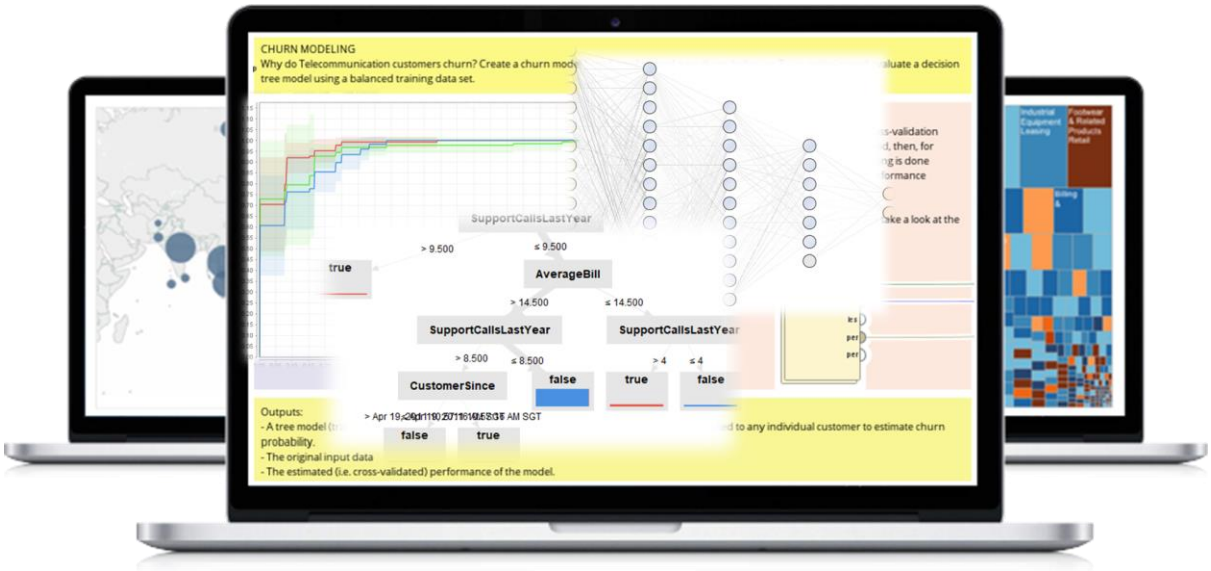
QUANDATICS ACADEMY

Data Science with RapidMiner

- Professional -



rapidminer

Contents

Course Overview

In this course, Data Science with RapidMiner, we will introduce you to some of the important techniques in data science and analytics. We will show you show to better understand your data so that you can create actionable value using RapidMiner.

1. What Will You Learn?

You will learn how to build and run reusable RapidMiner processes that will let you iteratively explore and transform your data sets. You will also learn to construct analytical models to interpret historical data and find patterns that will predict likely future outcomes. At the course conclusion, you will be able to:

- Understand fundamental of data science and analytics
- Recognize the business value of data science and analytics
- Feel confident building basic processes using RapidMiner
- Apply knowledge gained to own business requirements

2. How Will We Teach?

We will introduce you to RapidMiner and Data Science using simple examples that broadly illustrate concepts. Once the framework is established, we will relate the concept to a business case, which we will develop and expand throughout the course. In the objective to achieve the learning outcomes stated above, we will bring you through three main courses that also served as certification exams in RapidMiner:

- Application and Use Cases Professional
- Data Engineering Professional
- Machine Learning Professional

3. The Business Scenario

While the concepts in this course are very interesting from learning perspective, most find that the true value in learning is the ability to apply these concepts into real-world situations. Here is a sample scenario to help you take the step for application.

Throughout this course, we will be returning to the same developing business context to tie the theoretical to the practical:

Customer Churn Prediction

We are working for a company that offers a platform and a mobile app for hotel bookings. The market for such apps is very volatile – users seldom stick to the same apps for more than a few months. Finding and adding new customers is more costly than keeping existing ones. We would like to identify which users are likely to stop using the apps (churn), in order to focus our retention strategy on them.

In the long run, we want to answer the questions: **How can we use data to prevent our customers from discarding our apps?** This business scenario will be used throughout the courses.

PART A

APPLICATIONS & USE CASES PROFESSIONAL

Applications & Use Cases – Professional

This course is primarily conceptual. It is an introduction to RapidMiner that provides a high-level overview to help people understand how to drive value. The course should be appropriate either for people that will use RapidMiner, or those that manage and provide oversight.

1. What is Analytics?

According to author Tom Davenport, in his book *Competing on Analytics: The New Science of Winning*, "Analytics is defined as the extensive use of data, statistical and quantitative analysis, exploratory and predictive models, and fact-based management to drive decisions and actions."

Wikipedia defines it simpler, "Analytics is the discovery and communication of meaningful patterns in data." [1]

Wikipedia adds, "Especially valuable in areas rich with recorded information, analytics relies on the simultaneous application of statistics, computer programming, and operations research to quantify performance."

In describing the difference between analytics and analysis, Wikipedia explains, "Analytics is a multidimensional discipline. There is extensive use of mathematics and statistics, the use of descriptive techniques and predictive models to gain valuable knowledge from data – data analysis. The insights from data are used to recommend action or to guide decision making rooted in business context. Thus, analytics is not so much concerned with individual analyses or analysis steps, but with the entire methodology."

[1] *https://en.wikipedia.org/wiki/Analytics*

By any definition, analytics uses quantitative methods to explore data and reveal patterns within. Useful patterns can be formulated into reusable models. Applied to business, these models are then used to derive insight and prompt data-driven action.

Analytics can be broken out into two topical trunks: **business intelligence (BI)** and **advanced analytics**. Both of these trunks, naturally, branch further into more specific subtopics. Both also share some common data structures, and infrastructures, as the basis for their information delivery. Figure 1.1 demonstrates one way of visualizing this hierarchy.

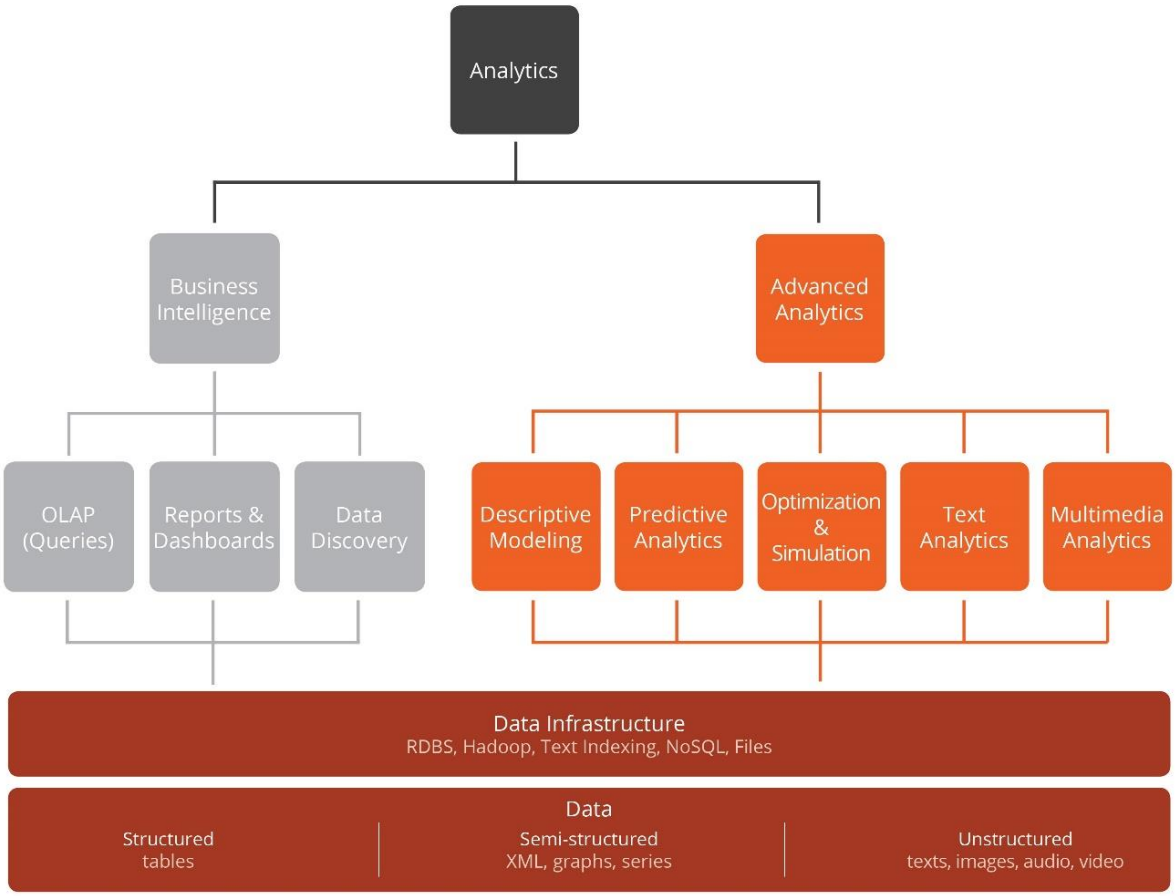


Figure 1.1: Analytics Hierarchy Chart

1.1 Business Intelligence (BI)

In our business scenario, the BI department is using monitoring tools to track the customer base in real time. Using BI, they can see what the user base has already done, or is currently doing, with the product. This information is interpreted by management to make strategic decisions.

However, there are some shortcomings with this approach:

- BI generally tells us where we have been. It can tell us what happened, but is less helpful explaining why. Without that key insight – the ability to go from facts to a pattern – we won't be able to change future outcomes.

- BI tends to give us an aggregated overview. Customers are viewed in mass, and strategic decisions are then applied across all customers; we ask questions referring to the aggregate and then hope to drill down. But our BI doesn't come with the kind of granularity to support action at that level - or only on few, manually searched and selected customers. Here's an example to illustrate that aggregation: Let's say that our BI report says that 20% of a certain group of customers has churned. If we drill down, that does not mean that we can expect that 20% of each customer will leave. Instead, all we can know is that, if the conditions stay the same, for every five customers, one will leave. We have no way to differentiate the four that will stay from the one that will not. Even though our customers behave as individuals, BI leaves us treating them in groups.

- While BI allows us to ask questions of our data, it relies on us to determine what those questions are. If we don't know enough to ask, BI will remain dutifully mute. In other words, to obtain any insight the BI, analyst has to put a great effort. That makes the process expensive and hard to deploy in a production workflow.

- Complexity and scale of BI. Modern data sets can be both deep (having lots of example rows) and wide (containing many attribute columns). Humans are very

good at a lot of things, but discerning meaning from more than a few dimensions at one time is not one of them. Relating this back to our third problem, if we can't understand what we're looking at, how can we know what to ask for?

To maximize our effectiveness – and our ROI (return on investment) – we want to go beyond asking what already happened to groups of customers. We would like, instead, to be able to have better analysis about each customer and what they're going to do. While we're at it, we'd like a little help from our computer in finding out the right questions to ask.

1.2 Advanced Analytics

Advanced analytics is an umbrella term that covers a variety of analytical techniques, including descriptive modelling, predictive analytics, and text and multimedia mining.

In our business case, we can use these techniques to analyse our customers at an individual level. With answers about each customer, we can then roll that information up to get aggregates, but then can always bring it back down to the granularity of a single customer.

We have a lot of data about our customers, including:

- Billing and other business-critical data.
- Social data, such as data collected from social media websites.
- Sensor data collected from the Internet of Things (IoT).

Given the volume and variety of our data, it is very hard for an analyst to manually analyse it. Given that, it makes sense to use advanced analytics to search for useful patterns relating to churn. In other words, RapidMiner will find the patterns for us, allowing us to focus on applying these results to retain customers instead of going over the whole data looking for an elusive pattern.

1.3 Predictive Analytics

In this course, we will focus on the foundations of predictive analytics.

Predictive analytics is an area of statistics that deals with extracting information from data and using it to predict trends and behaviour patterns. The core of predictive analytics relies on capturing relationships between explanatory variables and the predicted variables from past occurrences, and exploiting them to predict the unknown outcome. A central part in this process is defining or developing a model that describes these patterns. The success of the process depends both on the correctness of this model and its assumptions, and on the goodness of the data itself. Luckily, there are statistical methods to prove a model's validity.

Predictive analytics is often defined as predicting at a more detailed level of granularity, i.e., generating predictions for single customers or examples. This is often a big advantage over other analytical techniques.

In our business case, we want to know whether a particular customer will churn in the near future, based on his or her behaviour so far. If we can determine that for each customer, we can take action to prevent it, improving overall customer retention.

For example, we apply predictive analytics to our data, searching for statistical patterns that differentiate customers who churn from those who remain loyal. A simple pattern may reveal: "male customers above age 65 are loyal." Based on this discovered pattern, a male customer of age 72 is likely to be loyal. We can find similar rules to detect churning customers.

In summary:

- We have data about our customers' behaviour, including whether or not they've churned.
- We are going to use predictive analytics to find patterns in that data, allowing us to make accurate churn predictions.

- We can then use those predictions to take actions towards our goal of customer retention.

Of course, this approach is not only applicable to customer data: Whether a customer churns, a machine breaks, a pump leaks, a patent is worth reading, or even whether a caller's voice is angry or friendly, all this and much more can be tackled using advanced analytics.

1.4 Advanced Analytics with RapidMiner

RapidMiner provides a complete software suite for performing advanced analytics. Let's revisit our Analytics Hierarchy Chart. With RapidMiner, you can unite all of these functions, and more, with one tool. You can:

- Connect to existing data, or even generate and capture new data.
- Perform Extract, Transform, and Load (ETL) operations.
- Use traditional BI techniques.
- Create analytical models and apply other advanced analytics processes.
- Generate and export data, reports, and visualizations.
- Design interactive dashboards for technical and non-technical audiences.
- Integrate analytical processes into existing workflows via web services.

In the courses of Data Engineer and Machine Learning, we'll take a look at the RapidMiner platform components that support each of these functions.

2. Data Science in the Enterprise

Before delving into the specifics of how to solve our problem of improving customer retention by detecting churn before it actually happens (and how RapidMiner helps to do that), let's take a step back and see how these kinds of problems should be approached in general.

Different people try different approaches to solving a problem, with varying results. Over time, the most successful approaches become recognized as "best practices". So, it is with data mining. One of the most popular best practices frameworks, and the one we'll use here, is **CRISP-DM**.

2.1 The CRISP-DM Model

CRISP-DM (Cross Industry Standard Process for Data Mining) is the de facto standard for data mining. Rather than being a unidirectional, sequential design process model, CRISP-DM is cyclical, with feedback loops throughout. A diagram outlining the process is shown in the Figure 2.1.

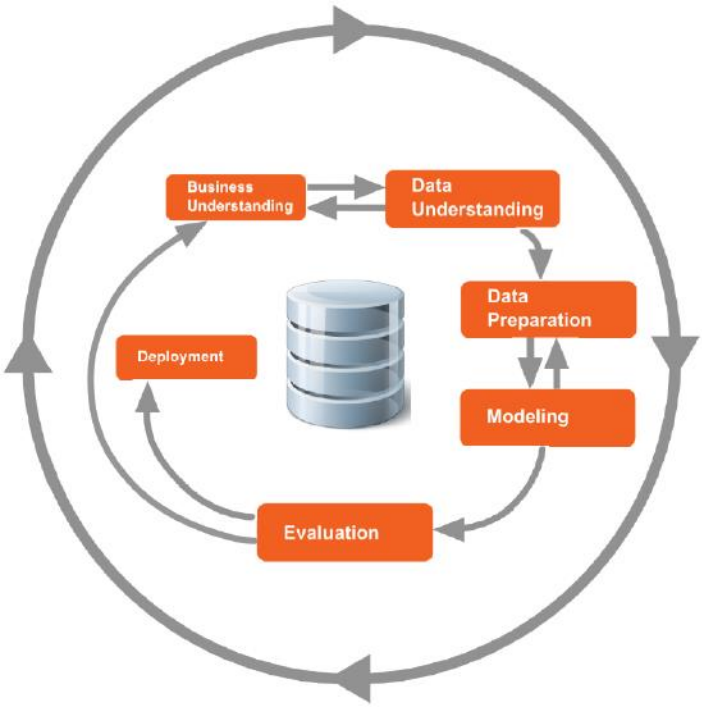


Figure 2.1 The CRISP-DM Model

Many well-managed projects follow the CRISP-DM model, but in a flexible manner. Sometimes they will shorten a path. Often, they will loop back. When basing their work on an earlier project, some of the established phases or builds may be skipped. CRISP-DM can tolerate these variations, allowing them to re-purpose, grow, and modify models and analyses quickly and efficiently.

The following sections describe the CRISP-DM phases.

2.2 Business Understanding

In the business understanding phase, you define the purpose and value of the analysis. Specifically, you want to determine:

- Business objectives
- Data mining goals
- Definition of success criteria

Let’s apply this initial phase to our business scenario:

- Our business sells mobile apps and our objective is to reduce customer churn (improve retention).
- We want to predict who is likely to leave so that we can take proactive steps to encourage them to stay.
- We will gauge our success by increasing customer retention at an acceptable ROI.

2.3 Data Understanding

You need to understand the data that you are using. The data understanding phase includes collecting, describing, exploring, and verifying data quality. Some questions to ask about the data include:

- What attributes about customers did we collect?
- How are those attributes coded? What do each data points mean?
- What is the quality of your data? Is the data accurate? Is the data representative?
- How, where, why, and by whom was the data collected?
- Is the data complete? Did we collect all the data that may be relevant?

Data exploration might reveal unexpected, even surprising, properties, such as:

- Relative importance, and distribution, of key attributes
- Correlations between pairs or small groups of attributes
- Results of simple aggregations (patterns)
- Properties of significant sub-populations
- Outliers

2.4 Data Preparation

Once we have a handle on the data, we need to prepare it for the modelling step. The data preparation phase shapes and transforms data into an appropriate, usable format. This includes: selecting columns, sampling rows, deriving new or compound variables, filtering data and merging data sources.

Some things to consider when preparing data:

- The data representation is key to success. The wrong representation can hide important patterns.
- Different modelling approaches require different data representations or structures. There is no universal data preparation.
- As we learn and/or try new models, we might return to this step for modifications.
- Expect to spend time in this phase – usually more than half, and sometimes even up to 90% of total analysis time is spent in data prep

2.5 Modelling

In the modelling phase we search for patterns in the data. These patterns winnow out unnecessary data and characterize the influence of the attributes that do matter. From these patterns, we will create a model that not only is descriptive, but also predictive.

There are many different kinds of models you can use, each looking at your data from a different perspective. As the *No Free Lunch theorem* describes, there is no single "best" model across all problems. Algorithms perform differently based on the input data and assumptions. Because of this, you may want to try several models, and different parameters within them, to return the best results.

2.6 Evaluation

The Evaluation phase looks in two directions – at the model and at the business objectives:

- We need to validate our models from the prior step. (We will explore model validation later in the course.) Consider:
    - Achieving precision, applicability, and understandability may require trade-offs.
    - Often, an understandable model that provides deeper insights is preferred over a more accurate model that is harder to interpret.
- We also need to evaluate progress towards our business goals:
    - Does this model help meet the success criteria?
    - Does new insight gain here funnel back into our business understanding?
    - Should we loop through the CRISP-DM process again with our new information?

2.7 Revisit Data Preparation

When we perform our evaluation, it may indicate a need to revisit and change the data preparation. For example, we may want to improve the representation and data structures in order to get better results out of our models – this is the iterative nature of CRISP-DM. If we are not careful, we might trick ourselves into seeing patterns that aren't really there. We will explore some of this process during the course:

2.8 Revisit Modelling

By evaluating our model and business objectives, we prove the obvious – there is more than one approach to modelling. This is not a problem though, as we can try different techniques and re-evaluate them to determine which is best suited for the data. There's no way to know with certainty which algorithm will work best, so after visiting the modelling phase we must always circle back to the evaluation phase.

2.9 Deployment

Once we have results that meet our goals, it's time to put them into use. Here lies the beauty of the RapidMiner platform – deployment can happen as quickly as a few minutes: We have all the necessary tools to perform the complete CRISP-DM cycle inside a single platform!

After the deployment phase, we can see the fruits of our efforts: Our predictions will be integrated in daily decision making throughout the company, making the right calls and generating income.

3. RapidMiner Platform

Now that we understand a little of the task at hand and the CRISP-DM model, let's discuss RapidMiner, the tool. We will see how RapidMiner functionality maps into the stages required for successful model deployment.

The RapidMiner platform for deployment consists of two main parts – **RapidMiner Studio** and **RapidMiner AI Hub** (previously known as RapidMiner Server).

RapidMiner Studio is for designing your data processes. It is a graphical integrated development environment for data analysis that each user runs locally on his or her computer.

3.1 RapidMiner Studio

RapidMiner Studio provides tools for all the steps of model building – from data loading, and perhaps even data gathering, to transforming and preparing the data (collectively, ETL), analysing the data, and producing predictions.

However, the RapidMiner Studio software extends beyond the phases of CRISP-DM, however. With it, we can also develop the building blocks for interactive dashboards and produce reports for technical and non-technical audiences. To actually make interactive dashboards and web applications out of such building blocks we need RapidMiner AI Hub – to be discussed in next section.

RapidMiner Studio can connect to various data sources for both reading and writing data, including data file formats like CSV, Microsoft Excel, Microsoft Access, SAS, and SPSS. Analysts can also connect to databases, including MySQL, Microsoft SQL Server, Oracle, PostgreSQL, and others. RapidMiner Studio lets users interact with web services and cloud sources. Using the RapidMiner Radoop extension, it is possible to use Hadoop clusters inside RapidMiner processes, for data retrieval, analysis and predictions.

RapidMiner Studio satisfies a spectrum of needs, from the rigorous requirements of the technical user with a knowledge of advanced analytics to the less experienced data scientist who needs a simple and intuitive environment. The easy-to-understand graphical interface and in-app support tools help even the novice analyst be productive quickly.

3.2 RapidMiner AI Hub (RapidMiner Server)

RapidMiner AI Hub brings collaboration to the RapidMiner platform. RapidMiner AI Hub runs on a dedicated server in the local network or on a remote server connected via the Internet. It seamlessly integrates with RapidMiner Studio and can extend the platform to provide:

- Sharing workflows and data.
- Acting as a central storage point for common configurations (for example, database connections) that can be used by multiple analysts.
- Performing heavy duty calculations that would otherwise take too much time or too many resources from the analyst's local hardware.
- Scheduling regular or recurring tasks, such as data updates from external sources, scoring of new data, and various housekeeping tasks.
- Providing results as interactive dashboards and reports, making them available for different areas inside the company.

It is important to note, that even for processes deployed in RapidMiner AI Hub and Radoop, all the CRISP-DM phases except deployment occur inside RapidMiner Studio. Therefore, it is a very important tool that will be used in almost every step of your data science career.

3.3 Tool For This Course

We will spend most of this course exploring RapidMiner Studio and the foundations of Data Science. After becoming familiar with RapidMiner Studio's user interface, you will load a dataset and start to work with it. Before creating your first model, we'll introduce RapidMiner's visual data exploration capabilities. You will learn to apply several modelling algorithms, evaluating their pros and cons so that you can make an informed choice for your real-world projects after class.

Integrated with the modelling steps, you will learn to apply different methods for data cleansing and pre-processing. This will bring the data into a format that is suitable for the respective algorithms.

You may download RapidMiner Studio through this link:
https://my.rapidminer.com/nexus/account/index.html#downloads

4. Familiarizing RapidMiner Studio

There are two major sections, or **Views**, within RapidMiner Studio: **Design** and **Results**. We will explore each of these briefly and then spend most of our time building things in the Design view and working with output in the Results view.

4.1 The New Process Dialog

When you first open RapidMiner Studio, you start in the *New process* dialog, as shown in Figure 4.1.

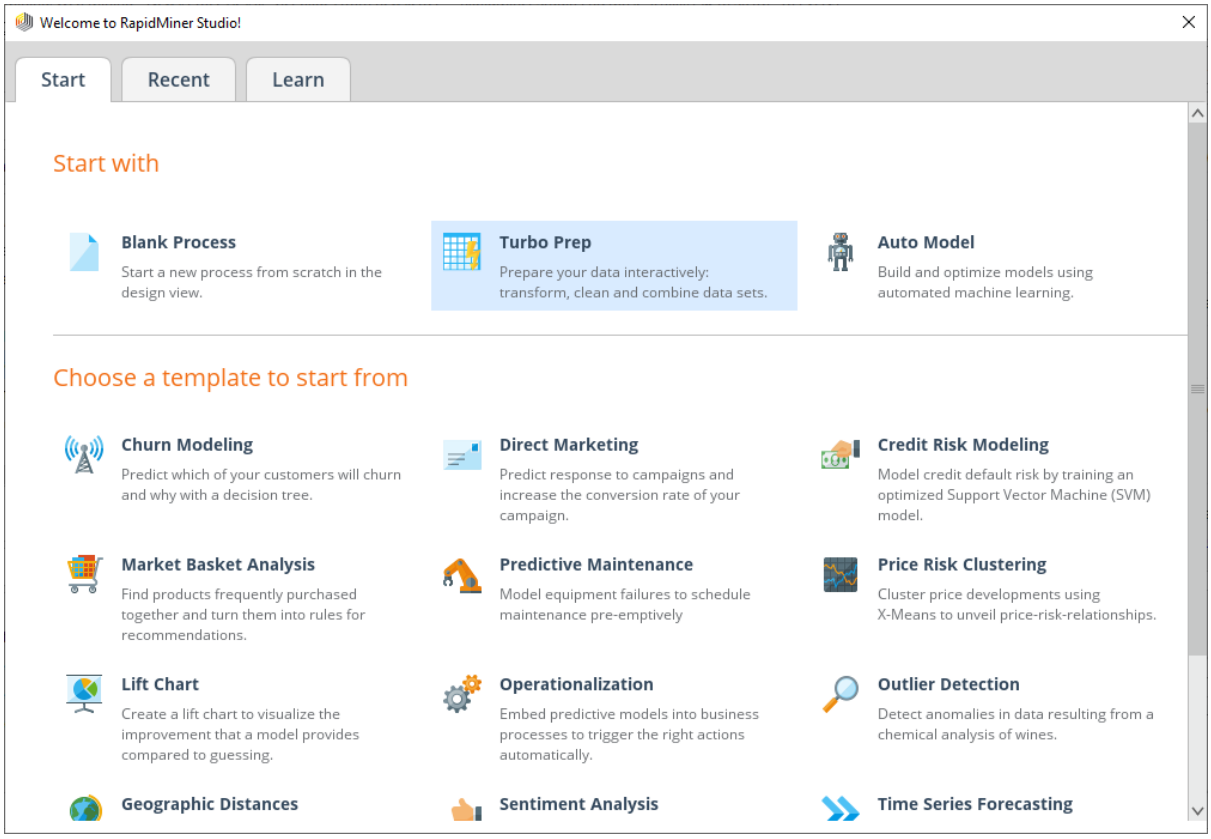


Figure 4.1 New Process Dialog in RapidMiner Studio

From the *New Process* dialog, you can:

- Create a new process – *Start* tab
- Open an existing process – *Recent* tab
- Follow tutorials from online resources – *Learn* tab

When creating a new process, you can start with a blank design, or you can select one of the available templates. The buttons on the middle of the toolbar inside the upper section of RapidMiner Studio let you navigate between the different views, as seen in Figure 4.2.

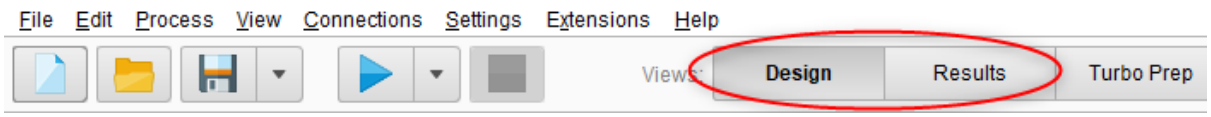


Figure 4.2 Views Selector

There are other views such as Turbo Prep and Auto Model that are designed to make data preparation and modelling easier. Anyhow, this course will be on design and results view which served as a fundamental for RapidMiner Studio. You may refer to the RapidMiner Documentation for more information at the links below:

- Turbo Prep: https://docs.rapidminer.com/latest/studio/guided/turbo-prep/
- Auto Model: https://docs.rapidminer.com/latest/studio/guided/auto-model/

4.2 Online Resources and Tutorials

RapidMiner Studio includes some simple, built-in tutorials to help you get started and walk you through some of the basic functionality. You can access them from the *Learn* tab in *New process* dialog, as shown in Figure 4.3.

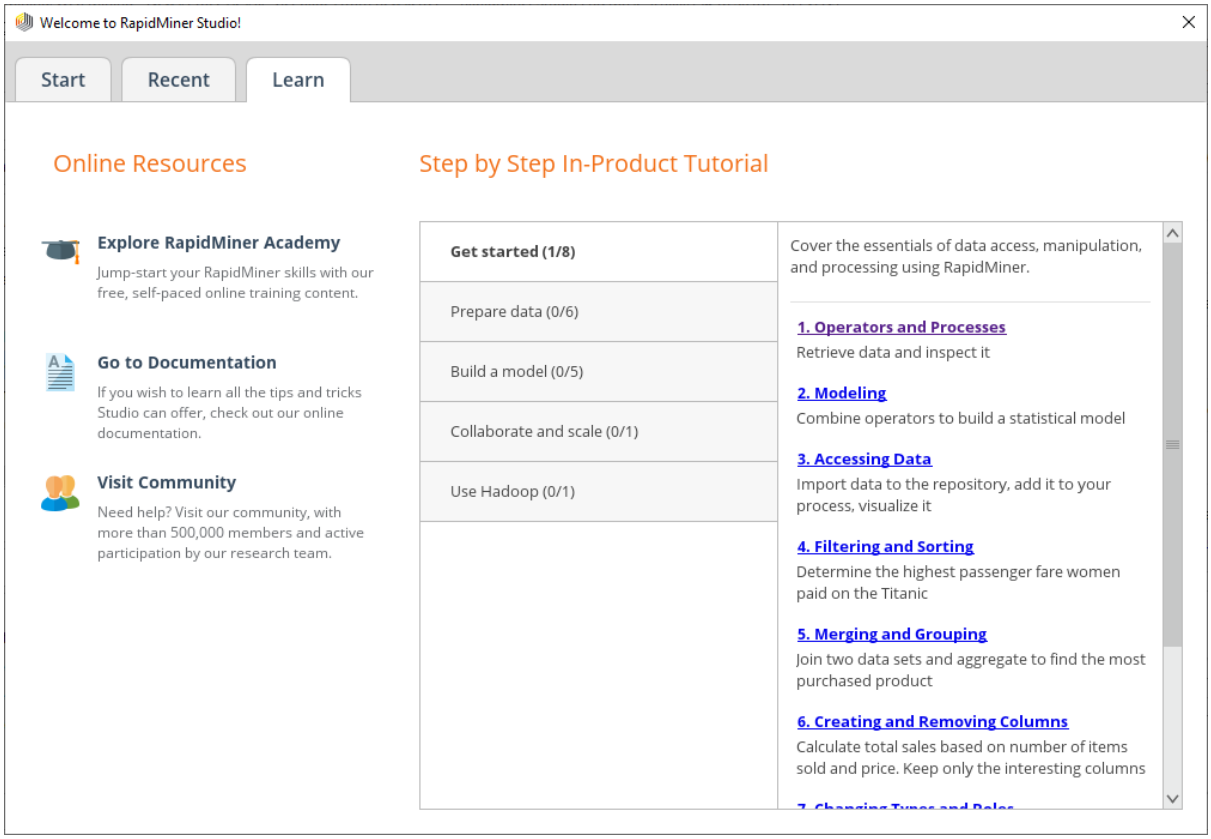


Figure 4.3 *Learn* Tab in *New Process* Dialog

There are also some online resources that could be helpful for your learning:

- **RapidMiner Academy**: Series of online courses and videos
- **Documentation**: A comprehensive documentation of RapidMiner products
- **Community**: A forum for all RapidMiner users for discussion

4.3 The *Design* View

The Design view is where you select the steps that you want RapidMiner to perform – where you design your process. You can navigate to the Design view in a few ways:

- Select the *Design* view in the toolbar
- Select the *New Process* button from the toolbar to start a new process

- Select the *Open Process* button from the toolbar to return to prior work

We will build many processes throughout this course. Figure 4.4 is an example of a RapidMiner process. A process is the chain of actions that you have selected for RapidMiner to perform. You will work with the tools available from the *Design* view to build your processes.

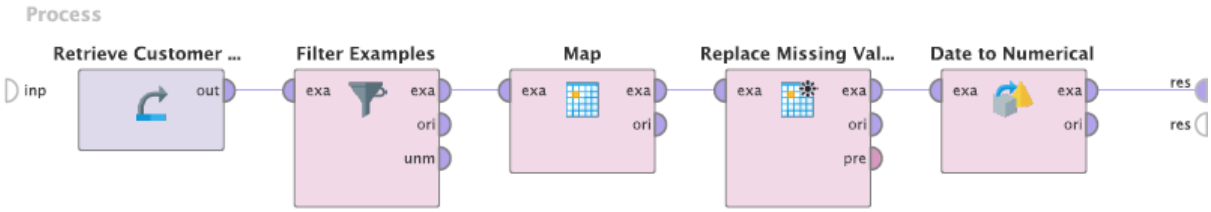


Figure 4.4 A Sample of RapidMiner Process

As a part of your process, you often ask for analytical output as a result. In those cases, after your process runs, you are taken to the *Results* view to show that output. We will return to a discussion of the *Results* view once we have something to see.

4.4 Starting A New Process: Templates

You can start a process with a blank design, or you can use one of the available templates, which give you a jump-start on some common business objectives that can be addressed by data mining and advanced analytics. Some available templates are:

- Churn Modelling – implement customer retention strategies (preventing customers from leaving the company.)
- Direct Marketing – predict response to campaigns in order to increase the conversion rate
- Predictive Maintenance – avoid unplanned maintenance disasters
- Market Basket Analysis – gain insight into customer opinion

Templates can speed your progress towards any of the above objectives and also nicely showcase the capabilities of RapidMiner (and what you will be able to do at course completion).

Let's have a look at the Churn Modelling template, as shown in Figure 4.5.

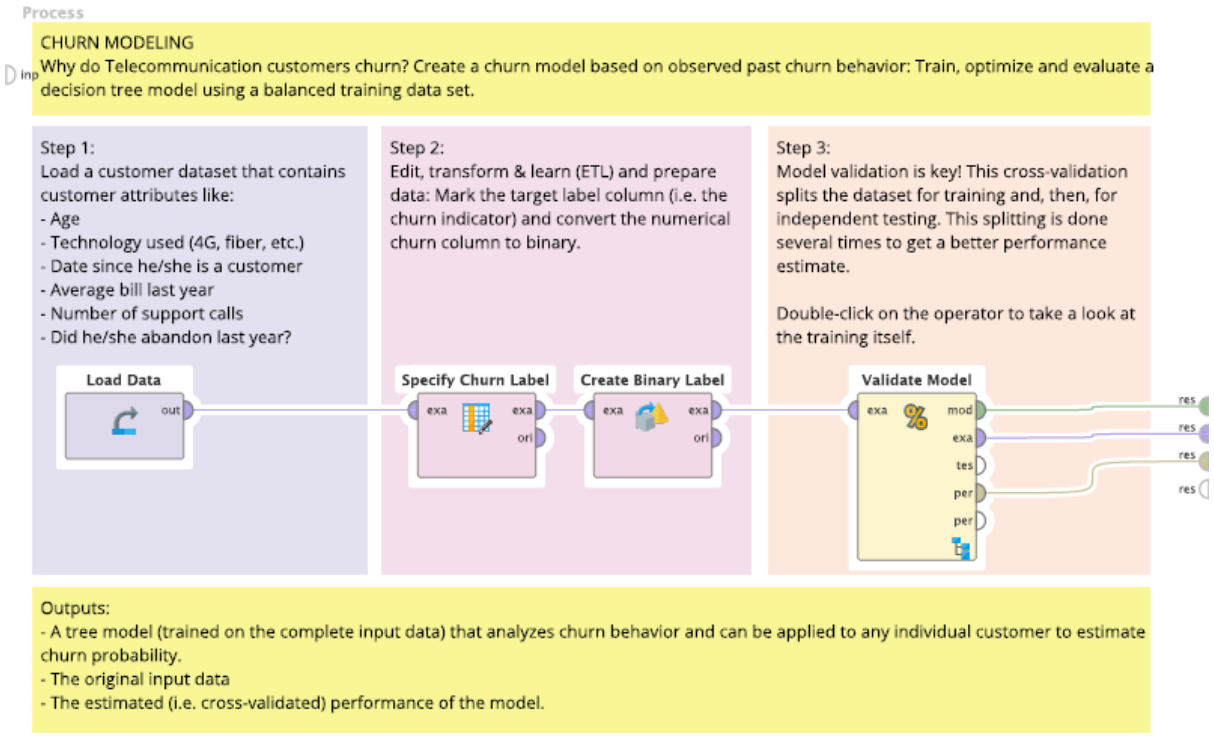


Figure 4.5 The Churn Modelling Template

We can see that a complete process has been created. A process defines the operations and steps that RapidMiner performs to analyse a data set. In our case, the underlying process tells RapidMiner to load the demo data, process it into a state in which it can be used for modelling and finally create a model and estimate its performance. The obtained model can be used for prediction if applied to present costumer data. The comments around each step give insight about the process.

Let's have a closer look: we see a collection of connected boxes. Those boxes, called **Operators**, are the underlying process. They are connected by lines or connections. Let's observe that a process (and thus all the data analysis in RapidMiner) consists of

small steps (operators represented as boxes) that interact with each other via connections, using the result of one operator as the input of another.

We will later go deeper into the Design view, for now let's run the process to see if we can finally see some results. There are two options for running a process:

- Pressing the key F11
- Clicking on the Play button on the toolbar, as shown in Figure 4.6

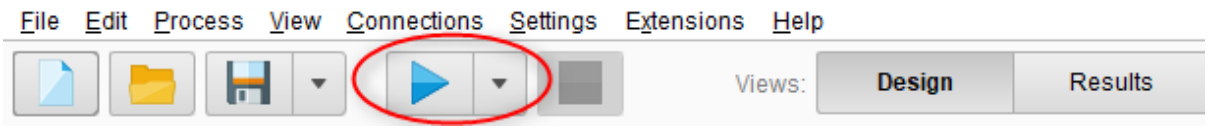


Figure 4.6 Running a Process

After we have run the process, we are automatically taken to the *Results* view, as shown in Figure 4.7. In this case, we obtain three result tabs:

- Performance of the predictive model
- Parameters of the predictive model, a decision tree
- The processed data set which was used for the modelling

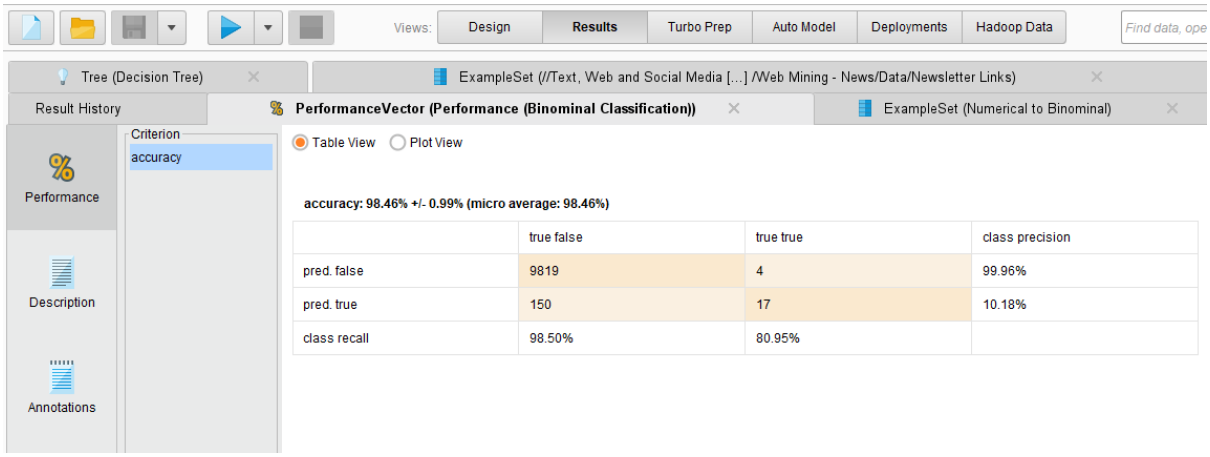


Figure 4.7 Results for the Churn Modelling Process

We will see later those results can be aggregated and represented through charts. For now it is important to recognize how the dynamic of a data mining process with RapidMiner Studio is: We create our process in the *Design* view, including the loading of the data set, pre-processing, modelling and eventually application (prediction); after running the process we get the results in the *Results* view, which can be for example the final parameters for the predictive model, predictions made on unknown values of our data or intermediate results like the data set after the pre-processing step.

Now, instead of becoming overwhelmed by the complexity of the churn process, let's go back one step and start with something simpler. We'll slowly work our way through the analytics landscape until we arrive at creating our own churn prevention models. When all is said and done, you'll be able to solve arbitrary data analysis processes with a single software suite: RapidMiner!

4.5 Our First Process: Working with Loaded Data

The simplest way to start is to build a process that works with data already loaded in RapidMiner. Before switching over to the data for our business case, in the next steps we'll use RapidMiner's sample data to build a generic first process:

1. In the upper left, under the menu items, click the *New Process* button and select a Blank process in the dialog. If prompted to save the current process, answer *No*. The *Design* view opens.

2. Find the *Repository* panel in the upper left portion of the screen.

3. Within this panel, find and expand the Samples folder to reveal the data subfolder.

4. Expand the data folder. In the list of sample data, find *Iris*.

5. Drag-and-drop Iris onto the main process window (white space) in the middle of your screen.

6. On the right side of the Iris box, there is a semi-circle (a port) next to the word out. This is an output port.

7. On the right side of the main process window area, there is a similar semi-circle port icon, labelled res. This is a results port.

8. Click on the output port of *Iris* and click again on the results port to connect them.

9. On the toolbar at the top of the screen, there is the Play button. Click it.

Once the process has finished (which should happen almost immediately), RapidMiner opens the *Results* view. Congratulations! You have just built and run your first RapidMiner process. It may not seem like you did much, but in fact there is now a lot of information available. Let's take a deeper look at the *Results* view.

4.6 The *Results* View

When you ran your process, RapidMiner delivered the results in the *Results* view. You can also visit this view by clicking on the icon in the middle of the toolbar or by hitting F9 key in the keyboard.

4.6.1 The Data Tab: View the Data

When you connected the output port of the Retrieve operator to the results port and clicked the Play button, you asked the process to deliver that data to you. It did just that, producing an Example Set in the data tab.

Within the data tab, as you might expect, you can sort the data columns by clicking a column header. You can also resize the column widths by dragging a column's right margin. You can filter rows (by "missing" or "not missing" attributes or labels) using the drop-down in the upper right corner. By default, the results display "all."

Just a note: sometimes your data will have missing values, indicated by question marks. Good news though – we don't have any missing data in this set. Actually, we are working with one of the most well-known data sets in all of Data Science, the Iris data set. Iris is small, simple, and clean. It is usually among the first data sets that a new, aspiring Data Scientist encounters. Exploring Iris data is the "Hello World!" of Data Science. Welcome to the Data Scientist's Club!

4.6.2 The Statistics Tab: Summary of Data

A click on the Statistics tab to the left displays some summary data about Iris. Note that different types of data (in this example, real and nominal) provide different kinds of summary data. We will explore this topic more when we look at our business case data set, which uses a wider array of data types.

Think about how long it might take you in another tool to set up this variety of summary calculations, one by one, broken out by data type. In RapidMiner, all you have to do is press Play.

4.6.3 The Visualizations Tab: Visual Data Inspection

Getting calculated descriptions of our data is helpful. Sometimes, though, a visual representation is more helpful – and provides even more insight.

There are two ways to open the Visualizations tab:

- From the Statistics tab, click on any row of data attributes to show more detail about the summary data, displayed in a distribution chart. Under each chart is an Open Chart link. (To hide the chart, you can click the row again.)
- On the left side, simply click the Visualizations tab.

Either of these methods opens the Visualizations section for your Example Set, as shown in Figure 4.8.

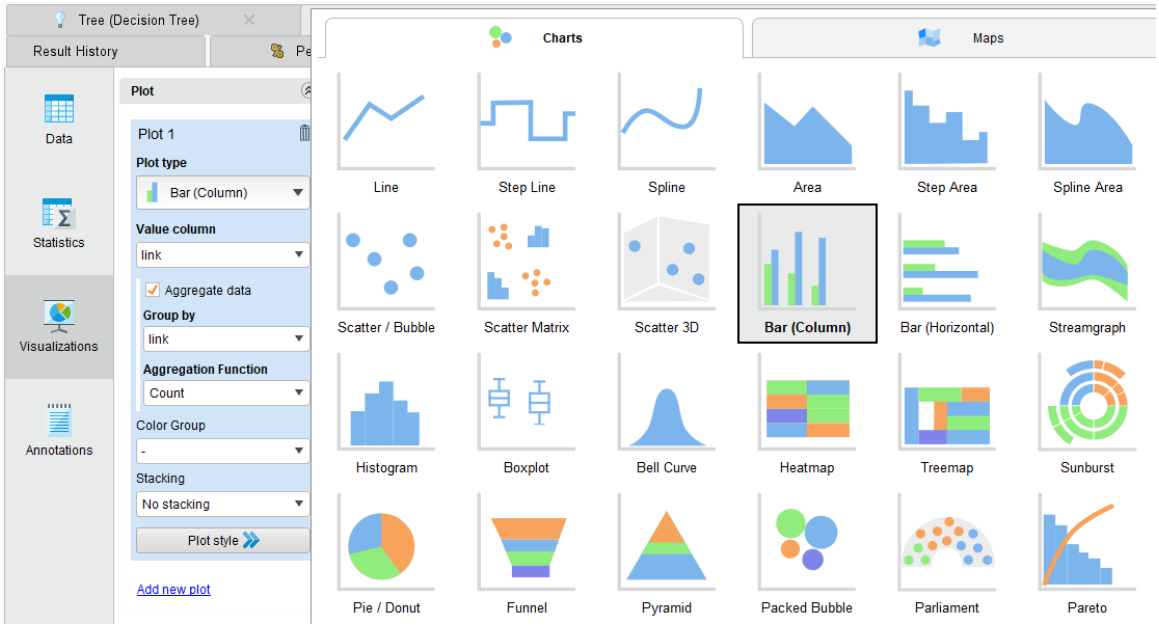


Figure 4.8 Visualizations Options in RapidMiner

Let's take a brief look at the Visualizations tab now. We will explore it more when we work with the business case. Click the dropdown menu of the Plot type, the page displays a screen full of built-in charts, including the most common – Scatter, Histogram, Parallel, Quartile, and others. Click any of the chart types to display the data in that chart's format. Click back on the dropdown menu in the upper left to see the full selection of types again.

