Exercise I: **R**: Programming Language for Data Analysis

Wen-wen Tung

5/21/2019

Following the Warm-up demonstration, this set of notes provide further explanations and instructions on **R**'s data types and data structure, as well as examples showing how **R** manages its memories. Self- or peer-assess your learning by completing the assigned questions.

---

1. Create variables

1.1 Variable names and assignment

Rules for variable names in **R**:

- Variable names in **R** ARE case sensitive, so y is not the same as Y.
- Variable names should NOT begin with numbers (e.g., 1x) or symbols (e.g., %x).
- Variable names should NOT contain blank spaces (use back.pay not back pay).

R "assign" values to objects by the "gets arrow" `<-` . For example, to create a scalar constant x with value 1 (that is, x gets value 1), we type:

```
x <- 1
```

1.2 Integer

R is an object-oriented programming language. An object is created and assigned with a class. Like C or Fortran, **R** uses the integer class.

Try the following:

```
x <-c(1,2,3,4)
is.integer(x)
```

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

```
is.numeric(x)
```

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

```
class(x)
```

```
## [1] "numeric"
```

Now, change the object class and type of x to integer:

```
x <- as.integer(x)
is.integer(x)
```

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

```
is.numeric(x)
```

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

```
class(x)
```

```
## [1] "integer"
```

further more, the function `integer` acts like `trunc` when applied to real numbers, and removes the imaginary part when applied to complex number

```
x <- 1.8
as.integer(x)
```

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

```
x <- -1.8
as.integer(x)
```

```
## [1] -1
```

```
x <- 1 + 2i
as.integer(x)
```

```
## Warning: imaginary parts discarded in coercion
```

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

1.3 Factors

Factors are categorical variables that have a fixed number of levels. For example, a factor vector called gender can have two levels: ‘female’ and ‘male’:

```
gender <- factor(c("female", "male", "female", "male", "female"))
class(gender)
```

```
## [1] "factor"
```

Following are some functions dealing with factors:

```
fruits <- factor(c("orange","kiwi","cherry","apple","kiwi","orange","apple","apple","melon"))
is.factor(fruits)
```

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

```
levels(fruits)
```

```
## [1] "apple"  "cherry" "kiwi"   "melon"  "orange"
```

```
nlevels(fruits)
```

```
## [1] 5
```

```
length(levels(fruits))
```

```
## [1] 5
```

By default, factor levels are treated in alphabetical order. If you prefer otherwise, then you can define it on your own:

```
fruits <- factor(c("orange","kiwi","cherry","apple","kiwi","orange","apple","apple","melon"),
                 levels=c("melon","apple","orange","kiwi","cherry"))
levels(fruits)
```

```
## [1] "melon"  "apple"  "orange" "kiwi"   "cherry"
```

Factors are in fact stored intenerally by **R** as (numeric) integers. To turn factor levels into numbers (integers) use `unclass` function:

```
as.vector(unclass(fruits))
```

```
## [1] 3 4 5 2 4 3 2 2 1
```

Here is a general rule to reveal the structure of an object:

- Use `class` to determine the object class
- Use `mode` to strip away the object-oriented features and reveal the underlying structure
- Use `str` to show the internal structure and contents

```
class(fruits)
```

```
## [1] "factor"
```

```
mode(fruits)
```

```
## [1] "numeric"
```

```
str(fruits)
```

```
##  Factor w/ 5 levels "melon","apple",..: 3 4 5 2 4 3 2 2 1
```

1.4 Generate sequences and repeats of data

```
#### A sequence from 0 up to 10
0:10
```

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

```
#### A sequence from 10 down to 0
10:0
```

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

Generating a sequence given a step:

```
#### A sequence from -1 to 1 with interval of 0.2

seq(-1, 1, by=.2)
```

```
##  [1] -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4  0.6  0.8  1.0
```

```
#### A sequence from 1 to -1 with interval of -0.2, note that "by" can be dropped

seq(1, -1, -0.2)
```

```
##  [1]  1.0  0.8  0.6  0.4  0.2  0.0 -0.2 -0.4 -0.6 -0.8 -1.0
```

Generating a sequence given a length:

```
#### A sequence of length 10 from 0 with a step of 0.1

seq(from=0, by=0.1, length=10)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

```
#### A sequence having the same length as another vector x
x <- 1:15
seq(0, by=0.1, along=x)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
```

```
seq(from=0,to=1.4,along=x)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
```

Create a vector of sequences

```
sequence(5)
```

```
## [1] 1 2 3 4 5
```

Question 1: Describe what you get with the following commands generating x and y, and explain why.

