comment so that we can easily recognize what its task is in the workflow. The default comment under our *CSV Reader* is "Node 1" because it was the first node we created in the workflow. In order to change the node's comment:

- Double-click the "Node 1" label under the *CSV Reader* node
- Enter the node's new comment (for example "Adult data set")
- Click elsewhere

| | |
|---|---|
| knime://LOCAL/ | refers to the current workspace location |
| knime://LOCAL/../../knime-workspace | moves two levels up from the current workspace location to a new workspace folder named knime.workspace |
| knime://knime.workflow/ | refers to the current workflow location |
| knime://knime.workflow/../../data | moves two levels up from the current workflow location to a new folder named data |
| knime://<knime-mountID>/ | refers to a KNIME Server available in the KNIME Explorer panel |
| knime://<knime-mountID>/<path>/data | moves to the <path>/data folder on the referenced KNIME Server |

*Figure 2.17. Possible paths using knime:// protocol.*

We have now changed the comment under the *CSV Reader* node from "Node 1" to "Adult data set with *file:// protocol*".

***Note.*** Notice the three dots in the lower left corner of the node. Clicking on them allows you to connect to an external file system and access files from there.

After configuration, in order to make the node really read the file, we need to execute it. Thus, proceed as follows:

- Right-click the node
- Select "Execute"

***Note.*** If the reading process has no errors, the node switches its traffic light to green.

***Note.*** On every configuration window you will find a tab, called "Flow Variables". Flow Variables are used to pass external parameters from one node to another. However, we are not going to work with Flow Variables in this book, since they belong to a more advanced course on KNIME functionalities.

The "IO" → "Read" category in "Node Repository" contains a number of additional nodes to read files in different formats, like Excel, CSV, KNIME proprietary format, and more. The "IO"/"File Handling" category has nodes to read special formats and special files, like for example ZIP files, remote files, etc.

## 2.4. KNIME Data Structure and Data Types

If the node execution was successful, you can now see the resulting data.

- Right-click the “CSV Reader” node
- Select option “File Table”

A table with the read data appears. Let’s have a look at this table to understand how data is structured inside KNIME. First of all, data in KNIME is organized as a **table**. Each row is identified by a **Row ID**. By default, Row IDs are strings like “Row n” where “n” is a progressive number. But RowIDs can be forced to be anything, with the only condition that they must be unique. Not unique RowIDs produce an error.

Columns are identified by column headers. If no column headers are available, default column headers like “Col n” – where “n” is a progressive number – are assigned. In adult.data file column headers were included. We enabled the checkbox “Read column headers” in the configuration window of the *CSV Reader* node and we now have a header for each column in the final data table. Even column headers need to be unique. If a column header occurs more than once, KNIME Analytics Platform adds a suffix “(#n)” (n = progressive number) to each multiple occurrences of the column header.

Each column contains data with a set data type. Available data types are:

- Double (“D”)
- Integer (“I”)
- String (“S”)
- Date&Time (calendar + clock icon)
- Unknown (“?”)
- Other specific domain related types (like *Document* in the text processing extension, *Image* in the image processing extension, or *Smiles* in chemistry extensions)

Date&Time type can come from importing data from a database. It does not appear from reading data from a file. In text files, dates and times are read just as strings. You then need a *String to Date&Time* node to convert a String into a *Date&Time* type column.

Unknown type refers to columns whose type could not be determined, like for example with mixed data types or with all missing values.

Missing values are data cells with a special "missing value" status and are displayed by default with a question mark ("?"), unless the display character for the missing values was set otherwise in the CSV Reader node configuration.

> ***Note.*** Missing values are represented by default with question marks. They are not question marks, they are missing and are represented with question marks. Question marks in the text file are correctly read as question marks, but they are not missing data. Missing values could be represented by anything else as defined in the configuration window of the *CSV Reader* node.

## KNIME Data Structure

Data in KNIME are organized as a table with a fixed number of columns. Each row is identified by a **Row ID**. Columns are identified by column headers. Each column represents a data type:

- Double ("D")
- Integer ("I")
- String ("S")
- Date&Time (calendar + clock icon)
- Unknown ("?")
- Other domain related types

Clicking the header of a data column allows to sort the data rows in an ascending / descending order. Right-clicking the header of a data column allows to visualize the data using specific renderers. For Double/Integer data, for example, the “Bars” renderer displays the data as bars with a proportional length to their value and on a red/green heatmap.

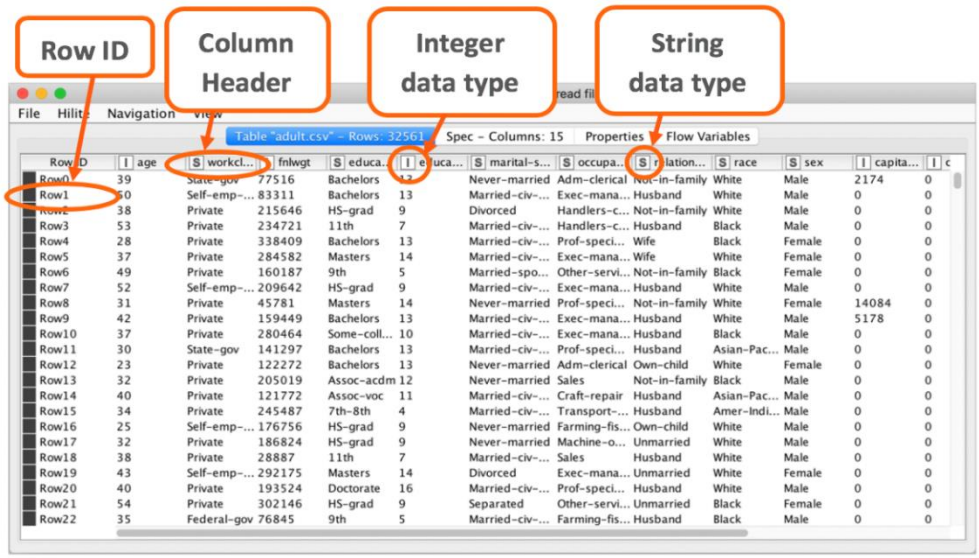



*Figure 2.18. The KNIME Data Structure.*

You can temporarily sort the data by clicking the column header and select the kind of sorting. You can temporarily change the data renderer by right-clicking the column header and change to a different renderer either numerical or bar based. Both those operations are just temporary. If you close the data table and reopen it, the default window will be back.

## 2.5. Filter Data Columns

In the next step, we want to filter out the column “final weight” from the read data set. In the “Node Repository” panel, on the left, there is a whole category called “Manipulation” with nodes dedicated to managing the data structure. This category includes operations on columns, rows, and on the full data matrix.

### Create a *Column Filter* Node

- In the “Node Repository” panel, find the node “Column Filter” under “Manipulation” → “Column” → “Filter” or search for “Column Filter” in the search box.

- Drag and drop the “Column Filter” node from the “Node Repository” to the workflow editor or double-click it in the “Node Repository”
- The description for this node appears in the “Node Description” panel on the right
- Connect the “Column Filter” node with the previous node (in our workflow, the “CSV Reader” node) by clicking at the output of the “CSV Reader” node and releasing at the input of the “Column Filter” node.

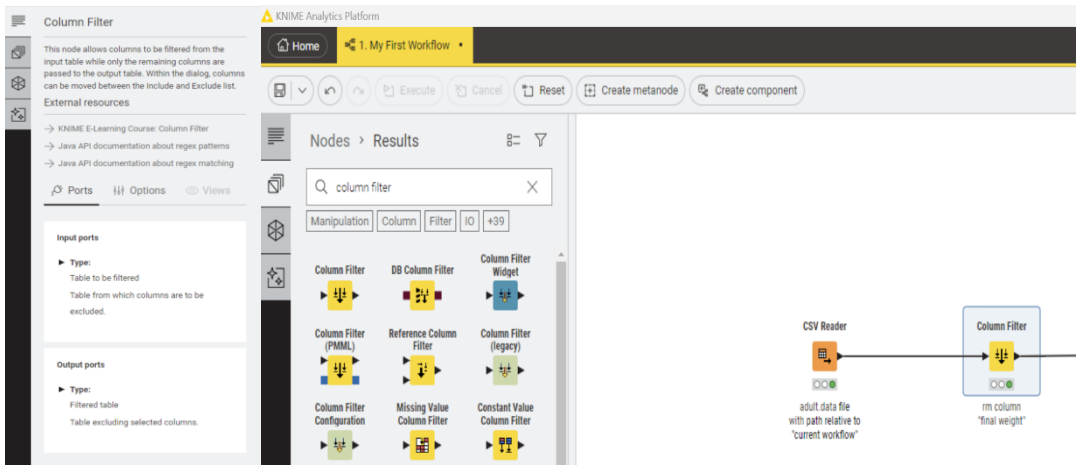



*Figure 2.19. Creating a Column Filter node.*

To configure the node:

- Double-click the node or right-click the node and select “Configure”
- The configuration window opens. The node’s configuration window contains all settings for that particular node.
- Set the node configuration settings
- Click “OK”

## Configure the *Column Filter* Node

The first setting in the configuration window is the type of filtering. You can select and retain columns **manually**, **by type**, or **by name**, according to the options at the top of the configuration window (Figure 2.20).

## Manual Selection:

If the “Manual Selection” option is selected, the configuration window shows 2 sets of columns (Figure 2.20):

- The columns to be included in the data table (“Includes” set on the right)
- The columns to be excluded from the data table (“Excludes” set on the left)

The “**Search**” bar allows to search for a specific column.

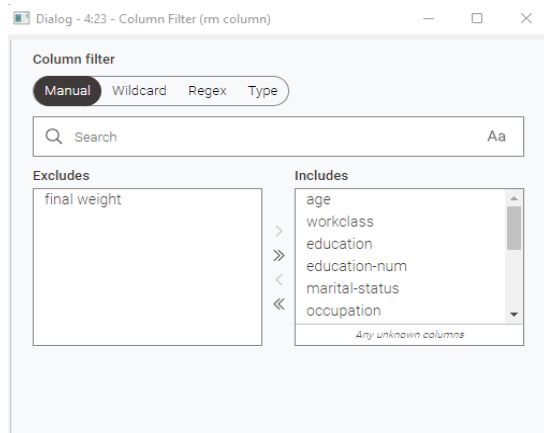



*Figure 2.20. Configuration window of the Column Filter*

You can add and remove columns from one set to the other on double click.

- “Includes” keeps the “Include” set fixed. If one more input column is added from the previous node, this new column is automatically inserted into the “Exclude” set.
- “Excludes” keeps the “Exclude” set fixed. If one more input column is added from the previous node, this new column is automatically inserted into the “Include” set.

## Wildcard/Regex Selection:

In case the “Wildcard/Regex Selection” option is enabled, the configuration window presents a textbox to edit the desired wildcard or regular expression.

Columns with names matching the expression will be included in the output data table.

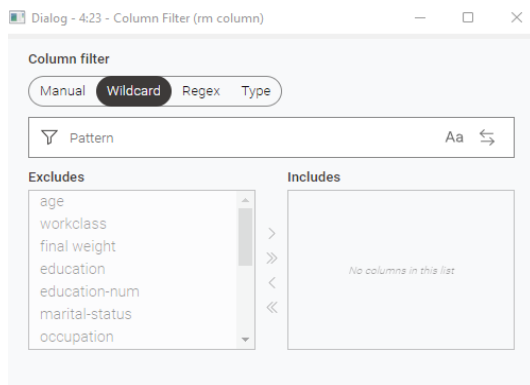



*Figure 2.21. Column Filter node Configuration: "Wildcard/Regex Selection".*

## Type Selection:

If the “Type Selection” option is enabled, you are presented with a series of checkboxes about the types of columns to keep in the output data table.

Selecting all Numbers checkboxes, for example, **Number(integer)** will keep all numerical columns only in the node’s output table.

Selected String will keep all String type columns only in the output table.

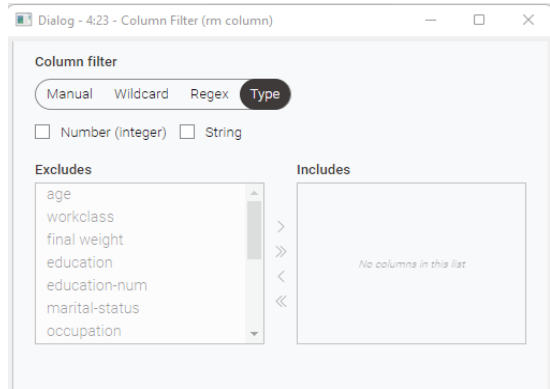



*Figure 2.22. Column Filter node configuration: "Type.*

Remember this column selection frame that comes with the “Manual Selection” option, because it will show up again in all those nodes requiring column selection.

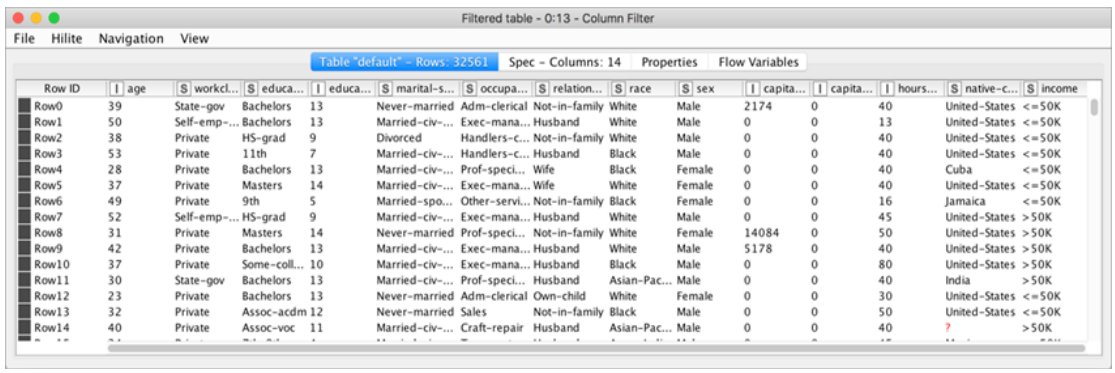
In our example workflow “my First Workflow”, we wanted to remove the “final weight” column.

We set the column filter mode to “Manual Selection” and we populated the Exclude panel with column “final weight”.

After completing the configuration, we right-clicked the “Column Filter” node and commented it with “rm column ‘final weight’”.

We finally right-clicked the node and selected “Execute” to run the column filter.

To see the final processed data, we clicked the “rm column “final weight”” node and see the “Filtered Table” in the Node Monitor. The column “final weight” was not to be found in the Column Filter’s output data table.




| Row ID | age | workcl... | educa... | educa... | marital-s... | occupa... | relation... | race | sex | capita... | capita... | hours... | native-c... | income |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row0 | 39 | State-gov | Bachelors | 13 | Never-married | Adm-clerical | Not-in-family | White | Male | 2174 | 0 | 40 | United-States | <=50K |
| Row1 | 50 | Self-emp-... | Bachelors | 13 | Married-civ-... | Exec-mana... | Husband | White | Male | 0 | 0 | 13 | United-States | <=50K |
| Row2 | 38 | Private | HS-grad | 9 | Divorced | Handlers-c... | Not-in-family | White | Male | 0 | 0 | 40 | United-States | <=50K |
| Row3 | 53 | Private | 11th | 7 | Married-civ-... | Handlers-c... | Husband | Black | Male | 0 | 0 | 40 | United-States | <=50K |
| Row4 | 28 | Private | Bachelors | 13 | Married-civ-... | Prof-speci... | Wife | Black | Female | 0 | 0 | 40 | Cuba | <=50K |
| Row5 | 37 | Private | Masters | 14 | Married-civ-... | Exec-mana... | Wife | White | Female | 0 | 0 | 40 | United-States | <=50K |
| Row6 | 49 | Private | 9th | 5 | Married-spo... | Other-servi... | Not-in-family | Black | Female | 0 | 0 | 16 | Jamaica | <=50K |
| Row7 | 52 | Self-emp-... | HS-grad | 9 | Married-civ-... | Exec-mana... | Husband | White | Male | 0 | 0 | 45 | United-States | >50K |
| Row8 | 31 | Private | Masters | 14 | Never-married | Prof-speci... | Not-in-family | White | Female | 14084 | 0 | 50 | United-States | >50K |
| Row9 | 42 | Private | Bachelors | 13 | Married-civ-... | Exec-mana... | Husband | White | Male | 5178 | 0 | 40 | United-States | >50K |
| Row10 | 37 | Private | Some-coll... | 10 | Married-civ-... | Exec-mana... | Husband | Black | Male | 0 | 0 | 80 | United-States | >50K |
| Row11 | 30 | State-gov | Bachelors | 13 | Married-civ-... | Prof-speci... | Husband | Asian-Pac... | Male | 0 | 0 | 40 | India | >50K |
| Row12 | 23 | Private | Bachelors | 13 | Never-married | Adm-clerical | Own-child | White | Female | 0 | 0 | 30 | United-States | <=50K |
| Row13 | 32 | Private | Assoc-acdm | 12 | Never-married | Sales | Not-in-family | Black | Male | 0 | 0 | 50 | United-States | <=50K |
| Row14 | 40 | Private | Assoc-voc | 11 | Married-civ-... | Craft-repair | Husband | Asian-Pac... | Male | 0 | 0 | 40 | ? | >50K |

*Figure 2.23. The column filtered table does not contain the column "finale weight".*

# 2.6. Filter Data Rows

If you have had a deeper look into the data we are currently analyzing, you have seen that each record describes a person in terms of age, job, education, and other general demographic information. We have seen how to remove a data column from a data table. Let's see now how to exclude data rows from a data table.

Let's suppose that we want to retain all records of people born outside of the United States. That is, we want to retain only those rows with "native-country" other than "United States". We need to use a Row Filter node.

## Create a *Row Filter* Node

In the "Node Repository" panel, open the node category "Manipulation" and navigate to the node "Row Filter" in "Manipulation" → "Row" → "Filter" or search for "Row Filter" in the search box.

Drag and drop or double-click the "Row Filter" node in the "Node Repository" to create a new instance in the workflow editor panel.

The task and settings description for this node can be found in the "Node Description" panel on the right or clicking the help button in the configuration window at the right of the "Cancel" button.

Connect the "Row Filter" node with the "Column Filter" node previously created.

## Configure the *Row Filter* Node

Double-click the "Row Filter" node to open its configuration window.

The node implements three filter criteria:

- Select rows by attribute value (pattern matching)
    - Value matching: column value matching some pre-defined pattern value (wild-cards and regular expression are allowed in pattern definition)
    - Range checking for numerical columns: column value above or below a given value
    - Missing Value Matching
- Select rows by row number

- Select rows by RowID (pattern matching on RowID)

Each of these criteria can be used to include or to exclude rows.

- Implement your row filter criterion
- Click “OK”

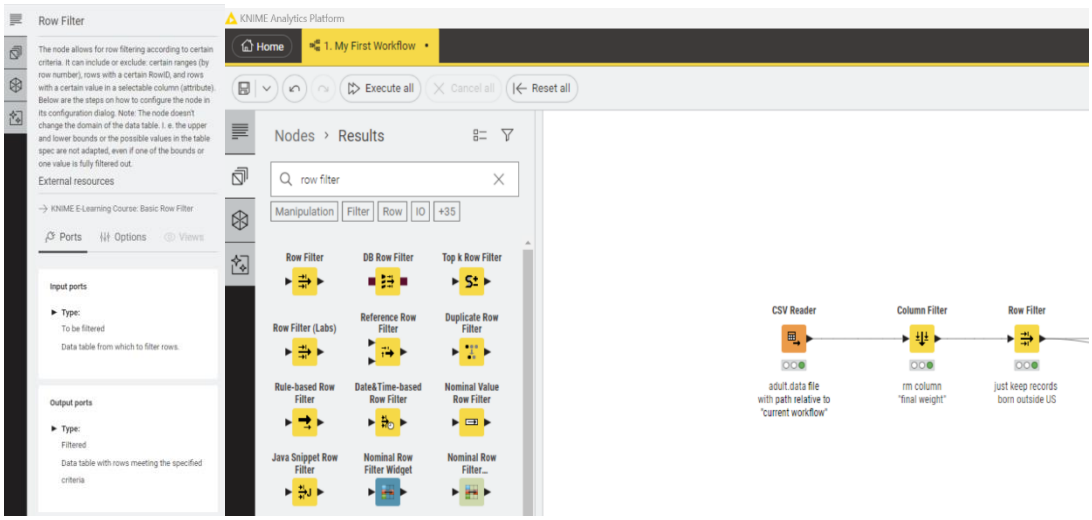



*Figure 2.24. Creating a Row Filter node.*

Below you can find a more detailed description of the row filter criteria available in the Row Filter node configuration.

The Row Filter node is not the only way to perform row filtering in KNIME, even though probably it is the easiest one and works for 80% of your row filtering needs. Other row filtering options are offered by:

- “Nominal Value Row Filter” node for multiple pattern matching in “OR mode” (example: native-country = “United Stated” OR native-country=”Canada” OR native-country=”Puerto Rico”);
- “Rule Based Row Filter” node to define an arbitrarily complex set of IF-THEN row filtering rules, even spanning multiple columns;
- “Geo-Coordinate Row Filter” node in KNIME Labs category for row filtering based on geographical coordinates;
- “Date&Time-based Row Filter” node to perform a row filtering on a Date&Time type column;

- "Database Row Filter" node to implement a row filtering SQL query to run directly on the database.

## Row Filter Criteria

### By Attribute Value:

All rows, for which the value in a given column matches a pre-defined pattern, are filtered out or kept. After you select the "column to test", you need to define the matching mode.

For **String/Integer/Date&Time** values, "use pattern matching" requires the given pattern to be either entered manually or selected from a menu populated with the column values as possible pattern values. A matching value with wildcards * (for example "United*") or with a regular expression is also possible.

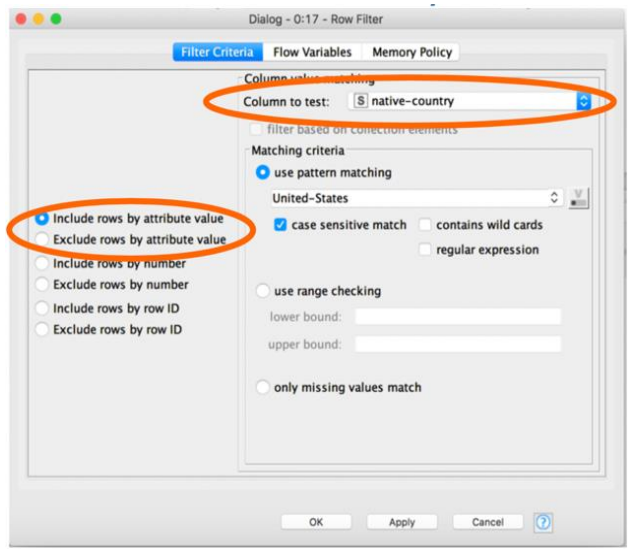



*Figure 2.25. Row Filter criterion by attribute value.*

For **Integer** values, "use range checking" requires a lower boundary and/or an upper boundary, which will coincide if the condition is equality.

For **Missing** values, choose the last matching option.

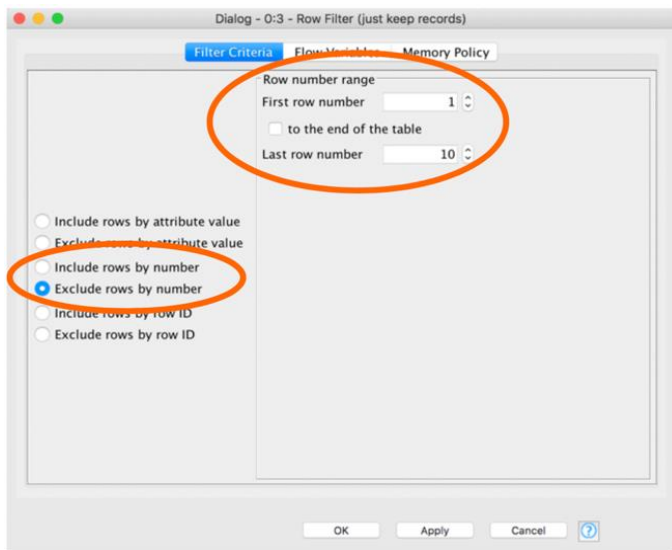



*Figure 2.26. Row Filter criterion by row number.*

### By Row Number:

If you know where your desired or undesired rows are, you can just

enter the row number range to be filtered out.

For example, if I know that the first 10 rows are comments or just garbage, I would select the filter criterion “exclude row by number” and set the row number range 1-10.

### By RowID:

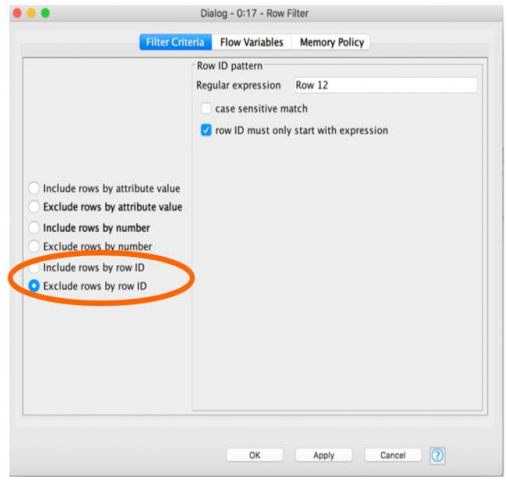



*Figure 2.27. Row Filter criterion by RowID.*

A special row filter by attribute value runs on the RowIDs.

Here the matching pattern is given by a regular expression. The regular expression has to match the whole RowID or just its beginning.

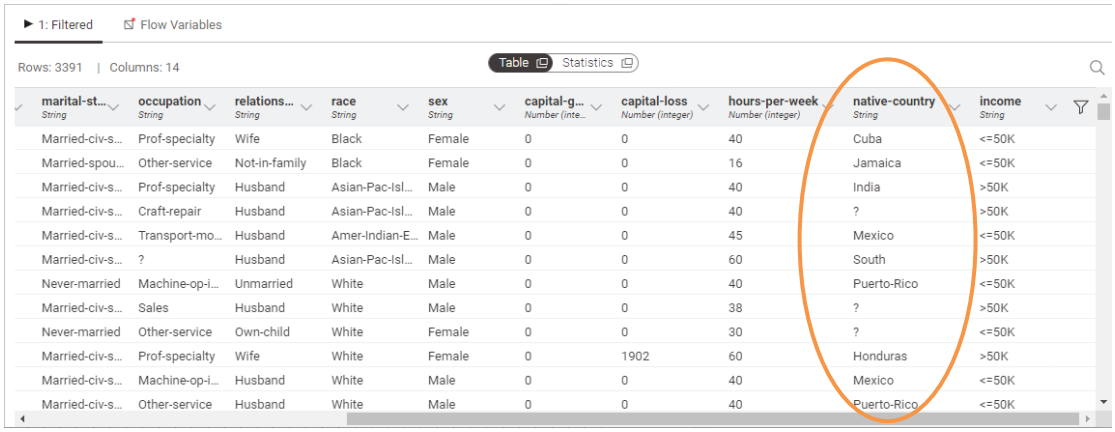
In order to retain all rows with data referring to people born outside of the United States, we need to:

- Set filter mode “exclude row by attribute value”
- Set the column to test to “native-country”
- Enable “use pattern matching”, because it is a string comparison
- Set pattern “United-States”

We have just implemented the following filter criterion: `native-country != "United States"`

- Give the *Row Filter* node a meaningful comment. We commented it with “just keep records born outside US”. The comment on a node is important for documentation purposes. Since KNIME is a graphical tool, it is easy to keep an overview of what a workflow does, if the name of each node gives a clear indication of its task.
- Right-click the node and select “Execute” to run the row filter

To see the final processed data, right-click the node “born outside US” and see the filtered table below in the Node Monitor. There should be no “United States” in column native-country.




| marital-st... String | occupation String | relations... String | race String | sex String | capital-g... Number (inte... | capital-loss Number (integer) | hours-per-week Number (integer) | native-country String | income String |
|---|---|---|---|---|---|---|---|---|---|
| Married-civ-s... | Prof-specialty | Wife | Black | Female | 0 | 0 | 40 | Cuba | <=50K |
| Married-spou... | Other-service | Not-in-family | Black | Female | 0 | 0 | 16 | Jamaica | <=50K |
| Married-civ-s... | Prof-specialty | Husband | Asian-Pac-Isl... | Male | 0 | 0 | 40 | India | >50K |
| Married-civ-s... | Craft-repair | Husband | Asian-Pac-Isl... | Male | 0 | 0 | 40 | ? | >50K |
| Married-civ-s... | Transport-mo... | Husband | Amer-Indian-E... | Male | 0 | 0 | 45 | Mexico | <=50K |
| Married-civ-s... | ? | Husband | Asian-Pac-Isl... | Male | 0 | 0 | 60 | South | >50K |
| Never-married | Machine-op-i... | Unmarried | White | Male | 0 | 0 | 40 | Puerto-Rico | <=50K |
| Married-civ-s... | Sales | Husband | White | Male | 0 | 0 | 38 | ? | >50K |
| Never-married | Other-service | Own-child | White | Female | 0 | 0 | 30 | ? | <=50K |
| Married-civ-s... | Prof-specialty | Wife | White | Female | 0 | 1902 | 60 | Honduras | >50K |
| Married-civ-s... | Machine-op-i... | Husband | White | Male | 0 | 0 | 40 | Mexico | <=50K |
| Married-civ-s... | Other-service | Husband | White | Male | 0 | 0 | 40 | Puerto-Rico | <=50K |

*Figure 2.28. The row filtered table has no pattern "United States" in column "native-country".*

# 2.7. Write Data to a File

Now we want to write the processed data table to a file. There are many nodes that can write to a file. Let's choose the easiest and most standard format for now: the CSV (Comma Separated Values) format.

## Create *CSV Writer* Node

In the "Node Repository":

- Expand category "IO"/"Write" or search for "CSV Writer" in the search box
- Drag and drop (or double-click) the node "CSV Writer" to create a new node in the workflow editor
- Right-click the "CSV Writer" node and select "Configure" or double-click it to open its configuration window.
- Configuration window has four tabs: Settings, Advanced Settings, Comment Header, and Encoding

The configuration window of the CSV Writer node is similar to the configuration window of the CSV Reader node.

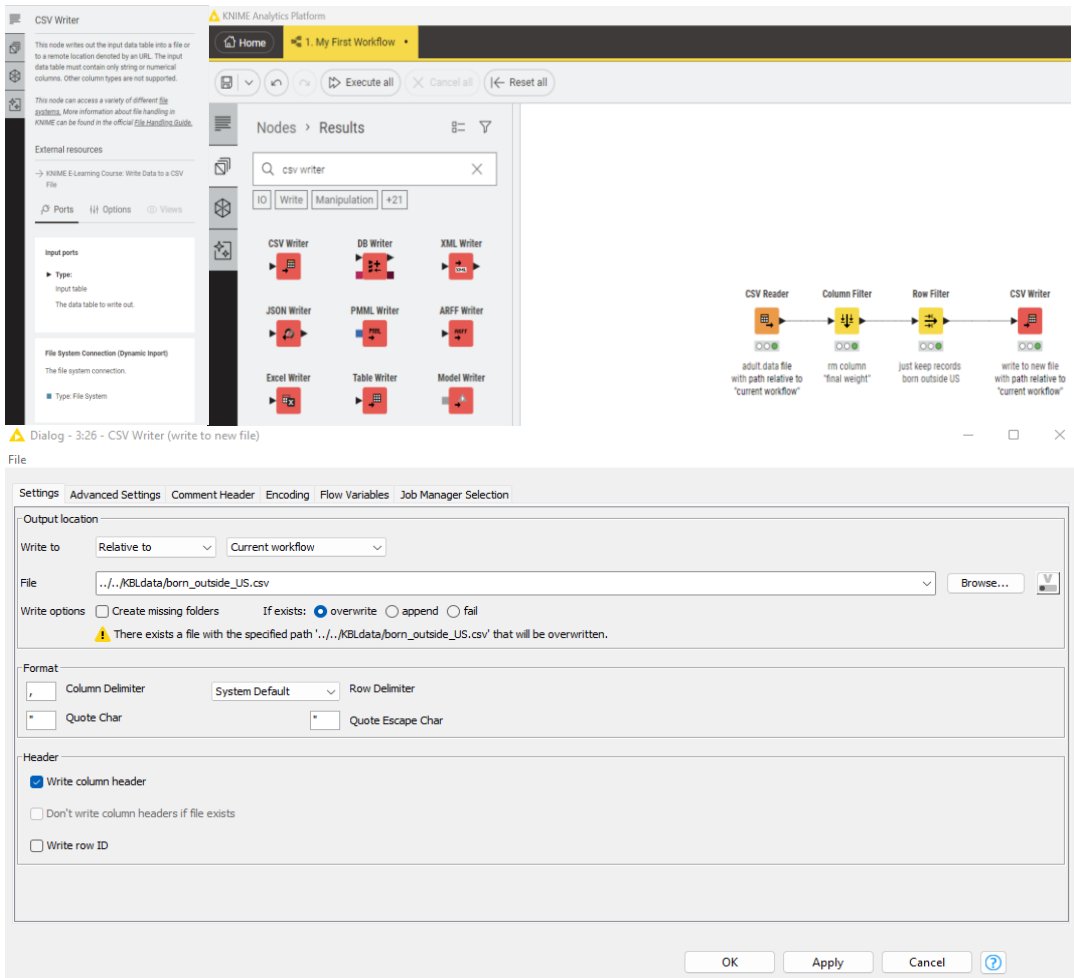



*Figure 2.29. Create and configure a "CSV Writer" node.*

## Configure the *CSV Writer* Node

The "**Settings**" tab is the most important tab of this configuration window. It requires:

- The path of the output file in Output Location. Notice that Output Location offers the same options as the Input Location in the CSV Reader node.
- A few additional options about the data structure, such as:
  - Whether to write the column headers and/or RowID in output file
  - The Column Delimiter character

- The writing mode if file already exists: Overwrite, Append, Fail (does not write to file)

The **Advanced Settings tab** allows for a few more specs, like the quote character, the decimal separator, or the compression to a gzip file. The **Comment Header tab** allows to automatically write a header with comments on top of the data. The **Encoding tab** chooses the appropriate encoding for the text. The **Memory Policy** tab contains a few options that might speed up the node execution. This tab is common to the configuration window of all nodes.

In this book we do not investigate the **Flow Variables** tab and the **Job Manager Selection** tab.

> ***Note.*** Writing in mode "Append" can be tricky, because it just appends the new data to an existing file without checking the data structure nor matching the column by name. So, if the data table structure has changed, for example because of new or deleted columns, the output CSV file will not be consistent anymore.

In some cases, you might want to select "Fail" as over-writing mode, in order to avoid overwriting the existing file.

- Let's now change the node's comment:
- Click the node label under the node
- Enter the node's new comment (for example "write new file")
- Click elsewhere
- Right-click the node and select "Execute"

You can also make the node comments more verbose if you want to add more information about the node settings and implemented task. At this point we also add a few annotations to make even clearer what each node or group of nodes does.

We have created our first workflow to read data from a file, reorganize rows and columns, and finally write the data to an output file.

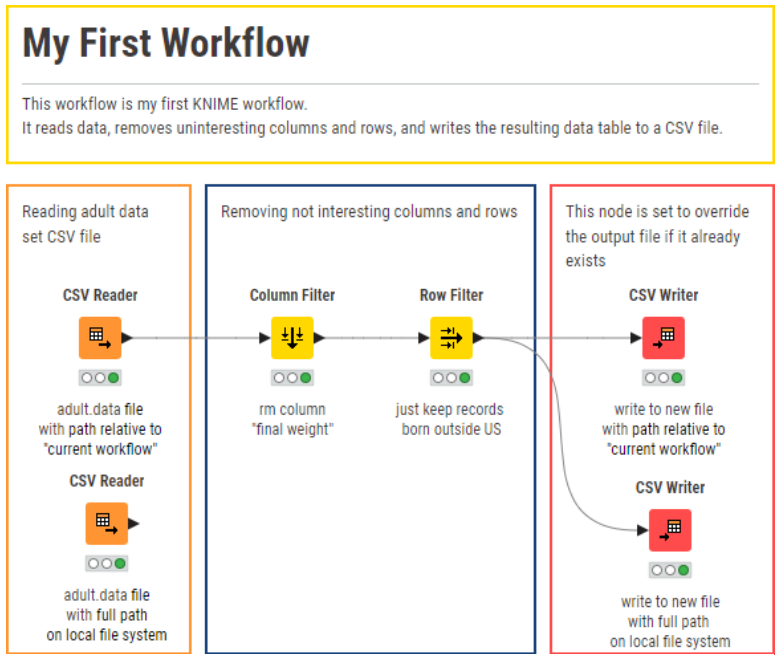



*Figure 2.30. Workflow "My First Workflow".*

# 2.8. Exercises

## Exercise 1

In a workflow group “Exercises” under the existing workflow group “Chapter2” create an empty workflow “Exercise1”. Workflow “Exercise1” should perform the following operations:

- Read file data1.txt (from the KBLdata folder) with column “ranking” as String and named “marks”;
- Remove initial comments from data read from file;
- Remove column “class”
- Write final data to file in CSV format (for example with name “data1_new.csv”) using character “;” as separator

Enter a short description for all nodes in the workflow. Save and execute workflow “Exercise1”. Execution must be without errors (green lights for all nodes).

## Solution to Exercise 1

The file has some comments at the very beginning, which of course do not have the same length as the other lines in the file.

First, you need to enable the options “has column headers” and “support short data rows” in the “Settings” tab and rename column “ranking” as “marks” in the “Transformation” tab.

Then you can do one of the two approaches:

1. Set the “Skip first data rows …” to 5 in the “Limit Rows” tab.
2. Use the “Row Filter” node to exclude the first 5 rows of the read data.

Again, the solution workflow is available in the folder of workflows you downloaded from the KNIME Community Hub.

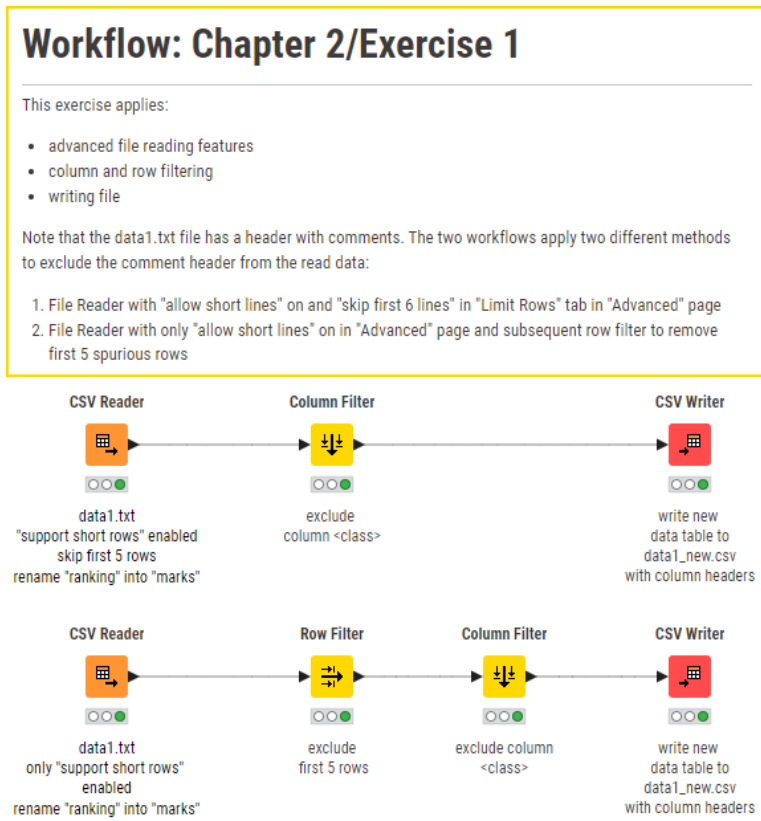



*Figure 2.31. Exercise 1: Two possible solution workflows.*

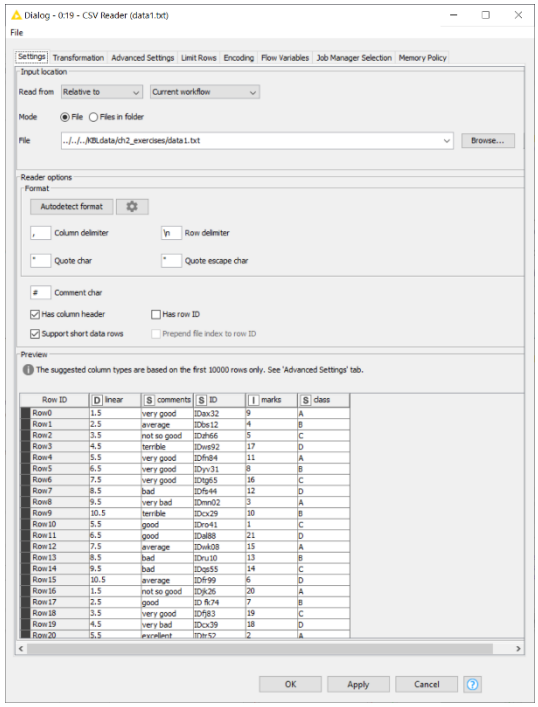



*Figure 2.32. Exercise 1: CSV Reader "Settings" tab configuration.*

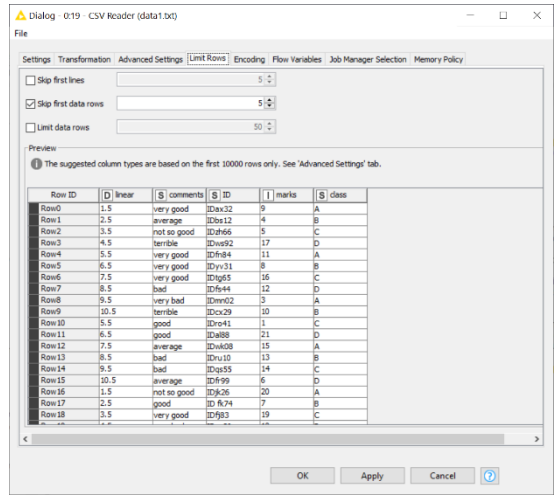



*Figure 2.33. Exercise 1: CSV Reader "Limit Rows" tab configuration.*

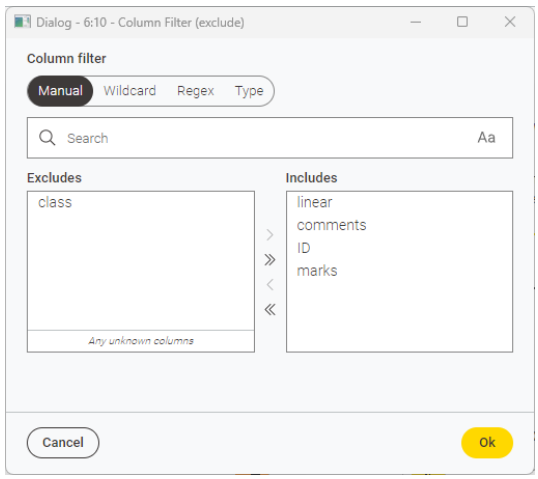



*Figure 2.34. Exercise 1: Column Filter configuration.*

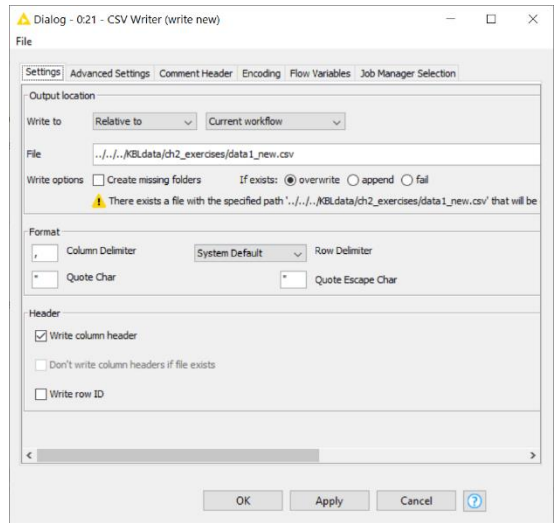



*Figure 2.35. Exercise 1: CSV Writer "Settings" tab configuration.*

## Exercise 2

In the workflow group “Chapter2\Exercises” create a workflow “Exercise2” to perform the following operations:

- Read the CSV file written in Exercise1 (“data1_new.csv”) and rename column “marks” to “ranking”
- filter out rows with value ‘average’ in the ‘comments’ column
- Exclude Integer type columns
- Write final data to file in “Append” mode and with a tab as a separating character
- Rename all nodes where necessary. Save and execute workflow “Exercise2”.

### Solution to Exercise 2

We recycled the workflow structure from the workflow created in Exercise 1. That is, we did a “Copy and Paste” operation (Ctrl-C, Ctrl-V) on the whole “Exercise 1” workflow from the workflow editor for “Exercise 1” into the workflow editor for “Exercise 2”.

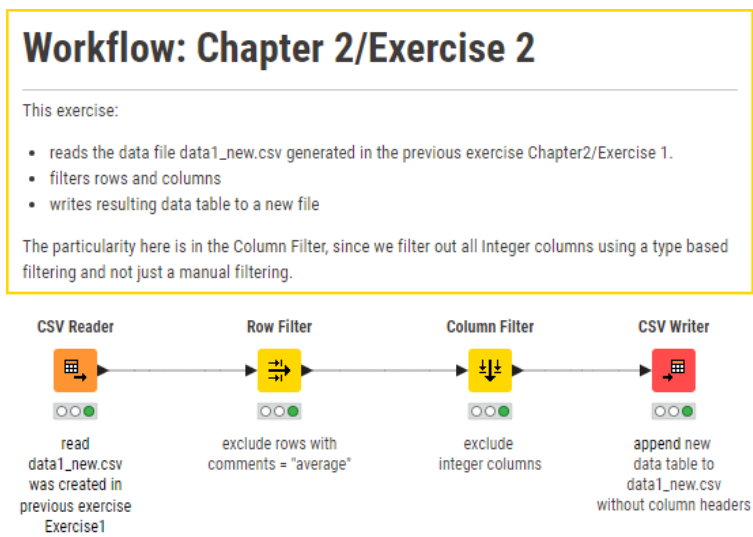



*Figure 2.36. Exercise 2: The workflow.*

***Note.*** After copying the “CSV Reader” node from Exercise 1, you need to disable the option “Limit Rows” in the “Advanced Settings” window, because this file has no initial comments.

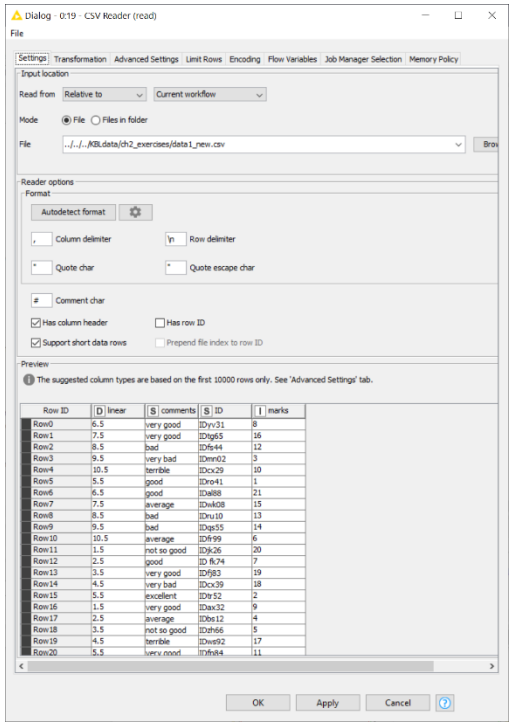



*Figure 2.37. Exercise 2: CSV Reader "Settings" configuration.*

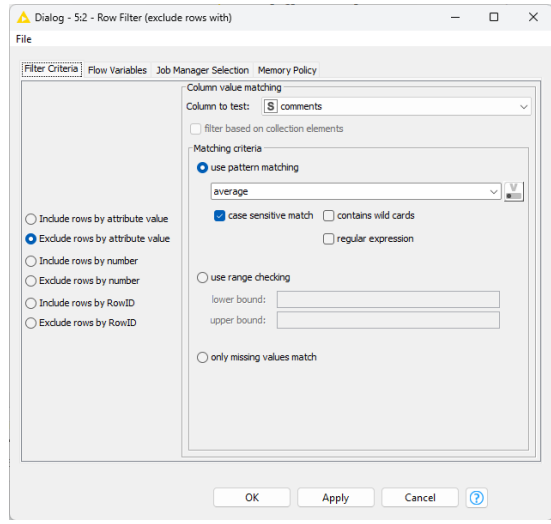



*Figure 2.38. Exercise 2: Row Filter configuration.*

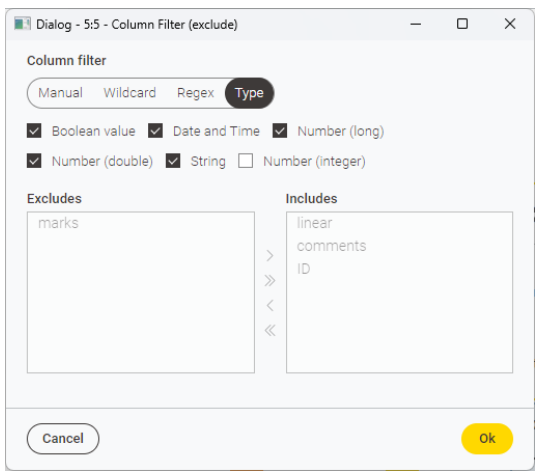



*Figure 2.39. Exercise 2: Column Filter configuration.*

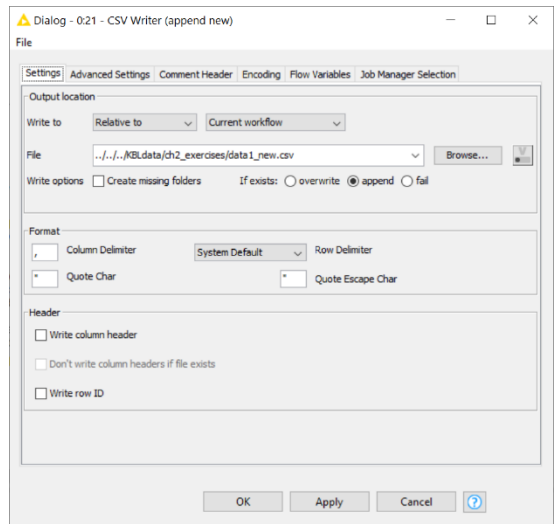



*Figure 2.40. Exercise 2: CSV Writer configuration.*

***Note.*** We saved the data in “Append” mode into the CSV file. The data from Exercise 2 has only 3 columns, while the existing data in the file has 4 columns. The “CSV Writer” node does not check the consistency of the number and the position of the columns to be written with the number and the positions of the existing columns. It is then possible to write inconsistent data to a file. You need to be careful when working in “Append” mode with a “CSV Writer” node.

# Chapter 3: My First Data Exploration

## 3.1. Introduction

This chapter describes a few data wrangling nodes useful to bring the data into the desired shape, involving data transformation, string manipulation, rule application, and other similar tasks. For that, we will use three workflows: “Column Rename Example”, “Write To DB” and “My First Data Exploration”. “Column Rename Example” is a very simple workflow showing the usage of the Column renamer node, “Write To DB” writes data into a database, and “My First Data Exploration” reads the same data from the database and graphically explores the data.

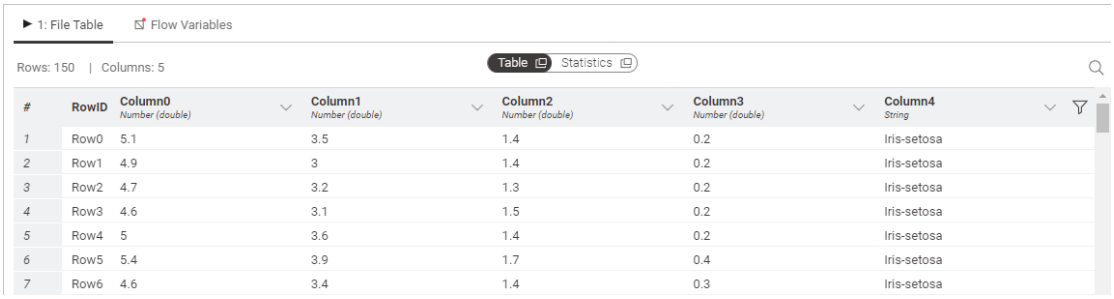
The goal of this chapter is to become familiar with:

- nodes and options for database handling
- the “Views” category containing nodes for graphical data exploration
- a few more column operation nodes, like nodes for string manipulation and missing value handling

We start from the very well-known Iris Dataset, downloaded from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets/Iris) and available under the KBLdata folder, to prepare the data for the next graphical exploration. The Iris dataset describes a number of iris plants by means of 4 attributes:

- the sepal length
- the sepal width
- the petal length
- the petal width

The plants described in the data set belong to three different iris classes: Iris setosa, Iris versicolor, and Iris virginica.



1: File Table | Flow Variables

Rows: 150 | Columns: 5 — Table | Statistics

| # | RowID | Column0 Number (double) | Column1 Number (double) | Column2 Number (double) | Column3 Number (double) | Column4 String |
|---|---|---|---|---|---|---|
| 1 | Row0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 2 | Row1 | 4.9 | 3 | 1.4 | 0.2 | Iris-setosa |
| 3 | Row2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 4 | Row3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 5 | Row4 | 5 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| 6 | Row5 | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa |
| 7 | Row6 | 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa |

*Figure 3.1. The iris dataset.*

This dataset has been used for many years as a standard for classification. The three classes are not all linearly separable. Only two of the three iris classes can be separated by using a linear function on two of the four numeric attributes. For the third class we need to use something more sophisticated than a linear separation. The first two classes and their possible linear separation can be clearly identified by using graphic plots. This is the reason why we use this dataset to illustrate the KNIME nodes for visual data exploration.

We will use this chapter also to explore string manipulation and how to create new rule-based values from existing columns' values.

In the "KNIME Explorer" panel, we create now a new workflow group named "Chapter3", to contain all workflows created in this chapter of the book. Under workflow group "Chapter3" we create two empty workflows: "Write To DB" and "My First Data Exploration". As we said at the beginning of this section, "Write To DB" will show how to build a new dataset and how to write it into a database, while "My First Data Exploration" will describe how to perform a visual exploration of the data.

Let's start with reading the data in the "Column Rename Example" workflow. Since the Iris dataset file (iris.data) is not in a standard data format, we cannot directly drag and drop it to the workflow editor. Instead, we read it with a "CSV Reader" node. If we do not change the name of the data columns in the Transformation tab, then the node will read a data table like the one in figure 3.1. We write the comment "read the iris.data file from KBLdata folder" under the "CSV Reader" node, for a quick overview of the node task.

***Note.*** The iris dataset file does not contain column names. The "CSV Reader" node then assigns to each column a default name like "Column0", "Column1", "Column2", "Column3", and "Column4". Besides "Column4", where we can see that this is the iris class, we need to read the file specifications in file "iris.names" to understand which column represents which numerical attribute.