4.7 Revisit: The *Design* View

Now that you have finished designing your first process with Iris, and before we move on to our business case, let's take a look at what you just did using the language of RapidMiner to identify the steps:

1. Click on the Design button (or press F8) to go back to the Design view.

2. Find the Iris data set in the Repository panel (more about repositories in a bit) and drag it to the Process panel.

3. Connect the Iris output port to the results port.

4. Finally, hit Play.

Next, let's talk about **panels** in *Design* view.

Panels are the working areas within RapidMiner Studio. In the Iris example you used the Repository and Process panels. Later, we'll explore other panels – Operators, Parameters, and Help. Figure 4.9 illustrates the panels in the default *Design* view.

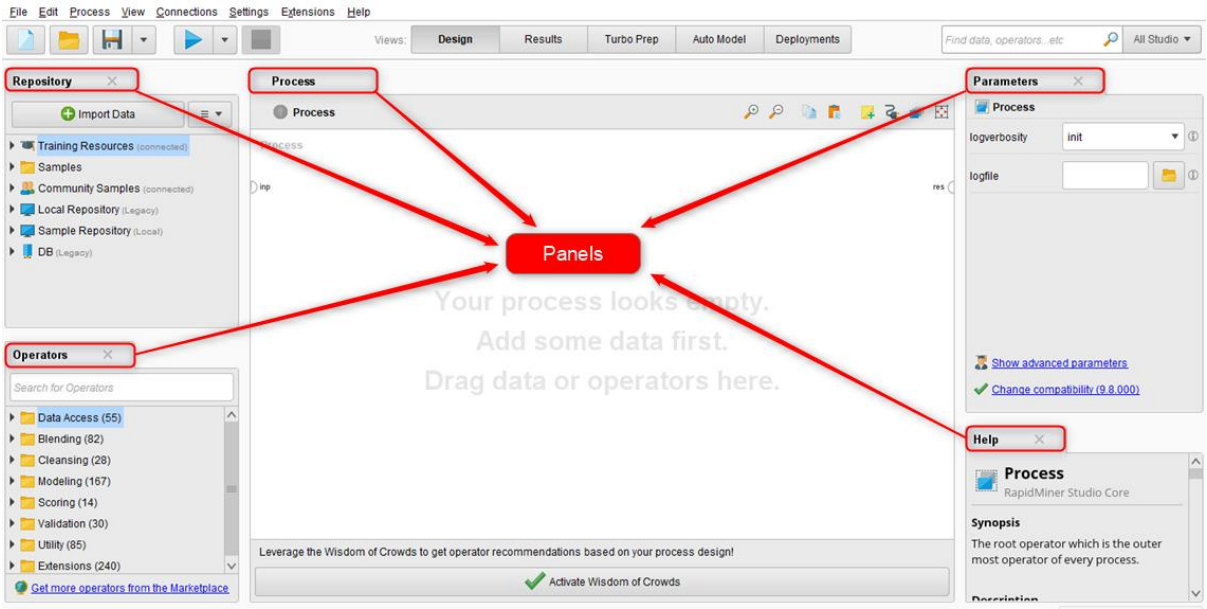


Figure 4.9 Panels in RapidMiner

Panels can be closed, maximized, minimized, pinned, and detached. All these options are available in each panel header's right-click menu.

Each panel provides a window into a different part of RapidMiner Studio functionality. Since you will be building many processes, you can expect to spend a lot of time in the Process panel. There are more panels than those visible in the default display.

These panels can be toggled on and off from Show Panel in the View pull-down menu, as shown in Figure 4.10.

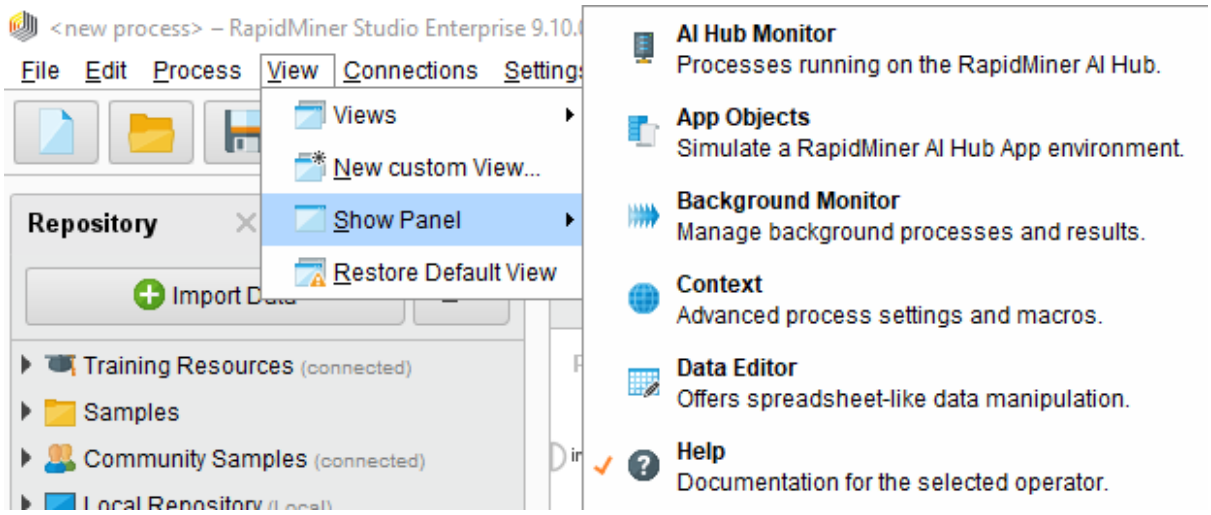


Figure 4.10 Panels in RapidMiner Studio

It's configured that way so that if you need a particular functionality you can turn on that panel, but if you don't, you can save the screen space by hiding it. Later in this course, you will learn about a handful of useful panels that are not visible by default.

Panels can also be repositioned within the *Design* view by dragging them to other parts of the screen. Click in the panel header tab and as you drag, an outline displays of the panel's new position if you were to release your mouse button. This allows you to align the panel where you like before you drop it in its new place.

Note that if you make a mess of things, you can always return to the original configuration by selecting *Restore Default View* in the View menu.

5. Hands-on: Repository

You should now be familiar with the core elements and concepts of the RapidMiner user interface. You have created your first process and explored some sample data. Now it's time for something real-time to have a look at our business case.

When you first investigated the Repository panel, you saw that it contained folders and subfolders. Inside the Samples folder, for example, were the subfolders "data" and "processes", among others. A RapidMiner repository is a place to store things like data and processes. Because some are confused by the difference between data and processes, to avoid any confusion going forward, let's clarify:

- **Data** means the raw material, the source for our analysis.
- **Processes** are the things that we do to and with that material. Sometimes the output of a process is more data, which we can review or store for later use.

As a simple way to think of it, data are the groceries and processes are the recipes.

To keep your work organized, you can create multiple repositories. In our example, Samples was our Repository and data and processes were folders used to help organize our work within Samples. In the real world you will often create a top level of folders within a repository, one folder per project. You will then make data and processes subfolders within each project. This is a common organizational structure and the one we'll use here in class.

Note – you can have both local repositories (for use with RapidMiner Studio) and RapidMiner AI Hub repositories. Here's the difference:

- Local Repository:
    - Lives on the local hard disk
    - Is only accessible by a single user

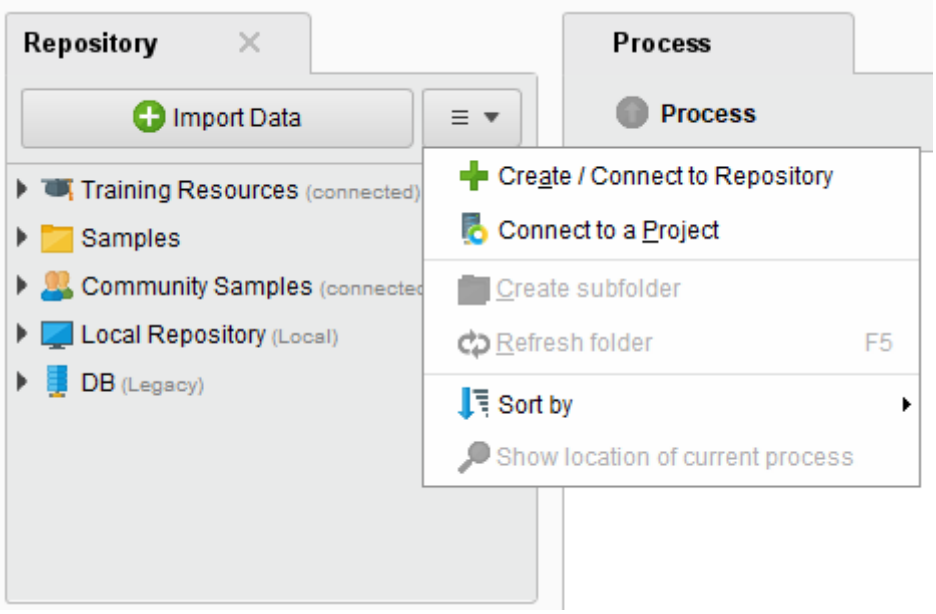
- Can be copied and shared (which can become problematic to track in production)

- RapidMiner AI Hub Repository:
  - Stored on a RapidMiner AI Hub
  - Accessible by many users (Of course, any server has access rights management to allow working privately, with specified users, or with a team.)

Exercise 1. Creating a Repository

As we work our way through the course, you will be using specific data and building many processes to work with that data.

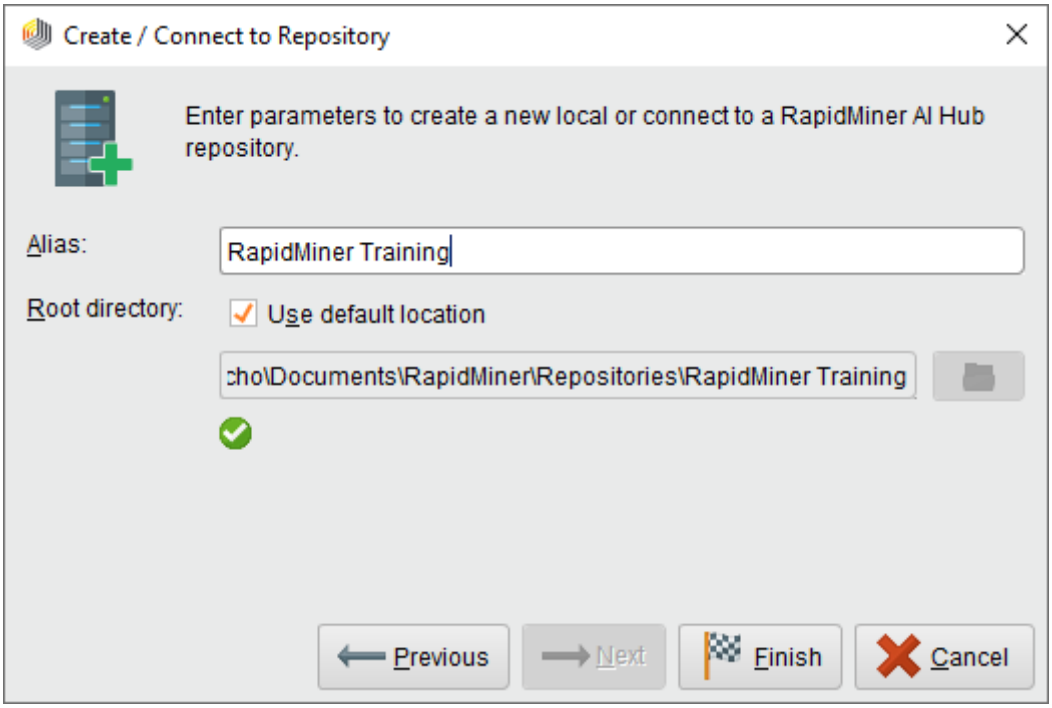
To start, go to the Repository panel, click the drop-down menu on the right and select Create repository – a plus sign icon – to add a new repository.



A new window opens, select *New local repository* and click Next.

In the New Repository window that opens:

- Choose an alias to name the new repository. We recommend *RapidMiner Training* for this class.
- Specify a Root Directory that will contain the location of the created folder
- Click Finish

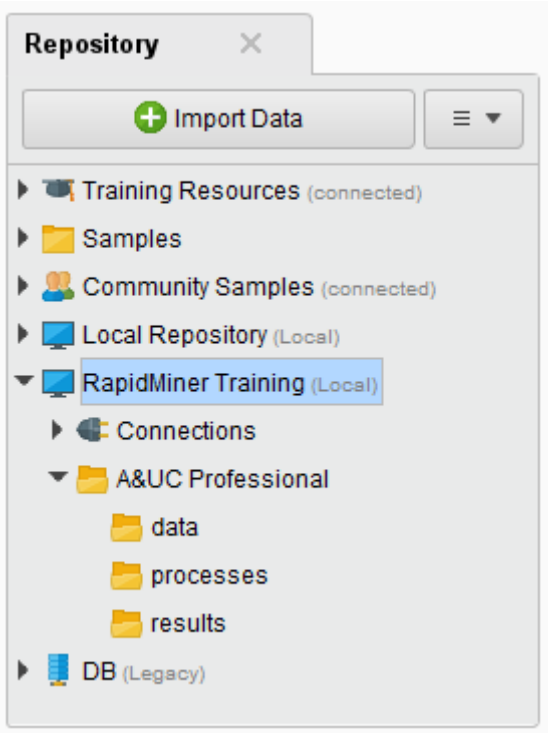


The example uses the alias Training and accepts the default root directory. You can do the same or create your own alias. If you do choose a different alias or location, keep that in mind for any class instructions that refer to the choices here (like the instructions below for creating subfolders).

Exercise 2. Creating Repository Subfolders

Now create the folder hierarchy within your Training repository (you will probably do the same for your own projects). Since this use case may not be the only training that we do, create a project-specific folder for it.

1. Right-click on the new Training repository. A context menu appears.
2. Select Create Subfolder. A New Folder dialog appears.
3. Enter a name for the new folder. We'll call it *A&UC Professional*.
4. Using the same technique, create another 3 subfolders for the *A&UC Professional* folder. That is, right-click the *A&UC Professional* folder, select Create Subfolder and name the new folders: data, processes and results.



PART B

DATA ENGINEERING PROFESSIONAL

Data Engineering – Professional

This course is designed to be in-depth hands-on training in RapidMiner. The course is appropriate for anyone that will be using RapidMiner and need to be able to perform data preparation at an intermediate level.

Assuming that you've gone through the Applications and Use Cases Professional course, under the RapidMiner Training repository that you've created, open another folder named DE Professional. Open 3 subfolders named data, processes and results under DE Professional folder for the use for this course.

Now that you've seen the basics of the RapidMiner interface, let's get our hands dirty with some real data. Referring to the Business Scenario of Customer Churn Prediction, we run a company that sells mobile apps and we want to retain existing customers. Using the terminology of the CRISP-DM cycle, this is Business Understanding phase.

1. The Data We Will Use

Now let's see the data that IT provided so that we can proceed to the next step in the CRISP-DM cycle – Data Understanding. In the data folder supplied with this document, there is a file named Customer Churn Data.xlsx. Open it with your favourite spreadsheet application (for example, Excel or OpenOffice).

PostalCode	HashCode	Age	Gender	Payment Method	LastTransaction	ChurnDate
49278	BOI2gvcX	64	male	credit card	2012-04-17 02:05:40	2014-01-24 18:27:13
39982	IJC8cDTW	35	male	cheque	2011-11-25 06:58:03	2012-08-09 13:01:39
87213	tKlbadnh	25	female	credit card	2012-02-15 17:29:26	
38548	RcW2Pb3w	39	female	credit card	2010-10-09 11:22:28	2013-11-07 10:27:31
38794	z9twA4AJ	39	male	credit card	2012-06-13 10:13:08	
44573	akWNQJ4e	28	female	cheque	2010-07-16 09:39:10	2011-06-23 07:08:53
70936	glrPDLzY	21	female	credit card	2012-03-15 22:17:03	
71302	Pn6FkbuL	48	male	credit card	2011-06-16 21:46:18	
49705	3rGPBX98	70	female	credit card	2011-03-30 14:17:44	2012-07-05 02:34:33

Figure 1.1 Customer Churn Data.xlsx

Figure 1.1 shows the first rows of this file. The Customer Churn Data.xlsx file has 8 columns and 1000 lines of data. Each row contains information about a single customer. There is an Age, a Gender, and a PostalCode column. These three columns are called base data – they contain basic, mainly unchangeable information on each customer. The rest of the data contains information about the buying behaviour of the customers and other information. We will talk about all columns in the next sections.

While we want to enable you to work on real data, we also want to keep things simple in this introductory course. This data set is made up and a bit simplified compared to real-world data. We think that learning the basic concepts on clean and friendly data, without all the quirks that the big bad world adds, is a good approach. Be prepared though, because in the next course things get much more complicated (and realistic)!

Now it's time to begin with what the data mining scene calls **ETL – Extract, Transform and Load**. That is, you will extract the data from the Excel file, transform and prepare it for the tasks that you want to conduct, and then load (or actually store) it into RapidMiner's repository.

2. Getting Data into RapidMiner

One good thing about Excel data is that it is a common format that we all understand and can easily open and view. But we want more than to look at the data – we want to analyse it with RapidMiner.

So first you must get it into RapidMiner. (Lucky for us, because it is RapidMiner, this is a very easy task). Click the New Process icon in the toolbar and select a new blank process.

Next, go to the Operators panel and search for the operator Read Excel. You can do it very easily by typing into the search field the first two letters of each word in the operators name, as can be seen in Figure 2.1.

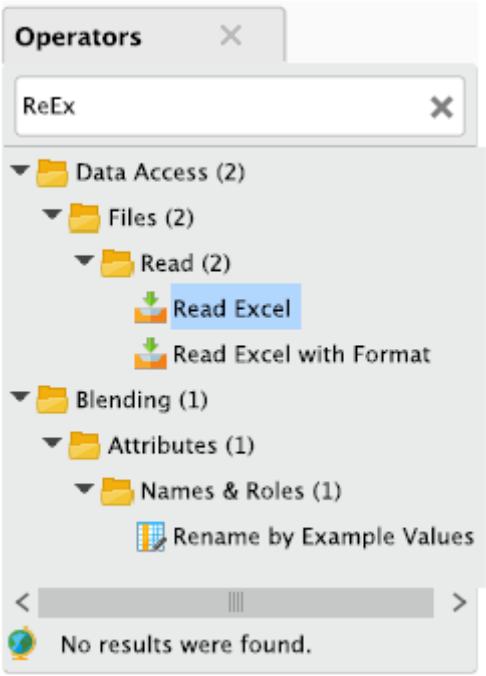


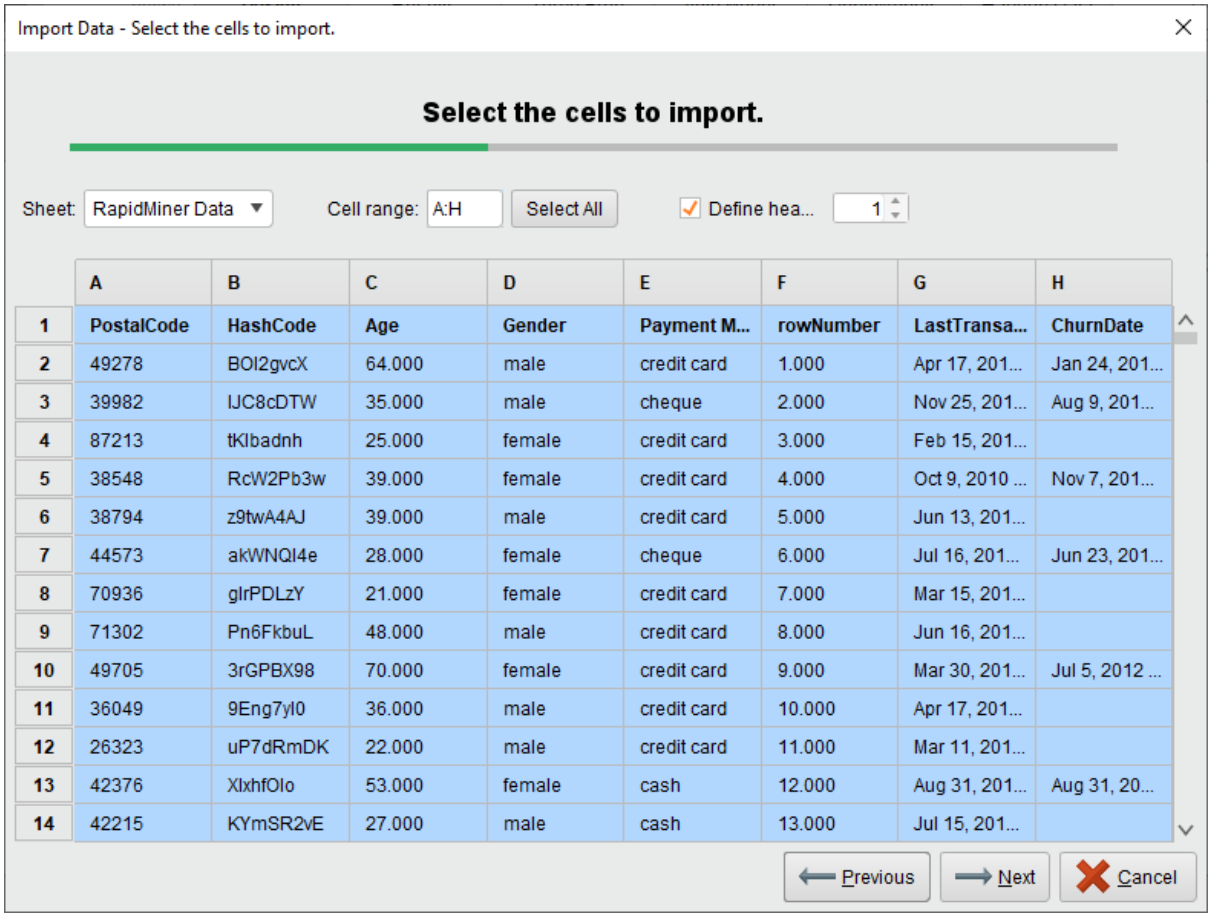
Figure 2.1 Searching for *Read Excel* Operator

Once found, drag and drop it into the Process panel. You should now see a box corresponding to the Read Excel operator. When you execute the process, this operator loads the data from the Excel file and delivers it as a RapidMiner data table at its output port. However, we must configure it first to point it to our Excel file.

When you click the Read Excel operator, the Parameters panel on the right displays the operator's configuration options. In Parameters, click the Import Configuration Wizard button to open Read Excel's configuration wizard. The wizard will guide you through the loading process in four steps.

2.1 Selecting the Data

In the first step of the wizard, select the file that you want to read by browsing into Customer Churn Data.xlsx's location. Click Next to proceed to next step where you can select the cells to import. The complete Excel sheet is read by default, which is fine in this example, but you can also mouse around to select a range of cells. If you do select a range and then decide to read all data, simply press Ctrl+A to select all cells. See Figure 2.2 for reference.



	A	B	C	D	E	F	G	H
1	PostalCode	HashCode	Age	Gender	Payment M...	rowNumber	LastTransa...	ChurnDate
2	49278	BOl2gvcX	64.000	male	credit card	1.000	Apr 17, 201...	Jan 24, 201...
3	39982	IJC8cDTW	35.000	male	cheque	2.000	Nov 25, 201...	Aug 9, 201...
4	87213	tKIbadnh	25.000	female	credit card	3.000	Feb 15, 201...	
5	38548	RcW2Pb3w	39.000	female	credit card	4.000	Oct 9, 2010 ...	Nov 7, 201...
6	38794	z9twA4AJ	39.000	male	credit card	5.000	Jun 13, 201...	
7	44573	akWNQl4e	28.000	female	cheque	6.000	Jul 16, 201...	Jun 23, 201...
8	70936	glrPDLzY	21.000	female	credit card	7.000	Mar 15, 201...	
9	71302	Pn6FkbuL	48.000	male	credit card	8.000	Jun 16, 201...	
10	49705	3rGPBX98	70.000	female	credit card	9.000	Mar 30, 201...	Jul 5, 2012 ...
11	36049	9Eng7yl0	36.000	male	credit card	10.000	Apr 17, 201...	
12	26323	uP7dRmDK	22.000	male	credit card	11.000	Mar 11, 201...	
13	42376	XlxhfOlo	53.000	female	cash	12.000	Aug 31, 201...	Aug 31, 20...
14	42215	KYmSR2vE	27.000	male	cash	13.000	Jul 15, 201...	

Figure 2.2 Selecting the Data

Proceed to next step by clicking the Next button again.

2.2 Format the Columns

In this step, seen in Figure 2.3, is by far the most important. It comprises 4 tasks:

- defining data types
- defining attribute roles
- defining column names
- selecting imported columns

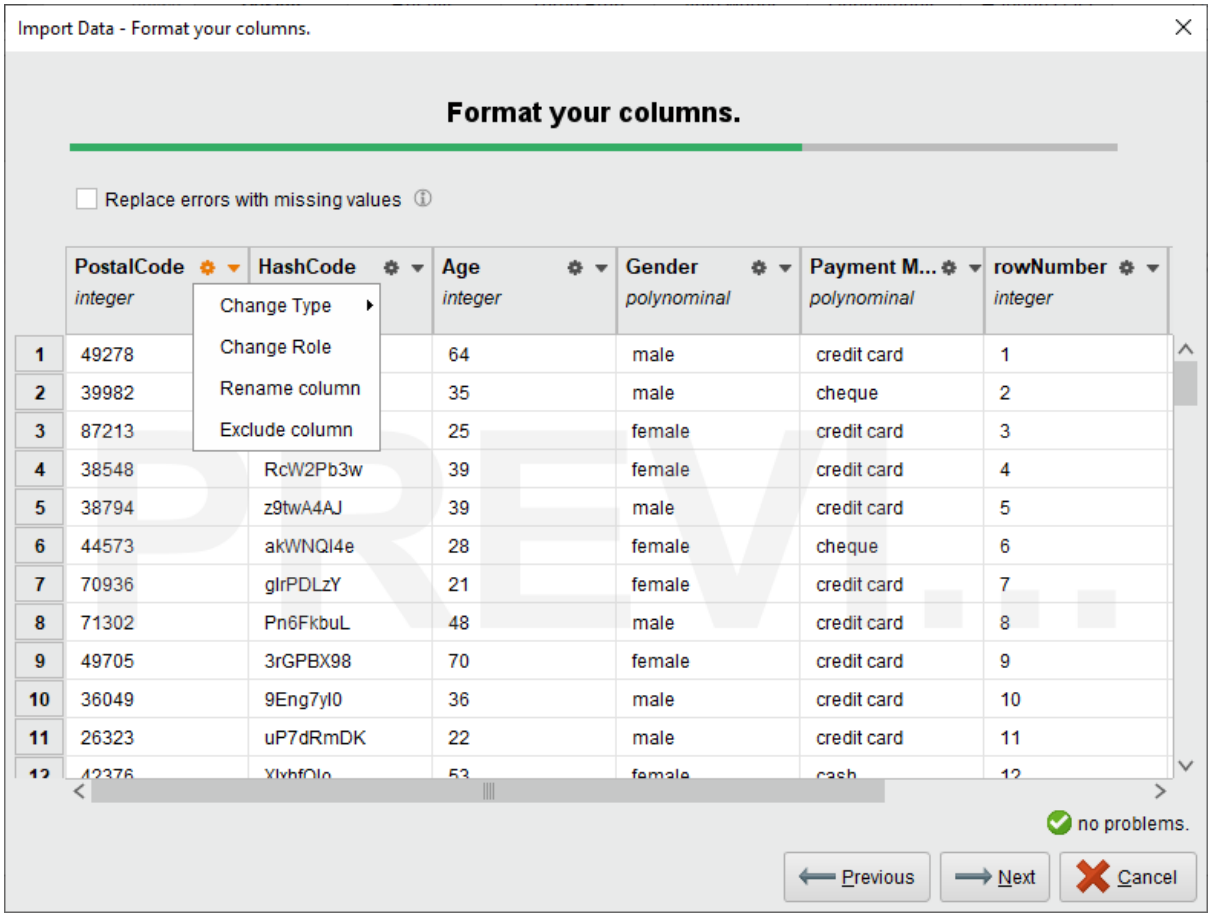


Figure 2.3 Format the Columns

The drop-down icon next to the attribute names allow us to format the columns (or **attribute** as RapidMiner calls columns) through the 4 tasks mentioning above. We will skip the attribute roles here and talk about them later, but let's go through the other three points.

2.2.1 Selecting Columns

Click the Exclude Column to select or unselect attributes for import. Deselected attributes will be greyed out. You can select it by clicking on the drop-down button again and click Include Column.

2.2.2 Defining Column Names

By clicking on the Rename Column, a new window for entering new name will be popped out. Renaming is just as easy as entering the new name that you would like to assign into this window and click OK.

2.2.3 Define Data Types

Defining a data type specifies the kind of values allowed for an attribute. RapidMiner supports the natural division of numbers, texts and dates. Numeric is the label for numbers, nominal for texts or strings, and date_time for dates. To allow more granular assumptions on the data, the main types can be subdivided. Figure 2.4 illustrates the hierarchy and provides a complete list of data types.

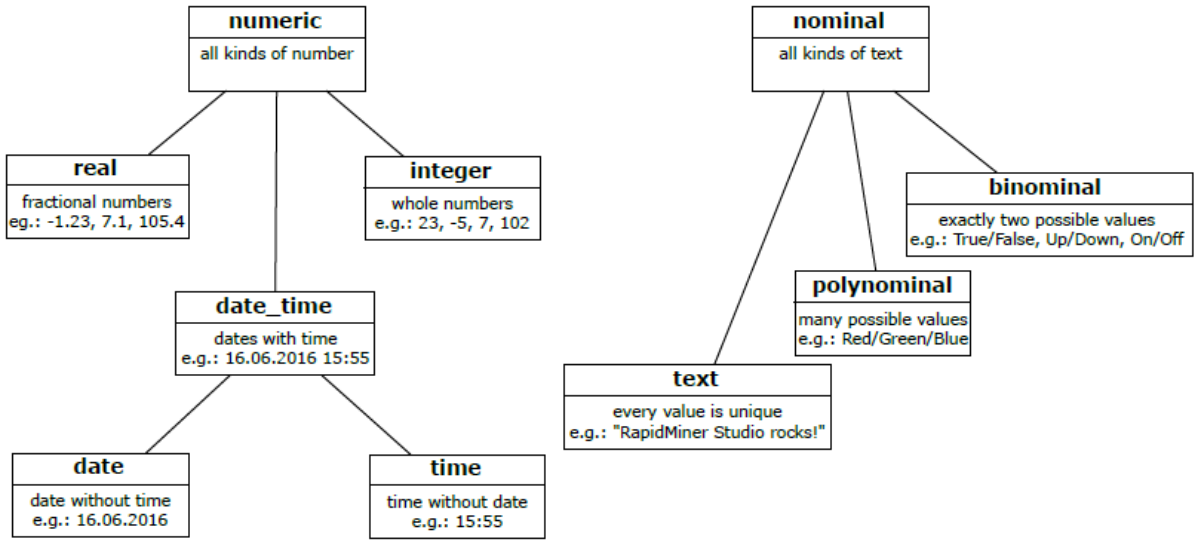


Figure 2.4 Data Type Hierarchy

It is important to define the correct data type for each column for several reasons:

- RapidMiner can check that all fields in the input data are valid. For example, if you specify “integer”, i. e., whole number, for the age, but in a certain row there is something that is not a number, e. g., twentythree, then RapidMiner can issue an error or warning.
- RapidMiner can more easily check pre-requisites for certain methods and algorithms that have certain assumptions on the data or offer help to get the data type right. We will see examples of that soon.

RapidMiner, being the helpful suite that it is, tries to guess the data types based on the data it sees. In this case it has done that job quite well. It has resolved the following for the data:

- PostalCode contains only numeric data, more precisely whole numbers, so type integer makes sense.
- HashCode contains different strings, so polynominal is fine, even though the semantics of these cryptic strings might be unclear.
- Age contains whole numbers, so integer is a good choice.
- Gender seems to be a good candidate for a binominal attribute – male and female. A closer look shows a third value, an ominous männlich. We certainly will have to take a closer look at that later. Because of that third value, for the data import we must choose a polynominal data type.
- PaymentMethod, with its three values, certainly makes sense as a polynominal.
- rowNumber is an integer for sure.
- LastTransaction and ChurnDate clearly contain dates including a time, so date_time is the perfect setting.

Usually, instead of reading the complete table, RapidMiner uses only the first 100 rows to guess the data types. This can be a big performance boost for tables with thousands of rows. On the other hand, it might guess the wrong data type if, for example, the first 100 columns contain only data that looks like numbers, but further down contains strings. To force RapidMiner to look at the complete table, uncheck Preview uses only first 100 rows and press the Guess values types button.

Now, since everything looks good, let's Finish with the Data Import Wizard and run the process by pressing the blue Run button in the toolbar. The process should run successfully, and we see... nothing. What happened?

As you might remember, in RapidMiner the data is generated (loaded, in this case) or modified by the operators and then delivered via the operators' output port(s). If you want to use the data, you have to connect the output ports to another port. So, connect the output port of Read Excel to the process result port on the far right, as shown in Figure 2.5. Remember, everything you connect to the results port will be available in the Results view for visual inspection.

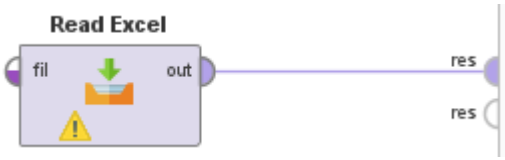


Figure 2.5 Connect Output Port to Results Port

2.3 Storing the Data Set in RapidMiner Studio

Now that you have seen that the behaviour of the Read Excel is what you expected, you should proceed to saving the obtained data in RapidMiner's local repository. That way we will have access to a static copy of the data, even if the original Excel file gets

moved, deleted or modified later (with the disadvantage that if the file gets replaced with an updated version with more information, we would need to run the Data Import process again to update the local repository's data set).

To do that we will use a new operator, the Store operator. You can search for this operator just like you did for Read Excel; after you've found it drag and drop it on top of the connection line between the out port of the Read Excel operator and the res port. This inserts the operator and connects its in and out ports automatically. The last step is specifying the repository entry parameter.

In the beginning of this course, you will see that we have already created a subfolder specially for data. Go ahead and save it there with an appropriate name (i.e. churn data), as shown in Figure 2.6.

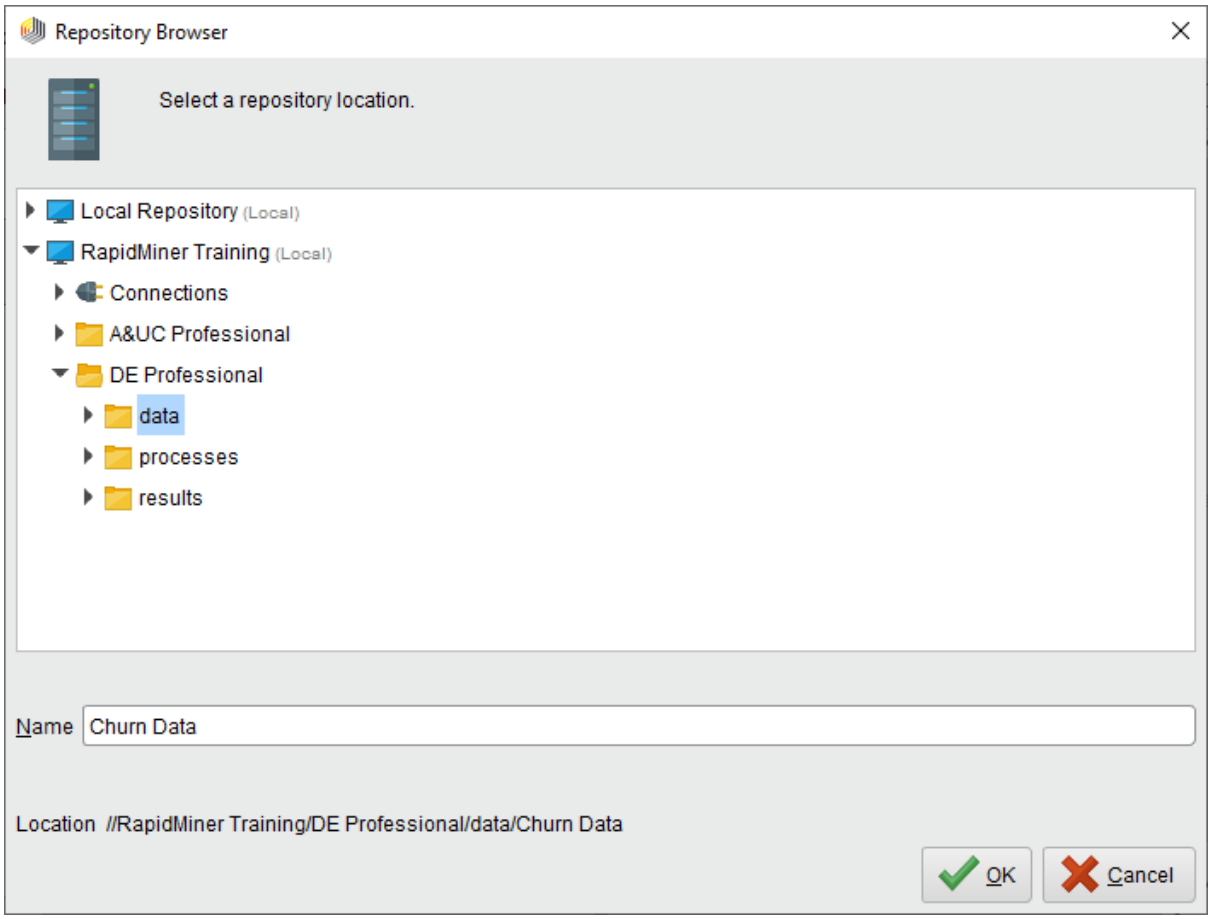


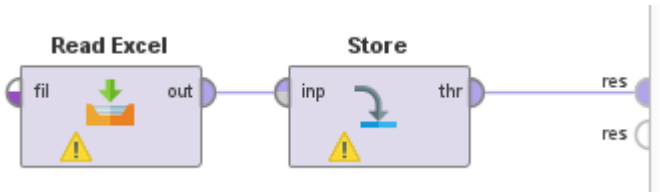
Figure 2.6 Selecting the Entry in the Local Repository

Exercise 1. Data Import

If you haven't done so yet, follow along the instructions in the current chapter to create a new process, load the data from the Excel file and store it in the local repository. Once you have loaded and stored the data, save the process by pressing the save button, browse to the "processes" subfolder and save it under an appropriate name (i.e. read excel).

Answer to Exercise 1

Your Process panel should look like

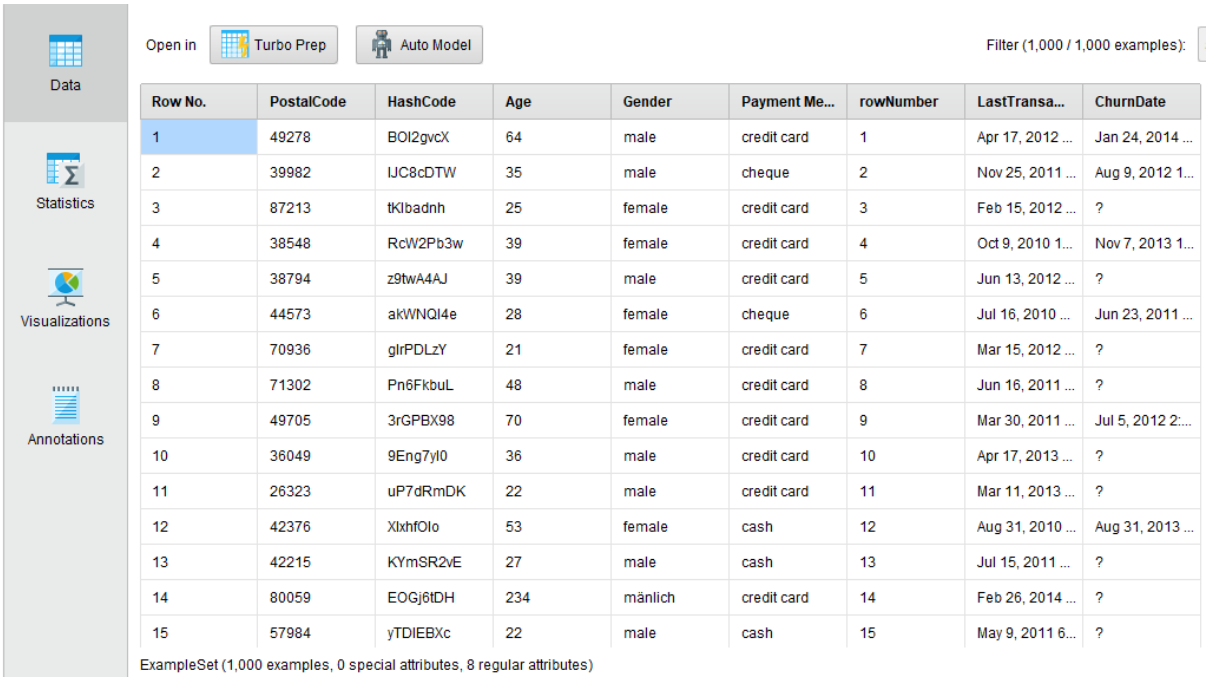


You might have noticed that as soon as you connect Read Excel to the first results port, a second results port appears. This comes in handy if you ever produce more than one result. You can connect as many results as necessary and inspect all of them in the Results view when the process has finished.

2.4 Data Exploration

Now that you have configured and run the process that loads and stores the Excel file, let's see what you have in the Results view.

The first thing you might notice is that it is the same data that we saw in Excel – you are looking at the Data tab, seen in Figure 2.7. Nothing special here, only that the data is now in RapidMiner. In addition to the data, the view reports, at the top of the table, the number of examples (rows) and attributes (columns).



Row No.	PostalCode	HashCode	Age	Gender	Payment Me...	rowNumber	LastTransa...	ChurnDate
1	49278	BOI2gvcX	64	male	credit card	1	Apr 17, 2012 ...	Jan 24, 2014 ...
2	39982	IJC8cDTW	35	male	cheque	2	Nov 25, 2011 ...	Aug 9, 2012 1...
3	87213	tKIbadnh	25	female	credit card	3	Feb 15, 2012 ...	?
4	38548	RcW2Pb3w	39	female	credit card	4	Oct 9, 2010 1...	Nov 7, 2013 1...
5	38794	z9twA4AJ	39	male	credit card	5	Jun 13, 2012 ...	?
6	44573	akWNQI4e	28	female	cheque	6	Jul 16, 2010 ...	Jun 23, 2011 ...
7	70936	glrPDLzY	21	female	credit card	7	Mar 15, 2012 ...	?
8	71302	Pn6FkbuL	48	male	credit card	8	Jun 16, 2011 ...	?
9	49705	3rGPBX98	70	female	credit card	9	Mar 30, 2011 ...	Jul 5, 2012 2:...
10	36049	9Eng7yl0	36	male	credit card	10	Apr 17, 2013 ...	?
11	26323	uP7dRmDK	22	male	credit card	11	Mar 11, 2013 ...	?
12	42376	XlxhfOlo	53	female	cash	12	Aug 31, 2010 ...	Aug 31, 2013 ...
13	42215	KYmSR2vE	27	male	cash	13	Jul 15, 2011 ...	?
14	80059	EOGj6tDH	234	männlich	credit card	14	Feb 26, 2014 ...	?
15	57984	yTDIEBXc	22	male	cash	15	May 9, 2011 6...	?

ExampleSet (1,000 examples, 0 special attributes, 8 regular attributes)

Figure 2.7 The Data Tab in Results View

Just a usability note here. If a column is too small, you can easily resize it by dragging the border between the header fields. You can also rearrange the order of the columns. Please note that this only changes the visualization; it does not change the data itself.

Now take a look at the data. Note that there are a lot of question marks (?) in the DateChurn attribute. A question mark indicates that RapidMiner does not have a value for this cell, it is missing. In the case of DateChurn that is because there are no values for this attribute in the input data. Using the Filter box in the upper right, you can select

to show all, only those with missing attributes, or only those with no missing attributes. There is also an option to filter by label, but since you don't yet know what a label is, ignore that option for the moment.

While the data tab is a nice way to understand the data and get a feeling for what is stored in which column, it does not provide a great overview. Commonly a data scientist may face questions like "What is the range of a numeric attribute?", "which values does a nominal attribute have?", "what is the distribution of the attribute?", etc. While not immediately apparent in the data tab, this information is easily accessible in the Statistics tab, as seen in Figure 2.8. Click the corresponding entry in the list on the left to open the Statistics tab.