```
x <- sequence(c(3,5,7,4))

y <- seq(length=11, from=-1, by=.2)
```

The function to generate repeats of numbers or characters is `rep` . For example, we could generate five 8s with:

```
rep(8,5)
```

```
## [1] 8 8 8 8 8
```

Question 2: Compare the following results of r1, r2, r3, and r4. Are they the same or different, why?

```
x <- 1:3

### repeat x 5 times

r1 = rep(x, times=5)

### Compare with this:

r2 = rep(x, each=5)

### Then, compare with this:

r3 = rep(x, each = 5, times = 5)

### How about this?

r4 = rep(x, c(1,3,5))
```

Question 3: Write a command using `rep` to generate the following result:

[1] “rainy” “cloudy” “cloudy” “thunder” “thunder” “thunder” “foggy” “foggy” “clear”

```
#### Answer
rep(c("rainy","cloudy","thunder","foggy", "clear"),c(1,2,3,2,1))
```

```
## [1] "rainy"   "cloudy"  "cloudy"  "thunder" "thunder" "thunder" "foggy"
## [8] "foggy"   "clear"
```

1.5 Missing values, infinity, and things that are not numbers

Calculations can lead to plus or minus infinity. Calculations involving infinity can be evaluated.

```
3/0
```

```
## [1] Inf
```

```
-12/0
```

```
## [1] -Inf
```

```
exp(-Inf)
```

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

```
0/Inf
```

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

```
(0:3)^Inf
```

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

Some calculations lead to quantitites that are not numbers `NaN` .

```
0/0
```

```
## [1] NaN
```

```
Inf-Inf
```

```
## [1] NaN
```

```
Inf/Inf
```

```
## [1] NaN
```

Missing values are expressed as `NA`.

```
x <- c(1:3, NA, NA, 11:15, NA, 21:24)
is.na(x)
```

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

```
y[is.na(x)]
```

```
## [1] -0.4 -0.2  1.0
```

```
y[!is.na(x)]
```

```
##  [1] -1.0 -0.8 -0.6  0.0  0.2  0.4  0.6  0.8   NA   NA   NA   NA
```

```
which(is.na(x))
```

```
## [1]  4  5 11
```

```
which(!is.na(x))
```

```
##  [1]  1  2  3  6  7  8  9 10 12 13 14 15
```

Many functions do not work by default when there are missing values, such as `mean`. But, there are ways to get around it by using `na.rm=TRUE`.

```
mean(x)
```

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

```
mean(x, na.rm=TRUE)
```

```
## [1] 13.41667
```

The following line fill in the missing values in the x vector with 999:

```
ifelse(is.na(x), 999, x)
```

```
##  [1]   1   2   3 999 999  11  12  13  14  15 999  21  22  23  24
```

---

2. Data Structure

R's base data structures can be organized by their dimensionality (1D, 2D, or nD) and whether they are homogeneous (all contents must be of the same type) or heterogeneous (contents can be of different types). There are five data types most often used in data analysis:

- Atomic vector (1D, Homogeneous)
  - Common types from the least to the most flexible: logical, integer, double (numeric), and character
  - Individual numbers or strings are vectors of length one
- Matrix (2D, Homogeneous)
- Array (nD, Homogeneous)
- List (1D, sometimes recursive, Heterogeneous)
  - Used to build up more complicated data structure
- Dataframe (2D, Heterogeneous)

2.1 Vectors

Vectors come in two favors: atomic vectors and lists. They have three common properties:

- Type, `typeof()`, what it is.
- Length, `length()`, how many elements it contains.
- Attributes, `attributes()`, additional metadata.

An atomic vector x is usually created by combining values with the combine function `c`:

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7, seq(14, 17, 0.3))
attr(x, "x_attribute") <- "This is an atomic vector"
```

Question 4: Print x to see its contents. Use `typeof`, `length`, and `attributes` to find out about the vector's three basic properties. Then, reveal the structure of x using `class`, `mode`, and `str`. Describe each of the results.

```
#### Answer
x
```

```
##  [1] 10.4  5.6  3.1  6.4 21.7 14.0 14.3 14.6 14.9 15.2 15.5 15.8 16.1 16.4
## [15] 16.7 17.0
## attr(,"x_attribute")
## [1] "This is an atomic vector"
```

```
length(x)
```

```
## [1] 16
```

```
typeof(x)
```

