**Data Pipeline Processing**

pogRomcy danych (Data Masters) Season 1 / Episode 19

All rights reserved.  
The use shall be subject to prior consent of the author.

Press A to see the plain text instead of the slides.  
Press T to display the table of contents.

**What Is This Episode About?**

The data processing usually is composed of many steps. The data are filtered, sorted, grouped, summed up, re-filtered, pasted with the others, etc.

Such a sequence of the operations resembles a pipeline, where the data flow from the source, and are processed at the successive stages.

Relatively recently, the operator has been introduced to the R program, which enables to easily describe such processing pipelines. You can live without it, but once you get to know it and begin to work with it, the life without it loses its taste.

In this Episode, we will learn:

* How to combine the calls of the data processing functions to get the processing pipelines.
* What instructions is the pipeline processing operator equivalent to.

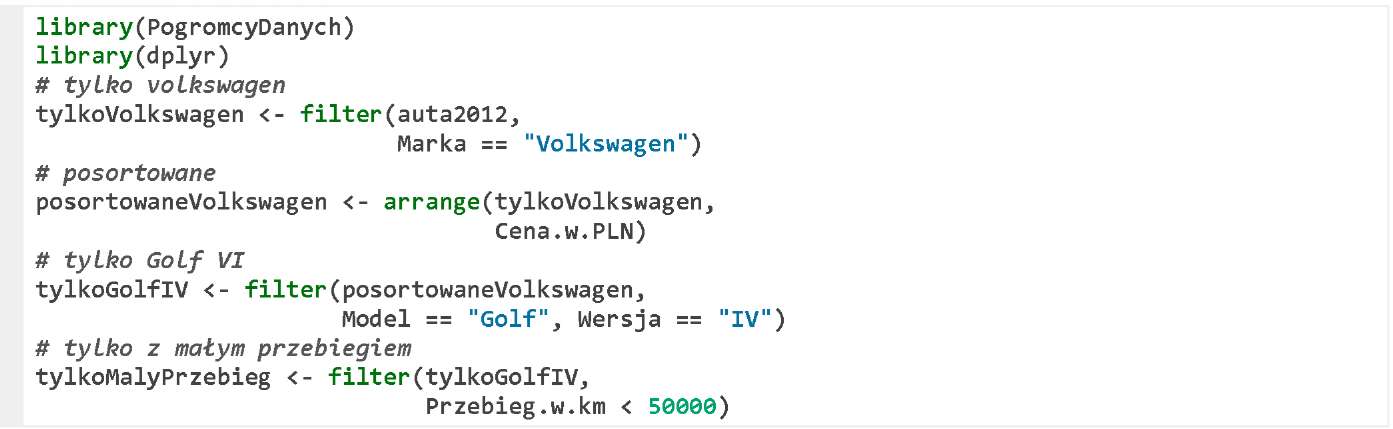
To illustrate these issues, we will use two sets of data auta2012 available in the package PogromcyDanych.

**‘Step by Step’ Processing**

As it has been already mentioned, the data analysis usually consists in data processing up and down, back and forth.

Let’s look at an exemplary operation series. Let’s start with the data on all car sales offers. We will filter out the cars, which are not Volkswagen, the other cars will be arranged by their price. Next, we will leave only Golf IV cars with the mileage of less than 50,000 km.

The below instruction is not the shortest possible. It was purposely extended to better present the advantages of the pipelines.



This sequence of the operations creates four auxiliary sets of data: tylkoVolkswagen, posortowaneVolkswagen, tylkoGolfIV, and tylkoMalyPrzebieg.

If we solely need the last one, then the others clutter the memory, so it would be better to remove them immediately, or not create them at all.

**Multi-Layer Structure Processing**

Since in the R, the results of one function can be transferred to another function, the mentioned operations can also be put together to get a big multi-layer structure.

To get a huge function call in the function, which directly transfers the results of one function to another. Such a large multi-layer structure would be, as follows.



In this solution, no redundant variables are created, but the format itself is very illegible. Even with the indentations, it is difficult to see, which arguments belong to which function.

If the processing is more complex, such block can even grow, and thus it will be more difficult to understand what happens in the code.

