Note 15. Classes

Introduction to Statistical Programming

Chanmin Kim

Department of Statistics
Sungkyunkwan University

2022 Spring

Class & Method

- Everything in R is an object.
- Class: Definition of an object.
- Method: A function that performs specific calculations on objects of a specific class.
- Generic function:
  - A function with a collection of methods.
  - A generic function is used to determine the class of its arguments and select the appropriate method.

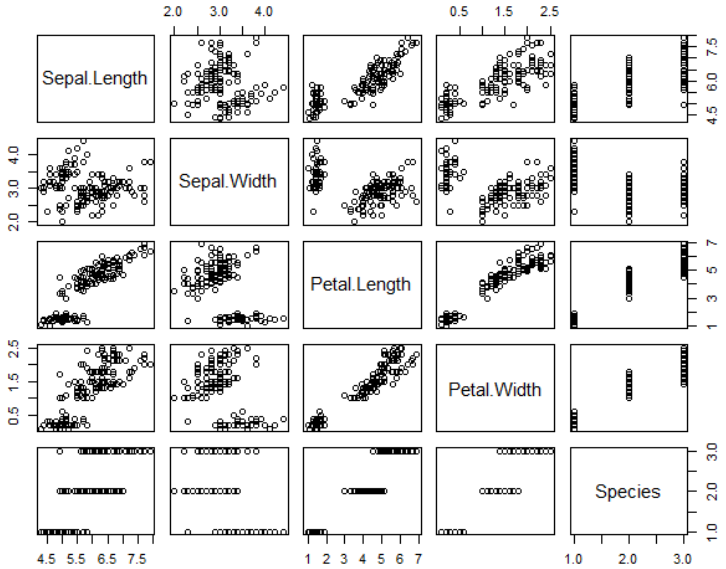
Example

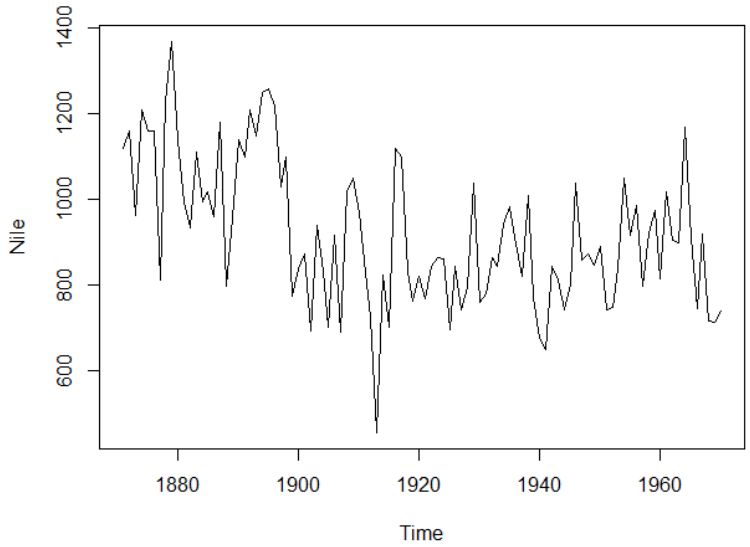
- Consider `iris` and `Nile` datasets.
- `plot`: A generic function; It performs differently for different classes.
- `methods(`*`class=' '`*`)`: It shows a list of functions for the specified class.

```
> class(iris)
[1] "data.frame"

> class(Nile)
[1] "ts"
```

Example

```
> plot(iris)
> plot(Nile)
```





Example

```
> methods(class='data.frame')
 [1] $              $<-            [              [[
 [5] [[<-           [<-            aggregate      anyDuplicated
 [9] as.data.frame  as.list        as.matrix      by
[13] cbind          coerce         dim            dimnames
[17] dimnames<-     droplevels     duplicated     edit
[21] format         formula        head           initialize
[25] is.na          Math           merge          na.exclude
[29] na.omit        Ops            plot           print
[33] prompt         rbind          row.names      row.names<-
.....

> methods(class='ts')
 [1] [              [<-            aggregate      as.data.frame
 [5] cbind          coerce         cycle          diff
 [9] diffinv        initialize     kernapply      lines
[13] Math           Math2          monthplot      na.omit
[17] Ops            plot           print          show
.....
```

S3 & S4 Classes

- There are two types of classes in R.
  - S3 Class: Old style, quick and dirty, informal.
  - S4 Class: New style, rigorous and formal.

- S3 approach:
  - Very simple to implement; Just add a class name to an object.
  - There are no formal requirements about the contents of the object, we just expect the object to have the right information.

- S4 approach:
  - It gives a rigorous definition of an object.
  - Any valid object of an S4 class will meet all the requirements specified in the definition.

- Despite the shortcomings of S3 classes they are widely used and unlikely to disappear.

Why Class & Method

- S4 classes reduce errors. When a function acts on a S4 class object it knows what type of information is in the object and that the information is valid.
- Methods simplify function calls and make computations more natural and clearer.
- For generic functions with methods we no longer need a separate function for each class of object.

S3 Class & Method

- To create an S3 class all we need to do is set the class attribute of the object using `class()`.
- To create an S3 method write a function with the name *generic.class*, where *generic* is a generic function name and *class* is the corresponding class for the method.
- Examples of generic functions are `summary()`, `print()` and `plot()`. See `?UseMethod` for how to create new generic functions for S3 methods.

S3 Class & Method

Functions related to S3 class & method:

Function	Description
`class(x)`	Get or set the class attributes of `x`.
`unclass(x)`	Remove class attributes of `x`.
`methods(generic.function)`	All available S3 methods for a generic function.
`methods(class='class')`	Get all S3 methods for a particular class.
`is(object)`	Return `object`'s class & all super-classes.
`is(object,class)`	Test if `object` is from `class`.
`str(object)`	Display the internal structure of an `object`.

Example

```
> # A function to create a class object student
> student = function(ID, sex, age, ht, wt)
+ {
+   out = list(ID=ID, sex=sex, data=data.frame(Age=age, HT.cm=ht,
+              WT.kg=wt))
+   class(out) = 'student'
+   invisible(out)
+ }

> # Print method for student class
> print.student = function(object)
+ {
+   cat('ID =', object$ID, '\nSex =', object$sex, '\n')
+   print(object$data)
+ }
```

Example

```
> # Plot method for student class
> plot.student = function(object)
+ {
+   data = object$data
+   par(mfrow=c(1,2))
+   plot(data$Age, data$HT.cm, type="o", pch=19, col="blue",
+        xlab="Age (months)", ylab="Height (cm)",main="Height vs Age")
+   plot(data$Age, data$WT.kg, type="o", pch=19, col="blue",
+        xlab="Age (months)", ylab="Weight (kg)",main="Weight vs Age")
+   mtext(paste("Subject ",object$ID,", ",toupper(object$sex),sep=""),
+         side=3, outer=TRUE, line=-1.5, cex=1.5)
+ }

> # student data
> age = c(8,10,12,14,16,18)
> male.wt = c(27.8,35.5,45.5,55.4,62.4,65.8)
> female.wt = c(26.9,34.7,43.8,50.9,53.6,54.1)
> male.ht = c(129.1,139.4,151.8,165.5,171.8,173.4)
> female.ht = c(127.8,139.9,152.7,158.5,160.0,160.7)
```

Example

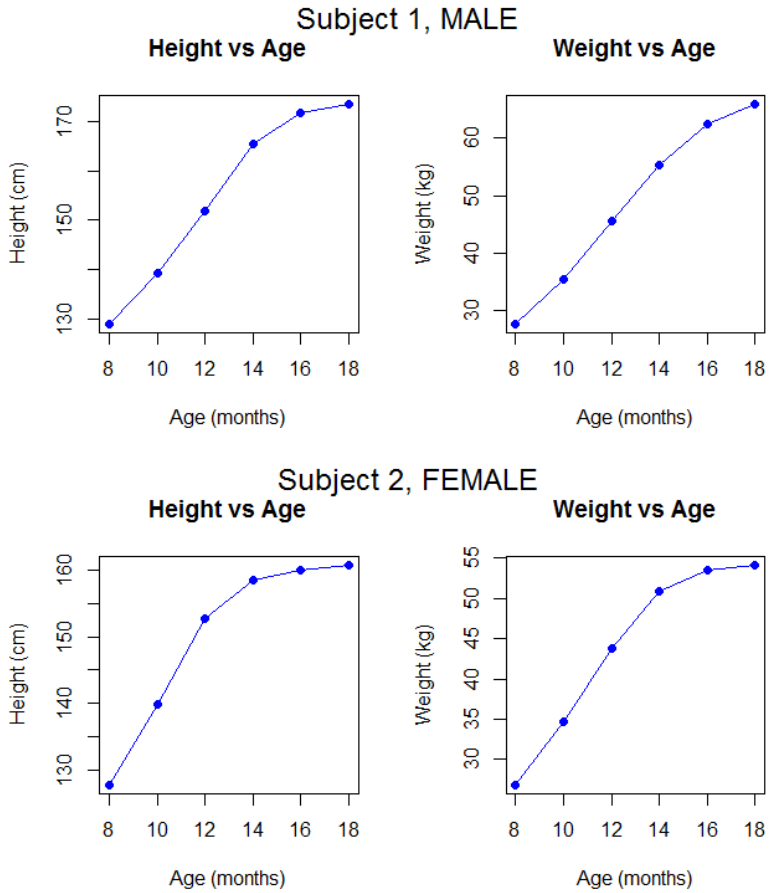
```
> # Create student objects
> x = student(1, 'male', age, male.ht, male.wt)
> y = student(2, 'female', age, female.ht, female.wt)
> class(x)
[1] "student"
> class(y)
[1] "student"

> # Print student objects
> x
ID = 1
Sex = male
  Age HT.cm WT.kg
1   8 129.1  27.8
2  10 139.4  35.5
3  12 151.8  45.5
4  14 165.5  55.4
5  16 171.8  62.4
6  18 173.4  65.8
```