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

```
attributes(x)
```

```
## $x_attribute
## [1] "This is an atomic vector"
```

```
class(x)
```

```
## [1] "numeric"
```

```
mode(x)
```

```
## [1] "numeric"
```

```
str(x)
```

```
##  atomic [1:16] 10.4 5.6 3.1 6.4 21.7 14 14.3 14.6 14.9 15.2 ...
##  - attr(*, "x_attribute")= chr "This is an atomic vector"
```

Following expressions effectively lead to the same outcome:

```
y <- c(10.4, 5.6, 3.1, 6.4, 21.7)

assign("y", c(10.4, 5.6, 3.1, 6.4, 21.7))

c(10.4, 5.6, 3.1, 6.4, 21.7) -> y
```

Following lines further explore the vector y:

```
#### calculation with no assignment

1/y
```

```
## [1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
```

```
#### I only want to display 2 significant figures

round(1/y, 2)
```

```
## [1] 0.10 0.18 0.32 0.16 0.05
```

```
#### finding the third element of the vector y

y[3]
```

```
## [1] 3.1
```

Vector elements can have names. Their values can be retrived via names:

```
names(y) <- c("Mon", "Tue", "Wed","Thu","Fri")
print(y)
```

```
##  Mon  Tue  Wed  Thu  Fri
## 10.4  5.6  3.1  6.4 21.7
```

```
y["Wed"]
```

```
## Wed
## 3.1
```

Question 5: In one command line, find the first through the third elements of y. Then, in another command line, find the Wed and Fri values of the vector y (Hint: Use sequence-generating `:`, or use `c` and the names of vectors).

```
#### Answer

y[1:3]
```

```
##  Mon  Tue  Wed
## 10.4  5.6  3.1
```

```
y[c("Wed","Fri")]
```

```
##  Wed  Fri
##  3.1 21.7
```

2.2 Matrix

```
### Create a 5 by 4 numeric matrix

z <- matrix(1:20, nrow=5, ncol=4)

### Discover the elements in z
### Can you picture these on your own?

z[4,]
```

```
## [1]  4  9 14 19
```

```
z[2,3]
```

```
## [1] 12
```

```
z[,3]
```

```
## [1] 11 12 13 14 15
```

```
z[c(1,5), c(3:4)]
```

```
##      [,1] [,2]
## [1,]   11   16
## [2,]   15   20
```

Another way to create a matrix, using `dim` :

```
z <- 1:8
dim(z) <- c(4,2)
z
```

```
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    6
## [3,]    3    7
## [4,]    4    8
```

```
dim(z) <- c(2,4)
z
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Yet another way, using `nrow` and `ncol` .

```
cells <- c(1,26,24,68,NA,33,49,17,55)

cells
```

```
## [1]  1 26 24 68 NA 33 49 17 55
```

```
rlabels <- c("R1", "R2", "R3")
clabels <- c("C1", "C2", "C3")

### Note the 'byrow' option and the line break
mymatrix <- matrix(cells, nrow=3, ncol=3, byrow=TRUE,
                   dimnames = list(rlabels, clabels))

mymatrix
```

```
##    C1 C2 C3
## R1  1 26 24
## R2 68 NA 33
## R3 49 17 55
```

Question 6: 1) Calculate the mean of mymatrix using `mean` , discarding the missing value. 2) Calculate the mean of each row of mymatrix using `rowMeans` , discarding the missing value. 3) Calculate the sum of each column using `colSums` , discarding the missing value. 4) Fill in the missing value with -999 using `ifelse` . (Hint: Use help `?` to find out how to use unfamiliar functions)

```
#### Answer
mean(mymatrix, na.rm=TRUE)
```

```
## [1] 34.125
```

```
rowMeans(mymatrix, na.rm=TRUE)
```

```
##       R1       R2       R3
## 17.00000 50.50000 40.33333
```

```
colSums(mymatrix, na.rm=TRUE)
```

```
##  C1  C2  C3
## 118  43 112
```

```
ifelse(is.na(mymatrix), -999, mymatrix)
```

```
##    C1   C2 C3
## R1  1   26 24
## R2 68 -999 33
## R3 49   17 55
```

2.3 Arrays

Matrices can be generalized to 3-dimensional or even n-dimensional structures as arrays:

```
A <- sequence(12)
dim(A) <- c(2,3,2)
print(A)
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

Note here, **R** prints one slice of the 3-D structure at a time.