**Pipeline Processing**

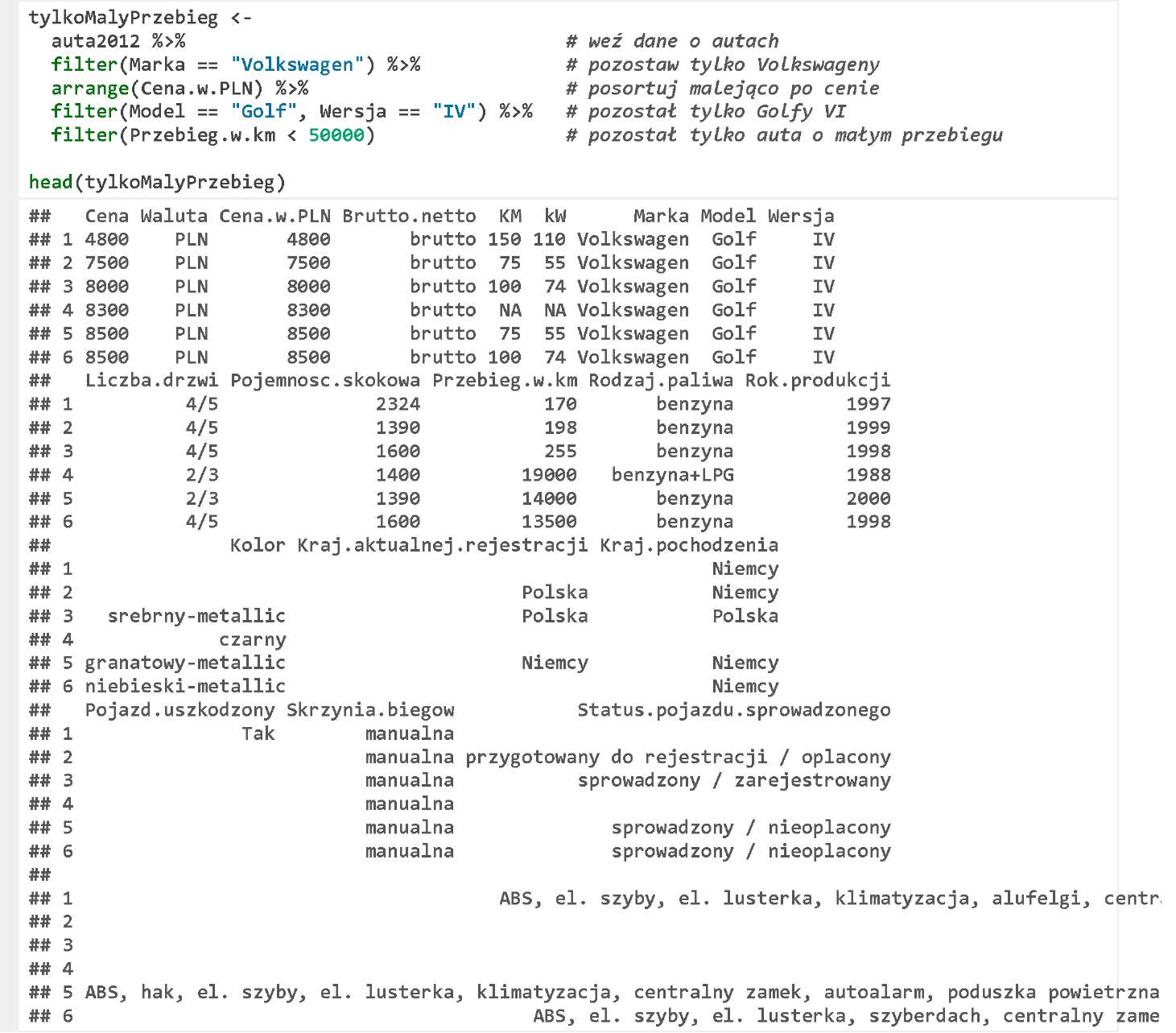
The solution to this problem is the use of a special operator for the pipeline processing %>%. This operator comes with the package magrittr (the quote from its documentation: to be pronounced with a sophisticated French accent), and is available after running the package dplyr (as a dependent package).

How does this operator work?

It moves the left side of the operator to be the first argument of the right side of such operator.

Therefore, the instruction a %>% f(b) is equivalent to the instruction f(a, b).

This simple trick can significantly shorten the notation and make it much clearer.



