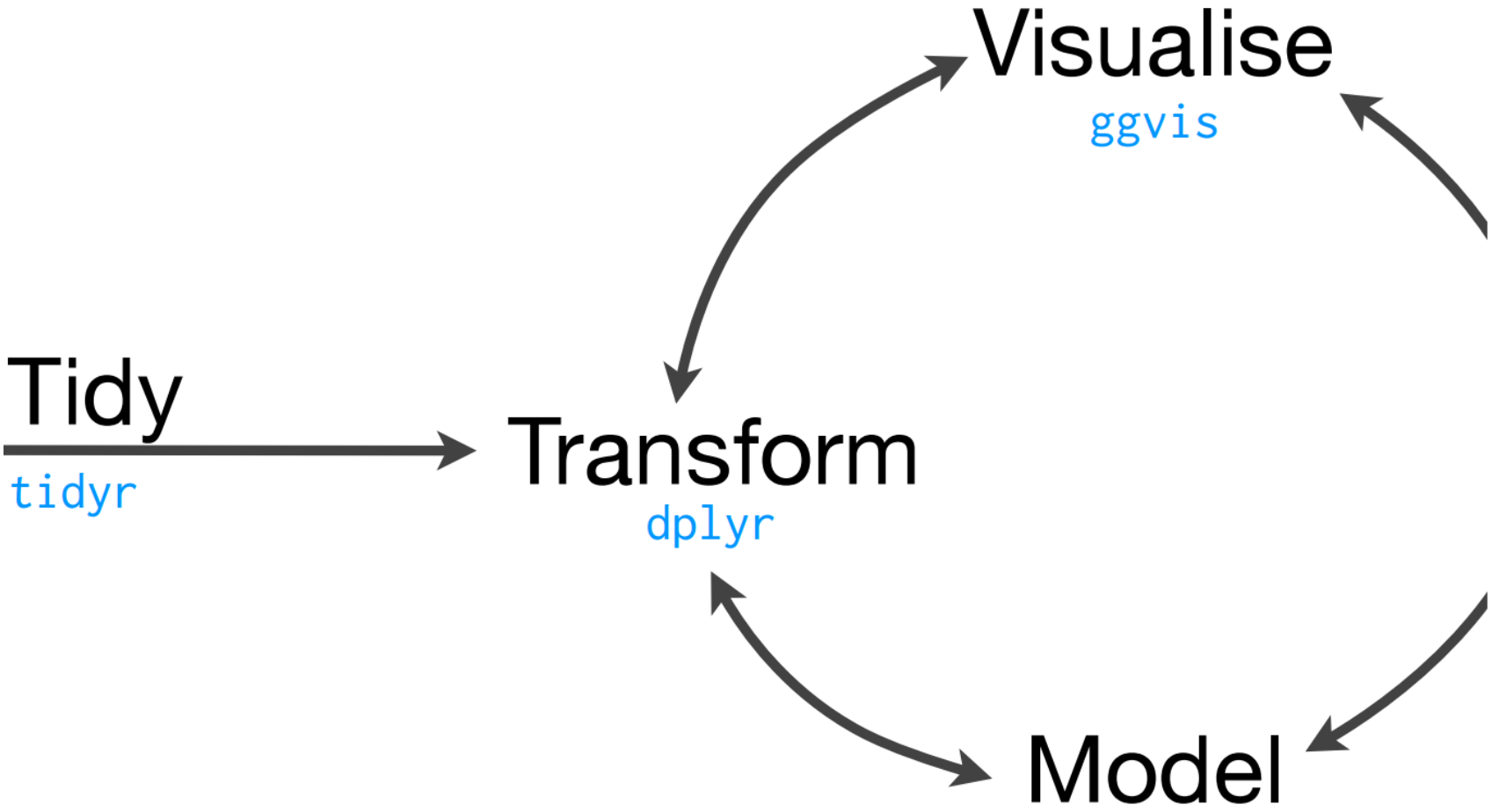
Современный взгляд на предобработку данных: dplyr, tidyr и magrittr

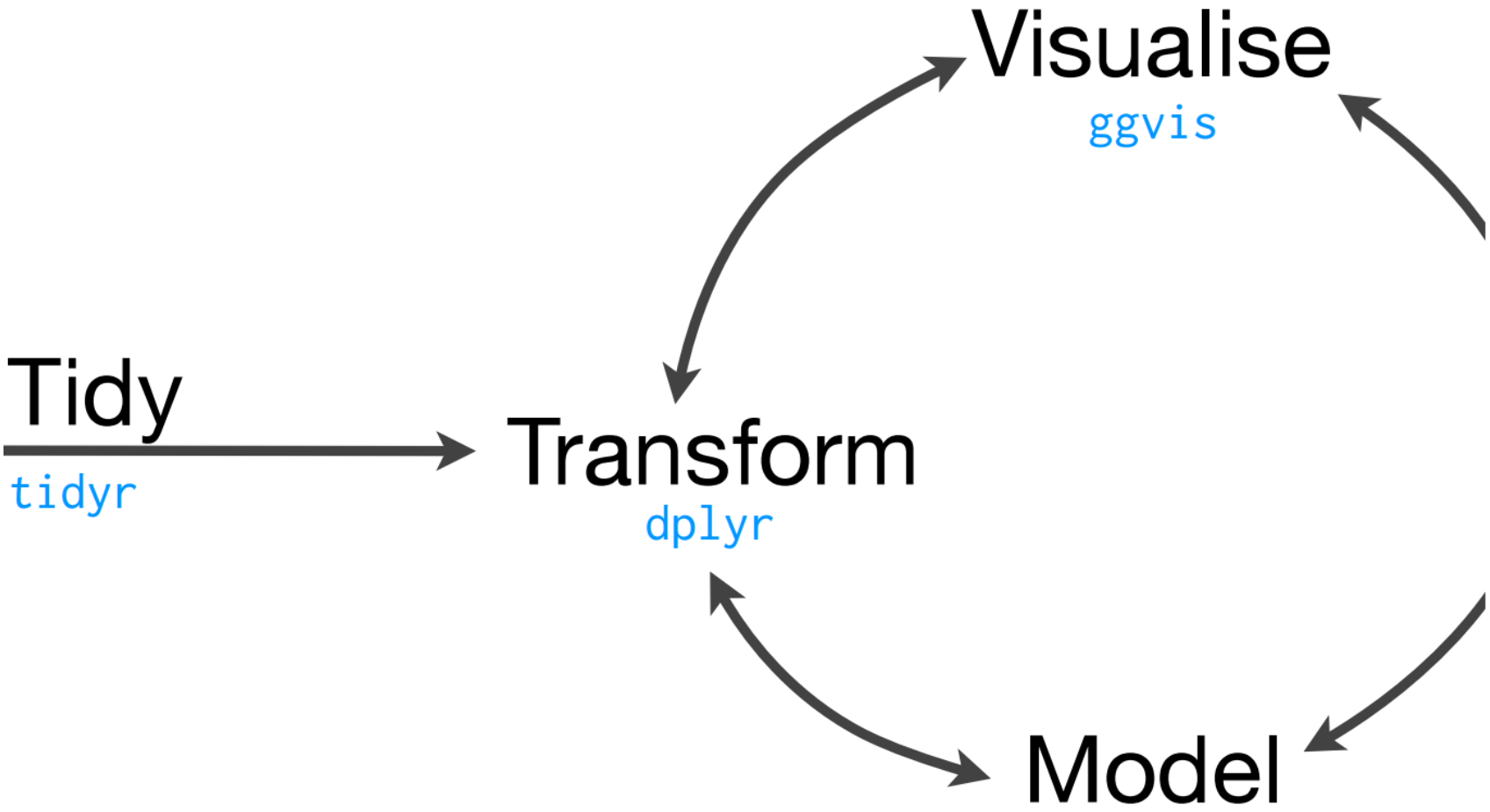
Антон Антонов

28 марта 2015

Цикл жизни данных в R



Цикл жизни данных в R



- tidyr 0.2 (Hadley Wickham, RStudio)
- dplyr 0.4 (Hadley Wickham, Roman Francois, RStudio)
- magrittr 1.5 (Stefan Milton Bache, Hadley Wickham)

magrittr::%>%

При необходимости последовательно вызвать несколько функций подряд есть следующие возможности.

magrittr::%>%

При необходимости последовательно вызвать несколько функций подряд есть следующие возможности.

- Вложенные вызовы (nested calls):

```
arrange(
  summarize(
    filter(data, variable == 42),
    Total = sum(variable)
  ),
  desc(Total)
)
```

magrittr::%>%

При необходимости последовательно вызвать несколько функций подряд есть следующие возможности.

- Вложенные вызовы (nested calls):

```
arrange(
  summarize(
    filter(data, variable == 42),
    Total = sum(variable)
  ),
  desc(Total)
)
```

- Создание промежуточных переменных:

```
a <- filter(data, variable == 42)
b <- summarise(a, Total = sum(variable))
c <- arrange(b, desc(Total))
```

magrittr::%>%

При необходимости последовательно вызвать несколько функций подряд есть следующие возможности.

- Вложенные вызовы (nested calls):

```
arrange(
  summarize(
    filter(data, variable == 42),
    Total = sum(variable)
  ),
  desc(Total)
)
```

- Создание промежуточных переменных:

```
a <- filter(data, variable == 42)
b <- summarise(a, Total = sum(variable))
c <- arrange(b, desc(Total))
```

- Конвейеры (pipes) при помощи оператора %>%:

```
data %>%
  filter(variable == 42) %>%
  summarise(Total = sum(variable)) %>%
  arrange(desc(Total))
```

magrittr::%>%

Следующий кусок кода содержит вложенные вызовы, формулу с точкой и анонимную функцию.

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

```
##   cyl     hp       kpl
## 1   4 111.00 11.010090
## 2   6 122.29  8.391474
## 3   8 209.21  6.419010
```

magrittr::%>%

Используем оператор, работающий по двум основным правилам:

```
x %>% f(y)             # same as f(x, y)
x %>% g(y, param = .) # same as g(y, param = x)
```

magrittr::%>%

Используем оператор, работающий по двум основным правилам:

```
x %>% f(y)             # same as f(x, y)
x %>% g(y, param = .) # same as g(y, param = x)
```

Тот же код в виде конвейера:

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

##   cyl     hp       kpl
## 1   4 111.00 11.010090
## 2   6 122.29  8.391474
## 3   8 209.21  6.419010
```

magrittr::%>%

Конвейеры можно (но не нужно) использовать практически везде:

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

```
##   cyl      hp       kpl
## 1   4 111.00 11.010090
## 2   6 122.29  8.391474
## 3   8 209.21  6.419010
```

Tidy data

Концепция “tidy data”:

- каждая колонка (столбец) – переменная;
- каждый ряд (строка) – наблюдение.

Tidy data

Концепция “tidy data”:

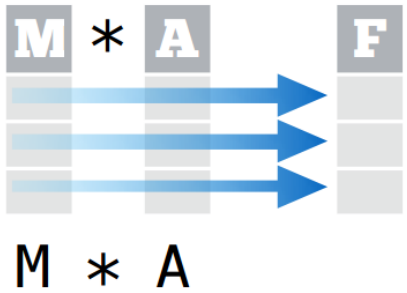
- каждая колонка (столбец) – переменная;
- каждый ряд (строка) – наблюдение.

Tidy data

Концепция “tidy data”:

- каждая колонка (столбец) – переменная;
- каждый ряд (строка) – наблюдение.

Для R такой формат данных наиболее естественен из-за векторизованности многих базовых операций.



Возьмём результат некоторого воображаемого эксперимента:

```
messy

##   id     group    work.T1   home.T1   work.T2    home.T2
## 1  1 treatment 0.08513597 0.6158293 0.1135090 0.05190332
## 2  2   control 0.22543662 0.4296715 0.5959253 0.26417767
## 3  3 treatment 0.27453052 0.6516557 0.3580500 0.39879073
## 4  4   control 0.27230507 0.5677378 0.4288094 0.83613414
```

Возьмём результат некоторого воображаемого эксперимента:

```
messy

##   id     group    work.T1   home.T1   work.T2    home.T2
## 1  1 treatment 0.08513597 0.6158293 0.1135090 0.05190332
## 2  2   control 0.22543662 0.4296715 0.5959253 0.26417767
## 3  3 treatment 0.27453052 0.6516557 0.3580500 0.39879073
## 4  4   control 0.27230507 0.5677378 0.4288094 0.83613414
```

Почему это не “tidy data”?

Возьмём результат некоторого воображаемого эксперимента:

```
messy

##   id     group    work.T1   home.T1   work.T2    home.T2
## 1  1 treatment 0.08513597 0.6158293 0.1135090 0.05190332
## 2  2   control 0.22543662 0.4296715 0.5959253 0.26417767
## 3  3 treatment 0.27453052 0.6516557 0.3580500 0.39879073
## 4  4   control 0.27230507 0.5677378 0.4288094 0.83613414
```

Почему это не “tidy data”?

- Переменные места и времени смешаны;
- Одной строке соответствует четыре наблюдения.

tidyr::gather

Преобразуем данные из т.н. “широкого” формата в “длинный” (wide to long):

```
tidier <- messy %>%
  gather(key, bpr, -id, -group)
tidier %>% head(8)
```

```
##   id     group     key       bpr
## 1  1 treatment work.T1 0.08513597
## 2  2   control work.T1 0.22543662
## 3  3 treatment work.T1 0.27453052
## 4  4   control work.T1 0.27230507
## 5  1 treatment home.T1 0.61582931
## 6  2   control home.T1 0.42967153
## 7  3 treatment home.T1 0.65165567
## 8  4   control home.T1 0.56773775
```

tidyr::spread

Обратное преобразование (long to wide):

```
tidier %>%
  spread(key, bpr) %>%
  print %>%
  all.equal(messy)

##   id     group    work.T1   home.T1   work.T2    home.T2
## 1  1 treatment 0.08513597 0.6158293 0.1135090 0.05190332
## 2  2   control 0.22543662 0.4296715 0.5959253 0.26417767
## 3  3 treatment 0.27453052 0.6516557 0.3580500 0.39879073
## 4  4   control 0.27230507 0.5677378 0.4288094 0.83613414
## [1] TRUE
```

tidyr::separate

Разделяем переменные:

```
tidy <- tidier %>%
  separate(key, into = c("location", "time"), sep = "\\.")
tidy %>% head(8)

##   id     group location time        bpr
## 1  1 treatment     work   T1 0.08513597
## 2  2   control     work   T1 0.22543662
## 3  3 treatment     work   T1 0.27453052
## 4  4   control     work   T1 0.27230507
## 5  1 treatment     home   T1 0.61582931
## 6  2   control     home   T1 0.42967153
## 7  3 treatment     home   T1 0.65165567
## 8  4   control     home   T1 0.56773775
```

Склеиваем переменные обратно:

```
tidy %>%
  unite(key, location, time, sep = ".") %>%
  head(8)

##   id     group     key        bpr
## 1  1 treatment work.T1 0.08513597
## 2  2   control work.T1 0.22543662
## 3  3 treatment work.T1 0.27453052
## 4  4   control work.T1 0.27230507
## 5  1 treatment home.T1 0.61582931
## 6  2   control home.T1 0.42967153
## 7  3 treatment home.T1 0.65165567
## 8  4   control home.T1 0.56773775
```

dplyr::select

Выбираем подмножество столбцов (переменных):

```
tidy %>%
  select(id, group, location, bpr) %>%
  head(8)

##   id     group location        bpr
## 1  1 treatment     work 0.08513597
## 2  2   control     work 0.22543662
## 3  3 treatment     work 0.27453052
## 4  4   control     work 0.27230507
## 5  1 treatment     home 0.61582931
## 6  2   control     home 0.42967153
## 7  3 treatment     home 0.65165567
## 8  4   control     home 0.56773775
```

dplyr::select

Выбираем подмножество столбцов (переменных):

```
tidy %>%
  select(id, group, location, bpr) %>%
  head(8)

##   id     group location        bpr
## 1  1 treatment     work 0.08513597
## 2  2   control     work 0.22543662
## 3  3 treatment     work 0.27453052
## 4  4   control     work 0.27230507
## 5  1 treatment     home 0.61582931
## 6  2   control     home 0.42967153
## 7  3 treatment     home 0.65165567
## 8  4   control     home 0.56773775
```

```
tidy %>% select(1:3, 5)
tidy %>% select(id:location, starts_with("b"))
tidy %>% select(-time)
tidy %>% select(-contains("m"))
tidy %>% select(-matches("t.m"))
```

dplyr::filter

Выбираем подмножество строк (наблюдений):

```
tidy %>%
  filter(group == "control", bpr > 0.5, id %in% 4:5)

##   id   group location time       bpr
## 1  4 control     home   T1 0.5677378
## 2  4 control     home   T2 0.8361341
```

dplyr::group_by, dplyr::summarize

Агрегация по группам:

```
tidy %>%
  group_by(time, location) %>%
  summarize(mean_t = mean(bpr), sd_t = sd(bpr))

## Source: local data frame [4 x 4]
## Groups: time
##
##   time location    mean_t       sd_t
## 1   T1     home 0.5662236 0.09731072
## 2   T1     work 0.2143520 0.08906863
## 3   T2     home 0.3877515 0.33127700
## 4   T2     work 0.3740734 0.20030382
```

dplyr::arrange

Сортировка (например, в разных направлениях по разным переменным):

```
tidy %>%
  arrange(time, desc(bpr))
```

```
##    id     group location time        bpr
## 1   3 treatment     home   T1 0.65165567
## 2   1 treatment     home   T1 0.61582931
## 3   4   control     home   T1 0.56773775
## 4   2   control     home   T1 0.42967153
## 5   3 treatment     work   T1 0.27453052
## 6   4   control     work   T1 0.27230507
## 7   2   control     work   T1 0.22543662
## 8   1 treatment     work   T1 0.08513597
## 9   4   control     home   T2 0.83613414
## 10  2   control     work   T2 0.59592531
## 11  4   control     work   T2 0.42880942
## 12  3 treatment     home   T2 0.39879073
## 13  3 treatment     work   T2 0.35804998
## 14  2   control     home   T2 0.26417767
## 15  1 treatment     work   T2 0.11350898
## 16  1 treatment     home   T2 0.05190332
```

dplyr::mutate

Создание новых переменных:

```
tidy %>%
  mutate(perc = bpr/sum(bpr),
         cperc = cumsum(perc)) %>%
  select(id, bpr:cperc)
```

```
##    id        bpr        perc       cperc
## 1   1 0.08513597 0.013799264 0.01379926
## 2   2 0.22543662 0.036539896 0.05033916
## 3   3 0.27453052 0.044497283 0.09483644
## 4   4 0.27230507 0.044136570 0.13897301
## 5   1 0.61582931 0.099816700 0.23878971
## 6   2 0.42967153 0.069643314 0.30843303
## 7   3 0.65165567 0.105623616 0.41405664
## 8   4 0.56773775 0.092021780 0.50607842
## 9   1 0.11350898 0.018398105 0.52447653
## 10  2 0.59592531 0.096590559 0.62106709
## 11  3 0.35804998 0.058034534 0.67910162
## 12  4 0.42880942 0.069503579 0.74860520
## 13  1 0.05190332 0.008412751 0.75701795
## 14  2 0.26417767 0.042819240 0.79983719
## 15  3 0.39879073 0.064638000 0.86447519
## 16  4 0.83613414 0.135524811 1.00000000
```

SQL joins

Для иллюстрации join рассмотрим два набора данных:

```
df1

##              Name Instrument
## 1   Roger Waters       Bass
## 2  David Gilmour     Guitar
## 3    Syd Barrett       Bass
## 4 Richard Wright  Keyboards
## 5     Nick Mason      Drums

df2

##   Instrument StrCount
## 1       Bass        4
## 2     Guitar        6
## 3     Violin        4
```

SQL joins

Всем известные joins:

```
left_join(df1, df2, "Instrument")
```

```
##             Name Instrument StrCount
## 1   Roger Waters       Bass        4
## 2  David Gilmour     Guitar        6
## 3    Syd Barrett       Bass        4
## 4 Richard Wright  Keyboards       NA
## 5     Nick Mason      Drums       NA
```

```
full_join(df1, df2, "Instrument")
```

```
##             Name Instrument StrCount
## 1   Roger Waters       Bass        4
## 2  David Gilmour     Guitar        6
## 3    Syd Barrett       Bass        4
## 4 Richard Wright  Keyboards       NA
## 5     Nick Mason      Drums       NA
## 6           <NA>     Violin        4
```

```
#right_join(df1, df2, "Instrument")
#inner_join(df1, df2, "Instrument")
```

SQL joins

Не настолько известные, но тоже joins:

```
semi_join(df1, df2, "Instrument")

##             Name Instrument
## 1  Roger Waters       Bass
## 2   Syd Barrett       Bass
## 3 David Gilmour     Guitar

anti_join(df1, df2, "Instrument")

##              Name Instrument
## 1     Nick Mason       Drums
## 2 Richard Wright  Keyboards
```

rbenchmarking

Небольшая демонстрация быстродействия dplyr (трудоёмкие операции написаны на C++).