2.4 List

Unlike vectors, whose elements are “homogeneous” of the same modes, lists can contain heterogeneous elements of different modes. Lists can even recursively contain other structured objects, such as lists and dataframes.

To create a list from individual data items, use the `list` function:

```
x<-2.71828; y<- "thunderstorm"; z <- c(1,3,5)
Lst <- list(x,y,z, mean)
print(Lst)
```

```
## [[1]]
## [1] 2.71828
##
## [[2]]
## [1] "thunderstorm"
##
## [[3]]
## [1] 1 3 5
##
## [[4]]
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x7fedde3fb120>
## <environment: namespace:base>
```

Note that when **R** prints the list, it identifies each list element by its position ([[1]], [[2]], [[3]], [[4]]) and prints the element’s value (e.g., [1] 2.71828) under its position. Following example shows how to select list elements by position:

```
Lst[c(1,3)]
```

```
## [[1]]
## [1] 2.71828
##
## [[2]]
## [1] 1 3 5
```

```
#### Note the difference
#### This is the third element of the list
print(Lst[[3]])
```

```
## [1] 1 3 5
```

```
class(Lst[[3]])
```

```
## [1] "numeric"
```

```
#### This is the list that contains the elements taken from the 3rd element of Lst
print(Lst[3])
```

```
## [[1]]
## [1] 1 3 5
```

```
class(Lst[3])
```

```
## [1] "list"
```

Each element of a list can have a name, and can be retrieved by its name. For eample:

```
Lst <- list(Number=x,Weather=y,Records=z, Operation=mean)

#### Examine following results
Lst[["Weather"]]
```

```
## [1] "thunderstorm"
```

```
Lst$Weather
```

```
## [1] "thunderstorm"
```

```
Lst[c("Weather","Records")]
```

```
## $Weather
## [1] "thunderstorm"
##
## $Records
## [1] 1 3 5
```

```
Lst["Records"]
```

```
## $Records
## [1] 1 3 5
```

In computing there are instances that a vector is more useful than a list. Basic statistical functions work on vectors but not on lists, for example.

2.4.1 Subsetting Lists

Lists are very versatile but can appear confusing when we want to retrive the right information. Subsetting a list works in the same way as subsetting an atomic vector. Using `[` will always return a list; `[[` and `$` let you pull out the components of the list. `[[` is similar to `[`, except it can only return a single value and it allows you to pull pieces out of a list. `$` is a useful shorthand for `[[` combined with character subsetting.

“If list x is a train carrying objects, then x[[5]] is the object in car 5; x[4:6] is a train of cars 4-6. – @RLangTip”

Question 7: Given following two lists `a` and `b`, (1) find the value of `orange` in `a`, (2) write an alternative way to retrieve `b[[c("a", "b", "c", "d")]]`

```
a <- list(orange = 1, ball = 2)
b <- list(a = list(b = list(c = list(d = 1))))

#### Answers
#### 1
a[[1]]
```

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

```
a[["orange"]]
```

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

```
#### 2

b[["a"]][["b"]][["c"]][["d"]]
```

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

Note that `$` is a very useful subsetting syntex. One significant difference between `$` and `[[` is that `$` does partial matching while `[[` does not:

```
a$orange
```

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

```
a$o
```

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

```
a[["o"]]
```

```
## NULL
```

2.5 Data frame

A data frame is a special kind of list. It is a list of equal-length vectors, therefore appears as a 2-D structure. Microsoft Excel users find this very familiar. It is the most common way of storing data in **R**. A data frame shares properties of both the matrix and the list. Functions that work on matrices also work on data frames: `names`, `colnames`, and `rownames`, although `names` and `colnames` are the same thing. The `length` of a data frame is the length of the underlying list and so is the same as `ncol`; `nrow` gives the number of rows.

