Chapter 1

Rstudio

- 赋值 `x <- 3*4` `3*4 -> x`
- `()` 表示输出运算值，优先运算
- 运算符
  - Unary
    - `-` for arithmetic negation
    - `!` for Boolean
  - Binary
  - Comparisons
    - `==`, `!=`, `<`, `>`, `>=`, `<=` 返回true或false
  - Boolean
    - `&`, `|`
- R 是通过函数实现运算
- `seq()` makes regular sequences of numbers `seq(1,10)`
- for help
  - `?seq`
  - `help(seq)`
- packages
  - `install.package("package-name-in-quotes")`
- environment
  - `objects()`,`ls()`
  - `rm()` remove
  - `rm(list=ls())` **remove everything**
- Restart R session frequently : session > Restart R
- Rmarkdown
  - code chunk

    `command` + `shift` + `return` run

Git&GitHub

- Version control

  git will track and version your files, GitHub stores this online and enables you to collaborate with others (and yourself).

Chapter 2

Data Type

- All data is represented in **binary** format by bits
  - Booleans

    `TRUE`, `FALSE`
  - Integers
  - Characters

    strings = sequences of characters
  - Floating point numbers

    a fraction times an exponent, in binary form

    Finite precision $\Rightarrow$ arithmetic on doubles $\neq$ arithmetic on $\mathbb{R}$

    Example.

    ```
    0.45 == 3 * 0.15
    ```

    ## [1] FALSE

    ```
    0.45 - 3 * 0.15
    ```

    ## [1] 5.551115e-17

    Solution.

    Usually better to use `all.equal()` than exact comparison `==`

    ```
    all.equal(0.45,3*0.15)
    ```

    ## [1] TRUE

  - Missing or ill-defined values

    `NA`, `NaN`
- 关于数据类型的函数
  - `typeof()` returns the type
  - `is.foo()` returns Booleans for whether the argument if of type foo

    (`is.character`, `is.na`, ...)
  - `as.foo()` cast its argument to type foo (可能出错)

Example.

```
as.character(5/6)
```

[1] "0.833333333333333"

```
as.numeric(as.character(5/6))
```

[1] 0.8333333

```
print(as.numeric(as.character(5/6)), digits = 12)
```

[1] 0.833333333333

```
6*as.numeric(as.character(5/6))
```

[1] 5

```
5/6 == as.numeric(as.character(5/6))
```

[1] FALSE

- variables
  - a few variables are built in (`pi` , ...)

Data structures

Vectors

- A vector is a sequence of values, **all of the same type**
- `c()` returns a vector containing all its arguments in order

```
x <- c(7, 8, 10, 45)
```

- 调用
  - `x[1]` is the first element
  - `x[4]` is the 4th element
  - `x[-4]` is a *vector* containing all *but* the fourth element
  - 用向量调用：

```
x[c(2,4)]
```

[1] 8 45

```
x[c(-1,-3)]
```

[1] 8 45

- Boolean vector 调用：

  ```
  x[x>9]
  ```

  ## [1] 10 45

  - `which()` turns a Boolean vector in vector of `TRUE` indices (返回指数)

    ```
    places <- which(x > 9)
    places
    ```

    ## [1] 3 4

- `vector(length=6)` returns an *empty vector* of length 6

  helpful for filling things up later:

  ```
  weekly_hours <- vector(length=5)
  weekly_hours[5] <- 8
  ```

- Vector arithmetic
  - Operators apply to vectors *pairwise* or *elementwise*
- Recycling

  Recycling repeat elements in shorter vector when combined with longer

  ```
  x + c(-7,-8)
  ```

  ## [1] 0 0 3 37

  x有4个元素，于是`c(-7,-8)`被自动按循环填充.
- Comparisons
  - Pairwise comparisons

    ```
    x > 9
    ```

    ## [1] FALSE FALSE TRUE TRUE
  - To compare whole vectors, use `identical()` or `all.equal()`:

    Boolean operators work elementwise, while `identical()` or `all.equal()` compares vectors as whole.

```
(x>9) & (x<20)
```

[1] FALSE FALSE TRUE FALSE

Difference between `identical()` and `all.equal()` :

```
identical(c(0.5 - 0.3, 0.3 - 0.1), c(0.3 - 0.1, 0.5 - 0.3))
```

[1] FALSE

```
all.equal(c(0.5 - 0.3, 0.3 - 0.1), c(0.3 - 0.1, 0.5 - 0.3))
```

[1] TRUE

- Name

  You can give names to elements or components of vectors `names()`

  ```
  names(x) <- c("v1","v2","v3","fred")
  ```

  调用：

  ```
  x[c("fred","v1")]
  ```

  ## fred v1

  ## 45   7

  ```
  which(names(x)=="fred")
  ```

  ## [1] 4

- Functions on vectors
  - `mean()` , `median()` , `sd()` , `var()` , `max()` , `min()` , `length()` , `sum()`
  - `sort()` returns a new vector
  - `hist()` takes a vector of numbers and produces a histogram
  - `ecdf()` prodeces a cumulative-density-function object
  - `summary()` gives a five-number summary of numerical vectors
  - `any()` , `all()`
- Not all functions require arguments
  - `date()` returns a vector of current time