## 3.2. Replace Values in Columns

After reading the description of the iris data set in the iris.name file, we discover that the five columns are organized as follows:

1. Sepal length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal width in cm
5. class

And that there are no missing values in the data set. Thus, the first step is to rename the data set's columns, in order to be able to talk clearly about what we are doing on the data. KNIME has a node "Column Rename" to be used exactly for this purpose.

### Column Renamer

The node "Column Renamer" can be found in the "Node Repository" panel under "Manipulation" → "Column" → "Convert & Replace".

The "Column Renamer" node allows for the renaming of columns in the input data table.

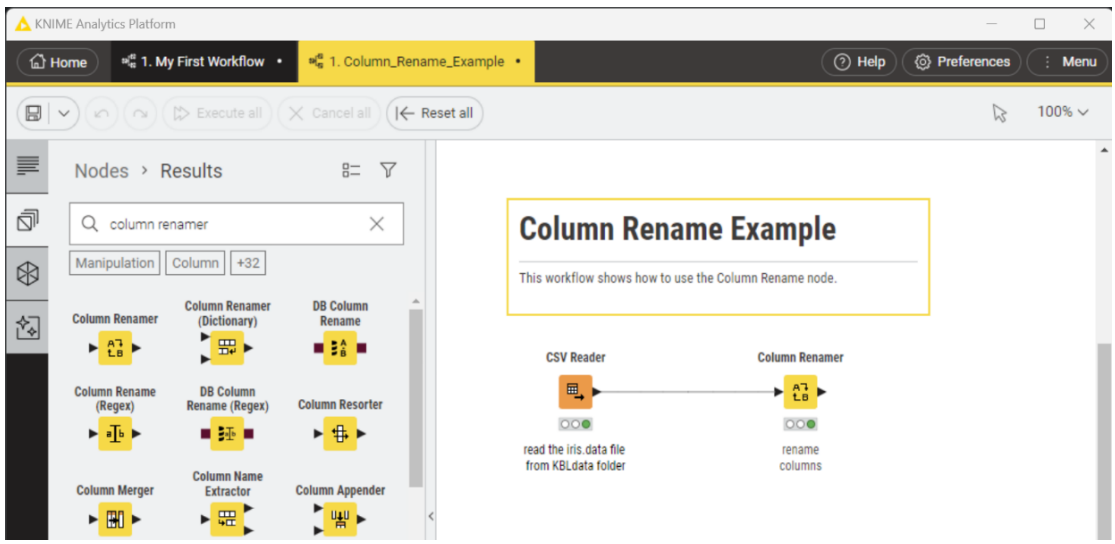



*Figure 3.2. Create a Column Renamer node.*

In the configuration window we have on the left the list of candidate columns for renaming; on the right the list of the columns actually selected for renaming.

The configuration dialog requires:

- to select the columns on which to operate by double-click in the left panel
- to flag the columns whose name or type needs changing (checkbox)
- to provide the new column names
- and optionally the new column types

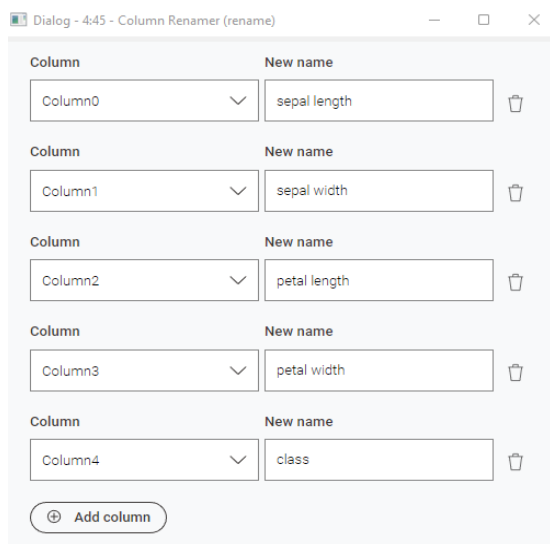



*Figure 3.3. Configuration dialog of the Column Renamer node. The button with "Trash" symbol is there to remove an already selected column from the renaming panel.*

We created a *Column Renamer* node, and we connected it to the *CSV Reader* node. We then assigned new names to the data table columns according to the "Iris.name" specification file and we ran the "Execute" command. The same operation could have been executed in the **Transformation tab** of the *CSV Reader* node. This is what we did in the second workflow of this chapter "Write To DB".

Let's suppose now that the iris names in the "class" column are too long or too complex for our task and that we would like to have just class numbers: "class 1", "class 2", and "class 3". That is, we would like to add a column where "Iris-setosa" from column "class" is translated into "class 1", "Iris-versicolor" into "class 2", and finally all remaining instances belong to a "class 3".

KNIME has a very practical node: the "Rule Engine" node. This node defines a set of rules on the values of the input data columns and generates new values according to the defined rule set. The new values can form a new column to append to the existing ones in the input data table or replace an existing data column.

The rule set that we would like to implement in this case is the following:

```
IF class = "Iris-setosa"                THEN          class 1
IF class = "Iris-versicolor"            THEN          class 2
ELSE                                                  class 3
```

The *Rule Engine* node uses the following syntax to express this same rule set:

```
$class$ = "Iris-setosa" => "class 1"
$class$ = "Iris-versicolor" => "class 2"
TRUE => "class 3"
```

Where $class$ indicates values in input column “class”, “Iris-setosa” is the match String for the “=” operator, “=>” introduces the rule consequent, and “class 1” is the consequent value.

***Note.*** Fixed string values need to be encapsulated in between quotation marks to be correctly interpreted as strings by the “Rule Engine” node.

The final keyword “TRUE” represents the ELSE value in our list of rules, i.e., the value that is always true if no other rule is applied first.

***Note.*** To insert a constant value in a data column, you can just use

`TRUE => <new constant value>`

with no other rule in a “Rule Engine” node. Alternatively, you can use the “Constant Value Column” node.

## Rule Engine

The node “Rule Engine” is located in the “Node Repository” panel in the category “Manipulation” → “Row” → “Other”.

This node defines a set of rules and creates new values based on the set of rules and the input column values.

The configuration dialog includes:

- The list of available input data columns
- The list of available functions and operators

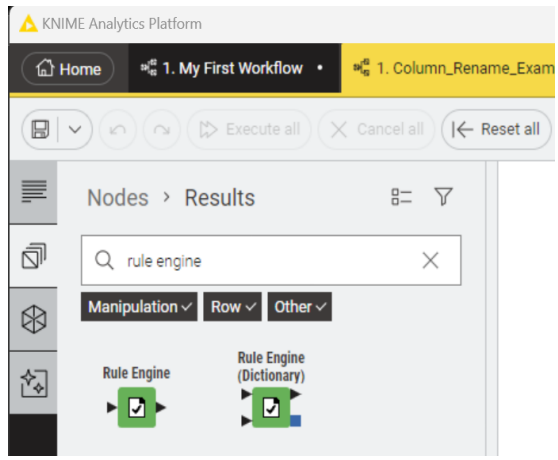



*Figure 3.4. Location of the "Rule Engine" node in the "Node Repository" panel.*

- A description panel to describe the usage and task of the selected function/operator
- A rule editor where to edit the set of rules
- The option to create a new column in the output data table or to replace an existing one

## Column List

The first panel on the left upper corner of the “Rule Engine” configuration window shows all available columns from the input data table. Those are the columns our set of rules is going to work on.

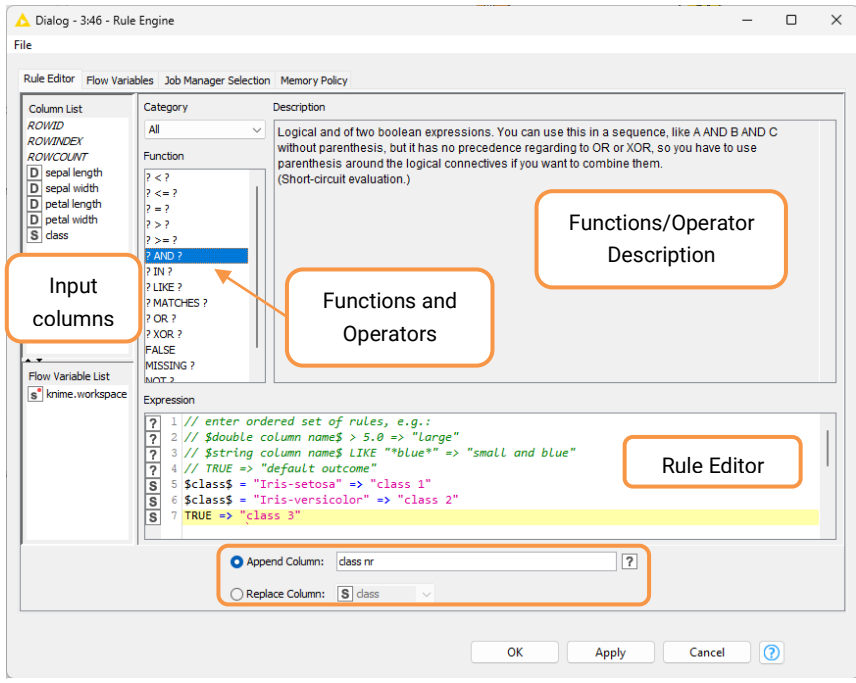



*Figure 3.5. The configuration window of the "Rule Engine" node.*

## Flow Variable List

The panel right below the “Column List” panel contains all flow variables available to the node. However, flow variables are not treated in this book and we will ignore them when setting up our set of rules.

## Function

The “Function” panel contains a list of functions and logical operators available to create the rule set. The “Category” menu on top of the “Function” list, allows to reduce the function list to a smaller subset.

### Description

If a function or an operator is selected in the “Function” list, this panel provides a description of its task and usage.

### Expression

The “Expression” panel is the rule editor. Here you can type your set of rules. If you need to involve a data column or a function, just double-click the desired item in the respective panel and it will appear in the rule editor with the right syntax.

Every rule consists of a condition (antecedent), including a function or an operator, and of a consequence value. The symbol “=>” leads the condition to the consequence value, like: `<antecedent> => <consequence value>`. “TRUE” in the last rule leads to the default value, when none of the previous conditions apply. The rule can be edited and changed at any time.

To build our rule set, we typed in the set of rules described above.

### Append Column / Replace Column

At the bottom of the configuration window, there are the options to choose whether to create a new data column or replacing an existing one. The default option is “Append Column” and the default name for the new column is “prediction”. We selected the default option, and we named the new column “class nr”.

After configuration, we commented the Rule Engine node as “from iris names to class no” and we ran the “Execute” command.

## 3.3. String Splitting

In this section we explore how to perform string manipulation with KNIME. For example, how can we split the column “class” in a way as to have “Iris” in one column and “setosa”, “versicolor”, or “virginica” in another column? Vice versa, how can I build a key to uniquely identify each row of the data table?

In KNIME there are 3 nodes to split string cells.

- ***"Cell Splitter by Position"*** splits each string based on character position. The column to split contains string values. The node splits all strings in the column in k substrings, each of length n1, n2, n3,… nk, where n1+n2+n3 +… nk = L is the length of the original strings. Each substring is then placed in an additional column. Notice that for this node all input strings need to be at least L-character long.
- ***"Cell Splitter [by Delimiter]"*** uses a delimiter character to extract substrings from the original strings. Strings can be of variable length. If the delimiter character is found, the substring before and after will end up in two different additional columns. The name of the node is actually just "Cell Splitter". However, since it uses a delimiter character I will call it "Cell Splitter [by Delimiter]".
- ***"Regex Split"*** is a Cell Splitter by Regex. It uses a Regular Expression rule to recognize substrings. After the substrings have been recognized the node splits the original string into the recognized substrings and places them into different additional columns.

Unlike the split column operation, there is only one node to combine string columns: the "Column Combiner" node.

- The ***"Column Combiner"*** node concatenates the strings from two or more columns and puts the result into a new appended column.

> ***Note.*** All string manipulation nodes, like the "Cell Splitter" nodes and the "Column Combiner" node, are located in the "Node Repository" panel in: "Manipulation" → "Column" -> "Split & Combine".

In the column "class" we want to separate substring "Iris" from the remaining substrings "setosa", "versicolor", or "viriginica".

- If we split by position, we need to split at the 4th character (at the end of "Iris" and before the rest of the string) and at the 5th character (before "setosa", "versicolor", or "virginica").
- If we split by delimiter, we need to split around character "-".
- Finally, if we split by RegEx, we need to find a Regular Expression rules to express "Iris", "-", and the remaining letters. A possible regular expression could be: `((Iris)[\-]*([A-Za-z]*)`.

Let's see now in detail how to use the three "Cell Splitter" nodes to do that.

## Cell Splitter by Position

This node splits the column string values based on character position. The result consists of as many new columns as many position indices plus 1. The configuration window asks for:

- The split indices (the character positions inside the string on which to split) comma separated.
- The new column names (the new column names are always one more than the number of split indices). The new column names have to be comma separated.
- The name of the string column on which to perform the splits.

We selected:

- Split position indices 4 (at the end of word "Iris") and 5 (after "-")
- We will obtain 3 substrings: "Iris" in column "split_0", "-"in column "split_1", and "setosa"/"virginica"/"versicolor" in column "split_2"
- The column to perform the split on is "class"

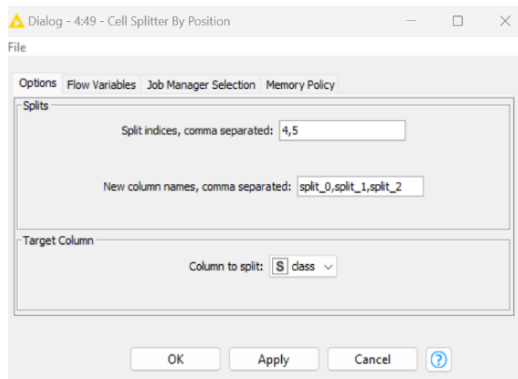



*Figure 3.6. Configuration dialog for the Cell Splitter by Position node.*

Column "split_1" will contain only strings "-". We can always remove it later on by means of a "Column Filter" node.

## Cell Splitter [by Delimiter]

This node splits the column string values at a delimiter character. The result will be as many new columns as many delimiter characters have been found plus one. The configuration window requires the following settings:

- The name of the column on which to perform the splits
- The delimiter character
- The output type:
    - As new columns to append to the data table (here you need to set the array size)

- As one column only containing the list/set of sub-strings (a set of strings is like a list but without duplicate values)

If many splits are forecasted, the first option can quickly add too many new columns to the output data set and become unmanageable. On the opposite, the second option adds only one additional column to the output data set, compacting all substrings into a collection type column.

The size of the resulting array of substrings can be set a priori or, if we do not know it, we can let the node guess the best size. This last option works for most of the string splitting tasks. For more complex tasks, we might need to set the array size manually ourselves.

In case we set a fixed array size, if the array size is smaller than the number of detected substrings, the last splits will be ignored. On the other side if the array size is bigger than the number of detected substrings, the last new col umns will be empty. We selected:

- Column to split = “class”
- Delimiter character = “-“
- Array size = 2 and substrings to be output as new columns

The substrings will be stored in new columns named “<original_column_name>_Arr[0]” and “<original_column_name>_Arr[1]”, that is based on our configuration settings “class_Arr[0]” and “class_Arr[1]”.

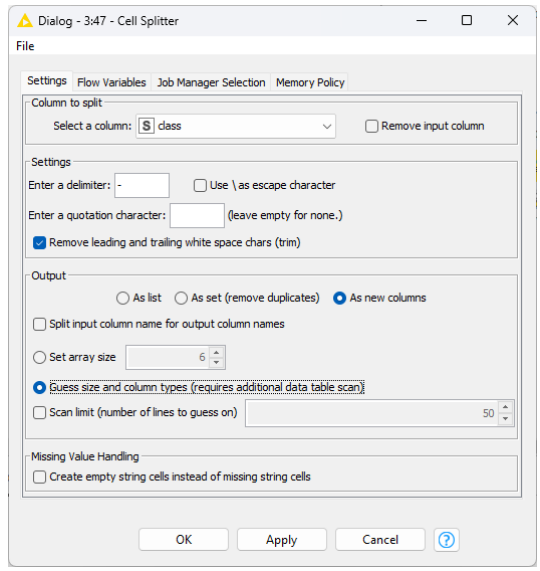



*Figure 3.7. Configuration dialog for the Cell Splitter node.*

***Note.*** Here there is no column with only strings “-“. All characters “-“ are lost.

## RegEx Split (= Cell Splitter by RegEx)

This node identifies substrings in a selected string column on the basis of a Regular Expression. Substrings are represented as Regular Expressions inside parenthesis. The original strings are then split in substrings identified by such regular expressions. Each substring will create a new column. The configuration window requires:

- The name of the column to split
- The Regular Expression patterns to identify the substrings, with the substrings included in parenthesis
- A few additional options to consider multi-line strings and use a case sensitive/insensitive match

To separate the word “Iris” from the rest of the string in column “class” by using a “RegEx Split” node, we selected:

- Column to split (Target Column) = class
- Regular Expression: `((Iris)[\-]*([A-Za-z]*)`, which means:
  - First substring in parenthesis contains the word “Iris”
  - Then comes a “-“ not to be used as a substring, since it is not in parenthesis
  - The second substring can contain any alphabetical character
  - The result are two substrings named “split_0” and “split_1”, one containing the word “Iris” and the other containing the remaining word “setosa”, “versicolor”, or “virginica”.

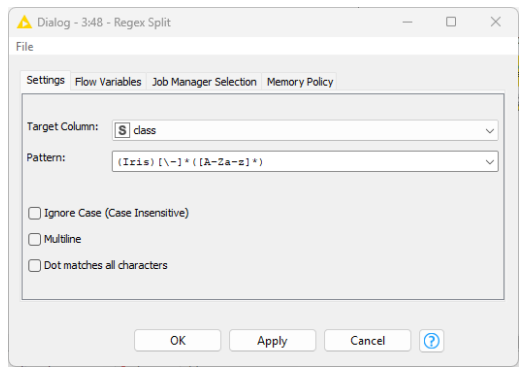



*Figure 3.8. Configuration dialog for the RegEx Split node.*

The same result could have been obtained with a more general Regular Expression, like for example `([A-Za-z]*)[\-]*(.*$)`, which means:

- First substring in parenthesis contains any alphabetical character
- Then comes a “-“ not to be used as a substring, since it is not in parenthesis
- The second substring can contain any alphanumerical character

These “Cell Splitter” nodes have all been named “iris + attr”, which describes the split between word “iris” and the following attribute “versicolor”, “virginica”, or “setosa”.

# 3.4. String Manipulation

Let's suppose now that we want to rebuild the iris class name but with a different string structure, for example "<attribute>:IRIS", with the word IRIS all in capital letters and <attribute> being "virginica", "setosa", or "versicolor". We need then to replace the string "Iris" with "IRIS" and to recombine it with the <attribute> string. In KNIME there are many nodes to perform all kinds of string manipulation. One node in particular, though, can perform most of the needed string manipulation tasks: the "String Manipulation" node.

## String Manipulation

The "String Manipulation" node can perform a number of string manipulation tasks, like to calculate a string length, to compare two strings, to change a string into only uppercase or lowercase characters, to replace a substring or all occurrences of a character inside a string, to capitalize the string words, to find the positions of a character or substring occurrence, to extract a substring from a string, and so on.

The configuration window of the "String Manipulation" node is similar to the one of the "Rule Engine" node.

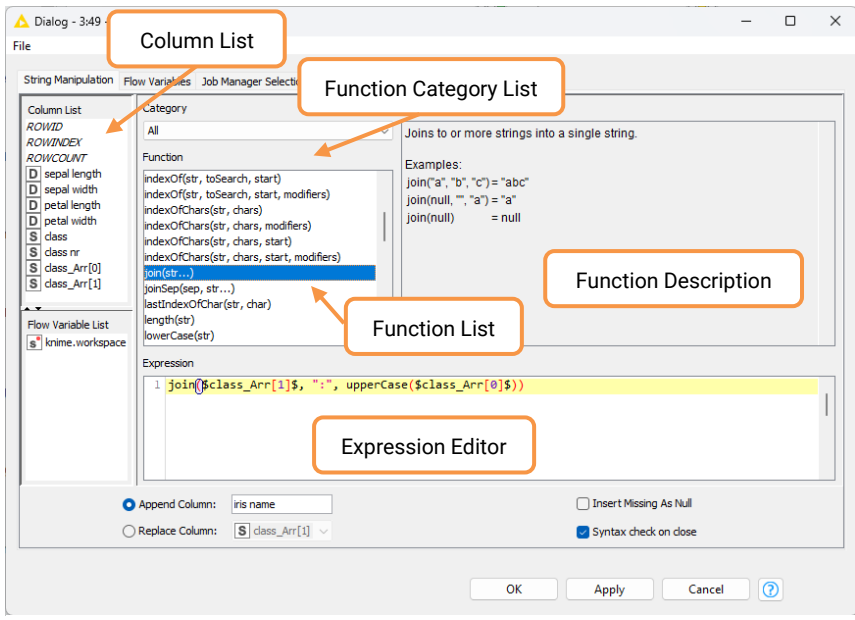



*Figure 3.9. Configuration dialog for the String Manipulation node.*

The "**Expression Editor**" is located again in the central lower part of the configuration window. Here a number of string functions can be nested and combined together to obtain the desired string transformation.

The available string functions are listed above in the "**Function List**" panel. Functions can also be visualized in smaller groups, by selecting a category in the "**Category List**" menu over the "Function List" panel.

The "**Description**" panel on the right explains the task of the selected function.

On the left, in the "**Column List**" panel, all available data columns are displayed. Double-clicking a column or a function automatically inserts it in the "Expression Editor" with the correct syntax. Fixed string values have to be reported in quotation marks, for example "abc", when introduced in the "Expression Editor".

The "**Insert Missing As Null**" flag enables the production of a null string, instead of an empty data cell, when the string manipulation function is somehow unsuccessful.

The configuration window finally requires the **name of the new or of the existing column**, depending on whether the resulting string has to overwrite existing data.

The "String Manipulation" node that we introduced in the "Write To DB" workflow follows the "Cell Splitter" node and combines (function "join()") the <attribute> part of the class name in column class_Arr[1] with fixed string ":" and with the uppercase version (function "uppercase()") of the word "iris". The result is for example "setosa:IRIS" for the original string "Iris-setosa". Notice that the splitting of the original string into the substrings contained in class_Arr[] could have also been obtained inside the String Manipulation node using a substr() function.

> ***Note.*** Functions "toInt()", "toDouble()", "toBoolean()", "to Long()", "toNull()"convert a string respectively into an integer, a double, and so on. They can be used to produce a non-string output column at the output port of the String Manipulation node.

The String Manipulation node is particularly useful when we want to combine a number of different string functions into a single more complex one. However, an alternative processing uses a sequence of single dedicated nodes. This approach leads to a more crowded workflow, but it provides an easier interpretation of all used string manipulation functions.

To switch from lower to upper case or vice versa, the "Case Converter" node in the category "Manipulation" → "Column" → "Transform" can be used.

## Case Converter

This node transforms the string characters into lowercase or uppercase depending on the "Select mode" flag. The configuration window requires:

- "Select mode": "UPPERCASE" or "lowercase"
- The names of the columns to transform. These columns are listed in the frame "Include". All other columns that will not be affected by the transformation are listed in the 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"

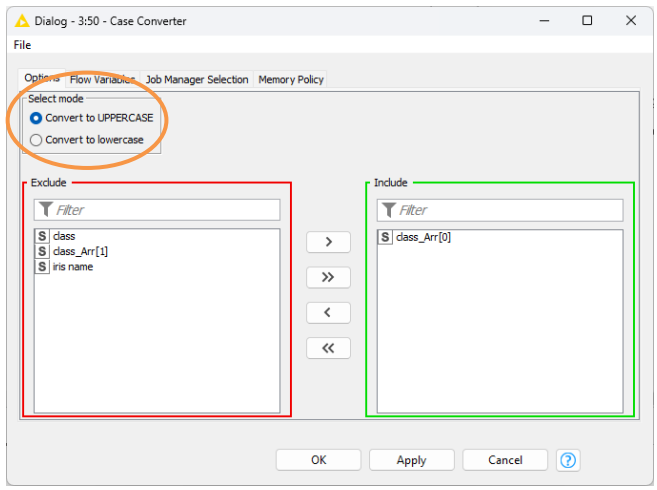



*Figure 3.10. Configuration window for the Case Converter node.*

We connected a "Case Converter" node to the output port of the "Cell Splitter" node. Of course we could have connected the "Case Converter" node to the output port of any of the "Cell Splitter" nodes. We chose the "Cell Splitter" node just as an example. Then we configured the "Case Converter" node like that:

- "Select mode" is set to "Convert to UPPERCASE"
- Columns to change is only "class_Arr[0]", which is the column containing the word "Iris", in the "Include" set

To replace a string in general, there is a "String Replacer" node in "Manipulation" → "Column" → "Convert & Replace".

This node has a variant “String Replace (Dictionary)” that performs the string replacements based on a previously formatted dictionary text file. This node can be useful to replace multiple strings and substrings with the same string value.

The corresponding function in the String Manipulation node would be *upperCase()*.

## String Replacer

The “String Replacer” node replaces a pattern in the values of a string type column. The configuration window requires:

- The name of the column where the pattern has to be replaced
- The pattern to match and replace (wildcards in the pattern are allowed)
- The new pattern to overwrite the old one

And a few more options:

- Whether the pattern to be matched and replaced contains wildcards or is a regular expression
- Whether the replacement text must replace all occurrences of the pattern as isolated strings only or as substrings as well
- Whether the pattern match has to be case sensitive
- Whether escape characters are indicated through a backslash
- Whether the result replaces the original column (default) or creates a new column

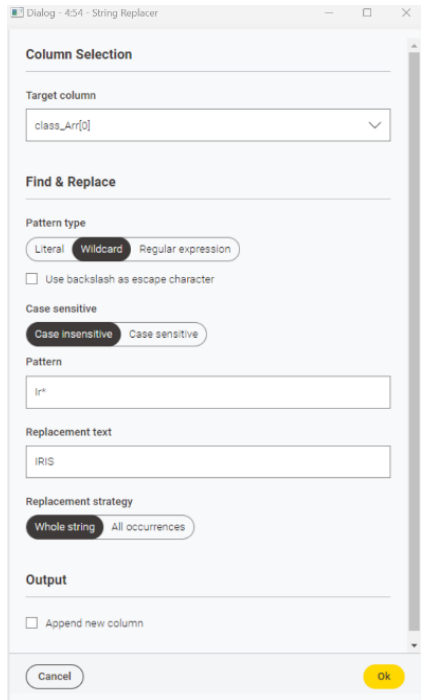



*Figure 3.11. Configuration window for the String Replacer node.*

To change string “Iris” into string “IRIS”, we connect a “String Replacer” node to the output port of the “Cell Splitter” node and use the following configuration:

- Target column is “class_Arr[0]”, which contains string “Iris”
- Pattern to be replaced can be “Iris” or more generally “Ir*” with a wildcard “*”
- The new pattern to overwrite the old one is “IRIS”

Result column “class_Arr[0]” contains all “IRIS” strings, exactly like the column generated with the “Case Converter” node. Finally, we want to combine all those substrings in a new string column named “iris name” and containing strings structured as: “<attribute>:IRIS”. To combine two or more string columns, there is the “Column Combiner” node under “Manipulation” → “Column” → “Split & Combine”.

The corresponding function in the String Manipulation node would be *replace()*.

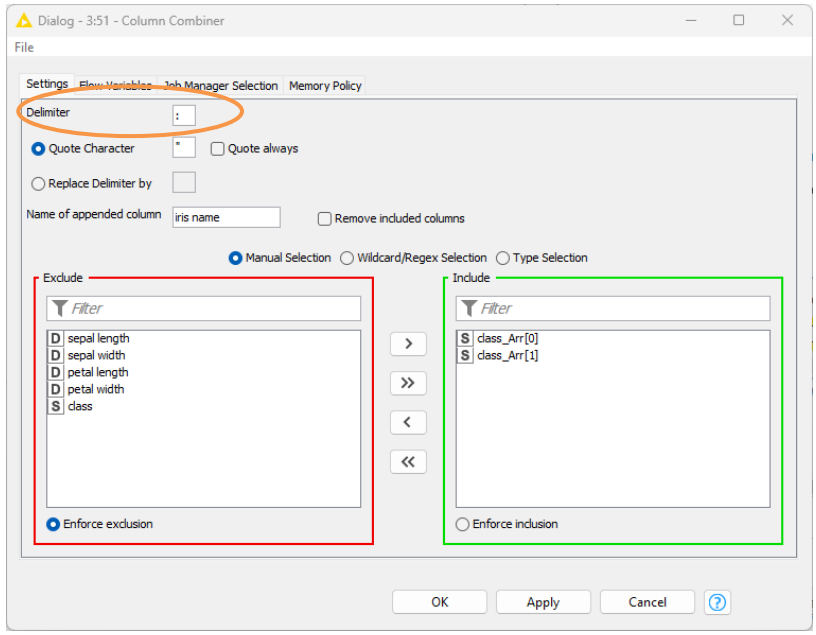



*Figure 3.12. Configuration window for the Column Combiner node.*

## Column Combiner

The “Column Combiner” node combines two or more string columns into a single string column, optionally joining them through a delimiter character. The configuration window requires:

- The delimiter character (if any, this field can also be empty)
- If we want to include the original substrings in quotes, then flag “Quote always” must be enabled and the “Quote character” must be supplied
- The name of the new column

- The names of the columns to combine. These columns are listed in the frame "Include". All other columns that will not be used for the combination are listed in the 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".

To obtain the final string values "<attribute:IRIS>", we need a "Column Combiner" node with the following settings:

- Delimiter is ":"
- Columns to combine in the "Include" frame are "class_Arr[1]" and "class_Arr[0]"
- No quotes around the original strings, that is flag "Quote always" is disabled
- Name of the new column is "iris name". Notice that this node has no option to replace an input column with the new values

The corresponding function in the String Manipulation node would be *join()*.

***Note.*** In the "Column Combiner" node it is not possible to arrange the columns' concatenation order. Columns are combined following their order in the input data table.

For example, column "class_Arr[0]" comes before column "class_Arr[1]" in the input data table and therefore the resulting combined strings will be "class_Arr[0]:class_Arr[1]", that is: "IRIS:<attribute>", which is not exactly what we wanted. To change the substring order, we need to change the column order in the input data table.

To change the columns' order in the input data table, we use a "Column Resorter" node located in "Manipulation" → "Column" → "Transform".

## Column Resorter

The "Column Resorter" node changes the order of the columns in the input data table.

The list of input columns with their order (left-to-right becomes top-to-bottom) is presented in the configuration window.

- To move one column up or down, select the column in the list and click button "Up" or "Down".

- To make one column the first of the list, select the column and click “Move First”. Same procedure to make one column the last of the list with button “Move Last”.
- To use an alphabetical order on the column names, click button “A-Z” for descending order and “Z-A” for ascending order.

We connected a “Column Resorter” node to the output port of the “Case Converter” node. We moved column “class_Arr[0]” one position down in the configuration window, that is after column “class_Arr[1]”. After commenting the “Column Resorter” node with “rearrange column order for next node column combiner”, we connected its output port to the “Column Combiner” node. Now the “Column Combiner” has the input columns in the right order to get the final strings structured as “<attribute>:IRIS”.

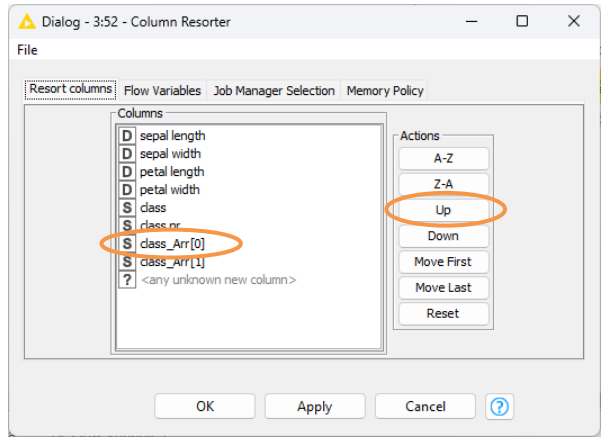



*Figure 3.13. The configuration window of the Column Resorter node.*

***Note.*** The “Column Combiner” node is useful to build unique keys to identify data rows.

## 3.5. Type Conversion

In the previous section we went through the string manipulation functionalities available in KNIME Analytics Platform. Before moving to the database section, I would like to spend a little time showing the “Type Conversion” nodes.

In this book we will not work with data type Date&Time. Excluding this data type, there are three basic type conversion nodes: *Number To String*, *String To Number*, and *Double To Int*. All these nodes are located in the “Node Repository” panel in: “Manipulation” → “Column” → “Convert & Replace”.

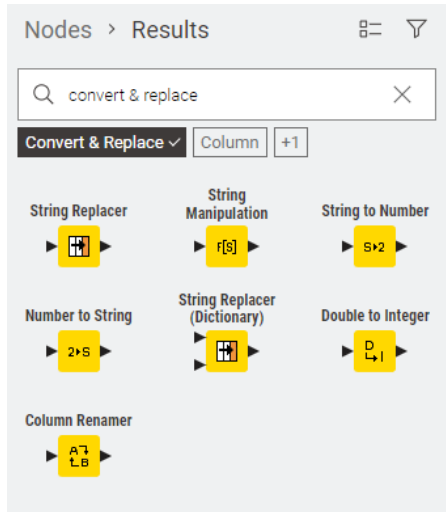



*Figure 3.14. Location of the Type Conversion node in the Node Repository panel.*

In order to show how these type conversion nodes work, we will pretend that we want to change one of the data columns, for example “petal width”, from type Double to type String. We will use a *Number To String* node for that.

## Number to String

The *Number To String* node converts all cells of a column from type *Double* or *Int* to type *String*. The configuration window requires:

- The names of the columns to be converted to type String. These columns are listed in the frame “Includes”. All other columns are listed in the frame “Excludes”.
- To move from frame “Includes” to frame “Excludes” and vice versa, double click the column name or select the column and then click on arrows in between the two windows.
- The “Number to String” node is equivalent to the function *string()* in the String Manipulation node.

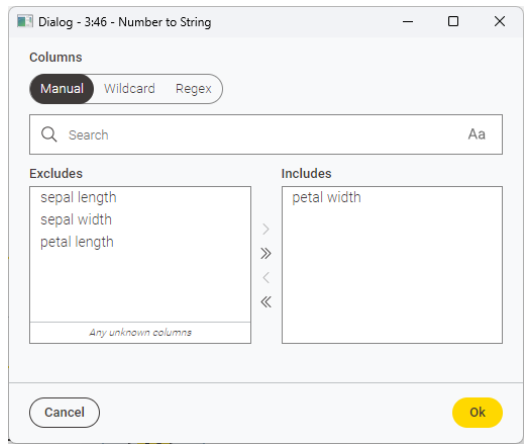



*Figure 3.15. Configuration window of the Number to String node.*

We inserted only the column “petal width” in the frame “Include” to be converted from type Double to type String.

Now for demonstration’s sake, let’s suppose that we want to isolate the floating and the integer part of column “petal width”. Since now column “petal width" is of type String, we will use a *Cell Splitter* node with delimiter character “.”. We named this node “int(petal width)”. At this point we have:

- The original String column “petal width”
- The first substring “petal width_Arr[0]” containing the integer part of “petal width” value
- The second substring “petal width_Arr[1]” containing the floating part of “petal width” value

To convert values in the opposite direction of the *Number To String* node, we find the *String To Number* node. For demonstration's sake, let's reconvert "petal width" from a String type to a Number type (Double, Long, or Int). In order to do that, we can use the *String To Number* node.

## String to Number

The *String To Number* node converts all cells of a column from type "String" to type "Double", "Long" or "Int". The configuration window requires:

- The final column type: Double, Long, or Int
- The decimal separator and the thousands separator (if any)
- The names of the columns to be converted to the selected type. These columns are listed in the frame "Includes". All other columns are listed in the frame "Excludes".
- To move from frame "Includes" to frame "Excludes" and vice versa, double click the column name or select the column and then click on arrows in between the two windows.
- The "String to Number" node is equivalent to *toInt()*, *toDouble()*, *toLong()*, and similar functions in the "String Manipulation" node.

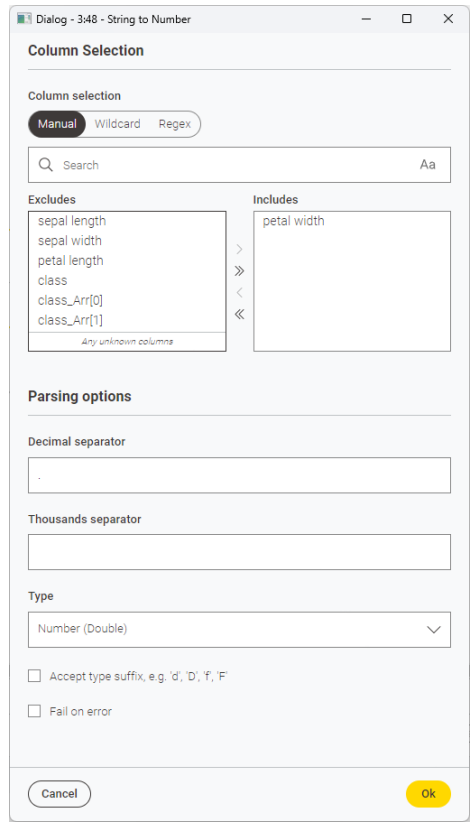



*Figure 3.16. Configuration dialog of the String to Number node.*

Let's still suppose, for the sake of nodes demonstration, that we have converted the "petal width" array columns to type Double, but that actually we wanted to have them of type Int. Let's ignore the fact that it would be enough to change option "Type" in the configuration window of the "String To Number" node and let's experiment with a new node: the "Double To Int" node

## Double to Integer

The “Double To Int” node converts all cells of a column from type “Double” to type “Int”. The configuration window requires:

- The rounding type: round, floor, or ceil. “round” is the standard rounding, “floor” rounds up to the next smaller integer, “ceil” rounds up to the next bigger integer.
- The selection of the columns to be converted to type Integer. Selection can be set manually or using wildcard or regex.
- For both selections: Columns to be transformed into type Int are listed in frame “Includes”. All other columns are listed in frame “Excludes”.

To move from frame “Includes” to frame “Excludes” and vice versa, double click the column name or select the column and then click on arrows in between the two windows.

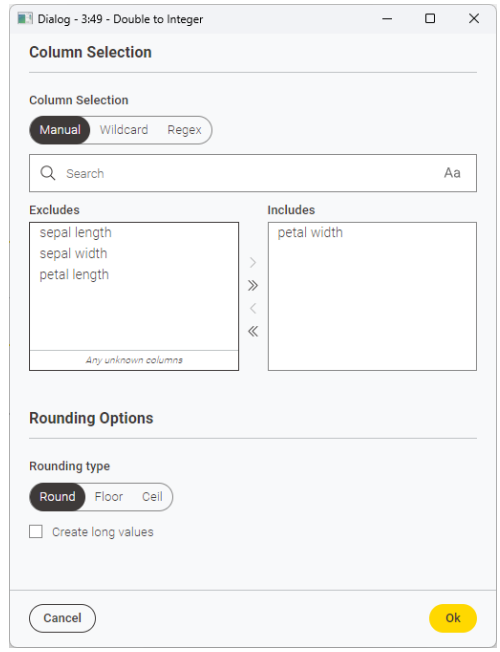



*Figure 3.17. Configuration window of the Double to Integer node.*

***Note.*** A node that covers many of the conversion operations listed in this section is the “Column Auto Type Cast”. This node scans a column to automatically define its data type. A quick scan is also possible, faster but riskier.

# 3.6. Database Operations

We have only shown the type conversion nodes to illustrate KNIME’s potentials. We did not actually need these type conversions to prepare the data for the visualization part. The functions in the *String Manipulation* node would have been sufficient. In the next workflow for visualization, we will use indeed just the data produced by the *String Manipulation* node.

We need now to write the data table generated by the String Manipulation node into a database. In the “Node Repository” panel there is a whole category called “DB” containing all nodes that perform operations on databases.

In order to access a database with KNIME Analytics Platform we first establish a connection to the database with a connector node and along that connection we use a *DB Writer* or a *DB Table Selector* followed by a *DB Reader* node to write or read the data table.

The connector nodes that KNIME provides to access databases come with pre-loaded JDBC drivers. The connector nodes cover the most commonly used and most recent database versions, such as MySQL, SQLite, Vertica, Hadoop Hive, H2, PostgreSQL, and more.

If the connector node for your database is not available, you can always use a generic DB Connector node. Here you will need to provide the driver file for your database. If this is not already in the list of pre-loaded driver files, you can always add it via the "File" → "Preferences" → "Databases" page (see later on in this chapter).

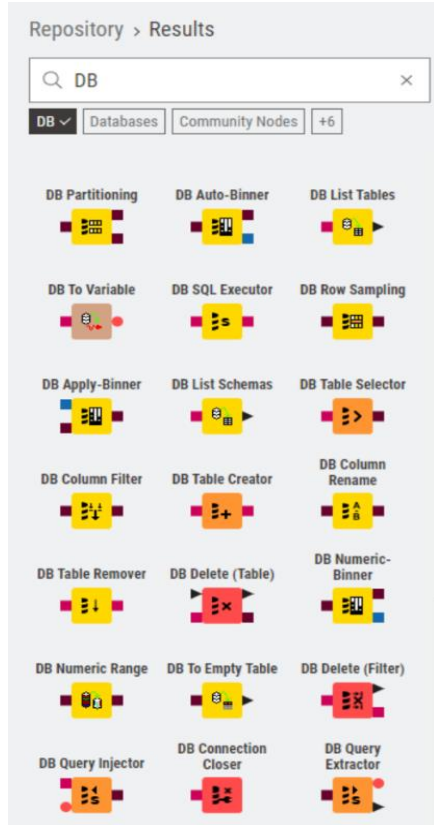



*Figure 3.18. "DB" category in the Node Repository.*

For this example, workflow we use the SQLite database (https://www.sqlite.org/). SQLite is a self-contained, serverless, zero-configuration, transactional file-based database which does not require authentication. This makes it easy for the distribution of the workflows associated with this book, since no installation and no configuration of a separate database are required. The database is contained in the file named "KBLBook.sqlite" in the KBLdata folder.

Just remember that a similar procedure, including authentication, with a similar sequence of nodes should be followed when using other databases.

First, we establish the connection to the database and then along this connection we write the data table to the database. For the first task – establishing the connection to a database – we use a connector node. For the second task – writing the data table into the database – we use the DB Writer node.

Under category "DB"/"Connector" you find a number of connector nodes to establish a connection to a database. Some of those nodes are dedicated connectors, which means they contain a pre-loaded JDBC driver file and present a customized User Interface. Only the "DB

Connector" node is a generic connector, to be used when the dedicated connector for the database of choice is not available.

## SQLite Connector

For the SQLite database, a dedicated connector node is available: the SQLite Connector node. Its configuration window just requires the path to the sqlite file and no password. The JDBC driver for the SQLite database is already pre-loaded.

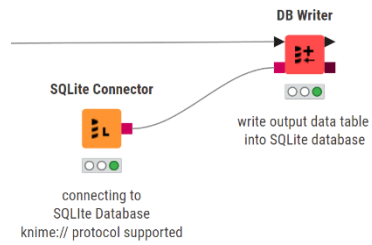



*Figure 3.19. SQLite Connector node + DB Writer node.*

In the configuration window of all connector nodes, there are five tabs: "Connection Settings", "JDBC Parameters", "Advanced", "Input Type Mapping", "Output Type Mapping".

- "Connection Settings" contains all necessary settings to connect to the database: JDBC driver, hostname, port, database name, and full credentials where required.
- Tabs "JDBC Parameters" and "Advanced" allow you to set specific commands to connect to the database; while tabs "Input Type Mapping" and "Output Type Mapping" allow to correctly map all data types from KNIME to the database and viceversa.

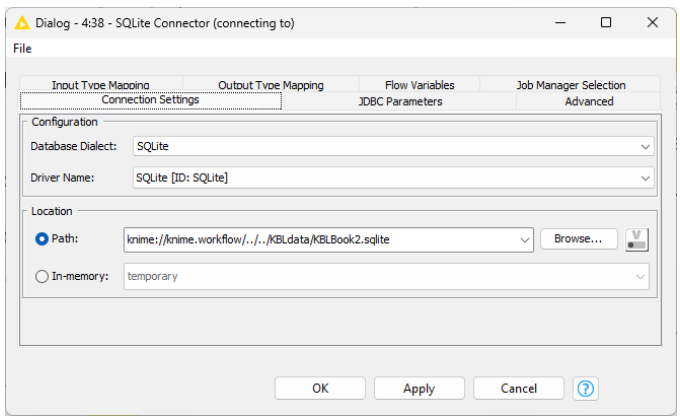



*Figure 3.20. SQLite Connector node: "Connection Settings" tab in the configuration window.*

***Note.*** Did you notice the full red square as input port? So far, we have seen only black triangles as input or output ports. A black triangle means data. A full red square means a database connection. An empty red square means an optional database connection. A full brown square means an SQL statement. There are many different types of ports, each one exporting or importing a different kind of object.

The SQLite database is a simple file-based database, requiring no credentials, which leads to a very simple configuration window. Let's check the configuration window of the connector node to a more complex database: the MySQL Connector node.

## MySQL Connector

The MySQL Connector node connects to a MySQL database and requires:

- The database driver (pre-loaded)
- The hostname and database name
- The username and password for the authentication

Credentials can be supplied as username and password by enabling the option "Username & password". Another option is to define them as credentials at the workflow level.

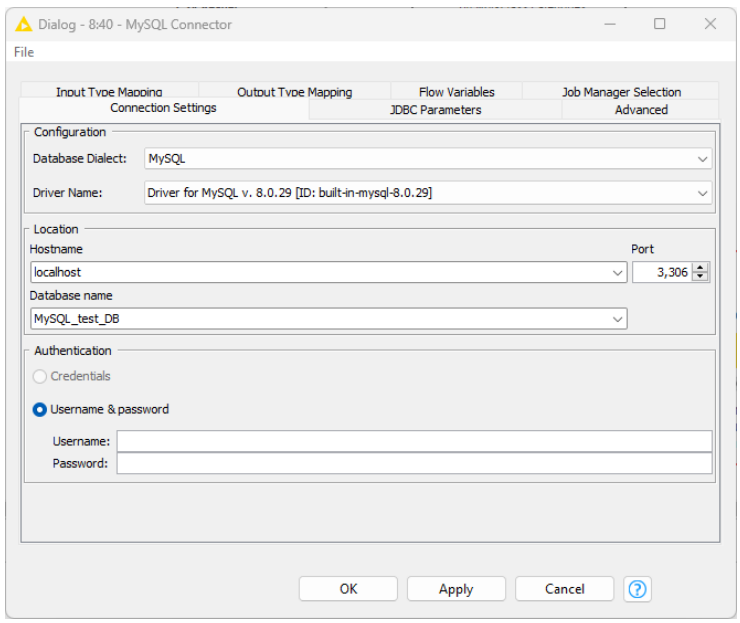



*Figure 3.21. MySQL Connector node: "Connection Setting" tab.*

The other tabs “JDBC Parameters”, “Advanced”, and “Mappings” include the same functionalities as for the SQLite Connector node.

Credentials at the workflow level are automatically encrypted and therefore more secure. Username and password provided directly into the configuration window are not automatically encrypted and require an extra step for security: a master key. This master key will then be used to encrypt usernames and passwords when provided into configuration windows.

## DB Writer

The node “DB Writer”, located in the “DB”/”Read/Write” category, writes the input data table into a database table. If the table does not exist, it is created. If the table already exists, column names in the KNIME data have to match exactly the column names in the table. The only required settings are:

- The name and optionally schema of the table in the database. The button “Select a table” allows you to browse the database content.
- The size of the batch of data to write at each time
- The columns to transfer into the database from the KNIME data through an Include/Exclude frame
- The flag for failure in case of error
- The flag to append the status of the writing operation for each data rows
- The flag “Remove existing table” allows to write in “Append” mode or in “Overwrite” mode
- The “Output Type Mapping” tab contains the specifications on the mapping of the KNIME data types on the database data types.

In order to write the processed iris data to the KBLBook.sqlite database, a “DB Writer” node was connected to the output port of the “String Manipulation” node named “build <attr>:IRIS”. In the configuration window of the “DB Writer” node, we set the data columns that we want to transfer from KNIME into the SQLite database.

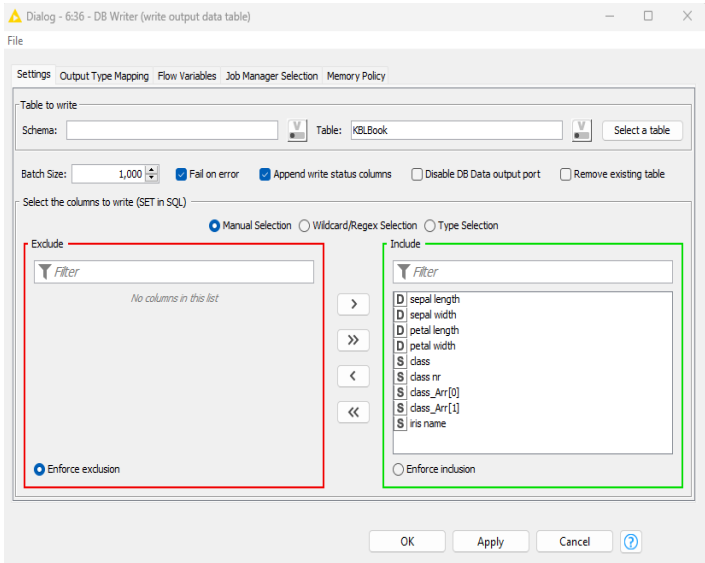



*Figure 3.22. Configuration of the DB Writer node.*

## Import a JDBC Database Driver

The JDBC drivers for the most common and recent databases are already pre-loaded and available in the connector nodes. However, it might happen that the JDBC driver for a specific database is not available. In this case, you need to upload the required database driver onto KNIME Analytics Platform. Usually the JDBC driver file (*.jar) can be found in the database installation or can be requested at the vendor's site as an accessory to the database installation. In order to load a database driver into KNIME, the driver file location must be specified in the KNIME "Preferences" window.

In the Top Menu, select "Settings" (beside the information(i) button). The "Preferences" window opens. The "Preferences" window sets the values for a number of general items, like "Help", "Plug-in", "Java" and so on. All these items are grouped in the list on the left of the "Preferences" window.

- Expand item "KNIME"
- Select sub-item "Databases"

The panel on the right displays the "Databases" settings.

In order to add a new database driver:

- Click the "Add File" or "Add directory" button

- Select the *.jar or *.zip file that contains the JDBC database driver
- The new JDBC driver appears in the "List of loaded database driver files and directories" in the center and becomes available for all database nodes.
- We just completed the workflow "Write To DB", where we read the Iris dataset and we performed a number of string manipulations and some type conversions. The data table emerging from the string manipulation node has been written into a SQLite database.

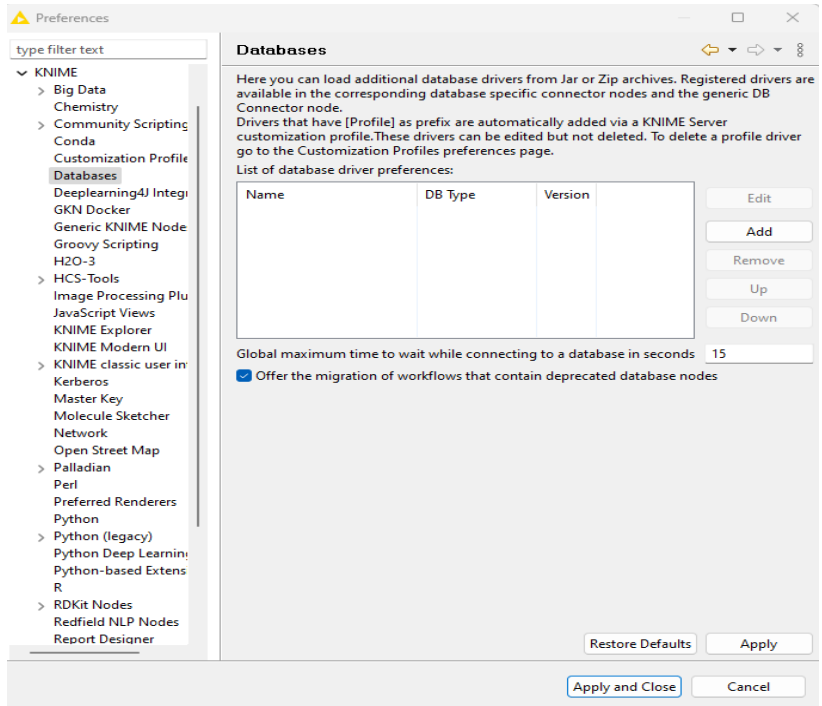



*Figure 3.23. The "Database Driver" page under "Preferences".*

Let's now read from the SQLite database the data we have just written, to perform some visual data exploration. Next to the "DB Writer" node in the "Node Repository" panel, we find the "DB Reader" node, which we will use to read the data from the database table created in the previous workflow.

We create a new workflow "My First Data Exploration" and we establish the connection with the database with a connector node (in this case a "SQLIte Connector" node), the select the database table to read from with a DB Table Selector node, and then read the data with a "DB Reader" node.