The simplest way to initialize a data frame is the function `data.frame`. Let us assume that you have several vectors of the same type and length. Now, you want to assemble them into a data frame using `data.frame`. Note how the column names are assigned.

```
v1 <- sequence(5)
v2 <- v1 + 2
v3 <- v2 * 3
f1 <- seq(from=100, by=1.5, length=5)
f2 <- v1+v2+v3+f1

df <- data.frame(v1=v1, v2=v2, v3=v3, f1=f1, f2=f2)
df
```

```
##   v1 v2 v3    f1    f2
## 1  1  3  9 100.0 113.0
## 2  2  4 12 101.5 119.5
## 3  3  5 15 103.0 126.0
## 4  4  6 18 104.5 132.5
## 5  5  7 21 106.0 139.0
```

The result above shows you that a data frame is a collection of columns. So, if your data is already organized into columns, then it's easy to build a data frame. If your data is in a list that contains vectors and/or factors, then use `as.data.frame` instead.

```
gender <- factor(c("female", "male", "female", "male", "female"))
listA <- list(score1=v1, score2=f1, gender=gender)
df <- as.data.frame(listA)
str(df)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ score1: int  1 2 3 4 5
##  $ score2: num  100 102 103 104 106
##  $ gender: Factor w/ 2 levels "female","male": 1 2 1 2 1
```

To append rows to the above data frame, for example, after a set of new observation has arrived, you need to create a second data frame containing the new rows and use `rbind` to append it to the original data frame.

```
newobs <- data.frame(score1 = 11, score2 = 106.1, gender = "female")
df <- rbind(df, newobs)
str(df)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ score1: num  1 2 3 4 5 11
##  $ score2: num  100 102 103 104 106 ...
##  $ gender: Factor w/ 2 levels "female","male": 1 2 1 2 1 1
```

Question 8: Please find out how to use `cbind` to add a new variable (a new column) of `avail <- c("Y","Y","N","Y","N","Y")` to the original `df` to form a new `df`. (Hint, you need to make `avail` a data frame and give the proper column name "availability").

```
avail <- c("Y","Y","N","Y","N","Y")

#### Answer
df <- cbind(df, data.frame(availability = avail))
str(df)
```

```
## 'data.frame':    6 obs. of  4 variables:
##  $ score1      : num  1 2 3 4 5 11
##  $ score2      : num  100 102 103 104 106 ...
##  $ gender      : Factor w/ 2 levels "female","male": 1 2 1 2 1 1
##  $ availability: Factor w/ 2 levels "N","Y": 2 2 1 2 1 2
```

2.5.1 Subsetting data frames

Data frames possess the characteristics of both lists and matrices. When it comes to subsetting, they behave like lists if you subset with a single vector; they behave like matrices if you subset with two vectors.

```
df[df$v1 == 2, ]
```

```
## [1] score1       score2       gender       availability
## <0 rows> (or 0-length row.names)
```

```
df[c(1, 3), ]
```

```
##   score1 score2 gender availability
## 1      1    100 female            Y
## 3      3    103 female            N
```

There are two ways to select columns from a data frame. Note the structural difference when stripping down to single vectors.

```
#### Like a list:
df[c("score1", "gender")]
```

```
##   score1 gender
## 1      1 female
## 2      2   male
## 3      3 female
## 4      4   male
## 5      5 female
## 6     11 female
```

```
#### Like a matrix
df[, c("score1", "gender")]
```

```
##   score1 gender
## 1      1 female
## 2      2   male
## 3      3 female
## 4      4   male
## 5      5 female
## 6     11 female
```

```
#### Like a list for single variable, note how the subset remains a data frame (or a list).
df1 <- df["availability"]
str(df1)
```

```
## 'data.frame':    6 obs. of  1 variable:
##  $ availability: Factor w/ 2 levels "N","Y": 2 2 1 2 1 2
```

```
#### Like a matrix for a single variable, note how it simplifies the output.
df2 <- df$availability
df3 <- df[,"availability"]
str(df2)
```

```
##  Factor w/ 2 levels "N","Y": 2 2 1 2 1 2
```

```
str(df3)
```

```
##  Factor w/ 2 levels "N","Y": 2 2 1 2 1 2
```

The built-in dataset `airquality` has daily air quality measurements in New York, May to September 1973. It is a data frame with 154 observations of 6 variables:

- `Ozone` numeric Ozone (ppb)
- `Solar.R` numeric Solar R (lang)
- `Wind` numeric Wind (mph)
- `Temp` numeric Temperature (degrees F)
- `Month` numeric Month (1–12)
- `Day` numeric Day of month (1–31)

Question 9: Please fix each of the following common data frame subsetting errors:

1. To extract rows where the Month is May: `airquality[airquality$Month = 5, ]` (Hint: logical operator issue)
2. To drop first row and extract rows 2 through 4: `airquality[-1:4, ]` (Hint: negative integers omit elements at the specified positions)
3. To extract rows where Month is May, June, or July: `airquality[airquality$Month <= 7]` (Hint: the command is for extracting columns; how to extract rows?)
4. To extract rows where Month is either May or June: `airquality[airquality$Month == 5 | 6, ]` (Hint: find out how to use `%in%`)

```
#### Answers

#### (1) airquality[airquality$Month = 5, ]

airquality[airquality$Month == 5, ]
```

```
##    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
## 7     23     299  8.6   65     5   7
## 8     19      99 13.8   59     5   8
## 9      8      19 20.1   61     5   9
## 10    NA     194  8.6   69     5  10
## 11     7      NA  6.9   74     5  11
## 12    16     256  9.7   69     5  12
## 13    11     290  9.2   66     5  13
## 14    14     274 10.9   68     5  14
## 15    18      65 13.2   58     5  15
## 16    14     334 11.5   64     5  16
## 17    34     307 12.0   66     5  17
## 18     6      78 18.4   57     5  18
## 19    30     322 11.5   68     5  19
## 20    11      44  9.7   62     5  20
## 21     1       8  9.7   59     5  21
## 22    11     320 16.6   73     5  22
## 23     4      25  9.7   61     5  23
## 24    32      92 12.0   61     5  24
## 25    NA      66 16.6   57     5  25
## 26    NA     266 14.9   58     5  26
## 27    NA      NA  8.0   57     5  27
## 28    23      13 12.0   67     5  28
## 29    45     252 14.9   81     5  29
## 30   115     223  5.7   79     5  30
## 31    37     279  7.4   76     5  31
```

```
#### (2) airquality[-1:4, ]
airquality[2:4,]
```

```
##   Ozone Solar.R Wind Temp Month Day
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
```

```
#### (3) airquality[airquality$Month <= 7]

airquality[airquality$Month <= 7, ]
```

```
##    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
## 7     23     299  8.6   65     5   7
## 8     19      99 13.8   59     5   8
## 9      8      19 20.1   61     5   9
## 10    NA     194  8.6   69     5  10
## 11     7      NA  6.9   74     5  11
## 12    16     256  9.7   69     5  12
## 13    11     290  9.2   66     5  13
## 14    14     274 10.9   68     5  14
## 15    18      65 13.2   58     5  15
## 16    14     334 11.5   64     5  16
## 17    34     307 12.0   66     5  17
## 18     6      78 18.4   57     5  18
## 19    30     322 11.5   68     5  19
## 20    11      44  9.7   62     5  20
## 21     1       8  9.7   59     5  21
## 22    11     320 16.6   73     5  22
## 23     4      25  9.7   61     5  23
## 24    32      92 12.0   61     5  24
## 25    NA      66 16.6   57     5  25
## 26    NA     266 14.9   58     5  26
## 27    NA      NA  8.0   57     5  27
## 28    23      13 12.0   67     5  28
## 29    45     252 14.9   81     5  29
## 30   115     223  5.7   79     5  30
## 31    37     279  7.4   76     5  31
## 32    NA     286  8.6   78     6   1
## 33    NA     287  9.7   74     6   2
## 34    NA     242 16.1   67     6   3
## 35    NA     186  9.2   84     6   4
## 36    NA     220  8.6   85     6   5
## 37    NA     264 14.3   79     6   6
## 38    29     127  9.7   82     6   7
## 39    NA     273  6.9   87     6   8
## 40    71     291 13.8   90     6   9
## 41    39     323 11.5   87     6  10
## 42    NA     259 10.9   93     6  11
## 43    NA     250  9.2   92     6  12
## 44    23     148  8.0   82     6  13
## 45    NA     332 13.8   80     6  14
## 46    NA     322 11.5   79     6  15
## 47    21     191 14.9   77     6  16
## 48    37     284 20.7   72     6  17
## 49    20      37  9.2   65     6  18
## 50    12     120 11.5   73     6  19
## 51    13     137 10.3   76     6  20
## 52    NA     150  6.3   77     6  21
## 53    NA      59  1.7   76     6  22
## 54    NA      91  4.6   76     6  23
## 55    NA     250  6.3   76     6  24
## 56    NA     135  8.0   75     6  25
```