Arrays

is a vector structure

```r
x <- c(7, 8, 10, 45)
x.arr <- array(x, dim=c(2,2))
x.arr
```

```
##      [,1] [,2]
## [1,]    7   10
## [2,]    8   45
```

- n 维数组，`dim` is a length n vector `dim()`
- `typeof()` returns the type of the *elements*

  ```r
  typeof(x.arr)
  ```

  ```
  ## [1] "double"
  ```

- `str()` gives the *structure*

  ```r
  str(x.arr)
  ```

  ```
  ## num [1:2, 1:2] 7 8 10 45
  ```

- `attributes()` 特性

  ```r
  attributes(x.arr)
  ```

  ```
  ## $dim
  ## [1] 2 2
  ```

- 调用

  ```r
  x.arr[1,2]
  ```

  ```
  ## [1] 10
  ```

  ```r
  x.arr[3]
  ```

  ```
  ## [1] 10
  ```

  ```r
  x.arr[c(1:2),2]
  ```

  ```
  ## [1] 10 45
  ```

  ```r
  x.arr[,2]
  ```

  ```
  ## [1] 10 45
  ```

- Functions on arrays
  - Using a vector-style function on a *vector structure* will go down to the underlying vector

`which()`

- *unless* the function is set up to handle arrays specially
- Many functions do preserve array structure

  `+` , ...
- Others specifically act on each row of column of the array separately

  `rowSums()` , `colSums()` , ...

Matrices

a specialization of a 2D array

- 赋值

```
factory <- matrix(c(40,1,60,3),nrow=2)
```

  `ncol` - specify number of columns, `byrow=TRUE` - to fill by rows.
- Compare whole matrices with `identical()` or `all.equal()`
- Matrix multiplication:
  - `%*%`
  - Mutiplying *matrices* and *vectors*: R silently casts the vector as either a row or a column matrix

```
output <- c(10,20)
factory %*% output
```

```
##      [,1]
## [1,] 1600
## [2,]   70
```

```
output %*% factory
```

```
##      [,1] [,2]
## [1,]  420  660
```

- `t()` : transpose
- `det()` : Determinant
- `diag()`
  - extract the diagonal entries of a matrix
  - used to *change* the diagonal:

```
diag(factory) <- c(35,4)
factory
```

```
##      [,1] [,2]
## [1,]   35   60
## [2,]    1    4
```

- create a diagonal or identity matrix

```
diag(c(3,4))
```

```
##      [,1] [,2]
## [1,]    3    0
## [2,]    0    4
```

```
diag(2)
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

- **Inverting** a matrix: `solve()`

```
solve(factory)
```

```
##             [,1]      [,2]
## [1,]  0.05000000 -1.0000000
## [2,] -0.01666667  0.6666667
```

不能直接`factory^(-1)`！！

- Name

  `rownames()`, `colnames()`

- `rowMeans()`, `colMeans()`, `rowSums()`, `colSums()` input matrix, output vector
- `summary()` vector-style summary of *column*
- `apply()` : 3 arguments - the array/matrix, 1/2 for row/column, the function

```
apply(factory,1,mean)
```

```
## [1] 50  2
```

Lists

not necessarily all of the same type

```
my.distribution <- list("exponential",7,FALSE)
my.distribution
```

```
## [[1]]
## [1] "exponential"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] FALSE
```

- Accessing (调用)

  Use `[]` or `[[]]`

  - `[]` can with vectors, `[[]]` only with a single index
  - `[[]]` drops names and structures
- Expanding and contracting lists:
  - `c()`

    ```
    my.distribution <- c(my.distribution,7)
    ```

  - Chop off the end of a list by setting the length to something smaller
- Naming list elements: `names()`
  - Use the name to access

    ```
    my.distribution[["family"]]
    ```

    ```
    my.distribution["family"]
    ```

    (returns a list)
  - special short-cut way of using names: `$` , drops names and structures. (same with `[[]]`)

    ```
    my.distribution$family
    ```

  - Removing a named list element: `NULL`

    ```
    my.distribution$family <- NULL
    ```

- Key-Value pairs/dictionaries/associative arrays/hashes

Dataframes

the classic data table

- *Not* just a matrix because columns can have different types.
- Many matrix functions also work for dataframes

- `rbind()` , `cbind()` : add rows or columns to an array or dataframe.

Structures of structures

Most complicated objects are (usually) lists of data structures.

Chapter 3

Visualization: ggplot2

Install

- When `library()` packages, might be a few name conflicted - alerting

  use `library(conflicted)` → `conflict_prefer`

- Use `library()` each time after you restart R

Plotting with `ggplot2`

- A grammer for graphics
- likes data in *long* format

grammer

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  )  +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

- 7 parameters
- `+` must be placed at the end of each line containing a layer!!
- `aes()` claims the aesthetics

plotting

- `ggplot()` :

  ```
  ggplot(data = ca)
  ```

- Add geoms: use `+` operator

  ```
  ggplot(data = ca) +
    geom_point(aes(x = year, y = visitors))
  ```

    - `geom_point()` for scatter plots, dot plots, etc.
    - `geom_bar()` for bar charts.
    - `geom_line()` for trend lines, time-series, etc.
-

Others

`complete.cases()`