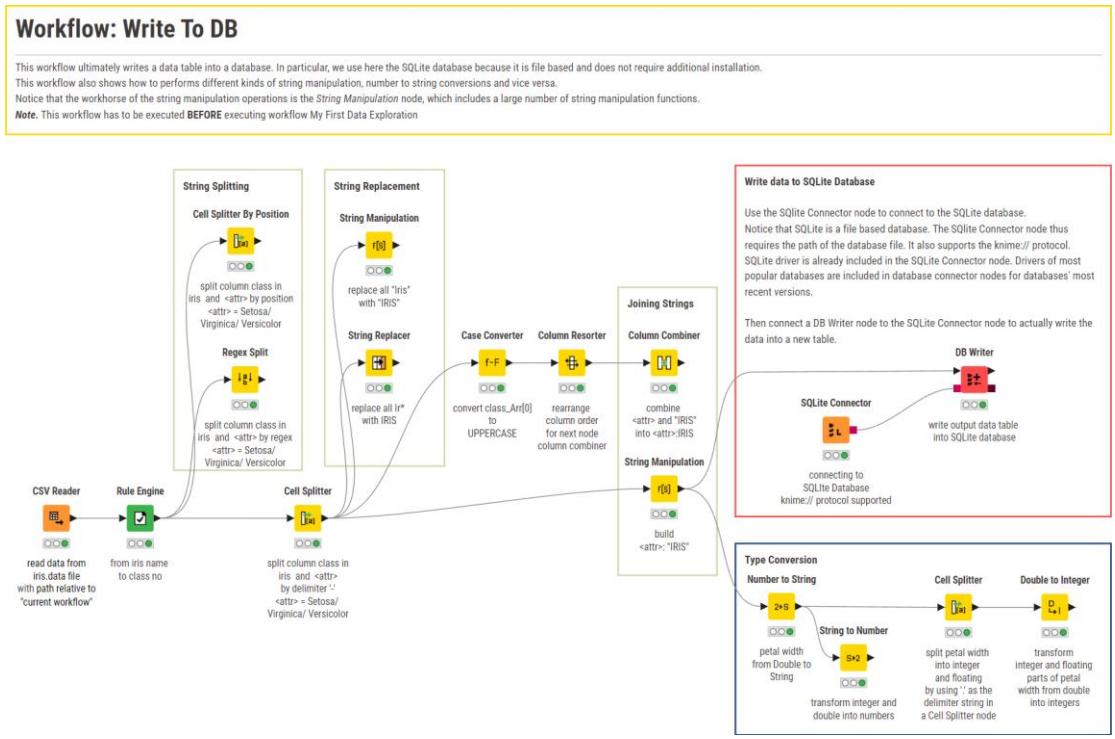



*Figure 3.24. Workflow "Write To DB".*

## DB Table Selector

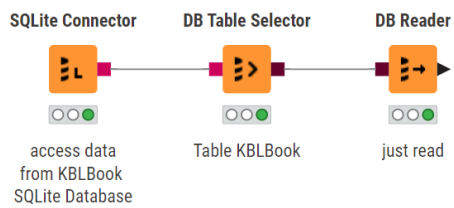



*Figure 3.25. SQLite Connector node + DB Table Selector node + Database Reader node to read data from an SQLite database.*

The node "DB Table Selector", located in "DB"/"Query", selects a table from the database provided with the database connection at its input port. Configuration settings require:

- The name and optionally schema of the database table. The "Select a table" button allows you to browse the content of the database to select the desired table.
- The default query is "SELECT * FROM #table#" where #table# is the selected table, which includes all data rows and all data columns of the table.
- It is also possible to extract a part or a transformation of the original #table#, by creating a custom query. The query editor appears when the flag "Custom Query" is selected.

- In the SQL editor (if available) you can then write your customized query to extract the data from #table#

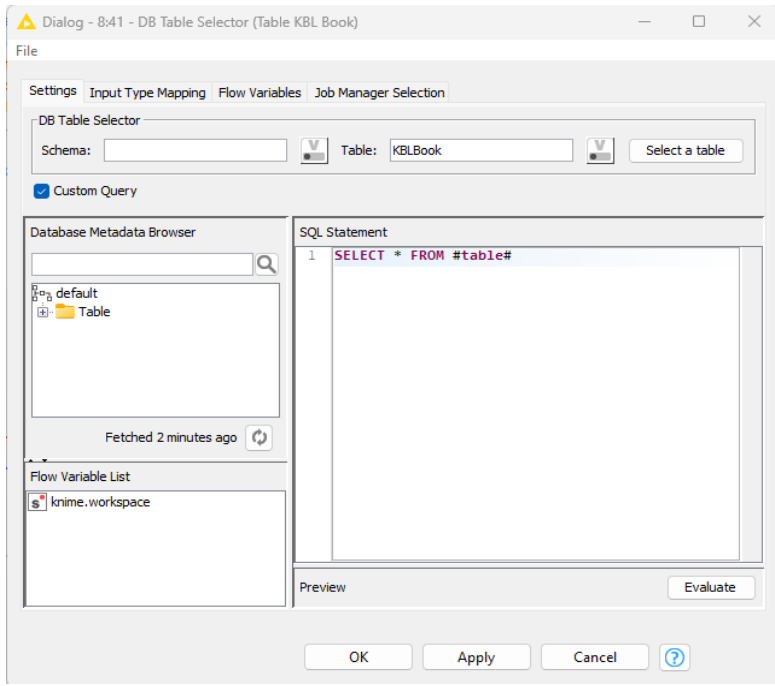



*Figure 3.26. Configuration of the Database Table Selector when following a Database Connector node.*

After the SQLite Connector node, the DB Table Selector node selects the table named KBLBook inside the database using the default query. We will work on the data of this table from now on.

## DB Reader

The node “DB Reader”, located in the “DB”/”Read/Write” category, reads a table from a database according to the input SQL query and imports it into a workflow.

Since the “DB Reader” node receives all information (connection to the database and SQL query) at the input port, it does not require any additional settings to run.

So, the only settings in the configuration window refer to the storage of the resulting data table.

The node reads all the columns from the table KBLBook in database KBLBook.sqlite:

- 4 columns -- “sepal width”, “sepal length”, petal width”, and “petal length” -- of data type “Double” come directly from the Iris dataset
- 1 column -- “class” -- represents the iris class and comes from the Iris dataset
- 1 column specifies the class number (“class 1”, “class 2”, and “class 3”) and was introduced earlier to show how the “Rule Engine” node works
- The remaining 3 columns are substrings or combination of substrings of the column called “class”. They were introduced as examples of string manipulation operations.

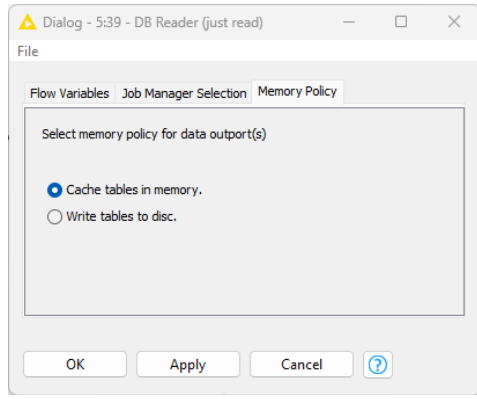



*Figure 3.27. Configuration window of the DB Reader node.*

## 3.7. Aggregations and Binning

As an example, let’s investigate the distribution of feature sepal_length across the whole data set. We will approximate this distribution visually with a histogram. The histogram needs ranges of values (bins) on which to count the number of occurrences. So, before proceeding with the drawing of the histogram, we define such bins on sepal_length value range. To do that, we use a “Numeric Binner” node.

We chose to build the histogram on the values of sepal_length only. We defined 9 bin intervals: “< 0”, “[0,1[“, “[1,2[“, “[2,3[“, “[3,4[“, “[4,5[“, “[5,6[“, “[6,7[“, and “>= 7”. A square bracket at the outside of the interval means that the delimiting point does not belong to the interval. We also decided to create a new column for the binned values. The column containing the bins was named “sepal_length_binned”.

We now want to count the number of iris plants for each species and with the “sepal_length” measure falling in one of the bins; that is, we want to count the number of iris plants by “sepal_length_binned” and by “class”.

In KNIME we can produce an aggregation of values based on groups and we can report the final aggregation values on tables with different structures by using two different nodes: the GroupBy node and the Pivoting node. Both nodes (“GroupBy” and “Pivoting”) are located in the “Node Repository” panel in the “Manipulation” → “Row” → “Transform" category.

Both nodes are quite important in the KNIME node landscape, since they are quite flexible and allow for a number of different aggregation operations, from simple row counting to the calculation of statistical measures, from correlation to value concatenation.

Both nodes group the input data according to the values in some selected columns and on the defined groups calculate a number of aggregation measures. The only difference is in the shape of the aggregated output data table. In the results of the "GroupBy" node each aggregation group is identified by the values in the first columns, while the final column contains the aggregated measure relative to that group. In the resulting table of the "Pivoting" node, each cell contains the aggregation measure for the group identified by the values in its column header and in its RowID. Given the importance of both nodes, we used both of them.

We set "sepal_length_binned" and "class" to identify the different group and we used "count" as aggregation measure on "sepal_length" column. "count" counts the rows in the defined group, that is for all irises in "class" iris-virginica with "sepal_length" between 6 and 7.

## Numeric Binner

The "Numeric Binner" node - located in the "Node Repository" panel in "Manipulation" → "Column" → "Binning" category - defines a series of intervals (i.e. bins) and assigns each column value to its bin.

The configuration window requires the following:

- The numerical column to be binned
- The list of bin intervals
- A flag to indicate whether the binned values should appear in a new column or replace the original column

To define a new bin interval:

- Click the "Add" button
- Customize the bin range in the Bin Editor

To edit an existing bin interval:

- Select the bin interval in the list of bin intervals
- Customize the bin range in the Bin Editor

- You can build a new bin representation by selecting another column and repeating the binning procedure.

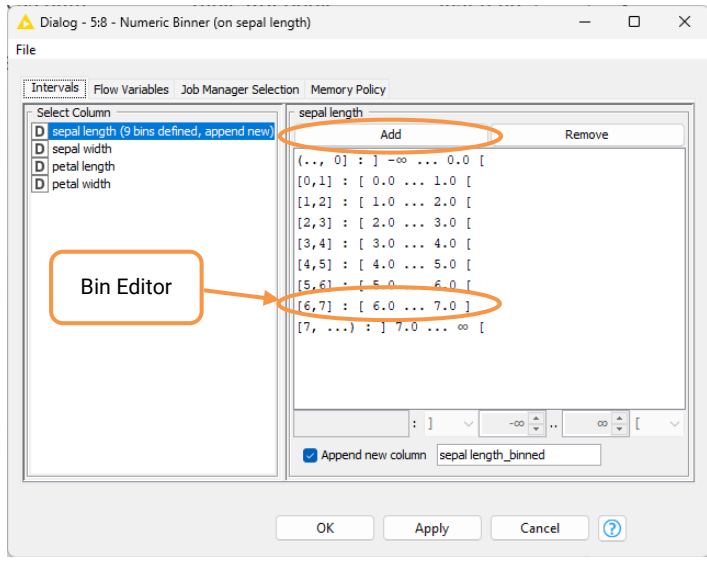



*Figure 3.28. Configuration window for the Numeric Binner node.*

***Note.*** The Aggregation method "count" just counts the rows in the group. It makes no difference which column it uses to count the rows, if we do not exclude those with missing values. However, this is the only aggregation method with this particularity. All other methods, such as average or sum or standard deviation, will of course produce different results when applied to different columns.

## GroupBy

### "Groups" Tab

The "GroupBy" node finds groups of data rows by using the combination of values in one or more columns (Group Columns); it subsequently aggregates the values in other columns (Aggregation Columns) across those groups. Column values can be aggregated in the form of a sum, a mean, just a count of occurrences, or using other aggregation methods (Aggregation Method).

The configuration window of the “GroupBy” node consists of a number of tabs. Here we check the tab named “Groups”. Tab “Groups” defines the grouping options. That is, it selects the group column(s) by means of 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.

The lower part of the configuration window

- sets the name of the new column
- keeps the row order or resorts them in alphabetical order
- rejects columns with too many different distinct values (default 10000), therefore generating too many different distinct groups
- option “Enable hilting” refers to a feature available in the old “Data Views” node.

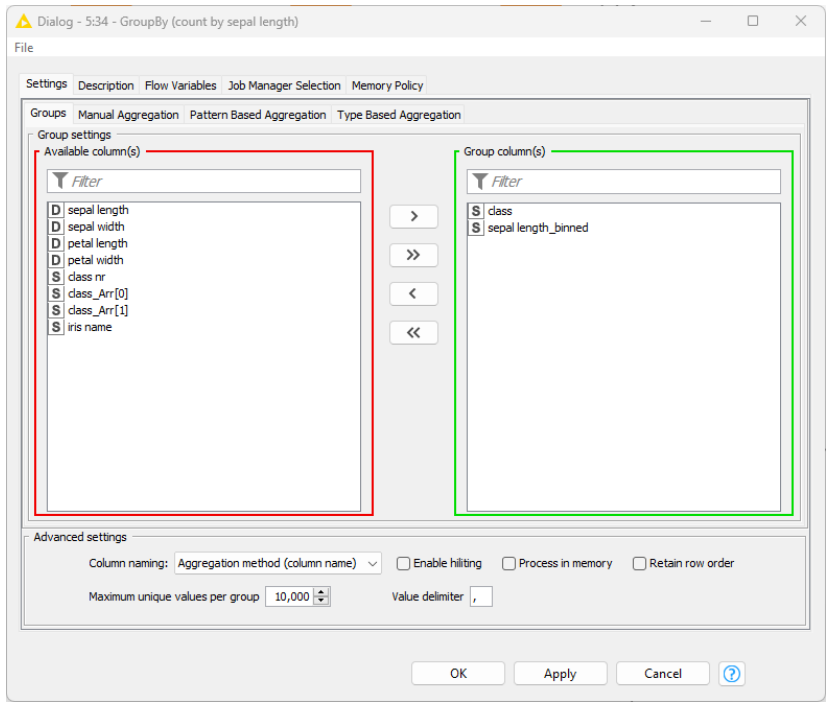



*Figure 3.29. Configuration window for the GroupBy node: the "Groups" tab.*

## “Aggregation” Tabs

The remaining tabs in the configuration window define the aggregation settings, that is:

- The aggregation column(s)
- The aggregation method (one for each aggregation column)

The different tabs select the columns on which to perform the aggregation using different criteria:

- Manually, one by one, by clicking on add button or clicking on add all button to select all the columns for aggregation
- Based on a regex or wildcard pattern: all columns with name matching the pattern will be used for aggregation
- Based on column type: all columns of the selected type will be used for aggregation

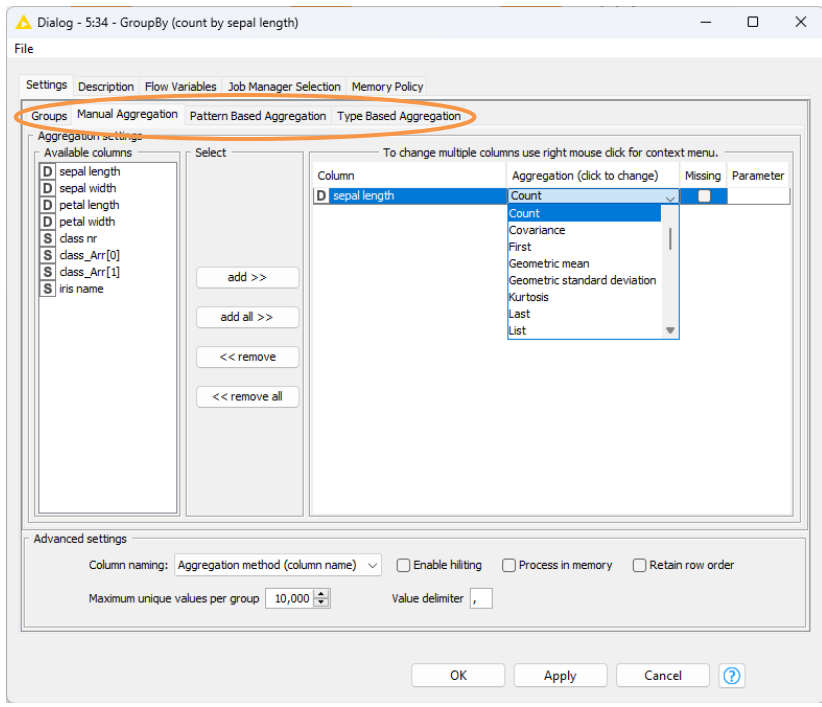



*Figure 3.30. Configuration window for the GroupBy node: the "Manual Aggregation" tab.*

Several aggregation methods are available in all aggregation tabs. All available aggregation methods are described in detail in the “Description” tab.

Aggregation methods differ for numerical columns (including here statistical measures, like mean, variance, skewness, median, etc.) and for String columns (including unique count for example).

Notice that aggregation methods “Count” and “Percent” just count the number of data rows for a group and its percent value with respect to the whole data set. That means that whichever aggregation column is associated with these two aggregation methods, the results will not change, since counting data rows of one group and its percentage does not depend on the aggregation column but only on the data group.

Aggregation methods “First” and “Last” respectively extracts the first and last data row of the current group.

The most frequently used aggregation methods for numerical columns are: Maximum, Minimum, Mean, Sum, Variance, and Sum. The most frequently used aggregation methods for nominal columns are: Concatenate, [Unique] List, and Unique Count.

## Pivoting

The “Pivoting” node finds groups of data rows by using the combination of values from two or more columns: the “Pivot” columns and the “Group” columns. It subsequently aggregates the values from a third group of columns (Aggregation Columns) across those groups. Column values can be aggregated in the form of a sum, a mean, just a count of occurrences, or a number of other aggregation methods (Aggregation Methods).

Once the aggregation has been performed, the data rows are reorganized in a matrix with “Pivot” column values as column headers and “Group” column values in the first columns.

The “Pivoting” node has one input port and three output ports:

- The input port receives the data
- The first output port produces the pivot table
- The second output port produces the totals by group column
- The third output port presents the totals by pivot column

The “Pivoting” node is configured by means of three tabs: “**Groups**”, “**Pivots**”, and “**Manual Aggregation**”.

Tab “**Groups**” defines the group columns by means of 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.

The lower part of the configuration window

- sets the name of the new column
- keeps the row order or resorts them in alphabetical order
- rejects columns with too many different distinct values (default 10000), therefore generating too many different distinct groups
- option “Enable hiliting” refers to a feature available in the old “Data Views” nodes

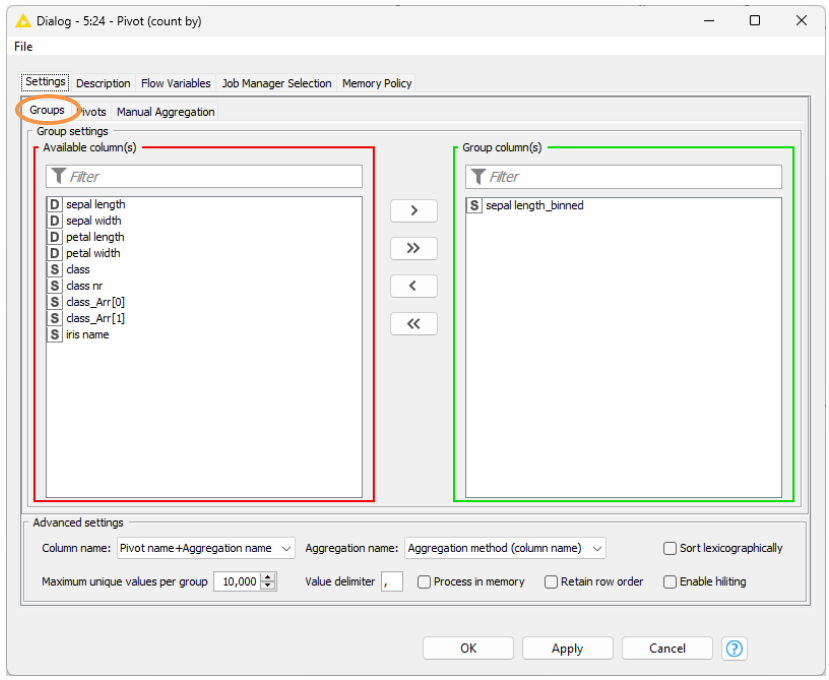



*Figure 3.31. Configuration window of the Pivoting node: the "Groups" tab.*

Tab “**Pivots**” defines the Pivot columns by means of 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 “Pivot 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.

At the end of this tab window there are three flags:

- “**Ignore missing values**” ignores missing values while grouping the data rows
- “**Append overall totals**” appends the overall total in the output table “Pivot totals”
- “**Ignore domain**” groups data rows on the basis of the real values of the group and pivot cells and not on the basis of the data domain. This might turn out useful when there is a discrepancy between the real data values and their domain values (for example after using a node for string manipulation).

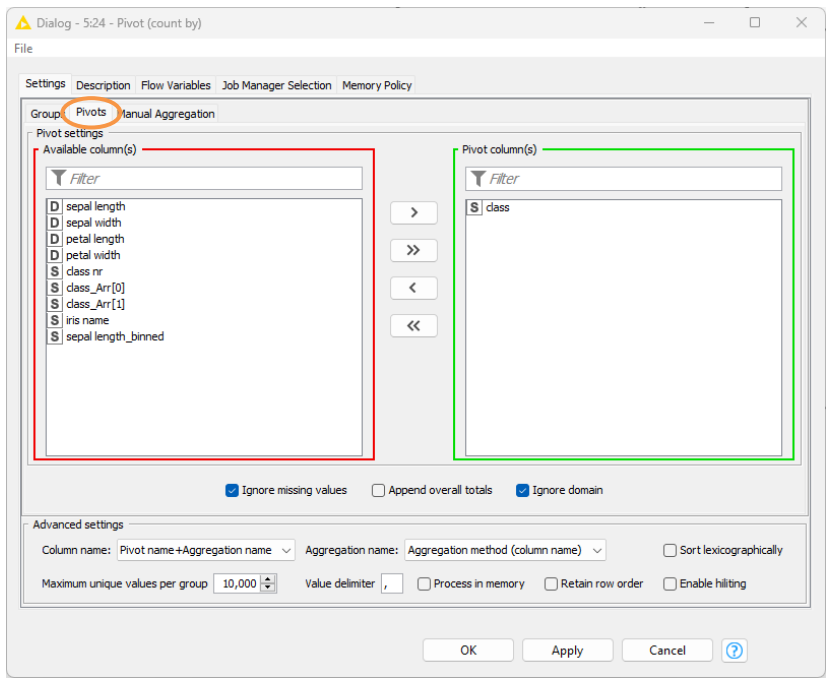



*Figure 3.32. Configuration window of the Pivoting node: the "Pivots" tab.*

Tab “**Manual Aggregation**” selects the aggregation columns and the aggregation method for each aggregation column. The column selection is again performed by means of clicking on the arrow buttons between the two windows.
For each selected aggregation column, you need to choose an aggregation method. Several aggregation methods are available. They are all described in the “Description” tab.

Aggregation methods “Count” and “Percent” just count the number of data rows in a group and therefore they are independent of the associated aggregation column.

Once the aggregation has been performed, the data rows are reorganized in the pivot table as follows:

- Column headers = <pivot columns distinct values> + <aggregation variable name selected criterion>
- First columns = distinct values in the group columns

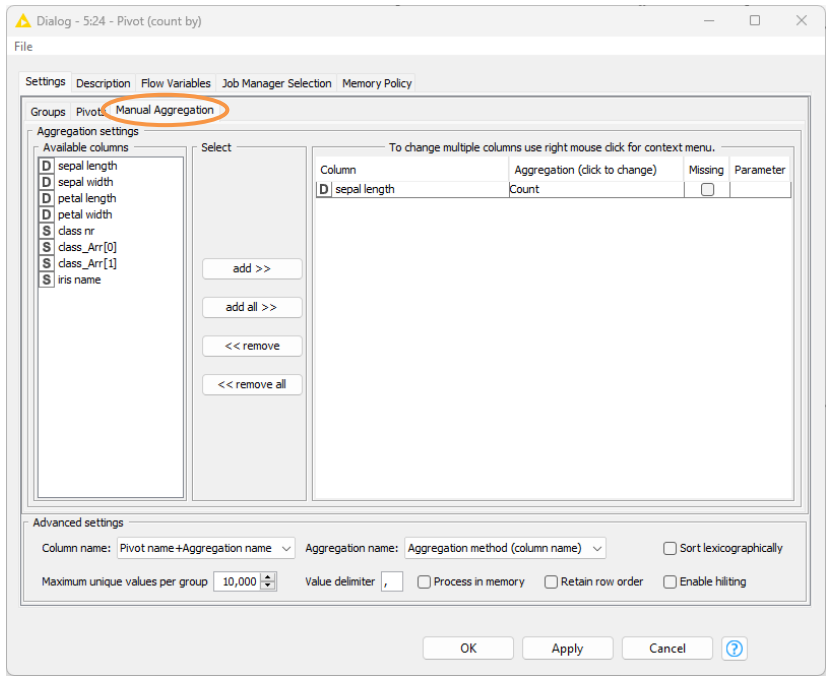



*Figure 3.33. Configuration window of the Pivoting node: the "Manual Aggregation" tab.*

## 3.8. Nodes for Data Visualization

Let’s move now into the data exploration and data visualization part through the graphic functionalities of KNIME Analytics Platform. For historical reasons, there are three possible ways to graphically represent data in KNIME: Data Views nodes, JFreeChart based nodes, and JavaScript based view nodes.

Data Views nodes are the newest nodes in KNIME Analytics Platform 5. They are located in the category “Views” in the “Node Repository”. These nodes get a data table as input and produce

a temporary graphical representation of the data, i.e., a view. Only the new visualization nodes are compatible with the new reporting framework.

JFreeChart nodes are located under “Views”/”JFreeChart” category in the “Node Repository”. These nodes are based on the Java JFreeChart graphical libraries. They are similar in contents and tasks to the Data Views nodes, but they produce a static image rather than a temporary view of the data graphical representation. The static image is exported into the KNIME workflow and can be used later on for reports, but not for interactive exploration of the data structure.

KNIME Analytics Platform also consists of the JavaScript based nodes. These nodes, located in “Views”/”JavaScript”, are based on JavaScript graphical libraries. These nodes produce a data table and a static image. The output data table is a copy of the input data table plus a column containing the selection flag for each data point. The output image is a screenshot of the node graphical view. It can be exported into the workflow for reporting using BIRT, but not with the new reporting framework.

## Scatter Plot

Let’s start our data exploration with a classic scatter plot. The node to use here is the “Scatter Plot” node.

The “Scatter Plot” node plots each data row as a dot by using two of its attributes as coordinates on the X-axis and the Y-axis. After reading the iris data set from the KBLBook.sqlite database, we want to produce a scatter plot of petal length vs. petal width, which is the view where the three groups of iris flowers are best recognizable.

The configuration window of a “Scatter Plot” node covers 5 sections: “Data”, “Plot”, “Reference Lines”, “Interactivity”, and “Image Generation”. The “Data” section defines columns to report to the x- and y-axis, columns to report on “color dimension”, and the maximum number of data rows to be visualized. The “Plot” section specifies the image options, such as the title, the axes limits, the choice of selecting axes scales, and the axes labels. The “Reference Lines” section allows to plot a reference line at a specific value on the y-axis. This section allows to change the line's label, border style, color, and size. The “Interactivity” section defines the allowed interactivity on the final view, such as the possibility of downloading the image, zooming, information display on hovering, and selection of points. The “Image Generation” section is a checkbox to reproduce a view into an image at the output port.

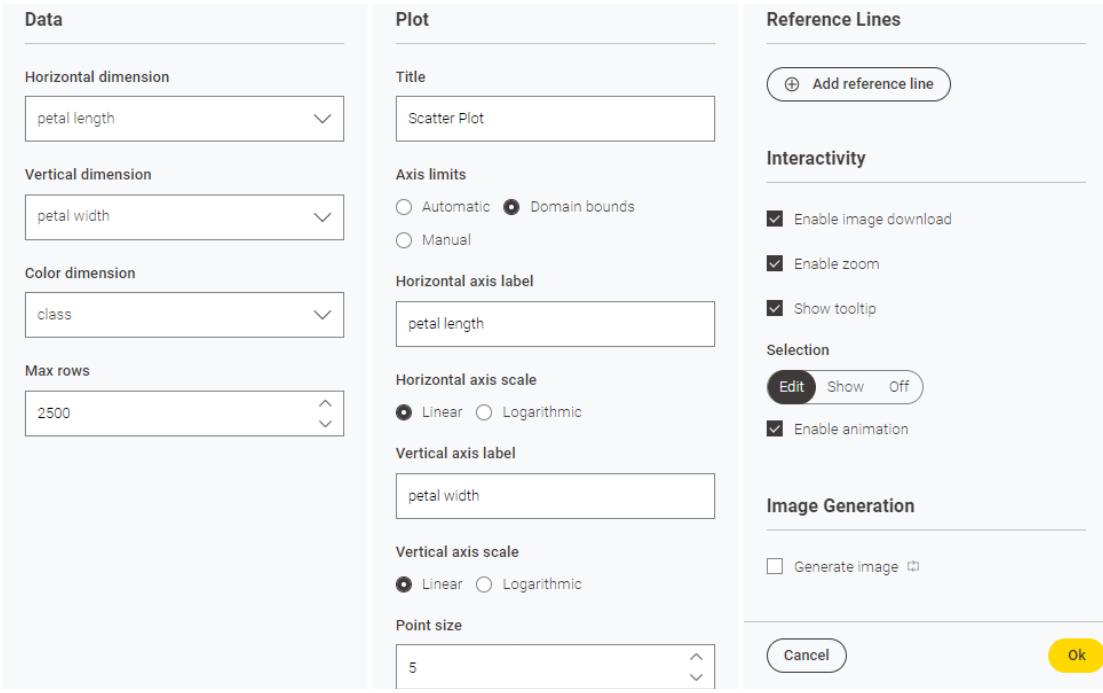



*Figure 3.34. Configuration window of the Scatter Plot node*

After execution, the node produces an interactive view. Right-click the node and select “Interactive View: Scatter Plot”. The level of interactivity of this view was decided in the settings of the “Interactivity” section of the node configuration window. Let’s explore this view and let’s see the kind of interactivity it allows.

The view of the “Scatter Plot” node opens using the settings of the configuration window. In our case, opens on petal length vs. petal width, with such axis label, no title, wheel zooming, and simple and rectangular selection enabled.

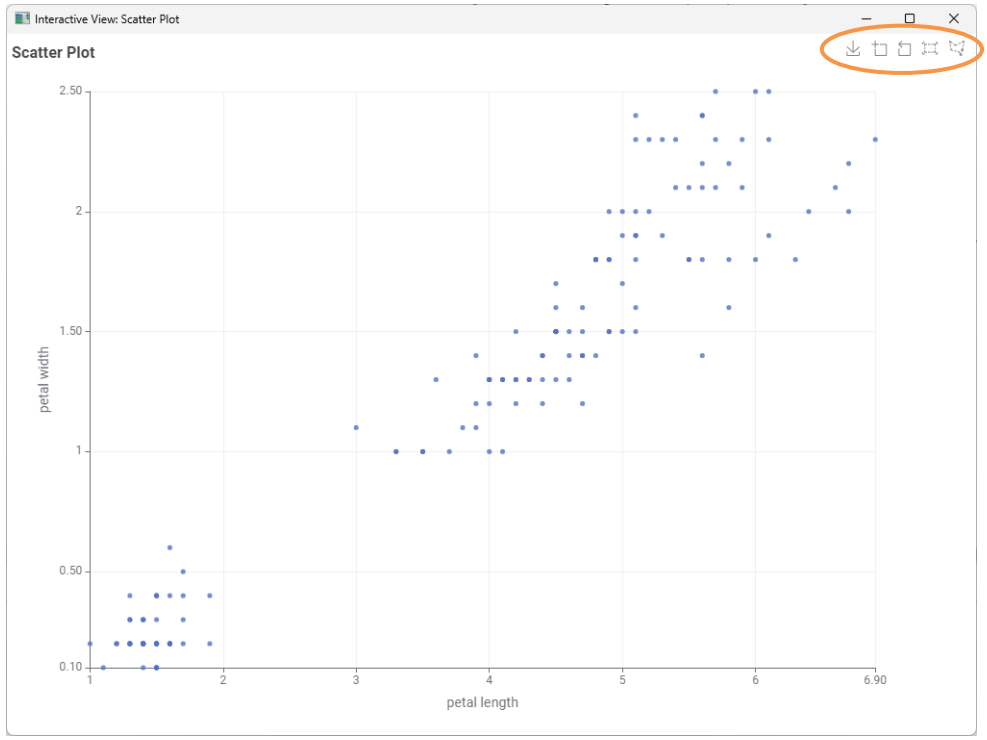



*Figure 3.35. The Interactive View of the Scatter Plot node.*

## Scatter Plot: Interactive View

This is the view of the *Scatter Plot* node, where you can see the dots of the scatter plot. There are five  buttons in the upper right corner. Those are the interactivity buttons.  There are five buttons in the upper right corner. Those are the interactivity buttons. Starting from the far left, the “Save Image” button lets you download the image locally. The second button, the “Zoom” button, zooms in when you click on a specific point in the plot. The third button, the “Zoom reset” button, resets the zoomed-in part to the default view. The fourth button from the right, “Box select”, allows a box-shaped selection on the plot. The fifth button is the “Lasso select” button, which allows us to select specific parts in a freeform. Additionally, scrolling back and forth on the plot in the interactive view lets you zoom in and out.

***Note.*** The flag “create image at output port” in the “Options” tab might slow down the node execution if the image is built on a larger number of input records. In this case, you might consider disabling this flag in the interest of speed execution.

Input data and view selection - 0:20 - Scatter Plot (JavaScript) (petal length)

File Hilite Navigation View

Table "default" - Rows: 150 | Spec - Columns: 10 | Properties | Flow Variables

| Row ID | sepal_l... | sepal_... | petal_l... | petal_... | class | class_nr | class_A... | class_A... | iris_name | Sel... |
|---|---|---|---|---|---|---|---|---|---|---|
| Row133 | 6.3 | 2.8 | 5.1 | 1.5 | Iris-virginica | class 3 | Iris | virginica | virginica:IRIS | true |
| Row137 | 6.4 | 3.1 | 5.5 | 1.8 | Iris-virginica | class 3 | Iris | virginica | virginica:IRIS | true |
| Row138 | 6 | 3 | 4.8 | 1.8 | Iris-virginica | class 3 | Iris | virginica | virginica:IRIS | true |
| Row139 | 6.9 | 3.1 | 5.4 | 2.1 | Iris-virginica | class 3 | Iris | virginica | virginica:IRIS | true |
| Row142 | 5.8 | 2.7 | 5.1 | 1.9 | Iris-virginica | class 3 | Iris | virginica | virginica:IRIS | true |
| Row146 | 6.3 | 2.5 | 5 | 1.9 | Iris-virginica | class 3 | Iris | virginica | virginica:IRIS | true |
| Row147 | 6.5 | 3 | 5.2 | 2 | Iris-virginica | class 3 | Iris | virginica | virginica:IRIS | true |
| Row149 | 5.9 | 3 | 5.1 | 1.8 | Iris-virginica | class 3 | Iris | virginica | virginica:IRIS | true |
| Row0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row1 | 4.9 | 3 | 1.4 | 0.2 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row4 | 5 | 3.6 | 1.4 | 0.2 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row5 | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row6 | 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row7 | 5 | 3.4 | 1.5 | 0.2 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row8 | 4.4 | 2.9 | 1.4 | 0.2 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |
| Row9 | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa | class 1 | Iris | setosa | setosa:IRIS | false |

*Figure 3.36. "true" and "false" values in the additional column named "Selected Scatter Plot" and produced by the Scatter Plot node. "true" is associated to all selected records. "false" indicates a not selected records and it is the default value.*

## Graphical Properties

Graphical plots in node views can be customized with color, shape, and size of the plot's markers. KNIME Analytics Platform has three nodes, in "Views" → "Property" in the "Node Repository" panel, to customize plot appearance: "Color Manager", "Size Manager", and "Shape Manager". These nodes take a data table as input and produce two objects at two separate output ports.

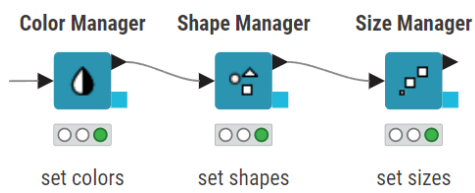



*Figure 3.37. The three views properties nodes, to set color, shape, and size of plot markers.*

- The first output port contains the same data table from the input port, with the additional graphical properties as color, size, and/or shape assigned to each data row.
- The second output port contains the graphical model; that is the color, shape, or size adopted for each record. This graphical model can be passed on to the "Size Appender" node and then can be applied to another data set.

Let's have a look at the *Color Manager* node as an example of how these graphical property nodes work.

## Color Manager

The “Color Manager” node assigns a color to each row of a data table depending on its value in a given column.

If a nominal column is selected in the configuration dialog, colors are assigned to each one of the nominal values.

If a numerical column is selected, a color heat map spans the column numerical range.

The configuration window requires:

- The column from which to extract values (nominal columns) or ranges (numerical columns)
- The color map for each list of values or range of values
- A default color map is assigned by default to the list / range of values. This can be changed by selecting the value / range and then assigning a different color from the color map displayed in the lower part of the configuration window.

Similarly to the *Color Manager* node, in the configuration window of the *Shape Manager* node, shape can be changed by clicking the row with the desired column value and assigning a shape from the menu list on the right.

The *Size Manager* node on the opposite uses a multiple of an input numerical column to scale the size of the plot markers. Its configuration window then requires the numerical column and the factor to use for the scaling operation.

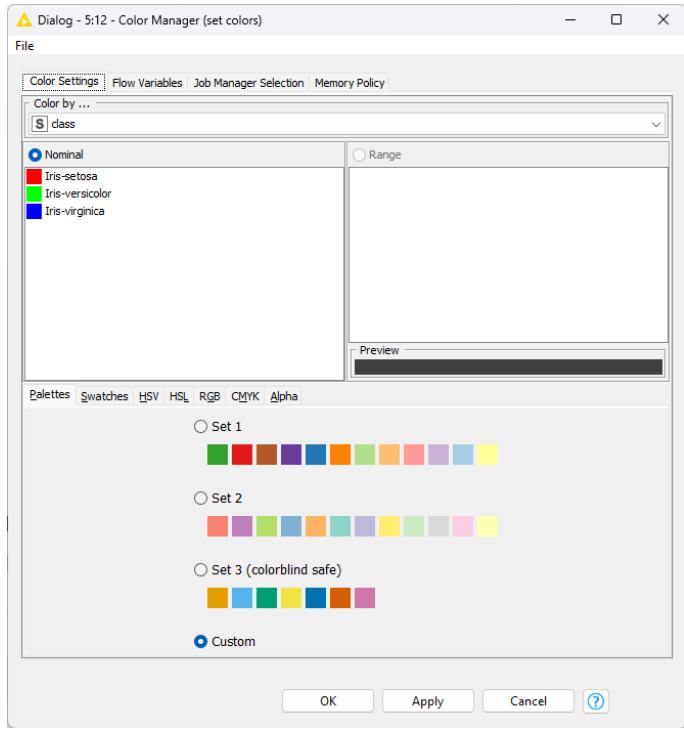



*Figure 3.38. Configuration window of the Color Manager node.*

***Note.*** As of KNIME Analytics Platform 5.2, the *Size Manager* node and the *Shape Manager* node are supported by the visualization nodes.

This time, a *Color Manager* node was applied to the original iris data before feeding the scatter plot node. In the configuration window we selected the "class" column for the marker assignment, and we allocated different colors to each one of the three iris labels found in the "class" column. The introduction of this graphical property transforms the scatter plot – reported above in black and white – into the following scatter plot.

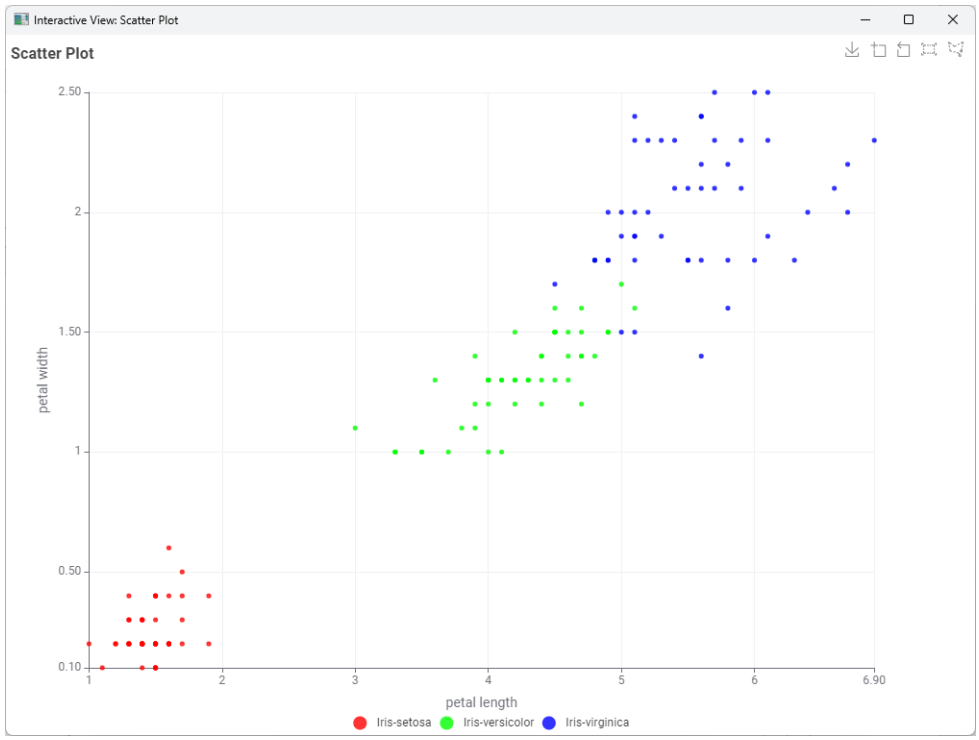



*Figure 3.39. The view of the Scatter Plot node with customized colors for plot dots.*

## Line Plots and Parallel Coordinates

Another useful plot is the line plot, to draw time series and other evolving phenomena along one dimension only. A line plot connects attribute values sequentially, i.e., following their order in the input data table. The row sequence represents the X-axis, while the corresponding attribute values are plotted on the Y-axis. Multiple lines, i.e., multiple columns, can be reported in the plot.

A line plot is usually developed over time, i.e., the row sequence represents a time sequence. This is not the case with the iris data set, where rows represent only different iris examples and have no temporal relationship. Nevertheless, we are going to use this workflow to show how a *Line Plot* node works.

## Line Plot

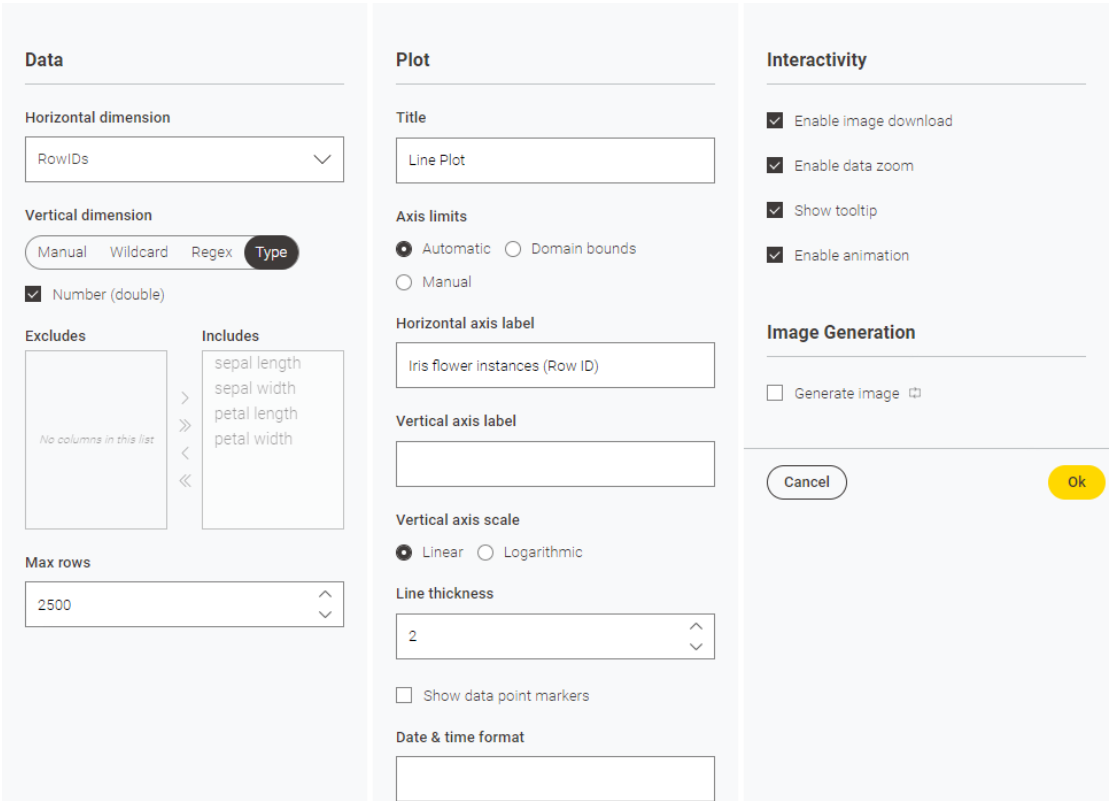



*Figure 3.40. Configuration window of the Line Plot node: the "Options" tab.*

The “Line Plot” node displays a line plot, using one column as X-axis and one or more column values as Y-axis.

As for the previous new visualization nodes, the configuration window of the “Line Plot” node has four sections: “**Data**”; “**Plot**” for the plot details and “**Interactivity**”; and “**Image Generation**” for the generating the image at the output port.

The main difference is in the “Data” section, where an “Includes”/”Excludes” frame allows to select the columns for the plot.

The final view of the *Line Plot* node is shown in the following figure, where RowIDs are displayed on the X-axis and iris measures are displayed on the Y-axis. We used here RowIDs for the X-axis, but we could have used any other column for that. Whatever had been chosen to be reported in the X-axis, the plot would have still drawn the column values in sequence, in order of appearance in the input data table.

***Note.*** As of KNIME Analytics Platform 5.2, the *Line Plot* node does allow interactivity.

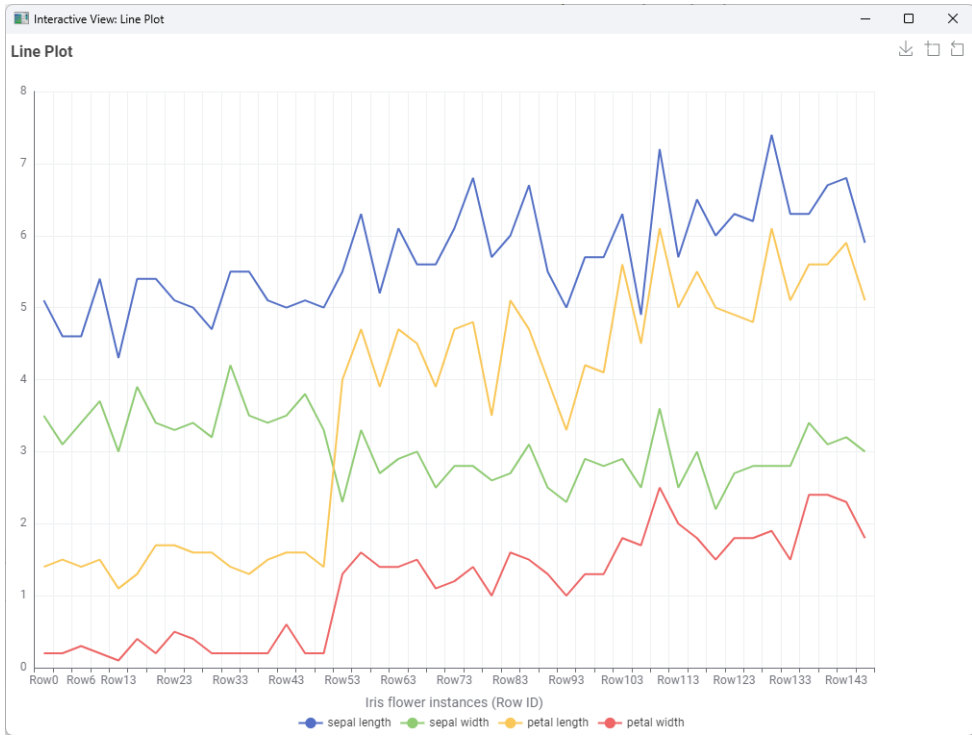



*Figure 3.41. Plot view of the Line Plot node.*

Another interesting plot is the Parallel Coordinates plot. Parallel coordinate visualization plots are useful to get an idea of pattern groups across columns. For example, for our iris data set, we can see that one of the iris classes gets easily separated from the other two along the coordinates “petal length” and “petal width”.

In the parallel coordinates plot, one column is one coordinate, i.e., one Y-axis. Multiple column values can be visualized on multiple coordinates, that is on multiple Y-axis. The data disposition along each axis can tell us some stories about the groups in the data set. The node that produces a parallel coordinate plot is the *Parallel Coordinates* node.

## Parallel Coordinates Plot

The “Parallel Coordinates” node displays the input data table in a parallel coordinates plot. A parallel coordinates plot unfolds the column names along the X-axis and displays each column value on a separated Y-axis. As a result a data point is mapped as a line connecting values across attributes.

The configuration window of this node has four sections.

- “**Data**” section contains an “Excludes/Includes” frame to insert/remove more columns (i.e. Y-axis) into/from the parallel coordinates plot. This section allows to decide the “color dimension” and maximum number of data rows to be included.
- “**Plot**” section defines general settings for the plot like the title, value axis limits, line shape, and line thickness.
- “**Interactivity**” section sets the interactivity level for the final view
- “**Image Generation**" section sets the option to generate the image at the output port.
- Line colors can come from a specific column containing the color as a graphical property (that is the result of the “Extract Color” node) or just from the graphical property associated to each row (flag “use color from spec”).

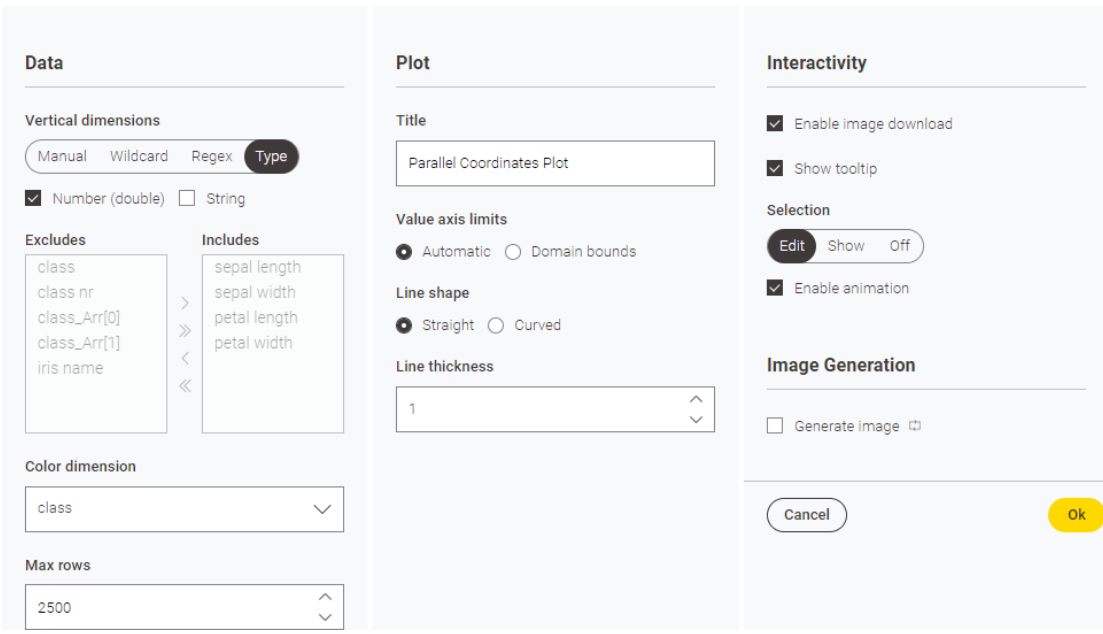



*Figure 3.42. Configuration window of the Parallel Coordinates node.*

Below is the view of the *Parallel Coordinates* node. As Y-axis we find: “sepal_length”, “sepal_width”, “petal_length”, “petal_width”. Each iris plant is then described by the line connecting its “sepal_length”, “sepal_width”, “petal_length”, and “petal_width” values. Line colors are determined by the color associated to each data row – i.e., to each iris plant – by the preceding *Color Manager* node.

Interactivity in the *Parallel Coordinates* node is also reduced with respect to, for example, the *Scatter Plot* node.

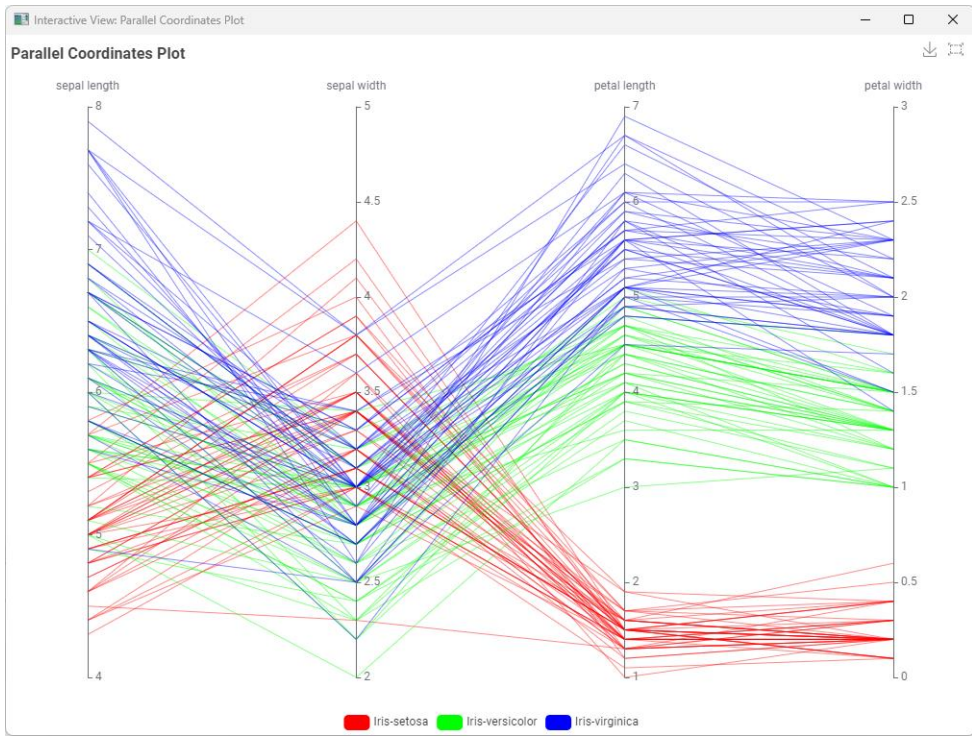



*Figure 3.43. View of the Parallel Coordinates plot, where the four Iris measures are displayed on the four Y-axis. One line corresponds to one Iris plant.*

## Bar Charts and Histograms

Of all the plots that are available to visually investigate the structure of the data, we cannot leave out the histogram. The histogram visualizes how often values in a given range (bin) are encountered in the value series. This section briefly takes a look at histograms and bar charts.

Properly speaking, there is not a dedicated JavaScript based node to draw a histogram plot. The histogram drawing functionality is hidden in the “Bar Chart” node.

We already binned the “sepal_length” attribute in 9 bins. Now each data row of the input data table is assigned to a given bin according to the value of its “sepal_length” attribute. To build the histogram of attribute “sepal_length”, it is enough to count the number of occurrences in each “sepal_length_binned” interval with a “Pivoting” node.

## Bar Chart

The “Bar Chart” node creates a generic bar chart. To do that, it needs:

- A category column, which in case of a histogram is the binned column
- An aggregation column and an aggregation method.

In the case of a histogram the aggregation method is “Occurrence Count. This just counts the data rows falling in each bin and therefore does not require a specific aggregation column.

