AVED: Mario strikes back (magrittr)

Štěpán Mikula

Contents

Some code examples

Let's see a code:

```
x <- round(sum(colSums(t(matrix(data = rnorm(50), ncol = 5, nrow = 10))), na.rm = TRUE), digits = 3)
```

. . . and x is of course equal to -3.345.

Hmm, what about writing it nicely:

```
x <- rnorm(50)
x <- matrix(data = x, ncol = 5, nrow = 10)
x <- t(x)
x <- colSums(x)
x <- sum(x, na.rm = TRUE)
x <- round(x, digits = 3)
```

We have a picture now, but it is still quite verbose.

Are you ready for a plumbing job?

```
rnorm(50) %>%
  matrix(data = ., ncol = 5, nrow = 10) %>%
  t() %>%
  colSums() %>%
  sum(na.rm = TRUE) %>%
  round(digits = 3) -> x
```

A concept of the pipe %>%

(Almost) every function has inputs (arguments) and outputs (values). Pipe %>% channels output of a function to input another function.

```
rnorm(50) %>%
  matrix(data = ., ncol = 5, nrow = 10) %>%
  t() %>%
  colSums() %>%
  sum(na.rm = TRUE) %>%
  round(digits = 3) -> x
```

Figure 1:

By default an object runnig through a pipe is fed to the **first argument** of the right-hand side function. If you want to fed another argument (or you want to be explicit) you need to use "placehoder" `.` to determine the destination of the pipe. You can use `.` more then once:

```
multiply <- function(x = 1, y = -1) x*y

multiply()
```

```
## [1] -1
```

```
5 %>% multiply()
```

```
## [1] -5
```

```
5 %>% multiply(y = .)
```

```
## [1] 5
```

```
5 %>% multiply(x = ., y = .)
```

```
## [1] 25
```

Pipe is not implemented in base R. It is part of packages `dplyr` and `magrittr`. `dplyr` contains only basic pipe (`%>%`). `magrittr` allows user to use more advanced options.

> **Where have I seen it before?** Of course you know the pipe from (a) Sherlock Holmes, and (b) your Unix/Linux shell (`|`). *(Sherlock Holmes is a fictional detective with a funny hat.)*

Pipes and aliases of `magrittr`

```
library(magrittr)
```

`magrittr` provides extra three pipes (their descriptions are from vignette). We will illustrate their function using famous `iris` dataset:

```
# Print few lines
iris %>% head
```

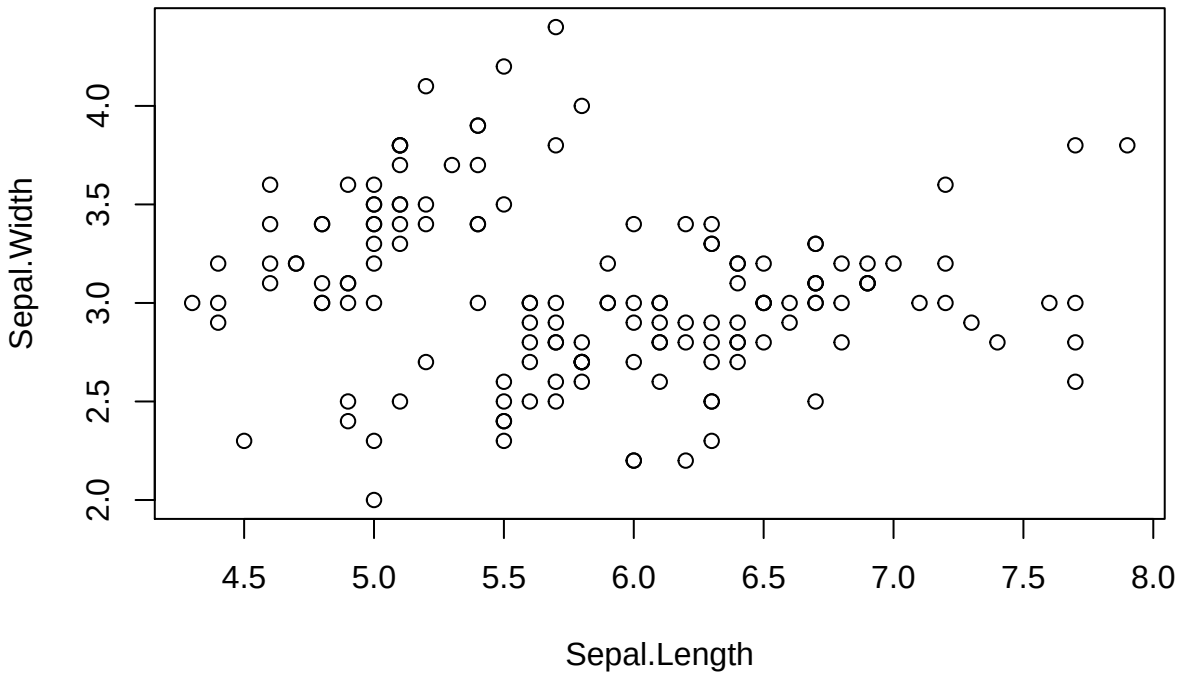
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