```
library(rbenchmark)
set.seed(11)
n <- 10e6
m <- 100
d <- data_frame(x = sample(m, n, replace=TRUE), y = runif(n))
dm <- data_frame(x = sample(m))
l <- vector("list", 5)
```

rbenchmarking

```
benchmark(
  d[d$x>=10 & d$x<20,],
  d %>% filter(x>=10, x<20),
  replications=5,
  columns=c("test", "elapsed", "relative"),
  order=NULL
) %>% print -> l[[1]]

##                              test elapsed relative
## 1     d[d$x >= 10 & d$x < 20, ]    3.89    2.016
## 2 d %>% filter(x >= 10, x < 20)    1.93    1.000
```

rbenchmarking

```
benchmark(
  d[order(d$x), ],
  d %>% arrange(x),
  replications=5,
  columns=c("test", "elapsed", "relative"),
  order=NULL
) %>% print -> l[[2]]

##                test elapsed relative
## 1  d[order(d$x), ]   29.44     1.23
## 2 d %>% arrange(x)   23.94     1.00
```

rbenchmarking

```
benchmark(
  d$y2 <- d$y + d$x/2,
  d %>% mutate(y2 = y + x/2),
  replications=5,
  columns=c("test", "elapsed", "relative"),
  order=NULL
) %>% print -> l[[3]]
```

```
##                          test elapsed relative
## 1         d$y2 <- d$y + d$x/2    0.25     1.00
## 2 d %>% mutate(y2 = y + x/2)    0.28     1.12
```

rbenchmarking

```
benchmark(
  tapply(d$y, d$x, mean),
  d %>% group_by(x) %>% summarize(ym = mean(y)),
  replications=5,
  columns=c("test", "elapsed", "relative"),
  order=NULL
) %>% print -> l[[4]]
```

```
##                                                 test elapsed
## 1                          tapply(d$y, d$x, mean)    4.20
## 2 d %>% group_by(x) %>% summarize(ym = mean(y))    2.66
##   relative
## 1    1.579
## 2    1.000
```

rbenchmarking

```
benchmark(
  merge(d, dm, by="x"),
  d %>% inner_join(dm, by="x"),
  replications=1,
  columns=c("test", "elapsed", "relative"),
  order=NULL
) %>% print -> l[[5]]

##                                test elapsed relative
## 1          merge(d, dm, by = "x")   40.33  118.618
## 2 d %>% inner_join(dm, by = "x")    0.34    1.000
```

Сводная таблица быстродействия
(относительное время исполнения, меньше – лучше):

	Filter	Sort	New Variable	Aggregation	Join
base	2.016	1.23	1	1.579	118.618
dplyr	1	1	1.12	1	1

Сводная таблица быстродействия
(относительное время исполнения, меньше – лучше):

	Filter	Sort	New Variable	Aggregation	Join
base	2.016	1.23	1	1.579	118.618
dplyr	1	1	1.12	1	1

Интересный факт: в Rstudio оператор %>% имеет своё сочетание горячих клавиш (Ctrl + Shift + M)!

magrittr::%<>%

С помощью конвейеров можно создавать не только анонимные функции:

```
0 %>% cos %>% sin

## [1] 0.841471

f <- . %>% cos %>% sin
f(0)

## [1] 0.841471

sin(cos(0))

## [1] 0.841471
```

magrittr::%<>%

С помощью конвейеров можно создавать не только анонимные функции:

```
0 %>% cos %>% sin

## [1] 0.841471

f <- . %>% cos %>% sin
f(0)

## [1] 0.841471

sin(cos(0))