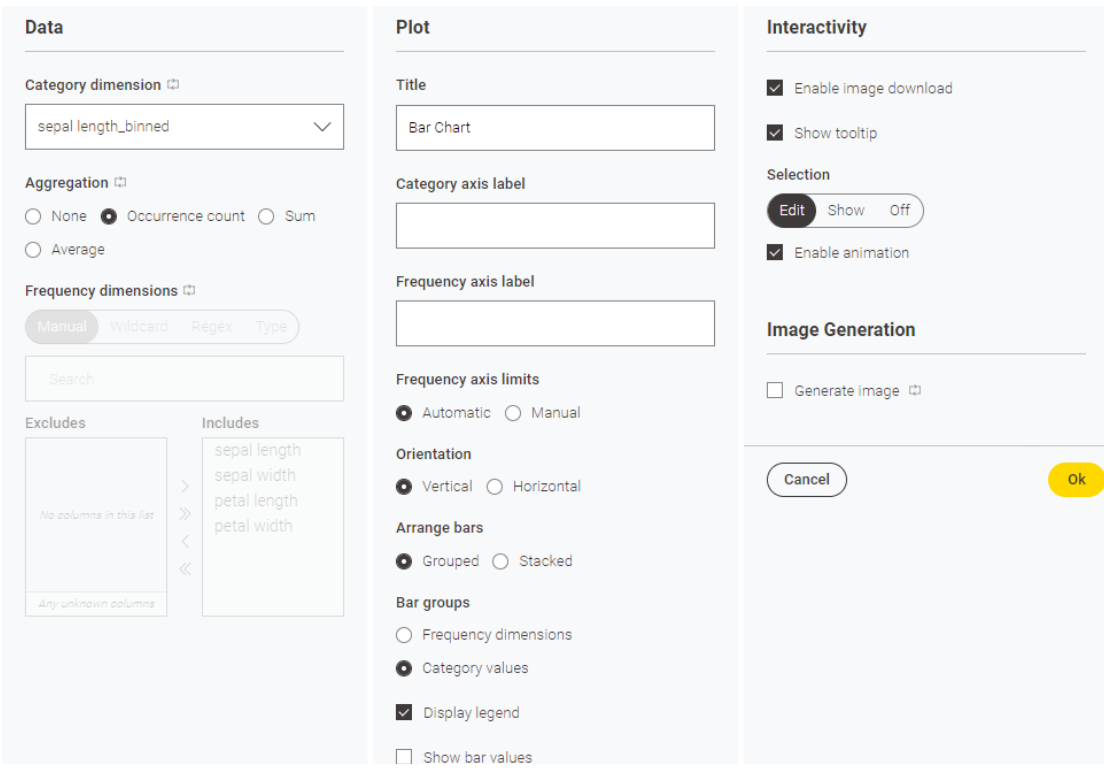



*Figure 3.44. Configuration window of the Bar Char node: the "Options" tab, configured to draw a histogram.*

These settings are all defined in the sections “Data” of the configuration window. Two additional tabs “Plot”, “Interactivity”, and “Image Generation” define respectively the plot

graphical details, enabled interactive view controls, and the option to generate the image at the output port. “Plot” section includes preferences for title, axis labels, plot orientation, legend, and an option to show the bar values. “Interactivity” section enabling image download, option to show tool tip when hovered on the chart, subscribing and publishing to selection views, and enabling animation. The “Image Generation” section allows to generate the image at the output port.

The “Bar Chart” node does not have an optional input port for a color map.

The “Histogram” view displays how many times the values of a given column occur in each interval (bin). The final histogram view is shown below.

> ***Note.*** The *Bar Chart* node does not sort the string categories on the X-axis. They are displayed in occurrence order. If we want them to be sorted, like in our case of binning intervals, a *Sorter* node needs to precede the *Bar Chart* node.

This histogram covers all instances of iris plants represented in the input data set. However, let’s suppose we want to isolate and compare the same histogram for the three separate classes: iris-setosa, iris-versicolor, and iris-virginica.

First, we need to separate the three groups and count the number of occurrences for each group and for each bin in “sepal_length” (“Pivoting” node); finally, we need to draw the counts into a bar chart (“Bar Chart” node with aggregation method “Average” on all three classes). The configuration window of the *Bar Chart* node and consequent histogram view are reported below.

The last node we would like to consider in this section is the “Table View” node. This node just displays the input data in a table.

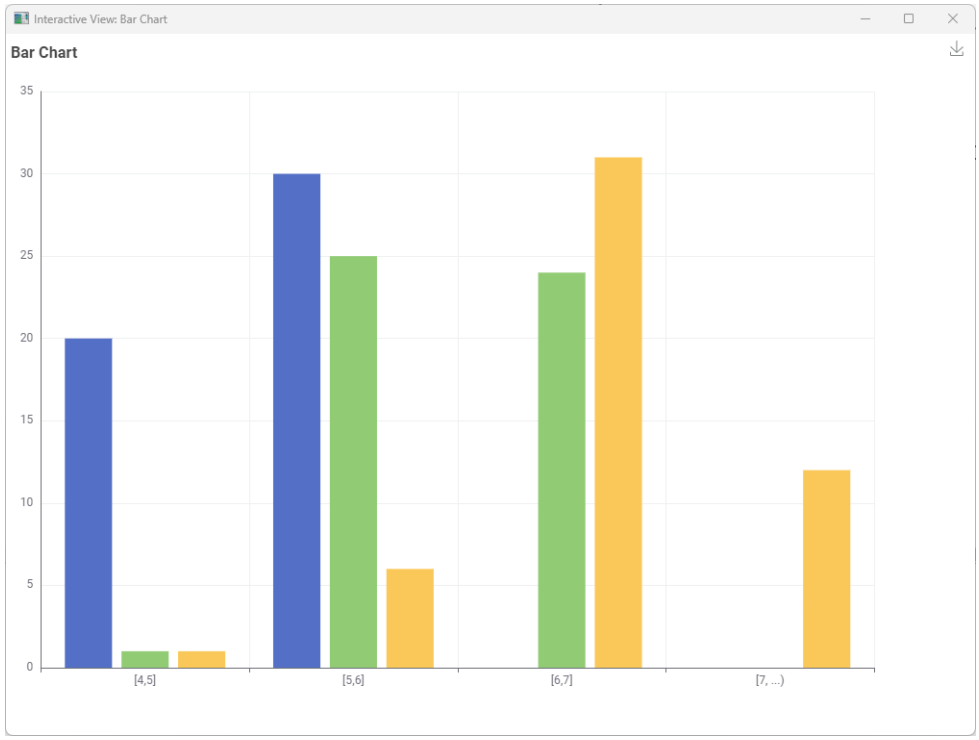



*Figure 3.45. The view of the histogram of "septal_length" obtained with a Bar Chart node, using "Occurence Count" as aggregation method.*

## Table View

The “Table View” node displays the input data in a table. The configuration window consists of three tabs:

- “**Data**” section defines the data selection, for example which columns to display
- “**View**” section contains the usual settings to change the title, adjust table pagination, and an option to compact rows in the table view.
- “**Interactivity**” section contains the usual settings to determine the level of interactivity in the produced view.

- Depending on the settings in the “Interactivity” section, the rows in the table view present a selection box on the left. In this way, it is possible to select only some of them. Selected rows will exhibit the flag “true” in the “Selected JavaScript Table View” appended column at the node output port.

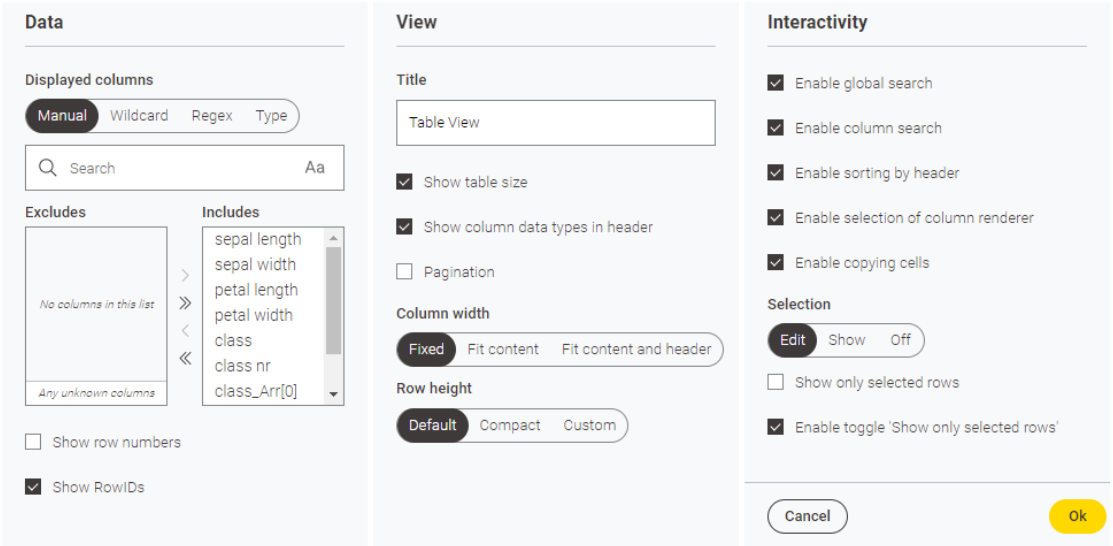



*Figure 3.46. Configuration window of the Table View node.*

This is where we finish our description of the nodes available in KNIME Analytics Platform for data visualization. There are a few additional interesting visualization nodes, such as “Lift Chart”, “Box Plot”, “ROC Curve”, “Pie Chart”, etc.

In particular, the “Generic JavaScript View” node allows for free JavaScript code. If you are a JavaScript expert and/or you prefer to use some specific JavaScript libraries, this is the node that allows to create arbitrarily complex JavaScript based graphics.

This is the final workflow “My First Data Exploration”.

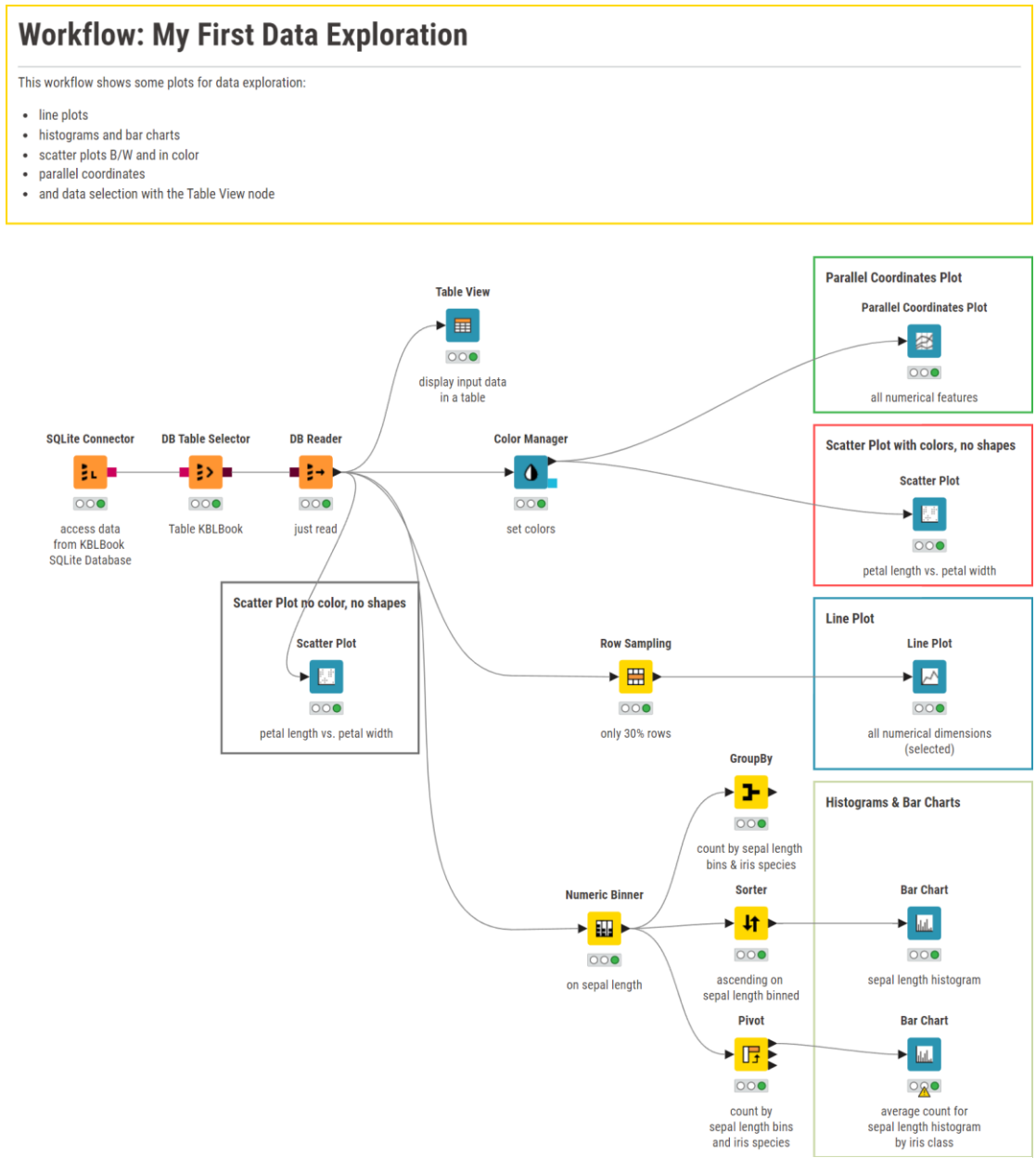



*Figure 3.47. The final version of the workflow "My First Data Exploration".*

# 3.9. Exercises

## Exercise 1

Read file “yellow-small.data” from the Balloons dataset (you can find this file in the KBLdata folder or you can download it from: https://archive.ics.uci.edu/ml/machine-learning-databases/balloons/). This file has 5 columns: “Color”, “Size”, “Act”, “Age”, and “Inflated”. Rename the columns accordingly. Add the following classification column and name it “class”:

```
IF      Color = yellow   AND    Size = Small      =>     class =inflated
ELSE                                               class = not inflated
```

Add a final column called “Final sentence” that says:

```
“inflated is T”
OR
“not inflated is F”
```

where “inflated/not inflated” comes from the “class” column and “T/F” from the “Inflated” column.

### Solution to Exercise 1

There are two ways to proceed in this exercise.

1. With a series of dedicated *String Manipulation* and *Rule Engine* nodes
2. With one *Rule Engine* node and one *String Manipulation* node with its functions

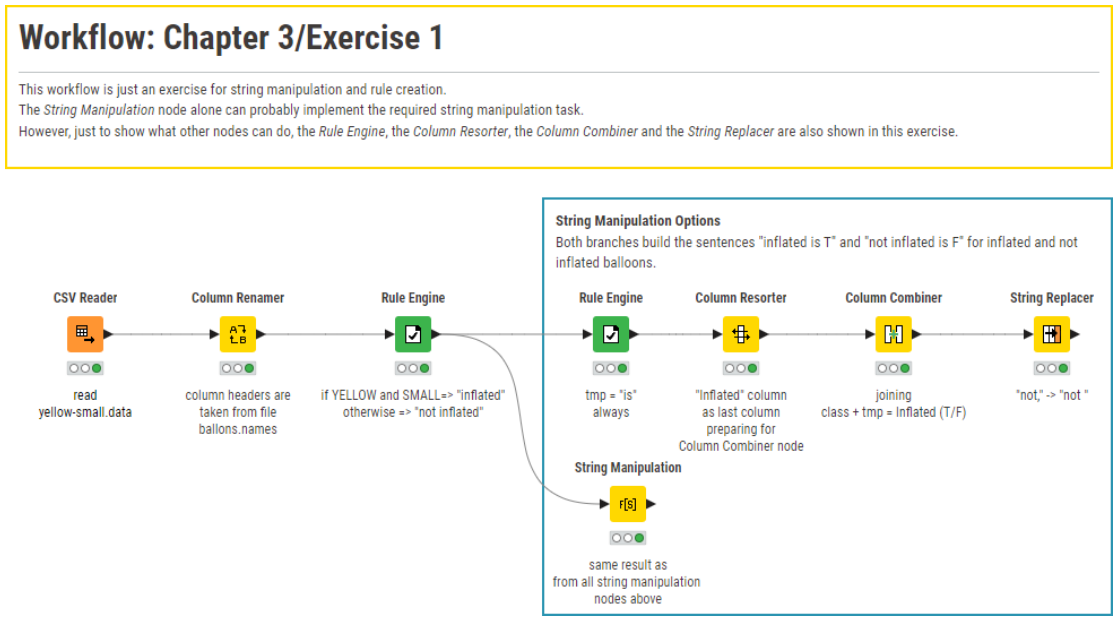



*Figure 3.48. Exercise 1: Workflow.*

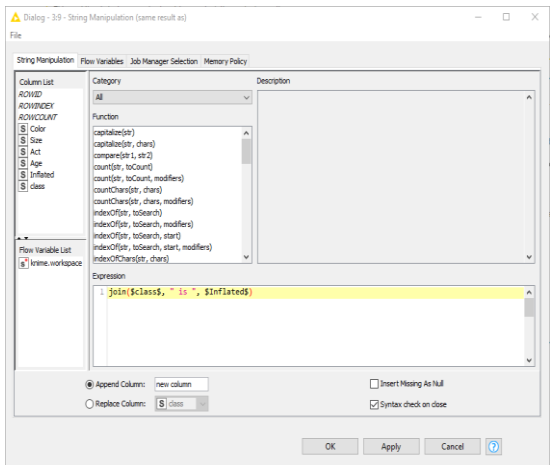



*Figure 3.49. Exercise 1: The String Manipulation node configuration (node commented with "same result as...").*

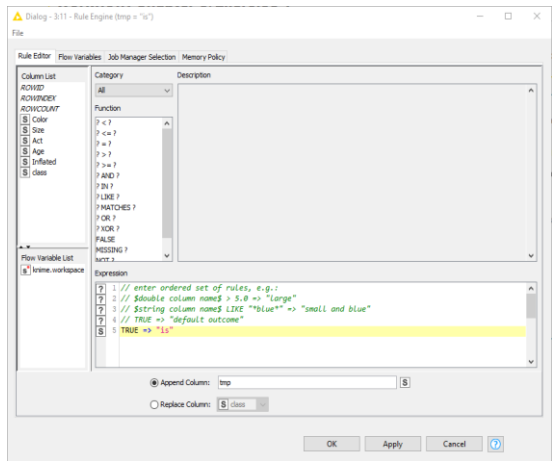



*Figure 3.50. Exercise 1: The Rule Engine node configuration (node commented with "if YELLOW and SMALL...").*

## Exercise 2

This exercise is an extension of Exercise 1 above. Write the last data table of workflow Exercise 1 into a table called “Chapter3Exercise2” in the SQLite database “KBLBook.sqlite”, using the *SQLite Connector* node and the *Database Writer* node.

### Solution to Exercise 2

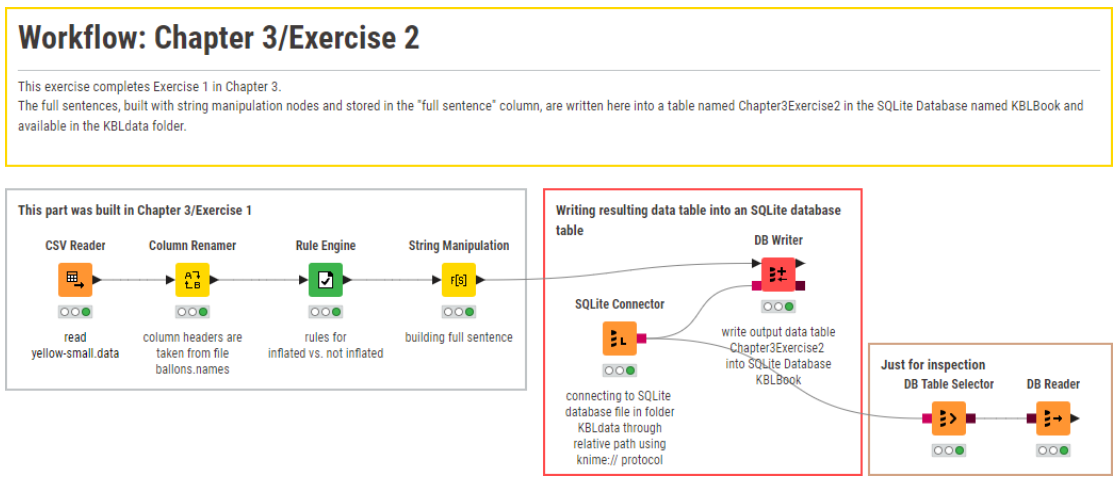



*Figure 3.51. Exercise 2: Solution Workflow.*

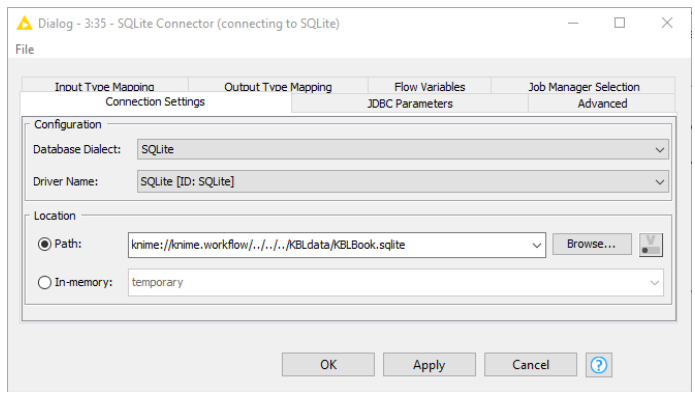



*Figure 3.52. Exercise 2: Configuration window of the SQLite Connector node.*

# Exercise 3

Read the adult.data file. From this data set display three plots:

- "Age" Histogram by sex on 10 age bins
- "Work class" Bar Chart as number of occurrences for each work class value
- "Age" vs. "hours per week" Scatter Plot

Build the histogram and the bar chart using a *Bar Chart* node and the scatter plot using a *Scatter Plot* node. In the "age" vs. "hours per week" scatter plot, select all points with "age" = 90 and extract them with a *Row Filter* node on column "Selected"(…)" = "true".

How many 90-year old people are included in the data set?

## Solution to Exercise 3

**Scatter Plot "age" vs. "hours per week":**

In order to make sure that all records are plotted we need to change the default value of the setting "Maximum Number of Rows" in the "options" tab of the configuration window of the "Scatter Plot" node. We need to make sure that this number is bigger than the number of records in the input data set. Plotting all records instead of only the default number will of course require a longer execution time.

In "Views Control" tab we need to enable rectangular selection. We open the node view, enable the selection button on the top right corner, and draw a rectangle around our 90-year old people on right of the scatter plot (if "age" has been placed on the x-axis). Then we click button "Close" in the lower right corner of the view and accept the changes.

A *Row Filter* node finally extracts the records with "Selected (…)" column = true. 43 points representing 90-year old people have been selected.

Optionally, we colored the dots in blue for male records and in red for female records with a "Color Manager" node.

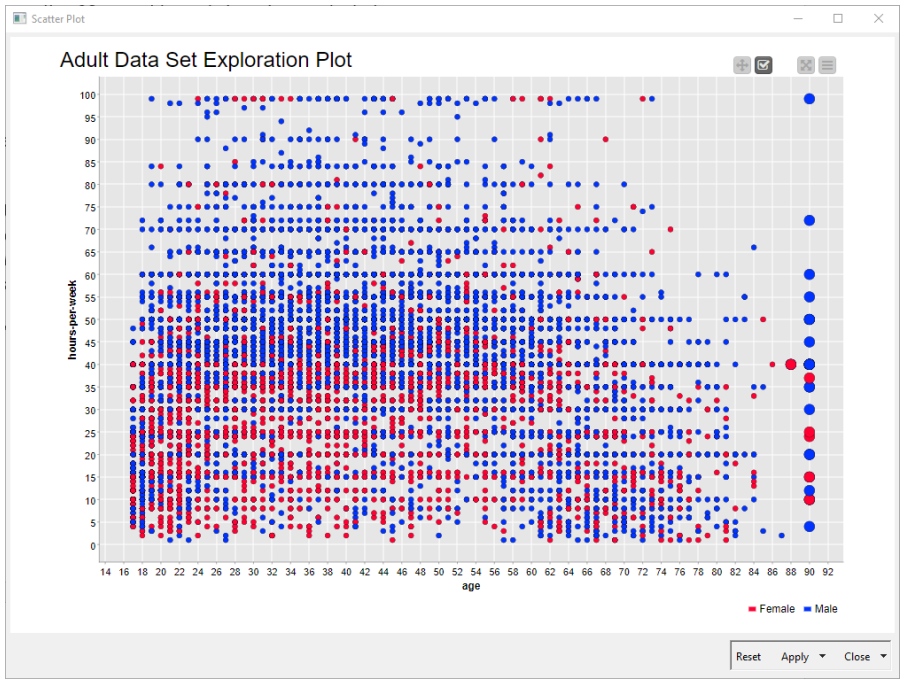



*Figure 3.53. Scatter Plot of "age" vs. "hours per week" for the adult dataset. 90-year-old people have been selected.*

**Bar Chart on Number of Occurrences in each work class:**

Here we used just a *Bar Chart* node counting number of occurrences on category “workclass”

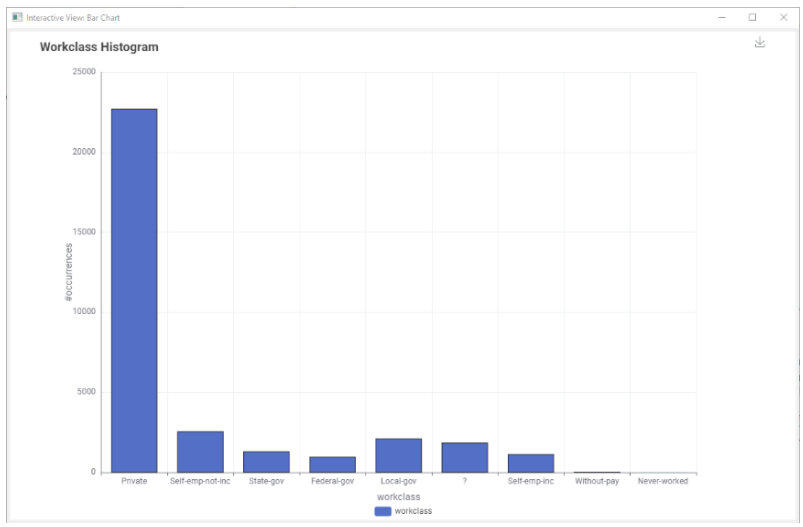



*Figure 3.54. Bar Chart of number of occurrences of "work class" values in the adult data set.*

**Age Histogram for Males and Females:**

First, we automatically build 10 age bins using the *Auto-Binner* node. Then we use a *Pivoting* node to count the number of occurrences for men and women in the different age bins. Using a *String Manipulation* node we change "[" into "(" for sorting purposes and then we sort the age bins in ascending order. Finally, a *Bar Chart* node displays the 2 numbers side by side for women and men. The side-to-side effect was obtained selecting "Grouped" as "Chart Type" setting in the "General Plot Options" tab in the node configuration window.

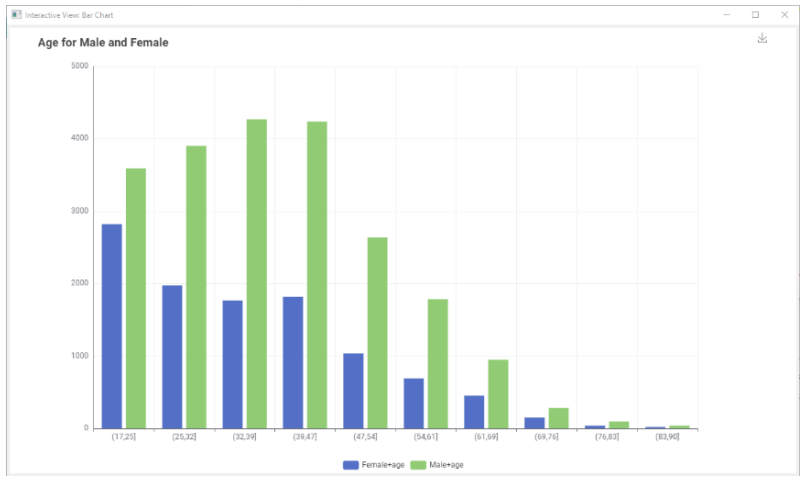



*Figure 3.55. Age Histogram for men and women from a Bar Chart node.*

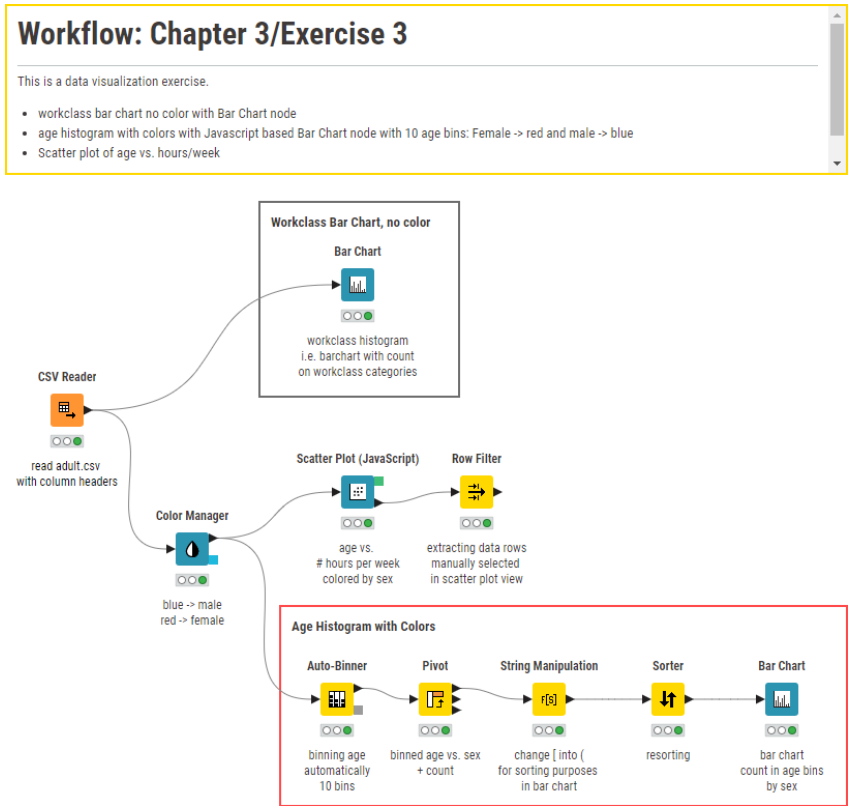



*Figure 3.56. Exercise 3: Solution Workflow.*

# 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.

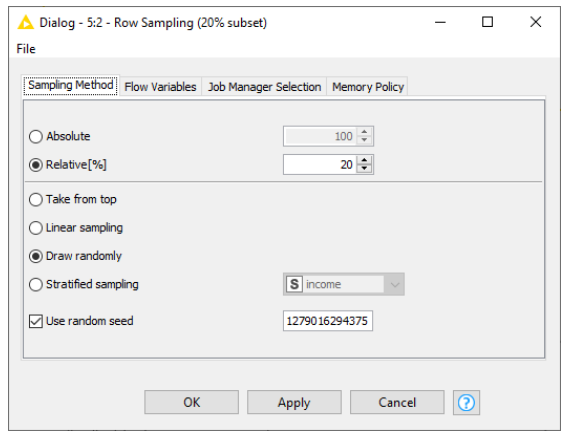



*Figure 4.1. Configuration window for the Row Sampling node.*

***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

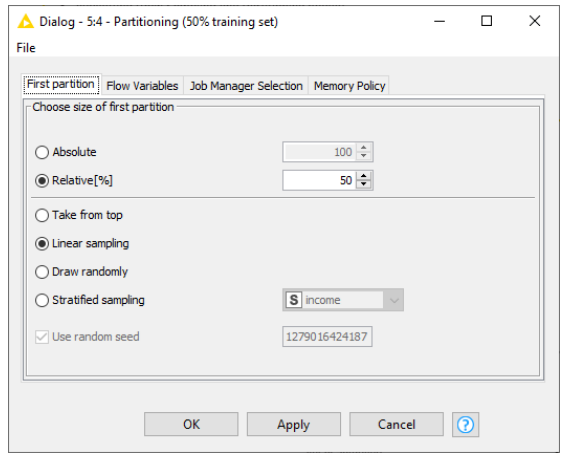



*Figure 4.2. Configuration window of the Partitioning node.*

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).

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.

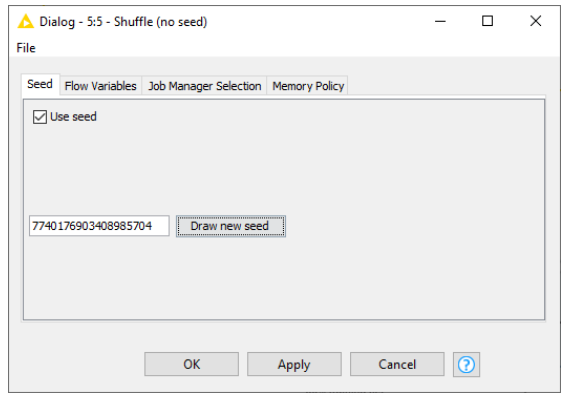



*Figure 4.3. Configuration window of the Shuffle node.*

## 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.

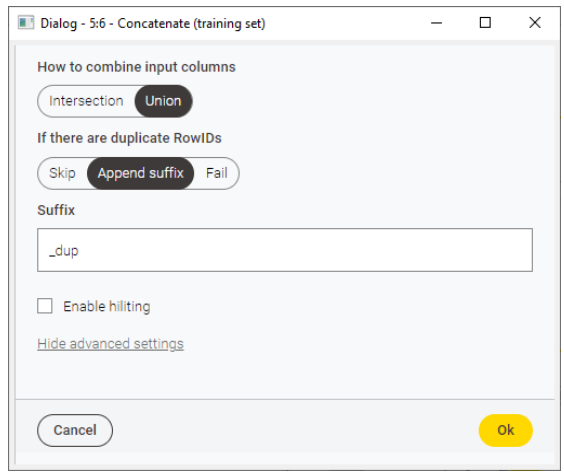



*Figure 4.4. Configuration window for the Concatenate node*

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 | Tinkerbell | 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 | Tinkerbell | 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.

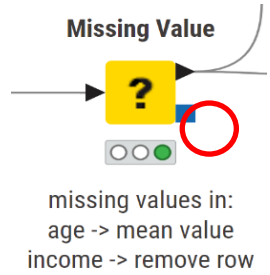



*Figure 4.5. The Missing Value node.*

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

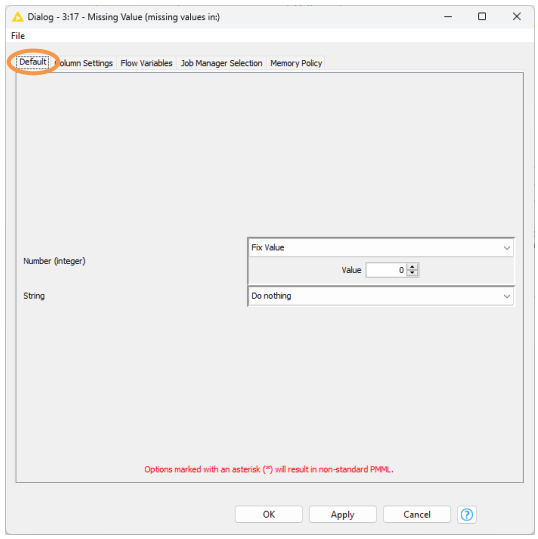



*Figure 4.6. Configuration window for the Missing Value node: the "Default" tab.*

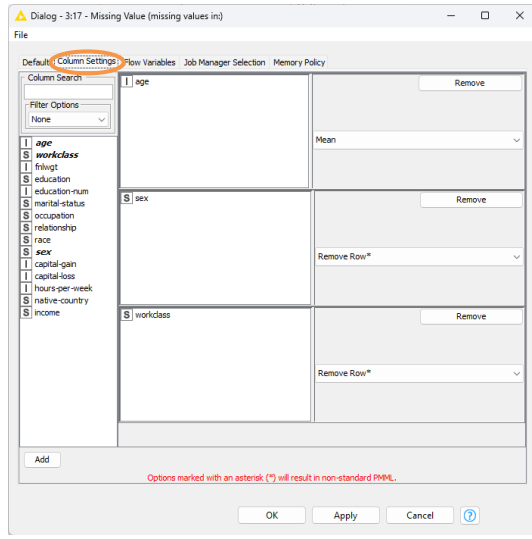



*Figure 4.7. Configuration window for the Missing Value node: the "Column Settings" tab.*

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)

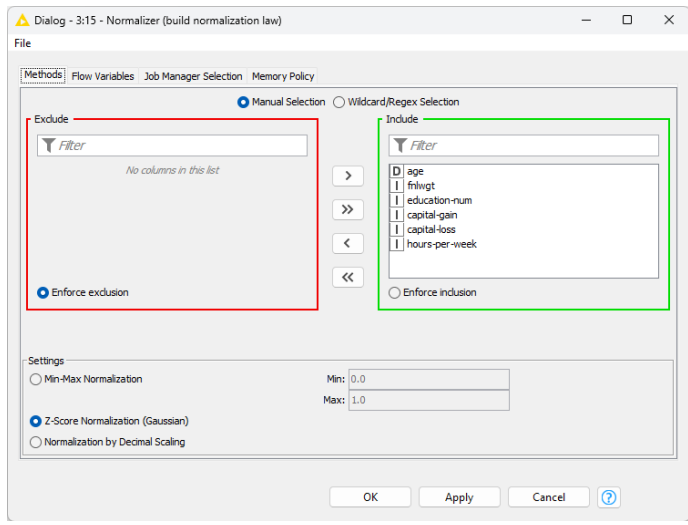



*Figure 4.8. Configuration window of the Normalizer node.*

***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.

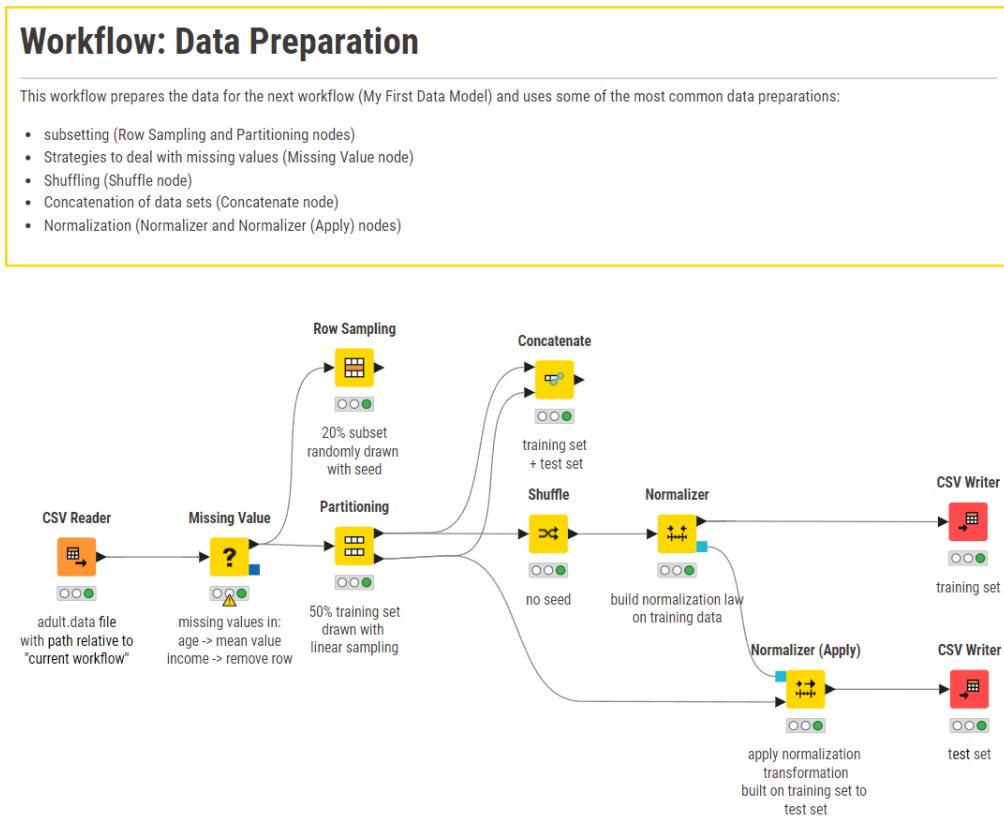



*Figure 4.9. The "Data Preparation" workflow.*

## 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*

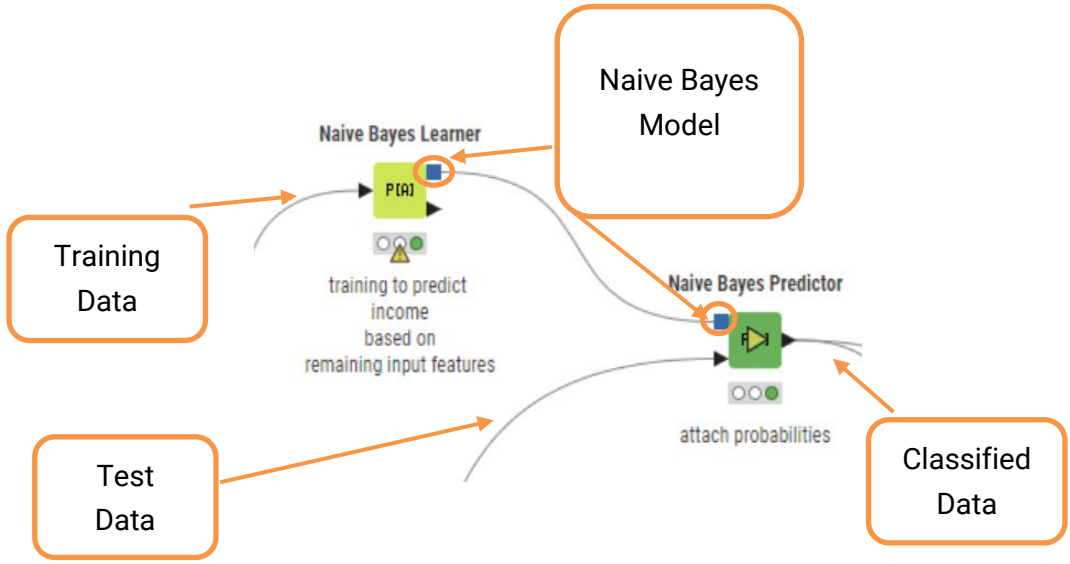



*Figure 4.10. Learner and Predictor nodes.*

## 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

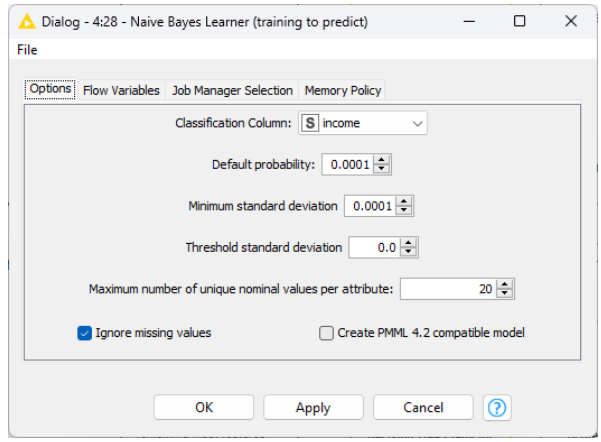



*Figure 4.11. Configuration window for the Naïve Bayes Learner node.*

## 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

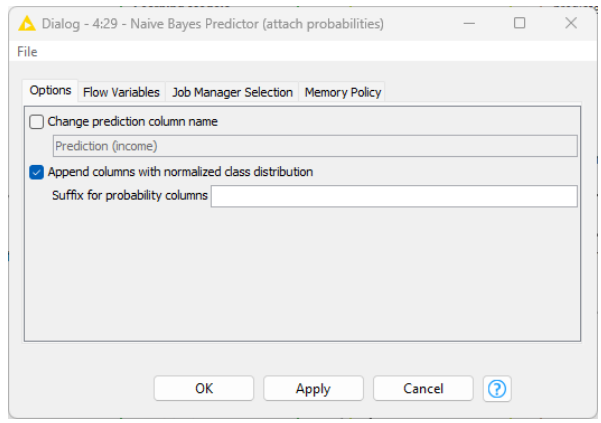



*Figure 4.12. Configuration window for the Naïve Bayes Predictor node.*

***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.

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.

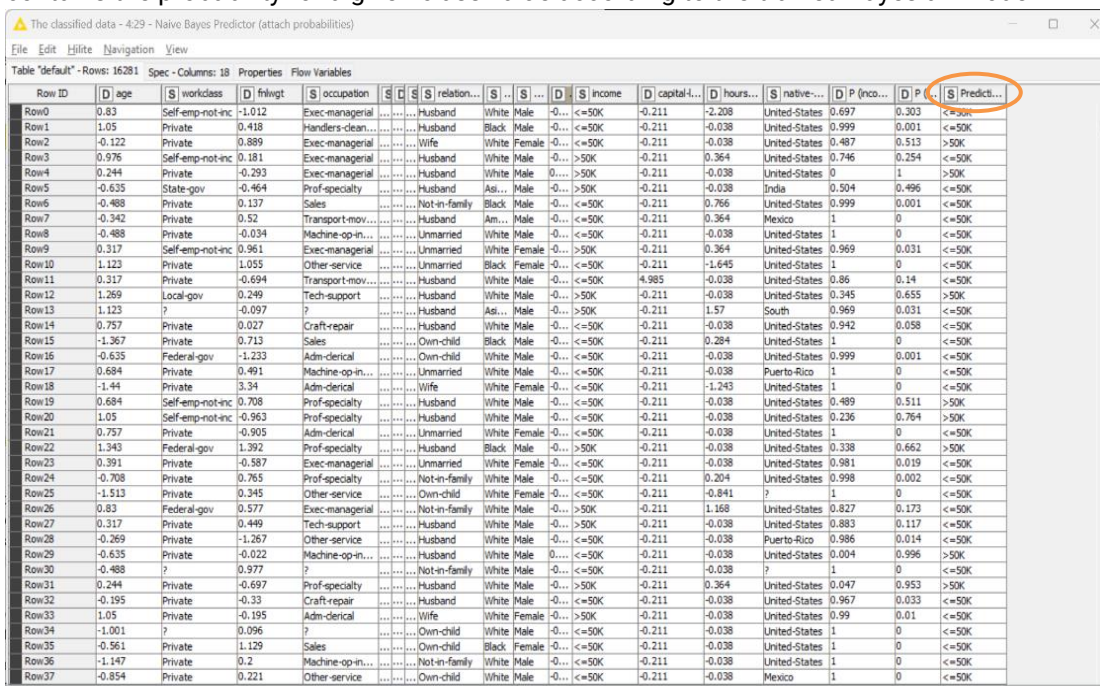



*Figure 4.13. Bayes Model's classified data.*

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”.

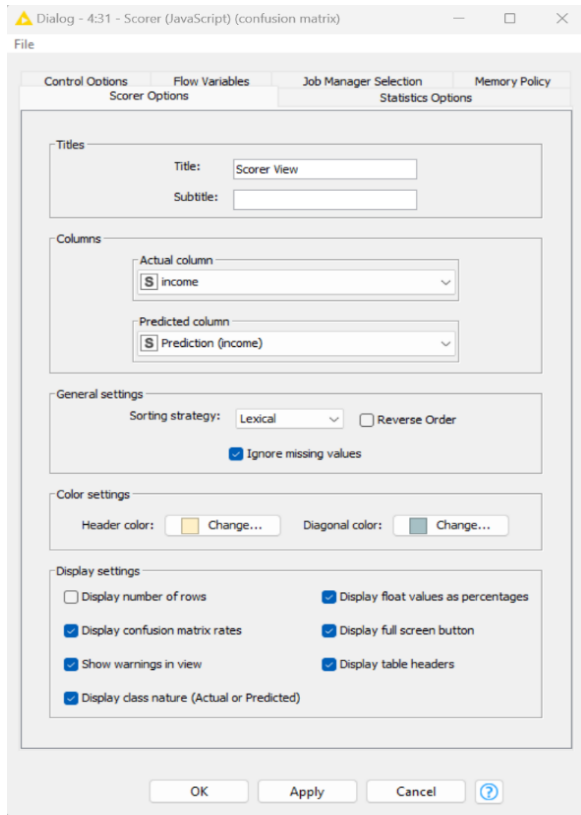



*Figure 4.14. Configuration window of the Scorer (JavaScript) node.*

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”.

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

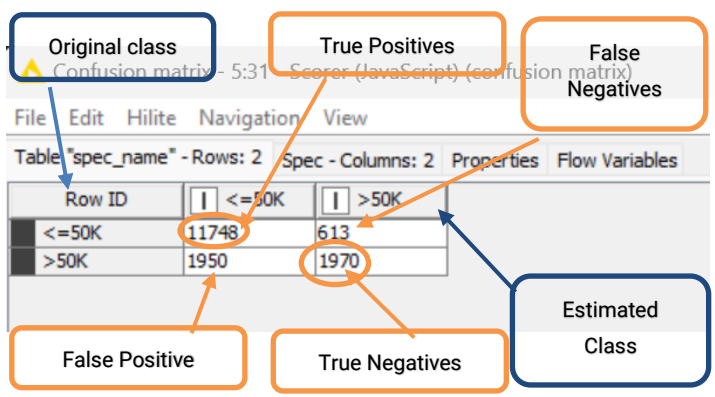



*Figure 4.15. True Positives, False Negatives, True Negatives, and False Positives in the Confusion Matrix for "<=50K" as the positive class.*

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 port 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.

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

"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).

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

"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.

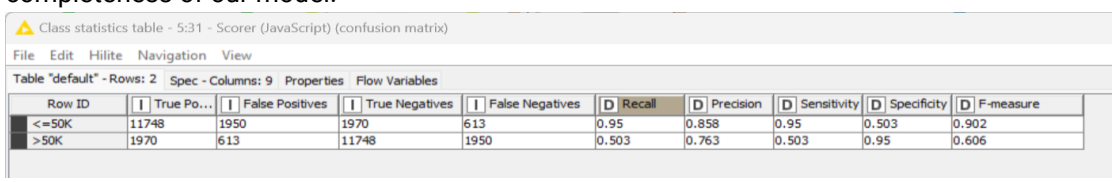
- $Accuracy = \frac{(TP+TN)}{(TP+FP+FN+TN)}$

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.


Class statistics table - 5:31 - Scorer (JavaScript) (confusion matrix)

File Edit Hilite Navigation View

Table "default" - Rows: 2 Spec - Columns: 9 Properties Flow Variables

| 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”.

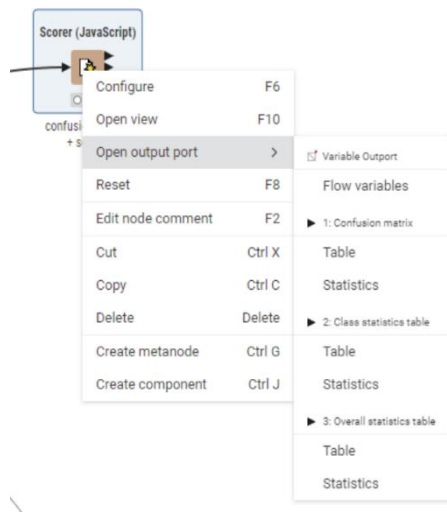



*Figure 4.17. The context menu of the Scorer (JavaScript) node.*

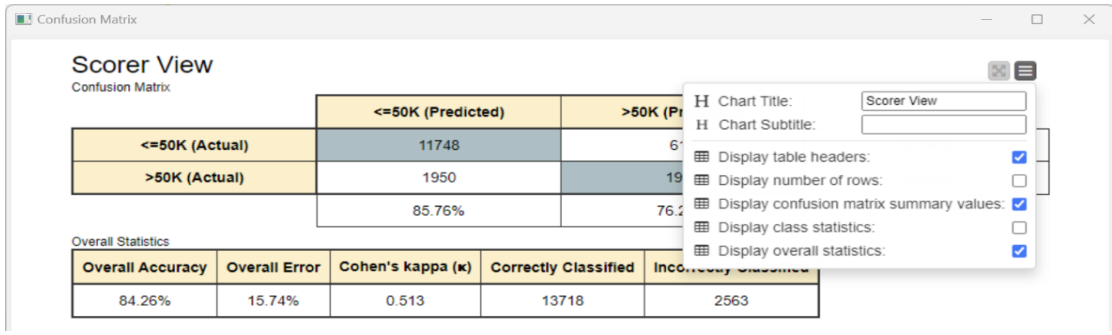



*Figure 4.18. Confusion Matrix from the Interactive View of the Scorer node.*

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

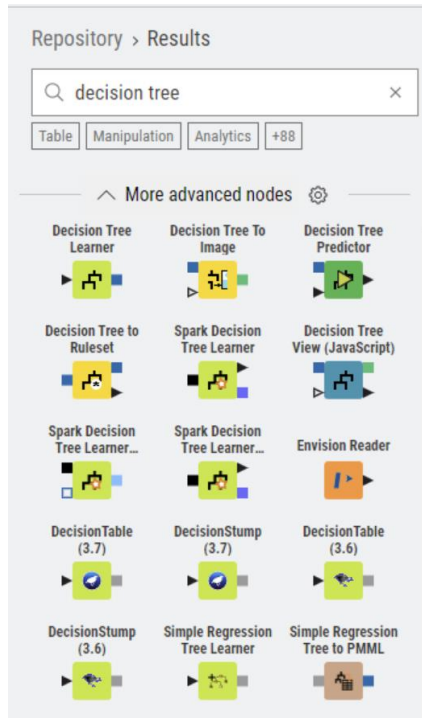



*Figure 4.19. Two nodes implement a Decision Tree: the Decision Tree Learner node and the Decision Tree Predictor node.*

[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).