The "tee" operator, `%T>%` works like `%>%`, except it returns the left-hand side value, and not the result of the right-hand side operation. This is useful when a step in a pipeline is used for its side-effect (printing, plotting, logging, etc.).

```r
iris[,1:2] %>% plot -> v1
print(v1)
```

```
## NULL
```

```r
iris[,1:2] %T>% plot -> v2
```




```r
print(head(v2))
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5.0         3.6
## 6          5.4         3.9
```

The "exposition" pipe operator, `%$%` exposes the names within the left-hand side object to the right-hand side expression.

Let's sum up `Sepal.Length` column:

```r
iris %>% sum(Sepal.Length)
```

In this case it ends up with an error. (You can try it.)

```r
iris %$% sum(Sepal.Length)
```

```
## [1] 876.5
```

It worked this time.

Compound assignment pipe operator %<>% *(aka Moebius pipe)* can be used as the first pipe in a chain. The effect will be that the result of the pipeline is assigned to the left-hand side object, rather than returning the result as usual.

```
x %>% sqrt -> x
```

is equivalent to

```
x %<>% sqrt
```

```
## Warning in sqrt(.): NaNs produced
```

> Notice that it is not necessary to write functions with `()`. `x %<>% sqrt()` is the same as `x %<>% sqrt`.

Aliases

When chaining functions you might need to use some pretty basic operations. It is impossible with bare `+`, `-`, or `/`. One need a series of functions similar to user-defined function `multiply()` defined above. `magrittr` provides a lot of such functions:

Alias (magrittr)	Base R
extract	`[`
extract2	`[[`
inset	`[<-`
inset2	`[[<-`
use_series	`$`
add	`+`
subtract	`-`
multiply_by	`*`
raise_to_power	`^`
multiply_by_matrix	`%*%`
divide_by	`/`
divide_by_int	`%/%`
mod	`%%`
is_in	`%in%`
and	`&`
or	`|`
equals	`==`
is_greater_than	`>`
is_weakly_greater_than	`>=`
is_less_than	`<`
is_weakly_less_than	`<=`
not	`!`
set_colnames	`colnames<-`
set_rownames	`rownames<-`
set_names	`names<-`

See a complex example:

```
iris[1:10,] %T>% print %$%
  raise_to_power(Sepal.Width,2) %T>% print %>%
  multiply_by(-1) %T>% print %>%
  is_weakly_greater_than(0) %T>% print %>%
  sum -> x
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
##  [1] 12.25  9.00 10.24  9.61 12.96 15.21 11.56 11.56  8.41  9.61
##  [1] -12.25  -9.00 -10.24  -9.61 -12.96 -15.21 -11.56 -11.56  -8.41  -9.61
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Tee pipe returns a mirror image of the content flowing in pipes. It is printed on the screen using `print`. The content itself remains intact.

In the first phase the whole `data.frame` is flowing in the tube but `%$%` allows `raise_to_power` to use specific (named) part of the `iris` table.

And naturally `x` is equal to 0.

Figure 2: