Основы программирования в R

Работа с данными с `tidyverse`: часть 1

Алла Тамбовцева, НИУ ВШЭ

Содержание

Загрузим библиотеку `tidyverse`, которую мы уже благополучно установили:

```
library(tidyverse)
```

В рамках этого занятия мы немного отойдем от политологии и поработаем с данными из файла `Characters.csv`, который содержит информацию по героям волшебного мира Дж.К.Роулинг. Файл взят с Kaggle, полное описание данных можно найти здесь.

В качестве разделителя столбцов в файле используется `;`, плюс, давайте сделаем так, чтобы текстовые столбцы считывались как текстовые, а не как факторные.

```
dat <- read.csv("https://allatambov.github.io/rprog/data/Characters.csv",
                sep = ";", stringsAsFactors = FALSE)
```

Введение в `dplyr` и оператор `%>%`

В библиотеке `dplyr` есть особый оператор `%>%` (*pipe operator*), который позволяет выполнять операции пошагово. Смысл этого оператора такой: возьми, то, что слева от `%>%` и передай это на вход функции, стоящей справа от `%>%`. Посмотрим на простом примере:

```
dat %>% head %>% View
```

В примере выше мы взяли датафрейм `dat`, передали его на вход функции `head()` для выбора первых строк и потом выбранные строки передали на вход функции `View()` для просмотра. Как можно заметить, в `head` и `View` уже нет ни скобок, ни названия датафрейма, потому что они и не нужны – R и так знает, с чем ему работать.

Для сортировки в `dplyr` есть функция `arrange()`. Внутри этой функции нужно указать столбец, в соответствии с которым мы хотим отсортировать строки в датафрейме. Отсортируем по столбцу `Name`, упорядочим героев по алфавиту:

```
dat %>% arrange(Name) %>% View
```

Важно: при применении функций через `%>%` изменения в исходном датафрейме не происходят. Чтобы сохранить изменения, нужно присвоить `dat` новое значение:

```
dat <- dat %>% arrange(Name)
```

При присваивании всегда обращайте внимание на то, что в конце строки с преобразованиями нет `View`, иначе в переменную будет сохранен не датафрейм, а пустой объект типа `NULL`.

Если нужно выполнить сортировку по убыванию, нужно внутри `arrange()` добавить функцию `desc()`, от *descending*:

```
dat %>% arrange(desc(Name)) %>% View
```

Выбор столбцов

Для выбора столбцов используется функция `select()`. Внутри этой функции можно просто перечислить названия столбцов, которые хотим выбрать, не оформляя их в виде вектора:

```
dat %>% select(Name, Gender, House, Wand) %>% View
```

Так как в R существует своя базовая функция `select()`, которая работает несколько иначе, иногда могут возникать конфликты названий. Если строчка выше не сработала, можно попробовать вместо `select()` написать `dplyr::select()`, чтобы R точно знал, откуда извлекать эту функцию.

Можем сохранить выбранные столбцы в новый датафрейм `personal`:

```
personal <- dat %>% select(Name, Gender, House, Wand)
```

Если нам нужно выбрать несколько столбцов, которые стоят в датафрейме рядом, это можно оформить в виде последовательности с `:`. Выберем столбцы, начиная с `Name` и заканчивая `Blood.status`, и выведем последние несколько строк на экран в отдельном окне:

```
dat %>% select(Name:Blood.status) %>%
  tail %>% View
```

Если мы хотим выбрать все столбцы, кроме некоторых, перед названиями столбцов можно добавить минус:

```
dat %>% select(-c(Id, Blood.status)) %>% View
```

Давайте сохраним изменения — уберем ненужный `Id` и статус (мы приличные люди и `Blood.status` нас не интересует):

```
dat <- dat %>% select(-c(Id, Blood.status))
```

Выбор строк

Для выбора строк в `dplyr` используется функция `filter()`. Выберем только те строки, которые соответствуют героям женского пола:

```
dat %>% filter(Gender == "Female") %>% View
```

Можем совместить `filter()` и `select()` и сохранить результат в `female`:

```
female <- dat %>% filter(Gender == "Female") %>%
  select(Name:Wand)
```

Если необходимо объединить несколько условий, используются уже знакомые операторы `&` для одновременного выполнения условий и `|` для условия *или-или*. Выберем представителей женского пола, у которых патронусом является кошка:

```
dat %>% filter(Gender == "Female" & Patronus == "Cat") %>% View
```

По умолчанию `filter()` считает, что все перечисленные внутри условия выполняются одновременно, поэтому оператор `&` можно опустить:

```
dat %>% filter(Gender == "Female", Patronus == "Cat") %>% View
```

Сформулируем условие *или-или*. Выберем строки, которые соответствуют представителям факультета Гриффиндор и Когтевран (Равенкло, если кому-то так ближе):

```
dat %>% filter(House == "Gryffindor" | House == "Ravenclaw") %>%
  View
```

Теперь давайте посмотрим на фильтрацию строк с учетом того, что мы узнали про регулярные выражения. Как работает функция `filter()`? Выражение в условии внутри `filter()` возвращает либо значение `TRUE`, либо значение `FALSE`, а потом `filter()` выбирает те строки в датафрейме, для которых было возвращено `TRUE`. Поэтому никто нам не мешает внутри `filter()` в качестве условия выставить функцию `str_detect()`, которая будет возвращать `TRUE` и `FALSE` в зависимости от того, есть ли в ячейке определенный текст или нет.