**How to Read Pipeline Processing**

How to read such a fragment?

It is easier, as the arguments are close to the function names, so we can read the processing description sentence by sentence.

The below example can be read, as follows.

Take the set of data auta2012.

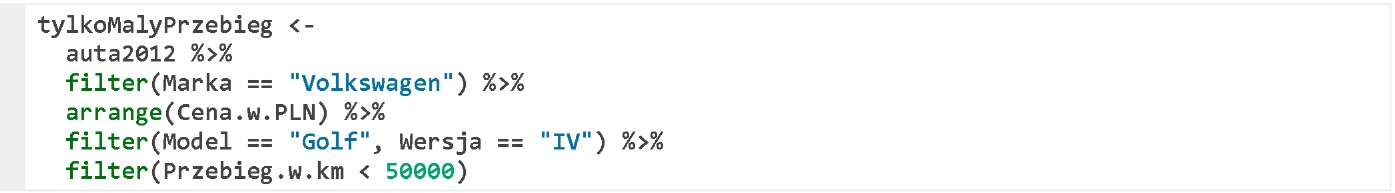
Next, apply the function filter leaving only the following car makes Marka == "Volkswagen".

Then, arrange the cars in the descending order by the variable Cena.w.PLN.

Then, use the function filter leaving only the car makes of Golf, version IV.

Next, apply the function filter leaving only the cars with the mileage below 50,000 km.

The notation with the operator %>% is often much clearer, and we are going to frequently use it in the data processing.



**Everything Flows**

If you prefer the full flow in reading from the pipeline top to the bottom, then instead of the assignment operator < - you can use the operator -> of the same meaning, but different order of the arguments.

In the below code, the processing will be performed, and in the last step, the result will be assigned to the variable tylkoMalyPrzebieg.



**What Can You Do with Pipeline**

The operator %>% is mostly used in working with the functions of the packages dplyr and ggvis, but it can be obviously applied to other functions, too. Eventually, the only thing it does, is transferring its left argument to be the first argument of the right side expression.

For example, to use the function dim() to check the dimensions of the frame tylkoMalyPrzebieg.

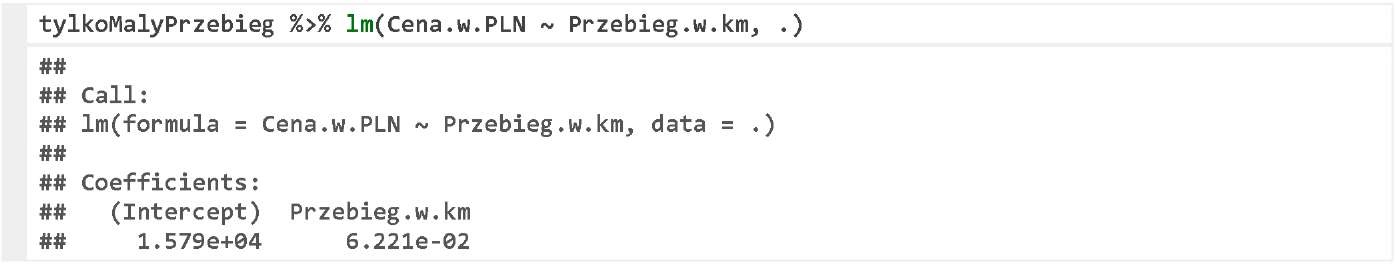
C:\Users\Kasia\AppData\Local\Temp\FineReader10\media\image6.jpeg

To display several first rows of the data frame tylkoMalyPrzebieg.

C:\Users\Kasia\AppData\Local\Temp\FineReader10\media\image7.jpeg

If we would like to move the dataset of the left side of the operator %>% to a place other than the first argument, we also can do it. The position, where the argument is to be placed should be, for this purpose, marked with a dot ..

In the below example, we use the function lm(), specify the formula as the first argument, and the set of data as the second argument. In this example, because the set of data is ., it will be replaced with the set tylkoMalyPrzebieg (left side of the operator %>%).



**Exercises**

* Use the operator %>% to select only ten cheapest Rolls-Royces from the set of data auta 2012.
* Use the operator %>% to select five Volkswagens with the largest engines from the set of data auta2012.

You can find the exemplary answers at https://rawgit.com/pbiecek/MOOC/master/0\_dane/9\_zadania.html.