Содержание

1. Управление наборами данных
2. Чтение и запись данных

1. Работа с наборами данных

Доступ к элементам структур данных

- В R используются несколько операторов для извлечения данных из объектов – “**[**”, “**[[**”, “**$**”.

  “**[**” – возвращает объект того же класса что и объект-оригинал. Пример – из вектора -> вектор, их списка -> список и т.д. Может быть выделено более 1 элемента!

  “**[[**” – используется для взятия элемента списка или таблицы (**Data Frame).** Класс элемента может быть различным. Может быть выделен только 1 элемент!

  “**$**” – применяется для выделения элемента списка или таблицы по имени **name**. Класс элемента не обязательно совпадает с исходными данными.

- Примеры использования оператора “[”.

```
> x <- c("a", "b", "c", "c", "d", "a")
> x[1]
[1] "a"
> x[2]
[1] "b"
> x[1:4]
[1] "a" "b" "c" "c"
> x[x > "a"]
[1] "b" "c" "c" "d"
> u <- x > "a"
> u
[1] FALSE  TRUE  TRUE  TRUE  TRUE  FALSE
> x[u]
[1] "b" "c" "c" "d"
```

Доступ к элементам матриц

- Доступ к элементам матриц осуществляется с использованием индексов в квадратных скобках – “**[**”. Первый индекс – **номер строки**, второй индекс – **номер столбца**.

```
> x <- matrix(1:6, 2, 3)
> x[1, 2]
[1] 3
> x[2, 1]
[1] 2
```

- Первый или второй индекс может отсутствовать. Тогда будет возвращена строка или столбец матрицы.

```
> x[1, ]
[1] 1 3 5
> x[, 2]
[1] 3 4
```

- По умолчанию возвращается вектор единичной длины того же класса что и исходный объект. Для получения объекта в виде матрицы необходимо установить параметр **drop**. Учет размерности данных - **drop = FALSE.**

```
> x <- matrix(1:6, 2, 3)
> x[1, 2]
[1] 3
> x[1, 2, drop = FALSE]
     [,1]
[1,]  3
```

```
> x <- matrix(1:6, 2, 3)
> x[1, ]
[1] 1 3 5
> x[1, , drop = FALSE]
     [,1] [,2] [,3]
[1,]  1    3    5
```

Доступ к элементам списка

- Для доступа к элементам списка можно использовать – “**[**”, “**[[**”, “**$**”.

```
> x <- list(foo = 1:4, bar = 0.6)
> x[1]
$foo
[1] 1 2 3 4
> x[[1]]
[1] 1 2 3 4
> x$bar
[1] 0.6
> x[["bar"]]
[1] 0.6
> x["bar"]
$bar
[1] 0.6
```

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> x[c(1, 3)]
$foo
[1] 1 2 3 4
$baz
[1] "hello"
```

- Оператор “[[” используется с индексами и именами, а “$” - только с именами объектов. “$” упрощает работу, т.к. не требуется определение порядка следования

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> name <- "foo"
> x[[name]] ## computed index for ‘foo’
[1] 1 2 3 4
> x$name ## element ‘name’ doesn’t exist!
NULL
> x$foo
[1] 1 2 3 4 ## element ‘foo’ does exist
```

- Оператор “[[” может принимать последовательность целых чисел и рекурсивно извлекать элементы из вложенного списка.

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
> x[[c(1, 3)]]
[1] 14
> x[[1]][[3]]
[1] 14
> x[[c(2, 1)]]
[1] 3.14
```

Частичное совпадение (Partial matching)

- **Partial matching** – определение элемента по его первым буквам имени. Используется для операторов “**[[**”, “**$**” с целью сокращения программного кода.

```
> x <- list(aardvark = 1:5)
> x$a
[1] 1 2 3 4 5
> x[["a"]]
NULL
> x[["a", exact = FALSE]]
[1] 1 2 3 4 5
```

- По умолчанию **Partial matching** работает для оператора “**$**”, но не работает для “**[[**”. В последнем случае необходимо использовать параметр `exact = FALSE`.

- Часто в ходе обработки данных необходимо найти и удалить из элементов объекта пропущенные значения типа **NA** или **NaN**. Процедура выполняется в два шага:

  1) Создается логический вектор, указывающий на элементы с пропущенными значениями.

  2) Из исходного объекта извлекаются «нормальные» элементы с помощью созданного логического вектора.

```
> x <- c(1, 2, NA, 4, NA, 5)
> bad <- is.na(x)
> x[!bad]
[1] 1 2 4 5
```

- Если необходимо сравнить два объекта и выделить пары соответственных элементов с непропущенными значениями, то используется функция **complete.cases**. Данный пример широко используется в ходе анализа биочипов ДНК, когда необходимо удалить гены, отсутствующие хотя бы на одном из анализируемых биочипов.