## [1] 0.841471
```

Есть ещё несколько конвейерных операторов, в том числе присваивающий...

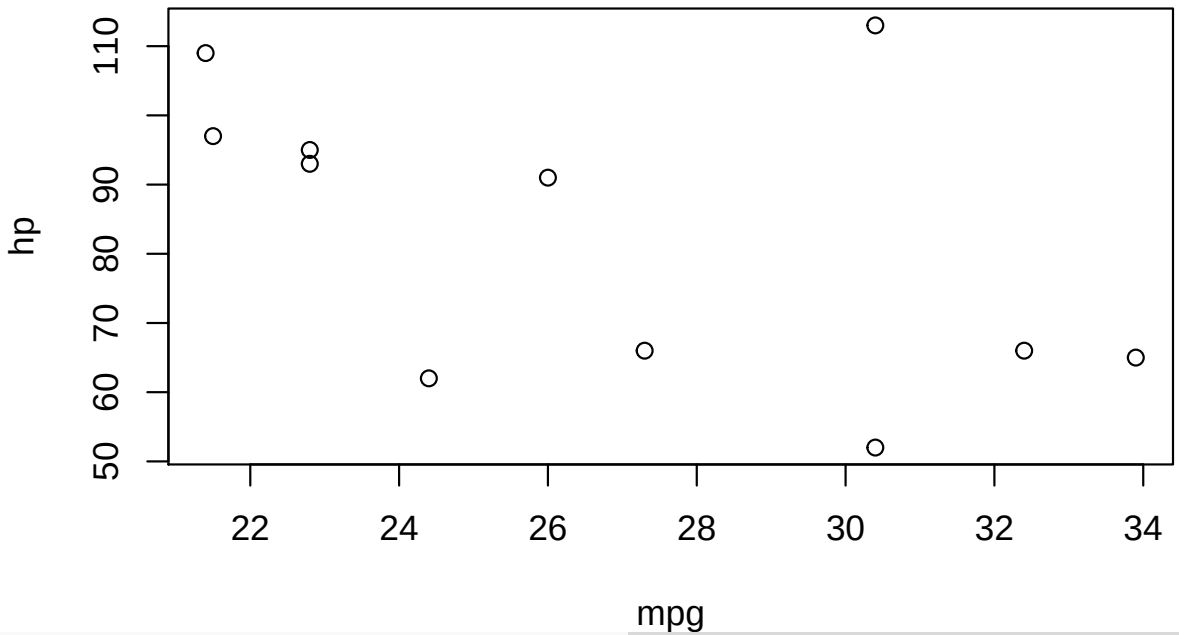
```
df <- na.omit(df)
df %<>% na.omit
```

magrittr::%T>%

... и Tee operator (производит side-effect):

```
mtcars %>%
  subset(cyl == 4, c(mpg, hp)) %T>%
  plot %>%
  colMeans

##       mpg        hp
## 26.66364 82.63636
```



Ссылки

При подготовке доклада были использованы следующие материалы:

- `https://github.com/hadley/tidyr`
- `https://github.com/hadley/dplyr`
- `https://github.com/smbache/magrittr`

Ссылки

При подготовке доклада были использованы следующие материалы:

- https://github.com/hadley/tidyr
- https://github.com/hadley/dplyr
- https://github.com/smbache/magrittr
- http://rpubs.com/bradleyboehmke/data_wrangling
- http://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Ссылки

При подготовке доклада были использованы следующие материалы:

- https://github.com/hadley/tidyr
- https://github.com/hadley/dplyr
- https://github.com/smbache/magrittr
- http://rpubs.com/bradleyboehmke/data_wrangling
- http://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
- http://datascience.la/dplyr-and-a-very-basic-benchmark/
- http://zevross.com/blog/2015/01/13/a-new-data-processing-workflow-for-r-dplyr-magrittr-tidyr-ggplot2/

Ссылки

При подготовке доклада были использованы следующие материалы:

- https://github.com/hadley/tidyr
- https://github.com/hadley/dplyr
- https://github.com/smbache/magrittr
- http://rpubs.com/bradleyboehmke/data_wrangling
- http://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
- http://datascience.la/dplyr-and-a-very-basic-benchmark/
- http://zevross.com/blog/2015/01/13/a-new-data-processing-workflow-for-r-dplyr-magrittr-tidyr-ggplot2/

Спасибо!