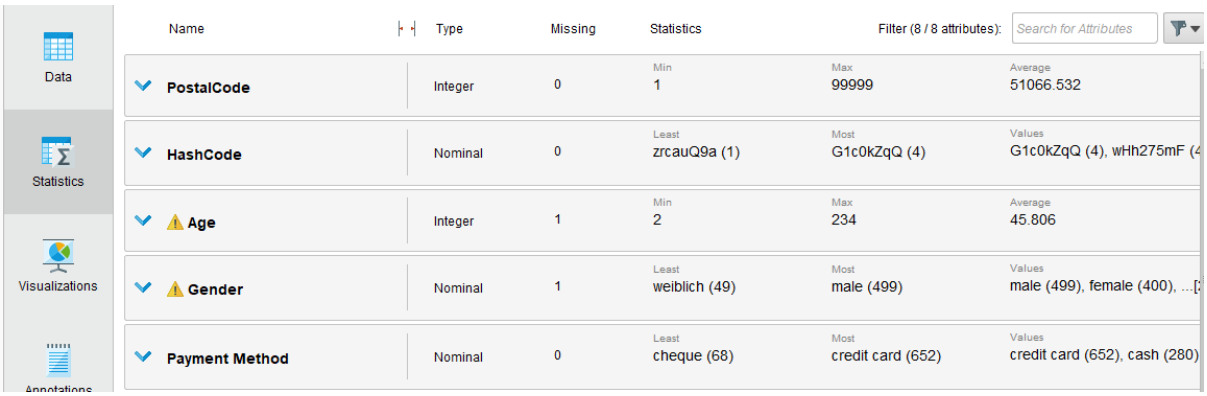


Figure 2.8 The Statistics Tab in Results View

In the Statistics tab you'll see a list of all attributes – every column from the data view is now summarized in its own row. The left-most column displays the attribute name, the next column lists the data type, while the third column indicates the number of missing values. The right half of the view displays various statistics (depending on the data type). For example, the view lists:

- minimum, maximum, average, and standard deviation for numeric attributes
- date range and total duration for dates
- most and least frequent values, as well as the frequencies of the values, for nominal attributes

A few more tricks:

- Click an attribute to display more detail, for example, data distribution in the form of a histogram.
- Click Details to list all values in the data, together with their frequency, for nominal attributes.
- Click the open chart link, beneath the frequency chart, to open the Visualizations tab.
- To return to the Statistics tab, click the corresponding entry on the left.

After the next section's introduction to supervised learning, we will unleash the full power of the statistics tab before cleaning and preparing your data.

2.5 Supervised Learning

The idea behind the term supervised learning is to derive patterns, rules, or models from past experiences. Imagine the following scenario: you're back in college, sitting in a data mining lecture. Your neighbour returns from the break and perches his newly filled coffee mug on the corner of his desk. While turning a page, he accidentally knocks the mug and it sails off the table. Observing this, your natural intelligence program switches on. You know immediately that within a half second the cup will hit the ground and spill coffee everywhere. Thanks to your fast reflexes, you reach out and catch the cup before the coffee spills, earning a grateful smile from your neighbour.

What just happened? You reacted based on experience: after observing dozens of coffee mugs falling to the ground during your lifetime, your brain developed a rule that reads something like "if a coffee mug falls from the table, it hits the ground, breaks, and spills coffee everywhere." Maybe you've observed people losing the grip on their cocktails glasses, resulting in colourful cocktails spilling to the floor. In that case, your rule could read "if a fragile container falls from an elevated position, it will break and spill its contents. That's bad."

The difference in these models is that the latter is much more abstract and transferable then the simple “coffee” rule. In both cases, though, the model is based on past experience: you have observed events (falling fragile item), seen the output (big mess), and learned from it. You can even transfer it to new situations (cocktail glass - hand - floor to coffee mug - table - leg). This is called supervised learning, it allows you to **predict the future**. You know that a big mess will happen while the glass is still in the air, and you can react. That is analytics in a nutshell – create a predictive model, peek into the future, and react to improve the outcome.

You can easily transfer this example to the current business case. Remember that your customers are using an app to buy your products. Based on their actions you want to predict who is likely to churn in the near future such that you can react and prevent customer churn by special offers and other marketing actions tailored to exactly the risky customers. Since computers don’t make experiences on their own, you have to deliver the observations in a suitable format.

Back to the hands-on example. You just loaded your observations or experiences from the past – that is, the customer churn data – into RapidMiner. Throughout this course you will feed that data into various modeling algorithms. Remember that you need observations, together with the outcome (or label, as we call it in RapidMiner. A complete observation, including the label, is called an example.

Looking at the data, you can easily identify the label and the regular attributes. Obviously, if a customer has churned in the past then the ChurnDate tells you the date when he uninstalled the shopping app. If he did not uninstall the app, i.e., is still a customer, the churn date is missing. Using the appropriate algorithms, you can find relationships between attributes and the label. For example you may learn something like “if the customer is male and older than 65, he is likely to churn,” or some other, more abstract but also more powerful rules or models.

Of course, to find good rules it is crucial that the input data be as clean as possible. With this in mind, let’s leverage the statistics tab and try to find anything that looks odd.

Exercise 2. Data Preparation Tasks

Before reading on, inspect the data and try to find any oddities.

Answer to Exercise 2

Found anything? Here's the complete list of issues. If you have found three or four of them you are on a good track:

- Missing values: you have already seen that ChurnDate contains a lot of missing values. On top of that, there are more missing values: Age and Gender contain one missing value each. This shows the power of the Statistics tab – in the Data tab it is virtually impossible to spot one or two missing values in a data set with thousands of rows.
- Range: remember that you are running an online shop. Your customers should be somewhere between 16 and 100 years of age. Here you have customers as young as 2 years and as old as 234 years – clearly something is wrong.
- Gender: you should expect exactly two different genders, but actually have four. In addition to male and female, you also have männlich (and also misspelled männlich) and weiblich – the corresponding German translations. Probably one of your employees is from Germany. You should unify the gender values.
- Irrelevant attributes: there are some attributes that don't contain any valuable and/or interpretable information, especially the HashCode.
- ID attributes: the rowNumber uniquely identifies each customer which can, as we will see, seriously confuse the algorithms. You should tell RapidMiner to treat that attribute as an ID (in other words, ignore it for prediction purposes).

An impressive list, right? And you discovered all this by simply exploring the data. In the following sections, we will walk through the process of dealing with each of these discovered issues.

2.6 Data Preparation Steps

As mentioned earlier, all modelling algorithms require clean data as input. For that reason, we will learn next how to properly prepare the data. Let's begin by creating a new process that will handle all the data preparation steps. To do so create a new blank process and search for the churn data set in the Repository panel. Once you've found it, drag and drop it into the Process panel, and you will see that an operator is created: Retrieve churn data. This will serve the same purpose as the Read Excel operator in the last exercise, but we will add a lot of extra steps to properly clean the data. Your new process should look like the one on Figure 2.9.

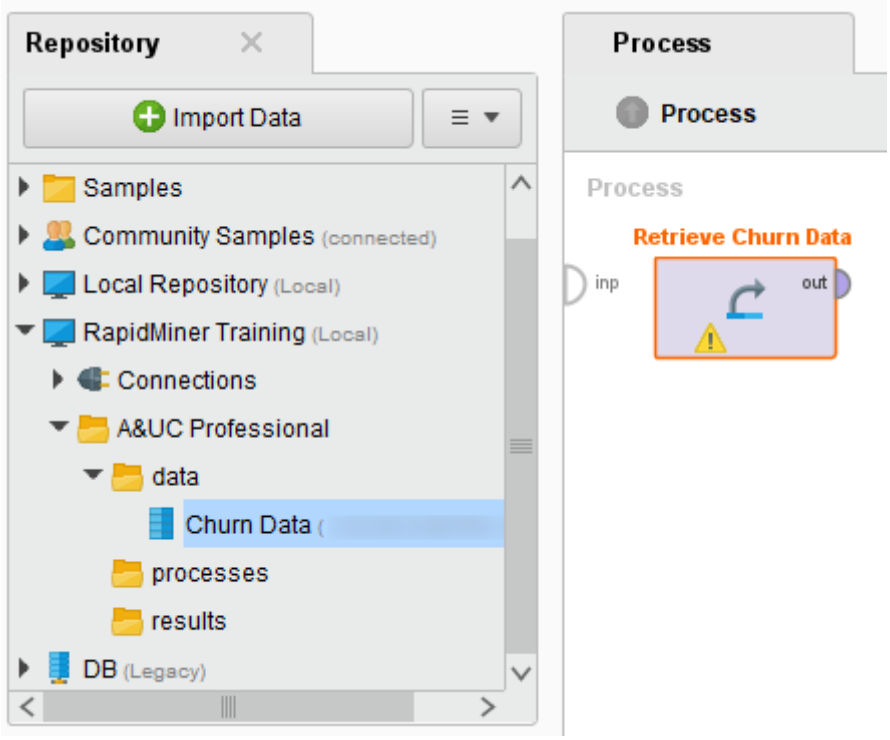


Figure 2.9 Retrieving the Churn Dataset

2.6.1 Missing Values

In our data, Age, Gender and ChurnDate all contain missing values. In his work, *Exploring Data with RapidMiner*, Andrew Chisholm describes at least three different kinds of missing data and how to deal with it. Here, we'll look at the types relevant to our use case.

Note that here we are dealing with two different kinds of missing values. The Age and Gender missing attributes mean that we don't have that information (for whatever reason). However, a missing value in ChurnDate actually has a meaning, and an important one: the respective customer did not churn.

Let's focus on age and gender first. Missing information can be blamed on a variety of causes. In our case, the customer probably did not supply it during sign-up. In other scenarios, missing data may be the result of defective sensors, sloppy data migration or copy actions, incomplete user input, etc. (Andrew lists a few additional sources and scenarios in his book.) Regardless of the reason, we have to deal with them.

Basically, there are three ways of dealing with our missing values:

- If an attribute contains a lot of missing values, we should consider removing the attribute.
- In a nominal attribute with missing values we can introduce a new category, e.g., unknown. While to our human eye this does not add any value, it gives RapidMiner a chance to create rules for examples having an unknown category. For the computer, unknown means something very different from missing.
- If only a few examples contain missing values, we can remove those examples without losing too much information.

Since our customer churn data only contains one missing instance of Age and one of Gender, we can filter out those two examples. This is done with the Filter Examples operator. Before looking deeper into that operator, let's see how to find it.

2.6.2 The Operator Tree

The operators panel in the bottom left corner of the Design view contains a list of all operators. The operators are categorized by their function: importing and exporting data, modelling and various other things. The Filter Examples operator that we are going to use soon is located in the Blending → Examples → Filter group.

By navigating through the hierarchy, you can explore the available operators and find what you need. If you already know the name of an operator, you can also use the search box above the list and simply type in the name of the operator (or even parts of it, as we saw with the Read Excel operator).

2.6.3 Filtering Examples

Now that you know how to locate new operators, let's have a look at Filter Examples: It takes an example set as input (left-side port). The operator delivers three output example sets on the right-hand side:

- The top-most port, example set (exa), delivers all examples of the input that match the filter rules.
- The second port, original data (ori), contains the same data as the input.
- The lowest port, unmatched examples (unm), delivers all examples of the input that do not match the input.

To configure the filter rules follow these steps after returning to the Design view, as shown in Figure 2.10.

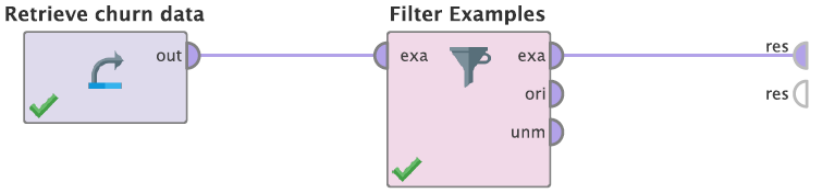


Figure 2.10 Filtering missing values: the process setup

1. Add the operator to the process and connect Retrieve churn data's output port to its example set input port.

2. Connect Filter Examples's example set output port to the process's result port at the far right.

3. In the Filter Examples Parameters panel, click Add Filters.... The Create Filters dialog opens with a familiar filtering interface. (Make sure the Filter Examples operator is highlighted to see the options for parameter configuration.)

4. In the first column, select an attribute. We want to filter on, e.g., Age. If it does not appear available, you can type in the name – note that the attribute names are case sensitive!

5. In the middle column drop-down, select a comparison tool. In our case we select is not missing. (If you are using an operation that requires a value to compare against, for example, "less than" or "contains," enter that value in the input field on the far right.)

6. Click the Add Entry button at the bottom of the dialog box. Now create a similar filter for gender.

7. Save and apply the filters with the OK button.

Figure 2.11 shows the operator's filter configuration dialog.

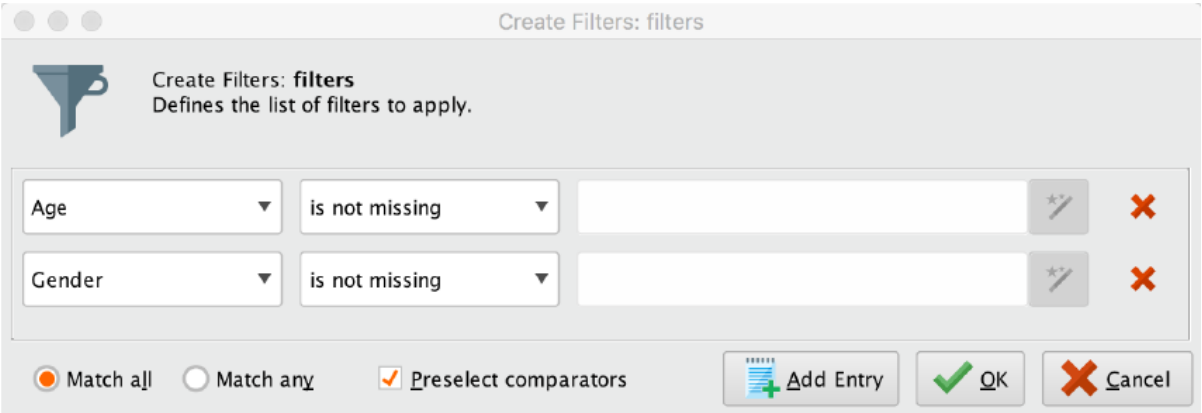


Figure 2.11 Filtering Missing Values: the Create Filters Dialog

Pro-tips:

- By default an example is matched (and kept) only if it matches all conditions. You can change the behaviour to keep an example if it fulfils at least one of the conditions by selecting the Match any option at the bottom of the dialog.
- If, for some reason, an attribute does not display in the attribute pulldown of the Create Filters dialog, you can simply type its name into the first column. If the list is entirely empty, make sure that the input port of the operator is properly connected.
- If the wrong comparators are shown in the second column (e. g., numerical comparators for a nominal attribute), you can disable the Preselect comparators option at the bottom of the dialog.
- If you check the invert filter parameter in the Parameters panel, the exa port delivers only examples that do not match the conditions.
- You can select different condition classes in Filter Examples, offering more advanced filter options. Read the operator documentation for the different options.

Exercise 3. Filter Missing Values

Using the steps above, add a Filter Examples operator to the process and configure it such that the result does not contain any missing values in Age and Gender.

Run the process and check the results. Your data set should now contain 998 examples and the Statistics tab should not list any missing values other than ChurnDate.

2.6.4 Creating A Label

Now that you've cleaned Age and Gender, let's move on to the third attribute with missing entries: ChurnDate. Since a missing churn date means that the customer is still active, removing those entries is not an option. What we really want to do is create a model that simply predicts whether a customer will churn in the near future or remain loyal. To train this model we need a new Churn attribute with the following rule:

if the churn date is missing then loyal otherwise churn.

That rule will result in a nice and neat binominal attribute that contains exactly the information that we want to predict later on – the perfect label for our supervised learning problem.

In RapidMiner, new attributes can be generated with the Generate Attributes operator. As input, it expects an example set; the output is an example set that contains the attributes from the input as well as the newly generated attributes. After adding the Generate Attributes operator after Filter Examples and connecting it, you configure the generation rules by clicking the Edit List button in the Parameters panel. The function descriptions dialog opens, as shown in Figure 2.12. This is where you specify the generation rules. In the left column, "attribute name", enter the name of the new attribute; in the right, "function expressions", specify the generation rule.

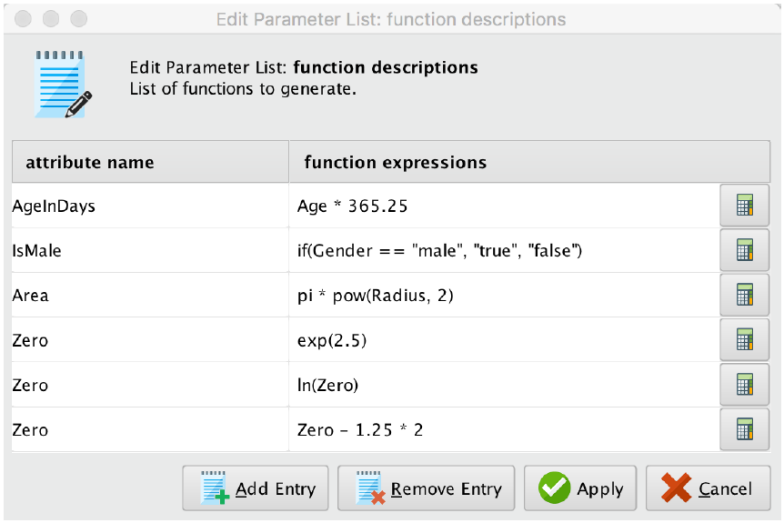


Figure 2.12 Generate Attributes: Function Descriptions Dialog

For the rules or **formulas**, use RapidMiner's expression syntax. To get editing help with the syntax, click the small calculator button next to the function expression input field. This opens the Edit Expressions dialog, as seen in Figure 2.13.

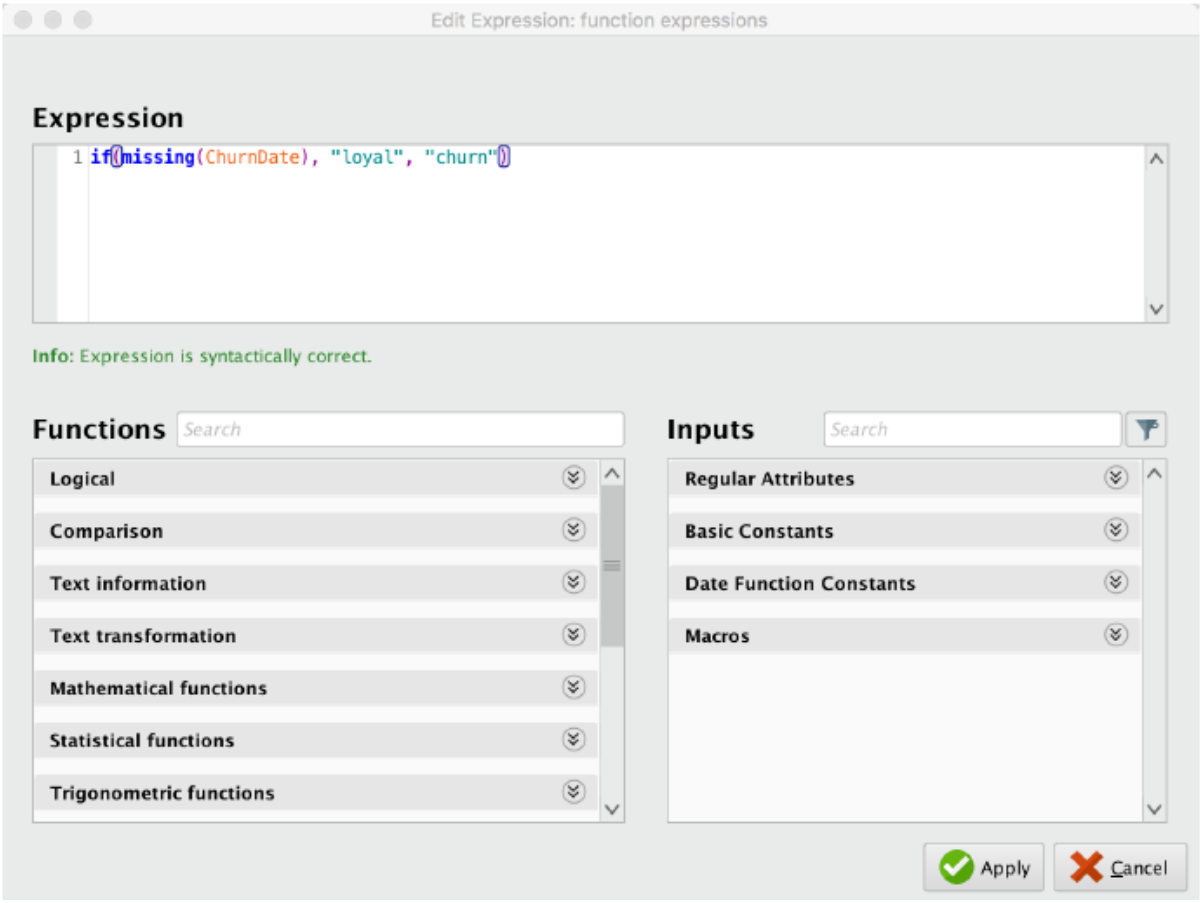


Figure 2.13 Generate Attributes: the Formula Editor

Enter the formula that defines the new attribute in the Expressions field. It can contain simple mathematical calculations as well as complex string transformations, if-then constructs, date arithmetic, and much more. You can browse the possible values for functions and inputs in the dialog at the bottom. When you click Apply, RapidMiner runs a validation check. If everything is OK, the expression is applied to the new named attribute on the previous page.

The following section provides some examples.

2.6.5 Examples Using Generate Attributes

In the simplest case, a formula consists of a single value:

365

When the operator is executed with this formula, the new attribute contains the value 365 in all examples. RapidMiner automatically detects the type of the new attribute – in this case it is Real.

This is a pretty static example. The fun starts when you use the values of existing attributes to generate a new attribute. Assuming we have an attribute called Age (which we actually do have in our data set), we can create a new attribute AgeInDays and write as expression:

365

This formula is evaluated for each line of the data set. After executing the operator, the AgeInDays contains the age of each customer in days.

You can also perform more advanced mathematical calculations, e. g., calculate the sine of a value:

sin(pi) → 0

This yields the sine of the predefined constant pi. While this is natural mathematical syntax, the expression showcases an important concept: sin() is a so-called function, and pi is the argument of the function. We can say that a function does something with its argument, in this case sin calculates the sine of pi. You can recognize functions because they are followed by an argument list that is always enclosed in parentheses.

Some functions have more than one argument:

round(1.234, 2) → 1.23

This expression rounds its first argument 1.234 to 2 fraction places, yielding 1.23.

Arguments can be optional. For round(), the second argument is optional. With only one argument, it rounds that argument to a whole number:

round(1.234, 2) → 1

Dealing with Strings

So far we have only worked with numerical values. Of course RapidMiner's expressions also support strings. To assign a string constant to the new attribute you would type:

"My String Value" → My String Value

Note that all string constants must be enclosed in double quotes.

You can yield the first characters of a constant with the prefix() function. For example:

prefix("My String Value", 2) → My

Returns the first two letters of My String Value, i. e., "My".

However, if you want to use nominal attributes, you must not use the quotes – attributes names must never be quoted! So, to get the first letter of the Gender attribute, you would type:

prefix(Gender, 1) → either m or f

Other Functions

The previous examples of functions all have in common that they perform a transformation of, or a calculation on, their arguments. There are other functions you can use to perform a test on arguments. One such example is the if() function, which expects three arguments. The first argument is a condition and must evaluate to true and false. 5 If the condition is met, i. e., the value is true, then it returns its second argument; if the value is false, it returns the third argument. Here's an example:

if(Age >= 18, "adult", "child") → child or adult

Here's one final example. The missing() function expects an attribute name as an argument and returns true if the attribute has a missing value or false if it does not. For example:

missing(ChurnDate) → true or false

We've just reviewed only a small subset of the available functions. You can explore them all in the function editor. They are sorted by topic; browse through the topics by expanding the titles through klicking on the arrows. Clicking on the "information" icon provides a description and usage example for each function.

Hints on using Generate Attributes and attribute names:

- As a best practice, attribute names start with an uppercase letter, do not contain any special characters or whitespaces, only use letters and numbers, and indicate new words using the CamelCase style. Examples: ChurnDate, DayOfBirth, etc.
- If your attribute names contain special characters and whitespace, you have to enclose them in square brackets to use them in a formula, e.g., [Average daily revenue] * 365.
- If your attribute names contain square brackets themselves, you have to escape them with a backslash, e.g., [duration\[hours\]] / 24.
- Common constants, such as pi and e, can be used in the formulae.

Pro-tip 1: You can use Generate Attributes to overwrite an existing attribute. Simply put the name of the existing attribute into the left column. You can even use the old attribute in the formula – in that case, the original value is used to derive the new one.

Pro-tip 2: You can split up complicated calculations into several rows because the results from calculations of previous rows are available to later rows. See figure 3.13 on page 57 for an example: the calculation of Zero uses this feature (admittedly a pure showcase example, but this feature comes in handy in longish real-world calculations).

Exercise 4. Generate Label

We have already seen how to check for missing values, but the result was always true or false. While this is fine for a computer, as humans we need something more readable and presentable:

- Create a new attribute called Churn that either takes the value churn or loyal, depending on the value of ChurnDate.
- As always, carefully check the results. Pay special attention to the logic – many students end up with inverted labels in the first run.

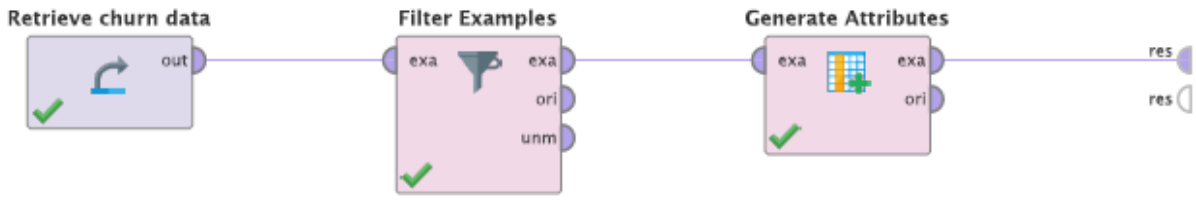
Hint: You will need the missing() and if() functions of Generate Attributes.

Answers to Exercise 4

Add a Generate Attributes operator to the end of the process, connect it, and add the following line to its function list (via Edit Parameters):

Churn	if(missing(ChurnDate), "loyal", "churn")

Your process should look like this:



2.6.6 Coming to Age

The next stop in our issues list is the Age attribute. Remember, we saw that some values are way out of range. One customer is 234 years old – probably a typo. Other customers are younger than 16, some even as young as two. Those may also be typos, but in any case, they are invalid values (since our shop's terms and conditions only allow customers older than 16).

As we have discussed previously, we need to clean the data before passing it on to model building. That means that we must get rid of those examples with an invalid Age attribute.

Exercise 5. Filtering by Age

Modify the process such that the final data set only contains customers aged between 16 and 110.

Answer to Exercise 5

There are several possibilities for solving this one. All use the Filter Examples operator.

- The easiest way to make sure we have only valid values is to extend the Filter Examples operator that we used to filter by missing values.
- You could also append a second Filter Examples operator to the end of the process. The new operator will contain only the age filters.
- (Advanced) If you add a second Filter Examples operator, you can leverage the expressions introduced in the Generate Attributes section to filter examples:
  - In Filter Examples, change the condition class (turn on advanced parameters to see it) to expression.
  - In the parameter condition, you can type an arbitrary condition that must evaluate to true or false. To keep all customers in the valid age range:

    ```
    Age >= 16 && Age <= 110
    ```

    The && tells RapidMiner that both conditions must be met. (To require only one or the other, use two vertical bars ||).

Run the new process with the Filter Examples operator in place. The data should now contain 996 examples.

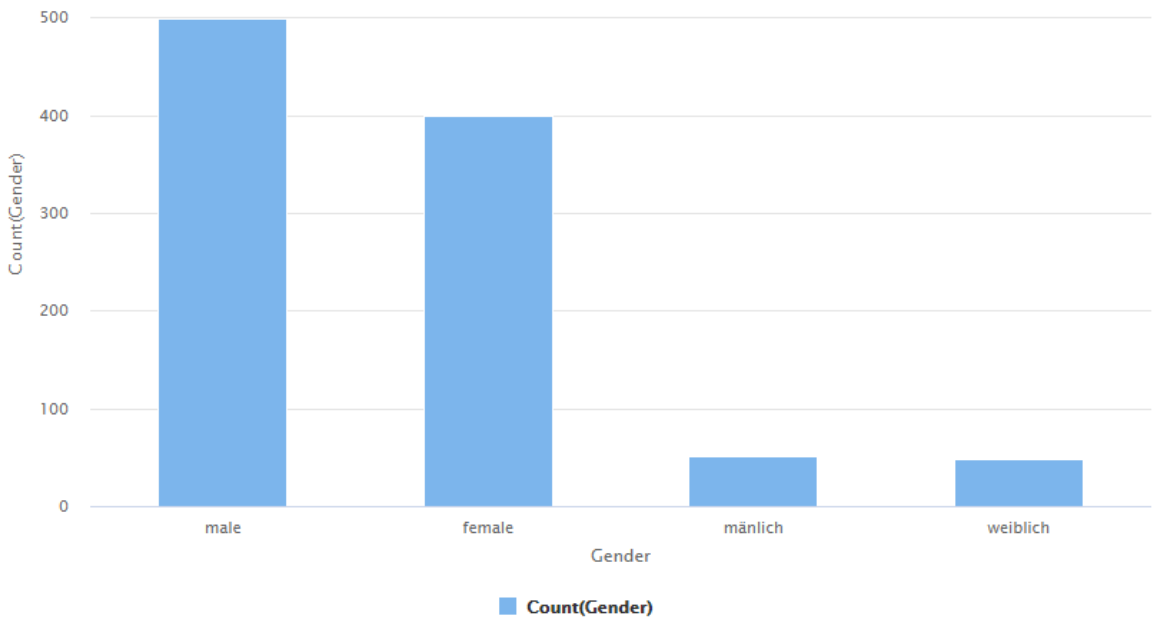
2.6.7 Back to Two Genders

We have seen that in addition to male and female, our Gender attribute also contains the German terms männlich (for male) and weiblich (for female). This is not only ugly, but can seriously confuse the algorithms. While we humans know (after consulting the dictionary) that male and männlich have the same meaning, the algorithms don't have that background knowledge and will just assume four different, unrelated terms. So let's correct that.

With what we've learned so far, we can already devise strategies to replace certain values of an attribute. Generate Attributes, RapidMiner's Swiss army knife, can get the job done with an expression like:

```
if(Gender == "männlich", "male", Gender)
```

From that expression, RapidMiner checks whether the value of attribute Gender is männlich. If so, it replaces it with male. Otherwise, the original value of Gender is used. We'd add the same kind of expression for weiblich and female.



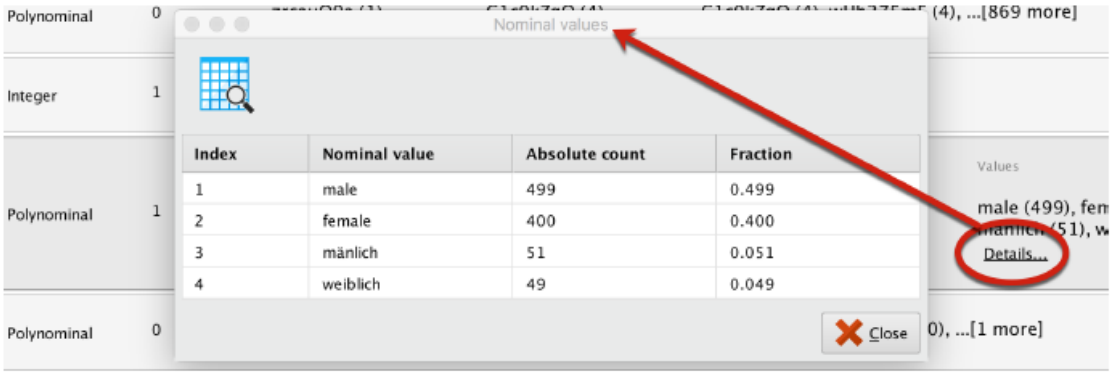
While this works, there is a simpler method. It uses another operator – Replace (which you can find in the Blending → Values group). In addition to the standard attribute selector that specifies on which attribute to work, Replace's replace what parameter takes a search string, and the replace by parameter a replacement string. To replace männlich with male enter männlich for replace what and male for replace by.

You will notice that you can replace only one string at a time. To make a second replacement (which you will need), you must add a second Replace operator.

The Map operator supports several replacements in one go, but does not support regular expressions.

If your keyboard layout does not support German umlauts (that is, the ä in männlich) you can copy the string from the Statistics tab. To do so:

1. Go to the Statistics tab.
2. Click in the row containing the Gender attribute to expand the listing.
3. Scroll to the right and click Details... below the values list.
4. In the dialog, select the values you want to copy and copy them into
5. the clipboard by pressing Ctrl + C.



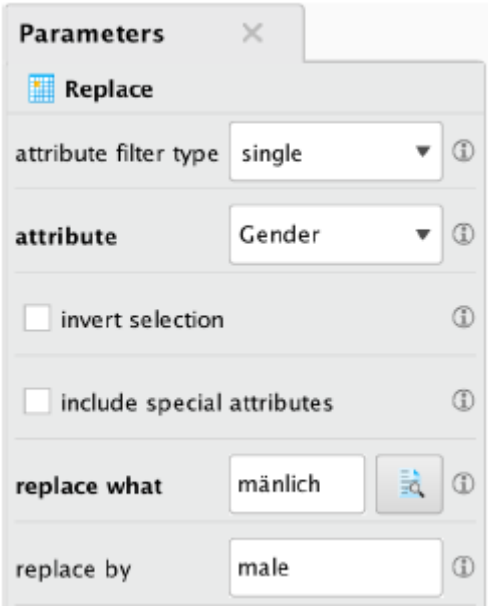
Exercise 6. Replacing Invalid Gender Strings

Extend the process so that the German terms for gender are replaced by the English terms male and female.

Check your results by looking at the Statistics tab. They should look similar to:

Answers to Exercise 6

Use either two Replace operators in a row, configured as figure below, or the Map operator (since you don't need a regular expression).



2.6.8 Select Attributes: Focus on Relevant Input

In our efforts to clean the data, so far we have focused on the values of individual attributes. Now let's take a step back and look at the big picture. Because truth be told, there are some attributes that are not necessary and can even be obstructive for model creation.

Noisy and Irrelevant Attributes

As we discussed, in supervised learning an algorithm tries to find relations between the input attributes and the label. The quality of many algorithms suffers from attributes that contain very noisy data or information that is completely unrelated to the label.

In our case, the HashCode is one such attribute. It contains a seemingly random collection of numbers and letters, derived from the other attributes via a so-called hash function. This is useful for fast database searching or equality comparisons between several roles, but for supervised learning it does not contain any useful information. Therefore it should be removed.

Redundant Attributes

Irrelevant attributes are not the only problem when it comes to model generation. Another problem is redundancy. Let's remember that we want to find a model to derive whether a customer will churn or not and, hence, created our label attribute Churn. If we now throw all other attributes into some magic rule generation algorithm (revealing that magic is the topic of the next chapter) and ask it to find a rule for Churn, it will come up with a simple but precise rule:

if ChurnDate is missing then loyal else churn

Sound familiar? Yes, that's exactly the rule that we used to generate the label.

Does it work? Obviously yes, the rule is correct.

But is it what we want? Despite its simplicity and precision, we can't actually call it a good rule. ChurnDate is merely a different representation of the information that we want to predict. The question to ask, when deciding whether we may use certain information for model creation or not, is the following:

Is the information available when we want to apply the model?

Remember that our goal is to predict churn or loyal for customers that we do not know yet whether they will churn or not. For such customers, there is no ChurnDate column, and therefore, the rule won't work and can't be applied. So we must also remove the ChurnDate attribute from the training data. By doing so, we force the magic learning algorithm to find real relations between properties such as age, gender, etc., and the churn behaviour.

The Select Attributes Operator

Now that we have identified two attributes (HashCode and ChurnDate) that must not be used for training our model, its time to actually remove them. In RapidMiner this is done using the Select Attributes operator.

With the Select Attributes operator, you select one or more attributes that remain in the data set. We'll use the attribute filter type, the same pull-down we used with the Replace operator. But while in Replace we selected a single attribute by setting the attribute filter type to Single, now we want to select several attributes, as follows:

- Set attribute filter type to subset
- Click the Select Attributes... button; in the resulting dialog, move the attribute that you want to keep to the right-hand side (from Attributes to Selected Attributes). All attributes that remain on the left-hand side will be removed from the data.

The Select Attributes operator can remove columns from the data set, with configuration set as shown in Figure 2.14.

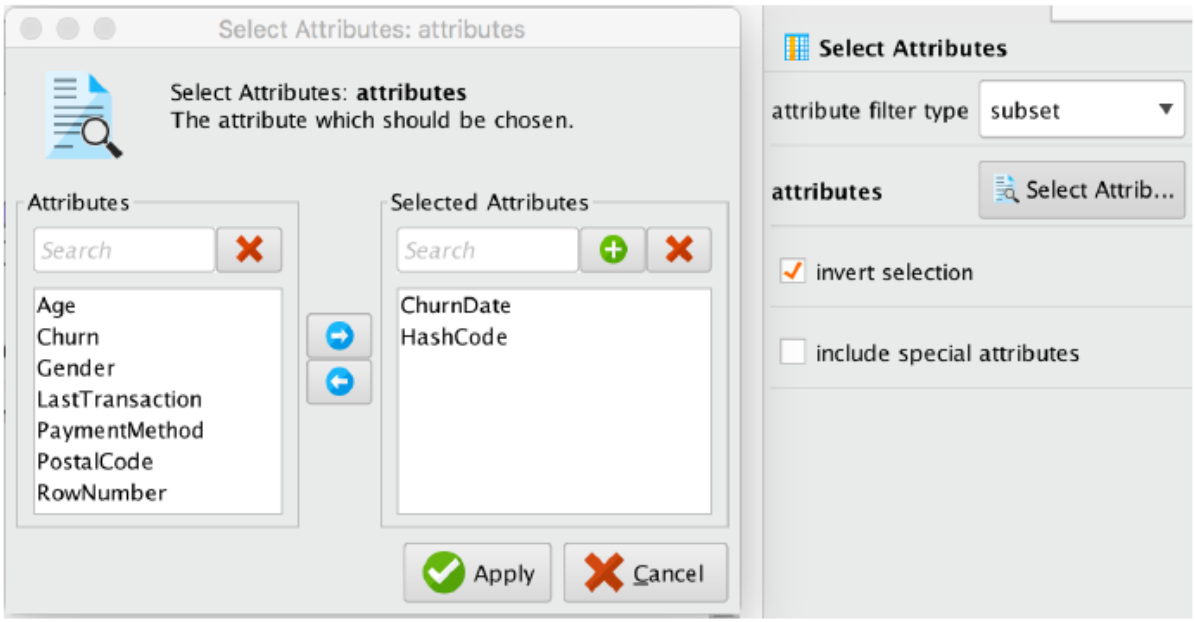


Figure 2.14 Select Attributes operator

If you only want to remove a few attributes, it may be easier to select the attributes to remove rather than the attributes to keep. Select Attributes supports this. Tick the invert selection option in the Parameter view and the behavior is inverted, i. e., the selected attributes will be removed. If you only want to remove a single attribute, you even don't need the subset filter type; use the Single attribute filter type option.

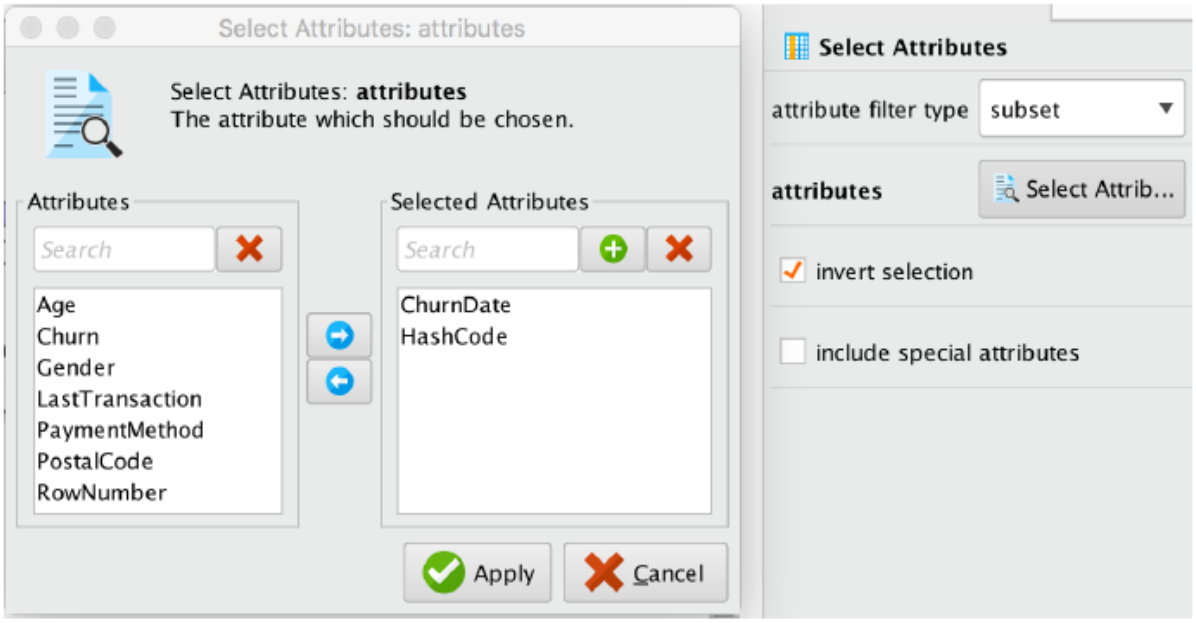
In more complex processes, you may find that not all data set attributes are listed in the dialog or the attributes lists. In that case, just type in the names. Ticking Synchronize Meta Data with Real Data on the Process menu also help.

Exercise 7. Select Attributes

Remove HashCode and ChurnDate from the example set.

To get a feeling for the operator, try both variants, i. e., with and without invert selection.

Answers to Exercise 7



2.6.9 Attribute Roles: Define the Function of an Attribute

In the last section we learned how to remove attributes from the data set, and we identified two attributes to remove. In this section we will identify one more problematic column and learn an alternative way of signalling RapidMiner to ignore it. We will also learn how to inform RapidMiner of special meanings, or roles, of the attributes.

ID-like Attributes

The rowNumber attribute contains a unique value for each row. With such a unique identifier, short id, another trivial model can be created:

- if id is 1 then churn
- if id is 2 then churn
- if id is 3 then loyal
- if id is ...

This model works perfectly on the training data. But it also works only on the training data. If you try to apply this to a new customer, with a row number that the model has not seen before, the model will utterly fail. We say the model is overfit and does not generalize. We've dedicated a complete chapter to this topic in the modelling discussion. For now, let's just take away the knowledge that ID-like attributes must not be used for model creation.

But rowNumber can still prove useful in other ways. For example, it allows us to identify each customer and find additional information – e. g., name, address, etc. – referenced by the respective row number. This is where roles come into the game.

Special Attribute Roles

You can assign roles to an attribute to tell RapidMiner of a special meaning or treatment. In general, roles have the following properties:

- Each role can be assigned only once per data set, i. e., they must be unique.
- Attributes with special roles, unless explicitly included, are ignored by all following operators.
- There are a number of predefined roles, which have a special meaning to RapidMiner. They will be introduced throughout this course.
- You can assign custom roles to attributes. Custom roles don't imply special treatment; Attributes with custom roles will simply be ignored.

Aha! A solution to our problem with rowNumber, which we want to keep in the data set, but not use for model creation. We simply assign a special role to it.

Again: The Label Attribute

Apart from ignoring attributes, the special roles have a crucial importance for supervised learning tasks and model creation. We have expended a lot of effort to generate the Churn attribute as our target variable. But how does RapidMiner know that we want to learn a rule for churn and not for the Age, Gender, or another attribute in our data?

Again, we need roles. If we assign the predefined label role to an attribute, RapidMiner then knows that this is our target variable. The modelling algorithms will try to find rules to guess the label attribute from the other attributes – more precise from the regular attributes, as other special attributes apart from the label will be completely ignored.

The Set Role Operator

We use the Set Role operator to assign roles.

- In the Parameter panel you select the attribute and define the role.
- To define a custom role, type a name in instead of selecting one from the list.
- To remove all special roles from an attribute, select regular from the dropdown list.
- To assign multiple roles using only one instance of the Set Role operator, use the set additional roles parameter.

Exercise 8. Role Assignment

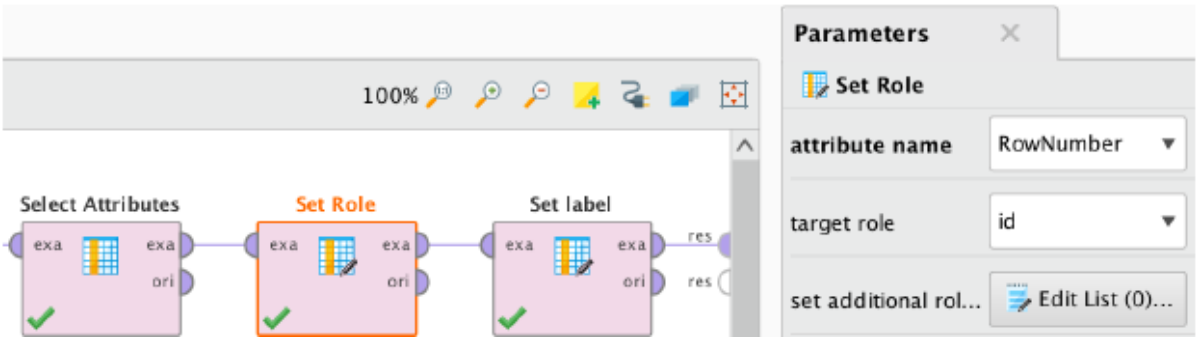
Assign special roles to the data as needed. That is, make RapidMiner know of our analysis target and exclude all irrelevant attributes from the analysis.

Answers to Exercise 8

- Make RapidMiner ignore the rowNumber attribute by assigning either the predefined id role or a custom role.
- Assign the label role to the Churn attribute.

To complete this task, you either need two Set Role operators or you can use the set additional roles parameter. In the latter case, assign the first role with the attribute name and target role parameters; use the Edit List dialog to assign the second role.

See figure below for a possible solution. In that figure, the second Set Role operator has been renamed to Set label to provide a better overview.