```
> x <- c(1,    2,  NA,  4,  NA,  5)
> y <- c("a", "b", NA, "d", NA, "f")

> good <- complete.cases(x, y)
> good
[1] TRUE TRUE FALSE TRUE FALSE TRUE

> x[good]
[1] 1 2 4 5

> y[good]
[1] "a" "b" "d" "f"
```

- Пример. Функция **complete.cases**.

```
> airquality[1:6, ]
  Ozone Solar.R  Wind  Temp  Month  Day
1    41    190   7.4    67      5    1
2    36    118   8.0    72      5    2
3    12    149  12.6    74      5    3
4    18    313  11.5    62      5    4
5    NA     NA  14.3    56      5    5
6    28     NA  14.9    66      5    6

> good <- complete.cases(airquality)
> airquality[good, ][1:6, ]
  Ozone Solar.R  Wind  Temp  Month  Day
1    41    190   7.4    67      5    1
2    36    118   8.0    72      5    2
3    12    149  12.6    74      5    3
4    18    313  11.5    62      5    4
7    23    299   8.6    65      5    7
```

Векторные операции (Vectorized operations)

- В R можно осуществлять операции над векторами. Это позволяет значительно упростить программный код, не программировать циклы, ускорить процесс написания программ.

```
> x <- 1:4; y <- 6:9
> x + y
[1] 7 9 11 13
> x > 2
[1] FALSE FALSE TRUE TRUE
> x >= 2
[1] FALSE TRUE TRUE TRUE
> y == 8
[1] FALSE FALSE TRUE FALSE
> x * y
[1] 6 14 24 36
> x / y
[1] 0.1666667 0.2857143 0.3750000 0.4444444
```

- Аналогично векторные операции осуществляются над матрицами.

```
> x <- matrix(1:4, 2, 2); y <- matrix(rep(10, 4), 2, 2)
> x * y ## element-wise multiplication
     [,1] [,2]
[1,]   10   30
[2,]   20   40
> x / y
     [,1] [,2]
[1,]  0.1  0.3
[2,]  0.2  0.4
```

- Оператор **“%”** используется для матричное перемножение по правилам линейной алгебры.

```
> x %*% y ## true matrix multiplication
     [,1] [,2]
[1,]   40   40
[2,]   60   60
```

2. Чтение и запись данных

Чтение данных

- Для чтения данных в R используются несколько базовых функций:

  `read.table, read.csv` – чтение табулированных данных

  `readLines` – чтение текстовых строк из файла

  `source` – загрузка в активное пространство памяти R–кодов (функции)

  `dget` – загрузка в активное пространство памяти одиночных R–объектов из (ASCII) текстовых файлов R

  `load` – загрузка объектов рабочего пространства

  `unserialize` – чтение одиночных R–объектов в бинарном формате

Запись данных

- Для записи данных используются функции:

`write.table` – запись табулированных данных

`writeLines` – запись текстовых строк в файла

`dump` – запись R–объектов в текстовый файл

`dput` – запись одиночных R–объектов в текстовый файл

`save` – запись объектов рабочего пространства

`serialize` – запись одиночных R–объектов в бинарном формате

Read.table

- **read.table(file, header, …)** одна из основных функций чтения небольших табулированных данных. Рассмотрим основные управляющие аргументы функции:

  file – служит для указания пути к импортируемому файлу. Пример – **file = "c:/Temp/MyData.dot"** . В качестве имени файла можно указать URL-ссылку.

  header – логический индикатор, указывает является ли первая строка заголовком (названия колонок таблицы). По умолчанию **header = FALSE**.

  sep – указывает на символ разделения между колонками. По умолчанию **sep = " ".** Для файлов .csv **sep = ",".**

  colClasses – вектор типа character, длина вектора равна количеству колонок данных, содержимое определяет типы данных в колонках.

Read.table

nrows – определяет число строк, которое должно быть считано из файла.

comment.char – указывает, что является символом, после которого идет текст комментария.

skip – указывает на число строк от начала файла, которые должны быть пропущены при чтении данных.

stringsAsFactors – логический флаг, указывает, надо ли перевести символьные переменные в данные типа **factor**. По умолчанию **stringsAsFactors = TRUE**.

Read.table

- Для небольших наборов данных рекомендуется использовать **read.table** с параметрами по умолчанию.

