Основы программирования в R

Векторы в R: часть 2

Алла Тамбовцева, НИУ ВШЭ

Содержание

Последовательности

Для создания векторов можно использовать последовательности (для владеющих Python: аналог `range()` и `arange()`, но в отличие от Python, здесь в вектор включаются оба конца). Например, последовательность из целых значений от 0 до 11:

```
0:11
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11
```

А вот последовательность из целых начений от 10 до 20 с шагом 3:

```
seq(from = 10, to = 20, by = 3)
```

```
## [1] 10 13 16 19
```

Названия аргументов, если мы сохраняем их порядок, можно опустить:

```
seq(10, 20, 3)
```

```
## [1] 10 13 16 19
```

Шаг в последовательности необязательно должен быть целым, он может быть дробным:

```
seq(10, 20, 0.5)
```

```
##  [1] 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5 16.0 16.5 17.0
## [16] 17.5 18.0 18.5 19.0 19.5 20.0
```

Векторы из повторяющихся значений

В R можно быстро составить вектор из повторяющихся значений. Например, три раза повторить “Repeat me”:

```
rep("Repeat me", 3)
```

```
## [1] "Repeat me" "Repeat me" "Repeat me"
```

Или четыре раза повторить вектор с двумя значениями 0 и 1:

```
rep(c(0, 1), 4)
```

```
## [1] 0 1 0 1 0 1 0 1
```

А можно попросить повторить каждое из значений вектора по четыре раза:

```
rep(c(0, 1), each = 4)
```

```
## [1] 0 0 0 0 1 1 1 1
```

Поиск уникальных значений и подсчет значений

А как получить вектор без повторяющихся значений? Для этого есть функция `unique()`:

```
p <- c(0, 4, 0, 6, 7, 4)
unique(p)
```

```
## [1] 0 4 6 7
```

Как посчитать, сколько раз в векторе встречаются различные значения? Для этого есть функция `table()`:

```
table(p)
```

```
## p
## 0 4 6 7
## 2 2 1 1
```

Если нас интересуют не абсолютные частоты, а доли значений, можно воспользоваться функцией `prop.table()`:

```
prop.table(p)
```

```
## [1] 0.0000000 0.1904762 0.0000000 0.2857143 0.3333333 0.1904762
```

Переведём доли в проценты:

```
prop.table(p) * 100
```

```
## [1]  0.00000 19.04762  0.00000 28.57143 33.33333 19.04762
```

Пропущенные значения

Можно создавать векторы с пропущенными значениями `NA`, от английского *not applicable*:

```
q <- c(0, 1, NA, NA)
q
```

```
## [1]  0  1 NA NA
```

Проверим, являются ли элементы пропущенными значениями:

```
is.na(q)
```

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

Выведем индексы пропущенных значений:

```r
which(is.na(q))
```

```
## [1] 3 4
```

Обычное равенство тут не сработает (вспомните про `None` в Python):

```r
which(q == NA)
```

```
## integer(0)
```

Обратите внимание: `NA` указывается без кавычек! Это не текст, который кодирует пропущенные значения, а особый «тип» данных; наличие `NA` не изменяет тип переменной, то есть, если `NA` встречаются в числовой переменной, переменная будет восприниматься R как числовая.

Наличие `NA` в векторе мешает выполнять некоторые действия с ним. Попробуем вычислить среднее значение по `q`:

```r
mean(q)
```

```
## [1] NA
```

Не получилось! Поэтому нужно задействовать дополнительный аргумент `na.rm` (от *NA remove*):

```r
mean(q, na.rm = TRUE)
```

```
## [1] 0.5
```

Это будет актуально и для других функций, которые вычисляют числовые характеристики, например, `min()`:

```r
min(q, na.rm = TRUE)
```

```
## [1] 0
```