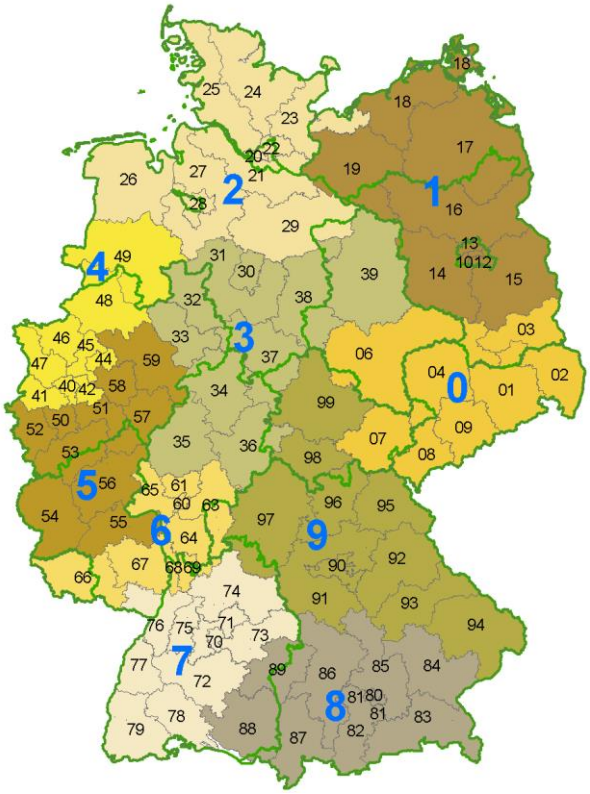


Challenge: A Better Representative of the Postal Code

Let's reinspect the original data set. In the statistics view, we saw that the postal code is interpreted as an integer attribute, i.e., as a number.

Does this make sense? In Germany, the city of Munich has postal code 80331. Hamburg's code is 20095. If we interpret the postal code as a number, that means that Munich = 4*Hamburg. It's pretty clear that it does not make sense to do that – the postal code represents categories, not ordered numbers. In RapidMiner, categories are stored as nominal attributes. So we need a method to convert the attribute from numerical to (poly-)nominal.

Furthermore, if you closely examine the data again you will find an additional problem: What does the postal code with an integer value of 1 represent? It is in fact 00001, but we are missing the leading zeros. However, if we salt the soup with some domain knowledge, we can find a solution for all our problems. Look at figure below, which shows the geographical postal code distribution in Germany. Does this trigger an idea?



The German postal codes are physically grouped by their first digit. We can leverage this fact by using only that first digit to reduce the number of categories from 100, 000 to 10!

To do so, pull out RapidMiner's Swiss army knife – Generate Attributes operator! This operator offers all necessary functions to both add leading zeros and also select the first digit only. Let's do it in two steps, starting with converting it to nominal and adding leading zeros. For this we will use three functions:

1. **str(<number>)** transforms a number into its string representation. This will take the place of the Numerical to Polynominal operator (although we won't use right now, it could come in handy to you later!).

2. **suffix(<attribute>, <number of characters>)**. It delivers the last n characters of a string. For example:

suffix("Hello RapidMiner!", 6) → "Miner!"

3. **concat(<string1, string2, ...>)** concatenates a series of strings. For example:

concat("Hello ", "RapidMiner!") → "Hello RapidMiner!"

The next step is easier, we need only take the first digit of the postal code. Can you guess what function could come to our service for this task? You guessed right, it is the **prefix(<attribute>, <number of characters>)** function that we learnt earlier! It delivers the first n characters of a string. For example:

prefix("Hello RapidMiner!", 5) ! "Hello"

With this information, you are ready for the challenge.

Answer for the Challenge

1. Convert the PostalCode attribute to its string representation.

   - The PostalCode attribute can be converted with the str function inside the Generate Attributes operator. We will continue to work inside this operator in the following steps.

2. Concatenate a string of leading zeros at the beginning of the postal code. For example, "1" becomes "00001".

   - The concat function will work for this task. You can represent leading zeros by the string "0000".

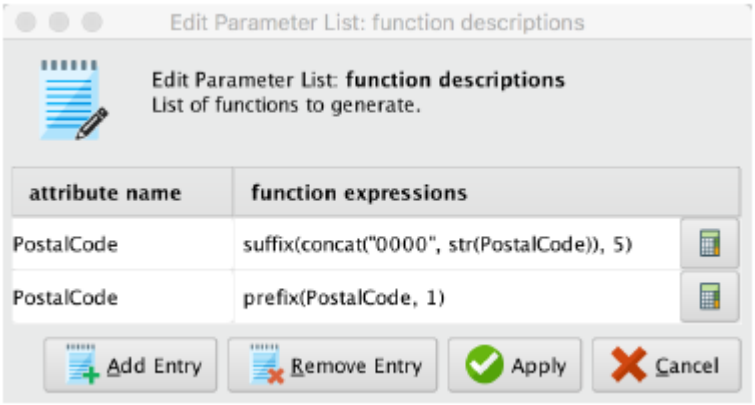
3. Take the last five digits of the result of the previous step (that way we end up only with five-digit strings).

   - Now it's the turn of the suffix function. You need to select the last 5 digits.

4. Overwrite the original PostalCode with the first digits of the five-digit strings from the last step. For example, "00001" becomes "0".

   - The first digit of the postal code is obtained through the function prefix.

   - Remember that you can overwrite an attribute by supplying its name as attribute name in the left column of the Edit Parameter List dialog.



2.7 Saving the Process

So far we have invested a lot of time and effort in implementing all the ETL steps. It's time to save the process. You have two options: the standard way and the smart way.

2.7.1 The Standard Way for Saving Processes

The standard way to save a process is the way you are used to from other software and what we have used for our first process. That is, use the Save button or the Save Process entry in the File menu. This method works in RapidMiner and will open the Save dialog – see Figure 2.15. The dialog is a bit different from those that you know from other programs. RapidMiner processes are not stored (directly) on disk, so you don't see your filesystem. Instead, they are stored in the repository, and that is what the dialog shows. Navigate into your RapidMiner Training repository, and open DE Professional folder. In there, select processes. Finally, enter load and prepare data into the Name field.

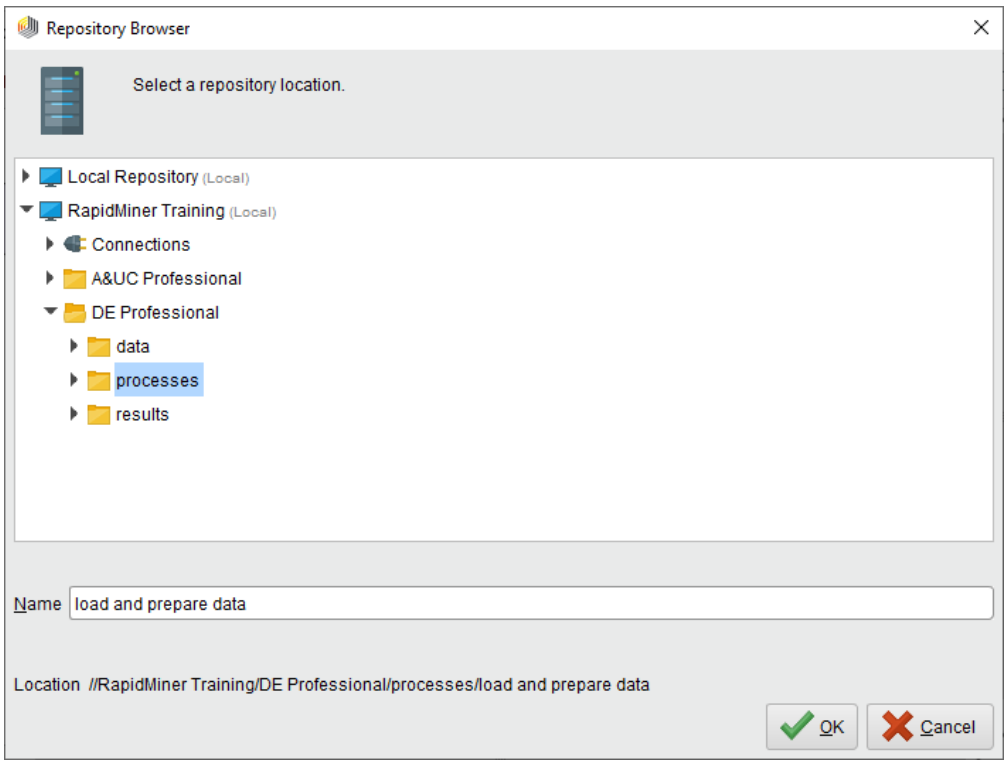


Figure 2.15 The Save Dialog

This is the filename under which the data preparation process will be stored in the processes folder. Press OK. The process appears in the Repository panel – compare Figure 2.16.

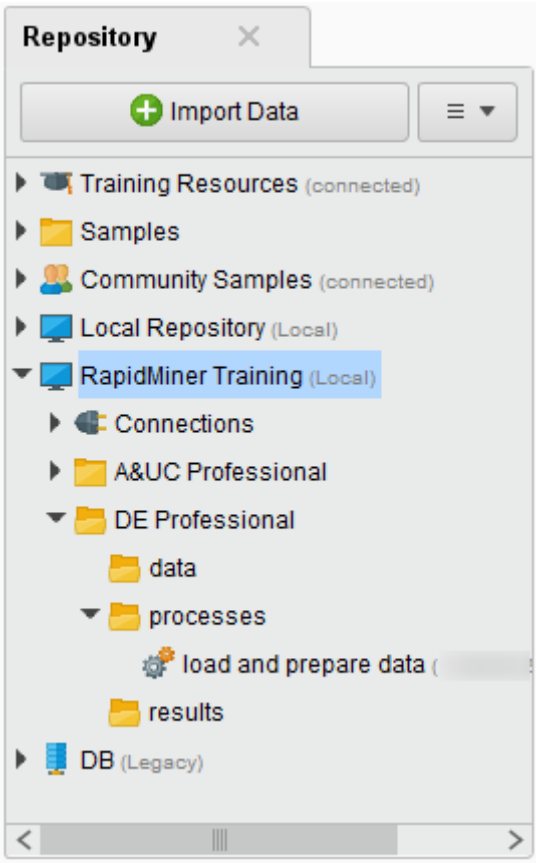


Figure 2.16 Repository after Saving the ETL Process

2.7.2 The Smart Way for Saving Processes

The second way of saving a process does not only save the process but also saves a few clicks. In the Repository panel simply navigate to the folder where you want to save your process, right-click and select the top-most entry Store Process Here – see Figure 2.17. In the resulting dialog, enter the name of the process. That's it! The effect is the same as if you had used the standard way of saving.

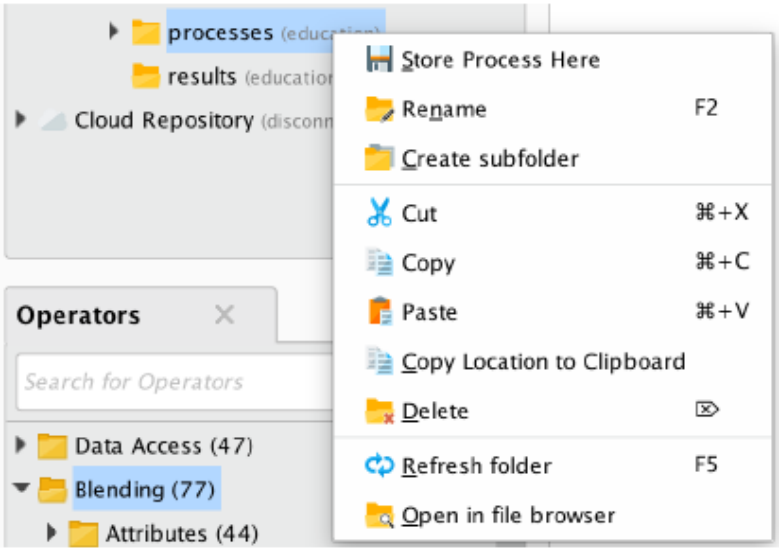


Figure 2.17 Storing a Process with a Right-click in Repository

Don't forget to save.

Have you ever attended a fitness course? Then you know the mantra for all exercises: Don't forget to breathe. There's also a mantra for working with digital content: Don't forget to save!

Save your processes frequently, especially before performing any major changes. If you save before a change, it assures that you can go back to the state before the change, in case something goes wrong.

Once you have checked that everything works as expected after the change, save again to capture the change, under a new name if you want to keep track of the changes.

In the beginning of this course, we mentioned RapidMiner Server and Server Repositories. Even though those are beyond the scope of this document, let's not forget to mention that with RapidMiner Server you get a fully-featured version control system, which allows you to switch back and forth between any version of the process that you have created, now or in the past.

2.8 Polishing the Process Appearance

By assigning roles, we have applied the final touch to our data in preparing it for the first model generation. So now that the process does what we want, let's clean up its appearance and structure. RapidMiner gives us several methods for accomplishing this, which we'll discuss in the following paragraphs.

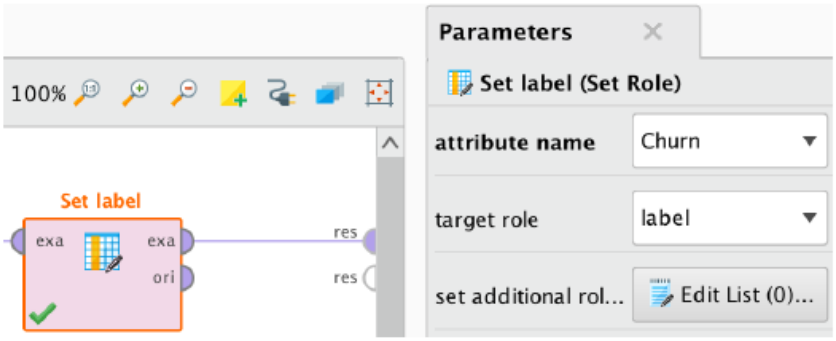
2.8.1 Renaming Operators

One super helpful cleaning task is to rename operators such that their name states clearly what their purpose is. We could, for example, rename the Read Excel operator in our process to "Read Customer Churn Data", Filter Example to "Filter on Age", etc.

To rename an operator:

- Left-click the operator and press F2, type in the new name and press Enter.
- Right-click the operator and select Rename, type in the new name and press Enter.

Even after renaming an operator, you can always find its original name in the Parameters panel. Each operator (re)name displays with the original name next to it, in parenthesis. Also, the operator's colour and icon indicate its type.



2.8.2 Subprocesses: Grouping Related Operators

Often you need more than one instance of an operator to perform one logical task. In our process, for example, we need two Replace operators to clean the Gender attribute. With the Subprocess operator you can group the operators. Picture a subprocess as a container holding several operators. When the Subprocess operator is executed, it passes its input to the first connected operator on its inside, executes the inner operators, and passes the output of the last executed operator to its own output port. For this to work, it is important to connect the inner ports of the subprocess correctly to the inner operators – see Figure 2.18 for an example.

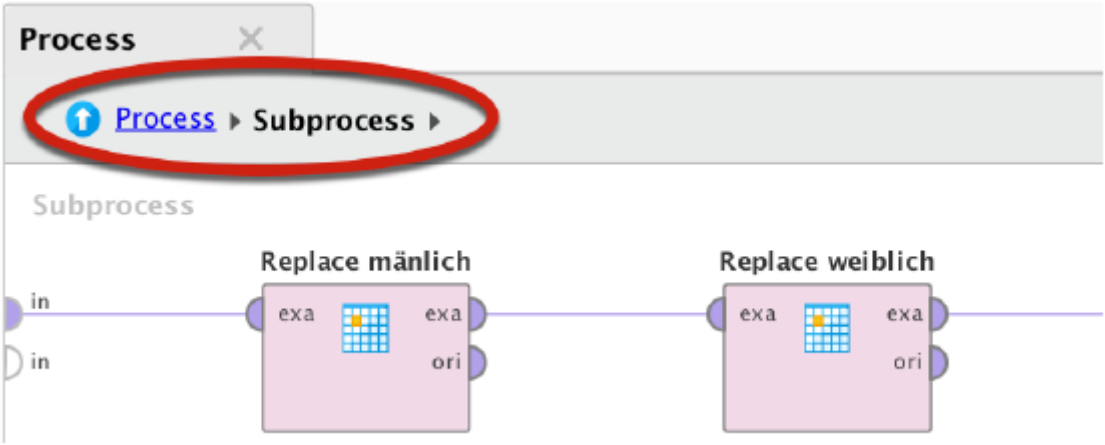


Figure 2.18 The inner process of a Subprocess operator

Navigation

Subprocesses are similar to folders in a file system – you can enter them and then

go one level up. To enter a subprocess, simply double-click the operator. To go up one level, you have several options:

- Click the arrow-up icon in the process header (marked in red in Figure 2.18). Alternatively, press Backspace.
- Click the name of the parent operator in the breadcrumb navigation.

2.8.3 Creating a Subprocess

To use the Subprocess operator you could add it to the process and select, cut and paste the operators you want to move into the subprocess. Then you would have to reconnect the operators in the process and in the subprocess. As these are way too many steps, there is an easier method: Select the operators, right click and choose Move into new subprocess – see Figure 2.19. This leaves you with the same result as the path described before in just two steps.

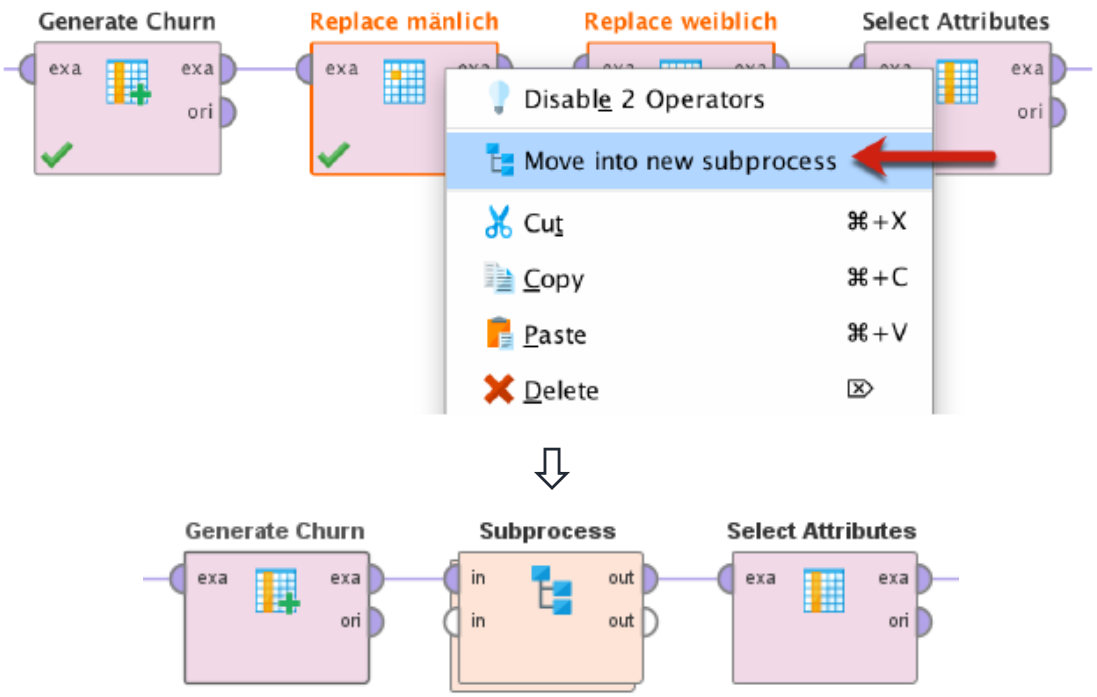


Figure 2.19 Moving the selected operators into a new subprocess

Subprocesses are an important tool to keep the process structure neat and clean. They allow you to group logical operations, and you can even create deep hierarchies (subprocesses in subprocesses in subprocesses). While that may seem complicated, it allows you to hide complex operations from the parent process. That is, on the top level there is only one subprocess operator with a defined output instead of a complicated mess of a dozen interconnected operators. This is especially helpful if you rename your subprocess to a logical name that summarizes what's going on inside.

2.8.4 In-Process Documentation

In the previous section, we practiced a few ways of cleaning the process structure and making it easier to "read". Nevertheless, some explicit documentation is needed to make your processes understandable by others.

Specially for subprocesses, it makes sense to write a brief description about what they do, and guess what? RapidMiner supports this – right in the process! You can write notes describing its function and of its operators.

Documentation for Process

To add a note to the process, you can do any of the ways below:

- Right-click at any blank space in the Process panel → Add note
- Double-click at any blank space in the Process panel
- Click the Insert Note icon at the right upper side of Process panel 

A note will then be shown as in Figure 2.20. You can now write a brief summary of what your process does, so if you return to your process after a few weeks you don't have to go through all the operators to know what your process is doing.

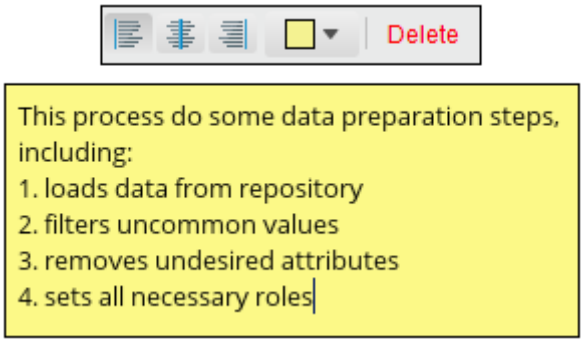


Figure 2.20 Add Note to Process

Documentation for Operators

If you want to change some things in your process, it is essential to know what your operators do. Renaming the operators helps, but sometimes there are operators which have to be commented in detail. You can do this by

- Starting a new note through right click at the operator → Add note
- Dragging an existing note and dropping it on the correspondent operator, so they get attached

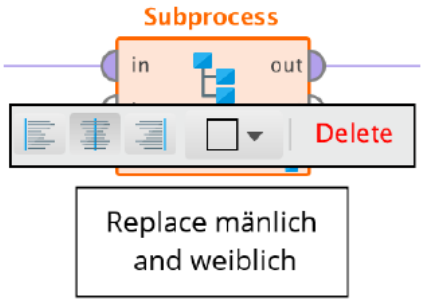


Figure 2.21 Add Note to Operator

To detach a note from operator,

- Right-click on the operator and choose Detach note from operator
- Simply drag the note again and remove it from the operator

If you have to comment a lot of operators it can get confusing. To avoid chaos, you can choose to hide/show the notes by doing a right click → Show all notes. When the notes are hidden you can still recall which of the operators has one attached by the small note symbol at the lower right corner, as shown in Figure 2.22.

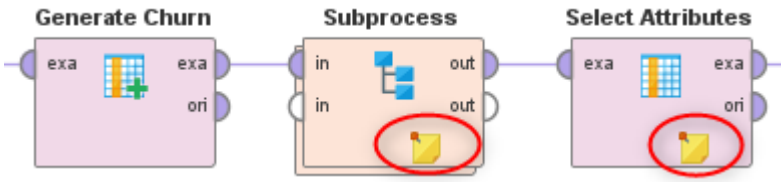


Figure 2.22 Small Note Symbol at Operator after Hiding Notes

Exercise 9. Process Structure and Documentation

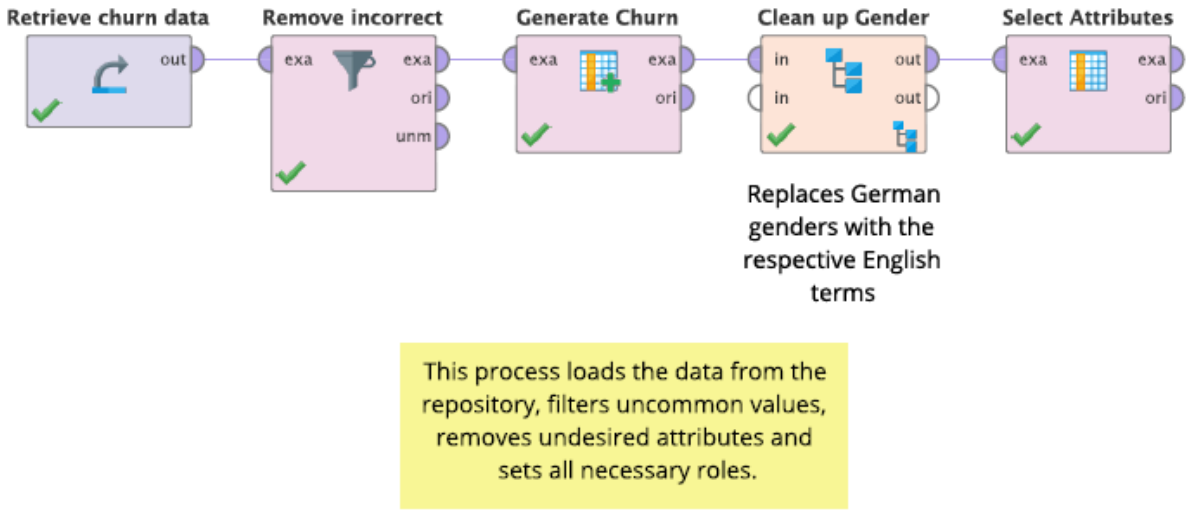
Polish the appearance of your process. Make use of everything you have learned in this section:

- Rename operators
- Group them into subprocesses
- Provide documentation on operator and process level

Don't forget to save the process when you're finished!

Answers to Exercise 9

Your process could look similar to:



3. What We Did and Where We Are?

In this section we learned how to get data from a common Excel file into RapidMiner. We then completed the basic data preparation, which included filtering out incorrect data, handling missing values, modifying attribute values in the Gender attribute, and deriving a new column from an existing one while we generated the label.

Next, we introduced attribute roles to tell RapidMiner about the semantics of special attributes (such as the label). We learned that special roles must be unique, and that special attributes are ignored by any further processing – apart from the label, which is obviously needed for model generation.

We finished with a clean data set that is ready to be used for deriving analytical models, i.e., to find relations between the regular attributes and the label. We will learn how to do that in the Machine Learning Professional.

In the next section, we will continue with advanced data preparation (or advanced ETL), such as how to handle multiple data sets, aggregations and so on. Let's start explore more!

4. Greetings From The Big Bad World – More Raw Data Sets

Remember the CRISP-DM cycle from the Applications and Use Cases Professional course? Look at Figure 4.1 for a refresher.

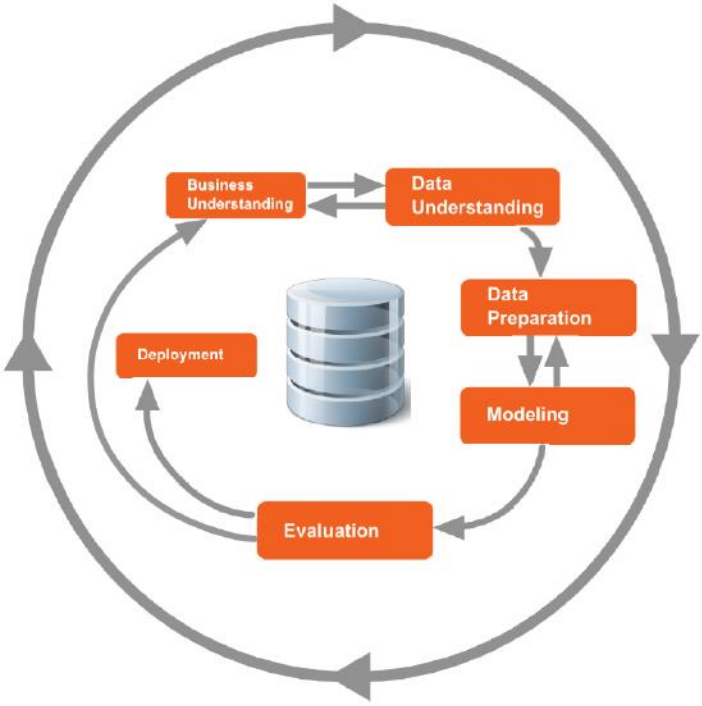


Figure 4.1 The CRISP-DM Model

In the last course you completed one full CRISP cycle. You learned about our business – a web store, where customers create accounts and can then order products. They either pay with credit card, cash, or cheque. When a customer deletes his account, we say he has churned. Because we want to reduce customer churn, we need to identify customers who are likely to churn in the near future. This scenario remains unchanged for the next sections.

However, the data that we are going to use has undergone a major change. In the previous sections, IT crafted for us a nice, self-contained table that included all the necessary information. It was great for us, but for them, it was a lot of effort that they are not willing to commit to our data science project. As the project goes on (in the next few sections of this course), we need to collect the data ourselves, from several data sources, and combine it into a single table.

While this scenario may sound a bit synthetic, it's actually quite a common use case. That is, it often happens that we get new insight into the data, or the data changes, and so we have to iterate once more through the data understanding part of the cycle. The next section guides you through this step. The remainder of this course adjusts and extends the earlier pre-processing and ETL to suit the new data sets given.

4.1 The New Data Sets

Take a look at Figure 4.2 on the next page for a list of the data – 5 Excel files – for this section. Before you can do anything with it, you first need to import it into RapidMiner. After that's accomplished, we'll look at the files one-by-one to understand the data it contains.

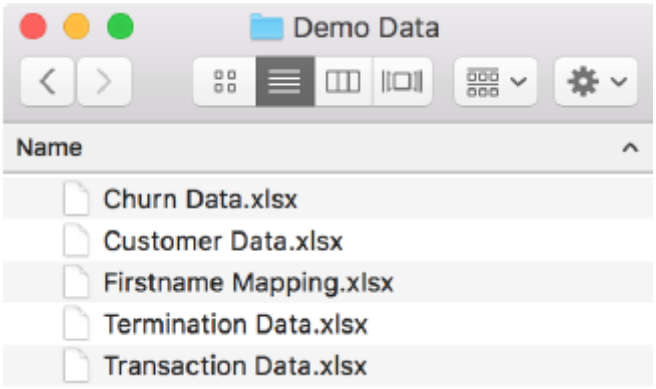


Figure 4.2 The data is spread over several Excel files

4.2 Loading Several Data Sets in One Go

In the first part of the course, you learned how to import Excel data. You could use the same technique here, but since you have 5 different files the process would be a bit cumbersome and repetitive. Usually, we want to keep our processes DRY, which stands for don't repeat yourself. So, what else can you do?

Here's a simple outline of the plan. For each Excel file in the data folder, you want to:

1. Load the file into RapidMiner with Read Excel
2. Store the file in the repository with Store

Sounds pretty simple, right? You know how to accomplish each step, but how do you do something for all files in a folder at once? In the remainder of this section you will learn the process for importing all files at once.

Let's rephrase our problem as loop over all files in the folder and do something. This can be accomplished by the Loop Files operator – Figure 4.3 shows how it's configured.

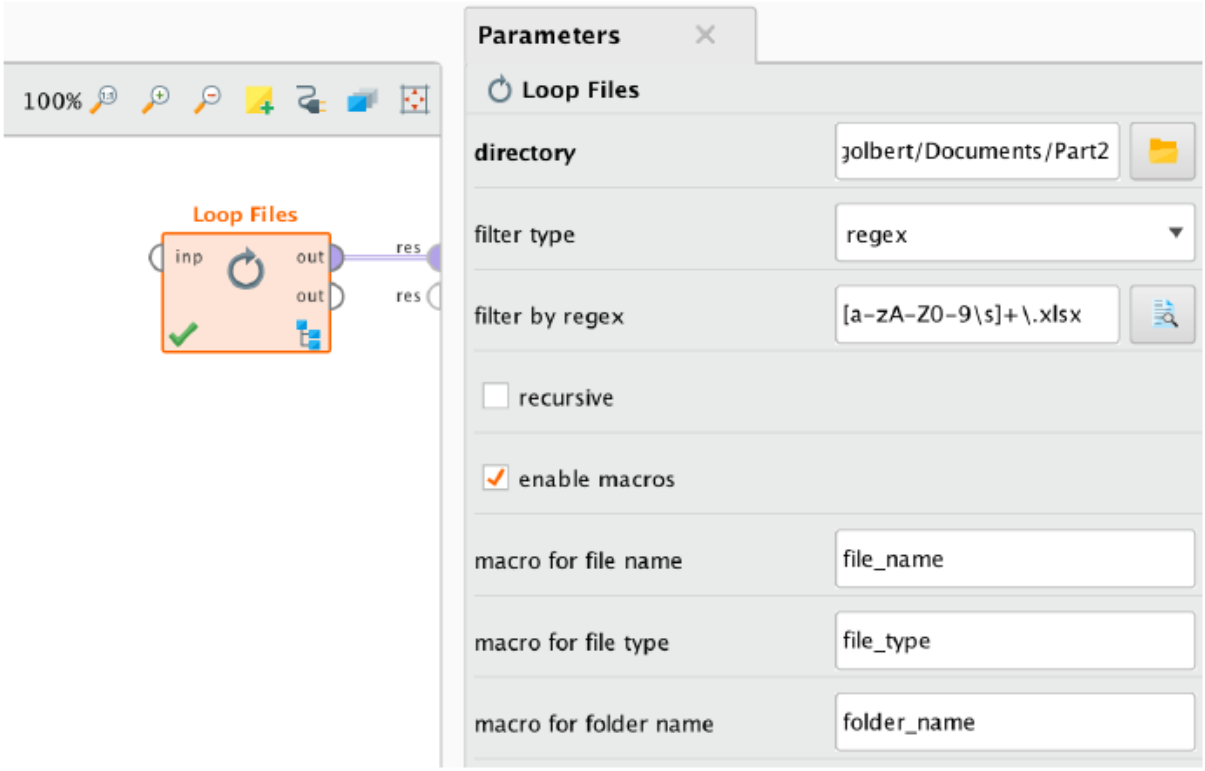


Figure 4.3 The Data Import Process

First, select the directory containing the files to load. By default, Loop Files iterates over all files in the selected folder and its subfolders. While this may, at first, seem OK for our use case, it can be a great problem if the folder has other files (for example temporary files), or if you only want to use a subset of all files present.

To prevent Loop Files from using these files, you need to provide a pattern that describes the desired files. This is done with the filter parameter by using a regular expression. Regular expressions are a powerful means of describing, matching, and searching strings. Although they can be very powerful, finding specific patterns across thousands of lines of text, we will use them only for simple cases.

4.2.1 Scratching the Surface of Regular Expressions

To describe a string pattern with a regular expression, you first have to compose the description in your mind. For filenames it could be as follows: an Excel filename is a string of one or more uppercase or lowercase characters, numbers and spaces followed by a dot and “xlsx”. You will soon devise a regular expression that meets this condition.

You can try out everything as we go: in the Loop Files’s Parameters panel, click the icon with the looking glass in the filter parameter. This opens the Edit Regular Expression dialog, as shown in Figure 4.4. You can insert a regular expression in the first input box and some test strings into the Text field. Any part of the input that matched the regular expression is marked in yellow. Ignore the other input fields for now.

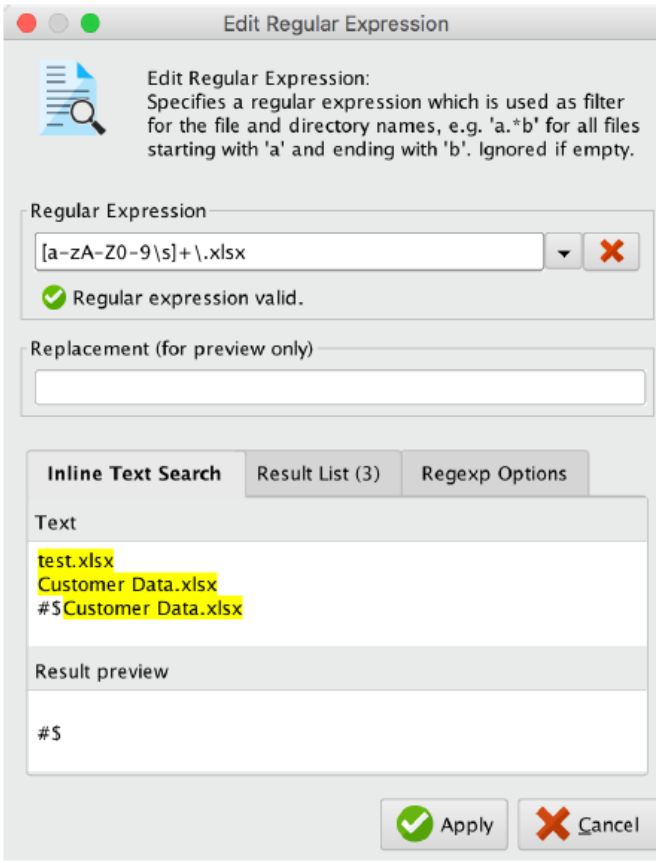


Figure 4.4 The regular expression editor

With the regular expression in place, Loop Files only matches filenames that are completely matched by the expression.

Let's start at the beginning and first describe one lower case character:

```
[a-z]
```

The square brackets in a regular expression mean "one of the enclosed characters," and a-z reads "a to z." So this expression matches one appearance of an arbitrary lower case character. By adding a plus sign, you extend this to "one or more of the enclosed characters":

```
[a-z]+
```

This expression matches a string of one or more lower-case characters, e.g., "hello" or "customer," but not "Customer" (with an upper-case C). To also match upper-case characters and digits, simply add them to the brackets:

```
[a-zA-Z0-9]+
```

This expression matches "Churn" and "Customer," but not "Customer Data." To include the latter, you need to add whitespace to the expression. In a regular expression, whitespace is written as \s:

```
[a-zA-Z0-9\s]+
```

Now the expression matches "Customer Data" but not "Customer Data.xlsx" (because of the dot between "data" and "xslx"). Since ".xlsx" is always the same for all relevant files, you can simply add it to the end of the expression. But be careful, the dot has a special meaning in regular expressions and hence must be preceded (or escaped) with a backslash:

```
[a-zA-Z0-9\s]+\.xlsx
```

This is the final expression. It matches all filenames that contain a combination of upper-case and lower-case characters, digits, and whitespace and are followed by ".xlsx". It does not match anything that contains "~" or "$" (since those characters are not included in the square brackets). Hence, the system files are ignored.

4.2.2 The Inner Process of Loop Files

Now that you know how to configure the parameters of Loop Files, take a look at its inner parts. Remember, the inner (nested) process is executed for each file in the selected folder that matches the filters.

You want to load Excel files, so obviously you need a Read Excel operator. But, unlike the previous training course, this time you do not run the import wizard. Instead, leverage the file (fil) port of Loop Files. (You'll see the port once you open the inner process.) Loop Files provides the current file at this port. You can pass it to Read Excel, which will do what it does best – read the contents of the Excel file as RapidMiner data. Figure 4.5 shows the setup. Note the connection from the file port to Read Excel.

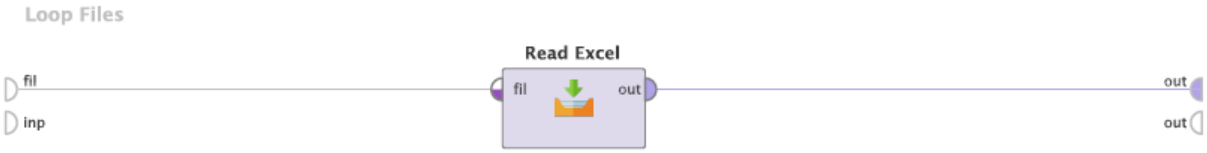
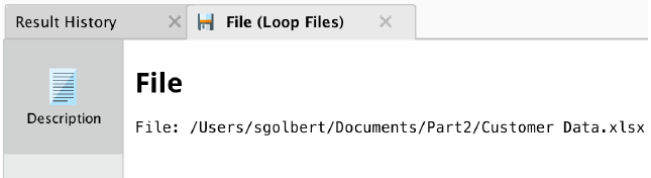
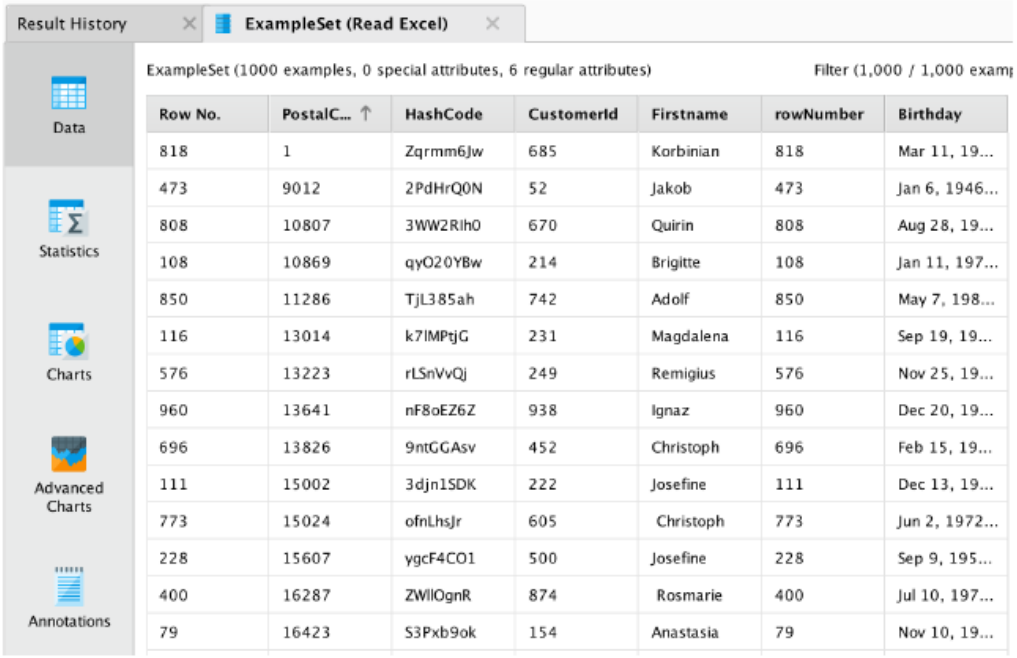


Figure 4.5 Read Excel inside Loop Files.

If you set a breakpoint before and after Read Excel and run the process, you can see both the file object that goes into the operator and the read data – see Figure 4.6 (a) to (c).

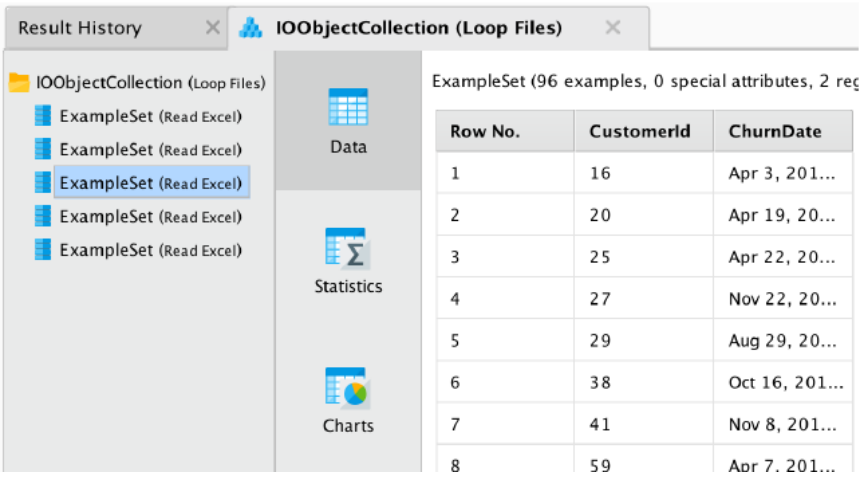


(a) The File Object



Row No.	PostalC...	HashCode	CustomerId	Firstname	rowNumber	Birthday
818	1	Zqrmm6Jw	685	Korbinian	818	Mar 11, 19...
473	9012	2PdHrQ0N	52	Jakob	473	Jan 6, 1946...
808	10807	3WW2RIh0	670	Quirin	808	Aug 28, 19...
108	10869	qyO20YBw	214	Brigitte	108	Jan 11, 197...
850	11286	TjL385ah	742	Adolf	850	May 7, 198...
116	13014	k7lMPtjG	231	Magdalena	116	Sep 19, 19...
576	13223	rLSnVvQj	249	Remigius	576	Nov 25, 19...
960	13641	nF8oEZ6Z	938	Ignaz	960	Dec 20, 19...
696	13826	9ntGGAsv	452	Christoph	696	Feb 15, 19...
111	15002	3djn1SDK	222	Josefine	111	Dec 13, 19...
773	15024	ofnLhsJr	605	Christoph	773	Jun 2, 1972...
228	15607	ygcF4CO1	500	Josefine	228	Sep 9, 195...
400	16287	ZWllOgnR	874	Rosmarie	400	Jul 10, 197...
79	16423	S3Pxb9ok	154	Anastasia	79	Nov 10, 19...

(b) The customer data as a standard example set



Row No.	CustomerId	ChurnDate
1	16	Apr 3, 201...
2	20	Apr 19, 20...
3	25	Apr 22, 20...
4	27	Nov 22, 20...
5	29	Aug 29, 20...
6	38	Oct 16, 201...
7	41	Nov 8, 201...
8	59	Apr 7, 201...

(c) The final result – a collection

Figure 4.6: Results of Read Excel on the breakpoints before and after the operator, and the final collection result

The file object that is passed from the file port to Read Excel is visualized as the filename. After processing by Read Excel, you get the data as an example set. As the process continues to run, notice that in the subsequent iterations the Churn Data.xlsx, the Firstname Mapping.xlsx and the remaining files are processed. In the very end,

when Loop Files has processed all files, the result is a collection, as shown in Figure 4.6 (c): a list of data objects that are delivered at one single port. In the process, a collection result is indicated by a double connection line at the output port of Loop Files.

In the collection result you can click the 5 different entries on the left-hand side; the data on the right updates accordingly. Obviously each entry contains the data of one Excel file in the data folder on disk. That means that you now have all the data imported into RapidMiner – but what to do with it? As long as it's in the collection, you can't really do anything. Even though there are means of extracting a single element from the collection, we don't know which element represents which file. Instead, you'll store the single data sets in the repository.

4.2.3 Storing the Data – and an Intro to Macros

You already know what you need to store data – the Store operator. You probably also can guess the correct location for the operator in this process. Since you want to store the single entries (and not the collection), Store must be put into the loop after Read Excel. So now the question is, what is the correct value for the filename (the name to save the file in the repository)?

Imagine you enter Churn Data, since the first file you load is just the Churn Data.xlsx file. Would that work?

Nope. In the first iteration, the process loads Churn Data.xlsx, and stores the data in the repository as Churn Data. So far so good, but what happens during the second iteration? Now Customer Data.xlsx loads, and the process stores it in the repository as – Churn Data, overwriting the entry from the first iteration.

In the third iteration, the process overwrites the entry again with the next file, and so it goes. In the end, you'd have an entry called Churn Data containing just the data of the last file.

Definitely not what you are aiming for. What you really need is something to set the value of the repository entry parameter of Store dynamically to the current filename. This is where **macros** come into play.

Generally speaking, in RapidMiner a macro is a global variable that can be used in all parameters of all operators. You could, for example, define a macro named file_name and give it the value Customer Data. Then, instead of writing Customer Data directly into the parameter of Store, we can use the macro by typing ../data/%{file_name}.

Now, when the Store operator is executed, the following happens:

1. RapidMiner searches for references to macros in the parameters (in the form **%{macroName}**)
2. All macros are replaced with their current value
3. The operator is executed as usual

If you assign the value Customer Data to the macro and use it as described in the Store operator, RapidMiner stores the data as Customer Data in the data folder of the repository. If you change the value of the macro, then you would also store the data under a different name.

So, to store each input file under its respective name, the only thing missing is a means of changing the macro value to the current filename. Luckily, Loop Files does this job for us! Once we click on the enable macros option, it automatically creates the file_name macro and ensures that its value contains the name of the current file.

This is best seen if you enable the Macros panel in the View menu. If you then run the process again and ensure that you still have the breakpoint on Read Excel you'll see that there are three macros defined:

- file_name – the name of the current file, without the path
- file_type – the extension of the current file
- folder_name – the path of the current file, without the filename

As you step through the process, you'll see in the macros panel how the macro values adjust to the current file. Additionally, you can see the data entries appear, one by one, in the data folder of your repository.

The final result is still a collection, but that's not a problem – you can now easily access the data from the repository.

You will be using macros all the time, not only with RapidMiner Studio but also with RapidMiner Server and Radoop. Therefore, it is a good idea to take the time and play with them a little before advancing on the course.

 How and where to add the Macros panel. You learned in the first part of the course how to add a panel, i.e., via the View → Show Panel menu. You added panels to only one view though, for example, the Tree panel to the Design view. You can add the same panel to several views. This is a good thing to do for the Macros panel, since the macros are needed both during process design and are also as a kind of result. Therefore, first add it to the Design view, e.g., next to the Help panel. When you switch to the Results view, it's a good idea to add it there.

Exercise 10. Data Import with Loop Files

It's time to get your hands dirty with data import and file looping.

1. Create a new process and store it as import all data.
2. Configure the process to load all Excel files in the data folder supplied with this document. You will need everything you have learned in this chapter:
   (a) Add Loop Files and connect its output to the process output.
   (b) Configure the directory.
   (c) Configure the filter.
   (d) Activate enable macros.
   (e) In its subprocess, add Read Excel and connect it to the file port.
   (f) Add a breakpoint on Read Excel and run the process. Inspect the results.
   (g) You should add the Macros panel to both the Design and the Results views to observe how the macros behave.
   (h) Add the Store operator after Read Excel.
   (i) Configure Store's parameter, leveraging the file_name macro.
   (j) Run the process again.
3. Once you're done, make sure that you have all 5 Excel files imported into the data folder in your repository.

Answer to Exercise 10

Refer to Section 4.2 for steps.

Now that you finally have the data loaded, let's have a look!

4.3 Again: Data Understanding

The new data forces us to take another trip through the CRISP-DM cycle, bringing us back to the data understanding phase. In this section we'll look at all the newly loaded data sets to understand their meaning and their relations. In the next section we'll devise a master plan for the data pre-processing phase.

Your repository should now look similar to Figure 4.7. You can double-click the data entries to open them in the Results view. Customer Data looks familiar, so let's start!

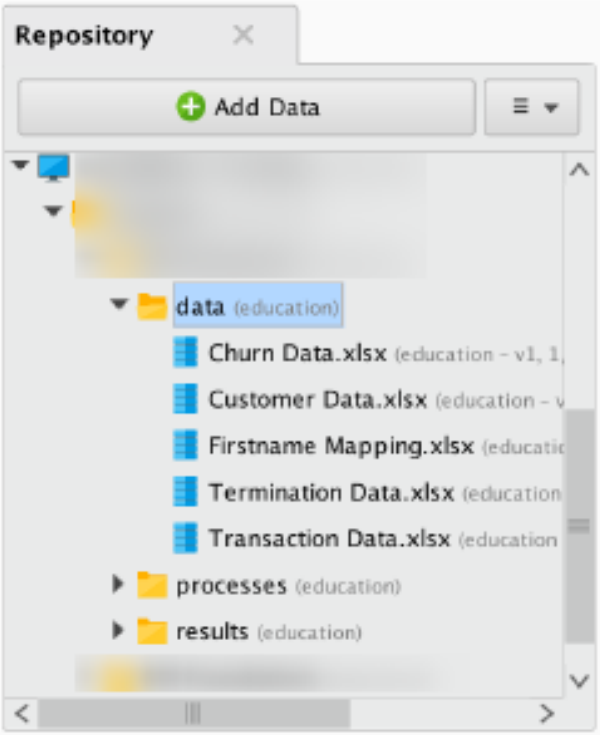


Figure 4.7 Data loaded in the repository

4.3.1 Customer Data

Figure 4.8 shows the first 15 lines of the customer data. It looks vaguely similar to the data set from the last course – obviously we are dealing with the base data of customers.

Row No.	PostalCode	HashCode	CustomerId	Firstname	rowNumber	Birthday
1	87213	tKIbadnh	3	Eva	1	Apr 13, 19...
2	38548	RcW2Pb3w	4	Kunigunde	2	Jun 21, 197...
3	44573	akWNQl4e	6	Notburga	3	Jun 5, 1985...
4	70936	glrPDLzY	7	Maximiliane	4	Jul 26, 199...
5	49705	3rGP8X98	9	Dorothea	5	Aug 4, 194...
6	42376	XlxhfOlo	12	Maria	6	Jul 29, 196...
7	52245	3ANQ9shn	16	Rosina	7	Jun 28, 196...
8	56625	BDEPLKmG	17	Susanne	8	Apr 14, 19...
9	66713	gQBDZ8Lw	20	Genoveva	9	Mar 31, 19...
10	70052	G1c0kZqQ	21	Katharina	10	Nov 16, 19...
11	44272	IJC8cDTW	22	Marlene	11	Jun 29, 197...
12	71962	n5YVLJ7v	23	Annemarie	12	Sep 22, 19...
13	45355	HVtuFvKk	25	Magdalena	13	Feb 18, 19...
14	59736	LJVy8PnW	26	Waltraud	14	Sep 20, 19...

Figure 4.8 Customer Data

Just as before, we have the PostalCode, the HashCode and the rowNumber. We do not have the age or the gender anymore – instead there's Birthday and Firstname. In the next sections we will infer the missing information. Obviously it is possible to calculate customer age from a birthday, and the first name can be an indicator of gender. There's also a new column called CustomerId. You'll see shortly how to use this column.

What's definitely missing is the churn information. But wasn't there a Churn Data table in the imported data? (Hint: yes.) Let's open it.

4.3.2 Churn Data

As you can see in Figure 4.9, the Churn Data contains just two columns – the CustomerId and the ChurnDate.

Row No.	CustomerId	ChurnDate
1	16	Apr 3, 2012 6:54:05 AM CEST
2	20	Apr 19, 2013 5:44:51 AM CEST
3	25	Apr 22, 2013 2:22:57 AM CEST
4	27	Nov 22, 2013 4:32:39 PM CET
5	29	Aug 29, 2011 8:00:12 AM CEST
6	38	Oct 16, 2013 3:44:53 PM CEST
7	41	Nov 8, 2010 3:52:05 AM CET
8	59	Apr 7, 2012 4:07:46 AM CEST
9	61	Oct 8, 2012 6:33:11 AM CEST
10	67	Dec 2, 2013 4:14:21 AM CET

Figure 4.9 Churn Data

It's probably time to talk about the CustomerId in more detail now. The "Id," short for identifier, uniquely identifies each customer (thus, CustomerId). In other words, there are no two customers with the same ID. Even if two customers have the same name and/or birthday, they are still distinguishable by their ID.

ID has some additional advantages. It's faster to compare a single number than the several columns that make a customer unique (e.g., name, birthday, address). While those column attributes may change over time, the ID remains the same per person.

In this case, you can leverage the customer ID to match rows from the Churn Data to a customer in the Customer Data. In that way you can join the churn date to the customer base data: a customer is listed in the churn data with a churn date if, and only if, he already churned. We'll talk about joins in more detail later in this chapter. For the moment, just remember this relation between the two tables:

- Each row in the Churn Data matches exactly one row in the Customer Data
- A row in the Customer Data has 0 or 1 matches in the Churn Data.

4.3.3 Firstname Mapping

In the previous section we discussed the idea of joining data from one table to another, specifically the churn data to the customer data.

You can use the same technique to join gender information to the customer data. Look at the Firstname Mapping data in Figure 4.10. It contains a list of German names and their associated gender.

Row No.	Firstname	Gender
1	Adolf	m
2	Alfons	m
3	Alois	m
4	Andreas	m
5	Peter	m
6	Rupert	m
7	Bartholomäus	m
8	Sebastian	m
9	Benedikt	m
10	Joseph	m
11	Bernd	m
12	Albert	m

Figure 4.10 Firstname Mapping

Since you have customer names, you can use Firstname Mapping to enrich the customer base data with gender information. To do so, obviously, you do not match the rows via the CustomerId attribute, but via the Firstname attribute.

Now we know where to get the data for all fields seen in the last course except the LastTransactionDate. Where, if not in the Transaction Data, could it be?

4.3.4 Transaction Data

Look at Figure 4.11. The Transaction Data contains customer transactions. A transaction is any kind of purchase – each time a customer buys something via the app, a transaction including the price of the product, the payment method, and the date is added to the database.

ExampleSet (4334 examples, 0 special attributes, 4 regular attributes)

Row No.	CustomerId	Transactio...	PaymentMe...	Date
1	1	11.508	credit card	Apr 17, 2012 2:05:39 AM CEST
2	2	330.070	cheque	Nov 25, 2011 6:58:02 AM CET
3	24	332.387	cash	Jun 23, 2010 3:58:56 AM CEST
4	27	365.862	credit card	Jan 25, 2012 12:02:22 PM CET
5	31	194.795	cash	Dec 9, 2010 7:08:47 PM CET
6	36	80.341	cash	Sep 27, 2010 9:16:16 AM CEST
7	46	223.609	cash	Mar 31, 2011 12:13:57 AM CEST
8	48	144.298	cash	Oct 4, 2011 10:23:30 PM CEST
9	54	85.005	cash	Feb 4, 2011 6:10:59 PM CET
10	55	104.832	cash	Jun 30, 2010 9:13:02 PM CEST

Figure 4.11 Transaction Data

Using the CustomerId, you can link the transaction data to the customer data.

Look at the example counts. The customer data contains 1,000 entries, the transaction data 4,334 rows. This indicates that a customer can have multiple transactions. This makes perfect sense, of course, since the hope is that customers make purchases at your shop more than once.

This one-to-many relationship has some implications on the data handling, but we will discuss that later in this chapter.

4.3.5 Termination Data

The Termination Data looks a bit strange. It's just a single column with customer IDs. Because you want to create models that detect churn, your training data should contain both customers that are still active and those who have cancelled their subscriptions. But there's a third group as well – those that did not actively churn by cancelling their subscription, but whose subscription term ended. Customers to whom this happened are listed in the Termination Data; you will remove them from your training data later.

4.3.6 Wrapping Up: The Relation Diagram

We have talked a lot about the relationships of the data, but a picture (Figure 4.12) is worth a thousand words.

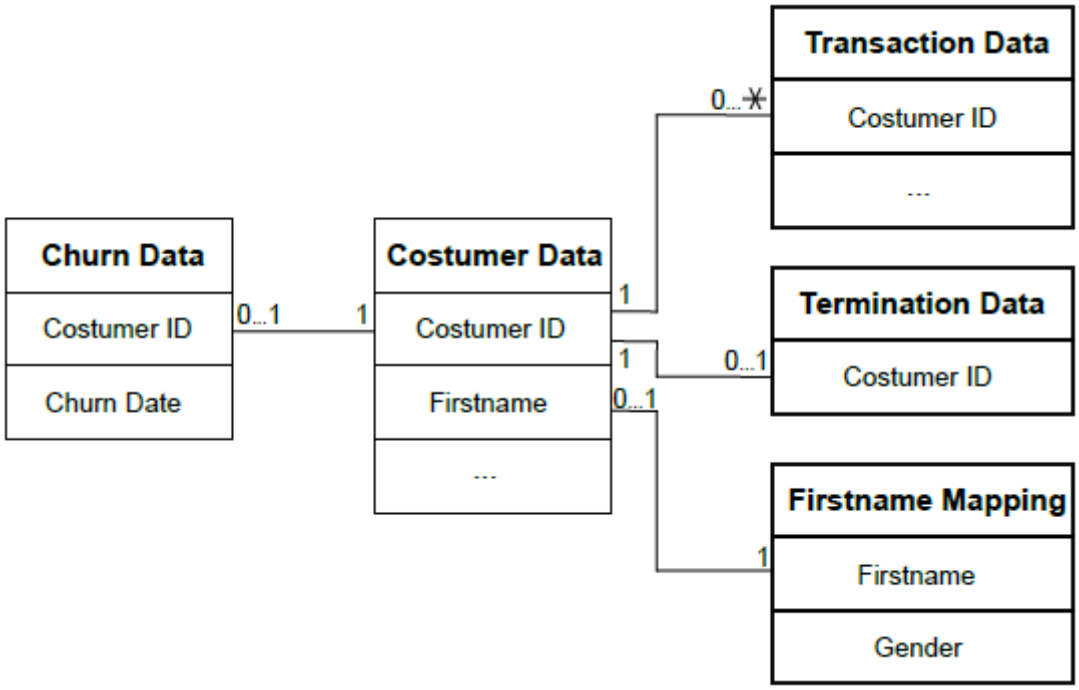


Figure 4.12 The Entity Relationship Model of our Data

Next to each table there is a symbol representing the correspondence between the two tables. This is their meaning:

- 1 Each element of this table appears exactly one time on the other table.
- 0...1 Each element of this table appears maximum one time on the other table, or it may not appear at all.
- 0...* Each element of this table appears any number of times on the other table, or it may not appear at all.

We will clarify the situation a bit with two examples:

1. Every customer in the Churn Data table has an ID that appears exactly once on the Customer Data table, thus we put the 1 next to Customer Data. On the other hand, every single customer in the Customer Data table may have churned or not, but they can only churn once! Thus we put 0...1 next to the Churn Data.

2. Similar to the churn data, every transaction has a CustomerId associated with it, and each ID appears exactly once in the Customer Data table. However, from the Customer Data table view, every customer can have any number of transactions (including zero). Consequently we put the 0...* symbol next to Transaction Data.

Exercise 11. Data Understanding

Open the data sets in your RapidMiner session. Feel free to explore the data in more depth by looking at the statistics and the charts in addition to the raw data.

Cleaning up the Results view.

After playing around with the data, your Results view may look a bit cluttered with several different open data sets. Instead of closing all the tabs one by one, you can close them in one go by right-clicking one of the tabs. In the resulting context menu you'll see an entry Close all results, which makes all tabs disappear.

4.4 Data Pre-processing – The Master Plan

Now that we understand the new data, it's time to advance one step in the CRISP-DM cycle. It's time for the **data preparation** phase!

We'll follow this plan:

1. Join the Churn Data to the Customer Data and calculate the label.
2. Join the Firstname Mapping to the Customer Data to get gender information for each customer.
3. Add the Transaction Data to the Customer Data.
4. Calculate the customers age from the Birthday column.
5. Clean the resulting data set by performing all best-practice steps learned:
    - Handle missing values.
    - Filter obvious outliers.
    - Define roles.
    - Remove unnecessary columns.
6. Remove customers that are in the Termination Data.

All steps introduce new concepts and help you to discover common pitfalls and implement best-practice checks in your future data analysis processes.

When all the steps are complete, you are ready to generate a new prediction model (you'll jump at the chance to play with new learning algorithms!). But first things first, let's start with the data preparation.

4.5 Getting Things Together – Joins

We briefly introduced the concept of joins in previous section as matching columns in one table (churn data) to another table (customer data) based on a common attribute (CustomerId).

4.5.1 Understanding Joins and Join Types

For a deeper understanding, let's look at a minimalistic example with the data sets in Figure 4.13, whose original excel files are provided in the "employee data - joins" folder. The first table lists employees of RapidMiner and a city ID. The other table contains a list of cities with a corresponding ID. In the following paragraphs we'll refer to the employees table also as the left table, and the cities table as the right table.

name	cityId
Fred	1
Ralf	2
Martin	2
Tom	3
Ingo	3
DataScientist #7	42

(a) Employees

id	city
1	London
2	Dortmund
3	Boston
4	Budapest

(b) Cities

Figure 4.13 Employees and Cities Data Sets

Using the ID, you can determine which employee works in which city. Just replace cityId in the employees table with the respective city from the cities tables. For Fred, Ralf, Martin, Tom and Ingo, that's pretty straight forward. Data Scientist #7, however, is everywhere and nowhere. He has a city ID that does not appear in the cities table, so we can't tell where he works.

What you do next depends on the requirements of the final data.

name	cityId	city
Fred	1	London
Ralf	2	Dortmund
Martin	2	Dortmund
Tom	3	Boston
Ingo	3	Boston

(a) Inner join

name	cityId	city
Fred	1	London
Ralf	2	Dortmund
Martin	2	Dortmund
Tom	3	Boston
Ingo	3	Boston
DataScientis...	42	?

(b) Left join

name	cityId	city
Fred	1	London
Ralf	2	Dortmund
Martin	2	Dortmund
Tom	3	Boston
Ingo	3	Boston
?	4	Budapest

(c) Right join

name	cityId	city
Fred	1	London
Ralf	2	Dortmund
Martin	2	Dortmund
Tom	3	Boston
Ingo	3	Boston
DataScientis...	42	?
?	4	Budapest

(d) Outer join

Figure 4.14 Different joins on the employee and city data

If you want to list only employees whose workplace is known, you'd skip Data Scientist #7 because he has no match in the cities table. This is called an inner join. The result of an inner join on the employee data is shown in Figure 4.14(a).

You could also decide to list all employees, regardless of whether they have a match in the cities table. If there is no match, then the city attribute is missing for the respective employee. The result of this left join is shown in 3.14(b).

So far the question is employee-centric: Give a list of all employees and tell me where they live. You can also turn the question to List all cities and tell me who works there.

If you perform an inner join to answer this question, you wouldn't list Budapest. As with Data Scientist #7, it's a matter of the requirements. If you want to see all cities including those don't appear in the employee list, you can perform a right join – Figure 4.14(c).

Finally, if you want to list all employees (even with unknown work place) and all cities in the result, you can perform an outer join, the combination of a left and a right join. Look at Figure 4.14(d) for the result of an outer join on the employee data.

4.5.2 Joins in RapidMiner

In RapidMiner, joins are performed by the Join operator (in the Data Transformation → Set Operations group). By default, the operator expects an attribute with role id in both data sets, which it then uses as the join attribute. You didn't define any roles yet, but Join allows you to define the join key manually by disabling the use id attribute as key parameter. You then select which attributes must match between the two input data sets in the key attributes parameter.

Summary: Join Types

- The **inner join** only adds rows that have a match in both tables. If a row does not have a match in both, it does not appear in the result.
- The **left join** keeps all rows from the left table, even if they don't have a match in the right table. If they don't have a match, they still appear, but the columns from the other table are filled with missing values.
- The **right join** behaves similarly to the left join; it guarantees that all rows from the right table appear in the result.
- The **outer join** guarantees that all rows from both tables appear in the result. Rows that don't have a match in both tables are filled with missing values. You can imagine an outer join as the combined result of a left and a right join.

Joins and Example Counts

The join operation can increase or decrease the number of data rows in the data set (compared to the input data).

- *More rows than before*. If a row in one of the tables has more than one match in the other, joined data table, it is duplicated. That is, it is added to each match in the other data set. For example, when we perform a right join in the employee and city data, the city of Dortmund occurs twice in the results because its ID is matched with both Marius's and Sebastian's cityId, and therefore we obtain one more row than the original cities' table.

- *Fewer rows than before*. If a row in one of the joined tables has no match in the other data set, it is omitted (unless it's in the left table of a left join or the right table of a right join). As a result, the size of your data set may decrease compared to the input data. Examples in the employee data are Data Scientist #7, who vanishes with the inner and right joins, and Paris, which disappears with the inner and left joins.

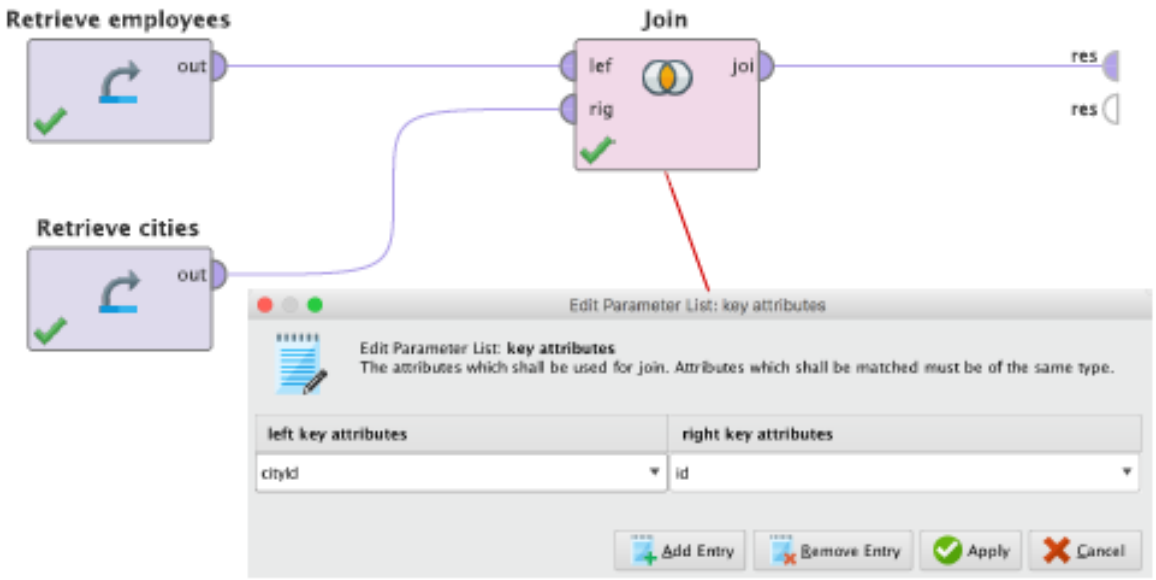
Exercise 12. Feel the Join

In the data supplied with this book there is an employees and a cities data set.

Create a new process, save it under a reasonable name (e.g., test joins), and join the two tables. Play around with the parameters of the Join operator to get a feel for it.

Answers to Exercise 12

The general process setup as shown:



4.5.3 Joining the Churn Data

Enough with RapidMiner's employees, and on to the customers! Now that you know about the different join types and have gathered some joining experience, it's time to perform a join on real data.

Remember that you want to add the ChurnDate to the Customer Data so that you can derive the label in the same way as in previous section. That is, via the rule:

if the churn date is missing then loyal otherwise churn

Exercise 13. Churn Data and Label Generation

Part I — Join the Churn Data

Start a new process and save it as prepare data. Next join the Churn Data to the Customer Data. Use the Customer Data as the left data set.

1. Which join type do you need?
2. Which attribute is the join key?

Part II — Generate the Label

When the join is working, you have the ChurnDate attribute in the same format as in RapidMiner & DataScience: Foundations. As you did there, generate the label from that column as follows:

Generate a new attribute called Churn, and set its values to either loyal or churn, following the rule described above. Do not set the role yet, we'll do that later when we clean up the complete data set.

Hints:

- *Work in small steps. Do not try to create the complete process in one go. Instead, test each significant change that you make. In this way you detect errors as soon as they occur, making error handling much easier.*
- *It is good practice to place breakpoints before and after a new Join operator to check whether the result makes sense with the given input.*
- *Pay special attention to the example counts before and after the join.*
- *You want the same number of customers in your data before and after the join, right? So choose your join type wisely.*
- *To generate the label column, you can use the Generate Attributes operator.*

Answers to Exercise 13

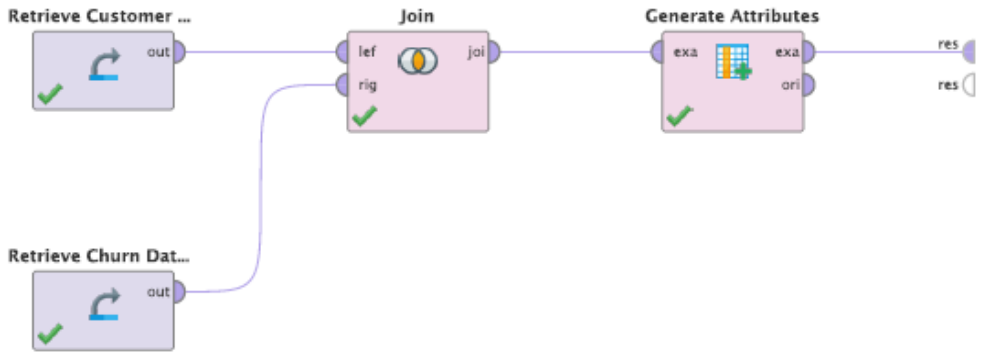
Since you need an entry for each customer in the result, you need to perform a left join (where the Customer Data is the left table). With an inner join you'd get only those customers with an entry in the Churn Data, meaning that you lose all information on loyal customers (and, as a result, wouldn't be able create a classification model later).

The join key is the CustomerId, as that's the attribute that identifies customers uniquely.

In Generate Attributes, you can use the following expression:

```
if(missing(ChurnDate), "loyal", "churn")
```

The complete process setup is shown below. Be sure to use the data from the repository; do not use Read Excel.



The resulting data after joining the churn data and creating the Churn column is shown:

Row No.	PostalCode	HashCode	CustomerId	Firstname	rowNumber	Birthday	ChurnDate	Churn
1	87213	tKlbadnh	3	Eva	1	Apr 13, 19...	?	loyal
2	38548	RcW2Pb3w	4	Kunigunde	2	Jun 21, 197...	?	loyal
3	44573	akWNQl4e	6	Notburga	3	Jun 5, 1985...	?	loyal
4	70936	glrPDLzY	7	Maximiliane	4	Jul 26, 199...	?	loyal
5	49705	3rGPBX98	9	Dorothea	5	Aug 4, 194...	?	loyal
6	42376	XlxhfOlo	12	Maria	6	Jul 29, 196...	?	loyal
7	52245	3ANQ9shn	16	Rosina	7	Jun 28, 196...	Apr 3, 2012 6:54:0...	churn
8	56625	BDEPLKmG	17	Susanne	8	Apr 14, 19...	?	loyal
9	66713	gQBDZ8Lw	20	Genoveva	9	Mar 31, 19...	Apr 19, 2013 5:44:...	churn
10	70052	G1c0kZqQ	21	Katharina	10	Nov 16, 19...	?	loyal

4.5.4 Joining the Gender Information

According to our master plan, after adding the churn information and generating the label column, we want to impute the gender from the first names of our customers by joining the Firstname Mapping table.

Generally, this is pretty straight forward, but to add some more realism to the example we have added two stumbling blocks. Can you find them?

Exercise 14. Imputing Gender Information

This is a long exercise. It requires everything you learned on joins earlier and careful analysis of your results. You will (hopefully) find two problems in the data . You'll even be guided to find new operators in the Operator tree that will solve your issues. To help with all this, there's a complete page of hints.

Extend the process prepare data from the previous exercise.

Join the Firstname Mapping to its result. As always, carefully check the results. Something is strange. Can you find it? Even fix it?

Try to solve the exercise on your own. In case you get stuck, feel free to use the hints. Each hint adds more information to make the exercise easier. Try to use as few as possible!

Remember that there are two challenges hidden in the data. Don't despair if the solution to the first seems to be "overdone" – it's just the second challenge!

General hint: This time you won't use the customer ID as the join key.

Hints for identifying the first challenge:

1. *The "strangeness" of the result is in the example count. Refer to the previous infobox for causes of increased or decreased example counts. With that information, can you identify the cause?*
2. *If you did not identify the cause for the increased row count, sort the Firstname Mapping by the Firstname and look again.*
3. *Some names appear twice or more in the mapping table. As you saw with the employee-city example, that causes the respective customers to be included twice.*

Hints for solving the first challenge:

1. *In the Cleansing → Duplicates group there's an operator to solve the problem. Remember that the Help panel provides documentation on each operator you select.*
2. *Remove Duplicates is your friend. Just add it to the correct branch in front of Join.*

As promised, solving the first challenge reveals the second one. Here are some

hints for round two:

1. *Again, the row count is wrong – too low, this time. Or, if you went for a left join, you have missings in the Gender column.*
2. *As the infobox tells you, both variants indicate that you don't have matches for all names. Look closely at the Firstname column, your join key. All names seem to have a match, so what's really happening?*
3. *In the Blending → Values group, there's an operator that will help you.*
4. *Trim is your friend in this challenge.*

Answers to Exercise 14

Such a long exercise deserves a detailed answer. Here it goes...

Part I — The Actual Join

To perform the actual join, add a Join operator and use the results of the join from the previous exercise as the left input. For the right input, drag the Firstname Mapping from the data folder of the repository into the process and connect it to the lower port of the new Join.

This time, the join attribute is the Firstname attribute, as that's the connecting information between the two tables. You should select either an inner join or a left join. If you do an inner join, customers without a match in the firstname mapping are removed from the result. With the left join they are included, but with missing gender information (and hence can be removed later during data cleaning).

Part II — The First Challenge: Too Many Rows

After performing the join, there are more rows in the result than original customers. You started with 1,000 customers, but after the (inner) join you have 1,150 rows. While it may at first seem awesome that you can increase your customer count with a simple join, performing a join on the data will not result in new, real customers registered with your app. So there must be a problem with the data.

As discussed in the employees example, data rows with more than one match in the partner table get added twice. So obviously some of the customers first names appear twice in the mapping. If you sort the Firstname Mapping in the results view, or look at its statistics, that suspicion is confirmed.

You need another step of cleaning. To remove duplicate first names, you can use the Remove Duplicates operator. The operator searches for identical rows in the data and removes all but the first instance.

By default it compares all attributes, and only if all attributes of two rows are equal, are the rows considered equal. In this case, though, you only want to compare the Firstname, so you should select exactly that in the Remove Duplicates parameters. After applying the operator on the Firstname Mapping data before the join, you don't have too many rows anymore, but…

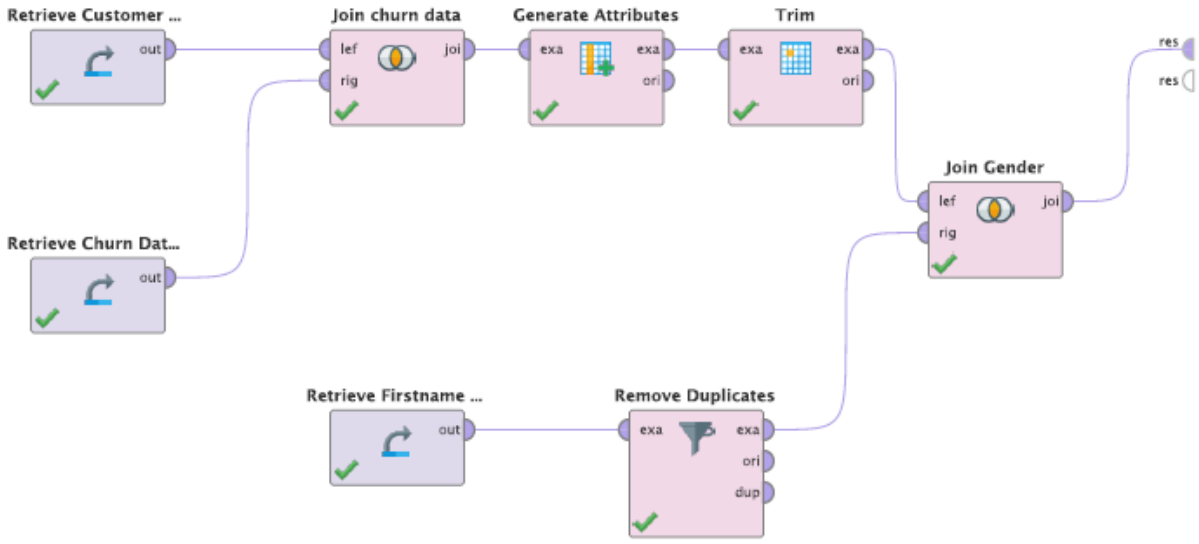
Part III — The Second Challenge: Too Few Rows

Now we have too few rows! (Or, if you went for a left join, you have missings in the Gender column). As you learned from the employees example, this means that some customers don't have a match in the gender table.

The reason is a tiny detail: some of the customers' names have a leading whitespace. Whether that comes from sloppy user input, wrong database setup, or just a course designer's mood is irrelevant, you need to remove it.

Here the Trim operator comes into the game, which removes all leading and trailing whitespace from an attribute.

Here the complete process setup:



The data after joining the gender information:

Row No.	HashCode	Firstname	Churn	PostalCode	CustomerId	rowNumber	Birthday	ChurnDate	Gender
1	tKlbadnh	Eva	loyal	87213	3	1	Apr 13, 19...	?	w
2	RcW2Pb3w	Kunigunde	loyal	38548	4	2	Jun 21, 197...	?	w
3	akWNQl4e	Notburga	loyal	44573	6	3	Jun 5, 1985...	?	w
4	glrPDLzY	Maximiliane	loyal	70936	7	4	Jul 26, 199...	?	w
5	3rGPBX98	Dorothea	loyal	49705	9	5	Aug 4, 194...	?	w
6	XlxhfOlo	Maria	loyal	42376	12	6	Jul 29, 196...	?	w
7	3ANQ9shn	Rosina	churn	52245	16	7	Jun 28, 196...	Apr 3, 201...	w
8	BDEPLKmG	Susanne	loyal	56625	17	8	Apr 14, 19...	?	w
9	gQBDZ8Lw	Genoveva	churn	66713	20	9	Mar 31, 19...	Apr 19, 20...	w
10	G1c0kZqQ	Katharina	loyal	70052	21	10	Nov 16, 19...	?	w

In this chapter, and this last exercise in particular, you've learned a lot. Not only can you perform joins, but you have learned how to identify common problems with joins. The most common problems with joins in real data result from issues with the join key, duplicate data rows, etc., In many cases, as you saw, you can detect problems by looking at the row counts before and after the join operation.

You have also learned about two new pre-processing operators, Remove Duplicates and Trim. Now it's time for the next step of the master plan – leveraging the transaction data.

4.6 Condensing the Data – Aggregations

As you can guess from the title of this section, adding the transaction data to the data set is about more than just joins.

Let's look again at the transaction data. Figure 4.15 shows the first few rows of the data set, sorted by the CustomerId. In the header you can see that there are far more transactions than customers (4,334 to be precise). From the sorted rows below, it's obvious that there are potentially several transactions per customer.

ExampleSet (4334 examples, 0 special attributes, 4 regular attributes)

Row No.	Custom...	Transactio...	PaymentMe...	Date
1	1	11.508	credit card	Apr 17, 2012 2:05:39 AM CEST
1001	1	374.218	cheque	Aug 7, 2010 2:05:39 AM CEST
1002	1	11.098	cash	May 28, 2011 2:05:39 AM CEST
1003	1	0.249	credit card	Feb 27, 2011 2:05:39 AM CET
1004	1	0.027	cash	Oct 8, 2010 2:05:39 AM CEST
1005	1	1.125	cash	Nov 25, 2010 2:05:39 AM CET
2	2	330.070	cheque	Nov 25, 2011 6:58:02 AM CET
1006	2	0.230	cheque	Sep 15, 2011 6:58:02 AM CEST
1007	2	23.732	cheque	Feb 9, 2010 6:58:02 AM CET
852	3	24.747	credit card	Feb 15, 2012 5:29:26 PM CET
3815	3	35.088	credit card	Jun 13, 2010 5:29:26 PM CEST
3816	3	26.119	credit card	Dec 12, 2011 5:29:26 PM CET
3817	3	2.819	cheque	Dec 4, 2010 5:29:26 PM CET
3818	3	30.298	credit card	Jun 12, 2011 5:29:26 PM CEST

Figure 4.15 The first rows of the Transaction Data, sorted by CustomerId

What happens if you simply join the data to the current data set by the customer ID? Since each customer in the current data has several matching transactions, his data will be duplicated (try it if you'd like). That's certainly not what you want. Remember that the goal is to train a model that predicts whether a single customer will churn or remain loyal – your model will work on a customer level. If you duplicate the customer data, keeping one row for each transaction, your model will be on a transaction level.

So, to generate a model on the customer level, you need exactly one row per customer. That means you must condense the transaction to one row per customer. Enter the **aggregation** operation.

Aggregation performs two simple steps:

1. Groups the data by the values of one column, e. g., the CustomerId
2. In each group, calculates one or more things, such as the sum of all transaction values or the most frequent payment method.

4.6.1 Aggregation in RapidMiner

In RapidMiner, the Aggregate operator performs these aggregations. Usually, the only parameters that you need to configure are group by attributes and aggregation attributes:

- *group by attributes*: This parameter specifies how to define a group. If all examples of the selected attribute(s) match, they are considered one group. So, to calculate information on the customer level, select the CustomerId.

  You can also specify more than one group attribute, e. g., the customer ID and the payment method. In that case, the group contains all transactions by a customer in which they used a specific payment method. This would answer questions like “how often did each customer pay with each payment method?” If you do not select anything, the complete dataset is treated as one big group. That means that the result will contain only a single row. Keep this in mind – it’s a common error to forget to define the group attributes.
- *aggregation attributes*: This parameter specifies which function to calculate on which attribute. Specify an attribute (on the left) and the function to calculate (on the right). See Figure 4.16 for an example. Have a look at the infoboxes below for a summary of the most common functions.

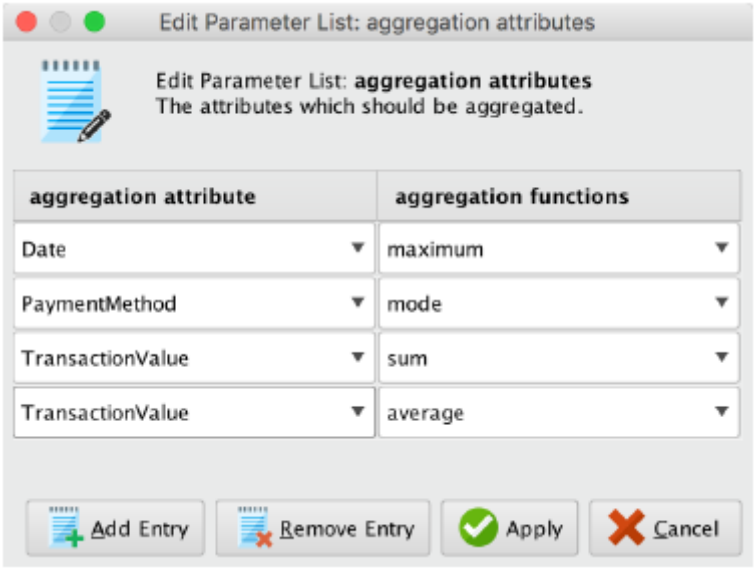


Figure 4.16 A sample of Aggregate's aggregation attribute parameter

The output of the operator contains only the group attributes and an attribute for each aggregation attribute, in the form <aggregation_function>(<source_attribute>), e. g., sum(TransactionValue).

Here's the most commonly used **nominal aggregation functions**:

- *concatenation*: the concatenation of nominal values in the group, with a vertical bar as separator. The concatenation of Hello and RapidMiner would result in Hello|RapidMiner.
- *least*: the least frequently appearing nominal value in a group. On the PaymentMethod column of the transaction data the least function returns cheque for CustomerId 1 and cheque for CustomerId 2 (since there are no other values).
- *mode:* the most frequently appearing nominal value in a group.

Here's the most commonly used **numerical aggregation functions**:

- *count*: the number of rows in a group. For CustomerId 1 in Figure 4.15 the count of any attribute would be 6. Note that the count function counts the number of rows, not the unique values, in the group. This means that it does not matter which attribute you select for the count method (as long as no missing values are involved).
- *average*: the average of the values in a group. For example, the average of 2 and 6 is 4.
- *maximum*: the maximum value of a numerical (or date) attribute. On the Date column, maximum returns Tue Apr 17 02:05:39 CEST 2012.
- *minimum*: the minimum value of a numerical (or date) attribute. On the Date column, minimum returns Sat Aug 07 02:05:39 CEST 2010.
- *median*: median sorts the values of a column and returns the one in the middle. For customer 2, median returns 0.230 on the TransactionValue.
- *standard_deviation*: the standard deviation of a numerical attribute in a group. The standard deviation is a measure for the steadiness of the values: a low standard deviation indicates that the values are very similar.
- *variance*: closely related to the standard deviation, the variance is the square of the standard deviation.
- *sum*: the sum of all values in a group. The sum of the transaction value for customer 2 is 354.032.

Exercise 15. Adding Transaction Information

Apply what you have learned. Continue the previous process prepare data and add the transaction data to its result.

Part I – Action

1. Aggregate the transaction data to a customer level. Calculate the following values:

- the total value of all transactions the customer has made
- the average value of each customer's transactions
- the customer's preferred payment method
- the date of the last transaction made by each customer

2. Join the aggregated transaction data to the customer data.

Hints:

1. *As always, do one thing at a time and test each step.*
2. *If your result after the aggregation contains only a single row, you probably*
3. *forgot to specify the group attribute.*

Part II – Questions

1. What happens to customers that have never performed a transaction?
2. Are there any such customers?

Answers to Exercise 15

Part I – Action

To aggregate the transactions you need to first load them by dragging the Transaction Data entry from the repository into the process. Then you can add the Aggregate operator and connect it to the transaction data.

In Aggregate, select CustomerId in group by attributes, and configure the aggregation attributes as shown in Figure 4.16.

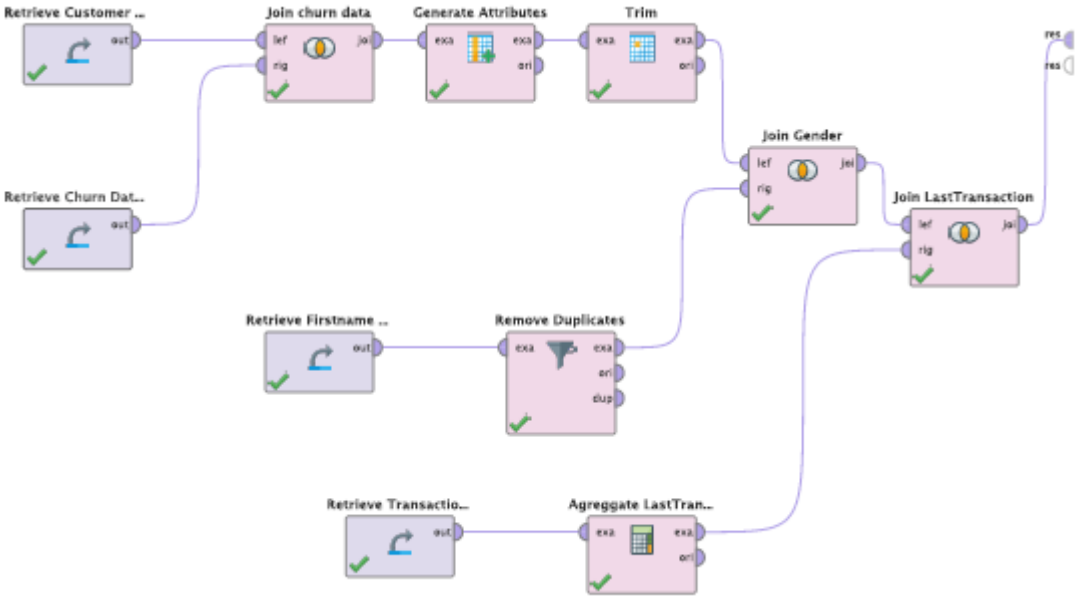
Let's talk about the semantics:

- sum(TransactionValue): the sum of all purchases the customer made.
- average(TransactionValue): the average of all purchases the customer made.
- mode(PaymentMethod): the payment method most frequently used by the customer (so probably his preferred one).
- maximum(Date): the latest date, or the customer's last interaction with our system.

To join the aggregated transaction data you have, add another Join operator. Connect the output of Aggregate to the Join's lower port, and the result of the previous process to its upper port. Don't forget to select the CustomerId in key attributes.

Use either a left or an inner join – for a recap on the effects of the choice, look at the answers to the questions below.

The final process is shown below:



The data after joining transaction information (some old columns on the left are not shown):

CustomerId ↑	maximum(Date)	mode(PaymentMethod)	sum(TransactionValue)	average(TransactionV...
1	Apr 17, 2012 2:05:...	cash	398.225	66.371
2	Nov 25, 2011 6:58...	cheque	354.032	118.011
3	Feb 15, 2012 5:29:...	credit card	119.070	23.814
4	Oct 9, 2010 11:22:...	credit card	398.223	132.741
5	Jun 13, 2012 10:1...	credit card	432.282	432.282
6	Jul 16, 2010 9:39:...	cheque	459.644	51.072
7	Mar 15, 2012 10:1...	credit card	111.438	111.438
8	Jun 16, 2011 9:46:...	credit card	512.803	512.803
9	Mar 30, 2011 2:17...	credit card	319.274	319.274
10	Apr 17, 2013 6:06:...	credit card	112.451	14.056

Part II – Questions

1. What happens to customers that have no transactions depends on the join type you chose. For the inner join, customers without transactions don't have a match in the transactions table and are not part of the result.

   For a left join, customers without transactions are part of the result, but the transaction information is missing.

2. If you look at the result, you'll see exactly 1,000 records, same as before. No example contains missing values in the newly joined transaction columns.

   Ta-da! Our data only contains customers who have had at least one interaction.

4.6.2 Housekeeping – Process Clean-up

You've achieved a lot in the previous exercises and your process has grown and grown. To keep control and not be overwhelmed by the sheer number of operators, let's apply some of your knowledge on process cleaning and add one or two Subprocesses.