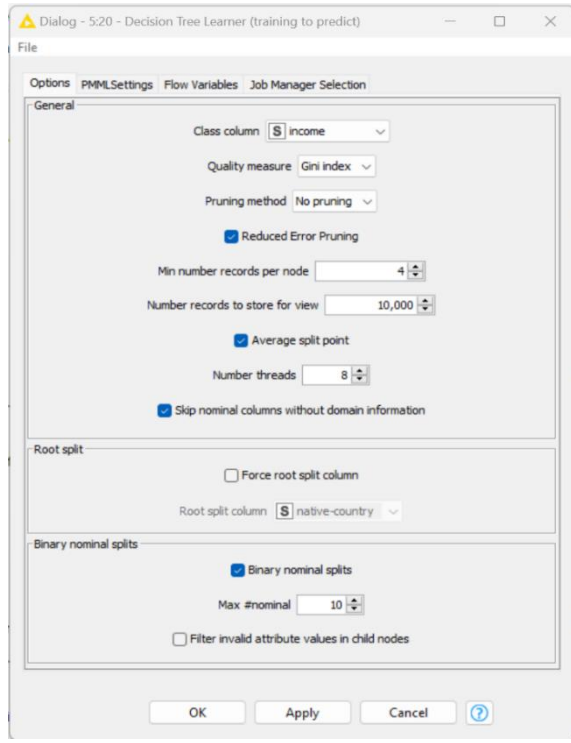



*Figure 4.20. Decision Tree Learner node: the "Options" tab.*

**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.

## 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.

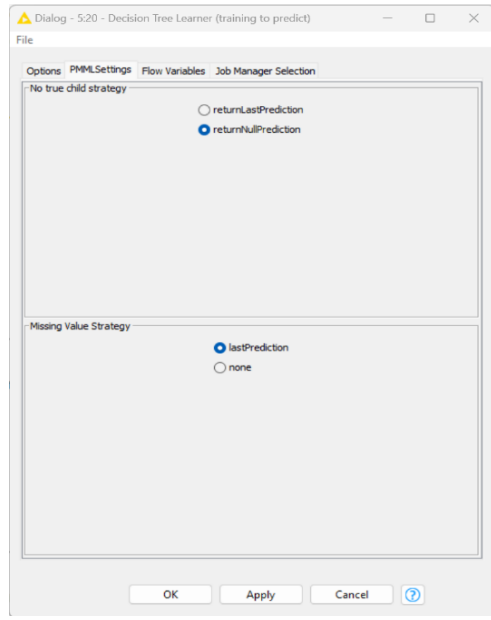



*Figure 4.21. Decision Tree Learner node: the "PMML Settings" tab.*

### 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)

### 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 hiliting (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

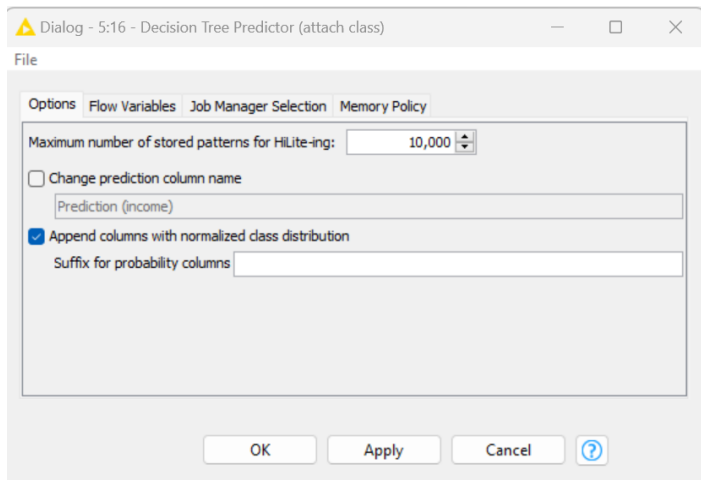



*Figure 4.22. Configuration window of the Decision Tree Predictor node.*

## 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.

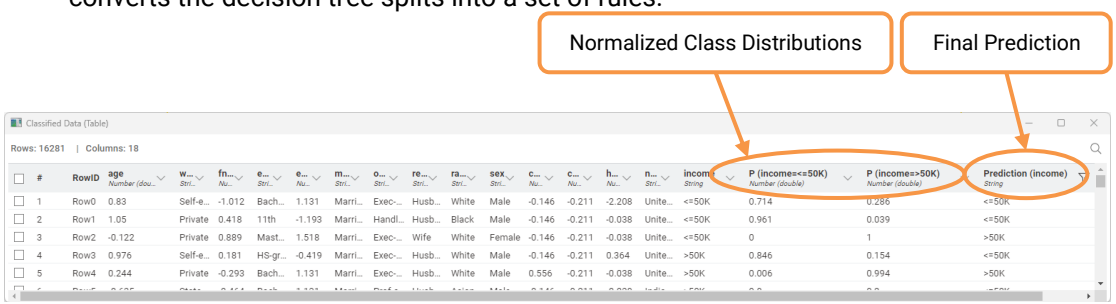



*Figure 4.23. Output data table from the Decision Tree Predictor node.*

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 shows 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.

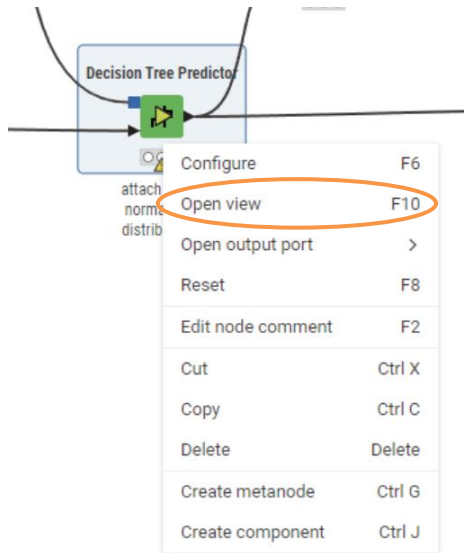



*Figure 4.24. Context Menu of the Decision Tree Predictor node.*

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 value[4]. 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

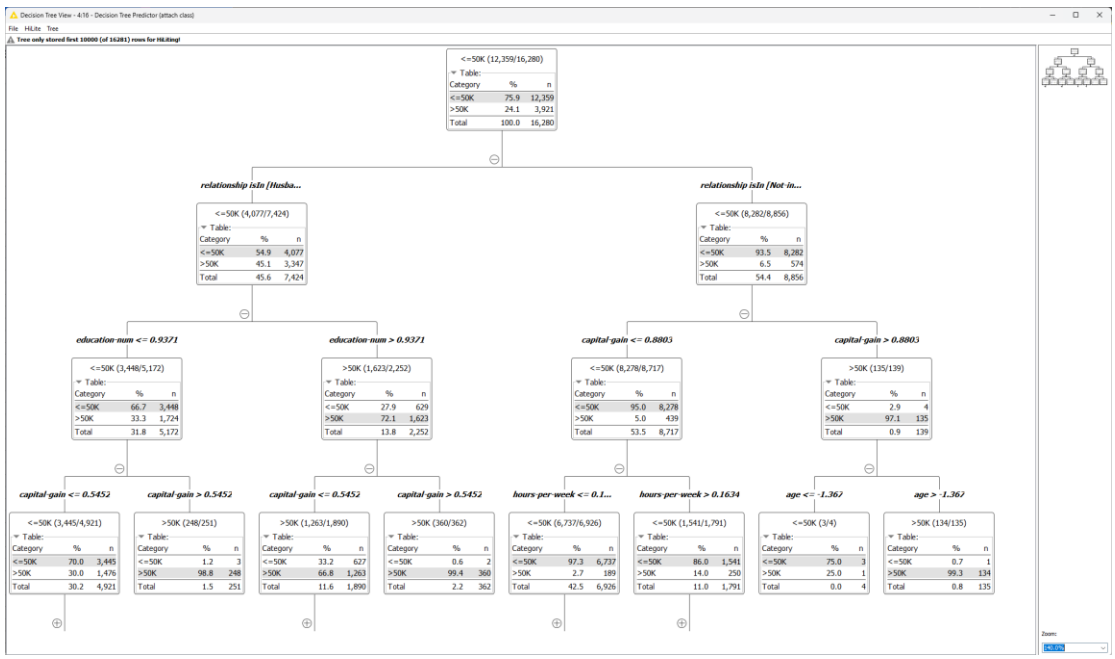



*Figure 4.25. View of the decision tree model created by the “Decision Tree Learner” node.*

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”.

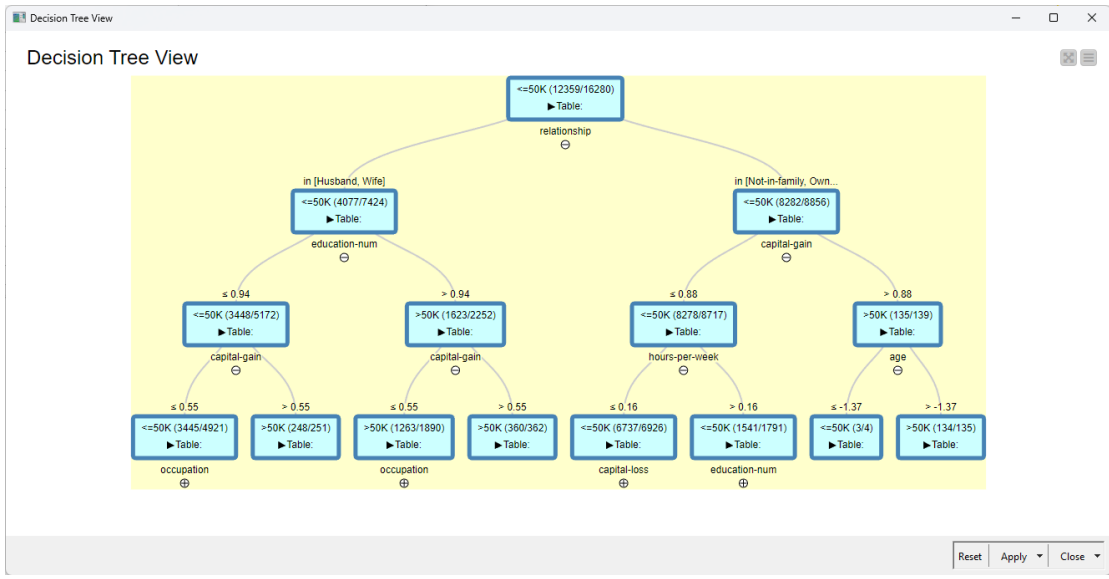



*Figure 4.26. View of the decision tree model as produced by the “Decision Tree View” node.*

“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.

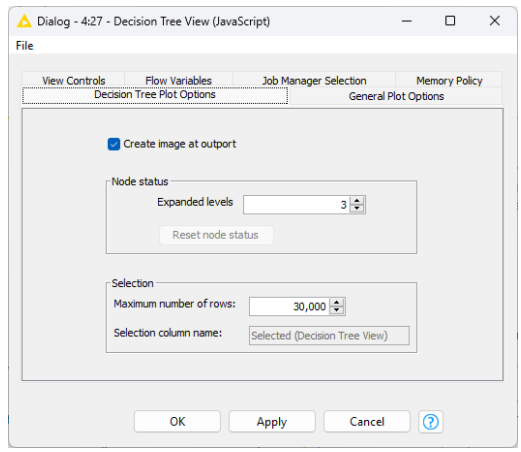



*Figure 4.27. Configuration window of the Decision Tree View node: the “Decision Tree Plot Options” tab.*

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

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.

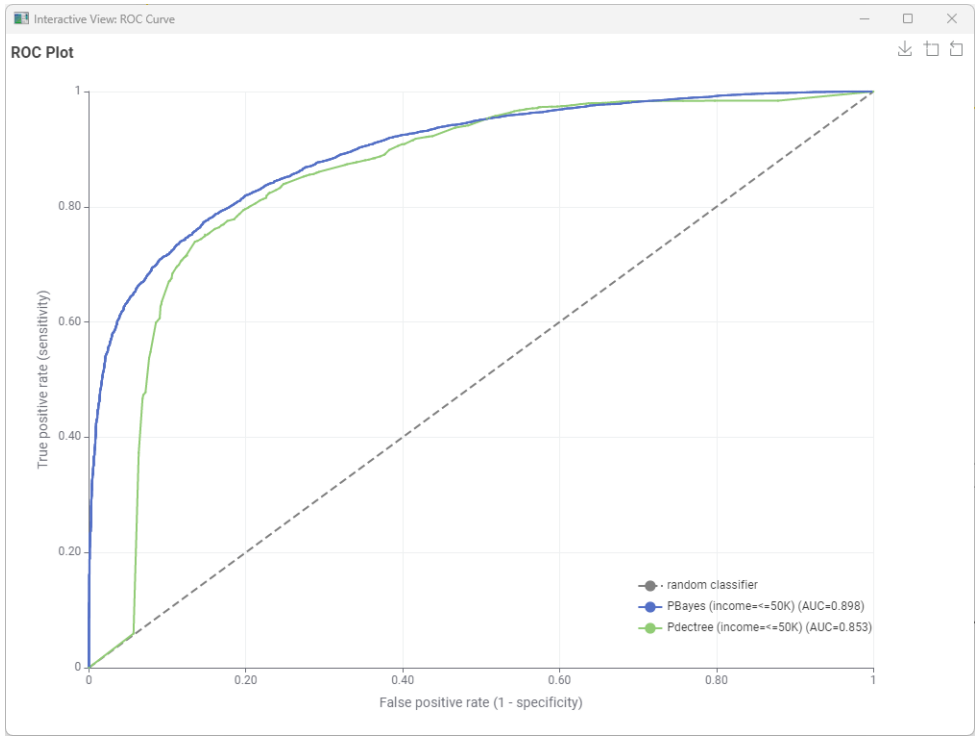



*Figure 4.28. View of the ROC Curve node.*

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

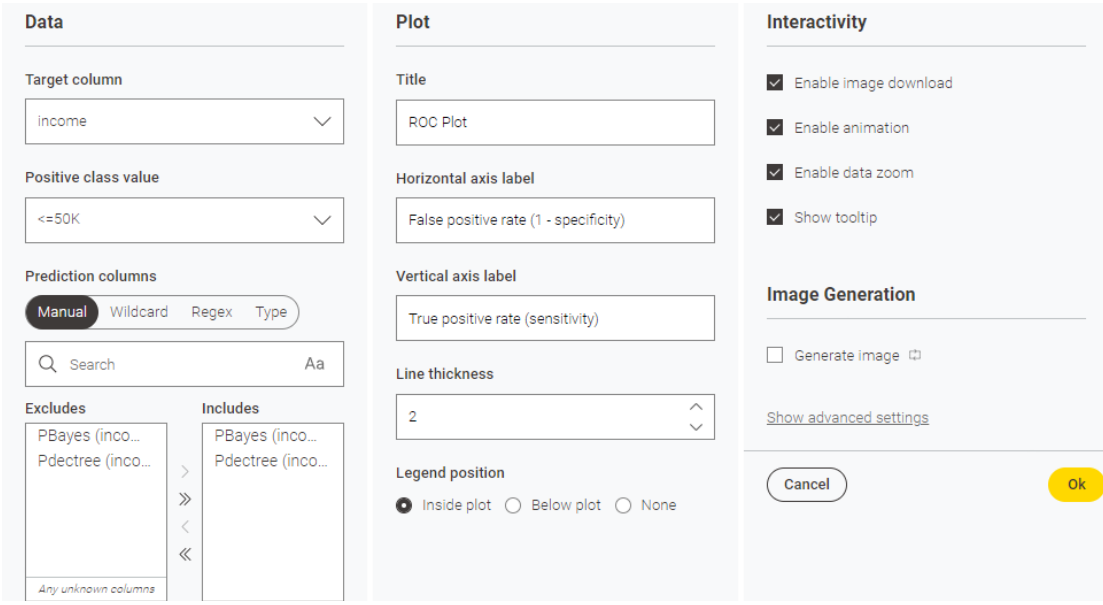



*Figure 4.29. Configuration window of the ROC Curve node.*

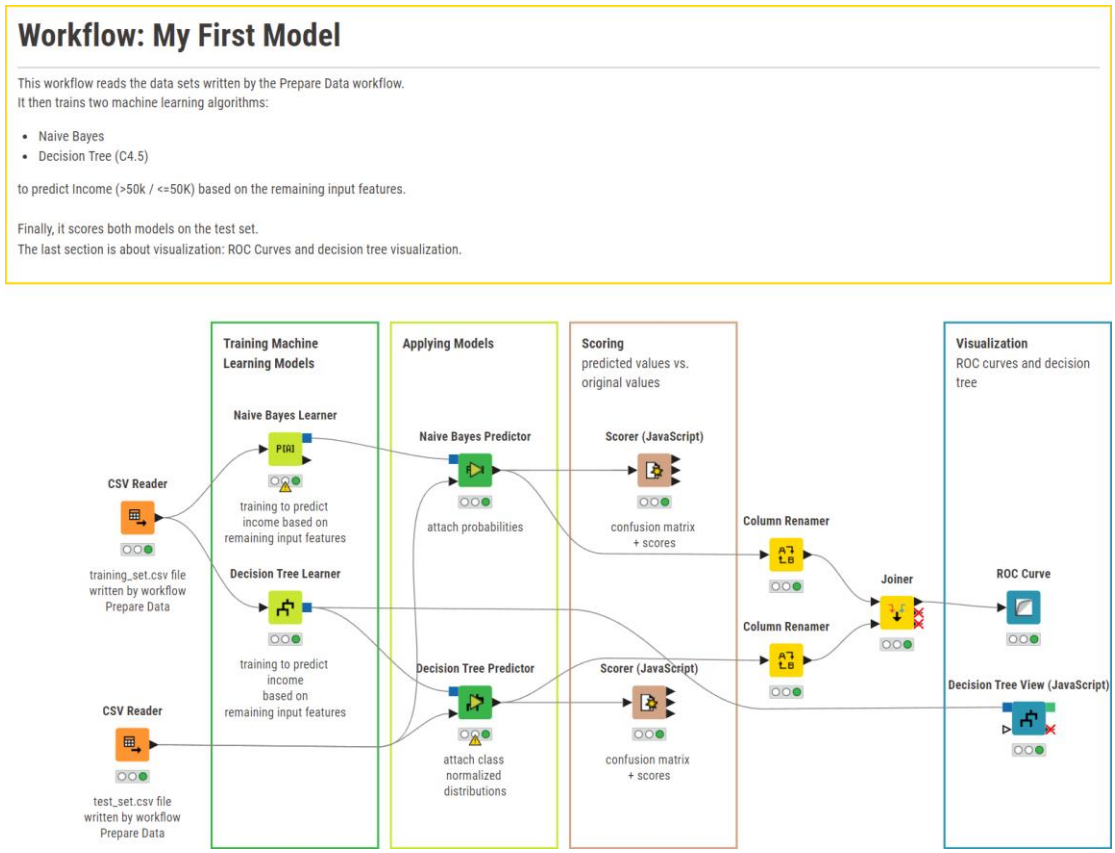



*Figure 4.30. Workflow "My First Model".*

## 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:

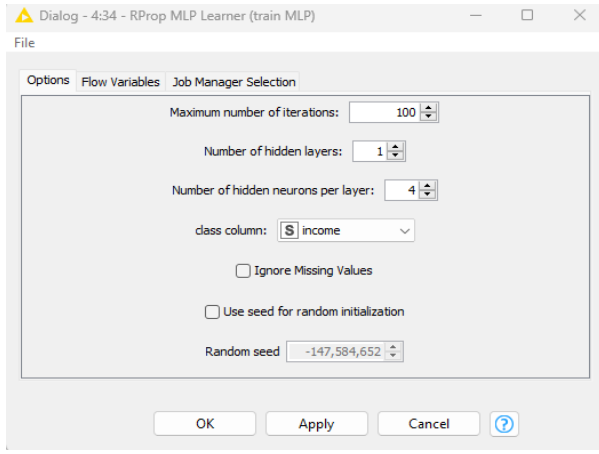



*Figure 4.31. Configuration window of the RProp MLP Learner node.*

- 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.

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.

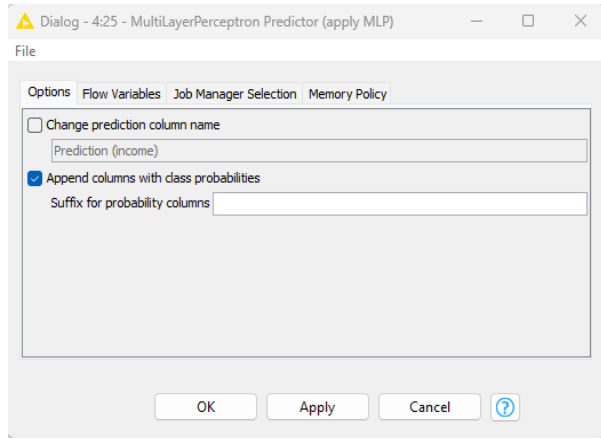



*Figure 4.32. Configuration window of the Multilayer Perceptron Predictor node.*

**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)".

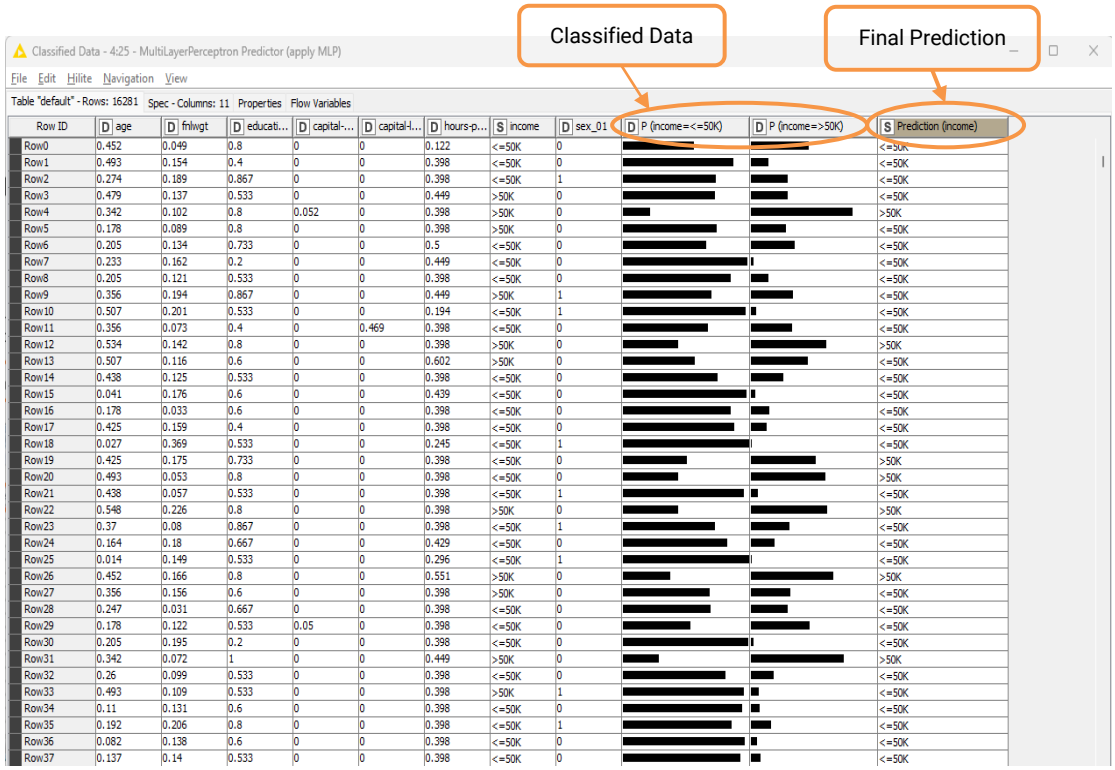
The firing value of the two output neurons is represented by a red-green bar instead of a double number. Red means a low number (< 0.5), green a high number (> 0.5). The highest firing output neuron decides the prediction class of the data row.

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

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




| Row ID | age | fnlwgt | educati... | capital-... | capital-l... | hours-p... | income | sex_01 | P (income=<=50K) | P (income=>50K) | 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:

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

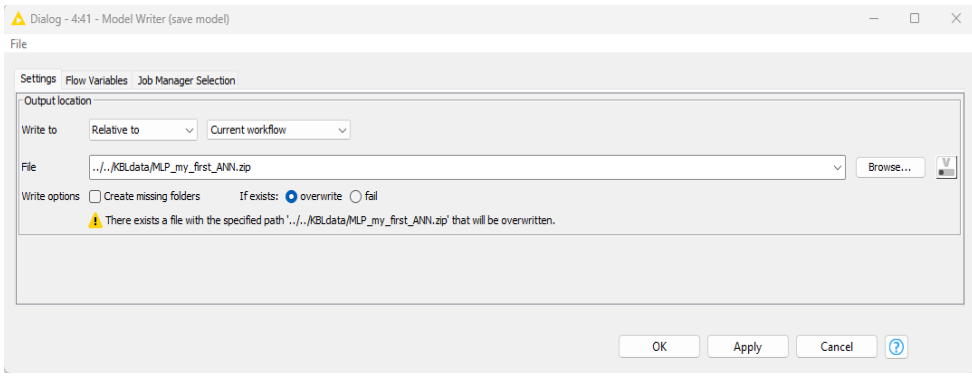



*Figure 4.34. Configuration window of the Model Writer node.*

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.

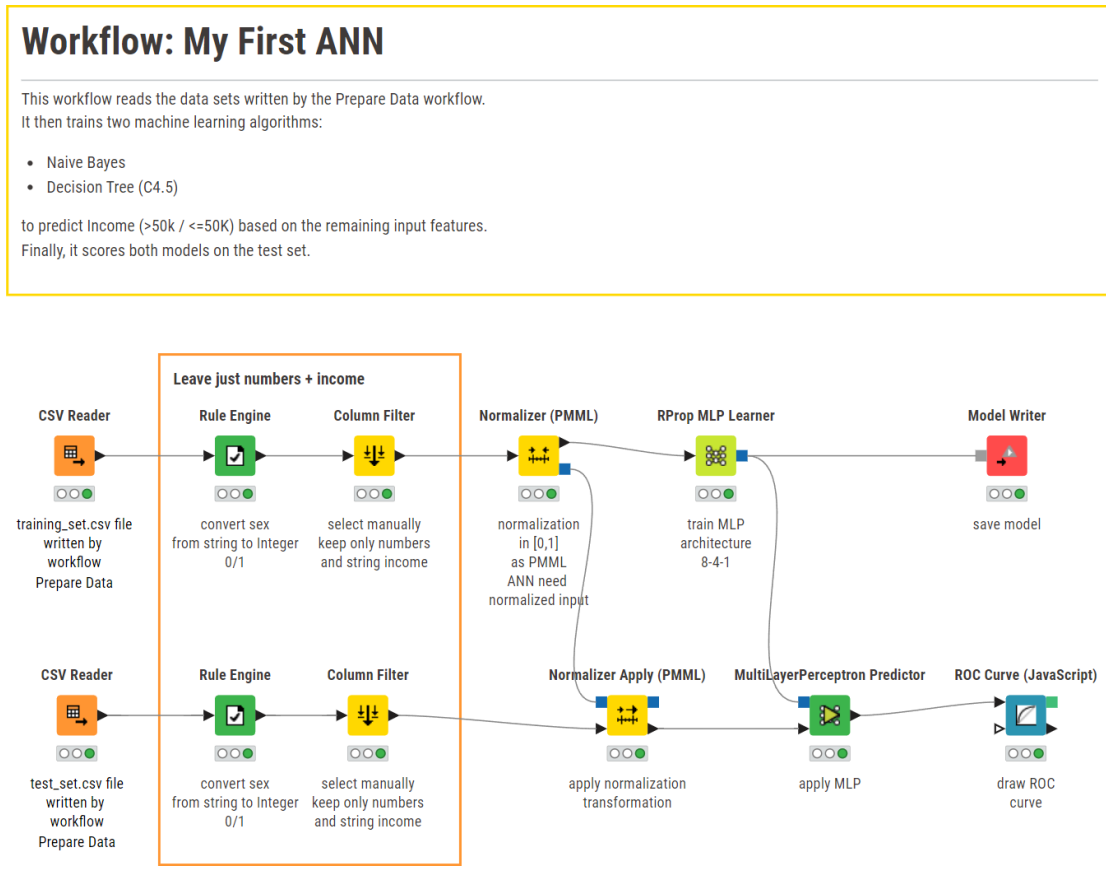



*Figure 4.35. Workflow "My First ANN".*

## 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.

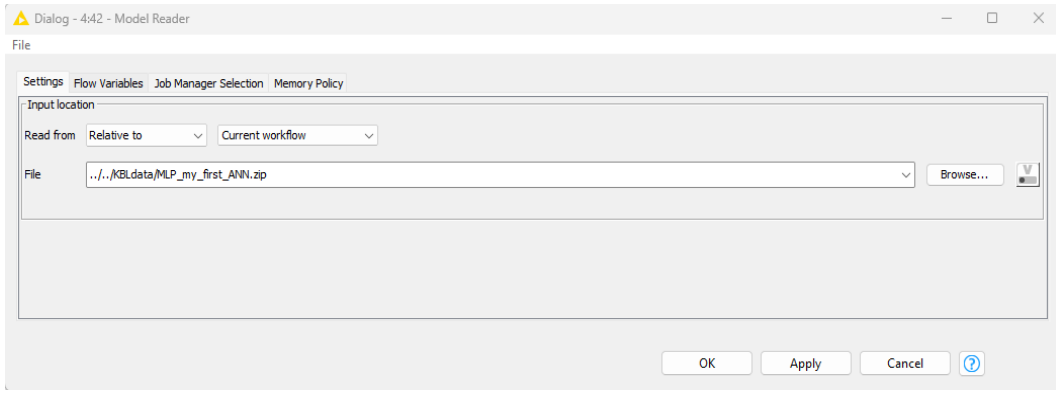



*Figure 4.36. Configuration window of the Model Reader node.*

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.

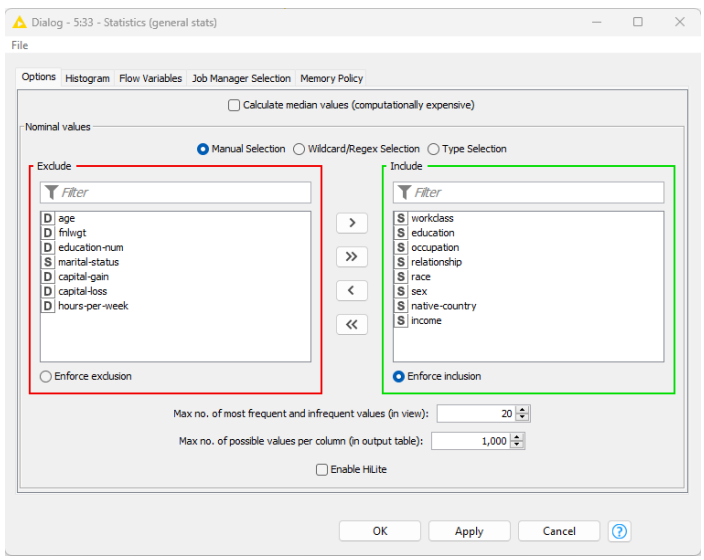



*Figure 4.37. Configuration window of the Statistics node.*

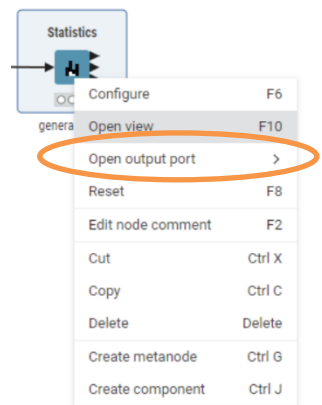



*Figure 4.38. Context menu of the Statistics node.*

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.

Statistics Table - 5:33 - Statistics (general stats)

File Edit Hilite Navigation View

Table "default" - Rows: 6 Spec - Columns: 16 Properties Flow Variables

| Row ID | Column | Min | Max | Mean | Std. de... | Variance | Skewness | Kurtosis | Overall ... | No. missings | No. NaNs | No. +oos | No. -oos | Median | Row count | Histogram |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| age | age | -1.587 | 3.759 | -0 | 1 | 1 | 0.557 | -0.163 | -0 | 0 | 0 | 0 | 0 | ? | 16280 | -2 4 |
| fnlwgt | fnlwgt | -1.683 | 11.947 | -0 | 1 | 1 | 1.39 | 5.506 | -0 | 0 | 0 | 0 | 0 | ? | 16280 | -2 12 |
| education-num | education-num | -3.517 | 2.293 | 0 | 1 | 1 | -0.332 | 0.656 | 0 | 0 | 0 | 0 | 0 | ? | 16280 | -4 2 |
| capital-gain | capital-gain | -0.146 | 13.424 | 0 | 1 | 1 | 11.968 | 155.278 | 0 | 0 | 0 | 0 | 0 | ? | 16280 | -0 13 |
| capital-loss | capital-loss | -0.211 | 10.873 | 0 | 1 | 1 | 4.749 | 22.075 | 0 | 0 | 0 | 0 | 0 | ? | 16280 | -0 11 |
| hours-per-week | hours-per-w... | -3.173 | 4.705 | 0 | 1 | 1 | 0.238 | 3.034 | 0 | 0 | 0 | 0 | 0 | ? | 16280 | -3 5 |

*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.

Occurrences Table - 5:33 - Statistics (general stats)

File Edit Hilite Navigation View

Table "default" - Rows: 41 Spec - Columns: 24 Properties Flow Variables

| 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-fish... |
| Row11 | ? | ? | ? | Doctorate | 212 | 0.013 | Tech-support |

*Figure 4.40. The "Occurrence Table" contains the number of occurences 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.

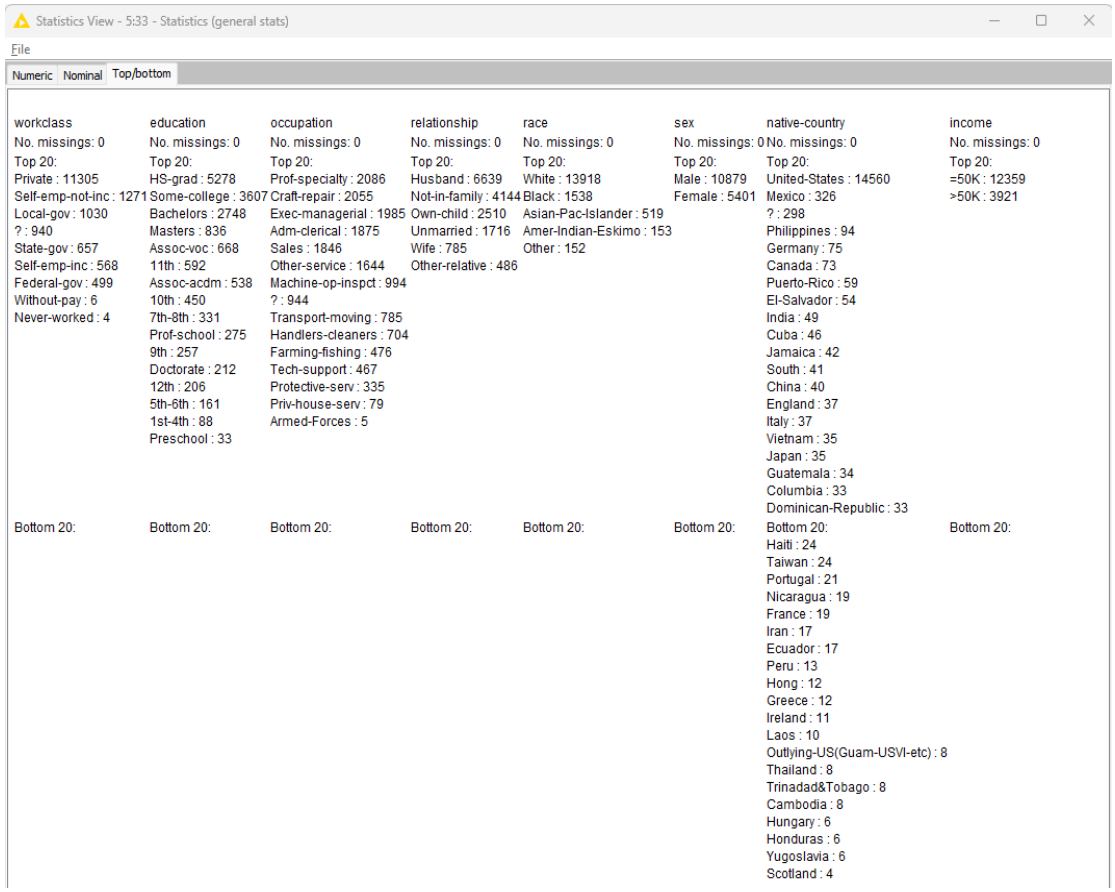



*Figure 4.41. "Statistics View" → Tab "Top/Bottom" with the number of occurrences of nominal values calculated only on the selected nominal columns and sorted in descending order.*

## 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-

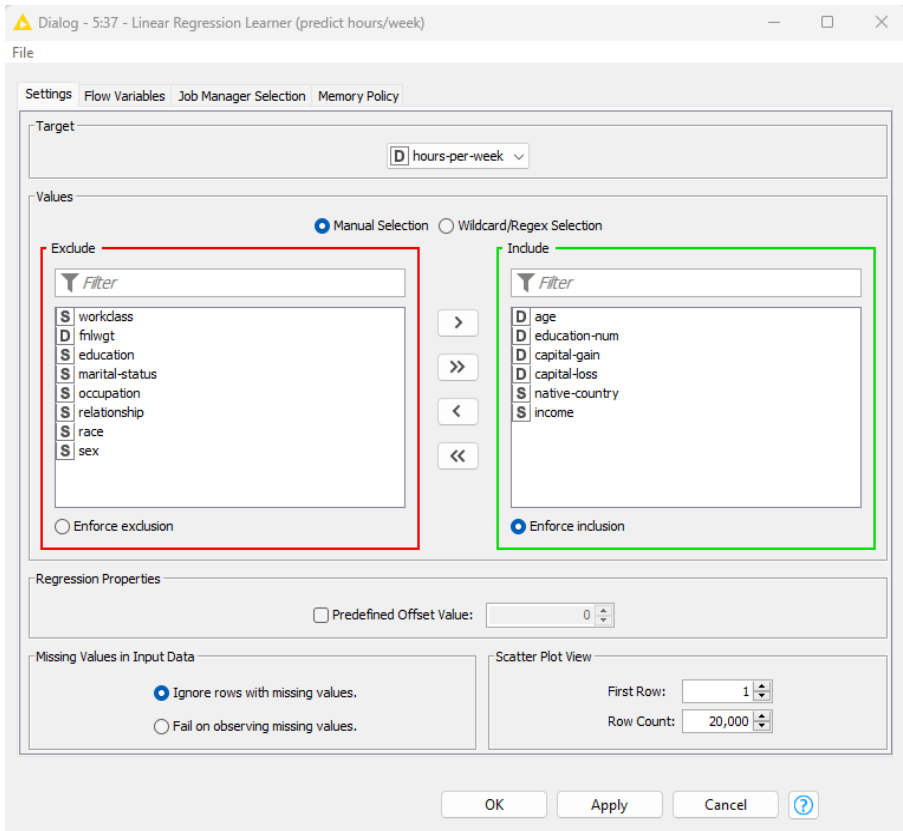



*Figure 4.42. Configuration window for the Linear Regression Learner node.*

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.

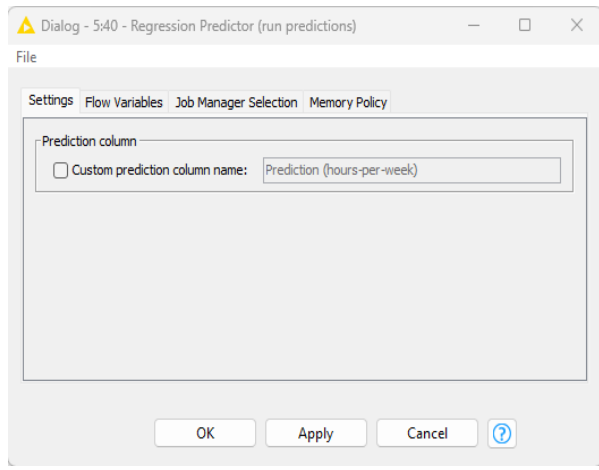



*Figure 4.43. Configuration window for the Regression Predictor node.*

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

## 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.

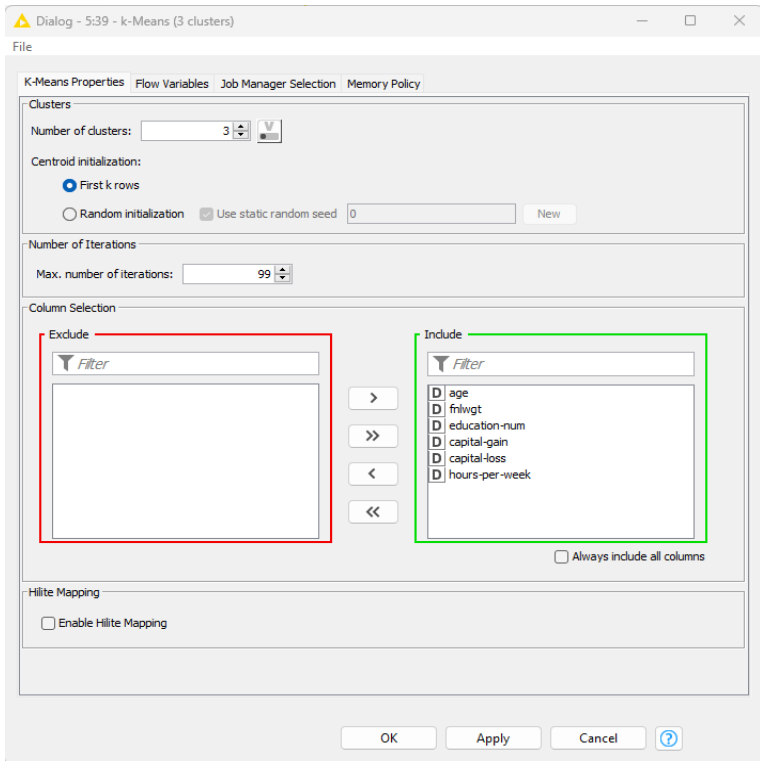



*Figure 4.44. Configuration window of the k-Means node.*

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.

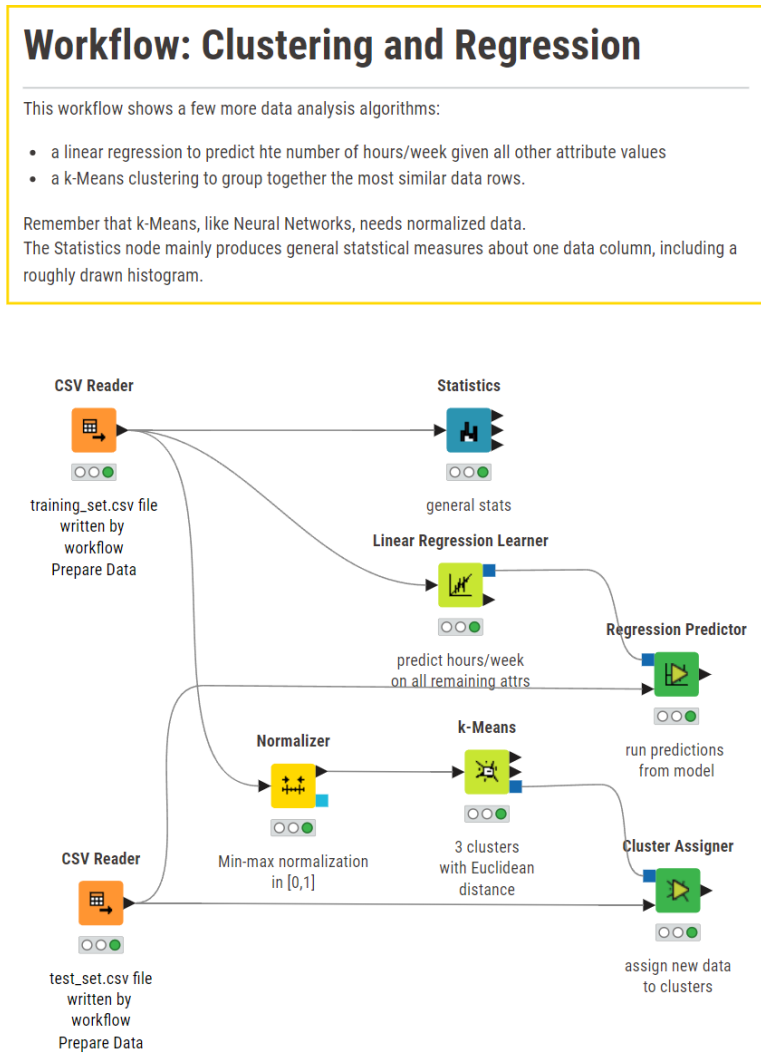



*Figure 4.45. Workflow "Clustering and Regression".*

***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.

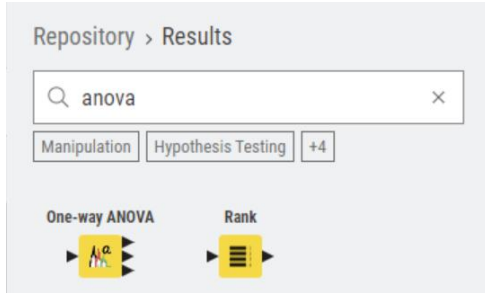



*Figure 4.46. The "Hypothesis Testing" sub-*

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"

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

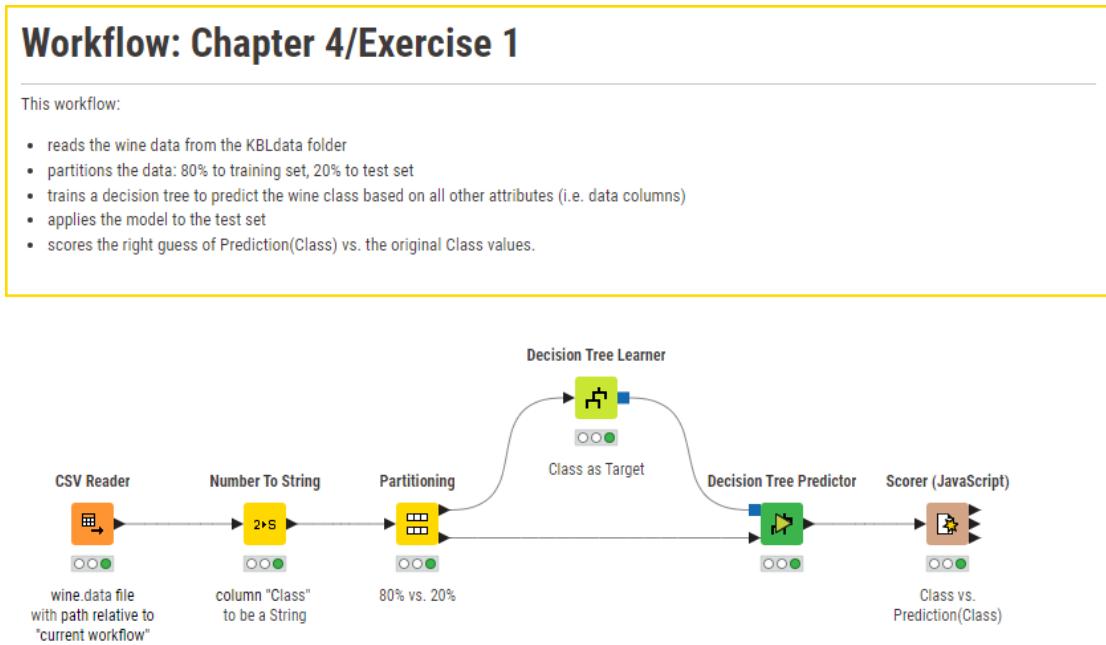



*Figure 4.47. Exercise 1: Workflow.*

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.

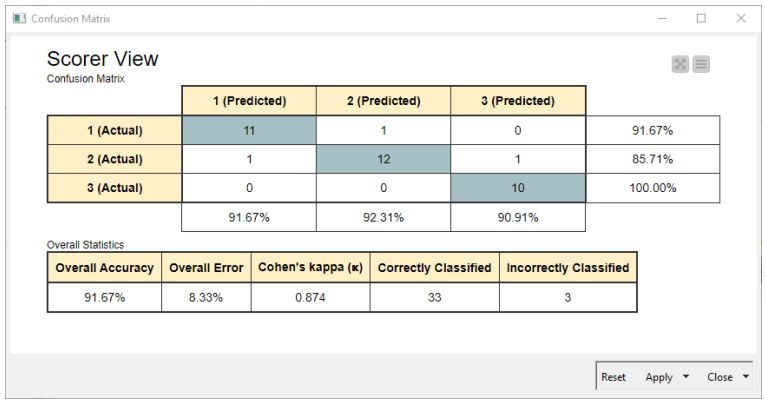



*Figure 4.48. Exercise 1: Confusion Matrix window of the Scorer (JavaScript) node.*

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".

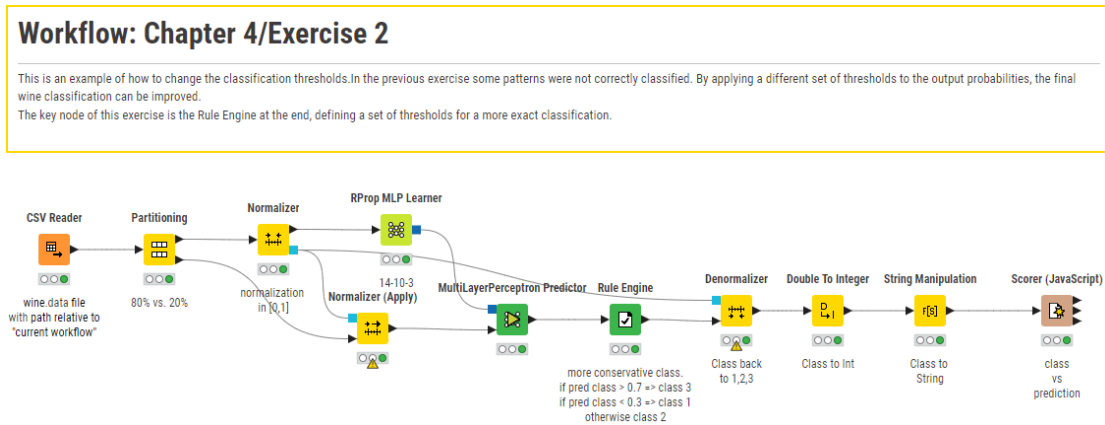



*Figure 4.49. Exercise 2: Workflow.*

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.

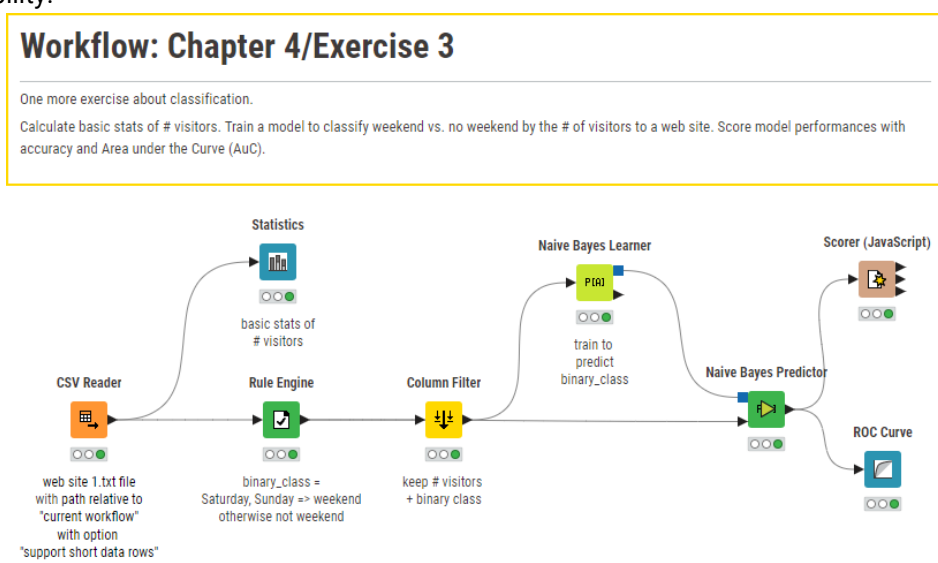



*Figure 4.50. Exercise 3: Workflow.*

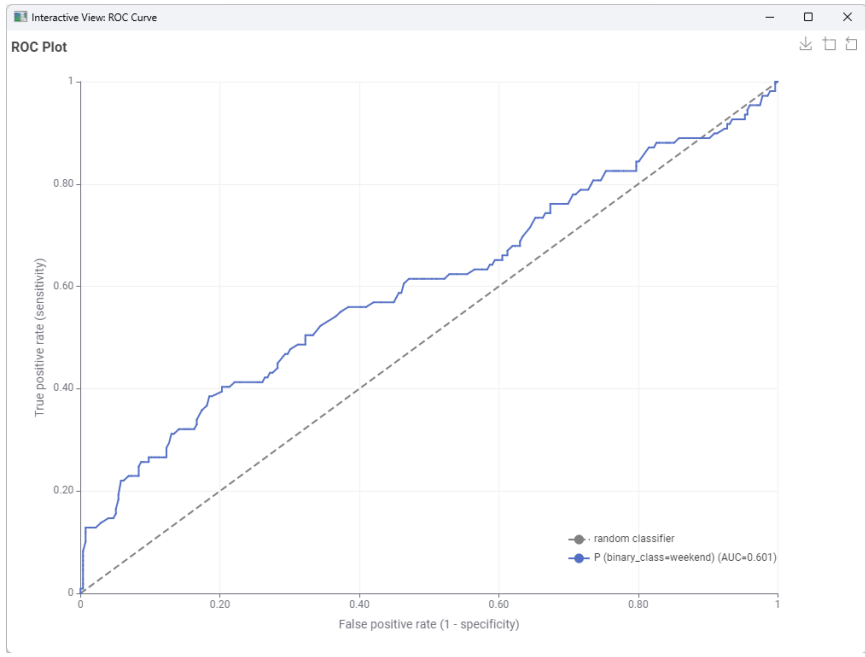



*Figure 4.51. Exercise 3: ROC Curve on the results of the Bayesian Classifier.*

# 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.

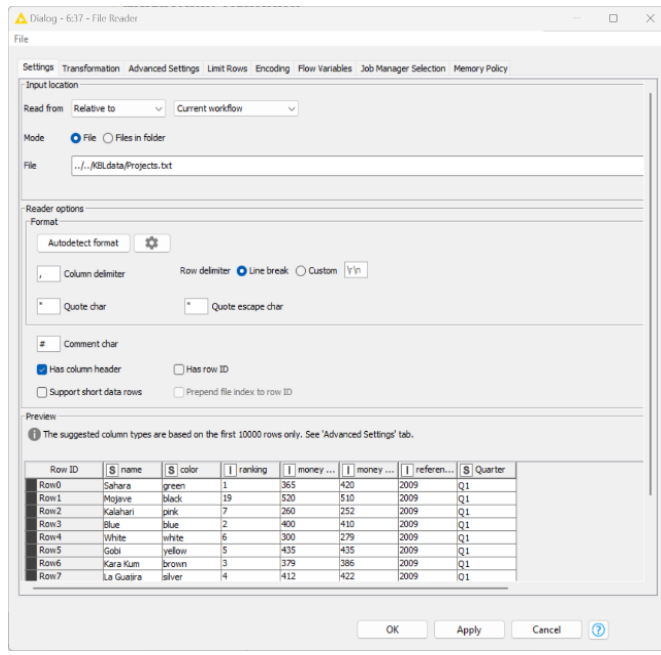



*Figure 5.1. Data Structure of the Projects.txt file.*

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.

