



# Программирование в среде R

Шевцов Василий Викторович,  
директор ДИТ РУДН, [shevtsov\\_vv@rudn.university](mailto:shevtsov_vv@rudn.university)

# Пакет magrittr и организация конвейеров операторами %...%

# Операторы

## **base::**

% \* % - матричное умножение,  
% / % - целочисленное деление,  
%in% - является ли компонентом

## **expm::**%^%

## **operators::**

%!~\*% (pattern)  
%!~+% (pattern)  
%!~% (pattern)  
%!in% (notIn)  
%\*>% (files)  
%\*>% (files)  
%+=% (plusEqual)  
%~% (patternSubstitution)  
%/~% (patternDivision)  
%2>% (files)  
%2>% (files)  
%<\*% (files)

%<2% (files)  
%<<\*% (files)  
%<<2% (files)  
%<<% (files)  
%<% (files)  
%>>% (files)  
%>% (files)  
%~\*% (pattern)  
%~+% (pattern)  
%~% (pattern)  
%but% (but)  
%s~% (patternSubstitution)  
%x=% (character decorator)  
%o\textasciitilde%  
%of%  
%without%

# Операторы

%...% определяет потоковый оператор, являющийся альтернативной записью функции `x %any_string% f`, а не `f(x)`.

Например, эта функция возвращает строку, состоящую из ее левого аргумента, за которым следуют запятая и пробел, а затем-правый аргумент.

```
"%,%" <- function(x, y) paste0(x, ", ", y)
"Hello" %,"% "World"
[1] "Hello, World"
```

```
"%qwerty%" <- function(x, y) {x*y}
2%qwerty%3
[1] 6
```

## magrittr::%>%

Оператор %>% семантически изменяет ваш код таким образом, что делает его более интуитивным как для чтения, так и для записи.

Рассмотрим модификацию набора данных mtcars

```
library(magrittr)
```

```
car_data <-  
  mtcars %>%  
  subset(hp > 100) %>%  
  aggregate(. ~ cyl, data = ., FUN = . %>% mean %>% round(2)) %>%  
  transform(kpl = mpg %>% multiply_by(0.4251)) %>%  
  print
```

	cyl	mpg	disp	hp	drat	wt	qsec	vs	am	gear	carb	kpl
1	4	25.90	108.05	111.00	3.94	2.15	17.75	1.00	1.00	4.50	2.00	11.010090
2	6	19.74	183.31	122.29	3.59	3.12	17.98	0.57	0.43	3.86	3.43	8.391474
3	8	15.10	353.10	209.21	3.23	4.00	16.77	0.00	0.14	3.29	3.50	6.419010

## Описание конвейера

```
car_data <-  
  mtcars %>%  
  subset(hp > 100) %>%  
  aggregate(. ~ cyl, data = ., FUN = . %>% mean %>% round(2)) %>%  
  transform(kpl = mpg %>% multiply_by(0.4251)) %>%  
  print
```

Мы начинаем со значения `mtcars` (dataframe). Исходя из этого, мы сначала извлекаем подмножество, затем агрегируем информацию на основе количества цилиндров, а затем преобразуем набор данных, добавляя переменную для километров на литр в дополнение к милям на галлон. Наконец, мы печатаем результат.

Обратите внимание, как код расположен в логическом порядке того, как вы думаете о задаче: `data` - `>transform` - `>aggregate`, который также является тем же порядком, в котором будет выполняться код.

## Два вида записи

```
car_data <-  
  mtcars %>%  
  subset(hp > 100) %>%  
  aggregate(. ~ cyl, data = ., FUN = . %>% mean %>% round(2)) %>%  
  transform(kpl = mpg %>% multiply_by(0.4251)) %>%  
  print
```

```
car_data <-  
  transform(aggregate(. ~ cyl,  
    data = subset(mtcars, hp > 100),  
    FUN = function(x) round(mean(x, 2))),  
    kpl = mpg*0.4251)
```

# Разбор конвейера

Необходимо добавить еще один шаг где-то в этом процессе. Это очень легко сделать в конвейерной версии, но немного сложнее в “стандартном ” примере.