Exercise 16. Housekeeping

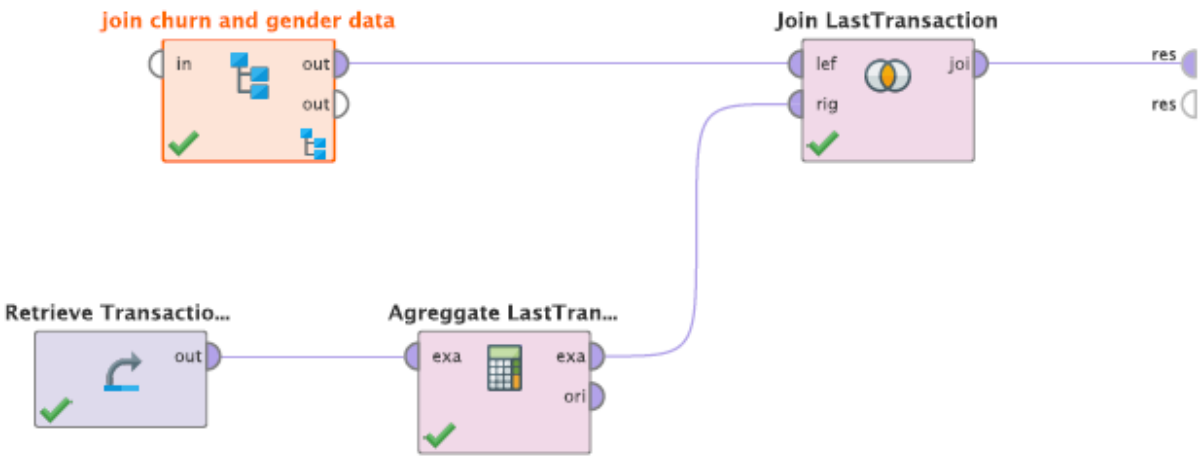
Put the complete upper branch into a Subprocess operator. Try to find a reasonable name for the operator, e. g., Join Churn and Gender. Feel free to further structure the subprocess, but for now, one Subprocess operator is probably sufficient.

Also, make sure that your operators are nicely arranged, without any crossing connections, for example. It's good feng shui and it greatly increases process readability.

Hint: The easiest way to move the operators into the subprocess is to select them all, right-click on one of them, and select Move into new subprocess.

Answers to Exercise 16

The resulting process is shown:



After this cleanup you're ready for the next challenge – leverage the transaction data to the max!

4.7 Rotating the Data – Pivoting

The aggregated data now contains each customer's preferred payment method in the attribute mode(PaymentMethod). But there's way more information you can leverage from the Transaction Data. For example, instead of just learning the preferred payment method, you could count how often each payment method was used. Knowing that may improve your predictions in the following chapters, so let's extract it!

This task is two-staged. First, you need to use a multi-key aggregation to count the payments per customer and payment method. That will produce up to three rows per customer, so in a second step you need to find a method to place all that information into a single row. (Remember, you are working on the customer level, and hence need all a customer's data in one row).

Every time you hear something like "calculate something per X and Y," it should trigger the thought "Aggregation." Set X and Y as group attributes, and the something you want to calculate as the aggregation attribute. That's just what you'll do in the exercise below, but as a spoiler, we printed the result in Figure 4.17.

Row No.	CustomerId	PaymentMethod	count(PaymentMethod)
1	1	cash	3
2	1	cheque	1
3	1	credit card	2
4	2	cheque	3
5	3	cheque	1
6	3	credit card	4
7	4	cheque	1
8	4	credit card	2
9	5	credit card	1
10	6	cash	1

Figure 4.17 Payments per customer and payment method

As you can see, for each group, i. e., for each combination of CustomerId and PaymentMethod, the result contains one row. You can also see that Aggregate omits combinations that do not appear in the input data: customer 2 has never paid by credit card or by cheque, so those entries are not listed in the result.

Anyway, you now have up to three rows per customer. You need to reduce it to the entity level, i. e., to one line per customer. This time, though, you can't simply aggregate again – you can't combine the information further (which is what aggregation does). What else can you do?

One common approach is to "rotate" the data such that you again have one row per customer and one column per payment method. This approach is called **pivoting** and its output is shown in Figure 4.18. Compare it to Figure 4.17 – it contains the same information, just in a different format – the desired format!

CustomerId	count(PaymentMethod)_cash	count(PaymentMethod)_cheque	count(PaymentMethod)_credit card
1	3	1	2
2	?	3	?
3	?	1	4
4	?	1	2
5	?	?	1
6	1	7	1
7	?	?	1
8	?	?	1
9	?	?	1
10	?	?	8

Figure 4.18 Pivoted transaction data

4.7.1 Pivot in RapidMiner

For the pivot operation you need to specify three things:

- *Group by attributes*: pivot creates exactly one row for each value in this attribute. In our case it's the CustomerId.
- *A column grouping attribute*: pivot creates one column for each value in this attribute. Here, that's the PaymentMethod.
- *Aggregation attributes*: This parameter specifies which function to calculate on which attribute, similar to the parameter in Aggregate operator. Here, we put count of PaymentMethod.

In the dataset from Figure 4.18, after the pivoting, there are columns' name started with "count(PaymentMethod)_xx". These attributes deliver the values of the cells in the result. If there is no information for a certain cell, e. g., cash for customer 2, then a missing value is inserted, as can be seen in the figure.

You'll take care of those in the exercise below.

In RapidMiner, the pivot operation is implemented via the Pivot operator in the Data Transformation → Rotation group.

Because you want to combine the payment method counts with the data you already have, you need another join. In the exercise below, test your knowledge on pivot by reproducing the results from the figures discussed here.

Exercise 17. Pivoting

Continue working on the previous process. Count how often each customer paid with which payment method and add the information to the dataset you already generated.

Follow this road map:

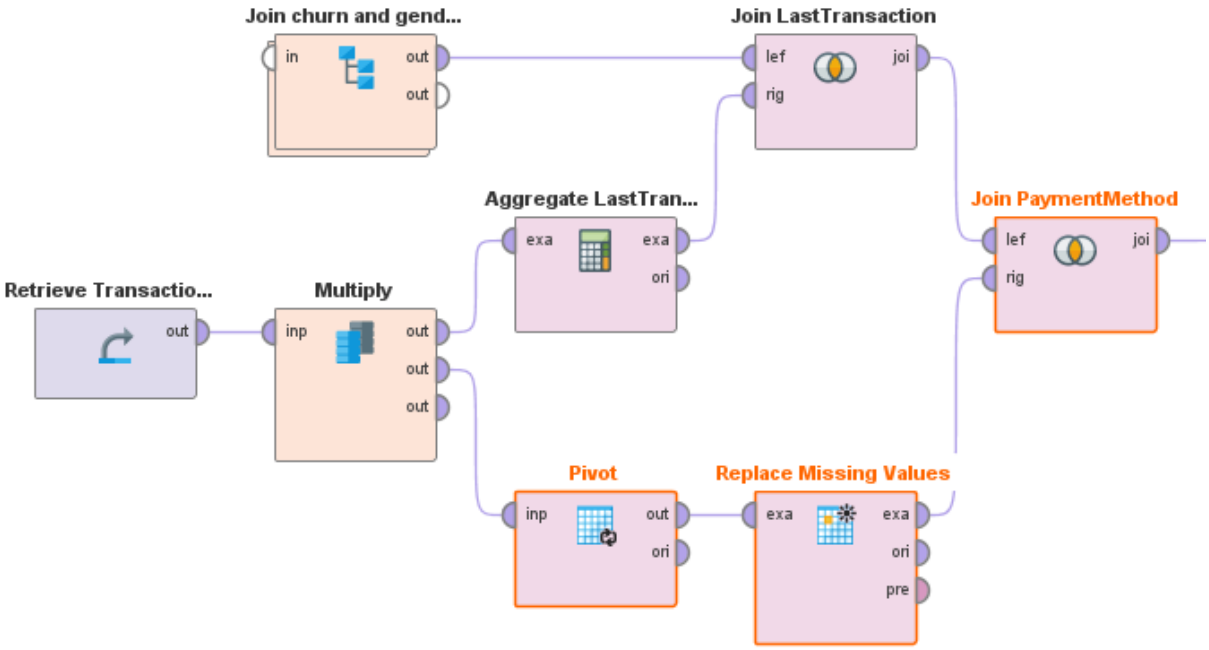
1. Pivot the data.
2. Take care of the missing values.
3. Join the data.

Hints:

- *The tricky thing, apart from implementing the pivot, is the placement of the operators. You actually need to multiply the transaction data a second time. You could simply load the data again and pass it to the Pivot operator, but a more elegant (and faster) method is to use the Multiply operator. Have a look at the infobox below for more information.*
- *Remember that you can treat missing values with Replace Missing Values. Think about the semantics, i. e., the meaning, of missing values in this dataset. Look at the documentation of Replace Missing Values to find a way to implement your ideas*

Answers to Exercise 17

Arrange the operators as shown below. The red frame highlights the newest members of the party.



1. In Pivot operator, you have to select the CustomerId as group by attributes. PaymentMethod is the column grouping attribute. In the aggregation attributes select the count function for PaymentMethod.

   The rule is one row per group, one column per column grouping, so this creates the desired output of one row per customer and one column per payment method.

2. In this dataset, missing values indicate that in the input data of Pivot the combination of CustomerId and PaymentMethod does not exist. That means that the respective customer never used that “missing” method so you can replace the missing values with 0.

   Using the Replace Missing Values operator, if you set the default parameter to value you can insert a 0 as the replenishment value. That is equivalent to setting the default parameter to zero.

3. This is the fourth Join operator you have configured, so you probably know the routine: untick use id attribute as key and set the CustomerId as key attribute on both sides.

The data after joining the pivoted payment information (only the right-most columns):

CustomerId	count(PaymentMethod)_cash	count(PaymentMethod)_cheque	count(PaymentMethod)_credit card
3	0	1	4
4	0	1	2
6	1	7	1
7	0	0	1
9	0	0	1
12	1	0	0
16	0	0	5
17	5	0	0
20	2	0	0
21	3	0	0
22	0	0	6

The **Multiply** operator can be used if you want to connect one output port to two or more input ports – it simply copies, without changes, its input to all its output ports. Since it magically produces a new output port each time you connect the previous one, you can connect it to as many subsequent operators as you wish.

While you could drag it manually into the process from the Process Control group, there's an easier way to get it.

Just draw a new connection to the second operator, an "add multiply"  button will appear on the operator. Click at the add multiply button and get a Multiply operator for free!

4.7.2 Cleaning the Attributes Names

The attribute names that have been created by the Pivot operator look a bit messy and are not very intuitive. Instead of count(PaymentMethod)_cash, wouldn't be a string like Payments with cash look much nicer?

In this section you will learn how to efficiently rename many attributes at the same time to apply that pattern to the new attributes. In the first part of this document you have already learned how to rename one attribute at a time with the Rename operator. You could use the same operator here, too, but since you want to rename three attributes that have the same pattern, that approach feels a bit tedious. Instead, follow this this rule of thumb:

To rename one attribute, use the Rename operator. If you want to rename a whole bunch of attributes following the same rule (like exchanging parts of the attribute name with something else), use Rename by Replacing. The latter takes a regular expression, and whenever a part of an attribute name matches the expression, it is replaced by a predefined string.

Look at Figure 4.19. This is the default configuration of Rename by Replacing – it replaces the magical regular expression \W with an empty string, i. e., it simply removes anything that is matched by \W.

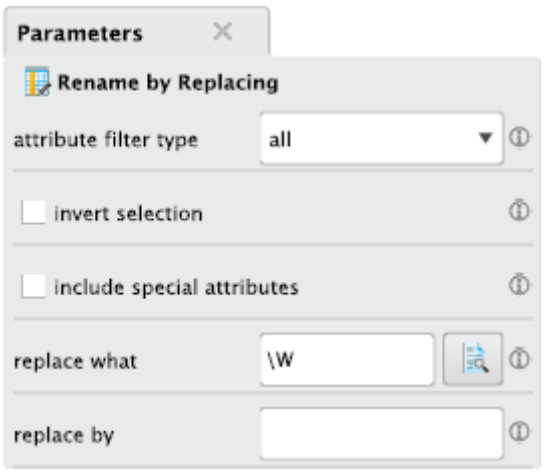


Figure 4.19 Default configuration of Rename by Replacing

\W is one of many predefined so-called character classes; it matches anything that is not a letter, a number, or an underscore. Let's look at the three attribute names:

- count(PaymentMethod)_cash
- count(PaymentMethod)_credit card
- count(PaymentMethod)_cheque

They all start with count(PaymentMethod)_, followed by the actual payment method. To change the names into

- Payments with cash
- Payments with credit card
- Payments with cheque

we simply need to replace count(PaymentMethod)_ by Payments with .

This can easily be done with the operator Rename by Replacing. Simply insert a regular expression that matches count(PaymentMethod)_ into the replace what parameter, and insert Payments with into replace by.

Exercise 18. Cleaning the Attributes Names

Rename all count(PaymentMethod)_XXX attributes to Payments with XXX by using a Rename by Replacing operator.

Answers to Exercise 18

Insert a regular expression that matches count(PaymentMethod)_ into the replace what parameter, and insert Payments with into replace by.

The regular expression in replace what could look like this:

```
count\(PaymentMethod\)_
```

Mind the backslashes in front of the brackets: since they have a special meaning in regular expressions, they need to be escaped.

Instead of typing the full text, you could also match count followed by something, up to an underscore:

```
count.*_
```

4.8 The Age

Why would you want to use age instead of birthday? Usually, age has a stronger and more stable influence than birthday. Often, people behave differently based on their age. For a single point in time, it doesn't matter whether you build your model based on age or birthday. But if you base it on birthday, you can't use it in the future. For example, someone born in 1985 is 30 years old. Today, the rules "customers aged 30 or older are likely to churn" and "customers born in 1985 or earlier are likely to churn" are equivalent. But in 5 years time, the latter rule would in fact mean "customers aged 35 or older..."

If we assume that actions are bound to age rather than birthday, we need to convert the birthday to an age.

You do this by calculating the difference between the current date and the birth date. In RapidMiner you can use the date_diff(start_date, end_date) function in Generate Attributes for this purpose.

The start date is, of course the birthday; the end date is the current date, which you can get with the date_now() function. With this information you can apply the date_diff() function to calculate the age. This function is capable of handling very short time frames and hence delivers the difference in milliseconds, but that doesn't suit our purposes. Therefore we must convert the result from milliseconds to years, which are a better representation for age:

- 1000 milliseconds equals 1 second
- 60 seconds equals one minute
- 60 minutes equals one hour
- 24 hours equals one day
- 365.25 days equals one year (on average, considering leap years)

To convert a duration from milliseconds to years, you have to divide it accordingly:

```
age_in_milliseconds / (1000 * 60 * 60 * 24 * 365.25)
```

As you will see during the exercise, the above calculates the age to several-digit precision. As you have learned in RapidMiner & DataScience: Foundations, you round values with the round() function. There are two more functions for removing digits – floor() to use the next smaller integer, and ceil() for the next higher integer:

```
round(31.5) → 32
```

```
round(31.4) → 31
```

```
floor(31.9) → 31
```

```
ceil(31.1) → 32
```

Exercise 19. Calculate the Age

With the help of the information above, create an Age attribute that contains the customers' age in years.

Hints:

As always, use small steps and test after each one.

1. *Use date_diff() with the correct parameters.*
2. *Add the conversion from milliseconds to years.*
3. *Round/cut superfluous digits.*

Answers to Exercise 19

Add a Generate Attributes operator at the end of the process.

Add a single line for the Age attribute with the formula:

```
floor(date_diff(Birthday, date_now()) / (1000 * 60 * 60 * 24 * 365.25))
```

4.9 The Final Touch

Still following the master plan, let's give the data a final polish. In this final section of the data preparation phase you'll perform all the steps that should be part of any data pre-processing:

- Handle missing values
- Filter extreme values/outliers
- Assign sensible roles to the attributes
- Remove unneeded attributes
- Perform some data set specific actions

While the first four points are pretty straight forward and similar (if not identical) to the actions taken in the Section 2 in this course, the last point requires some additional attention – remove the customers who are part of the Termination Data.

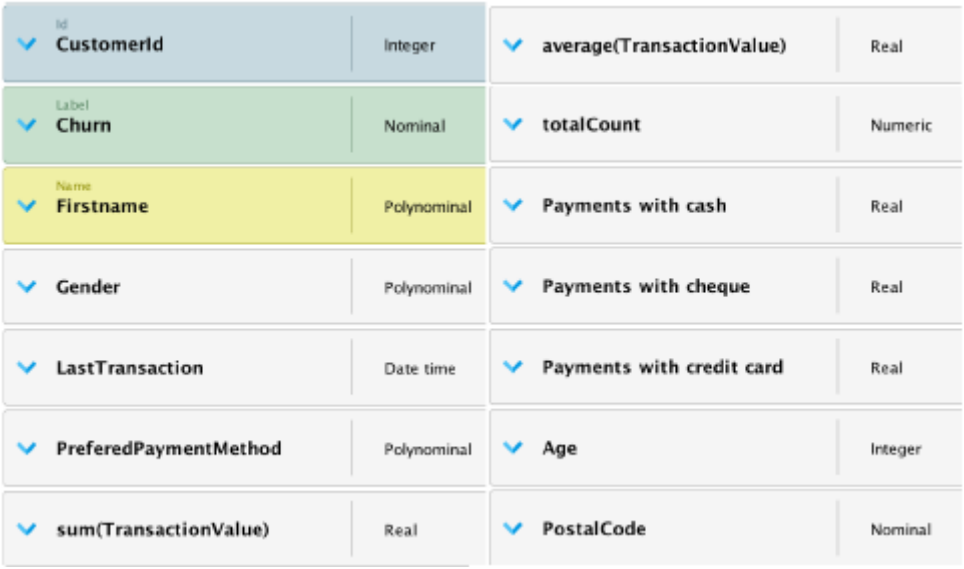
The entire next section is dedicated to this last point. You should be able to perform all other steps with the knowledge from Section 2, so we'll jump directly into an exercise.

Exercise 20. The Final Touch Up

Perform all necessary best-practice pre-processing steps:

1. Remove unnecessary attributes.
2. Set proper roles.
3. Filter examples that contain missing values.
4. Filter customers younger than 18 and older than 110.
5. Use only the first digit of the postal code.

Your attributes shall look similar to below:



Answers to Exercise 20

1. Use Select Attributes to remove all but the attributes listed in the question.

   While Firstname is technically not necessary for the analysis, it's nice to keep it in the data for easier reference to the data. It is more convenient to talk about Benno and Anna instead of customers 51 and 759. Use a special role to disable it for the analysis.

2. The roles can be set with a Set Roles operator. Assign them like this:

   - Churn → label
   - CustomerId → id
   - Firstname → firstname

   Hint: you can use set additional roles to assign more than one role with a single operator.

3. Use Filter Examples. You can either:

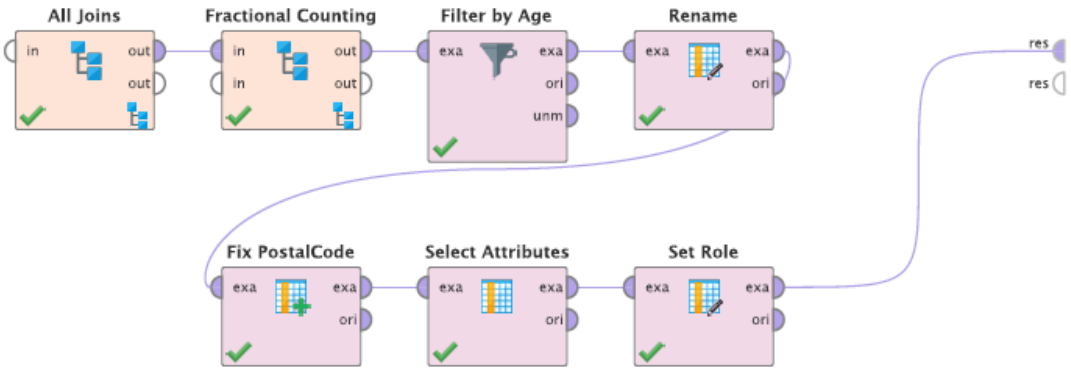
   - Use Add Filters with the custom_filters filter condition and specify manually which attributes to test for missing values.
   - Switch the condition class to no missing attributes, which will check all regular attributes (excluding the label and the id, since they are special) for missing values, and keep only those examples without any missing.

4. Use a second Filter Examples operator here.

5. You need a Generate Attributes operator. You should use the following formula:

   prefix(suffix(concat("0000", str(PostalCode)), 5), 1)

For further reference, look at the process below.



4.10 The Termination Data: Set Minus

In this last data preparation section you will remove the customers in the Termination Data from the data we have prepared so far. For reference, Figure 4.20 shows the Termination Data once again. It's just one column containing customer IDs – the IDs of those customers who did not actively churn by cancelling their subscription, but who simply forgot or chose not to renew their subscription term.

The next step is to remove all examples from the dataset where CustomerId is also part of the Termination Data. We call this operation the complement, realized by the Set Minus operator in RapidMiner.

Row No.	CustomerId
1	1
2	2
3	4
4	6
5	9
6	12
7	17
8	21
9	24
10	26

Figure 4.20 Termination Data

Set Minus compares the ID attributes, those with the special role id, in its two input datasets, and removes all examples from the first dataset whose ID also appears in the second dataset. Unlike other set operators, Set Minus is asymmetric: the declaration order of the parameters determines the result.

Exercise 21. Termination Data

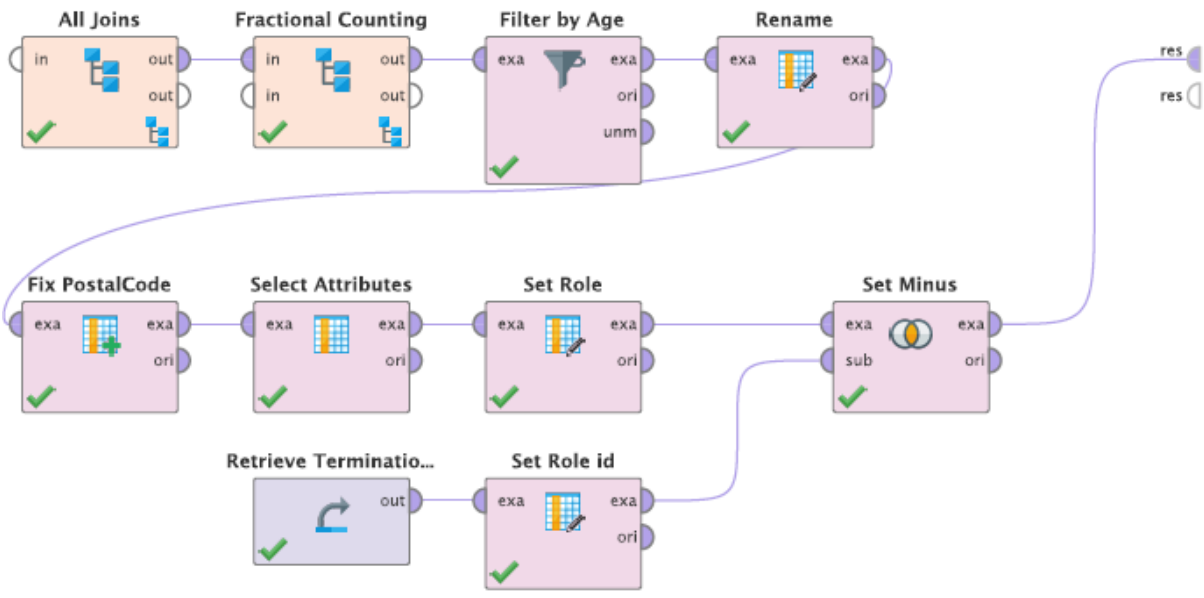
Extend the prepare data process. Remove all customers who are in the termination data from the current data set.

Answers to Exercise 21

You need to drag the Termination Data from the data folder in your repository into the process. Set the role of the CustomerId column to id.

Then, add the Set Minus operator; connect the current dataset to the exa port and the Termination Data to the sub port.

The complete process to remove the termination data from the customer data with the Set Minus operator is shown below:



Now finally we're ready to train our first model on the new data!

On the Importance of ETL

We spent lots and lots of time on data preparation. Remember, for real world projects it's not uncommon to spend 50% - 90% of your time on pre-processing. This is especially true when working with an advanced tool like RapidMiner, where the actual choice of an algorithm boils down to just dragging different learning operators into the process.

It is also often claimed that good data representation and cleaning contributes more to a successful analysis and a good classification model than the actual learning algorithm used. So put the techniques learned in this chapter into your standard toolbox – they are applicable on a huge variety of problems and form the basis of all data analysis tasks.

PART C

MACHINE LEARNING PROFESSIONAL

Machine Learning – Professional

This course is a mix of conceptual and hands-on. This course is designed for anyone that will build Machine Learning models in RapidMiner.

Assuming that you've gone through the Applications and Use Cases Professional and Data Engineering Professional courses, under the RapidMiner Training repository that you've created, open another folder named ML Professional. Open 3 subfolders named data, processes and results under ML Professional folder for the use for this course.

We are one step closer toward our goal of churn detection based on customer properties. In the previous chapter we readied our data to pass it on to a learning algorithm. Now, let's learn. OK, one more recap: we have "historic" data (collected in the past). We have customer properties, including age, gender, and last actions. Finally, we know whether they churned or not. In other words, for each data row we know the true values of our target variable.

This makes the problem a supervised learning problem. Now it is time to devise methods for deriving outcome from a customer's properties. This process is called learning or training. The output of a learning algorithm is a model. Some models are sets of rules such as: if the customer is female and did not buy anything in the last 2 weeks then she will churn soon. Other algorithms use mathematical formulas, such as:

```
Churn_probability = time_series_last_purchase / (3*Age)
```

and others use entirely different approaches. In the following chapters we will get to know some learning algorithms.

Now, assume that we want to scan through our current customer database and find those customers that are likely to churn in the near future. Of course, since we don't know the future, we cannot know for sure, but we can apply our model on the current

customers and predict whether they will churn or stay loyal. Because we don't know the true outcome beforehand for these customers, we call this data set new data. Figure 1.1 illustrates the complete workflow of training and application.

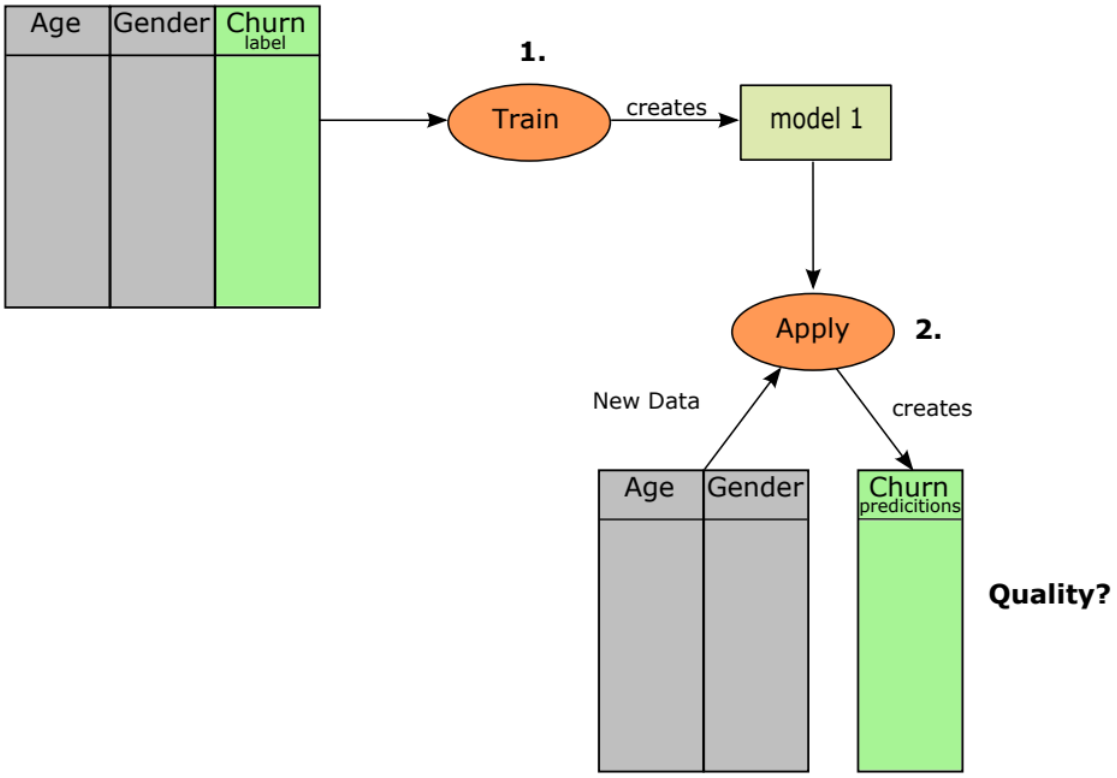


Figure 1.1 Model Training and Application

1. Our Very First Model: k-Nearest-Neighbours

There are thousands of modelling algorithms. In this course, we will introduce some of the most frequently used. As a starting point, we will work with a very intuitive one.

Imagine that your boss gives you the customer database with churn information, i.e., our training data, and a handful of new customers. She then asks you to tell her whether the new customers are likely to churn. What would you do if you didn't know anything about modern analytics? Probably you'd go through the list of new customers one by one, and for each find existing customers with similar properties (age, gender, etc.). Once you have a match with certain existing customers, you would look at their

behaviour. If the majority of them churned, you would assume that the same will hold for the new customer. This intuitive approach is called k-Nearest-Neighbours, or for short, k-NN. Let's now formalize this concept.

1.1 k-NN Formalized

We just described the overall function of the k-NN algorithm – look at existing examples (neighbours) and classify new records based on similarities. The formal algorithm has several important fine-tuning parameters, though. First of all, you need to decide how many similar neighbours you want the algorithm to consider. This number is the k in k-Nearest Neighbours. Your choice for the value of k largely depends on the analysed data. We will learn techniques to find good values later on. The next thing to consider is our definition of proximity. If we want to find similar data, i e., the nearest neighbours, we need to define a distance measure. It is only with such a distance measure that the closeness between two points can be evaluated. Let's start to devise distance measures for two-dimensional numeric data. Look at the two left-most graphics in Figure 1.2. There are two data points – p and q. The most natural distance between the two is the length of straight, direct connection between them, as shown in the left-most graphics. This is called the **Euclidean distance**.

Another distance measure is the **Manhattan distance**, shown in the second graphic. In this case, to reach q from p you are only allowed to walk on parallels to the axes, no diagonal walking is allowed (picture the grid-like blocks of downtown Manhattan). In this definition, the distance between the two points is the total length of all rectangular parts of the path.

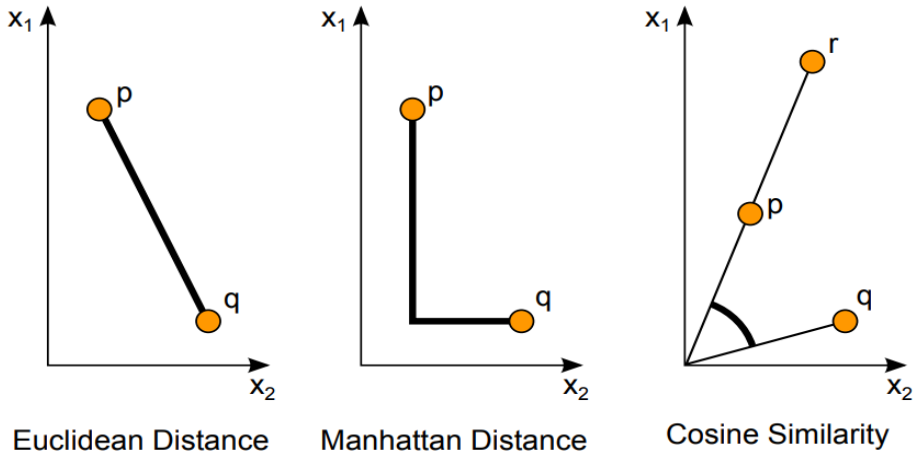


Figure 1.2 Distance Measures

The Manhattan distance and the Euclidean distance are also called L1-Norm and L2-Norm, respectively. Their calculation formulas are as follows:

- **Manhattan distance** (L1-Norm): $(p_1 - q_1) + (p_2 - q_2)$ in the two-dimensional case, or $\sum_1^m (p_i - q_i)$ in the general, m-dimensional case.
- **Euclidean distance** (L2-Norm): $(p_1 - q_1)^2 + (p_2 - q_2)^2$ in the two-dimensional case, or $\sqrt{\sum_1^m (p_i - q_i)^2}$ in the general case.

Finally, the **cosine similarity** is a third measure that comes in handy when the ratios, not the absolute values, of the attributes are important. For example, if you want to analyse texts and want to characterize documents by number of appearances of important words, then the word counts are obviously dependent on the length of the documents. If two documents discuss the same topic, and document p is two times as long as document r, then the word counts in document p are twice as high, while the ratio of important words in respect to the total word count will be the same. This is shown in the right-most graphics of Figure 1.2 (imagine that x1 and x2 represent the occurrences of two words in a document): p and r have different word counts, but the ratio is the same, such that they are both positioned on a straight line starting at the origin of the coordinate system. Document q has a different ratio. Now, how can we

treat p and r the same, while separating them from q? The Euclidean or Manhattan distance obviously won't do that. But if we look at the angle between the lines from the origin to the data points, then the angle is 0 between p and r, but quite large between p and q. This angle is what the cosine similarity measures, making it a good measure in the context of text processing.

1.2 Majority Vote and Weighted Voting

Now that similarity has been defined, we need to discuss how the final decision is made. If in a 3-NN setting two of the most similar customers are loyal, and the third one churns, we would by default perform a simple majority vote, i.e., predict loyal with a probability of 2/3. Have a look at Figure 1.3 for a visualization.

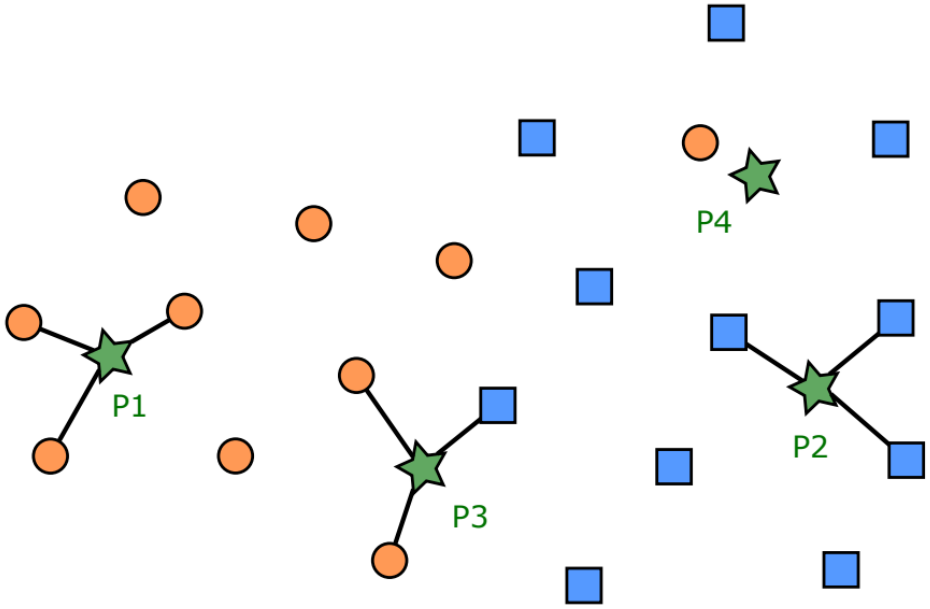


Figure 1.3 k-NN with k=3

The three nearest neighbours of p1 are all orange, so p1 is predicted as orange. Analogously, p2 is predicted as blue, while p3 is also predicted as orange. But what if the churning reference customer is very similar to the new customer? Shouldn't it get a higher weight, maybe even turn the prediction to churn? This behaviour is called weighted voting and is favourable on some data sets, while on other data sets a simple majority vote works better. You will see in just a moment how to evaluate which of the two options works better.

1.3 The Impact of k

Let's stay a bit longer with Figure 1.3 and this time discuss the point p4. It is close to an orange data point. That data point, however, looks a bit out of place here in the middle of blue land, so maybe it is erroneous data. That said, let's see what happens if we keep k at three: in that case, 2 of it's neighbours are blue, and one neighbour is orange. So, we predict blue and the erroneous orange data point is ignored. But what if the orange data point is valid and the "true" value of p4 should also be orange? Then we have to reduce the value of k. If k is 1, then it is the closest neighbour of p4, and p4 predicts as orange. In general we can say that higher values of k create smoother prediction boundaries and ignore stray data points. This is often a good choice if you have very noisy data. Lower values create very detailed models. They can handle complex data, but are prone to errors induced by noisy or unclean data. We will come back to this issue in section of Overfitting (and the Impact of k).

1.4 What is the Best Configuration for k-NN?

We have seen that cosine similarity is a good distance measure when dealing with text data. In all other cases, unfortunately, the only way to find the best measure is trial and error. On some data sets the Euclidean distance works best but on others, the Manhattan distance.

The same is true when setting the value of k: the optimal value depends heavily on the data. You must try different values, as illustrated in the preceding section. In the following chapters we will do just that – learn how you can test and benchmark different settings.

1.5 Nominal Values and k-NN

What we discussed in the previous sections has one precondition: it only works with numerical values. You can't easily put nominal values into a coordinate system and measure the distances between them.

To solve this, k-NN can use a simple heuristic: if the values of a nominal attribute are equal for two examples, then the distance is assumed to be 0, otherwise it is 1. Then, using those numerical attributes, you can apply the Euclidean distance. In RapidMiner, this method is called **mixed Euclidean distance**.

1.6 The k-NN Operator

In RapidMiner, the k-NN algorithm is implemented in the k-NN operator. Its parameters are pretty straight forward:

- k: The number of neighbours on which to base the decision. See above.
- measure type and measure: the distance measure to use. We have discussed the most important ones above. For the other options, consult the operator documentation.
- weighted vote: enable or disable weighted voting as discussed above.

As input, k-NN expects a data set with a label. On the output ports it delivers a model at the mod port, and the unchanged input data at the exa port. The next section discusses models in detail.

2. Models in RapidMiner

Until now, everything that flowed through or was produced by our operators was data, or Example Sets to use RapidMiner language. Now the k-NN operator is going to deliver us a model. We have defined models as something that can be used to create predictions. (In our context, a model tells us whether a new customer will churn or remain loyal.) Earlier we depicted a model as a set of rules. Generally speaking, a model contains all the information necessary to create predictions, and uses that information upon application.

So k-NN delivers such a model. As shown in Figure 1.1, we can now apply the model on a data set. In RapidMiner that is done with the Apply Model operator. It expects a model at its model port and data at its unlabelled data port. On the right-hand side, it delivers the data with added predictions at the labelled data port and the model at the lower model port.

2.1 Applying a Model in our Process

In our process, let's create a k-NN model and then apply it to the corrected training data. Figure 2.1 shows the process setup.

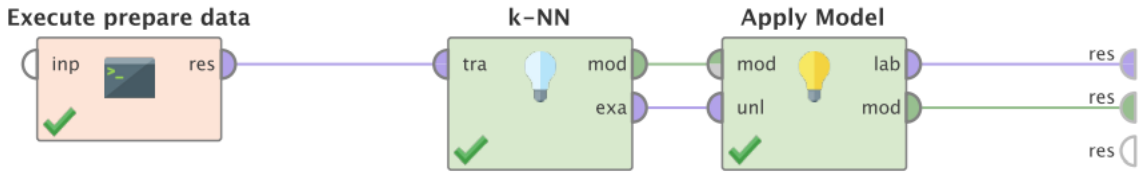


Figure 2.1 Training and Applying a k-NN Model on the Corrected Training Data

We'll leave the k-NN's parameter defaults, i.e.,k is 1 and the *MixedEuclideanDistance* is the distance measure of choice.

Exercise 1. Understanding k-NN and Predictions

It's time for you to train and apply the k-NN model.

- Start a new process and save it as train and apply k-NN on training data
- To start with the corrected data, simply drag and drop the prepare data process from its entry in the repository panel to the Process panel. An **Execute Process** operator will be created.

  ***prepare data process was built during the Data Engineering Professional course. You can replace this with the data set churn data_cleaned.xlsx*

  ***prepare data process used here were saved during Section 2 in the Data Engineering Professional course.*
- Add the **k-NN** and **Apply Model** operators to your process. Leave all settings at their defaults.
- Run the process and inspect the results:
  1. How many results does the process deliver? How can you access them?
  2. Open the *ExampleSet* result. Where do you see the predictions?
  3. Why are all confidences either 0 or 1? *Hint*: think about how confidences are created in k-NN and how you configured the k-NN operator.
  4. How many correct predictions did the model create? How many wrong predictions? *Hint*: You can use the Filter menu in the top right of the view.
  5. We said that a model "contains all the information necessary to make a prediction". For k-NN, what is that information?

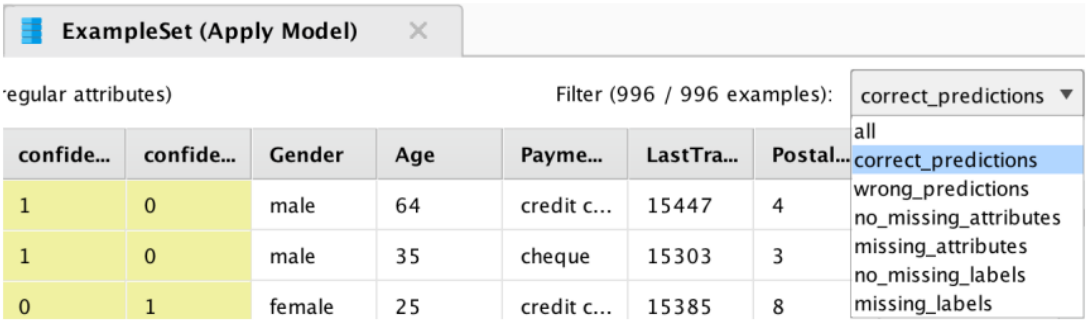
Answer to Exercise 1

1. If, as shown in Figure 2.1, both output ports of Apply Model have been connected to the process output, then the process delivers two results – the data with predictions and the model. You can access the results by selecting the respective tab in the Results view.

2. The predictions are reported in the prediction(Churn) attribute. If you switch to the Statistics tab, you can see that this attribute has the role **prediction**.

3. Remember that confidences are an indicator of how confident the model is when delivering a prediction. There's a confidence rating for each case, churn and loyal. They are listed in the confidence(churn) and confidence(loyal) attributes, respectively.
   In k-NN, the confidence results from the ratios in the majority vote. Our k-NN operator is configured with k = 1, so the "vote" is performed by a single neighbour, who is, of course, either 100 % churn or 100 % loyal. Because of that, in our example there won't be any fractions in the confidences.

4. A prediction is correct if it matches the customer's churn behaviour. Since we applied the model on the training data that still contains that information, for each line we can easily compare the prediction with the Churn attribute. In the visible top 20 rows there is not a single error. To check the complete data set, you can configure a filter via the Filter menu in the top right. Select wrong_predictions to see only erroneous predictions. Since the list is empty, our model predicted the behaviour of all customers correctly. (Whether that's a good result will be discussed in the following sections).



5. To make a prediction, the k-NN model searches for similar examples to the one being classified. Therefore, to make a prediction, it needs to contain the complete training data. Without the complete set, it cannot find the similar instances. Apart from the training data, no additional information is needed.

In this last exercise we learned that the k-NN model needs the complete training data to perform a prediction. When the model is trained, the only action is remembering the data. All the work of finding the nearest neighbours, calculations, etc., is done during application, when the new examples are classified. Hence, training is fast and application is relatively slow. This is different from most of the other model types, which do all their work during training. (We'll learn about them later.) This approach, delaying the hard work until application time, is called *lazy modelling*.

3. Validating the Model

In the last exercise we applied k-NN model on the training data. We saw that all predictions were correct, which, at first glance, seems like a fantastic result. But let's reconsider: the k-NN model was trained on training data. We kept the default setting of k = 1 and applied it on the same data. What does that mean? Well, if we search for the closest data point during application, for each data point we will find the point itself! So no surprise that we didn't see any prediction errors. But how would our model perform on new data that is not part of the training data? In other words, how well does it generalize? To answer this question, we'll introduce the concept of **overfitting**.

3.1 Overfitting (and the Impact of k)

Have a look at Figure 3.1(a). It shows a data set with a diagonal boundary between the blue and red classes. Some local anomalies – different-colored circles and some speckles – can be seen as noise. Noise is generally a bad thing, and we usually don't want to include it in our models. But that's what happens if the model is too detailed, e.g., with 1-NN. Figure 3.1(b) shows the predictions of a 1-NN model, where it is clearly visible that every noise speckle results in a dirty spot in the predictions.

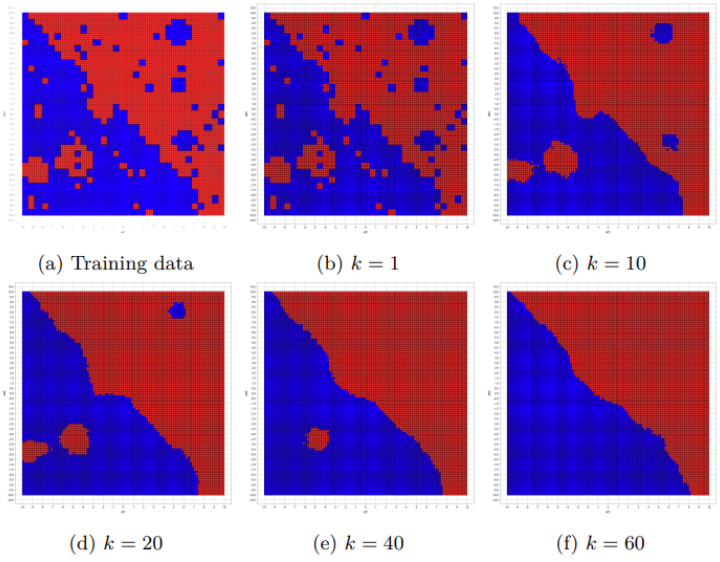


Figure 3.1 Effect of Changing k in k-NN

If k is increased slightly, the noise begins to smooth away. As k grows larger and larger, local effects play less and less of a role, until even the biggest local anomalies vanish (somewhere around k = 60). In other words, with low values of k the model is quite complex and can model the training data perfectly. However, it does not generalize well because it includes all the noise.

Then, around k = 10, we see a good compromise where the local effects of the training data are modelled, but the noise is cancelled out. This looks like a good general result.

Finally, though, we have to consider this. For larger k values the model becomes too simple to capture the essence of the data – both the generalization error and the training data error are, well, large. Figure 3.2 illustrates this behaviour. With increasing model complexity (smaller k), the training data is described perfectly and in detail (low training error), but with model details that we consider as noise (high test error). Decreasing the complexity (increasing k) results in the effects described above.

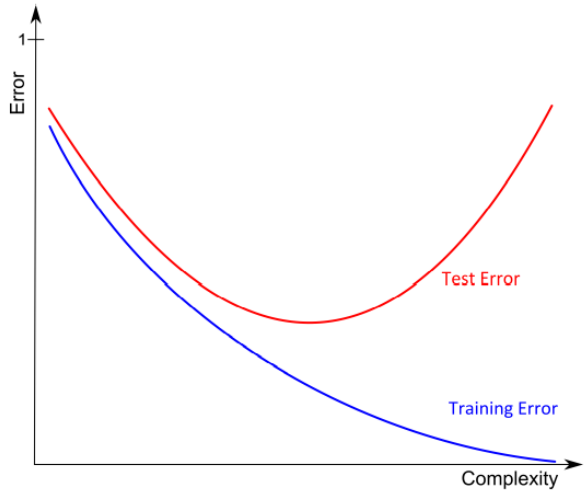


Figure 3.2 Test and Training Error vs. Model Complexity

The effect of modelling training data in too much detail, such that the model does not generalize anymore, is called **overfitting**. Overfitting is an issue not only for k-NN, but virtually all modelling algorithms. Some, like k-NN, have built-in parameter to influence the model complexity. With other algorithms it is necessary to manually adjust the input data so that it contains just the right amount of detail to describe the problem.

3.2 Validation Patterns

Here's the takeaway from the previous section. It is crucially important to test a model not only on training data, but on new data that has not yet been used for training. In a perfect world we would have a dedicated data set for training and a second set to test the model. In reality, though, data is often a scarce good, and we are left with just a single data set. In the following sections we will devise patterns to perform good and meaningful validations, even in this suboptimal setting.

3.3 Split Validation

If we have only one data set, but need one for training and one to test the model, what can we do? The easiest thing would be to split the data in two, so let's do that. Depending on the context, this technique is called either split validation or hold-out validation. By splitting the data, we can train the model and then calculate the so-called out-of-sample error on the independent test sample.

Figure 3.3 shows the workflow of such a validation. It looks complicated at first sight, but it's just an extension to Figure 1.1 with an additional data splitting and testing step. A common ratio for splitting the data is to use 70% of the complete data for training and the other 30 % for testing the model. The split can range from 60-40 to 90-10, however.

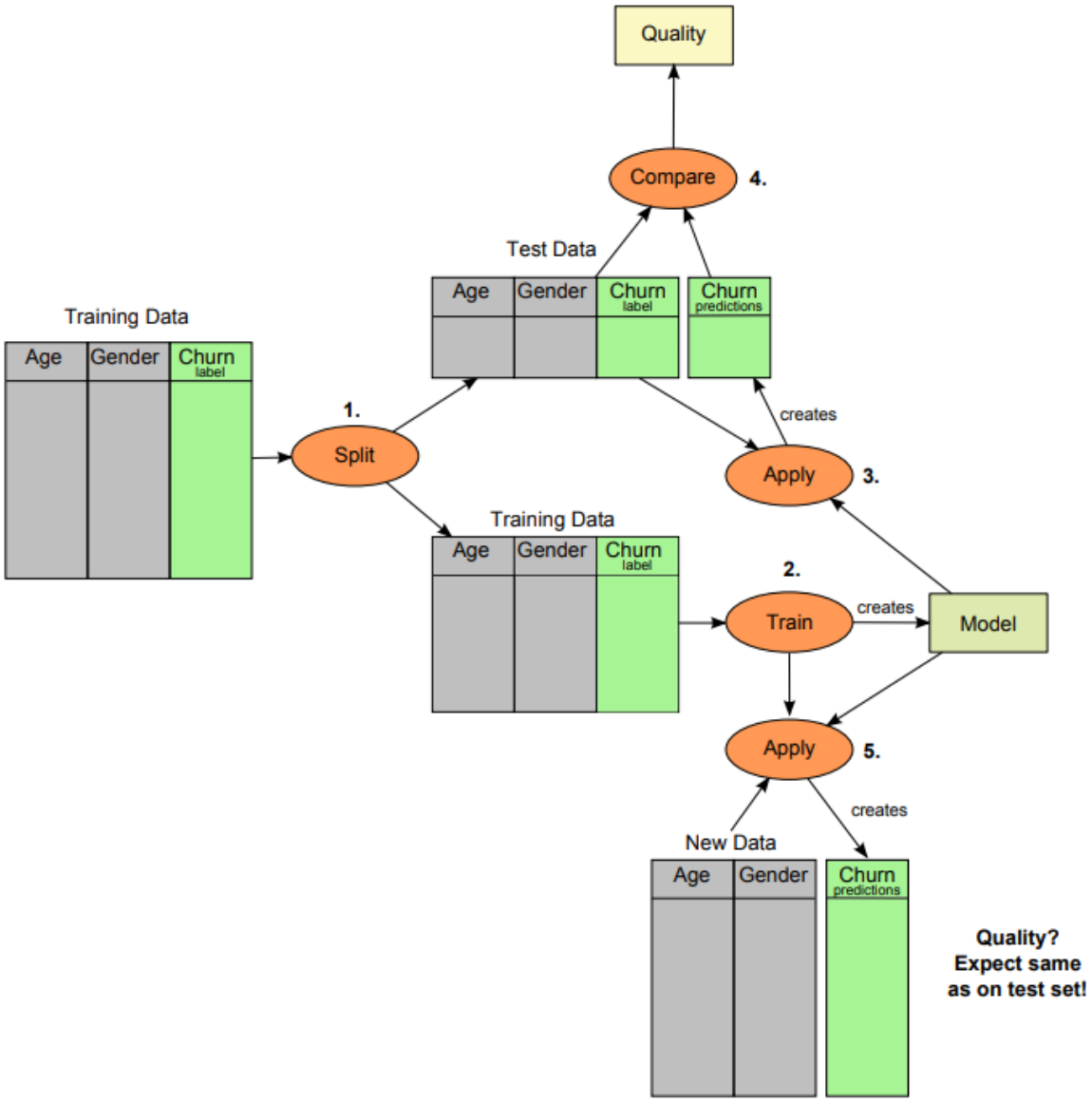


Figure 3.3 Model Training with Validation Step to Estimate Performance

3.4 Split Validation in RapidMiner: the Split Data Operator

In RapidMiner, what we just talked about can be implemented using the Split Data operator. It does exactly what the name suggests – you define the split ratio and the operators cuts the data into parts. Take a look at Figure 3.4. The operator takes the parameter partitions where you specify the sizes of the splits. Click the partitions edit button, and in the resulting dialog press the Add Entry button. Create as many entries as you need data sets – in our case that's two. In each row enter the ratio of the complete data set for that partition. For the proposed 70-30 ratio, we enter 0.7 and 0.3.

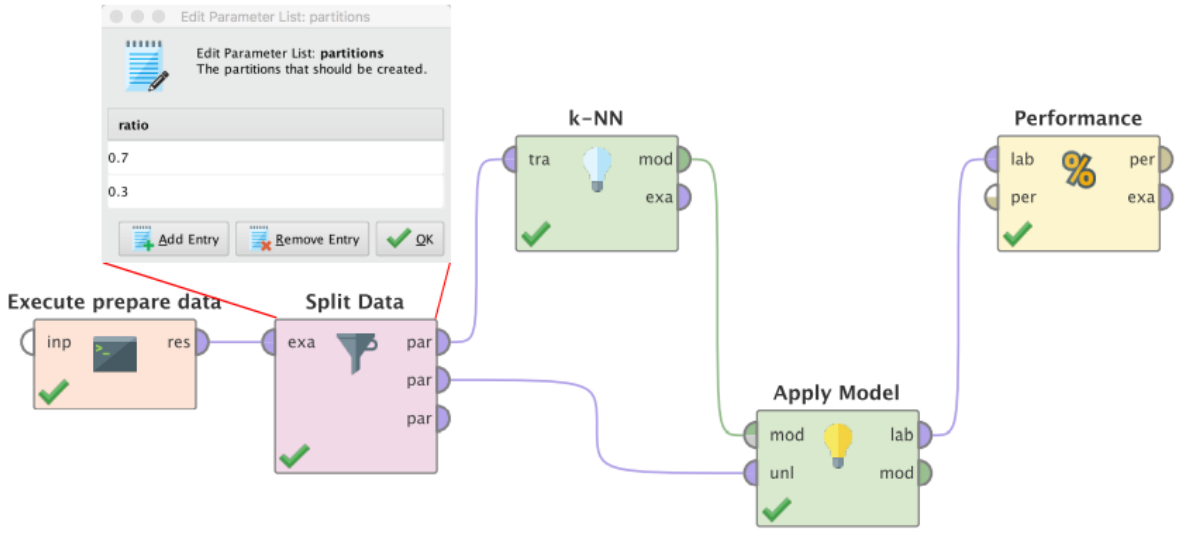


Figure 3.4 A Split Validation with Split Data Operator

RapidMiner delivers the split results at the output ports of Split Data operator in the order that you specified them – in our example the first input port delivers an example set with 70 % of the data. We want to use this training data to train the k-NN model, so let's connect it accordingly.

The sum of the ratios in Split Data's partitions parameter does not need to equal 1 (although that is considered best practice). Entering e.g., 1 and 0.5, is the same as 0.67 and 0.33, however RapidMiner will issue a warning.

As soon as Split Data's first output port is connected, another one appears, delivering the other 30 % of the data. To test the model, we need to apply it on that data set. As you hopefully remember, we need Apply Model for that.

We could now just connect the output of Apply Model to the process output and inspect the results as we did before. But simply looking at the data and manually counting correct and incorrect predictions is a very manual process – and no fun to do repeatedly. We need some kind of automation here. That's where the Performance operator comes into the game. It compares the model's predictions with the known labels and reports the fraction of correct partitions as the accuracy in the process results. The accuracy is available at the top output port labelled per as performance. The lower "exa" port delivers the unmodified input data.

On its input port the Performance operator expects a data set with both the original label and predictions created by a model – otherwise it couldn't do its job of counting correct vs. incorrect predictions.

Exercise 2. Split Validation with Split Data

Now it's time for some hands-on validation! Reproduce the process shown in Figure 3.4. Use the text of the previous section as a guideline. Have a look at the infobox for general hints.

When finished, run the process and inspect the results. Try to find the model's accuracy.

Process Design and Breakpoints. When designing larger processes, it's a good idea to check the output as you add each new operator. Even if the process already contains multiple operators, you can take a peek into the intermediate results using breakpoints. If you specify a breakpoint after an operator, the process execution pauses after the operator and displays its output in the Results view. Once you're ready to continue the process, just click the Play button again. To define a breakpoint, right-click the respective operator and select Breakpoint after. You can also press F7 while having selected the process. You can use the same methods to remove the breakpoint. Breakpoints are very useful for debugging larger processes and understanding their exact behaviour.

Answer to Exercise 2

After completing your process design and running it, RapidMiner displays the accuracy in the PerformanceVector (Performance) tab of the Results view.

The result of the split validation is shown below. The accuracy, i.e., the fraction of correctly classified examples, is stated in the confusion matrix.

accuracy: 58.53%

	true churn	true loyal	class precision
pred. churn	81	53	60.45%
pred. loyal	71	94	56.97%
class recall	53.29%	63.95%	

As you can see below, the 1-NN model got just a bit more than half of the examples correct, making it hardly better than a random guess for predicting churn or loyal customers.

Exercise 3. Overfitting

In the previous exercise we saw that the 1-NN model hardly performs better than random guessing. However, when we first applied the model in Exercise 1, we got 100 % of the examples right. What's the difference between the cases and what does it reveal about the model?

Answer to Exercise 3

In Exercise 1, we applied the model on the training data; we can also say we measured the in-sample error. This time we applied the model on a dedicated test set and hence are measuring the out-of-sample error. Because they differ so much, it is a strong indicator that the model is heavily overfit.

We already know how to control the grade of overfitting in k-NN (by modifying the value of k). If you like, you can try different values for k and see what happens.

3.5 Quantifying Prediction Performance

We have learned about model accuracy – the percentage of your data correctly classified. But the model performance results contain much more information than that. Probably the most important additional information is the confusion matrix. The confusion matrix lists the true versus predicted values.

Take a look at your results or at the confusion matrix generated from Exercise 2:

accuracy: 58.53%

	true churn	true loyal	class precision
pred. churn	81	53	60.45%
pred. loyal	71	94	56.97%
class recall	53.29%	63.95%	

In our data set, you can see from the true churn column of the confusion matrix that we have 152 customers that did churn (the sum of the values in the column). Of those 152 churning customers, our model predicted 81 correctly, listed in the first row labelled pred. churn (read that as predicted as churn). The other 71 have been wrongly detected as loyal customers; they are listed in the row labelled pred. loyal.

That said, we know that 81 of 152 churning customers were correctly detected or recalled. This gives us a recall of 81/152 = 53.29%.

Remember the use case – we run a web app store and want to detect customers that are likely to churn so that we can take further action to retain them. Hence, we can define churn as our positive class and loyal as the negative class. Now we can say that the 81 customers in the top left cell of the confusion matrix have been classified as positive, which is the truth. This makes them true positives (TP). Those 71 customers incorrectly classified as loyal in the second row, i.e., negative, are the wrong prediction and therefore false negatives (FN).

We can apply this pattern to the second column labelled true loyal. Read it as you did for the first column – 147 customers are loyal, the 94 of them in the pred. loyal row,

i.e., have been predicted as loyal, are true negative (TN). The 53 in the upper right have been wrongly classified as positive (read: they churned), and are thus false positives (FP). With this column we can calculate the recall of the loyal class.

Figure 3.5 shows our new naming schema.

	true churn	true loyal
pred. churn	**TP**	**FP**
pred. loyal	**FN**	**TN**

Figure 3.5 Confusion Matrix with TP, FP, FN, TN

So far we have talked about the columns. With the column-wise calculated recall we answered the question: which fraction of a class does my model recognize? Let's now focus on the rows. Each row groups all customers that were predicted as the same class (or outcome): the first row represents all prediction of churn, the second row all loyal predictions. Now we can pose another question: if my model says that a customer will churn, what's the probability that the model is correct? This information is called **precision**.

Let's recap accuracy, precision, and recall with an exercise.

Exercise 4. Precision, Recall, and Accuracy

1. Formulate the meaning of precision, recall, and accuracy in natural language (no math).
2. Rewrite the definition of recall for both classes, using the terms TP, FP, TN, FN.
3. Record the formula for accuracy.
4. Record the formula for precision for both the churn and loyal classes.

Answer to Exercise 4

1. **Recall** answers the question: If a customer is going to churn (or remain loyal), what is the probability that the model will detect it?

   Precision answers the question: If a customer is classified as churn (or loyal), what is the probability that the prediction is correct?

   Accuracy provides overall information: Given any prediction, what is the probability that it is correct? Or: Given any customer, with which probability will it get classified correctly?

2. **Recall for the churn class** (positive class in our example) is: $\frac{TP}{TP+FN}$

   Recall for the loyal class (negative class in our example) is: $\frac{TN}{FP+TN}$

3. **Accuracy** is $\frac{TP+TN}{TP+FP+FN+TN}$

4. **Precision for churn class** (positive class in our example) is: $\frac{TP}{TP+FP}$

   Precision for loyal class (negative class in our example) is: $\frac{TN}{FN+TN}$

Now that you've learned several measures for quantifying prediction quality, let's revise the process that actually performs the measure calculation.

3.6 A Cleaner Implementation of the Split Validation

If we look at our process, it's a bit cluttered. We have this Split Data operator followed by a modelling algorithm, i.e., k-NN, and then the model application and performance calculation.

While this setup works, it would be nice if instead we had just a single operator to perform the validation. The Split Validation operator is our answer.

The Split Validation operator expects the complete data set as its sole input. It then splits the data into the training data and test data, just as the Split Data operator did. You specify the ratio to use as training data using the split ratio parameter.

Referring to Figure 3.6, the operator's output is threefold:

- the validated model on the mod (model) port
- the unmodified input data at the tra (training data) port
- the performance at the ave (average) port

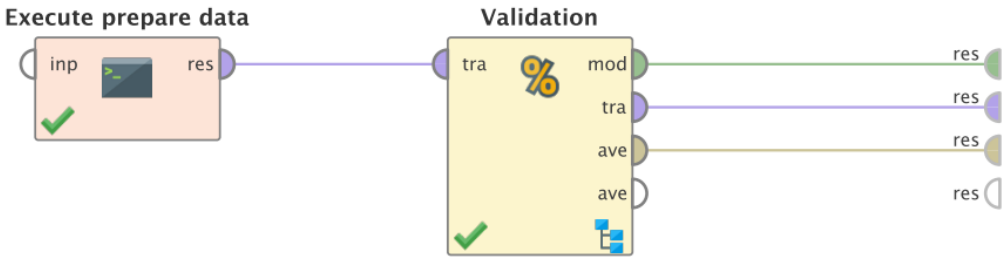


Figure 3.6 Validation with Split Validation Operator

You might wonder how the operator knows which model to validate. Hopefully you remember the Subprocess operator that we discussed during Data Engineering Professional course. If you look closely, you'll notice that Split Validation and Subprocess operators have the same blue icon in the lower right. It indicates that the operators contain a subprocess, and actually, Split Validation contains two.

Double-click the operator to open it and see the two subprocesses – refer Figure 3.7 for reference. The left subprocess is labelled training. Split Validation automatically delivers the training data, i.e., 70% of the input data, from its input port. Our goal is to create a model with this data and deliver it to the mod (model) port. We can use the k-NN operator, just as we did in the previous process.

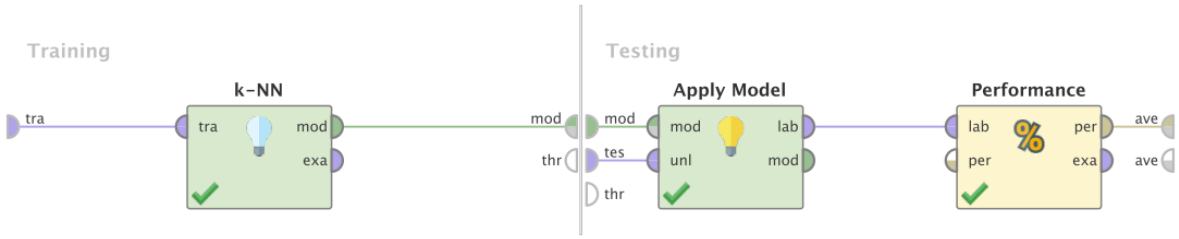


Figure 3.7 Subprocesses of the Split Validation

The model is we created is then transferred to the mod port of the right-hand process, which is labelled testing. There, the Split Validation operator delivers the other part of the data, the testing data, to the testing (tes) port. Finally, we need an operator to calculate the performance, just as we did with the process based on the Split Data operator, namely the Apply Model and the Performance operators.

The resulting process delivers the same results as the previous process. It may seem that we did not win much – the behaviour is the same and we need the same number of operators. But look at the outer process (see Figure 3.6): it contains only two operators, making the design very clear and intuitive.

As we will see in the next section, it also allows us to easily switch validation patterns without making changes to the process structure.

Exercise 5. Split Validation

Create a new process and validate the k-NN operator with the Split Validation operator.

Answer to Exercise 5

See Figures 3.6 and 3.7.

3.7 Cross-Validation

Previously we learned about split validation and discussed several performance measures. Split validation is much better than testing the model only on the training data, but it still has one drawback. Imagine that by chance we split the data in such a way that all the “easy” examples (easy to classify, meaning that even a simple model would classify them correctly) are in the test data. Then, even though the model may not work well on the border cases, it would get most customers in the test data right and we would overestimate our model, thinking that our model can be applied to new data for prediction with good accuracy, when that is not true. The problem also exists in the other direction – if many “hard” cases are in the test data then we underestimate our model. The essence of this is that the split validation still does not guarantee a solid and robust estimation of the model performance.

The way to overcome this shortcoming is to somehow repeat the validation several times with different random splits. Specifically, we can create n different combinations of training-to-test data ratios and then n different models on the training data (n being an integer value, for example 10), which we then test on the respective test set. If we could guarantee that each example is part of exactly one test set, then we would have the perfect validation pattern – we’d leverage all our data for use in testing and estimating the prediction quality.

In data science this widely used pattern is called cross validation. It consists of the following steps (see also Figure 3.8):

1. Split the data into n equally sized, non-overlapping sets.
2. Train a model on the union of the first n − 1 sets (leave out the last set).
3. Test the model on remaining set and record the performance (e.g., accuracy).
4. Repeat steps 2 and 3, each time using a different subset for training (leaving out a different set each time).

5. Average the performance values of each iteration and return the average.

6. Optionally, train a final model on the complete data set. This is the final model which will be used for prediction (we will use it only if its estimated performance turns out to be good).

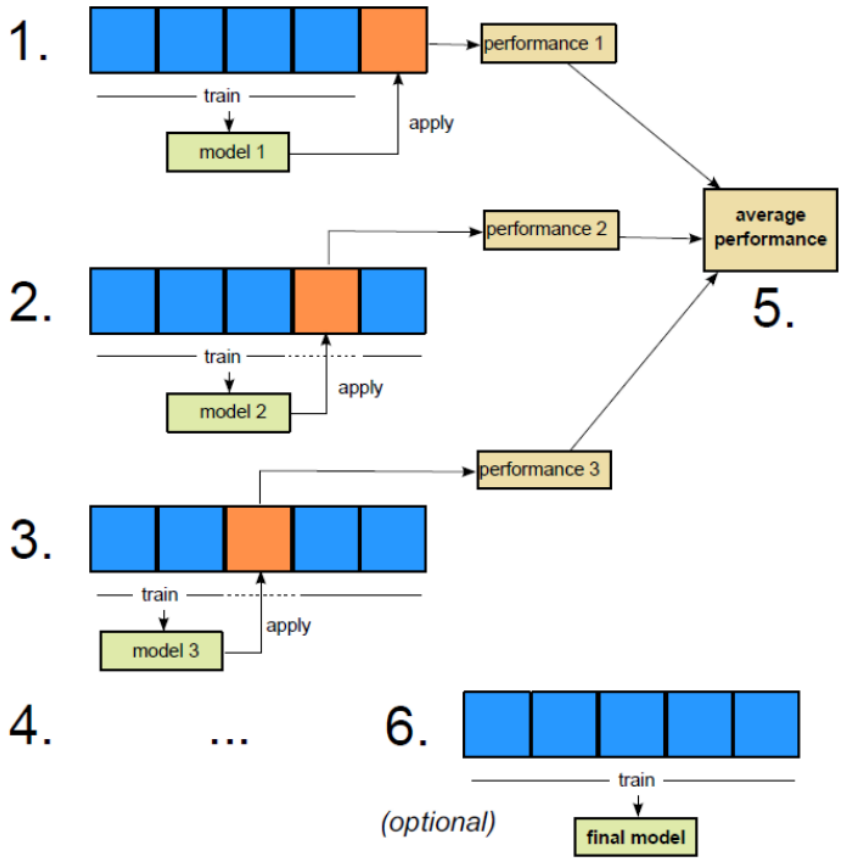


Figure 3.8 Schematic Cross Validation

Let's analyse this pattern in depth.

As we mentioned before, by repeating the train and test process several times we reduce the danger of "bad splits" and get a robust performance estimation. The more iterations we use in the cross validation, the better the estimation. A widely accepted standard value is 10, but you can increase to 20 for a better estimation or reduce to near 5 if you just need a rough estimation. Obviously, the more iterations that are calculated, the longer the execution time of the cross validation.

3.8 Stability of Cross Validation

The cross validation does not only provide a more solid estimation of the average model performance, it provides some important additional information. If we devise a learning algorithm like k-NN and use it on our data, we want to ensure that it delivers constant results by generalizing well. We do not want it to react to small changes in the data with vast performance fluctuations. By using different splits, we induce small changes to the data our algorithm sees in each run. So if we observe the changes of the model performance over the iterations, we know whether the algorithm is stable.

The fluctuation in several data points is known as the standard deviation. After performing a cross validation our goal is not only high accuracy (or other performance measure), but also low standard deviation, which indicates a stable model.

3.9 What Does Cross Validation Really Validates (and What Not)

What exactly does cross validation validate? Does it test a certain model? Something else? Let's have a look. Each iteration tests a single model, but after all are complete, we have created and tested 10 different models. What they all have in common, though, is that they were generated on similar data sets using the same algorithm. So in the end, the cross validation does not validate a certain model; instead, it evaluates whether a certain learning algorithm works well and is stable on the given data.

Remember that the described algorithm (above) lists an optional step to train a final model on the complete data set. This will probably be the best model since it can leverage all data, but, we can't test it. Because we used the complete data for the training, there's no test data remaining.

Is that a bad thing? Not at all! In earlier steps we tested the method used to create the model. When 10 models deliver a certain performance on the test data, we can expect that the final model performs similarly on any data we present to it.

3.10 Prerequisites for Using Cross Validation

There is one important prerequisite to creating valid results with cross validation. As we just discussed, we are validating a learning algorithm to create a model. That means that all steps that are part of the model generation must use only information from the training data. If there is any part of the model generation that leverages the test data, then we are cheating and can run into an overfitting problem.

So far we have generated our model using a single step (in form of the k-NN operator). This is about to change. But let's first see how we implement cross validation in RapidMiner.

3.11 Cross Validation in RapidMiner

Remember that we mentioned that the Split Validation operator allows for easy structural changes? Now comes the payoff for going through the trouble of rebuilding your process with that operator. To change your process so that it performs a cross validation you only need replace the Split Validation with the Cross-Validation operator (best practice suggests you should save your process with a new name like "cross-validate k-NN").

To replace the operator, you could delete the Split Validation operator and then configure the Cross-Validation operator from scratch. Or, you could leverage RapidMiner's Replace Operator functionality. To use it, you need to know the Cross-Validation position in the operator tree. If you're unsure, search the tree and remember where in the hierarchy it resides. Hint: Validation/CrossValidation.

- Right-click the Split Validation operator.
- Select Replace Operator.
- See Figure 3.9 or navigate through the menus to find the Cross-Validation operator.

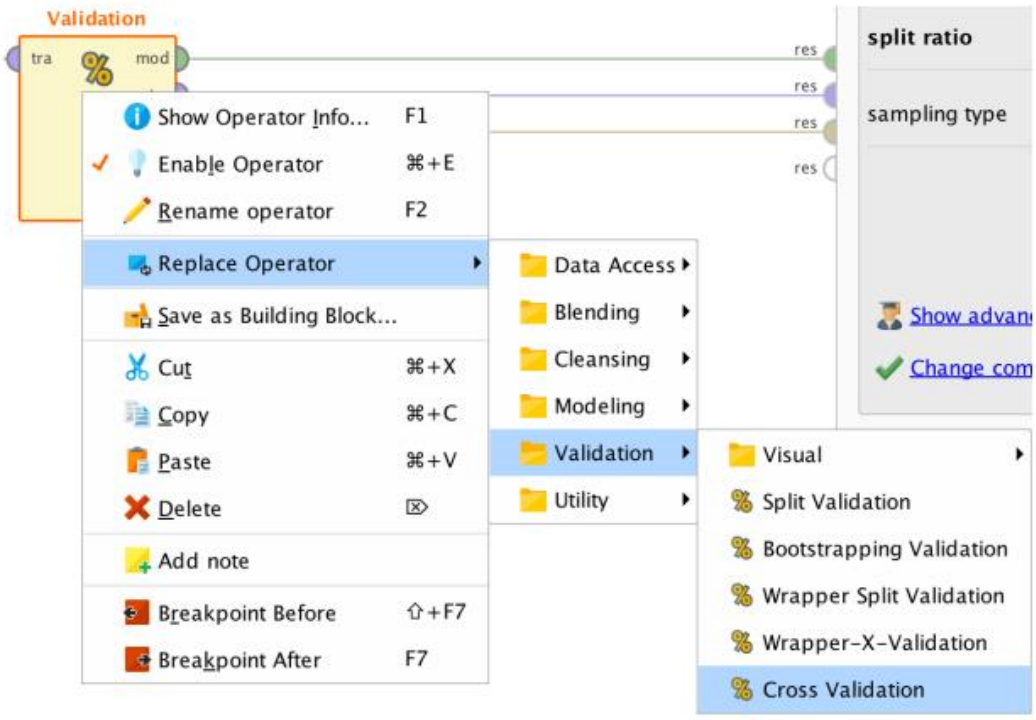


Figure 3.9 Replacing Split Validation with Cross Validation

The Cross-Validation operator is set up the same way as Split Validation – both have connections on the outside and also the two subprocesses on the inside. The important parameters of Cross-Validation are easy to identify and self-explanatory after the discussions above. The most important parameter is the number of folds. This is where you define how many iterations to perform. The default value of 10 is fine in most cases.

Exercise 6. Cross Validation

Save your process as cross-validate k-NN and replace the split validation operator with a cross validation operator.

Run the process and inspect the result. Where can you find the standard deviation?

Answer to Exercise 6

The standard deviation is listed as the +/- value to the right of the accuracy. The confusion matrix created by the Cross Validation operator contains the accuracy's standard deviation in addition to the average value.

accuracy: 59.64% +/- 5.95% (mikro: 59.64%)

	true churn	true loyal	class precision
pred. churn	298	199	59.96%
pred. loyal	203	296	59.32%
class recall	59.48%	59.80%	

The mikro-accuracy value listed next to standard deviation is the accuracy that is generated from the complete data set after cross validation occurs.

If you can't find the confusion matrix, make sure that you have selected the correct tab in the Results view. If there is no Performance tab, verify that the ave port of Cross-Validation is connected to the process output and rerun the process.

We have now learned how to properly validate our models and algorithms. The basic pattern is to split the data into training and test sets. Cross validation then perfects that technique by repeating the pattern several times, providing accurate performance estimates.

4. Reconsider k-NN

Now that we've got validation right, we still have only roughly 59% accuracy. Even by playing with different k values, we can't reach more than 66%, meaning that 34% of our customers still get classified wrong. That's better than random guessing, but we can do better!

The following section describes one operation that is crucial to creating good k-NN models.

4.1 Normalization

Have a look at the left part of Figure 4.1. When asked which of the orange points has the larger (Euclidean) distance from the blue point, what would you say? Most people answer "point C", which is how it looks. But in fact you should answer with another question: "what's on the axes"?

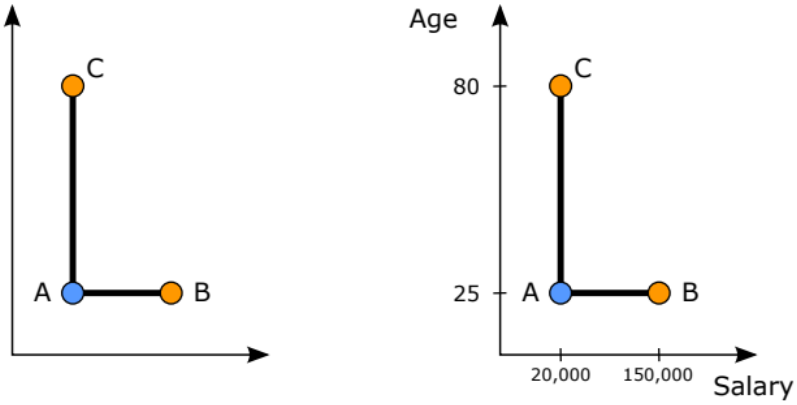


Figure 4.1 The Scaling Problem

This question becomes important if you look at the second sketch, where the axes are properly labelled. The vertical axis shows age, in the range of 0 and 80; the horizontal axis displays the salary range between 0 to 150, 000. Now what seemed to be a small distance between points A and B turns into a huge value of 130, 000, while the optically larger distance in the vertical direction turns out to cover only a difference of 55.

What's the takeaway of this thought experiment? Scale matters!

This is especially true for algorithms like k-NN, which rely on distances to calculate the model. In the example above, the salary would completely dominate the distance calculation. On a scale in the thousands, a trivial age-induced contribution of 55 doesn't make a big difference algo-wise – for the individual person, though, an age difference of 55 certainly matters!

Since we cannot adjust the algorithm, we need to modify the data such that age and salary have a similar impact. The key to this trick is normalization. Normalization means to adjust the range of an attribute to a predefined scale.

4.1.1 A Simple Approach: Range Transformation

A simple approach to adjusting ranges is to map the lowest value of an attribute to 0 and the highest value to 1. This method is called **range transformation**. By applying it to all attributes we guarantee that they all have a similar scale.

The upper part of Figure 4.2 visualizes this approach. There, you see the age attribute as the horizontal axis, each customer on it represented by a dot. The numbers on top of the axis show the original age values, the numbers below represent the values after applying range transformation. See how the lowest value 20 is mapped to 0, the highest one, 80, becomes 1. The other values are mapped such that the relative distances remain the same.

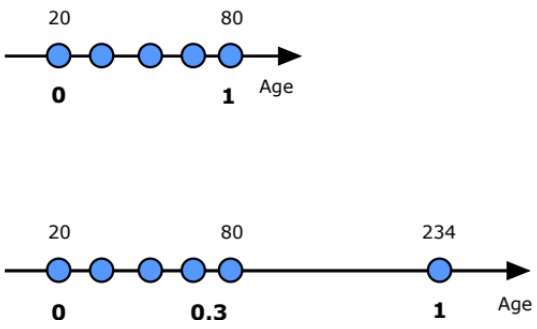


Figure 4.2 Range Transformation

The mathematical description of range transformation is as follows:

$$x_i' = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

where x_i is the transformed value, x_i the original value, and min(x) and max(x) are the minimum and maximum values of the attribute, respectively.

Looking at the lower part of the graphic you can immediately see one imminent problem of range transformation: it is very unstable with respect to outliers. One outlier can drastically change the complete transformation. In the lower part of Figure 4.15, the very old customer from our original customer churn data set is included. His age of 234 is used as the upper boundary of the transformation, pushing all other, good points into the range between approximately 0 and 0.3. Thus, the gain of using normalization is diminished.

This shows, again, the importance of careful data cleaning and outlier detection. But beyond that, more robust normalization techniques exist. One of them, the Z-Transformation, is covered in the next section.

4.1.2 A Better Approach: Z-Transformation

The aim of Z-transformation is to standardize the scales and ranges of attributes. But in comparison to the previously discussed range transformation, it takes a different approach: to normalize an attribute, a normal distribution is fitted. Then, the data is scaled and shifted such that the distribution has mean 0 and standard deviation 1.

Have a look at Figure 4.3. The maximum of the normal distribution is at the mean of the attributes' values. The standard deviation, visualized as a horizontal line at half the height of the distribution, indicates the scale.

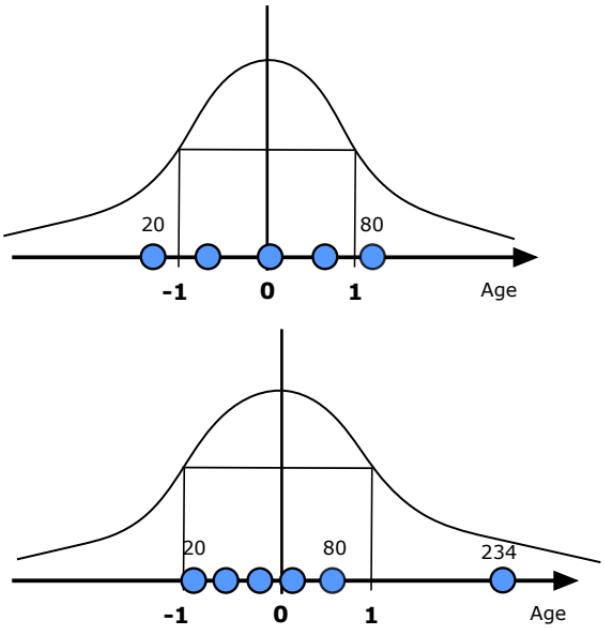


Figure 4.3 Z-Transformation

The mean is then mapped to 0, and the standard deviation to 1. Here is the formula:

$$x_i' = \frac{x_i - mean(x)}{stdev(x)}$$

How does the Z-transformation behave when it comes to outliers? Each data point, including outliers, influences the mean and the standard deviation of the data. However, the influence is much lower than in the range transformation (where the range directly relies on the most extreme data points). Look at the lower part of Figure 4.3 – the distribution only gets a slight right-shift.

Due to its greater robustness, Z-transformation is generally preferred over range transformation which is why it is the RapidMiner default.

4.2 Normalization and Nominal Attributes

The thing is, though, the two normalization methods can only be applied to numeric data. What happens when our data has nominal attributes?

If we remember the mixed Euclidean distance that we introduced in the k-NN, the distance of values in a nominal attribute is per definition, either 0 or 1. Thus, its range is similar to the one of a normalized numerical attribute (i.e. after the Z-Transformation). Therefore, these distances can be safely compared and RapidMiner needs no extra transformation for nominal attributes.

4.3 Normalization in RapidMiner

In RapidMiner, normalization is performed by the Normalize operator in the Cleansing/Normalization group. As input, it expects a data set. Output at the exa (example) port is the same data set, with the selected attributes transformed according to the chosen method. As always, the ori (original) port delivers the unmodified input data. We'll talk about the pre port in the exercise below.

Take a look at the Normalize parameters. The attribute filter type defaults to all, meaning that all numerical attributes are normalized. (You can select different attribute subsets from the pull-down.)

By default, Normalize performs the Z-transformation, but you have other options like proportion transformation and interquartile range. For example, if you change to the range transformation, you can use the default range (0 to 1) or set any other range that is suitable for your representation. Usually, changing the standard range is only necessary for visualization – from the algorithmic point of view there's rarely the necessity to do so.

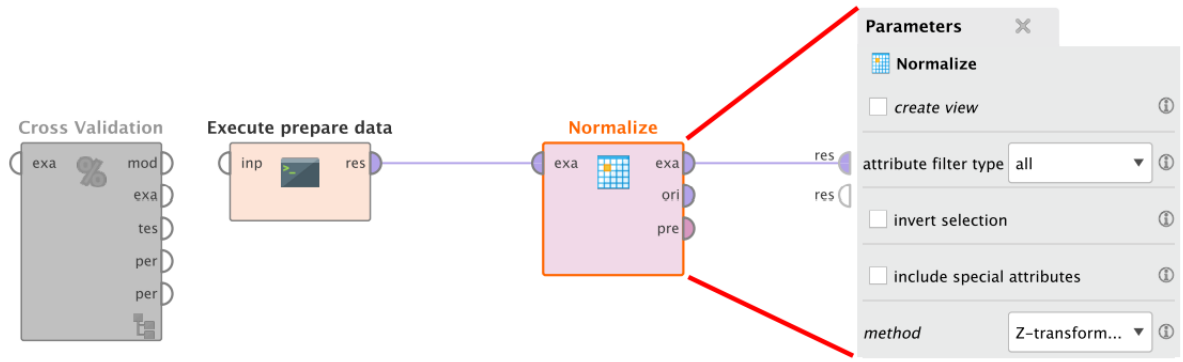
To get a grip on the operator, let's first try it out on its own, without the validation.

Exercise 7. Normalization

1. Save the process under a new name: normalize.
2. Deactivate the validation.
3. Add the Normalize operator directly after the data preparation.
4. On output, connect both the data port (exa) and the pre port, run the process, and inspect the results.
5. Try both the Z-transformation and the range transformation methods.

How can we check whether normalization has been applied correctly?

Can you guess what the output of the pre output is?



Answer to Exercise 7

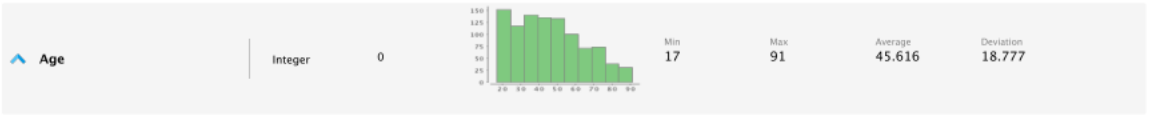
For the process setup, see figure shown in Exercise 7.

Check the results. First, you can see in the data view that the values of numerical attributes have changed. But to see that it's actually the correct change, switch to the Statistics tab.

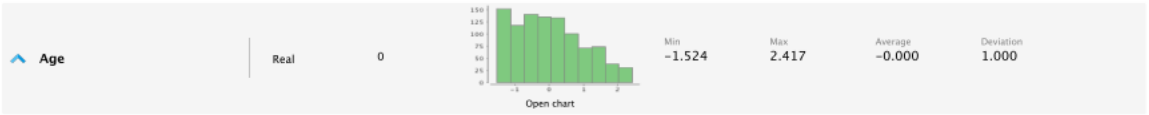
If you have just run the range transformation, you should see in each numerical attribute that the min value is 0 and the max value is 1. No statement about the mean or standard deviation is possible.

If you last performed the Z-transformation, the average of the attributes should be 0 and the standard deviation 1. No statement about minimum and maximum is possible.

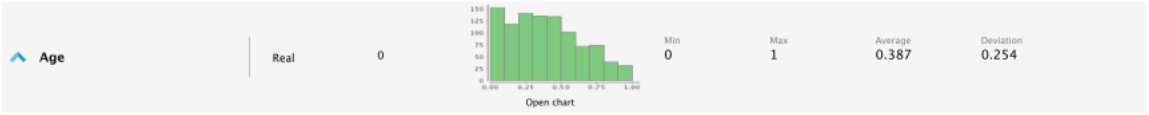
Now look at figure below which shows the statistics of Age after different types of transformation. Notice that the shape and distribution of the age attribute did not change at all. Only the range and scale differ between the non-normalized and the two transformed data sets. This means that by normalizing the data we did not lose any information.



(a) Original – untransformed



(b) Z-transformed



(c) Range-transformed

Now let's inspect the output of the pre port. First of all, it is time to find out what pre stands for: with your mouse, hover over the port and you'll see that it stands for

preprocessing model. Remember that preprocessing, in general, means to prepare the data for modeling. A normalization does exactly that, i. e., it morphs the data into the correct format.

Now what is a preprocessing model? Well this opens up a new can of worms, so we've dedicated the next section to it.

4.4 Pre-processing Models

To understand pre-processing models, we need to think of a model more generally – it's something that, according to some rules, modifies data upon application. The classification model applies prediction rules to add predictions to the data.

If you think about it, the transformation methods introduced in the previous chapters are quite similar to a prediction model. We processed the data to derive the scaling factors, i.e., min and max (in range transformation) or mean and standard deviation (in Z-transformation). By inspecting the data results of the normalization process, like we did in the exercise, we saw the result of applying these scaling factors.

But if we extract the scaling factors from the operator, we can later normalize any data set with the same scaling factors without the effort of crawling through the data set again to find min and max (or mean and standard deviation). This is where the pre-processing models come in. The pre output port delivers the scaling factors and transformation rules in the form of a pre-processing model.

Revisit the process from the exercise. Make sure that you select the Z-transformation as method and look at the output of the pre port. Compare Figure 4.4 for reference. For both of the data's numeric attributes, the output lists mean and standard deviation. Again, if you think about the Normalize operator, the information from the pre port is all you need to apply the transformation.

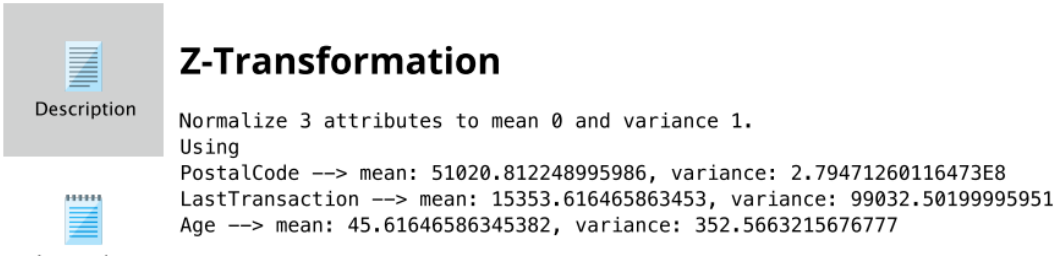


Figure 4.4 Z-Transformation Pre-processing Model

Exercise 8. Applying a Pre-processing Model

Let's take the pre-processing model for a spin:

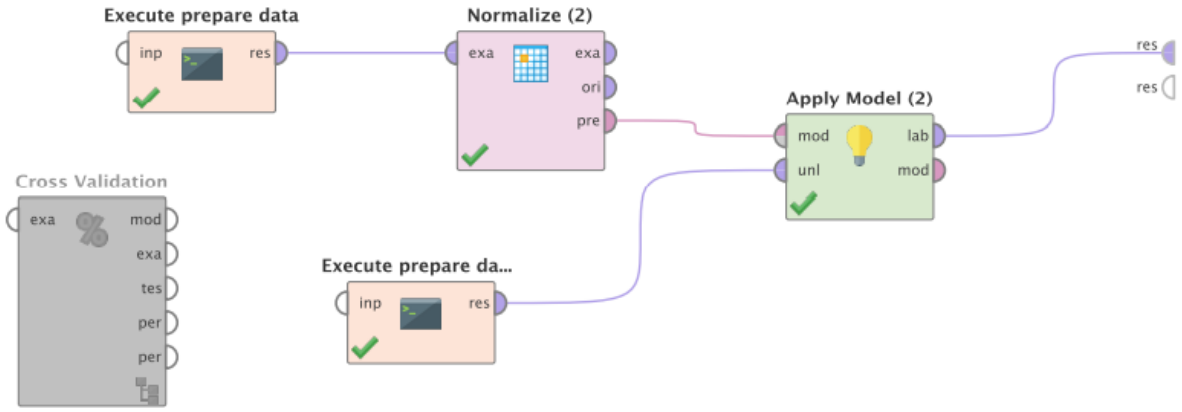
1. Load a second instance of the data by dragging a second data preparation process into the Design panel.
2. Apply the pre-processing model, generated by Normalize, on the second data set.
3. Set a breakpoint before Apply Model to explicitly see its input and output.