Давайте выберем те строки, где в значениях профессии встречается слово “professor” как с большой буквы, так и с маленькой:

```
dat %>% filter(str_detect(Job, "[P|p]rofessor")) %>% View
```

Если бы не знали про регулярные выражения, и нас бы интересовал текст безотносительно размера букв (строчные или заглавные), можно было бы воспользоваться функцией `regex()` и выставить опцию `ignore_case = TRUE`:

```
dat %>% filter(str_detect(Job,
                          regex("Professor", ignore_case = TRUE))) %>% View
```

Напоследок вернемся к обычному `str_detect()` и отметим, что у этой функции есть аргумент `negate`, который можно сделать равным `TRUE`, чтобы построить отрицание к выражению в `str_detect()`:

```
# все, кто не Professor, professor

dat %>% filter(str_detect(Job, "[P|p]rofessor", negate = TRUE)) %>% View
```

Добавление новых столбцов

Для добавления новых столбцов используется функция `mutate()`. Для примера добавим столбец `Female`, где значение 1 соответствует представителям женского пола, 0 — всем остальным:

```
# не забудем сохранить изменения через присваивание <-

dat <- dat %>%
  mutate(Female = ifelse(Gender == "Female", 1, 0))
```

Если нужно добавить более одного столбца за раз, необходимые выражения перечисляют в `mutate()` через запятую:

```
dat <- dat %>%
  mutate(Female = ifelse(Gender == "Female", 1, 0),
         Male = ifelse(Gender == "Male", 1, 0))
```

В `dplyr` есть несколько функций, относящихся к классу `mutate_`. Эти функции позволяют преобразовать все столбцы сразу или преобразовать те столбцы, которые удовлетворяют определенным условиям. Датафрейм `dat` нам сейчас не подойдет, возьмем встроенный датафрейм `swiss`. Можно запросить по нему `help()` и увидеть, что в нем хранятся довольно старые данные (1888 год) по 47 кантонам Швейцарии.

Все столбцы в этом датафрейме являются числовыми. Представим, что нам нужно каждый столбец логарифимировать. Чтобы избежать циклов и какого-то громоздкого кода, воспользуемся функцией `mutate_all()`:

```
# возвращает измененную копию датафрейма
# из логарифмов
```

```
swiss %>% mutate_all(log) %>% head
```

```
##   Fertility Agriculture Examination Education Catholic Infant.Mortality
## 1  4.384524    2.833213    2.708050  2.484907 2.298577         3.100092
## 2  4.420045    3.808882    1.791759  2.197225 4.440767         3.100092
## 3  4.527209    3.681351    1.609438  1.609438 4.536891         3.005683
## 4  4.452019    3.597312    2.484907  1.945910 3.519573         3.010621
## 5  4.342506    3.772761    2.833213  2.708050 1.640937         3.025291
## 6  4.332048    3.563883    2.197225  1.945910 4.506123         3.280911
```

Несложно догадаться, что если бы в `swiss` были текстовые столбцы, код выше бы не сработал. Для таких случаев пригодится функция `mutate_if()`. Внутри этой функции мы можем указать условие для отбора столбцов, и тогда преобразования будут применяться только к тем столбцам, для которых результатом проверки условия будет `TRUE`. Например, возьмем только числовые столбцы и применим к ним функцию для логарифмирования:

```
# log только для тех, где is.numeric = TRUE

swiss %>% mutate_if(is.numeric, log) %>% head
```

```
##   Fertility Agriculture Examination Education Catholic Infant.Mortality
## 1  4.384524    2.833213    2.708050  2.484907 2.298577         3.100092
## 2  4.420045    3.808882    1.791759  2.197225 4.440767         3.100092
## 3  4.527209    3.681351    1.609438  1.609438 4.536891         3.005683
## 4  4.452019    3.597312    2.484907  1.945910 3.519573         3.010621
## 5  4.342506    3.772761    2.833213  2.708050 1.640937         3.025291
## 6  4.332048    3.563883    2.197225  1.945910 4.506123         3.280911
```

Важно: функция `mutate_if()` возвращает измененную копию исходного датафрейма. Столбцы, которые не должны быть изменены, остаются как есть, а те, что должны быть изменены, преобразуются.

Если мы хотим применить какую-то функцию к конкретным столбцам, можно взять функцию `mutate_at()` и внутри нее в `vars()` перечислить названия столбцов:

```
swiss %>% mutate_at(vars(Agriculture, Catholic), log) %>%
  head
```

```
##   Fertility Agriculture Examination Education Catholic Infant.Mortality
## 1      80.2    2.833213          15        12 2.298577             22.2
## 2      83.1    3.808882           6         9 4.440767             22.2
## 3      92.5    3.681351           5         5 4.536891             20.2
## 4      85.8    3.597312          12         7 3.519573             20.3
## 5      76.9    3.772761          17        15 1.640937             20.6
## 6      76.1    3.563883           9         7 4.506123             26.6
```

И по-прежнему, возвращается копия датафрейма с внесенными изменениями.