Комментарии к примеру конвейера:

- По умолчанию левая сторона (LHS) будет передаваться по конвейеру в качестве первого аргумента функции, появляющейся на правой стороне (RHS). Это видно в выражениях `subset` и `transform` .
- `%>%` может использоваться вложенным образом, например, он может появляться в выражениях внутри аргументов. Это используется в преобразовании `mpg` в `kpl`.
- Когда LHS требуется в позиции, отличной от первой, можно использовать точку `'.'` в качестве заполнителя. Это используется в агрегатном выражении.
- Точку в Формуле не путать с заполнителем, который используется в агрегатном выражении.
- Всякий раз, когда требуется только один аргумент LHS, тогда можно опустить пустые скобки. Это используется в вызове `print` (который также возвращает свой аргумент). Здесь LHS `%>% print()` или LHS `%>% print(.)` тоже будет работать.
- Конвейер с точкой `(.)` так как LHS создаст унарную функцию. Это используется для определения функции агрегатора.



# Конвейер

Конвейеризация в анонимные функции, или лямбды:

```
car_data %>%  
(function(x) {  
  if (nrow(x) > 0)  
    rbind(head(x, 1), tail(x, 1))  
  else x  
})
```

```
car_data %>%  
{  
  if (nrow(.) > 0)  
    rbind(head(., 1), tail(., 1))  
  else .  
}
```

## Оператор % T %

Оператор "тройник", %T>% работает как %>%, за исключением того, что он возвращает значение левой стороны, а не результат операции правой стороны. Это полезно, когда шаг в конвейере используется для его побочного эффекта (печать, построение графиков, ведение журнала и т. д.). Пример (где фактический сюжет здесь опущен):

```
rnorm(200) %>%  
  matrix(ncol = 2) %T>%  
  plot %>% # plot usually does not return anything.  
  colSums
```

## Оператор % \$ %

Оператор конвейера "exposition" % \$ % предоставляет имена внутри объекта левой стороны правому выражению.

По сути, это короткая запись функций with.

Этот оператор удобен, когда функции сами по себе не имеют аргумента данных, как, например, lm и aggregate. Вот несколько примеров в качестве иллюстрации:

```
iris %>%  
  subset(Sepal.Length > mean(Sepal.Length)) %$%  
  cor(Sepal.Length, Sepal.Width)
```

```
data.frame(z = rnorm(100)) %$%  
  ts.plot(z)
```

## Оператор составного конвейера %<>%

Наконец, оператор составного конвейера %<>% может быть использован в качестве первой операции конвейера в цепочке. В результате результат конвейера будет назначен левому объекту, а не возвращать результат, как обычно. Это, по сути, сокращенное обозначение для таких выражений, как

```
foo <- foo %>% bar %>% baz,
```

которое сводится к

```
foo %<>% bar %>% baz.
```

Вот еще один пример

```
iris$Sepal.Length %<>% sqrt
```

# Псевдонимы

В дополнение к оператору `%>%`, `magrittr` предоставляет некоторые псевдонимы для других операторов, которые делают такие операции, как сложение или умножение, хорошо вписывающимися в синтаксис `magrittr`. В качестве примера рассмотрим:

```
rnorm(1000) %>%  
multiply_by(5) %>%  
add(5) %>%  
{  
  cat("Mean:", mean(.), "Variance:", var(.), "\n")  
  head(.)  
}
```

```
Mean: 5.168987 Variance: 26.60159  
[1] 8.0770368 6.4072487 4.1694138 0.3745420 -2.1506064 -0.6489339
```

компактная форма:

```
rnorm(100) %>% `*`(5) %>% `+`(5) %>%  
{  
  cat("Mean:", mean(.), "Variance:", var(.), "\n")  
  head(.)  
}
```

# Псевдонимы

magrittr предоставляет ряд псевдонимов, которые могут быть более приятными для использования при составлении цепочек с использованием оператора `%>%`.

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 (`n'est pas`)	`!`
set_colnames	`colnames<-`
set_rownames	`rownames<-`
set_names	`names<-`

# Пакет dplyr

В dplyr есть пять команд, предназначенных для выполнения большинства операций по работе с данными. Select — для выбора одного или более столбцов. Filter — для выбора строк на основании каких-либо критериев. Arrange — для сортировки данных по одному или нескольким столбцам по возрастанию или убыванию. Mutate — для добавления к данным новых столбцов. Summarise — для вычислений.

```
mtcars <- mtcars %>%  
  df() %>%  
  select(carb, mpg, wt)
```

```
# без группировки  
mtcars %>%  
  summarise (mean_mpg = mean(mpg))
```

```
# с группировкой  
mtcars %>%  
  group_by(carb) %>%  
  summarise (mean_mpg = mean(mpg))
```

# Спасибо за внимание!



Шевцов Василий Викторович

shevtsov\_vv@rudn.university  
+7(903)144-53-57