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).*

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 Hilite Navigation View

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

| Row ID | S name | I referen... | I Sum(money assigned (1000)) | I 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.*

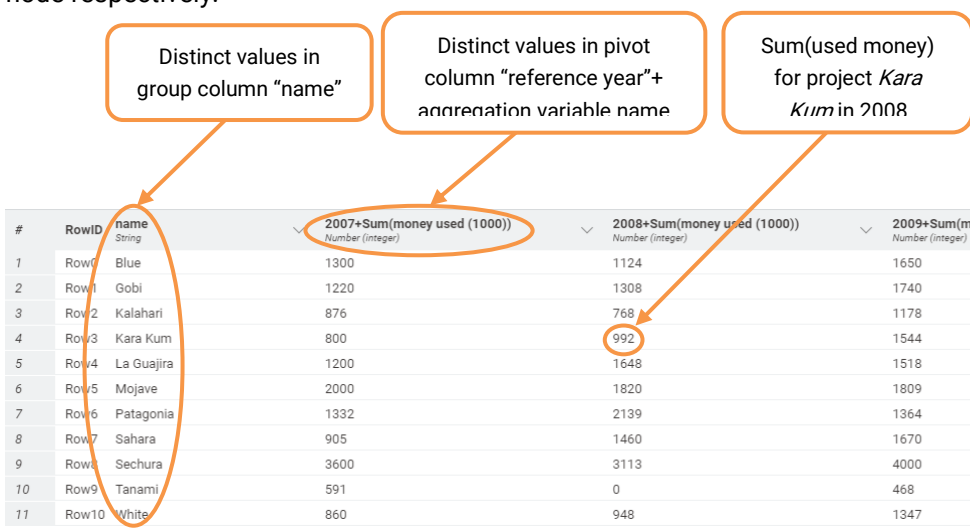
In one “Pivoting” node we calculated the sum (= aggregation method) of the values in column “money assigned(1000)” (= aggregation column) for each combination of values in the “reference year” (= pivot column) and “name” (= group column) columns.

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 (String) | 2007+Sum(money used (1000)) (Number (integer)) | 2008+Sum(money used (1000)) (Number (integer)) | 2009+Sum(money used (1000)) (Number (integer)) |
|---|---|---|---|---|---|
| 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.

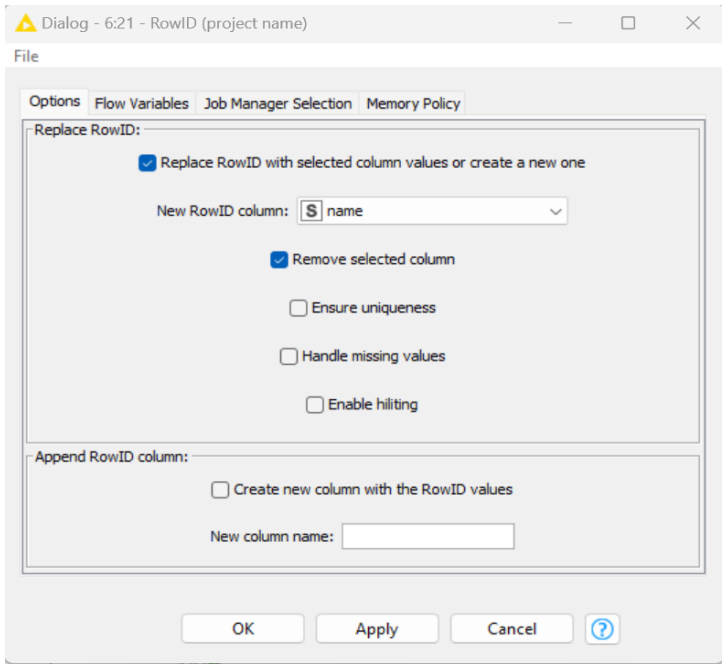



*Figure 5.4. Configuration window for the RowID node.*

- "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.

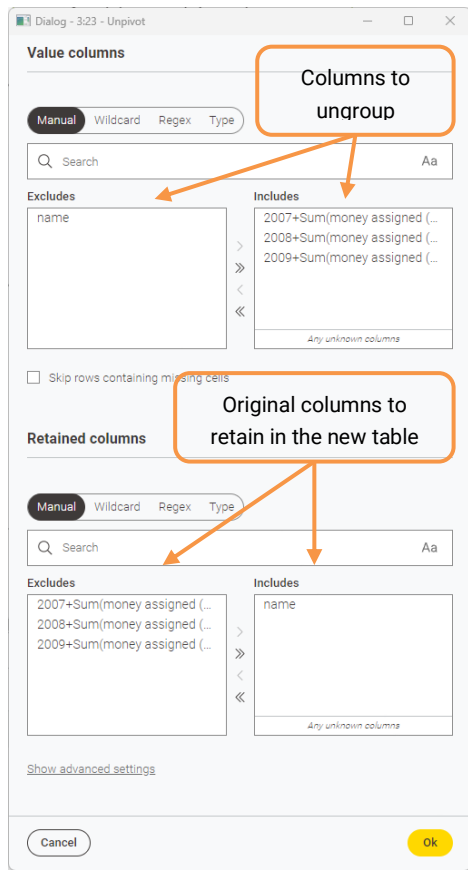



*Figure 5.5. Configuration window for the Unpivoting node.*

- “Excludes and Includes” keeps the included/excluded columns as fixed and adds possible new columns to the other column set.

**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

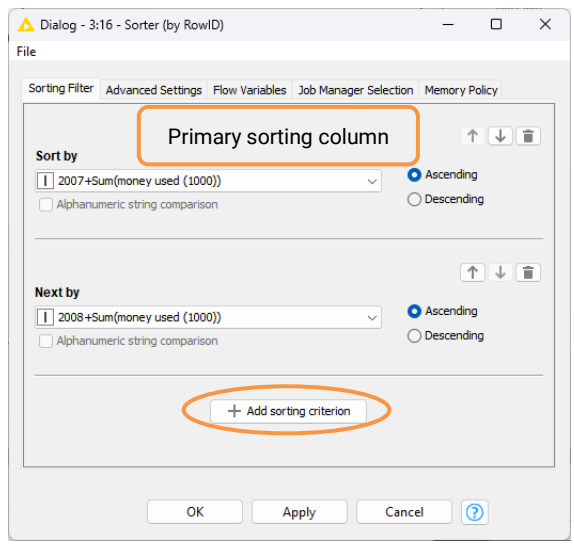



*Figure 5.6. Configuration window of the Sorter node.*

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.

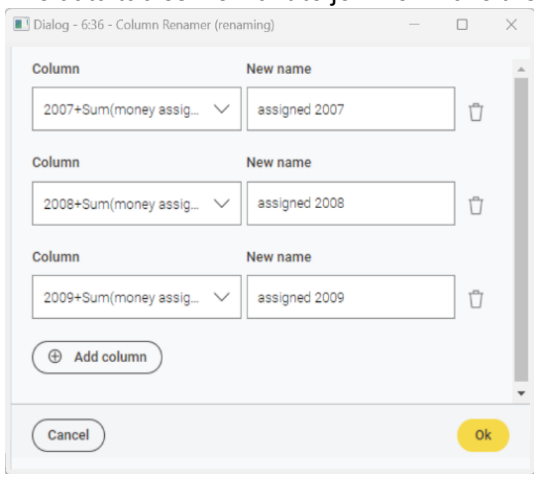



*Figure 5.7. Money assigned to each project each year.*

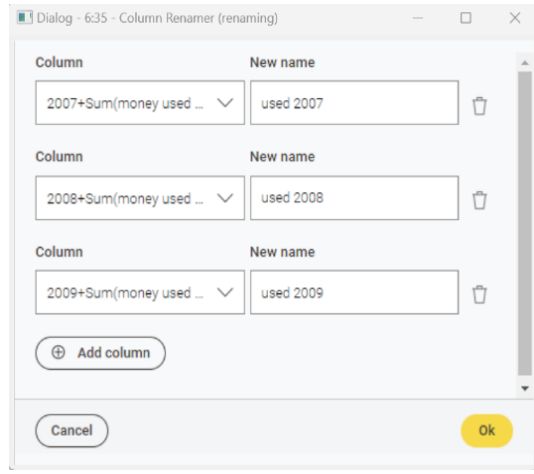



*Figure 5.8. Money used by each project each year.*

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.

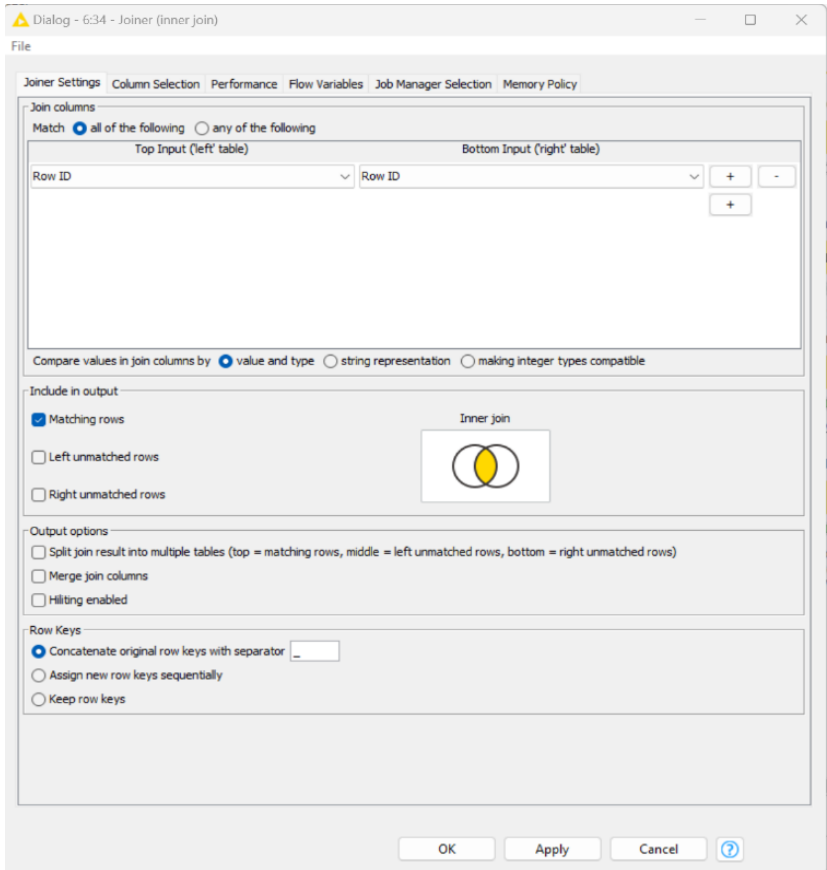



*Figure 5.9. Configuration window for the Joiner node which includes two tabs to be filled: "Joiner Settings" and "Column Selection".*

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.

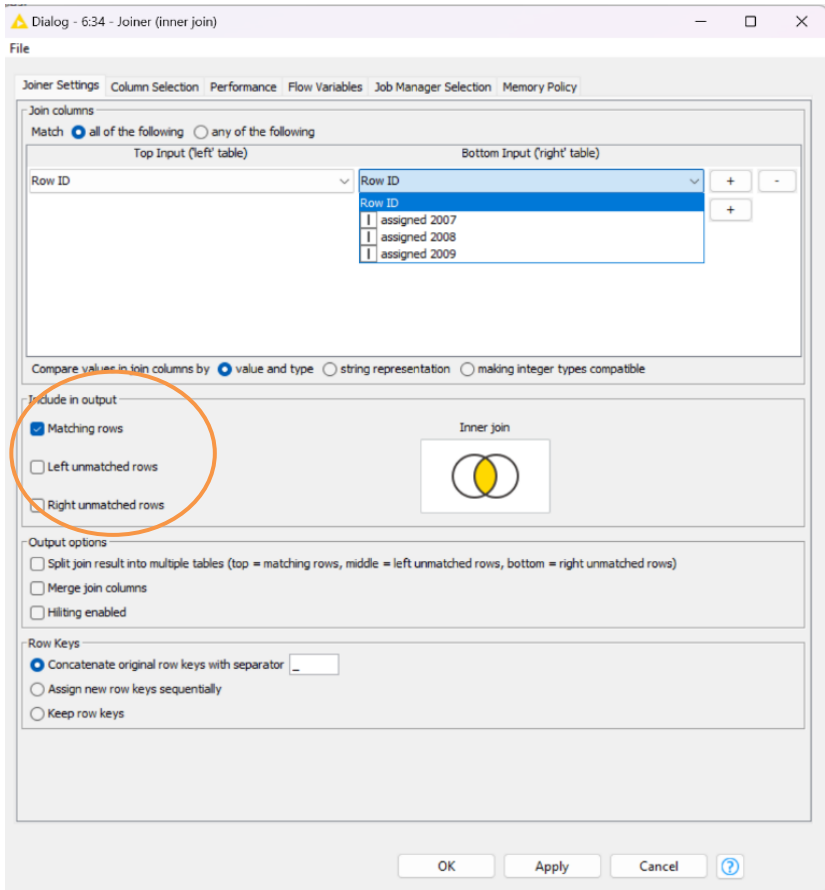



*Figure 5.10. Configuration window for the Joiner node: the "Joiner Settings" tab.*

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.

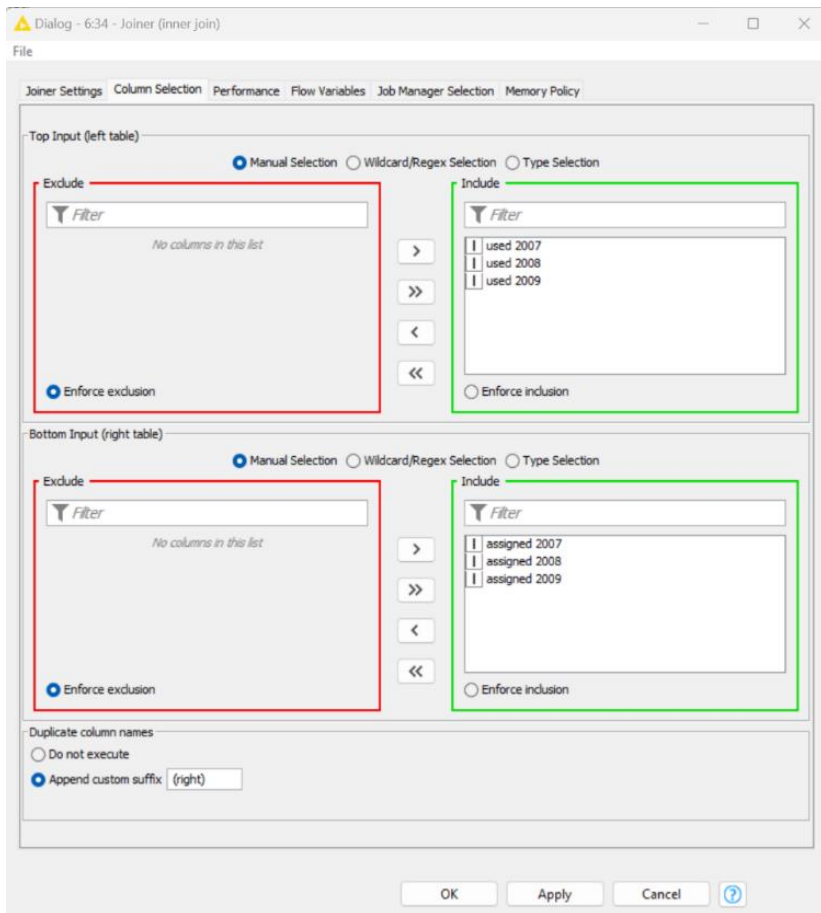



*Figure 5.11. Configuration window for the Joiner node: the "Column Selection" tab.*

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.

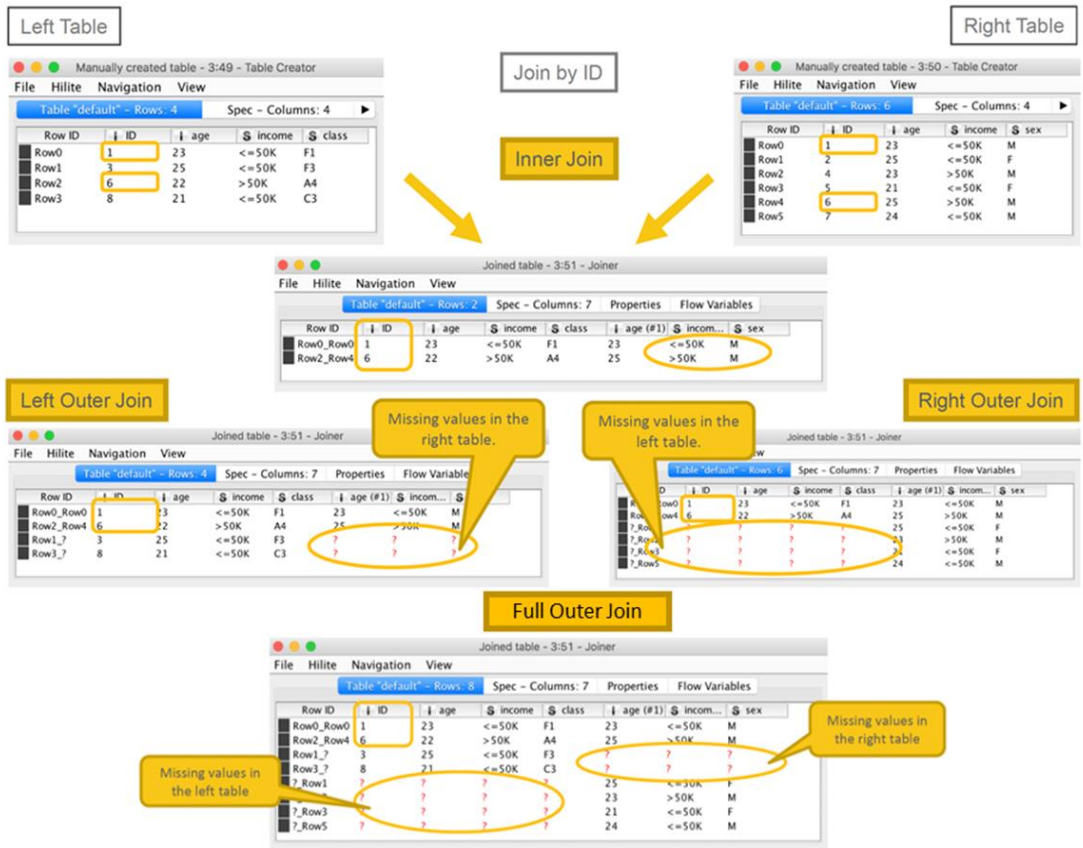



*Figure 5.12. Join modes with the "Filter duplicates" option enabled.*

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 pice a 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.

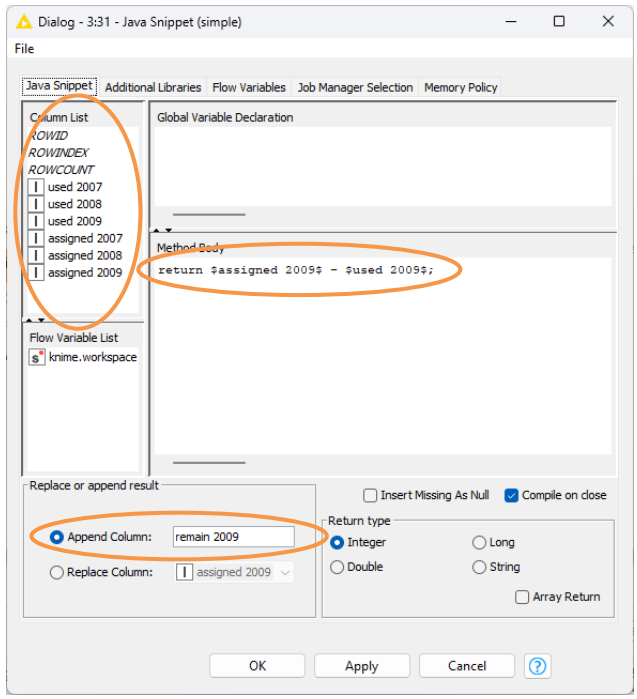



*Figure 5.14. Configuration window for the Java Snippet node.*

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

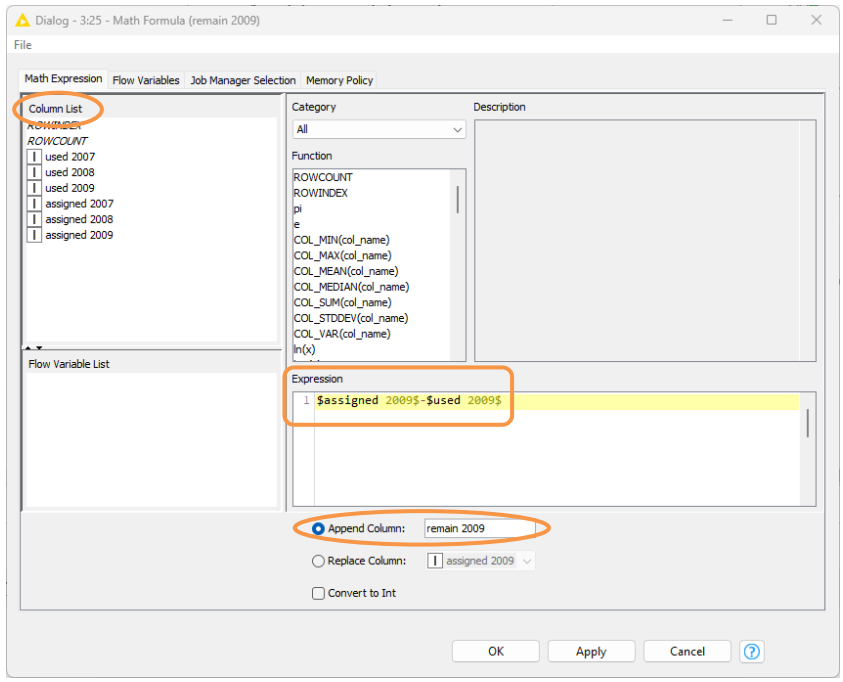



*Figure 5.15. Configuration window of the Math Formula node.*

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 <remain 2009> 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

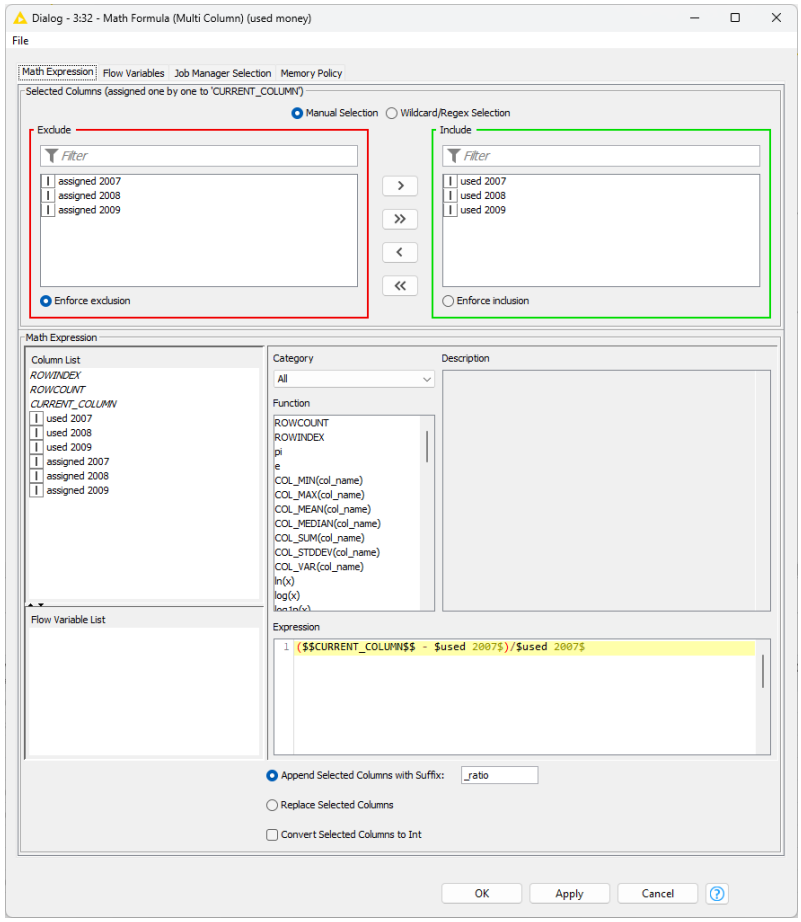



*Figure 5.16. Configuration window of the Math Formula (Multi Column) node.*

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.

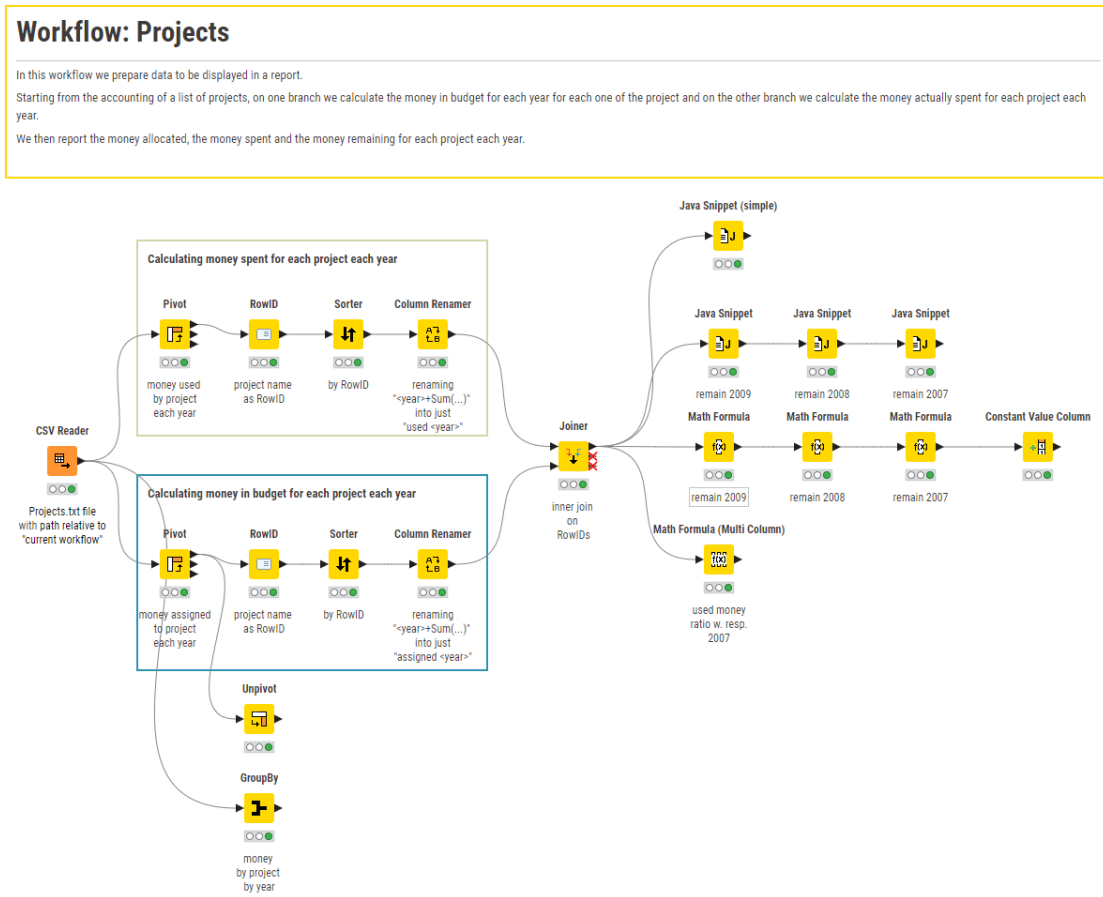



*Figure 5.17. Workflow "Projects".*

## 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.

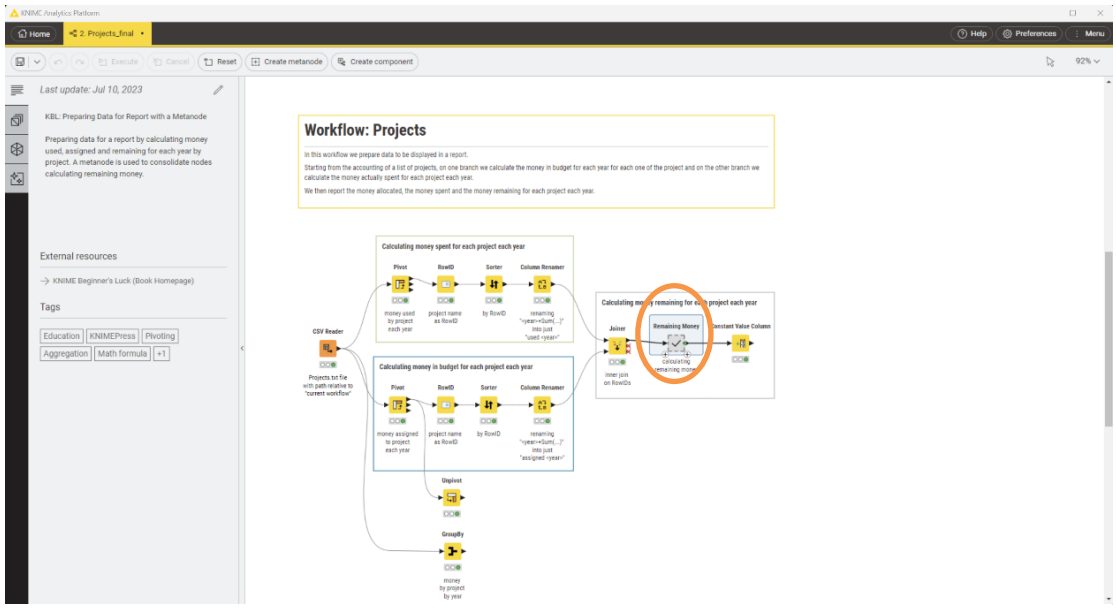



*Figure 5.19. Metanode.*

## Collapse Pre-existing Nodes into a Metanode

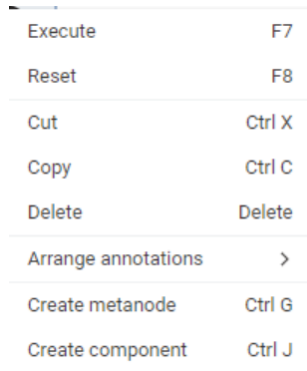



*Figure 5.18. "Create a Metanode..." option in the context menu of the selected nodes.*

- 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.

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

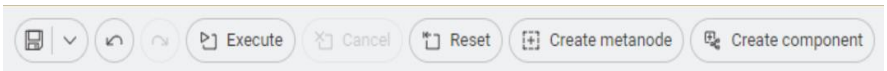



*Figure 5.20. Metanode icon in the tool bar.*

**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:

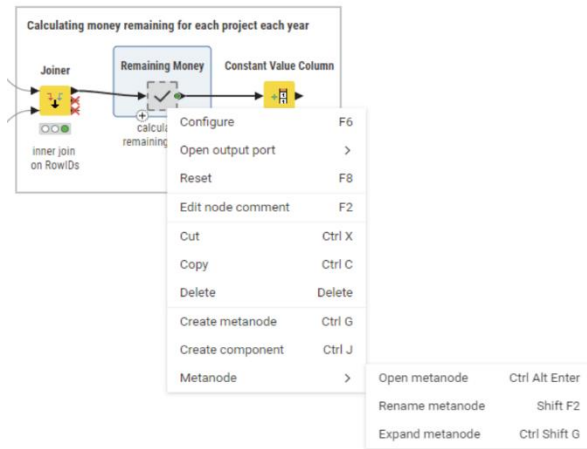



*Figure 5.21. Metanode context menu.*

- "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

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.

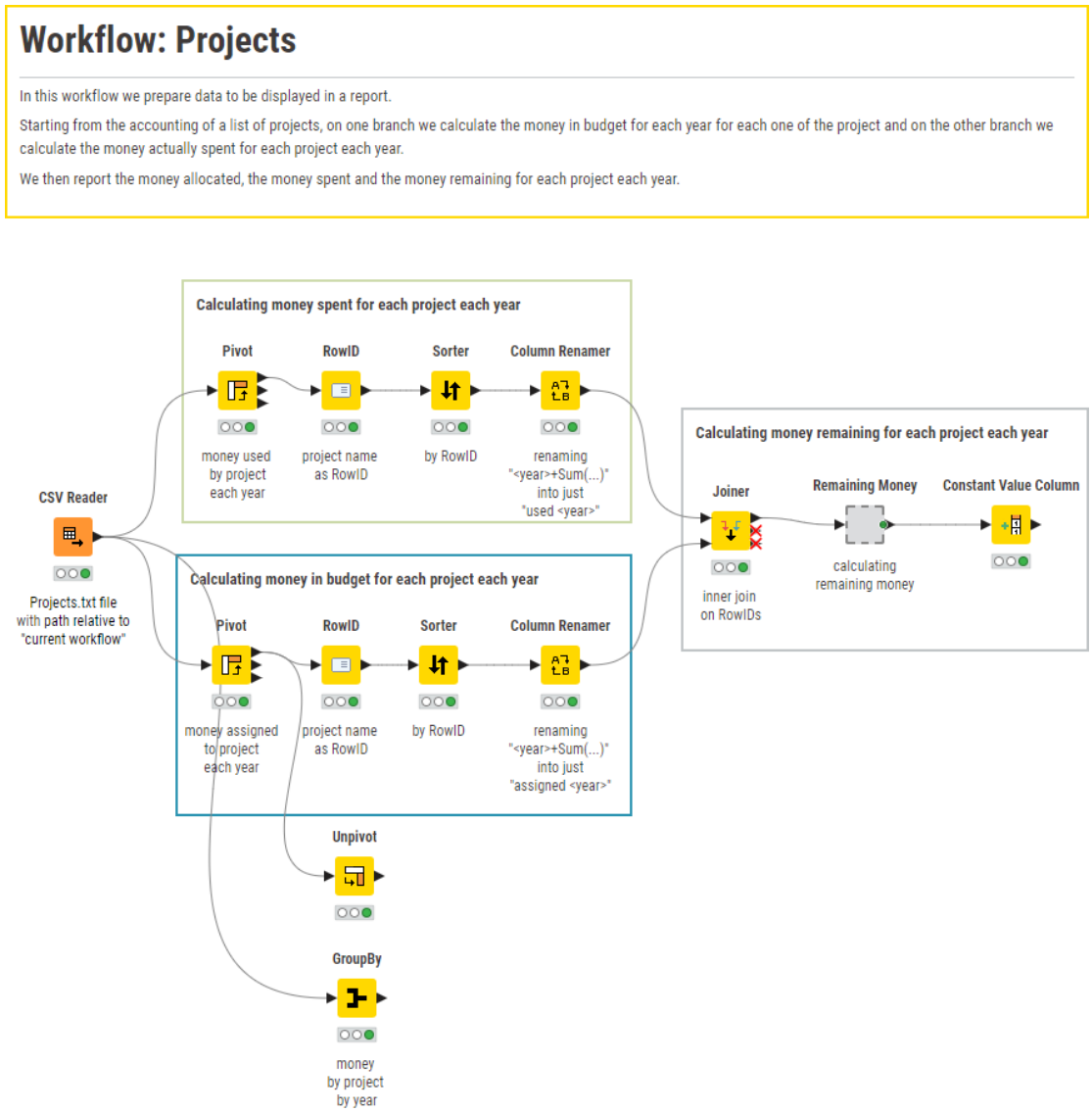



*Figure 5.22. Workflow "Projects_final".*

## 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.

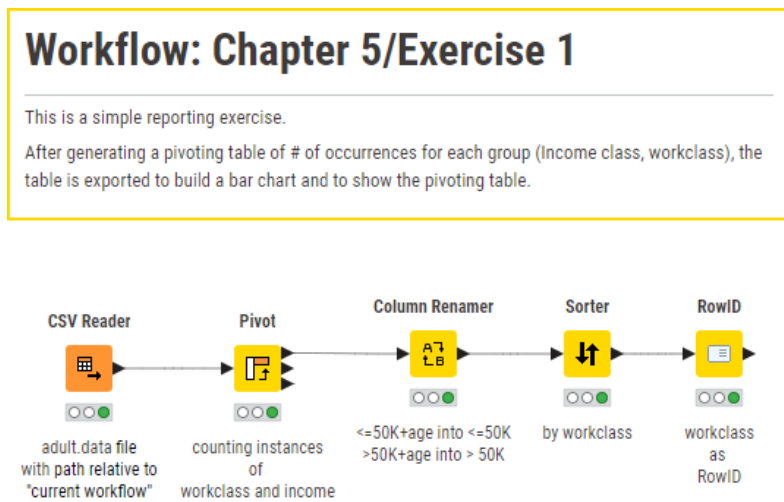



*Figure 5.23. Exercise 1: The workflow.*

## 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.

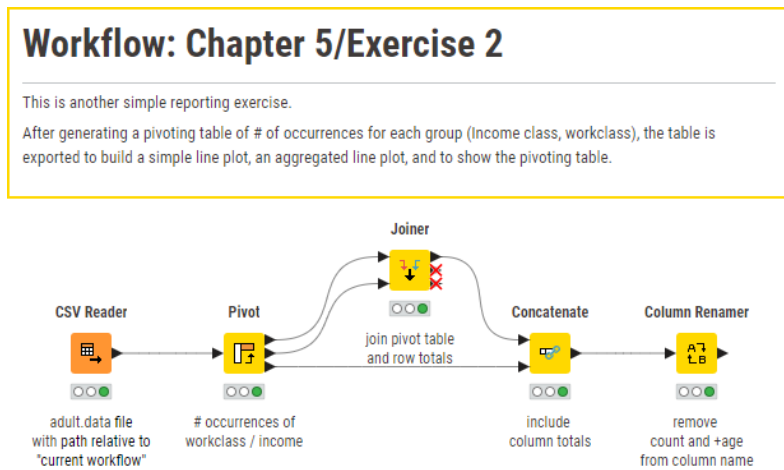



*Figure 5.24. Exercise 2: The workflow.*

# 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: (total number of scored goals – total number of taken goals)/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

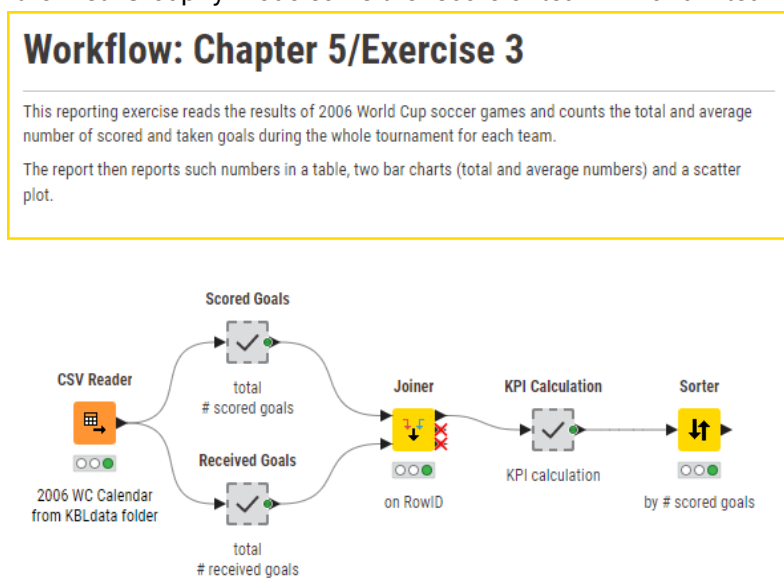



*Figure 5.25. Exercise 3: The workflow.*

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.

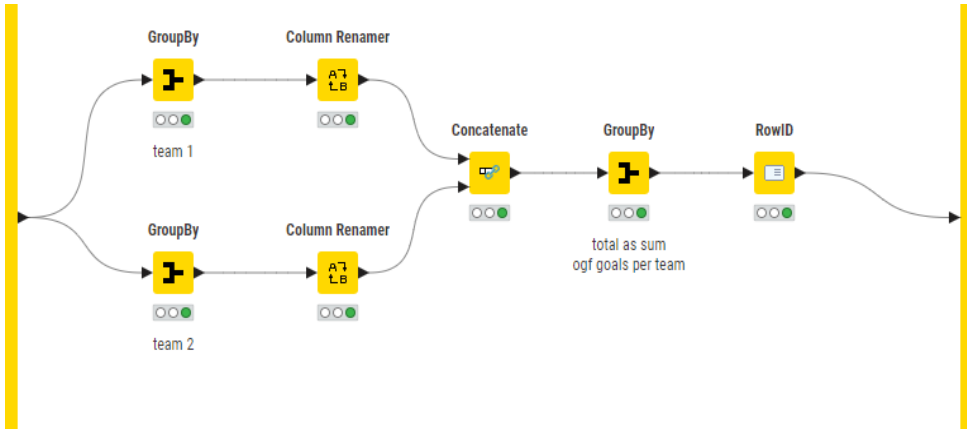



*Figure 5.26. Metanode "Score Goals” and “Received Goals” respectively.*

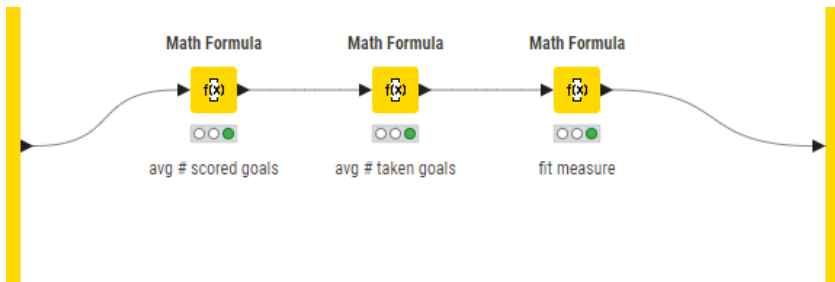



*Figure 5.27. Metanode "KPI Calculation".*

# 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.

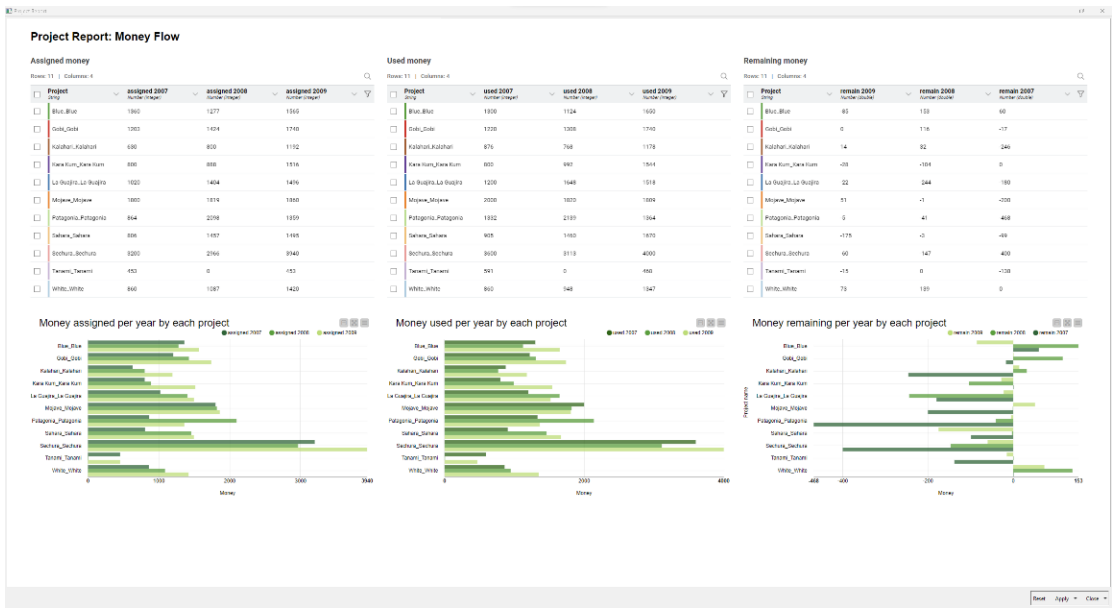



*Figure 6.1. The final dashboard visualizing the tables and bar charts for assigned, used, and remaining money across all 11 projects and the three years of observation.*

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:

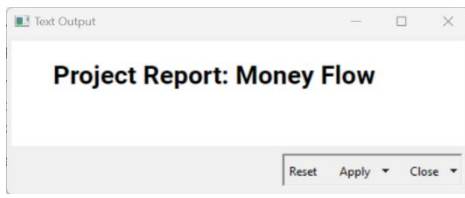



*Figure 6.2. The output of the Text Output Widget node.*

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.

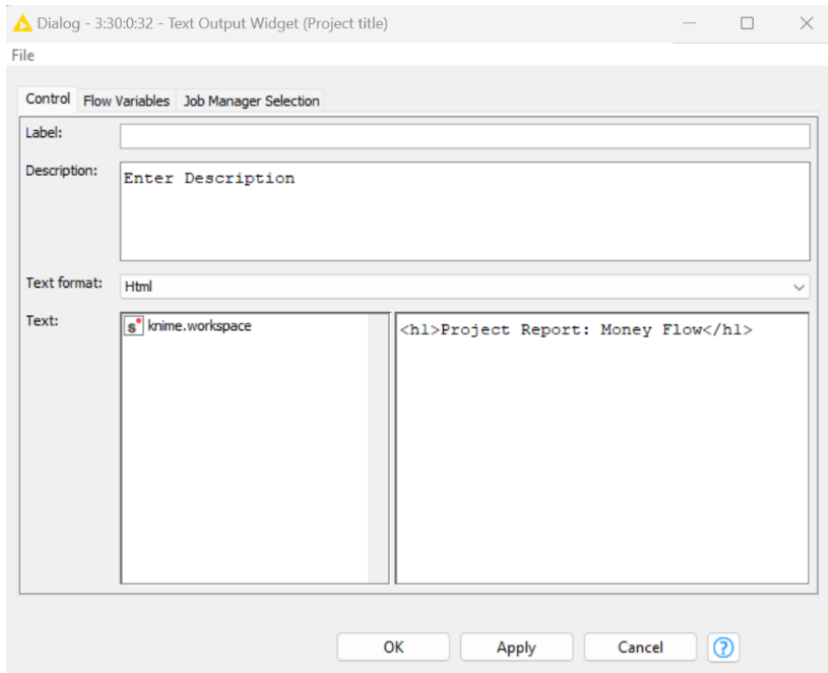



*Figure 6.3. Configuration window for the Text Output Widget node.*

## 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”.

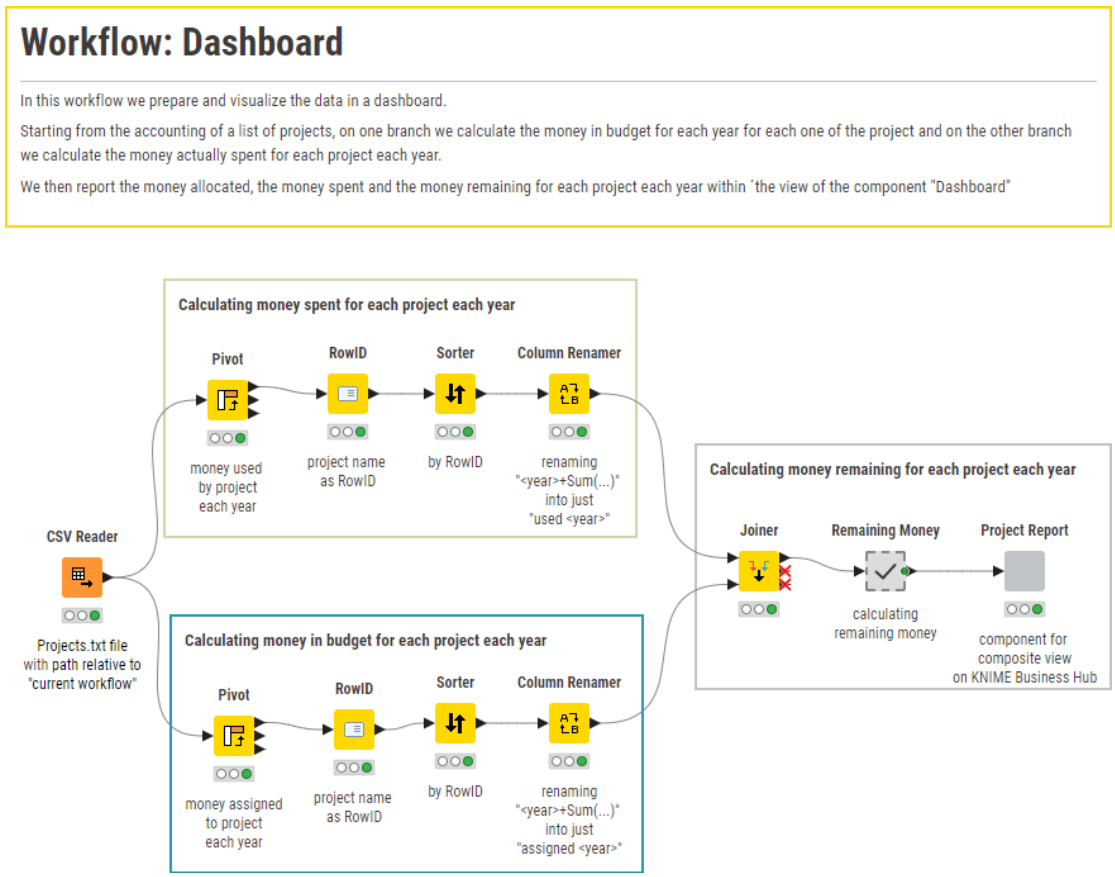



*Figure 6.4. New workflow with component "Dashboard" at the end of the data flow for visualization.*

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.

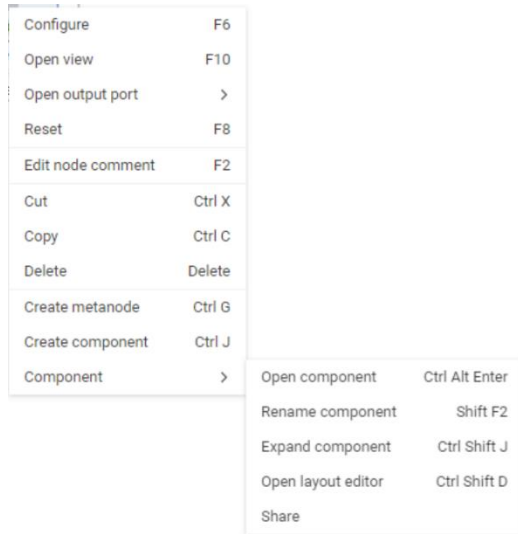



*Figure 6.5. Context menu of a component.*

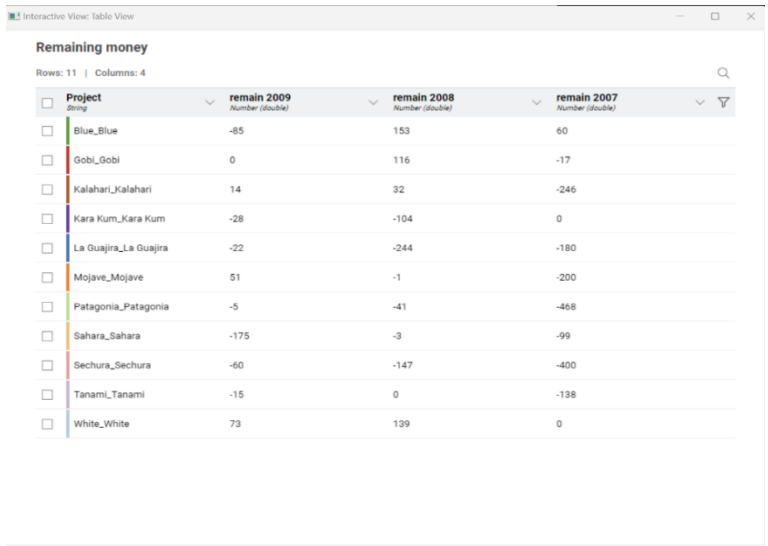
- ***A component can be given a view.*** Components can get a view, metanodes cannot. Inserting one or more nodes from the folder "Workflow Abstraction/Widgets" provides one or more items for your component view. The interactive views of these nodes are passed into the interactive view of the component. 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

(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.


Interactive View: Table View

Remaining money

Rows: 11 | Columns: 4

| Project<br>String | remain 2009<br>Number (double) | remain 2008<br>Number (double) | remain 2007<br>Number (double) |
|---|---|---|---|
| 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.

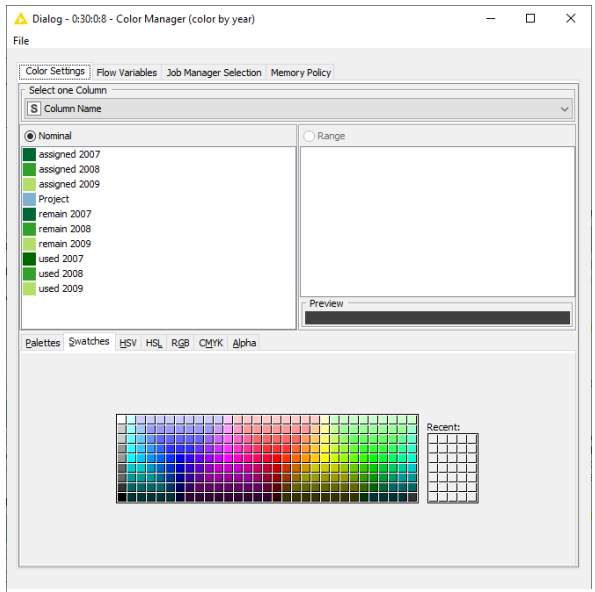



*Figure 6.7. Shades of green assigned to the yearly columns in the Color Manager node.*

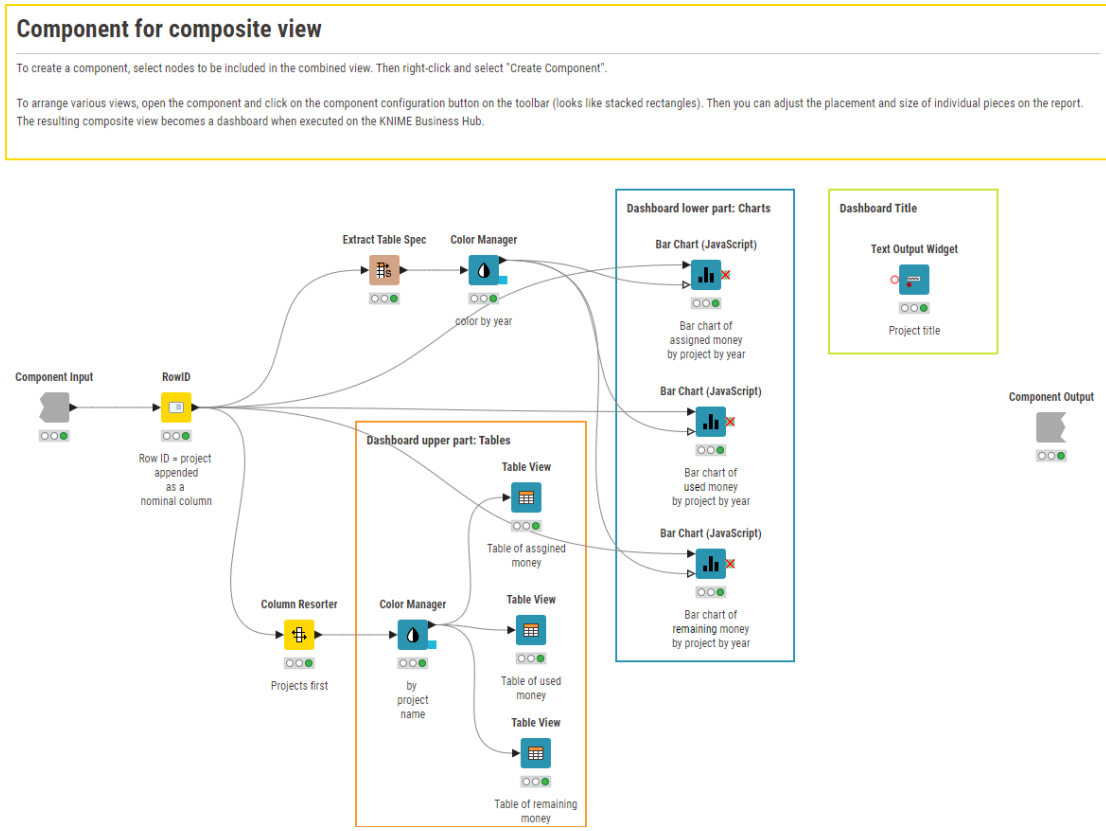



*Figure 6.8. Content of component "Dashboard".*

## 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.

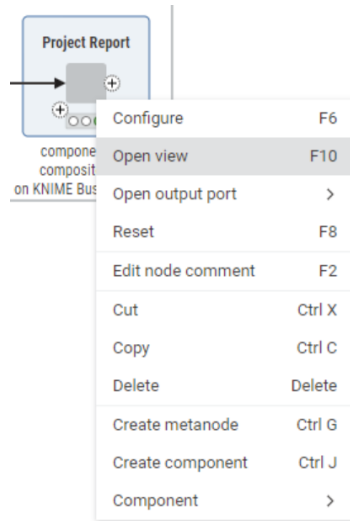



*Figure 6.9. How to open the composite view of a component.*

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

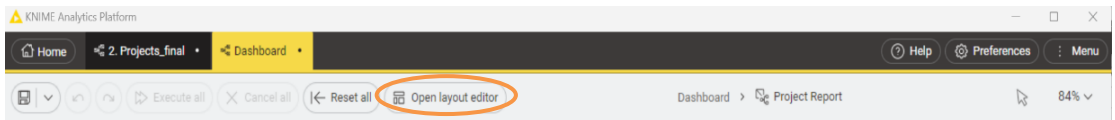



*Figure 6.10. The layout button in the tool bar in the KNIME Workbench.*

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.