```
data <- read.table("foo.txt")
```

- В результате выполнения функции:

  - data имеет структуру **Data Frame**
  - Автоматически пропускаются троки после **#**
  - Автоматически определятся типы колонок данных. Однако для ускорения работы рекомендуется определять типы данных напрямую.

- **read.csv** работает аналогично **read.table,** за исключением того, что разделитель колонок – запятая. Подобные файлы предоставляются программами типа **MS Excel**.

- Для **интерактивного** выбора загружаемого файла, который хранится вне рабочей папки, можно использовать вспомогательную функцию `file.choose`. Выполнение функции приводит к открытию диалогового окна Windows.

```
data <- read.table(file.choose( ), header=TRUE,sep = " ,")
```

- Для работы с большими файлами данных можно ускорить загрузку данных используя аргумент `colClasses`.

```
initial <- read.table("datatable.txt", nrows = 100)
classes <- sapply(initial, class)
tabAll <- read.table("datatable.txt" ,colClasses = classes)
```

dput, dget

- Для работы с текстовыми данными (не табулированными) используются функции **dput, dget**. Преимущество хранения данных в текстовом формате – их можно редактировать. В случае повреждения имеют высокий потенциал восстановления. Файлы содержат метаданные.

```
> y <- data.frame(a = 1, b = "a")
> dput(y)
structure(list(a = 1,
b = structure(1L, .Label = "a",
class = "factor")),
.Names = c("a", "b"), row.names = c(NA, -1L),
class = "data.frame")
> dput(y, file = "y.R")
> new.y <- dget("y.R")
> new.y
  a b
1 1 a
```

Dump, source

- Функция **dump** позволяет записывать сразу несколько объектов. Для восстановления объектов из текстовых файлов используется функция **source**.

```
> x <- "foo"
> y <- data.frame(a = 1, b = "a")
> dump(c("x", "y"), file = "data.R")
> rm(x, y)    # удаление объектов x и  y
> source("data.R")
> y
   a  b
1  1  a
> x
[1] "foo"
```

Функции установления соединений с внешними источниками данных

- Существует несколько альтернативных способов для установления соединения (**connection**) с внешними источниками данных.

  `file` – создать связь с файлом (стандартный несжатый файл).

  `gzfile` – создать связь с файлом, сжатым с gzip.

  `bzfile` – создать связь с файлом, сжатым с bzip2.

  `url` – открыть связь с web-страницей.

file

```
> str(file)
function (description = "", open = "", blocking = TRUE,
encoding = getOption("encoding"))
```

- **description** - имя файла.
- **open** - тип работы с кодом файла.

  “r” – только для чтения.

  “w” – только для записи и создания нового файла.

  “a” – добавление (appending).

  “rb”, “wb”, “ab” – по аналогии для бинарных файлов.

- **connection** позволяет упростить работу с большими файлами, когда нет необходимости загружать весь исходный файл данных.

```
con <- file("foo.txt", "r")
data <- read.csv(con)
close(con)
```

То же самое

```
data <- read.csv("foo.txt“)
```

readLines

- В режиме **connection** можно использовать функцию **readLines** для чтения заданного количества строк из файлов.

```
> con <- gzfile("words.gz")
> x <- readLines(con, 10)
> x
[1] "1080"      "10-point"  "10th"      "11-point"
[5] "12-point" "16-point"  "18-point"  "1st"
[9] "2"         "20-point"
```

readLines

- Возможно чтение строк из web-страниц.

```
## This might take time
con <- url("http://www.jhsph.edu", "r")
x <- readLines(con)
> head(x)
[1] "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0…
[2] ""
[3] "<html>"
[4] "<head>"
[5] "\t<meta http-equiv=\"Content-Type\”content=\”…
```

Полезные функции

- Подлежащий импортированию файл рекомендуется разместить в рабочей папке. Для отображения рабочей папки используется функция **getwd** .

```
> getwd( )
[1] "C:/Documents and Settings/Mikalai/My Documents"
```

- Изменить рабочую папку можно с помощью функции **setwd** .

```
> setwd("C:/Documents and Settings")
```

- Для создания новой директории применяется функция **dir.create** .

Полезные функции

- `ls` – выводит на экран список объектов в текущем рабочем пространстве.

```
> ls( )
 [1] "a"      "b"      "dx"     "g"      "j"      "mean_y" "N"
```

- `rm` – удалить один или несколько объектов из рабочей памяти.

```
rm( list = ls( ) )   # Remove all objects
```

- `History(#)` – выводит на экран последние `#` команд.
- `str( )` – вывод информации о структуре объекта.
- `q( )` – выйти из программы.

Выводы

В лекции рассмотрены:

1. Операции доступа к элементам данных с использованием операторов “**[**”, “**[[**”, “**$**”.
2. Обработка пропущенных значений.
3. Вычисления с векторами и массивами данных.
4. Способы загрузки и экспорта данных в R.