Урок 1. Проверка условий в R

Всем привет! В этом модуле мы обсудим важные конструкции в R: условные конструкции, циклы и функции. Они часто бывают полезны, когда мы работаем сразу с несколькими файлами с данными, особенно если эти файлы неединообразны по структуре. Например, мы можем написать функцию, которая будет заходить в указанную пользователем папку с данными, открывать каждый файл в цикле, проверять, есть ли там нужные столбцы или строки, и если есть, собирать нужную информацию и объединять в общий файл. Кроме того, функции пригодятся для создания интерактивных дэшбордов в RShiny, так как процесс их создания похож на написание собственного приложения, где каждый блок кода включен в отдельную функцию.

Начнем с обзора условных операторов. Вы, скорее всего, знакомы с ними из Python или SQL, но кратко повторим их на примере работы с переменными. Пусть у нас есть две переменные с числом пользователей сайта до изменения дизайна (`users.before`) и после (`users.after`).

```
users.before <- 125
users.after <- 138
```

Проверим, правда ли, что число посетителей сайта после изменения дизайна увеличилось:

```
users.after > users.before  # правда
```

```
## [1] TRUE
```

Теперь проверим, правда ли, что разница в числе посетителей не менее 10:

```
users.after - users.before >= 10  # правда
```

```
## [1] TRUE
```

Вспомним условие равенства:

```
users.after - users.before == 0  # разница не 0
```

```
## [1] FALSE
```

И не-равенства:

```
users.after - users.before != 0
```

```
## [1] TRUE
```

Теперь перейдем к сложным или составным условиям. Как и в Python, для одновременного выполнения условия (пересечения) в R используется оператор `&` (аперсанд). Проверим, правда ли, что число посетителей сайта после изменения дизайна превышает число посетителей до изменений, но при этом меньше 150:

```
users.after > users.before & users.after < 150
```

```
## [1] TRUE
```

Для выполнения хотя бы одного из условий (объединение, не-исключающее «или») в R используется `|` (пайп). Проверим, выполняется ли хотя бы одно из условий: 1) число посетителей до изменений больше, чем до них; 2) число посетителей до изменений не менее 100.

```
users.before > users.after | users.before >= 100
```

```
## [1] TRUE
```

В Python код для проверки такого условия выглядел бы так же.

Так как второе условие выполняется, все выражение возвращает `TRUE`. Разумеется, если оба условия верны, то тоже будет возвращено `TRUE`. В R есть специальный оператор для исключающего «или», `xor()`. Это новый оператор, его мы до этого не рассматривали (в Python нет отдельного оператора `xor`), поэтому давайте разберем, для чего он служит.

Что означает исключающее «или»? Верно ровно одно условие, то есть в случае двух условий первое верное, второе при этом неверное или второе верное и первое при этом неверное. Проверим, правда ли, что только одно условие из примера выше верно.

```
xor(users.before > users.after, users.before >= 100) #
```

```
## [1] TRUE
```

Важно: для исключающее «или» нет специального символа, и хотя `xor()` называется в R оператором (можно проверить, запросив help), по своему устройству он является функцией.
В завершение урока давайте посмотрим, можем ли мы использовать те же операторы для сравнения элементов вектора с числом или для поэлементного сравнения векторов.

В завершение урока давайте посмотрим, можем ли мы использовать те же операторы для сравнения элементов вектора с числом или для поэлементного сравнения векторов.

Задача. Дан вектор со значениями температуры воздуха на первой неделе сентября.

```
temps <- c(19, 21, 24, 17, 16, 14, 15)
```

Сколько дней на первой неделе сентября температура воздуха была выше среднего по этой неделе?

```
mt <- mean(temps)  # посчитаем и созраним среднее по вектору
temps > mt  # сравниваем каждый элемент вектора с числом
```

```
## [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

Теперь посчитаем число `TRUE` — суммируем полученные значения (мы уже убедились ранее в том, что значения `TRUE` легко заменяются на 1 внутри R, поэтому для ответа на вопрос задачи достаточно суммировать число единиц)

```
sum(temps > mt)
```

```
## [1] 3
```

Если вспомните, что-то похожее можно делать и с массивами NumPy в Python: смотреть на выполнение условий для каждого элемента массива и смотреть на число значений `TRUE`.

Альтернативный способ решить эту задачу, не прибегая к явному подсчету значений `TRUE` (хотя внутри R это будет примерно той же операцией) нам уже знаком — сформулировать условие внутри уже знакомых квадратных скобок и посчитать длину полученного вектора:

```
length(temps[temps > mt])  # даже проще
```

```
## [1] 3
```

Что интересно, таким же образом можно сравнивать два вектора одинаковой длины. Сравним оценки двух студентов за четыре домашних задания:

```
st1 <- c(4, 5, 5, 4)
st2 <- c(4, 3, 4, 5)
st1 == st2  # проверяем равенство попарно
```

```
## [1]  TRUE FALSE FALSE FALSE
```

Действительно, только оценки за первую работу совпадают. А что будет, если сравнить векторы разной длины? Об этом вам предстоит подумать в практическом задании. Ссылку на задание вы найдете под этим видео.