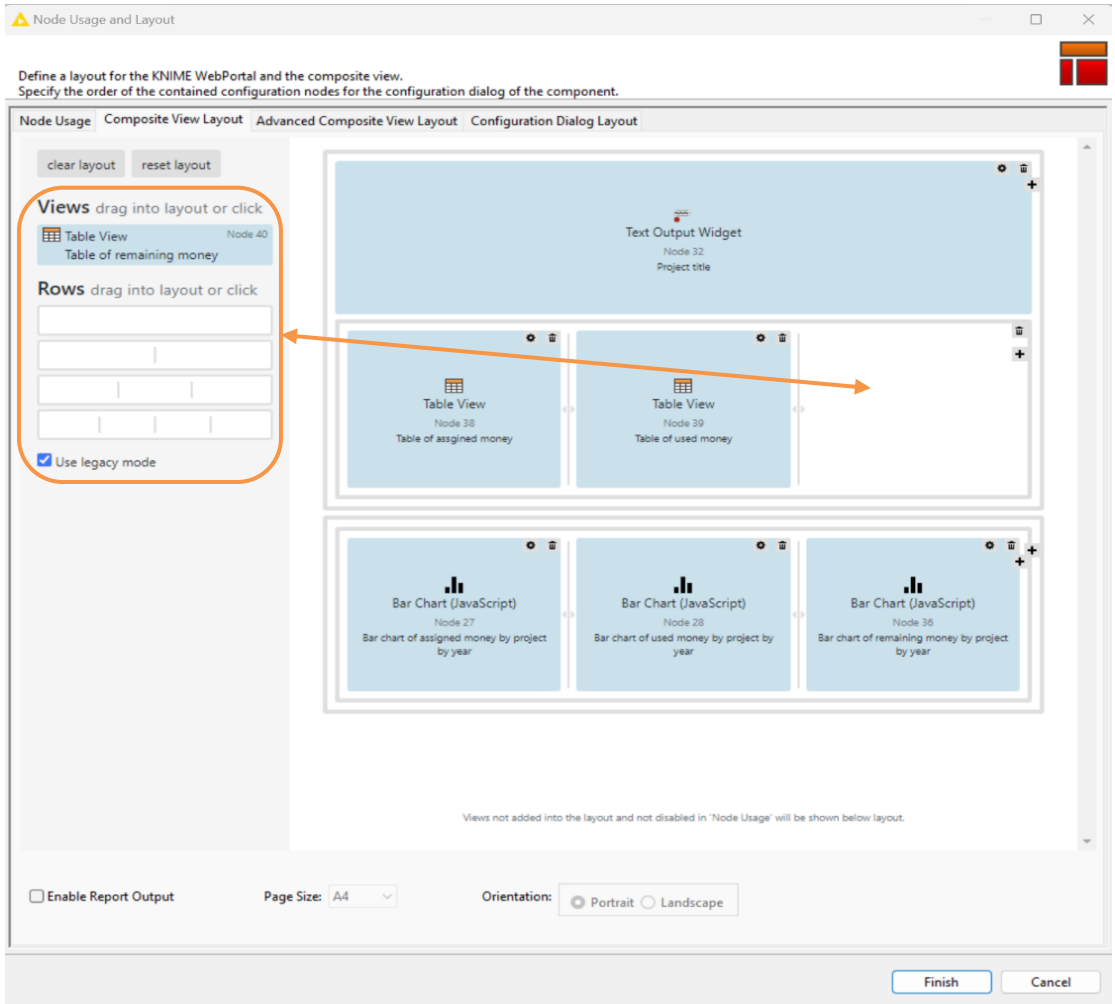



*Figure 6.11. The Layout Editor.*

## 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.

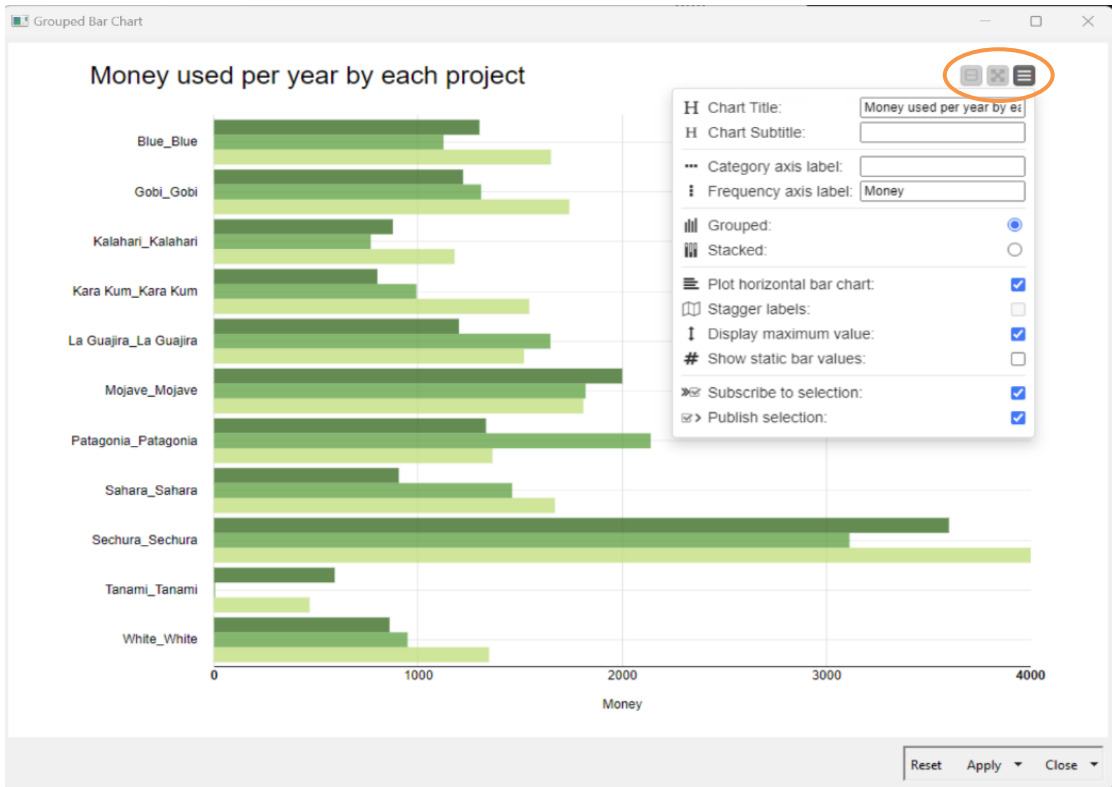



*Figure 6.12. One of the bar charts in the composite view of component "Dashboard".*

## 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

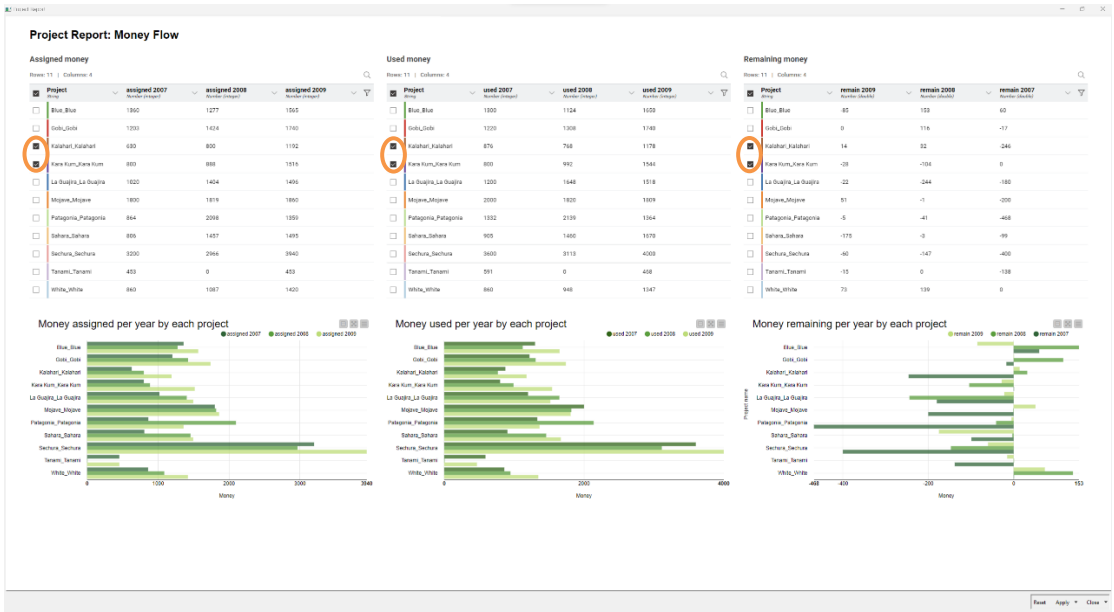



*Figure 6.13. Selection of points across all view items.*

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

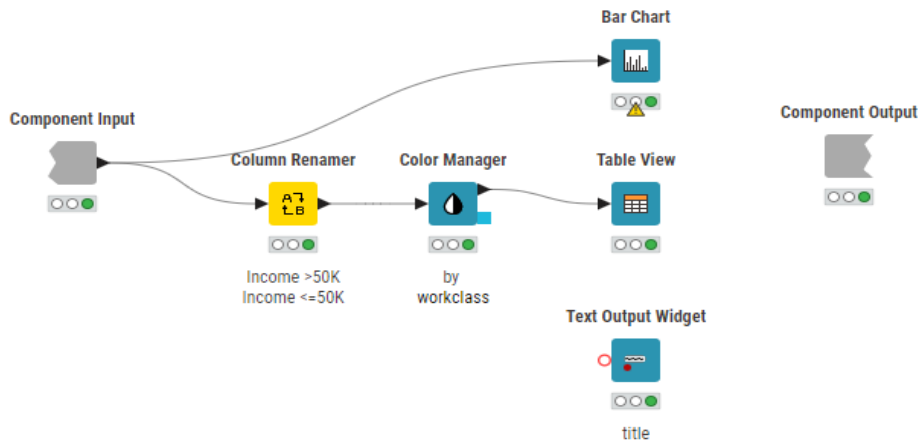



*Figure 6.14. Component "Dashboard2" to implement required composite view.*

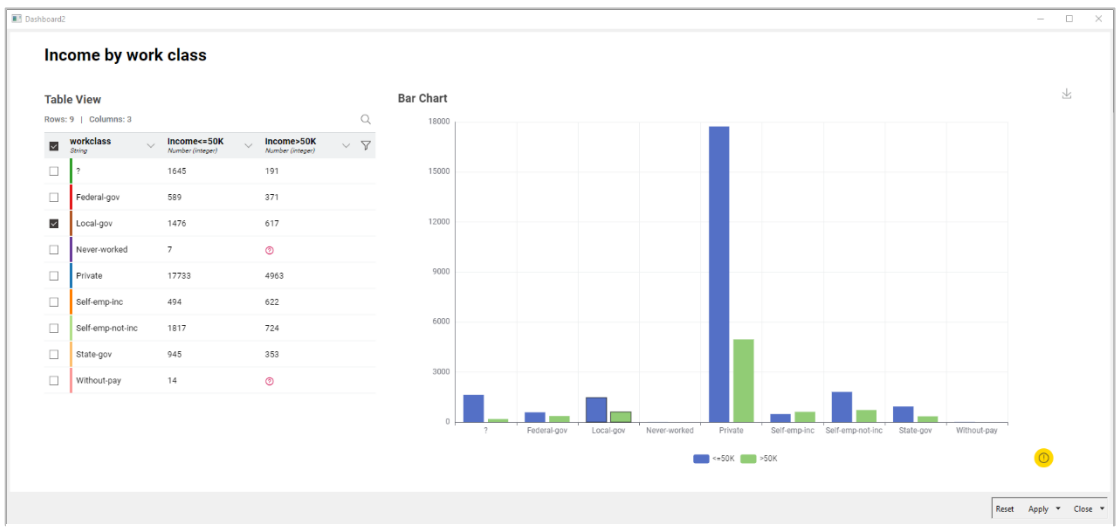



*Figure 6.15. The final composite view with class "Local-gov " selected in both the table and the bar chart.*

# 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

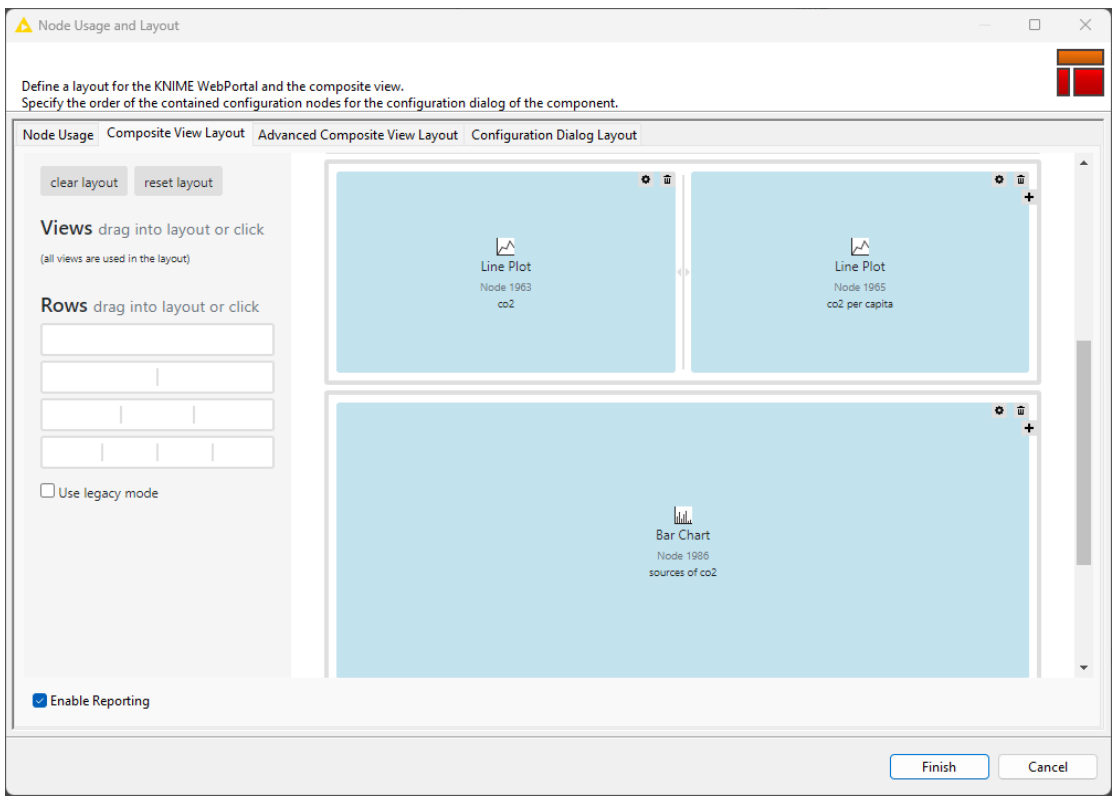



*Figure 7.1. The Node usage and layout window*

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.

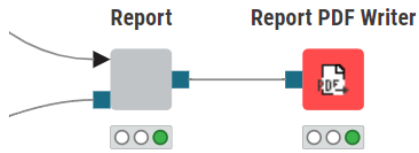



*Figure 7.2. The Report PDF Writer node.*

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.

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.

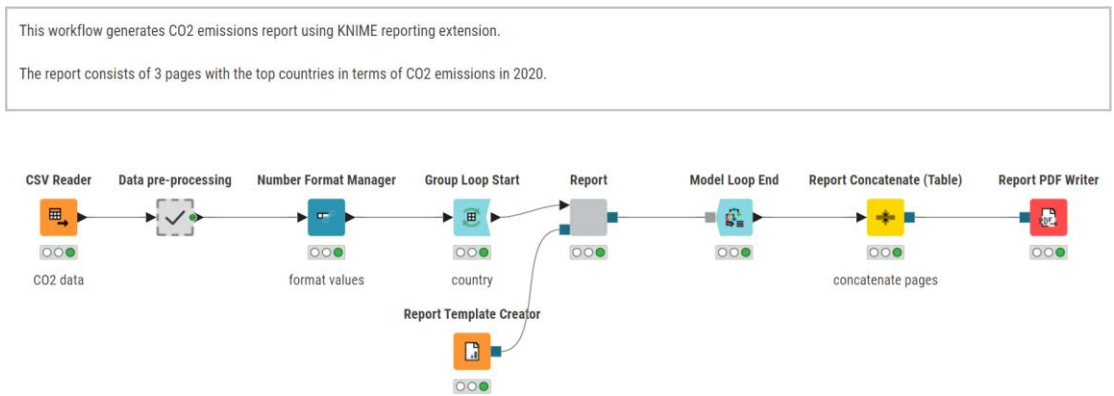



*Figure 7.3. Workflow generating CO2 emissions report using the KNIME Reporting extension.*

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.

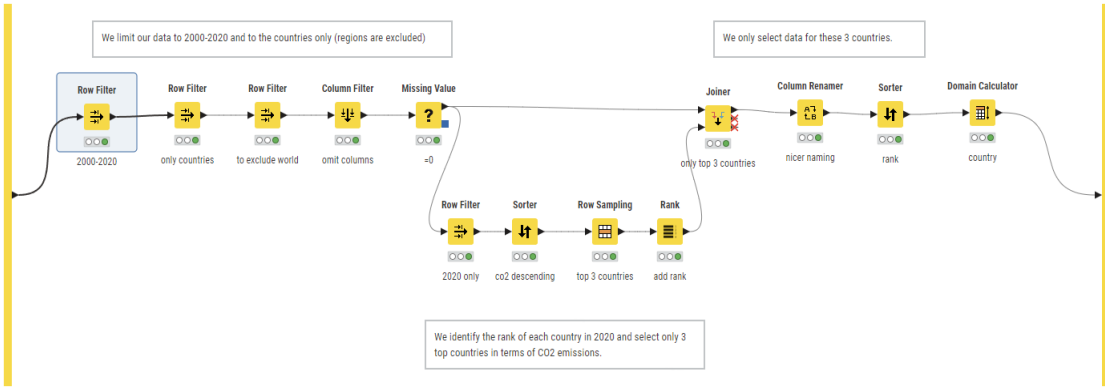



*Figure 7.4. Data Pre-processing steps inside the metanode.*

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.

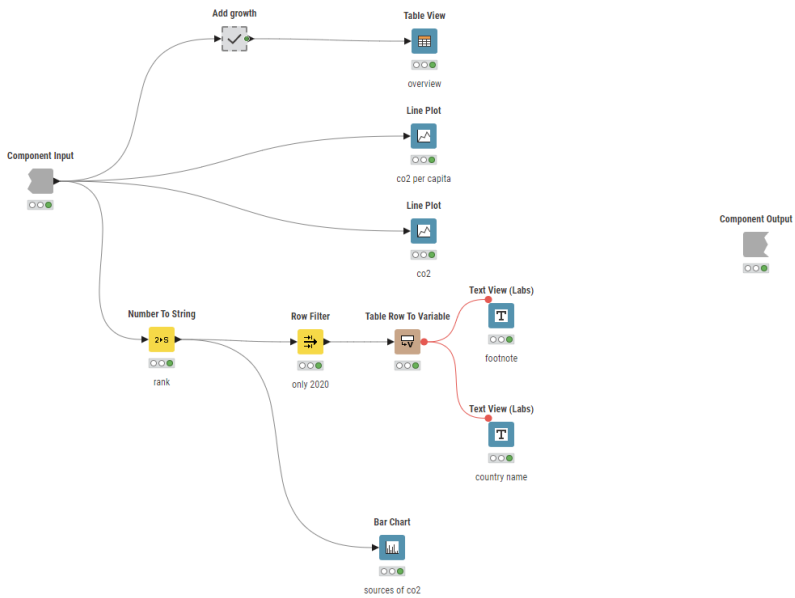



*Figure 7.5. Visualization nodes inside the "Report" component.*

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.

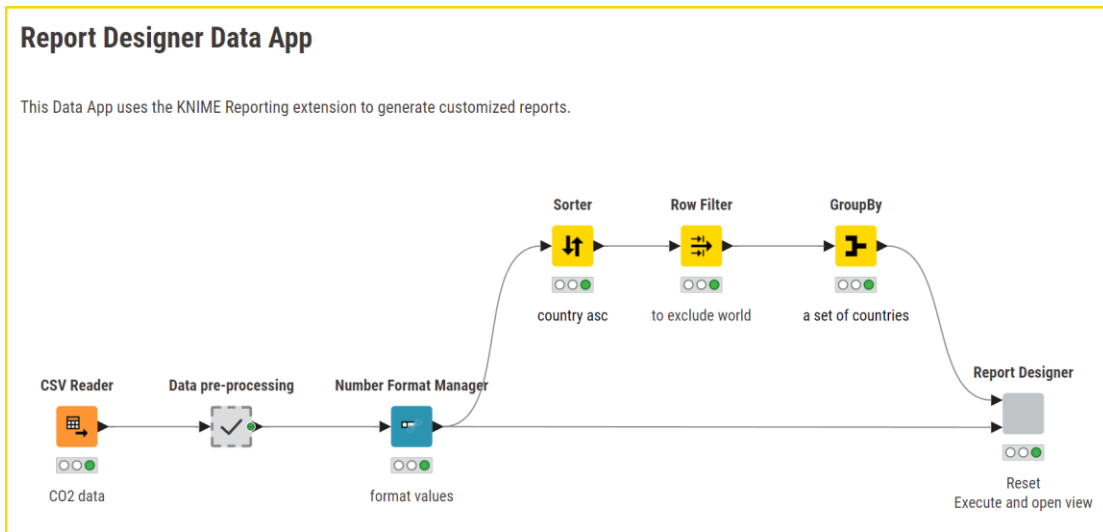



*Figure 7.6. Preprocessed data to be sent for report generation.*

## 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

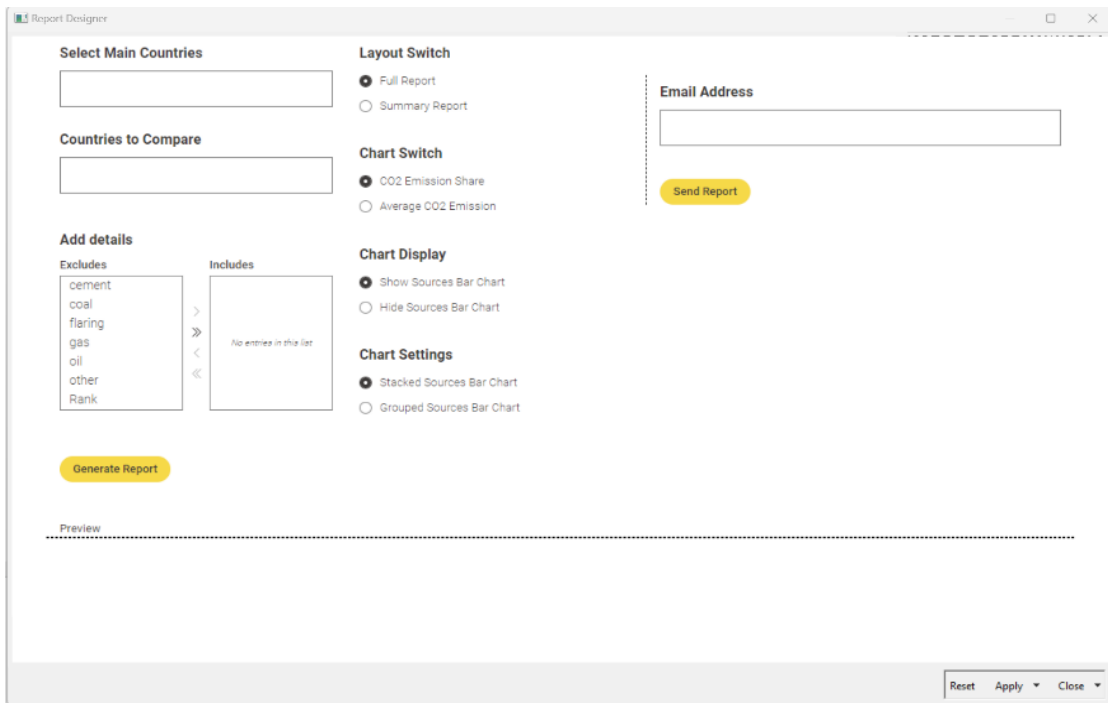



*Figure 7.7. DataApp showing different selection options.*

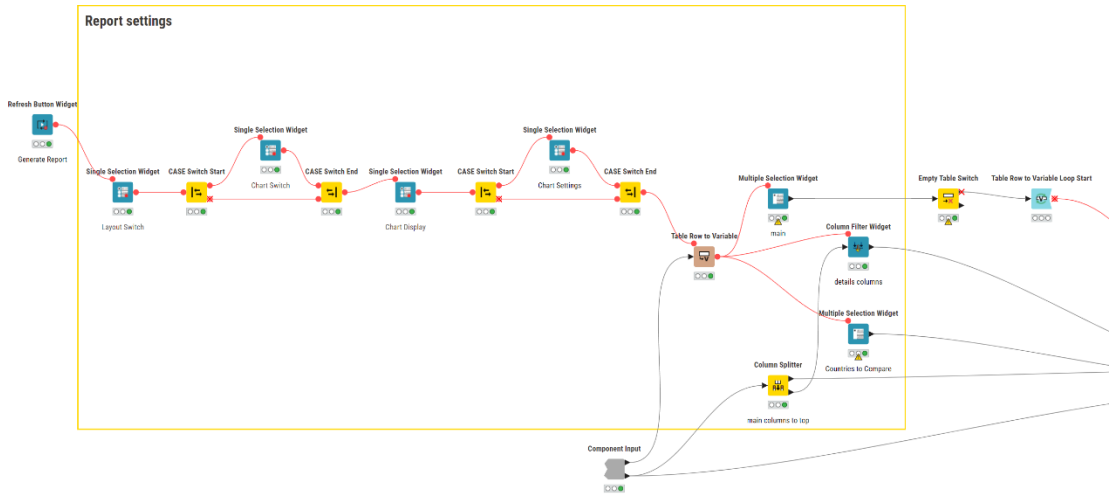



*Figure 7.8. Preprocessed data to be sent for report generation.*

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

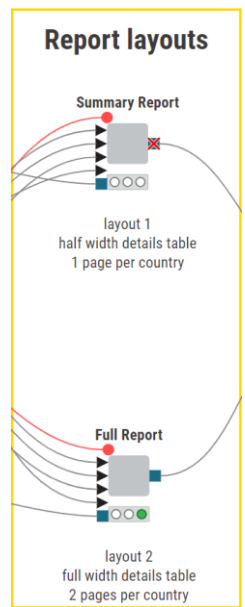



*Figure 7.9. Components to select different layout options.*

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.

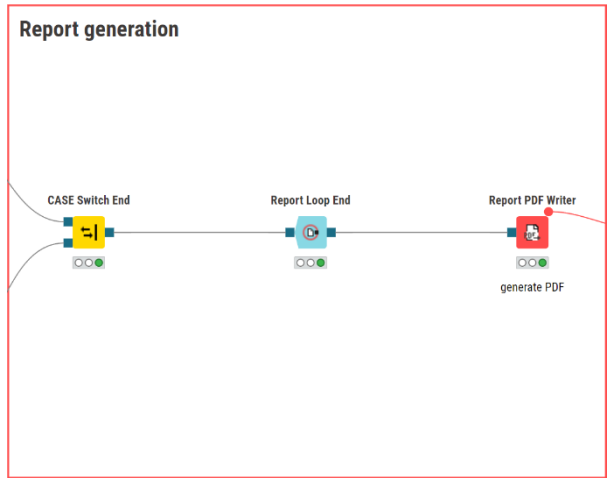



*Figure 7.10. Section of the workflow that generates report.*

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.

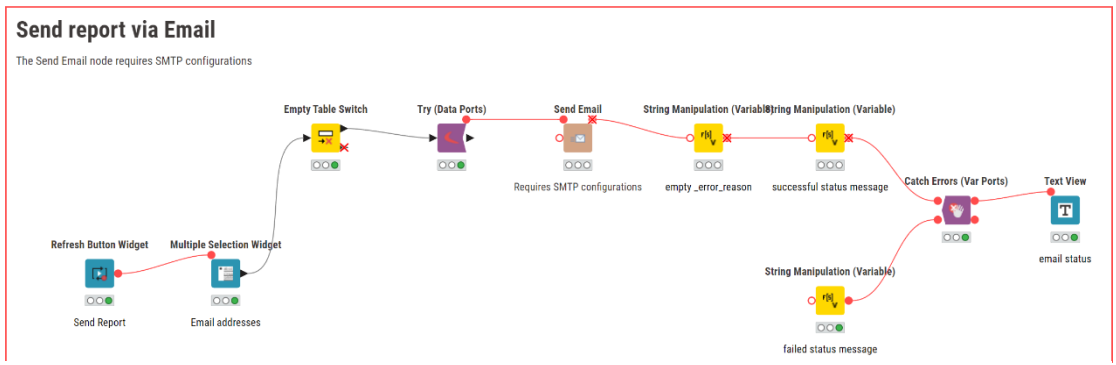



*Figure 7.11. Section of the workflow that gathers email addresses and sends the report using the “send email” node.*

## 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

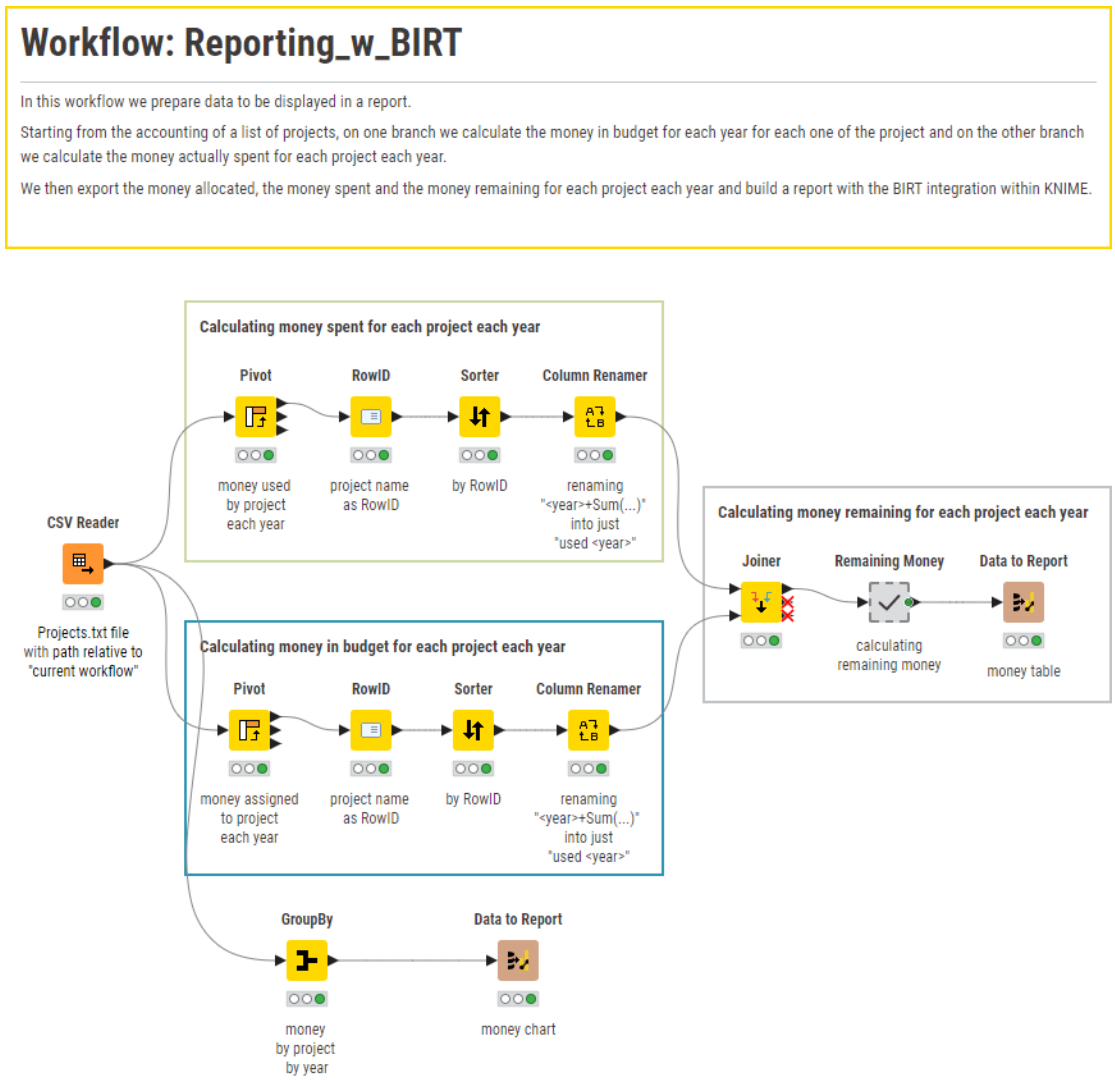



*Figure 7.12. The "Reporting_w_BIRT" workflow.*

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

Table View - 2:15 - Data to Report (Money table)

File Hilite Navigation View Output

| 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 and2009. 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

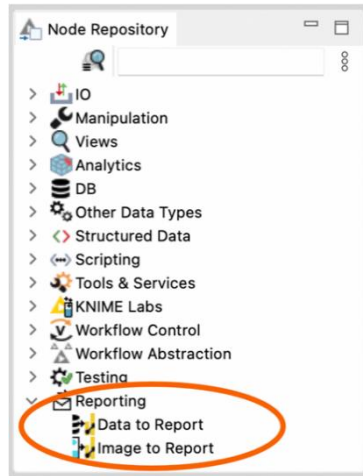



*Figure 7.14. The Reporting category in the Node Repository.*

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.

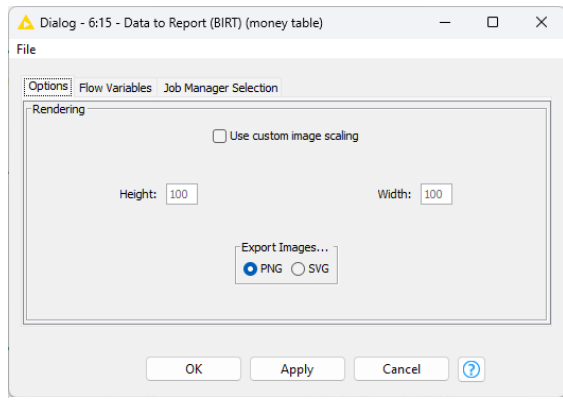



*Figure 7.15. Configuration window of the Data to Report node.*

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.

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.

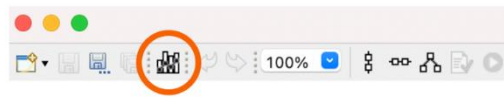



*Figure 7.16. The report icon in the tool bar.*


*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

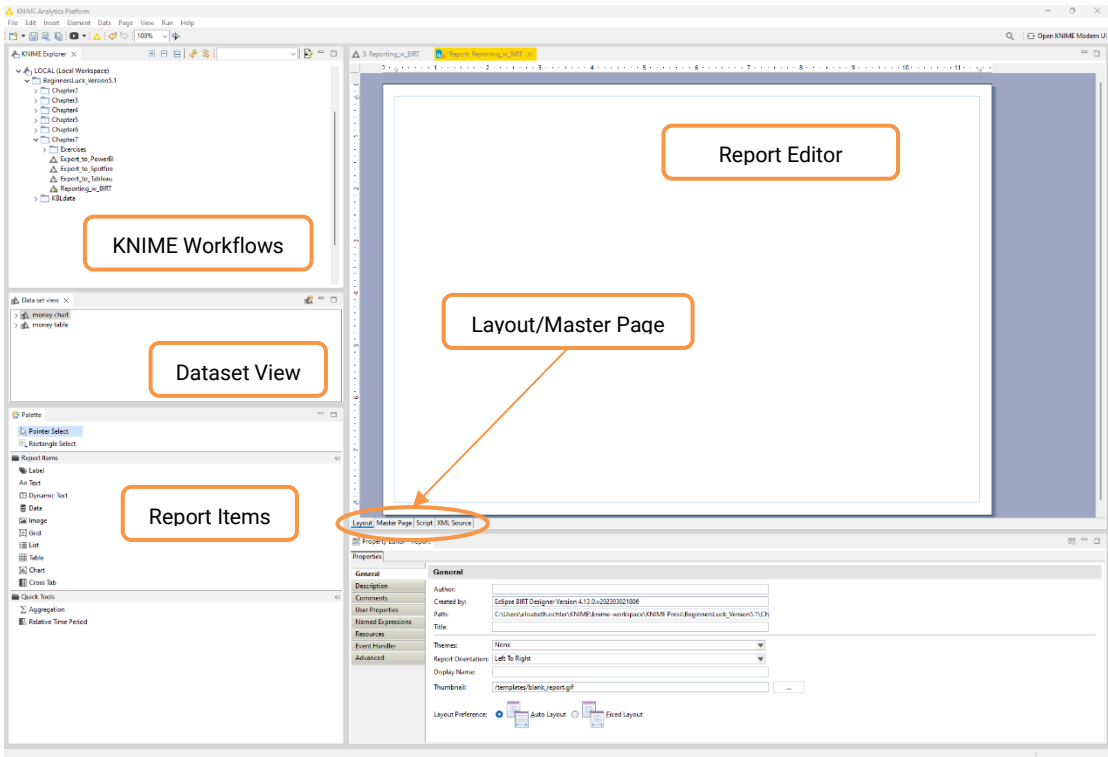



*Figure 7.18. The Report Editor in the BIRT environment.*

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 in[9] 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*

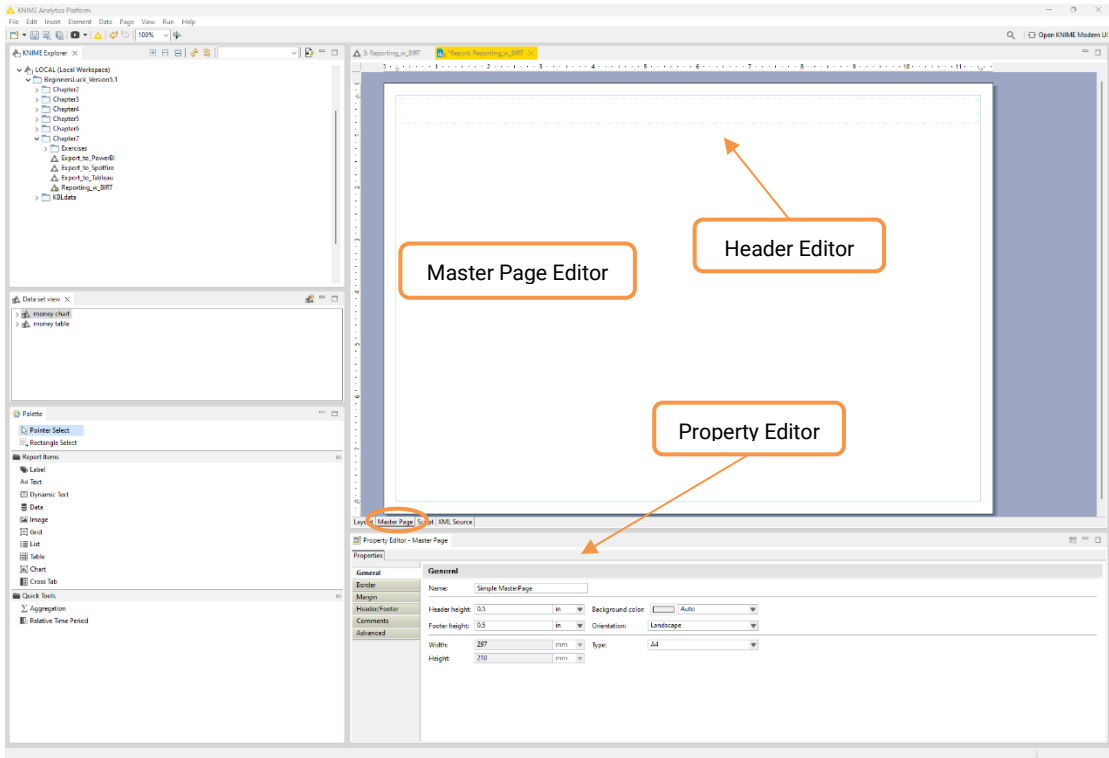



*Figure 7.19 The Header Editor inside the Master Page Editor.*

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.

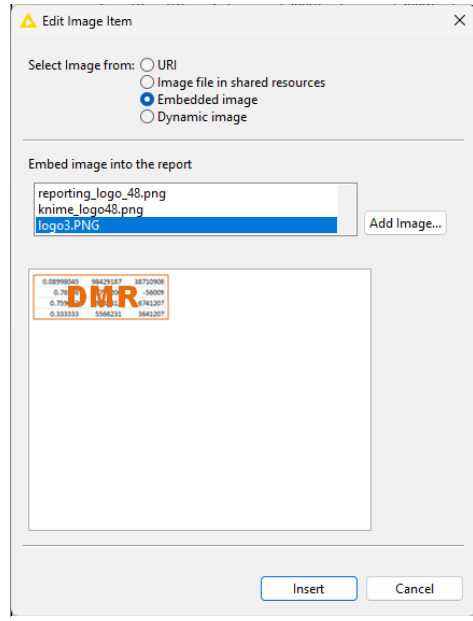



*Figure 7.20. The "Edit Image Item" window.*

## 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:

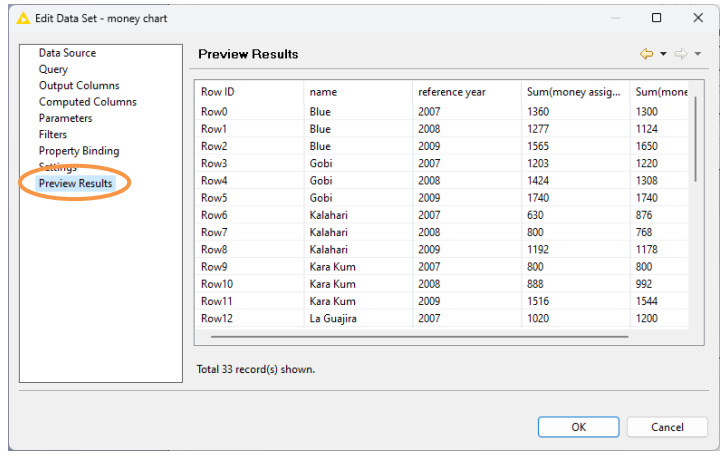



*Figure 7.21. "Preview Results" shows the content of the dataset.*

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

## 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.

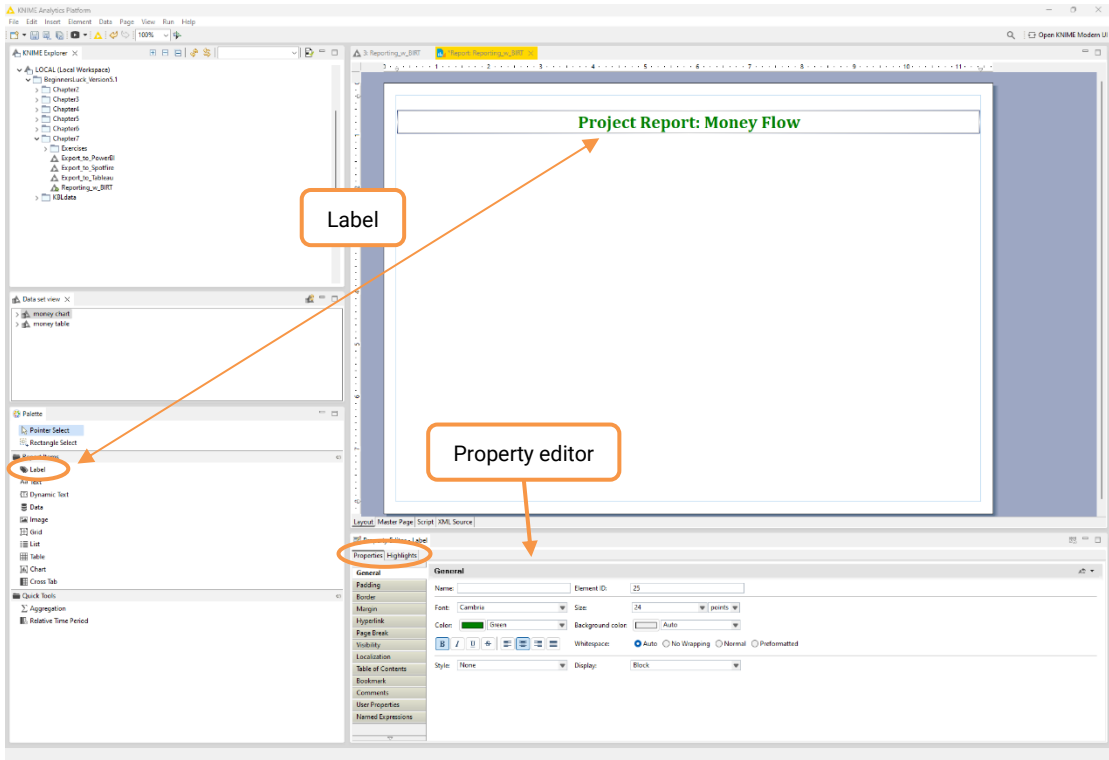



*Figure 7.22. Drag and drop a "Label" item into the Report Editor to create the report title.*

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.

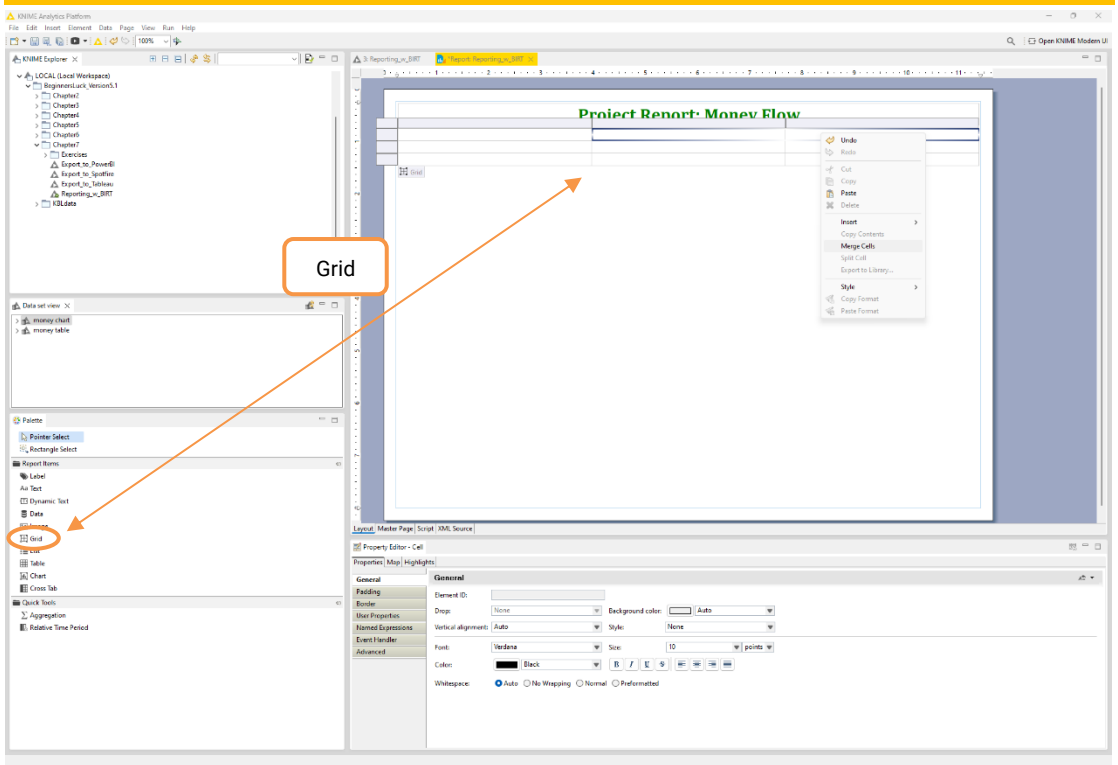



*Figure 7.23. Drag and drop the "Grid" report item into the Report editor, select 3 rows and 3 columns, and merge the two cells in the middle row.*

## 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.

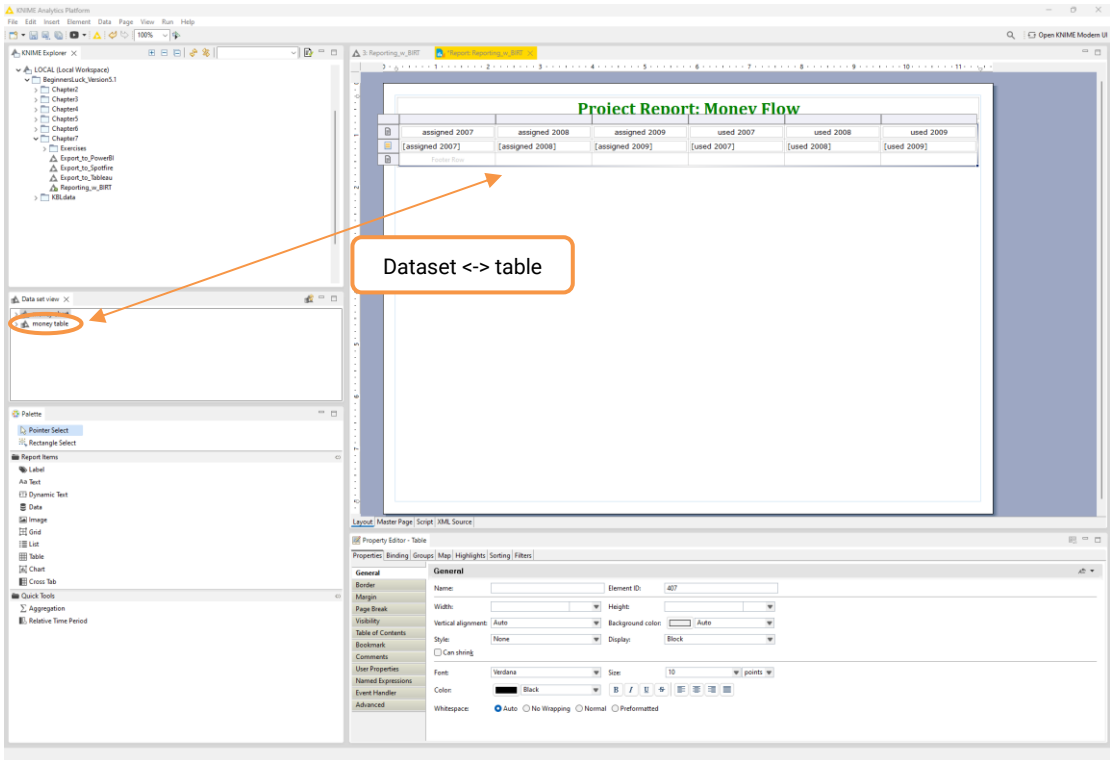



*Figure 7.24. Drag and drop a dataset from the "Data Set View" panel to produce a table with as many columns as many dataset's fields.*

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:

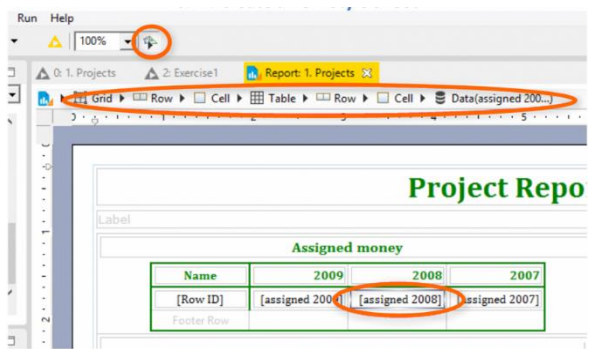


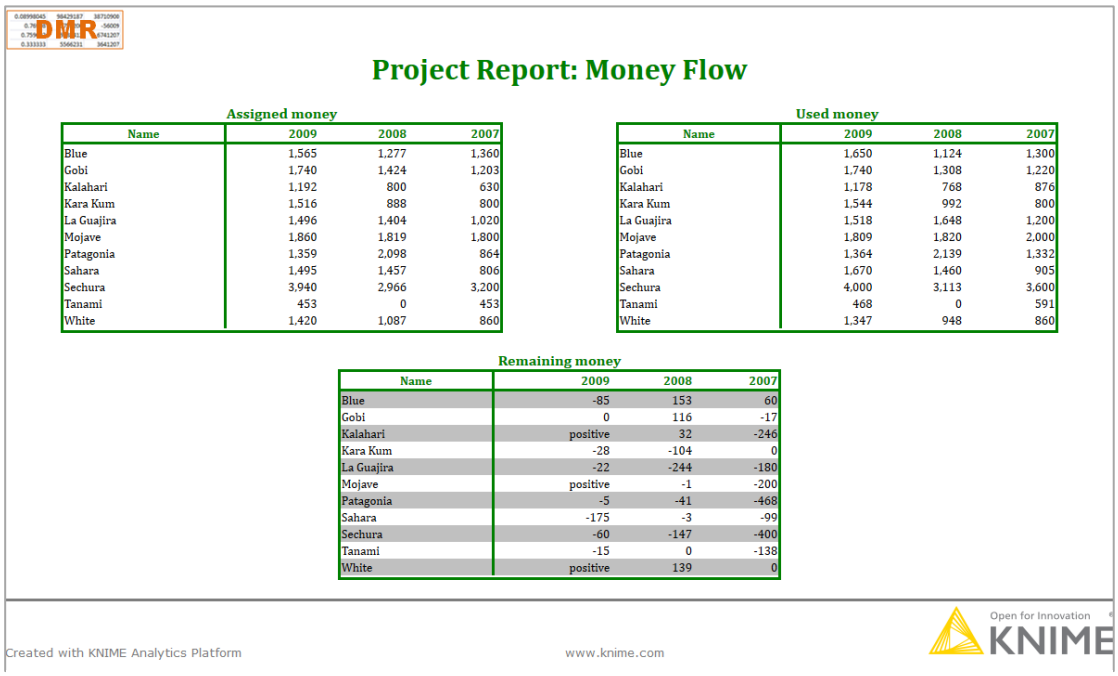

*Figure 7.25. Toggle Breadcrumb button.*

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.