```
## 57    NA     127  8.0   78     6  26
## 58    NA      47 10.3   73     6  27
## 59    NA      98 11.5   80     6  28
## 60    NA      31 14.9   77     6  29
## 61    NA     138  8.0   83     6  30
## 62   135     269  4.1   84     7   1
## 63    49     248  9.2   85     7   2
## 64    32     236  9.2   81     7   3
## 65    NA     101 10.9   84     7   4
## 66    64     175  4.6   83     7   5
## 67    40     314 10.9   83     7   6
## 68    77     276  5.1   88     7   7
## 69    97     267  6.3   92     7   8
## 70    97     272  5.7   92     7   9
## 71    85     175  7.4   89     7  10
## 72    NA     139  8.6   82     7  11
## 73    10     264 14.3   73     7  12
## 74    27     175 14.9   81     7  13
## 75    NA     291 14.9   91     7  14
## 76     7      48 14.3   80     7  15
## 77    48     260  6.9   81     7  16
## 78    35     274 10.3   82     7  17
## 79    61     285  6.3   84     7  18
## 80    79     187  5.1   87     7  19
## 81    63     220 11.5   85     7  20
## 82    16       7  6.9   74     7  21
## 83    NA     258  9.7   81     7  22
## 84    NA     295 11.5   82     7  23
## 85    80     294  8.6   86     7  24
## 86   108     223  8.0   85     7  25
## 87    20      81  8.6   82     7  26
## 88    52      82 12.0   86     7  27
## 89    82     213  7.4   88     7  28
## 90    50     275  7.4   86     7  29
## 91    64     253  7.4   83     7  30
## 92    59     254  9.2   81     7  31
```

```
#### (4) airquality[airquality$Month == 5 | 7, ]
mymonth=c(5,7)
airquality[airquality$Month %in% mymonth, ]
```

```
##    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
## 7     23     299  8.6   65     5   7
## 8     19      99 13.8   59     5   8
## 9      8      19 20.1   61     5   9
## 10    NA     194  8.6   69     5  10
## 11     7      NA  6.9   74     5  11
## 12    16     256  9.7   69     5  12
## 13    11     290  9.2   66     5  13
## 14    14     274 10.9   68     5  14
## 15    18      65 13.2   58     5  15
## 16    14     334 11.5   64     5  16
## 17    34     307 12.0   66     5  17
## 18     6      78 18.4   57     5  18
## 19    30     322 11.5   68     5  19
## 20    11      44  9.7   62     5  20
## 21     1       8  9.7   59     5  21
## 22    11     320 16.6   73     5  22
## 23     4      25  9.7   61     5  23
## 24    32      92 12.0   61     5  24
## 25    NA      66 16.6   57     5  25
## 26    NA     266 14.9   58     5  26
## 27    NA      NA  8.0   57     5  27
## 28    23      13 12.0   67     5  28
## 29    45     252 14.9   81     5  29
## 30   115     223  5.7   79     5  30
## 31    37     279  7.4   76     5  31
## 62   135     269  4.1   84     7   1
## 63    49     248  9.2   85     7   2
## 64    32     236  9.2   81     7   3
## 65    NA     101 10.9   84     7   4
## 66    64     175  4.6   83     7   5
## 67    40     314 10.9   83     7   6
## 68    77     276  5.1   88     7   7
## 69    97     267  6.3   92     7   8
## 70    97     272  5.7   92     7   9
## 71    85     175  7.4   89     7  10
## 72    NA     139  8.6   82     7  11
## 73    10     264 14.3   73     7  12
## 74    27     175 14.9   81     7  13
## 75    NA     291 14.9   91     7  14
## 76     7      48 14.3   80     7  15
## 77    48     260  6.9   81     7  16
## 78    35     274 10.3   82     7  17
## 79    61     285  6.3   84     7  18
## 80    79     187  5.1   87     7  19
## 81    63     220 11.5   85     7  20
## 82    16       7  6.9   74     7  21
## 83    NA     258  9.7   81     7  22
## 84    NA     295 11.5   82     7  23
## 85    80     294  8.6   86     7  24
## 86   108     223  8.0   85     7  25
```

```
## 87    20      81  8.6   82     7  26
## 88    52      82 12.0   86     7  27
## 89    82     213  7.4   88     7  28
## 90    50     275  7.4   86     7  29
## 91    64     253  7.4   83     7  30
## 92    59     254  9.2   81     7  31
```