Then answer these questions:

1. What is the result of applying the pre-processing model? Which difference does it have with the original data?
2. Can you think of a situation where the pre-processing model is needed? (we will come to this later in the course)

Answer to Exercise 8

The process setup is shown below:



On execution you can see that in the end, the data that flowed through Normalize and the data that flowed through Apply Model are normalized (and in this case are exactly the same).

The pre-processing model is necessary when making a prediction on new data, different that the training one. Because the model expects a normalized dataset, a normalization transformation must be applied to it. However, the new data usually have different values for the means and standard deviations, that's why it is necessary to save the exact transformation used during the training.

Let's recap: In this and the preceding section we learned about different normalization methods and why they are important in the context of k-NN. We also learned about the Normalize operator and saw that it both normalizes the input data and, at the same time, delivers the scaling factors as a pre-processing model.

Why is it important to have pre-processing models and in which context we can leverage them? Read on.

4.5 How to Validate the Normalization

Now that you understand normalization, let's revisit the real reason why we introduced it. We raised the topic of normalization because we wanted to improve k-NN. So let's enable Cross-Validation again and integrate it with the normalization.

It is clear that we need the normalization operator in the process, but where to place it?

Your first idea may be to put it directly in front of the cross validation action. While that seems obvious and easy, it's not correct. As we saw, both variants of the normalization scan through the data to calculate the scaling factors. That actually makes it part of model generation, and therefore, part of the modelling method. As discussed, that means that the normalization must be included in the validation, i.e., the Normalize operator must be placed into the Cross-Validation.

So, the normalization must happen before learning the k-NN model. After you nod in agreement with the explanation in the previous paragraph, it becomes clear that the only viable position for the Normalize operator is on the left-hand side of the cross validation, in the training subprocess.

But simply adding it to the left-hand side is only part of the story. With it, the training data is normalized and k-NN is trained such that it works well on normalized data. But in the test process on the right-hand side, it is now applied on the original, non-normalized data. This won't work. So how do we get the test data normalized correctly?

Maybe add a second Normalize operator? That doesn't work. Remember that Normalize relies on the input data to calculate the scaling factors. When using the operator on the test data, you may get very different results compared to when using it on the training data. To illustrate this, look at Figure 4.15 and imagine that the training data contains data similar to the upper example and the test data is similar to the lower example, including the outlier. It is clear from the graphic that the normalization is different again, resulting in the same bad results, i.e., not normalizing the test data.

What we need to do is to transfer the scaling factors to the right-hand side and apply them there, on the test data, just before applying k-NN. Luckily, we just learned about pre-processing models. If we can somehow pass the normalization model from the left process to the right process, all will be well.

The missing link are the thr (pass-through) ports in the cross validation. If you connect anything to the top-most thr port on the left during training, it will be accessible at the top-most thr port on the right, in the testing. Hence, the pre-processing model should be passed to the test side and applied there with a second Apply Model operator. Our first try could be something like Figure 4.5.

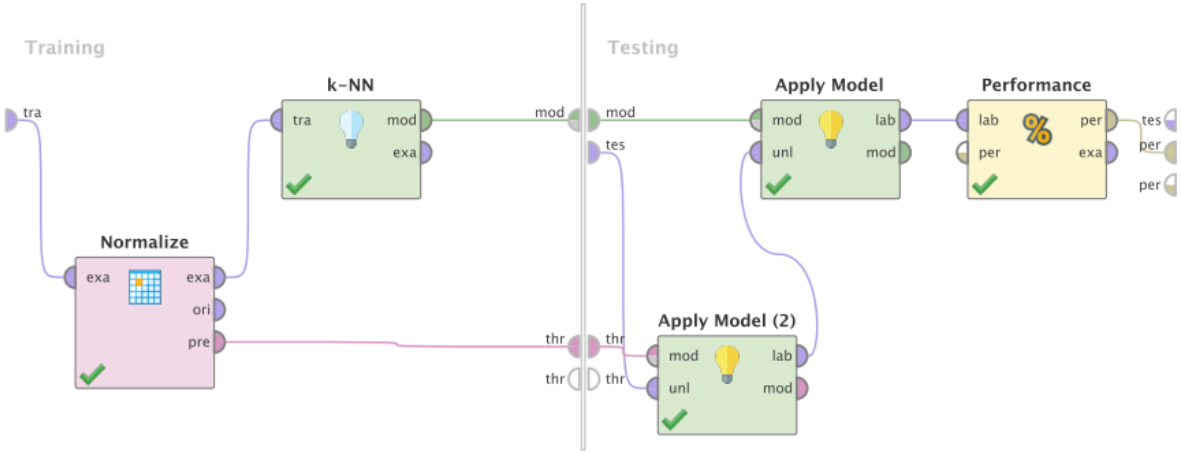


Figure 4.5 Validating the Normalization

This, however, looks a bit clumsy. We can solve the problem more elegantly with the Group Models operator, which combines several models into one. When the process is run, the contained sub-models are applied in the order in which they are connected to Group Models. This approach is shown in Figure 4.6. Be reminded, the order in which the models are connected to the operator are important!

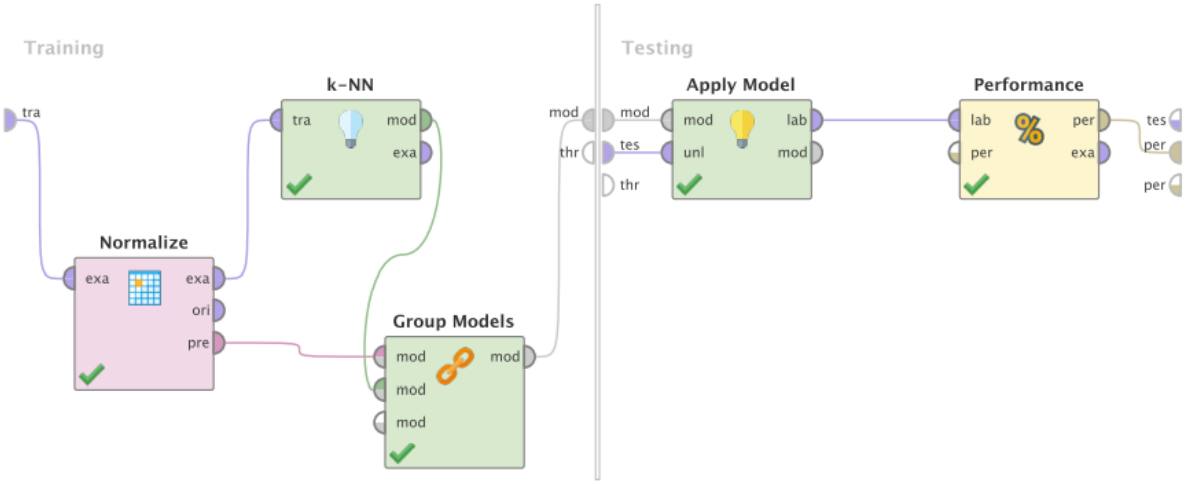


Figure 4.6 Validating the Normalization with Group Models operator

Figure 4.7 shows the output of running the process. This result was obtained by setting a breakpoint after the Group Models operator.

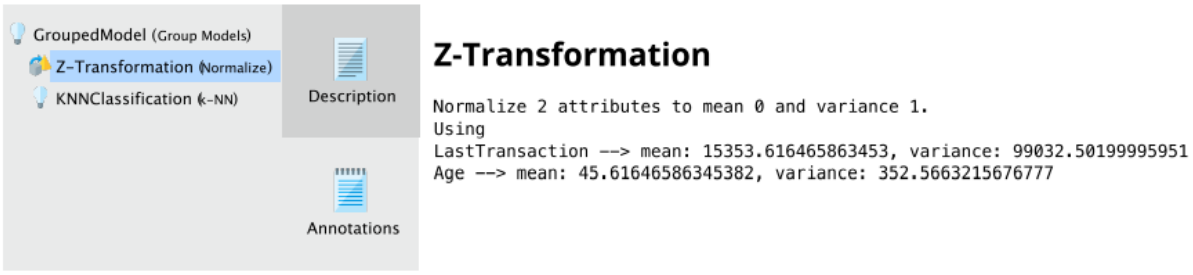


Figure 4.7 A Grouped Model in the Results View

Exercise 9. Validation of the Normalization: Group Models

Modify your process such that the normalization is properly validated. Look at Figure 4.6 for reference.

Inspect the result. Compare the accuracy to the earlier result.

Hint: On the top-level of your process, you only need the data preparation process and the Validation. All Normalize or Apply Model operators that we have added before can be deleted or deactivated.

Answer to Exercise 9

Look at Figure 4.7 for assistance.

If you don't see the performance, make sure that on the outside the avg port of the Validation is connected to the process output.

You'll notice that accuracy did not change significantly. Read on to learn why.

4.6 Normalization and Date Attributes

We are missing one last step to get the results that we expect – we need to explore a curiosity of our data. If you explore the data through Statistics tab, we saw that date_time is a sub-type of numeric. Why is that, you wonder? It's because under the hood, dates are represented as the number of milliseconds since Jan 1, 1970. As you can imagine, those numbers are quite large for the current date.

If you now consider what you learned about large-scaled attributes, it turns out that the LastTransaction attribute (of type date) completely dominates our analysis. The distance calculation of k-NN uses the number of milliseconds stored internally in that attribute. These numbers are in the billions, such that the other neatly normalized attributes fail to make any notable contribution.

But we performed a normalization, didn't we? Yes, but Normalize ignores date attributes (dates are not really numbers, only their computer implementation is), therefore we need to explicitly convert dates to numbers beforehand. How to do that?

Look at the Operators panel in the Blending → Attributes → Types group. In there, you'll find all imaginable type conversions – take a look at Figure 4.8. Our operator of choice here is Date to Numerical.

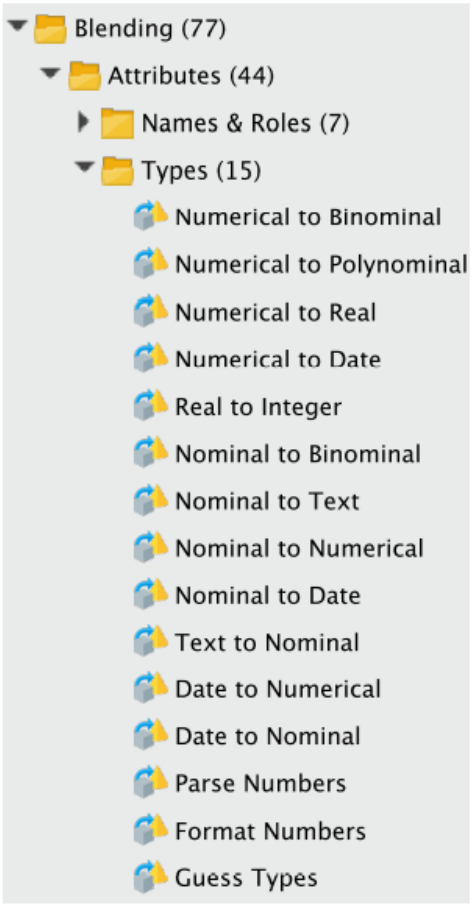


Figure 4.8 Type Conversion Operators

In the Date to Numerical parameters, you need to select which attribute to convert, a time resolution, and a reference point. You could, for example, calculate the day of the week by setting time unit to day and day relative to to week.

In our case, let's choose a resolution of days. Since we cover dates from several years, the day of the current year is not sufficient. We need to calculate the days since the beginnings of time (which is Jan 1, 1970 in the computer world and denoted as epoch).

Once that is done, LastTransaction is just another numeric attribute, which will be normalized together with the others. When you try this in the next exercise, you'll see that – finally – we have a decent accuracy on our dataset.

Date to Numerical does not depend on the data to calculate any conversion factors etc., it only relies on the current data row. Therefore, it is safe to apply before the cross validation, as shown in Figure 4.9.

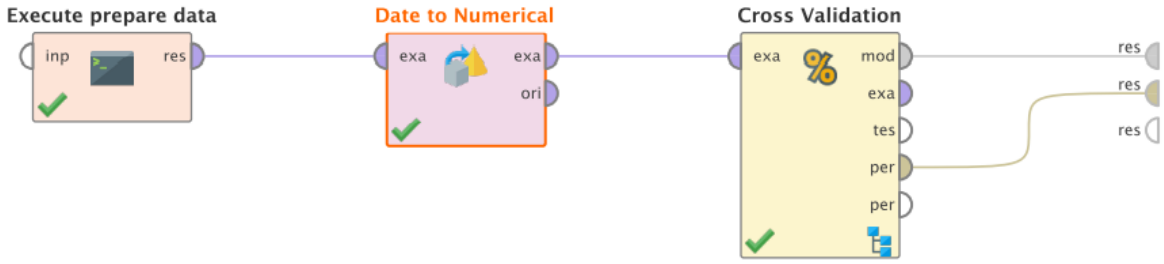


Figure 4.9 Converting the Date Attribute LastTransaction to Numerical

Exercise 10. Data Types Conversion

Add and configure the Date to Numerical operator to the process.

Run the process and inspect the results. How does the prediction performance change?

Answer to Exercise 10

Have a look at Figure 4.9.

As you can see in the result view after running the process, with the new configuration the accuracy increases to 68.37 ± 2.11 %. This is a vast increase over the 59 ± 5 % we achieved without the date conversion – we got not only an accuracy increase but a standard deviation decrease.

4.7 Putting Things Where They Belong

We put the Date to Numerical operator in front of the cross validation, which, as we discussed above, is a good position. But maybe there is even a smarter location.

Most algorithms benefit from converting the date to a numerical. Since we will use models other than k-NN in the future, it makes sense to include the date conversion into our standard pre-processing. This can be done by adding it to the data preparation process.

Exercise 11. Data Type Conversion

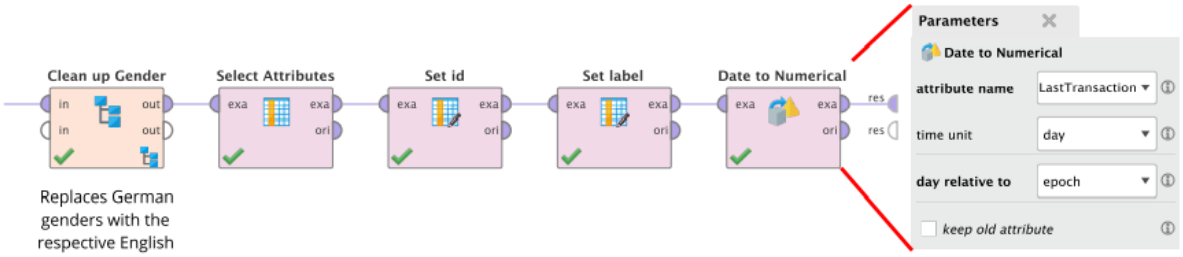
1. Remove Date to Numerical from the current process.
2. Open the data preparation process built during Data Engineering Professional course. Add Date to Numerical right at the end.

   Hint: if you followed the instructions in Data Engineering Professional course, your data preparation process should be called prepare data.
3. Open your validation process again and check that everything works as expected.

Answer to Exercise 11

You can open the data preparation process via the Repository panel. If you followed all the previous exercises, it should be under the processes subfolder.

After adding Date to Numerical, the process should look like:



5. Recap: k-NN and Validation

We made great progress in the previous sections:

- We introduced supervised learning, which describes the concept of deriving rules and models from past data to apply them on new, uncategorized data.
- With k-NN, k-Nearest Neighbours, we used our first algorithm to solve supervised learning problems. k-NN assigns a class to new data points by comparing it to the historical labelled data and predicting the class that is most common in the closest data points.
- We discovered that for k-NN it is very important to have a similar scale in all attributes. To achieve that, we investigated normalization. At the end of the chapter, we learned of the special care that must be taken for date attributes.
- It's bad practice to believe in unproven statements, and so we must validate our models to estimate their quality. We talked about split validation and the de-facto validation standard, cross validation. Both have in common that they split the data into test and training sets, training a model on one part of the data and testing it on the other part. Cross validation repeats this process several times to improve the estimation.
- We further discussed that cross validation not only validates a single model, but rather a modelling algorithm. That begs the question, what actually belongs to modelling? It's not only the actual learning process, but also all the pre-processing steps that leverage properties of the training data (such as the normalization). In that case, we concluded, pre-processing operations must be performed inside the validation. And so we were introduced to Group Models.
- Finally, we illustrated once more how subprocesses and included processes can use the common operations in several places by modifying previous processes.

In the following sections we will build on these concepts, introduce another learning algorithm and more pre-processing steps, and investigate other important analytics concepts.

6. Decision Tree

In this section, it is time to learn about another useful predictive model: decision trees.

To this point, our models have been rather abstract (though some, such as linear regression with its coefficients, have been quite informative). But in the very beginning of this course, we introduced models as a set of rules in the form "if a customer is older than 35 and male than he's likely to churn". This is exactly what the decision tree does. For an example of such a tree look at Figure 6.1.

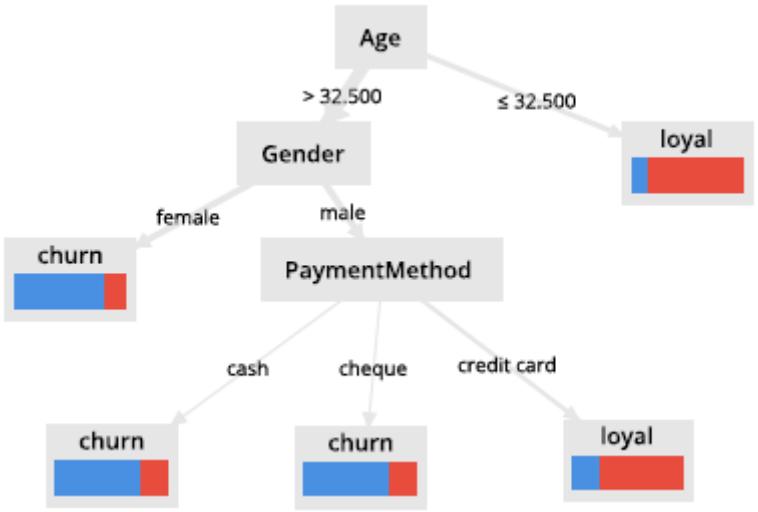


Figure 6.1 A Decision Tree Model

The tree consists of so-called nodes. The most prominent node is the top-most one, the root of the tree. From the root there are outgoing branches that lead to other nodes, that may again split into several branches. If there are no outgoing branches in a node, we call it a leaf.

How does such a tree make a classification model? If you have a new customer and want to know whether he will churn you have to pass him through the tree from the root to the leaves. The root is labelled Age, so we compare the customer's age with

the labels on the branches: if the customer is younger than 31.5 years we follow the right branch and end up in a leaf labeled loyal, meaning that we classify the current customer as loyal.

If he's older than 31.5 years we take the left branch. Here we end up in the Gender node. So before we can make a prediction we have to look at the customer's gender: if she's female we predict churn, and loyal otherwise.

You may notice that the leaves have a red-blue bar. The ratio of red and blue visualize how much of the training data ends up in the respective leaves when passed through the tree. Remember that we learn from the training data– we can consider it as our reference. So if many loyal customers end up in a certain leave, then we will classify new customers also as loyal with a high confidence – as in the right-most leaf in the example tree.

In the left leaf there are many churning customers, so here we predict churn with a high probability.

What about the middle leaf? Here there is no clear majority class, loyal just scraped into the lead. It may be worth trying to find another decisive attribute to split the leaf further. This could be for example the LastTransaction date. If we leave the tree as it is, then the prediction for customers in that leaf have a very low confidence.

6.1 Growing a Tree

In the section above we have explained the mechanics of a decision tree on a readily grown tree. But something must have happened to arrive at this ready-to-apply model.

So let's discuss the algorithm to grow a decision tree on our training data.

As for any new algorithm we should define our goal: in this case we could sum it up in the words pure leaves: pure leaves indicate that we found a set of conditions that separate the training data well, meaning that leaves are ideally either completely red

or completely blue. This is equivalent to the ”gain ratio” criterion, the default growth method for decision trees. Note that the Decision Tree operator can also be used for regression problems (predict numerical labels) by using the ”least squares” criterion.

Now that we specified the goal, pure leaves, let’s look again at the desired output: a tree. A tree is a hierarchical structure, and just as its real world equivalent, we can let it start growing at the root and grow more and more branches and leaves as necessary.

So, let’s find an attribute and a condition that makes a good first split: the decision tree algorithm looks at all attributes, one at a time, and tries out how pure a split with this attribute can be: for nominal attributes, it creates one branch for each value, e. g., male and female for the Gender and cheque, credit card and cash for the PaymentMethod. For numerical attributes it chooses the best possible split. Figure 6.2 shows all these splits.

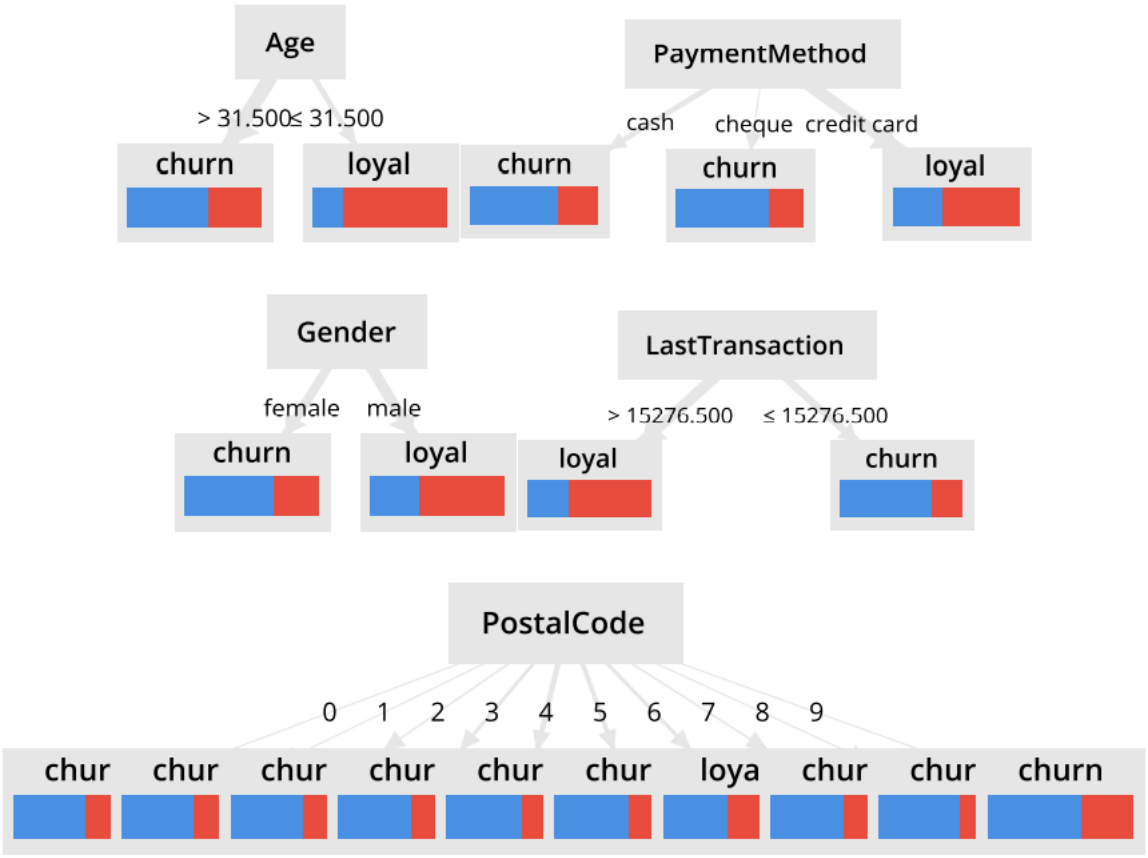


Figure 6.2 Splits on all Attributes of the Training Data

Given the splits we now select the split with the purest leaves as the root. For PostalCode all nodes have a nearly 50-50 distribution of churn vs. loyal, so it does not qualify as good attribute since we don't gain any information.

The other attributes introduce quite some skewness in their leaves, making them a bit purer. The winner is LastTransaction, which produces one completely pure node. If we choose this one, then we're done for the right branch. For the left-hand branch however there is still a 50-50 distribution, so we simply repeat the step from above on all customers that fall into the left leave of LastTransaction.

We continue splitting until either all leaves are pure, or until the leaves can't be further split. This results in a huge tree with nearly pure leaves and a humongous amount of branches. The result can be seen in Figure 6.3, where we can see that the number of nodes and leaves is extremely large.

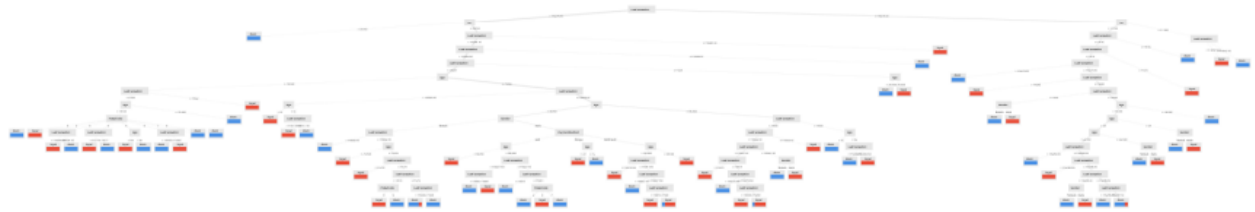


Figure 6.3 Fully Grown Decision Tree Model

The fully grown tree is very complex and certainly describes our training data very well. We have seen such behavior earlier, so take a step back and think: is such a detailed model a good model?

6.2 Pruning

Recall back two terms that we discussed earlier: overfitting and generalization. The fully grown decision tree has a very high capacity, meaning that it is very likely to be overfit and probably won't generalize, i.e., it won't work well on any data other than the training data.

The loophole is pruning the tree. Pruning is used to reduce the capacity of the tree by either cutting branches of a fully grown tree or by preventing the branches from growing in first place. The first variant is called post-pruning, the latter pre-pruning.

6.2.1 Pre-Pruning

There are several methods to prevent the tree from growing too detailed.

- **Minimal leaf size**: if a leaf contains less than a specified amount of examples after the split then the split is not performed.
- **Minimal size for split**: if a leaf contains less than a specified amount of examples then the algorithm doesn't even try to split it further.
- **Minimal gain**: a split is only performed if a specified minimal gain with respect to the splitting criterion is achieved. In non-technical language that means that the purity of the nodes must rise by a certain factor for every new split.
- **Number of prepruning alternatives**: This parameter plays a role whenever prepruning eliminates a node. In that case, the algorithm will try other nodes to see if they can be split, in order not to lose too much discriminative power.

All these means can be used to reduce the size of the tree during its creation.

6.2.2 Post-Pruning

In addition to preventing the tree from growing, it is also possible to prune it after it's grown.

In this case the algorithm iterates over all leaves and applies a heuristic to determine whether it is more advantageous to leave the leaves as they are, or to prune and combine them into their common parent node.

It can also work on the node-level: by replacing some of the nodes by its majority class (effectively eliminating a whole branch of the tree), the “misclassified” examples could be corrected by some other rule later on the tree, resulting on a less complex model with the same accuracy.

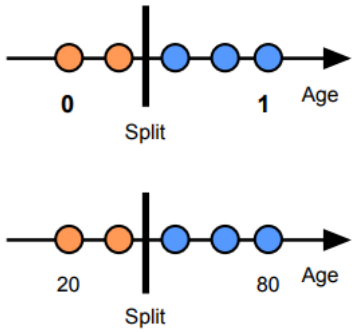
Exercise 12. Growing a Decision Tree

After lots of gardening theory it's finally time to grow our very own decision tree.

1. Validate the decision tree on our churn data.
2. Is it necessary to normalize the data for the decision tree?
3. Is it necessary to convert nominal or numerical attributes?
4. How often will a nominal attribute appear at most in a tree?
5. And a numerical one?
6. Compared to the linear regression, how well does the decision tree deal with outliers?

Answer to Exercise 12

1. The process is straight forward – all you need to do is load the data (through the data preparation process) and add a cross validation with nothing but the Decision Tree and the Apply Model and Performance operators.
2. No normalization is necessary for the decision tree. On numerical attributes it searches the optimal splitting value, but it doesn't matter whether the scale of an attribute is high or low – for higher scaled attributes the split value will just be higher, but it will still split between the same two examples. See figure below, no matter how the axis is scaled, the split always happens between the same two examples.



3. RapidMiner's implementation can deal with both nominal and numerical attributes. Usually it is faster, though, on nominal attributes, because here the split is clear: one branch per value. For numerical attributes several splits must be tried, which takes way more time.

4. When a split is performed on a nominal attribute then for each value of that attribute one new branch is created. For example, when you split on the PaymentMethod, then one branch is created with all customers who paid by credit card, one for those who paid cash and one for those who paid by cheque. That means with respect to the splitting attribute the branches are pure, and splitting again by the same attribute does not make sense.

5. For numerical attributes the case is different: if you split by Age < 30 in one node, you can easily split again in a sub-node by Age > 20 to get a leaf with all customers between 20 and 30. So to summarize: nominal attributes appear at most once in a tree, while numerical ones can appear more than once.

6. As we have seen, the linear regression can be completely distorted by one single outlier. The decision tree is more robust against outliers, since the splitting value is usually less heavily impacted or not even influenced at all by an outlier, as can be seen in figure 4.45 on the facing page.

6.3 A Word of Warning: Stability of the Tree

If you have children, you may have held speeches about the stability of trees yourself. Also for the purely digital decision trees it is important to keep in mind that they are quite unstable. Instability in the context of machine learning means that small changes in the data can cause huge changes in the model. Trees are valued even (or especially) by non-technical users for their ease of interpretation. One may be tempted to assume that the root of the tree is the most important attribute, but look at Figure 6.4. Those trees have been trained on the same data set, only different, random 10% of the data have been left out for training. The shape of the trees differs significantly, and in general even the root may change at times.

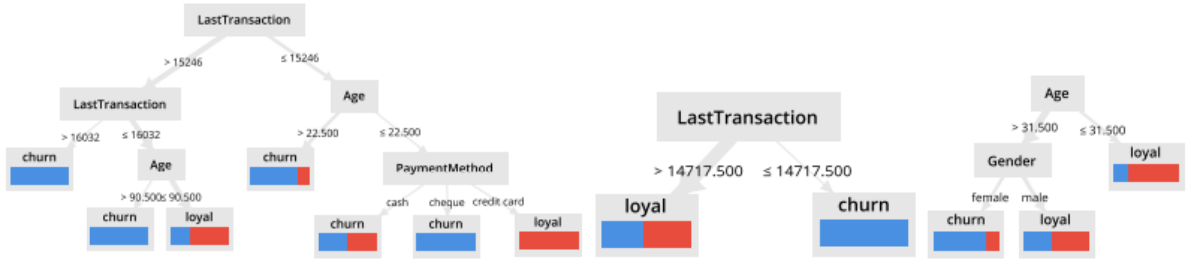


Figure 6.4 Three Decision Trees trained on very similar data sets

Despite its instability the decision tree often is a very good choice for classification problems. Just don't fall for the temptation of interpreting too much from it. There are also means of stabilizing decision trees, but that is part of the Machine Learning Master course.

7. Neural Networks

The core element of Neural Network is the perceptron. Similar to linear regression, the perceptron expects numerical input variables, assigns a weight to each input, and calculates a result from that. The main difference between this and linear regression is that it is not linear – the perceptron has a non-linear activation function. The activation function defines how (and whether) the signals are transmitted from one perceptron to another, similar as how the neurons are triggered in the brain. Have a look at Figure 7.1 for a schematic visualization.

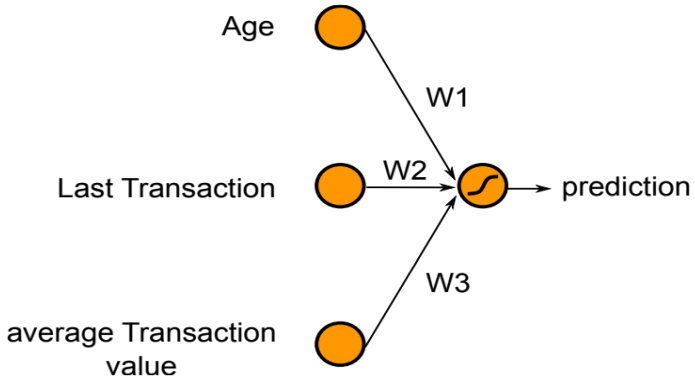


Figure 7.1 A Perceptron

The nodes on the left represent the attributes of the data; the middle node is the perceptron itself, which produces the prediction as output.

A single perceptron can solve some problems, but its true power is unleashed when combined with others. For example, you could train two or more perceptron, then pass their outputs to another perceptron that produces the final result – or is again passed to another layer of perceptron. Such a construct is called a neural network and is visualized in Figure 7.2.

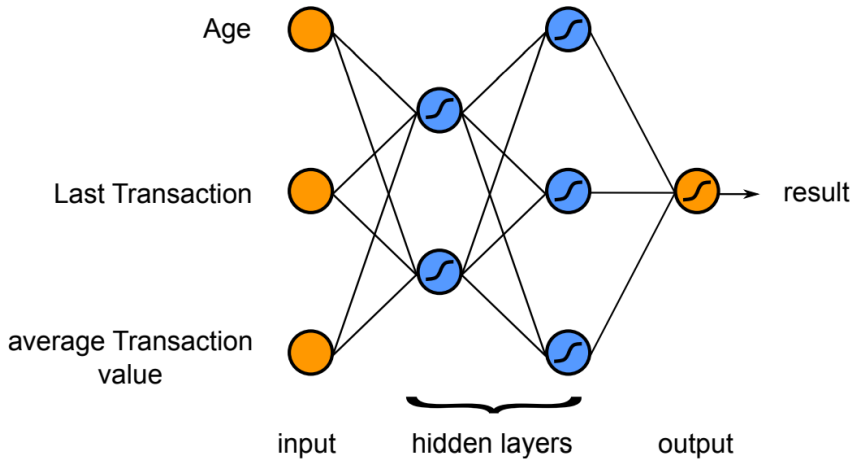


Figure 7.2 A Neural Network for Regression Tasks

As you can see, the results of each layer of nodes is passed to the next layer, until you finally reach the output nodes for the final prediction. Each perceptron assigns weights to its inputs (for the sake of readability, the weights have been omitted in the graphics). The small network visualized here has 15 connections, i.e., 15 weights, and 9 nodes, each calculating a non-linear function. While you can solve complex problems with this type of network, it's almost impossible to interpret the resulting model and understand the detail of what's going on.

As long as the model works well, we are only interested in its input and output. If we apply the model, that is also all we see. Therefore, the layers in the middle are called hidden layers. We can also refer to the neural network as a blackbox. We pass the input into the box, some magic happens, and on the other end we get the output. Figure 7.3 illustrates this interpretation of the neural network as a blackbox.

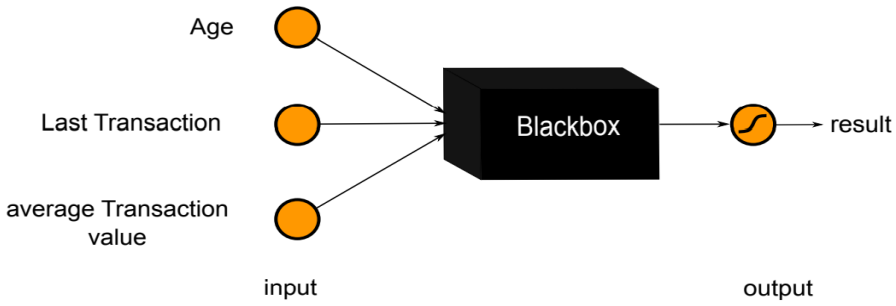


Figure 7.3 A Neural Network can be seen as a blackbox

But let's have a closer look. On one hand a neural network can be quite complex, and can describe arbitrarily complex data if you add enough hidden layers and nodes. As a result, it is capable of solving complex problems. On the other hand, such high model capacity comes with the danger of overfitting. Hence, for a neural network it is especially important to thoroughly validate the algorithm with a cross validation.

Furthermore, the neural network is limited to numerical input. That's quite obvious given the fact that each perceptron performs a regression and assigns factors to its input. The output of the perceptron is also always numerical, which raises the question of how a neural network could solve classification problems.

The answer is, a simple trick is used: instead of one numeric output for each label class, an output is generated as illustrated in Figure 7.4. During application, an output for each class is calculated, and the class with the highest value is predicted. Nominal input variables can be converted to numeric through dummy coding.

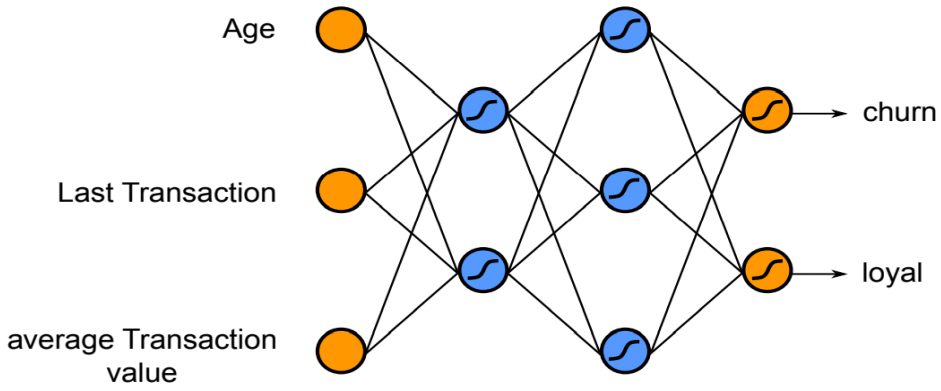


Figure 7.4 A Neural Network for classification problems

So far we have only described how a neural network works to make a prediction for a single example:

1. Start at the left, and pass the attributes of the example to the first layer of perceptron.
2. Each perceptron calculates an output, based on its input and its non-linear activation function.
3. Proceed the same way with the next layer, until the output node(s) at the right side of that network are reached.
4. Deliver the final output as the prediction.

Obviously, this application only works if you already have a neural network. How to create one is explained in the next section.

7.1 Training a Neural Network

To train a neural network, you must first specify the number of hidden layers and the number of nodes per layer. Then, RapidMiner applies the following steps:

1. Randomly initializes the weight of each connection.
2. Passes each example in the training data from left to right through the network ("forward"). The output is calculated and compared to the true label value of the respective example. Errors (with respect to the true output) are summed up.
3. Now backpropagation starts: from right to left ("backward"), adjusts weights of the connections so that the total error becomes lower.
4. Repeats the previous two steps until the error is acceptably low or no further improvement can be made.

7.2 Learning Rate and Momentum

Above you learned that the connection weights are adjusted and optimized iteratively. The learning rate controls how fast they are optimized, i.e., it controls the biggest leap that can be made towards the optimum.

Have a look at Figure 7.5. It sums up the problem of finding the optimal weights. The error is represented by the line, with the lowest achievable error (point A) at the very bottom. We start at a non-optimal point somewhere on the slope of the hill and try to reach the optimum point at the bottom.

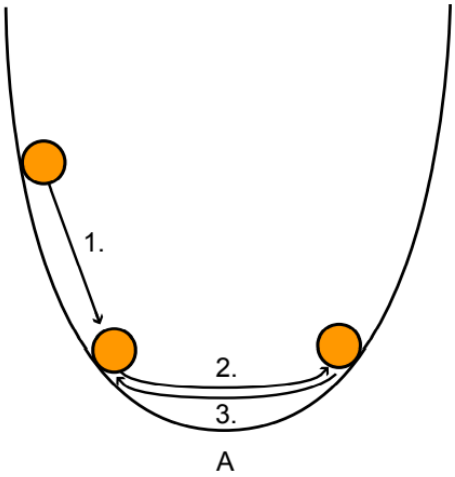


Figure 7.5 Learning Rate chosen too high

The learning rate controls the step size. If you choose it too low, the ball advances very slowly toward the bottom. If it is too high, the ball representing the current status oscillates around the optimum without ever reaching it.

As a best practice, start with a high learning rate and reduce it if you don't get satisfactory results with the original setting.

Have a look at Figure 7.6, there's another issue you may hit: local optima. A local optimum is a point that looks like the optimum when you're in it. Imagine being in a valley. You are surrounding by mountains, and it seems that you are at the lowest point of the world, even though the valley behind the next mountain is way deeper.

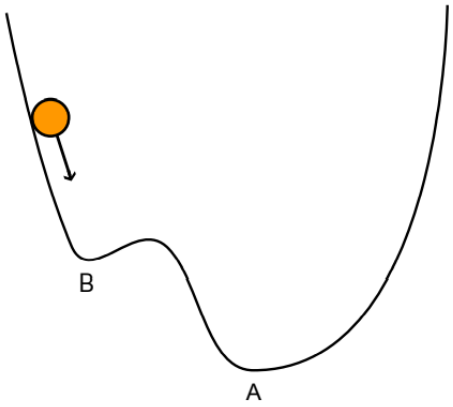


Figure 7.6 Momentum to overcome local optimum

To prevent the optimization from ending up in the local optimum, it needs some momentum. High momentum means that the ball keeps rolling uphill for a short time. In position B, that helps to overcome the local optimum. In position A, at the bottom, it may cause some additional oscillation, but in the end you'll level off.

Of course, there is a lot of math going on behind the scenes; we left it out for the sake of simplicity. The internet is full of introductions to neural networks in all levels of detail for a truly in-depth description. This conceptual explanation will be sufficient to use the RapidMiner operator effectively.

7.3 The Neural Network in RapidMiner

In RapidMiner, the neural network algorithm is implemented by the Neural Net operator in the Modelling → Predictive → Neural Nets group. The Perceptron operator is also there.

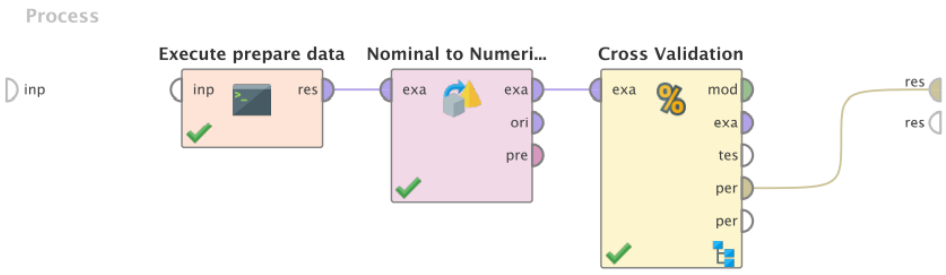
You can configure everything discussed above with Neural Net's parameters. Just note that when configuring hidden layers you must specify a unique name for each layer in the respective dialog.

Exercise 13. Apply the Neural Net

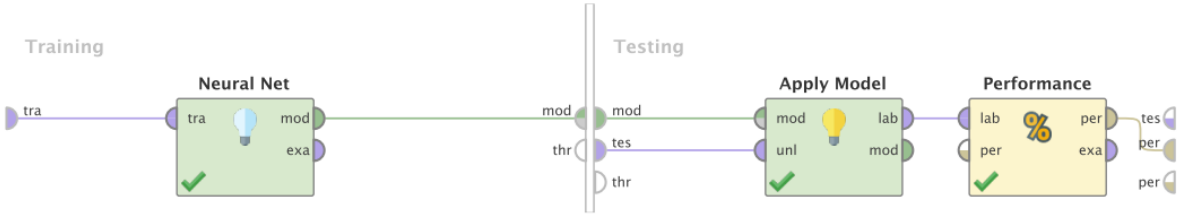
1. Create a new process and store it as neural network.
2. Execute the previous process, prepare data, to load and prepare the data.
3. Add a cross validation to validate the neural net algorithm on the customer churn data. Perform type conversions as necessary.
4. Experiment with the network's parameters to improve the performance.

Answer to Exercise 13

The complete process setup to train and validate Neural Network is shown.



(a) Root process



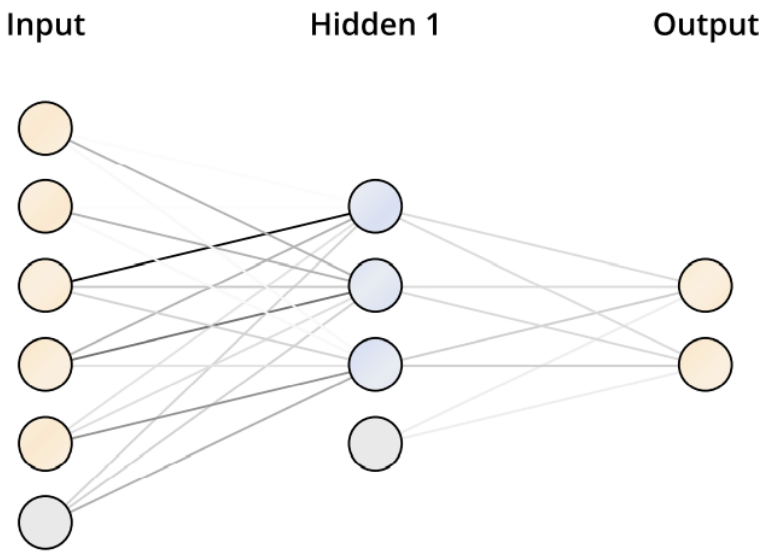
(b) Cross validation

In front of the Cross-Validation you need to use Nominal to Numerical to perform dummy coding on all nominal attributes (because the neural network can only deal with numerical input).

The cross validation is setup as usual, and only contains the Neural Net operator and the Apply Model and Performance operators.

Be sure to connect the mod and ave ports of the Cross Validation operator to the process output to see both the neural net model and its performance.

A simplified version of the model is shown (the actual model is trained with all attributes, but it was too messy to show in a figure):



A neural network with one hidden layer trained on selected attributes of the customer churn data. Note the two output nodes – one represents the churn class and the other the loyal class.

The performance of neural network is shown below:

accuracy: 83.59% +/- 5.67% (mikro: 83.59%)

	true loyal	true churn	class precision
pred. loyal	458	60	88.42%
pred. churn	37	36	49.32%
class recall	92.53%	37.50%	

The model depicts the hidden layer and connections and also suggests the complexity of a model trained with all the attributes. However, as long as the cross validation tells

you that a model works, then it works! Whether it really tells us so is to be discussed – we have dedicated the complete next section to this topic. For the moment, though, let's just observe that off the cuff the accuracy climbed to around 84%.