**Project Report: Money Flow**

Assigned money

| Name | 2009 | 2008 | 2007 |
|---|---|---|---|
| Blue | 1,565 | 1,277 | 1,360 |
| Gobi | 1,740 | 1,424 | 1,203 |
| Kalahari | 1,192 | 800 | 630 |
| Kara Kum | 1,516 | 888 | 800 |
| La Guajira | 1,496 | 1,404 | 1,020 |
| Mojave | 1,860 | 1,819 | 1,800 |
| Patagonia | 1,359 | 2,098 | 864 |
| Sahara | 1,495 | 1,457 | 806 |
| Sechura | 3,940 | 2,966 | 3,200 |
| Tanami | 453 | 0 | 453 |
| White | 1,420 | 1,087 | 860 |

Used money

| Name | 2009 | 2008 | 2007 |
|---|---|---|---|
| Blue | 1,650 | 1,124 | 1,300 |
| Gobi | 1,740 | 1,308 | 1,220 |
| Kalahari | 1,178 | 768 | 876 |
| Kara Kum | 1,544 | 992 | 800 |
| La Guajira | 1,518 | 1,648 | 1,200 |
| Mojave | 1,809 | 1,820 | 2,000 |
| Patagonia | 1,364 | 2,139 | 1,332 |
| Sahara | 1,670 | 1,460 | 905 |
| Sechura | 4,000 | 3,113 | 3,600 |
| Tanami | 468 | 0 | 591 |
| White | 1,347 | 948 | 860 |

Remaining money

| Name | 2009 | 2008 | 2007 |
|---|---|---|---|
| Blue | -85 | 153 | 60 |
| Gobi | 0 | 116 | -17 |
| Kalahari | positive | 32 | -246 |
| Kara Kum | -28 | -104 | 0 |
| La Guajira | -22 | -244 | -180 |
| Mojave | positive | -1 | -200 |
| Patagonia | -5 | -41 | -468 |
| Sahara | -175 | -3 | -99 |
| Sechura | -60 | -147 | -400 |
| Tanami | -15 | 0 | -138 |
| White | positive | 139 | 0 |

Created with KNIME Analytics Platform www.knime.com Open for Innovation KNIME

*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 "Pr operties" editor of table report items; that is cells, rows, columns, and even the whole table.

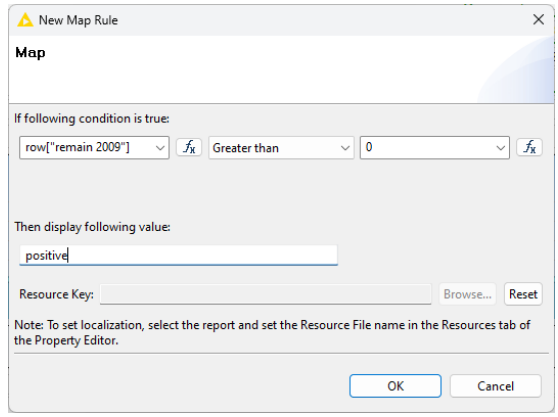



*Figure 7.27. The "New Map Rule" editor.*

- 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"

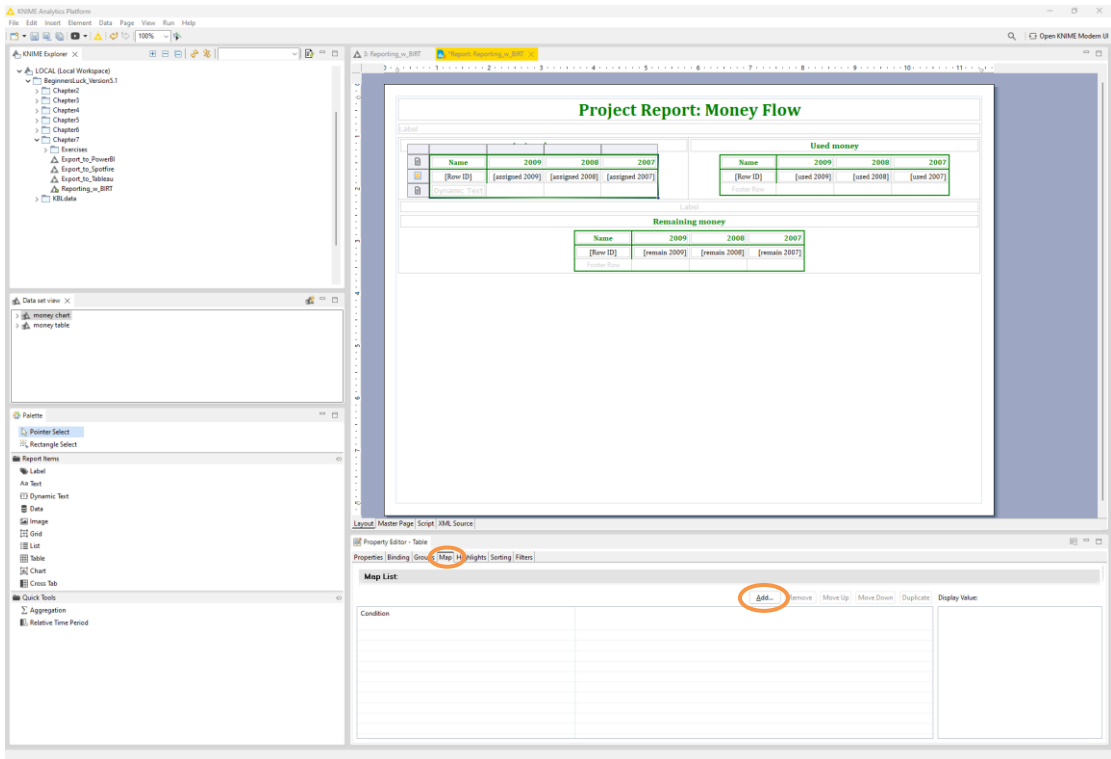



*Figure 7.28. The tab "Maps" in the Table Properties Editor defines text mapping for a table item.*

## 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.

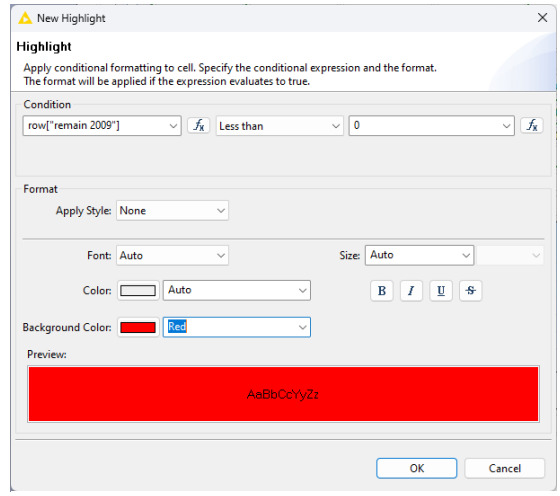



*Figure 7.29. The "Highlights Rule Editor".*

- 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.

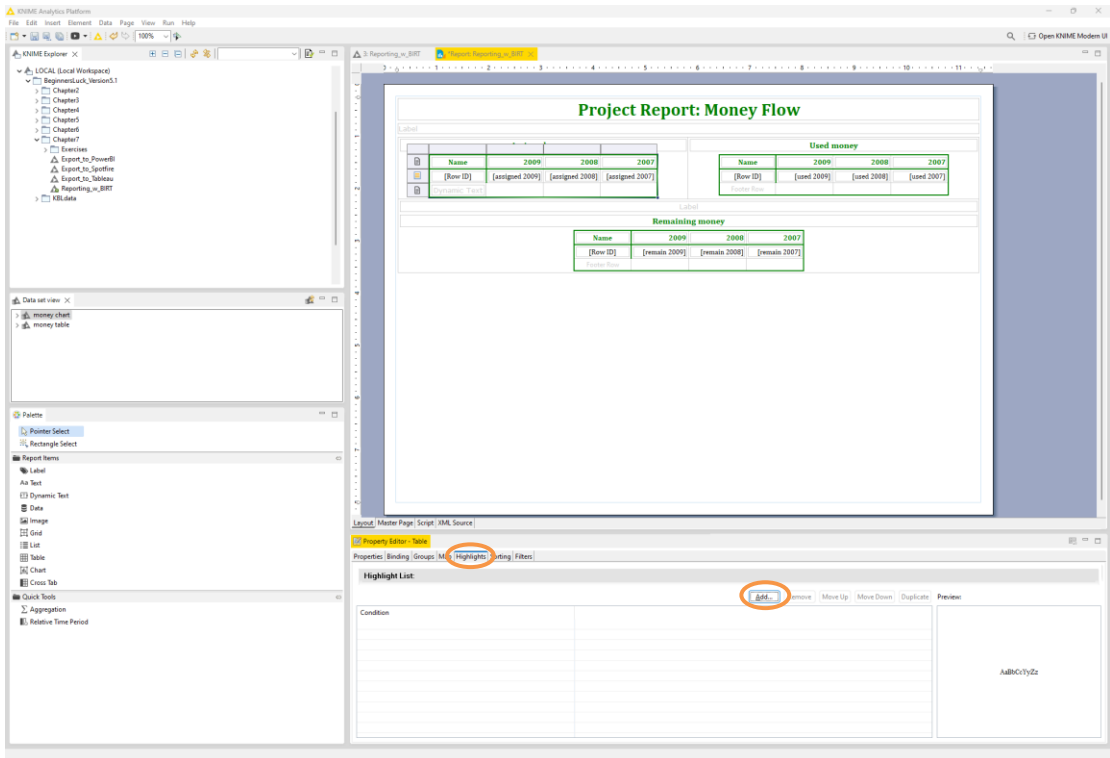



*Figure 7.30. The "Highlights" tab in the Properties Editor defines conditional properties for a table item.*

## 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.

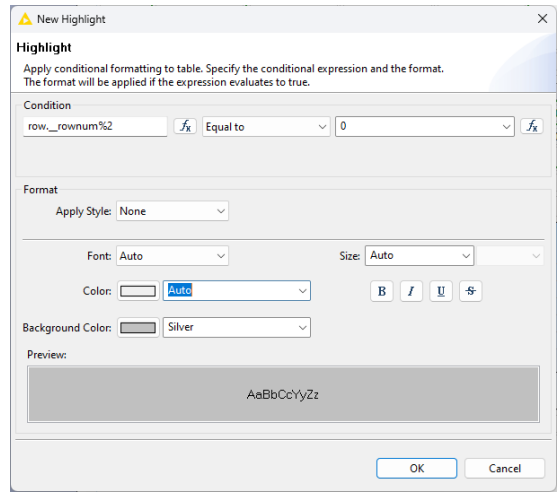



*Figure 7.31. The icon to open the "Expression Builder Editor".*

**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.

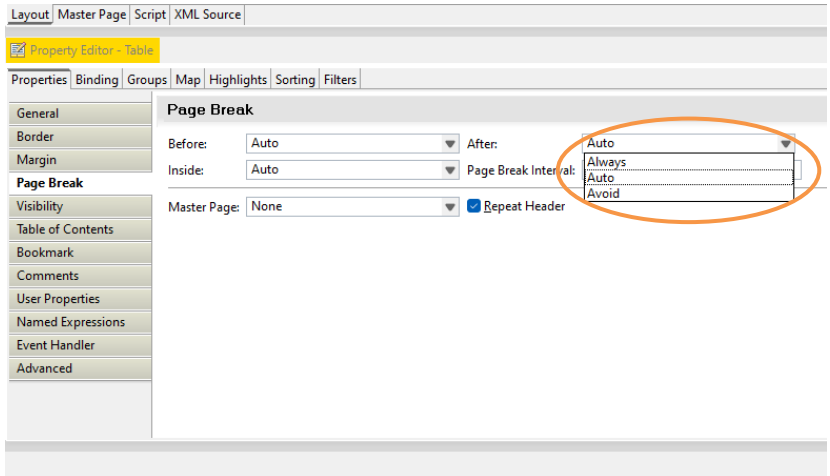



*Figure 7.33. "Page Break" tab in the Property editor.*

## 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.

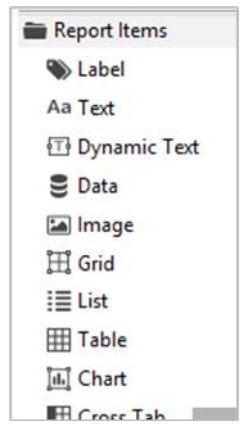



*Figure 7.34. "Chart" report item in the "Report Item" panel.*

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

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.

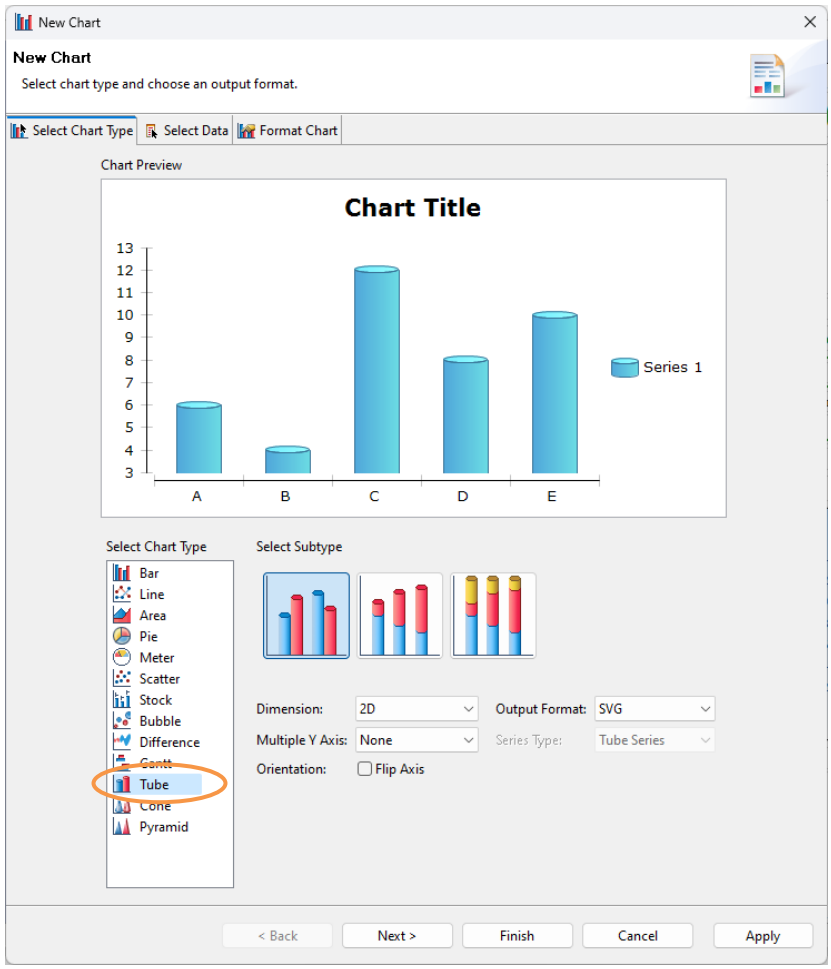



*Figure 7.35. First Step of the "Chart Wizard": Select Chart Type.*

## 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).

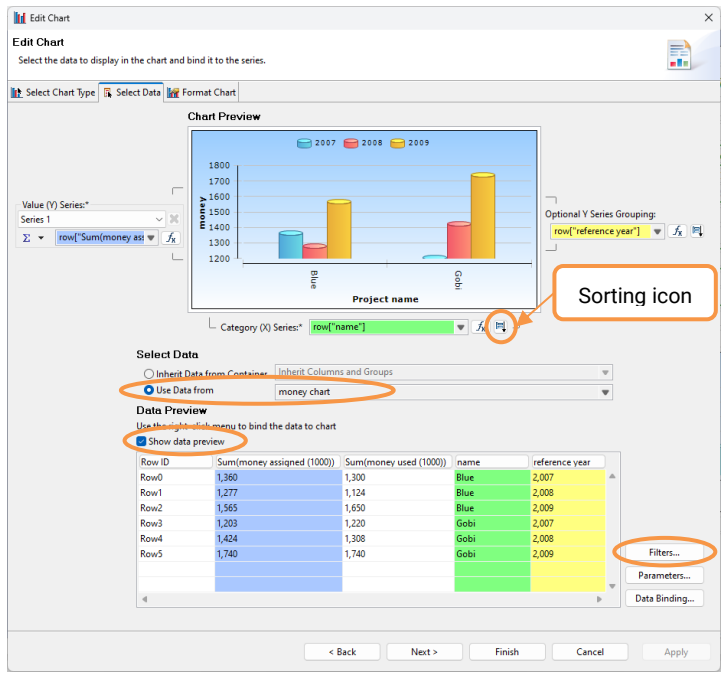



*Figure 7.36. Second Step of the "Chart Wizard": Select Data.*

- 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

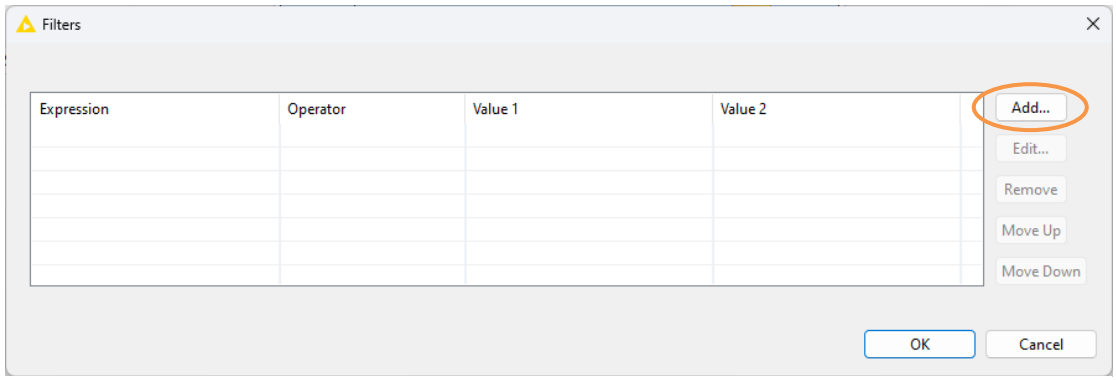



*Figure 7.37. The "Filters" window.*

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.

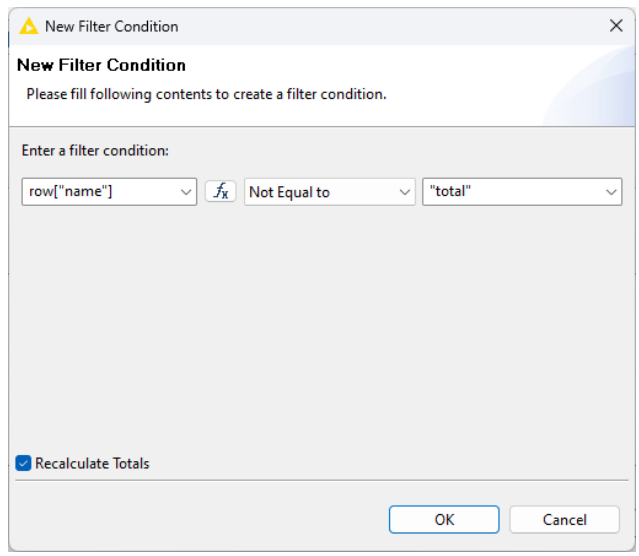



*Figure 7.38. The "Filter Condition Editor".*

## Format Chart

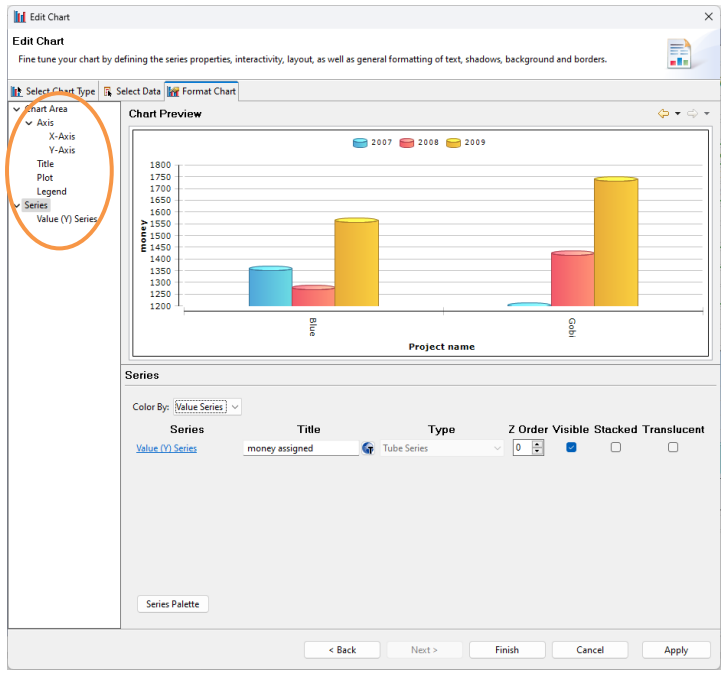



*Figure 7.39. Third step of the "Chart Wizard": Format Chart.*

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

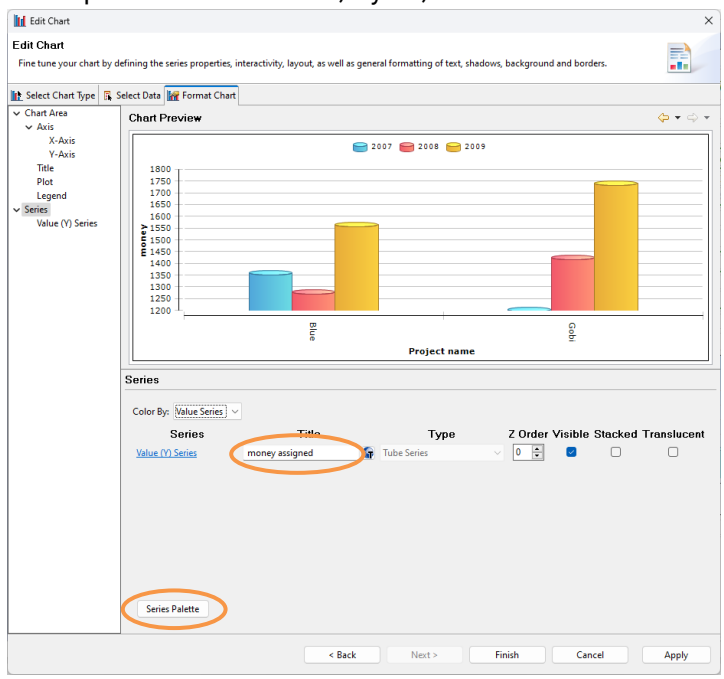



*Figure 7.40. Chart Format: Series.*

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.

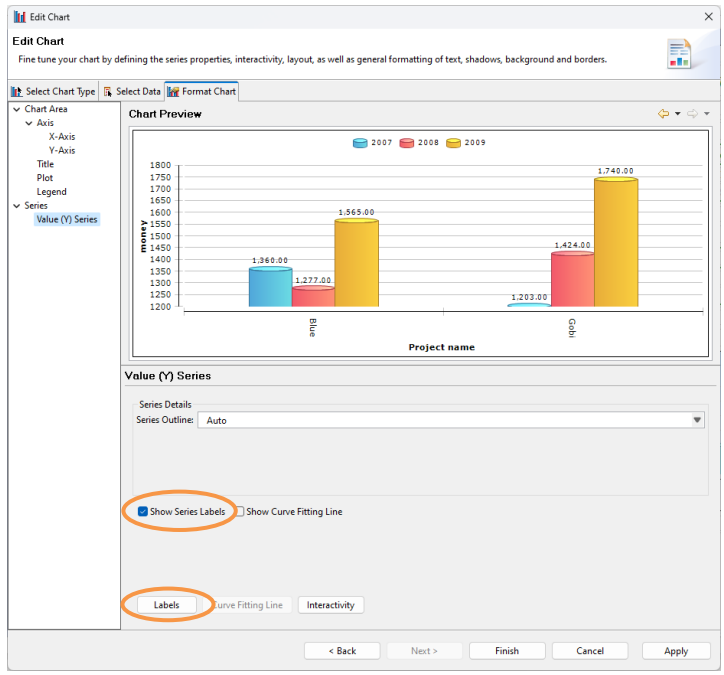



*Figure 7.41. Chart Format: Value (Y) Series.*

## 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).

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

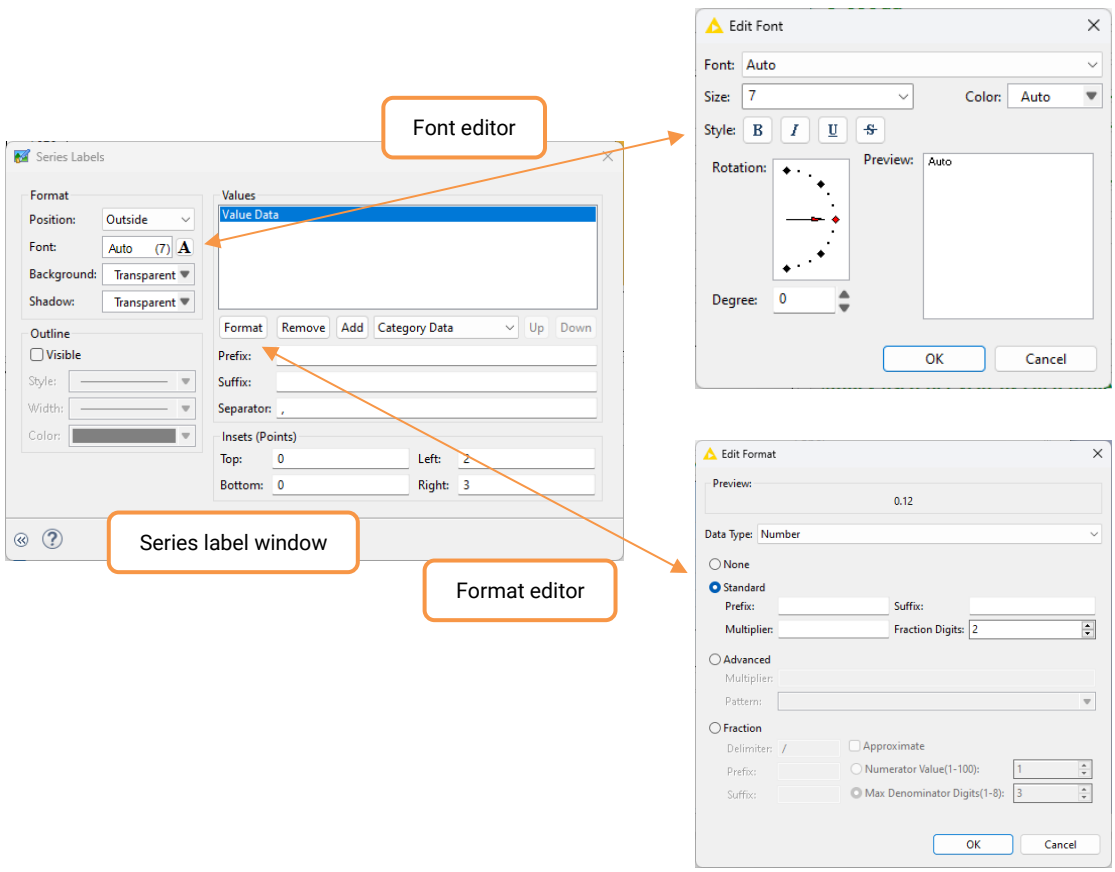



*Figure 7.42. The "Series Labels" window: The "A" button opens the "Font Editor". The "Format" button opens the "Format Editor".*

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

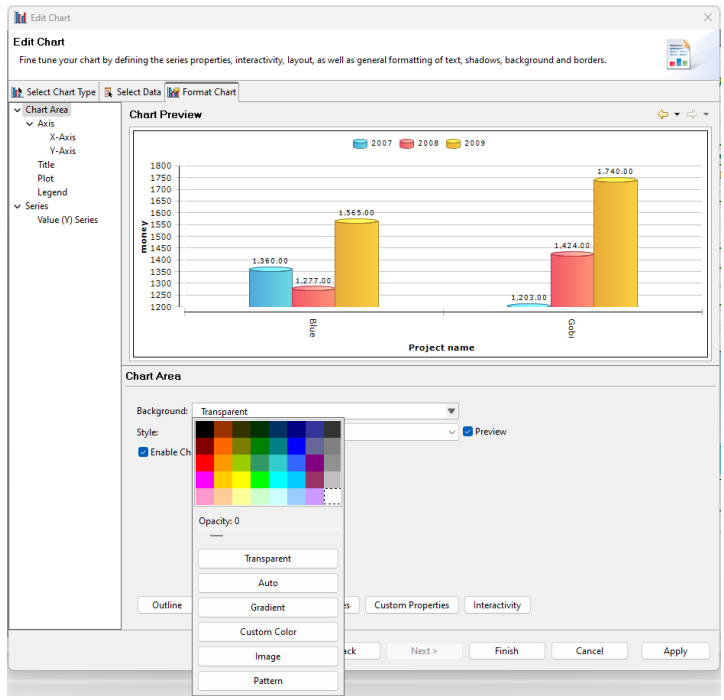



*Figure 7.43. Chart Format: Chart Area.*

- 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”.

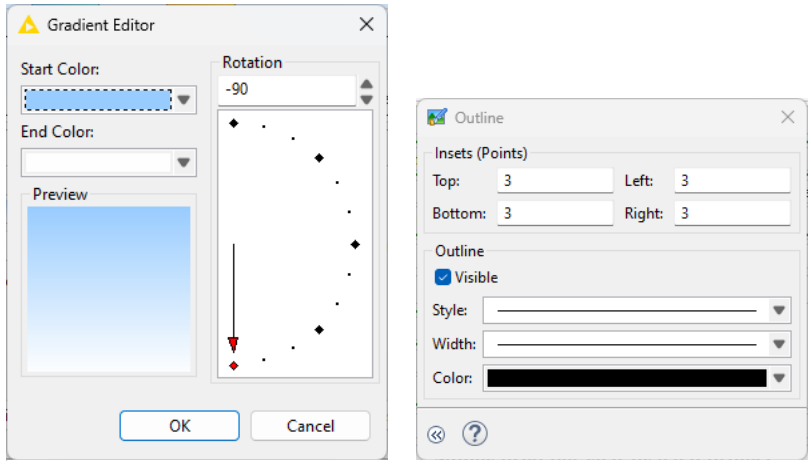



*Figure 7.44. The “Gradient Editor” and the “Outline Editor”.*

## 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

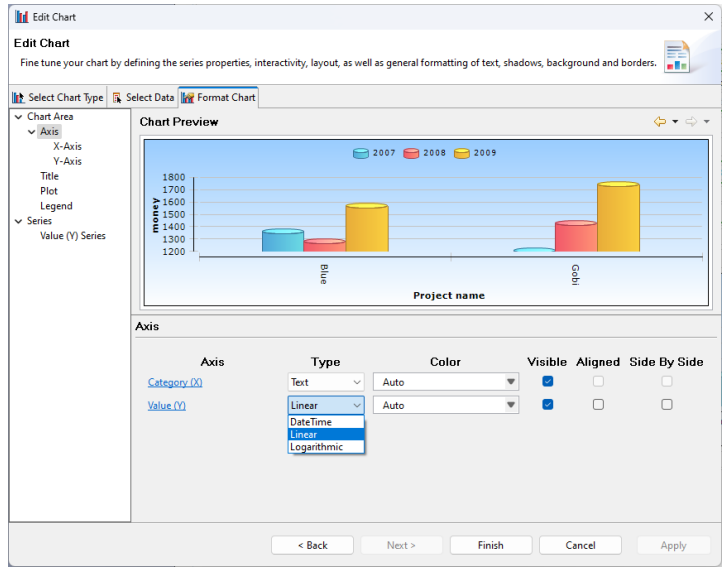



*Figure 7.45. Chart Format: Axis.*

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

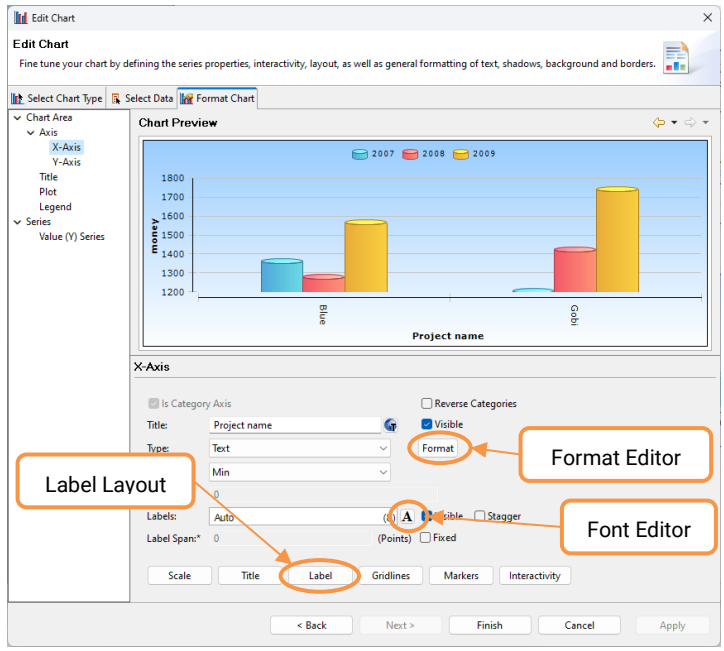



*Figure 7.46. Chart Format: X-Axis.*

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.

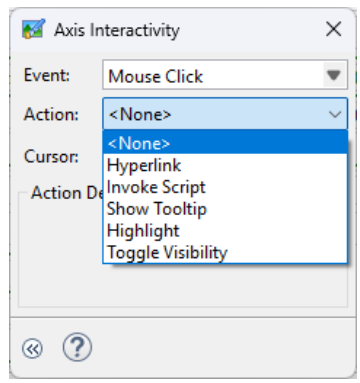



*Figure 7.47. The "Axis Interactivity" window.*

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

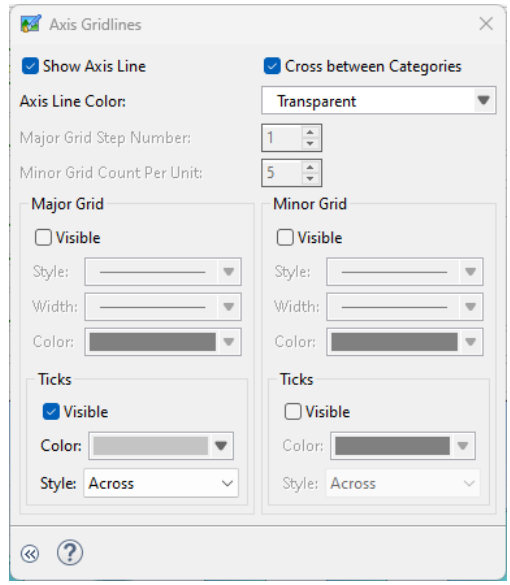



*Figure 7.48. The "Axis Gridline" window.*

## 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.

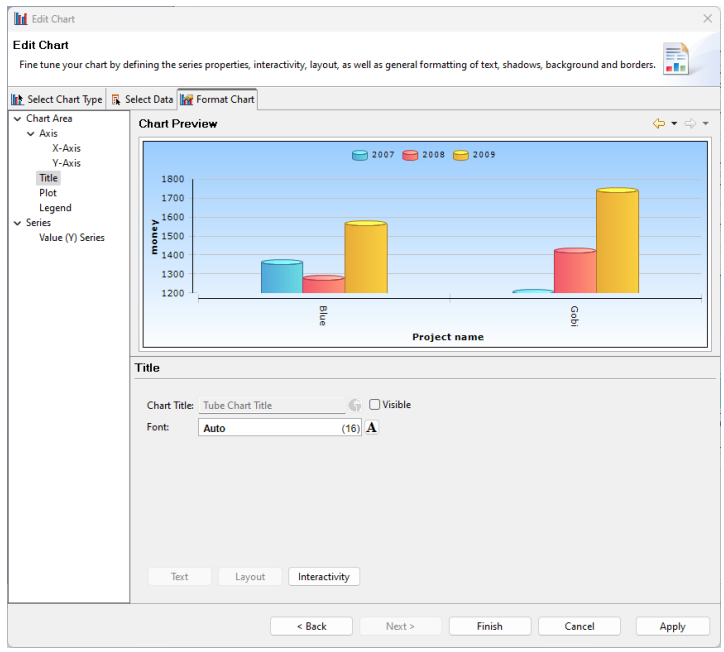



*Figure 7.49. Chart Format: Title.*

## 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).

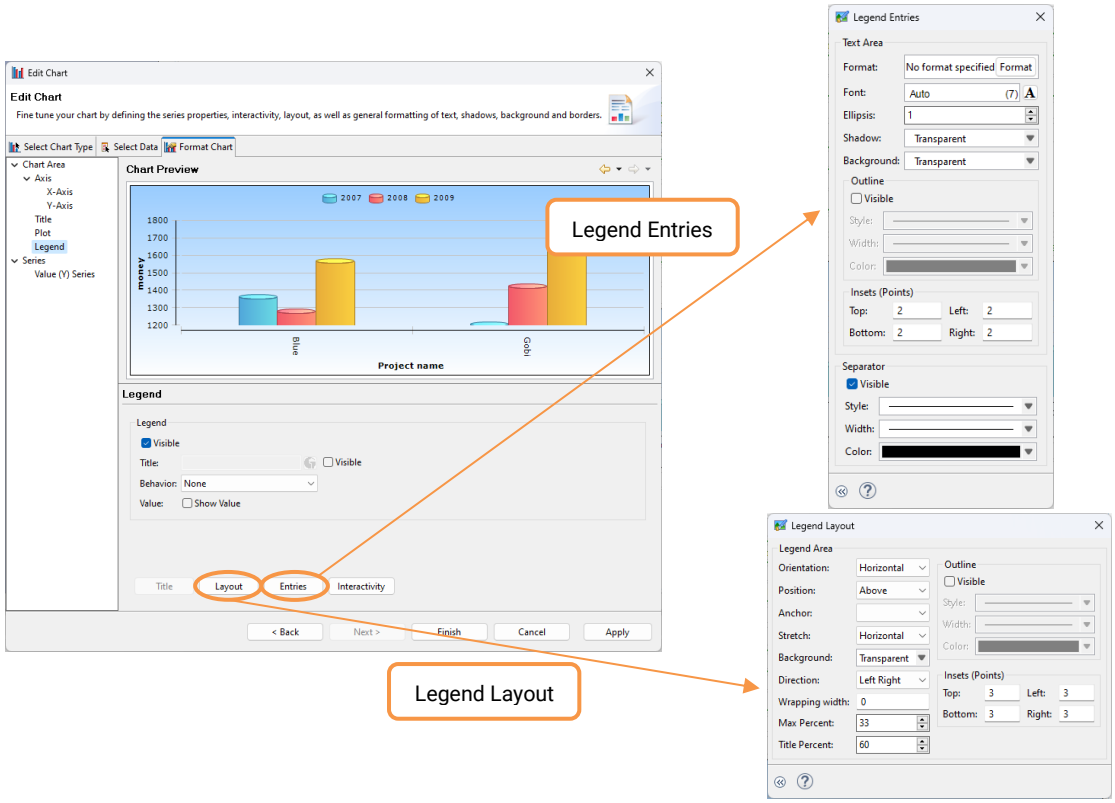



*Figure 7.50. Chart Format: Legend. Button "Layout" leads to the "Legend Layout" window. Button "Entries" leads to the "legend Entries" window.*

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.

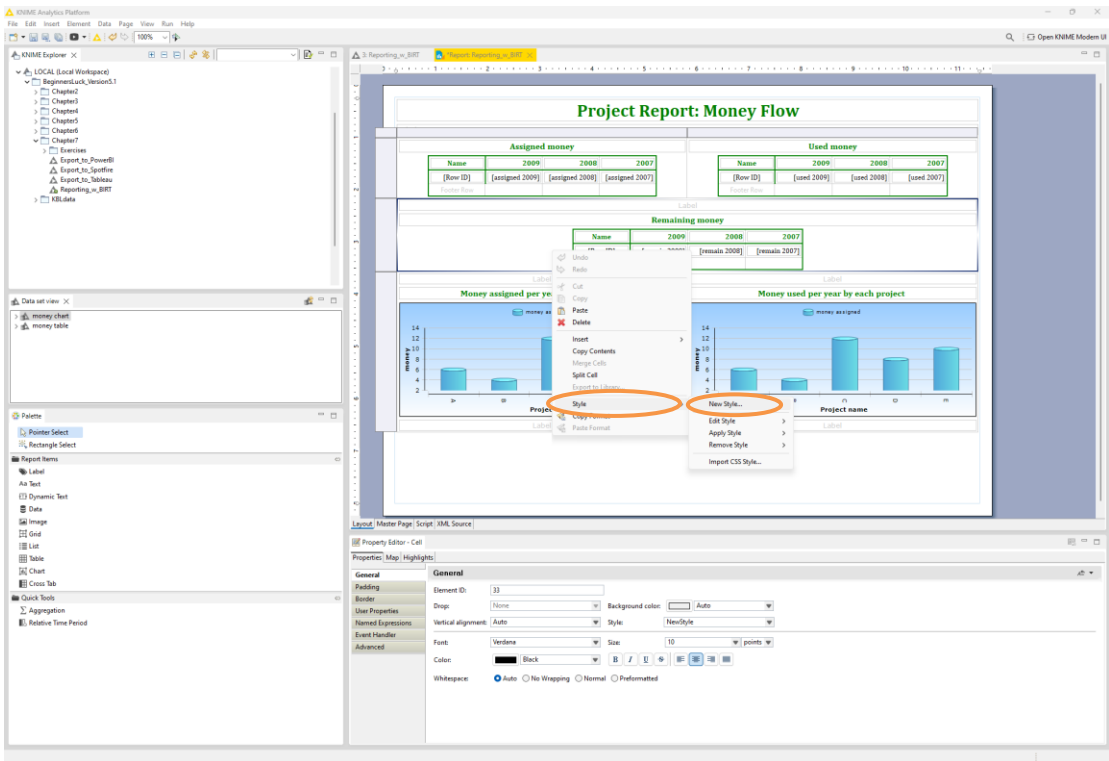



*Figure 7.51. Create a new Style Sheet.*

### 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

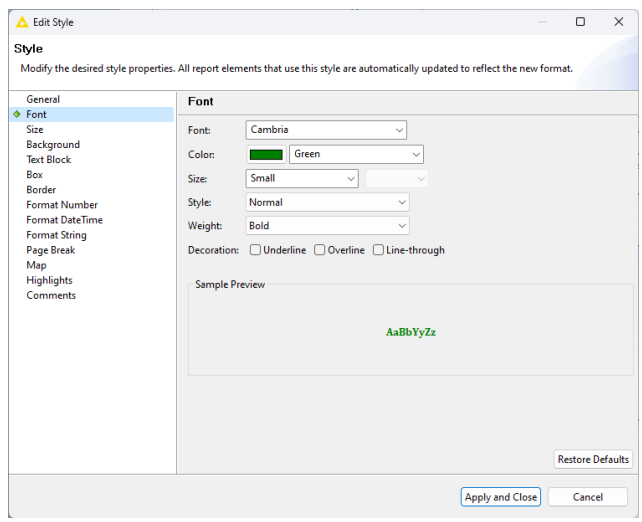



*Figure 7.52. The "Edit Style" window.*

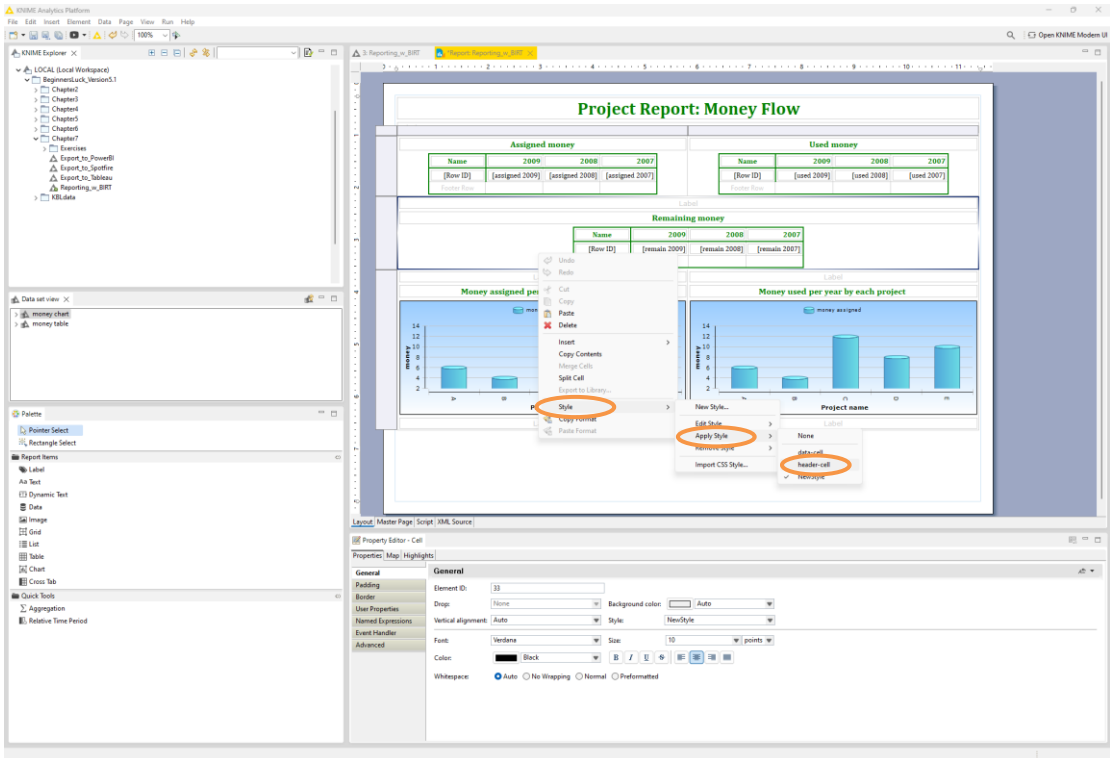



*Figure 7.53. Apply a Style Sheet to a report item, for example, a data cell.*

## 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.

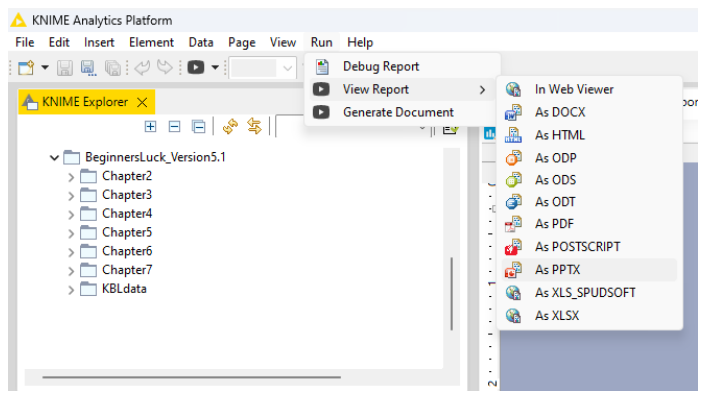



*Figure 7.54. Generate the final document.*

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”.

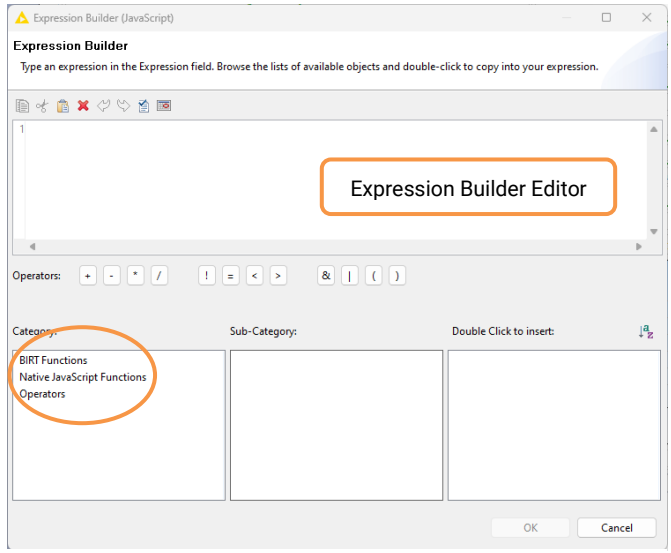



*Figure 7.55. The “Expression Builder” window.*

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.

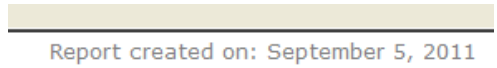



*Figure 7.57. Customized Current Date from "Today()" BIRT Function.*

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

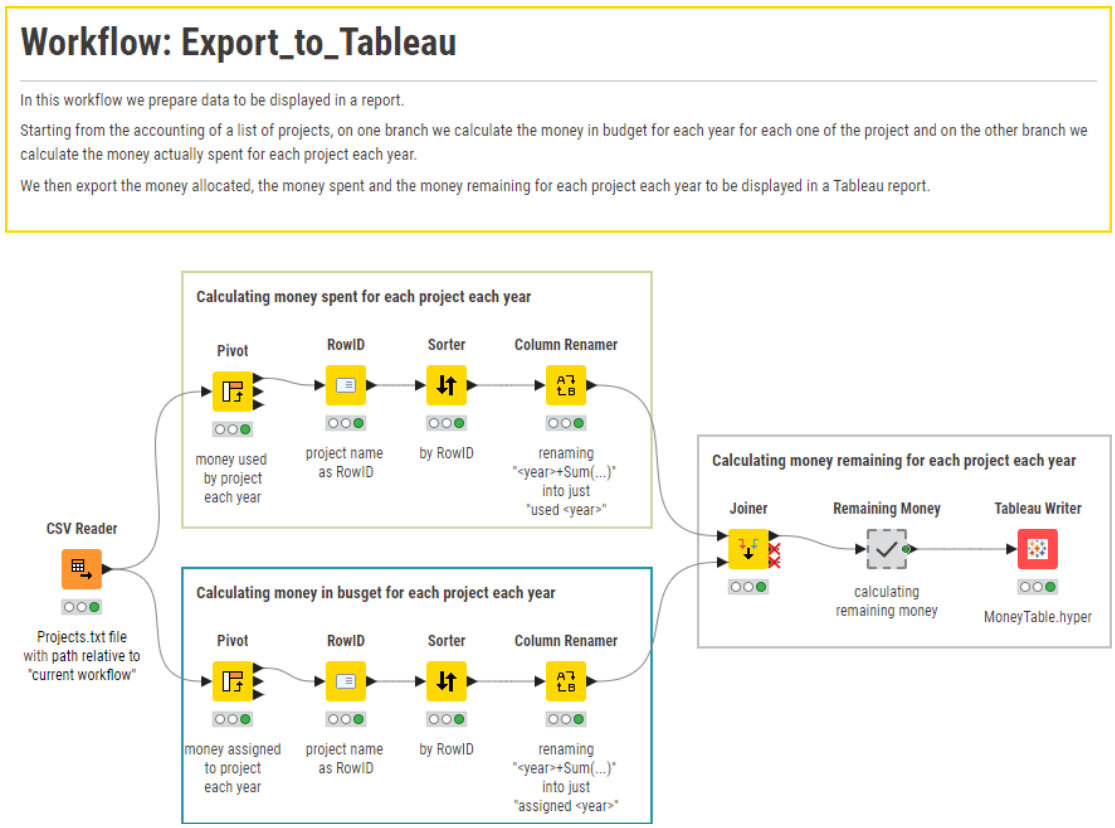



*Figure 7.59. The "Export_to_Tableau" workflow exports the data into a Tableau formatted file to be later imported into the Tableau reporting platform.*

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

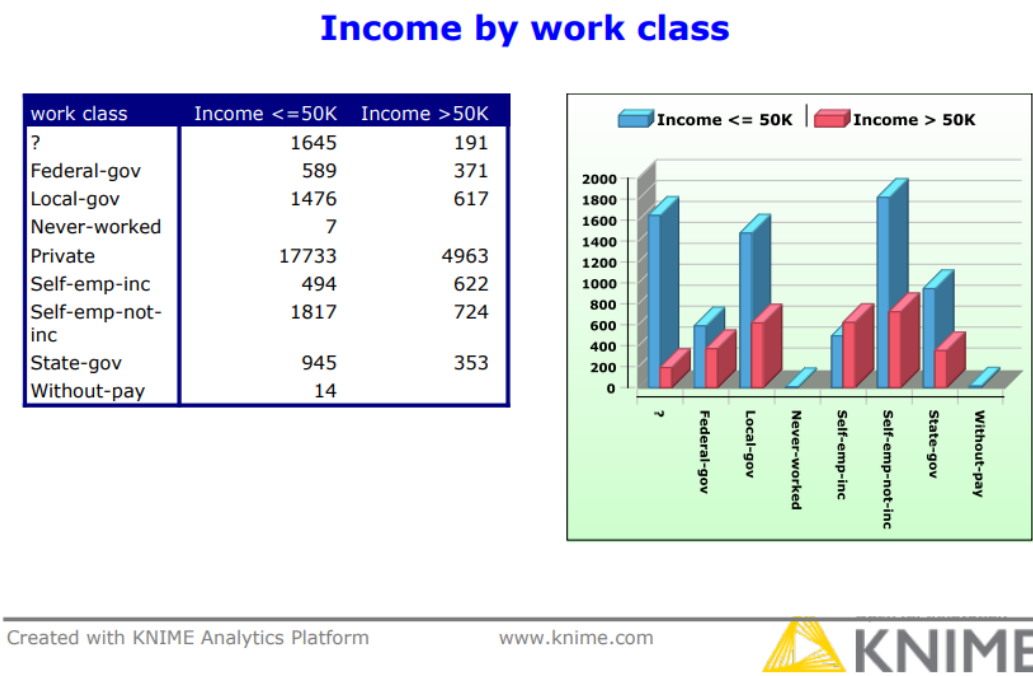



*Figure 7.60. Exercise 1: The Final Report.*

# Node & Topic Index

## E



## F



## G



## H



## I



## J



## K



## L



## M



## N

## P



## R



## S



## T



## U



## V

## W



## Z

# 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.

ISBN: **978-3-9523926-5-2**