Example

```
> y
ID = 2
Sex = female
  Age HT.cm WT.kg
1   8 127.8  26.9
2  10 139.9  34.7
3  12 152.7  43.8
4  14 158.5  50.9
5  16 160.0  53.6
6  18 160.7  54.1
```

Example

```
> # Plot student objects
> plot(x)
> plot(y)
```



Example

```
> # Functions related to S3 class & method
> methods(class='student')
[1] plot  print

> methods(plot)
 [1] plot.acf*           plot.data.frame*    plot.decomposed.ts*
 [4] plot.default        plot.dendrogram*    plot.density*
 [7] plot.ecdf           plot.factor*        plot.formula*
[10] plot.function       plot.hclust*        plot.histogram*
[13] plot.HoltWinters*   plot.infant         plot.isoreg*
[16] plot.lm*            plot.medpolish*     plot.mlm*
[19] plot.person         plot.ppr*           plot.prcomp*
[22] plot.princomp*      plot.profile.nls*   plot.raster*
[25] plot.spec*          plot.stepfun        plot.stl*
[28] plot.student        plot.table*         plot.ts
[31] plot.tskernel*      plot.TukeyHSD*
```

Example

```
> is(x)
[1] "student"
> is(x,'student')
[1] TRUE
> is(x,'ts')
[1] FALSE

> str(x)
List of 3
 $ ID  : num 1
 $ sex : chr "male"
 $ data:'data.frame': 6 obs. of  3 variables:
  ..$ Age  : num [1:6] 8 10 12 14 16 18
  ..$ HT.cm: num [1:6] 129 139 152 166 172 ...
  ..$ WT.kg: num [1:6] 27.8 35.5 45.5 55.4 62.4 65.8
 - attr(*, "class")= chr "student"
```

S4 Class & Method

- S4 classes and methods are created using functions in the methods package, this package is loaded automatically when R is started.
- Information in S4 classes is organized into slots.
- Each slot is named and requires a specified class.
- Slots are one of the advantages of S4 classes. The class of the data in the slot must match the class corresponding to the slot.
- For example, it is not possible to have numeric data in a slot that is designated for character data.
- This is not the case with S3 classes. In S3, we assume the class is right.

S4 Class & Method

Functions related to S4 class & method:

Function	Description
`setClass()`	Create a new class.
`setMethod()`	Create a new method.
`setGeneric()`	Create a new generic function.
`new()`	Generate a new object for a given class.
`getClass()`	Get the class definition.
`getMethod()`	Get the method definition.
`getSlots()`	Get the name and class of each slot.
`@`	Get or replace the contents of a slot.
`validObject()`	Test the validity of an object.

Create an S4 Class & Method / Validity

- `setClass(`*`class, representation`*`)`:
  - *class*: Name of the class to be created.
  - *representation*: Named list of the slots and the corresponding classes.

- Use `new()` to generate a new object from a class.

- `setMethod(`*`f, signature, definition`*`)`:
  - *f*: Name of the generic function.
  - *signature*: Character string of the corresponding class.
  - *definition*: Function definition.

- Validity of an object:
  - Slots help, but there maybe additional requirements.
  - To test for these requirements supply a validity checking method to the validity argument of `setClass()`.
  - This method should return TRUE if the object is valid

Example

```
> # Define a student1 class
> setClass('student1', representation(ID='numeric',
+          sex='character', data='data.frame'))

> getClass('student1')
Class "student1" [in ".GlobalEnv"]

Slots:

Name:          ID         sex        data
Class:    numeric   character data.frame

> getSlots('student1')
          ID          sex         data
   "numeric"  "character" "data.frame"
```

Example

```
> # Create an object of class student1
> x = new('student1', data=data.frame(age, male.wt, male.ht),
+         sex="male", ID=1)
> x@data
  age male.wt male.ht
1   8    27.8   129.1
2  10    35.5   139.4
3  12    45.5   151.8
4  14    55.4   165.5
5  16    62.4   171.8
6  18    65.8   173.4

> y = new('student1', data=data.frame(age, male.wt, male.ht),
+         sex=1, ID="001")
Error in validObject(.Object) :
  invalid class \student" object: 1: invalid object for slot "ID"
  in class "student1": got class "character", should be or extend
  class "numeric"

> # ID should be numeric.
```

Example

```
> # Show method, similar to print for S3 classes
> setMethod(f='show', signature='student1',
+           definition = function(object) {
+             cat("ID =", object@ID, "\nSex =", object@sex, "\n")
+             print(object@data)
+           })
[1] "show"

> show(x)
ID = 1
Sex = male
  age male.wt male.ht
1   8    25.0   130.0
2  10    35.5   139.4
3  12    45.5   151.8
4  14    55.4   165.5
5  16    62.4   171.8
6  18    65.8   173.4
```

Example

```
> # Extract method
> x[1:2, ]
Error in x[1:2, ] : object of type 'S4' is not subsettable
> x@data[1:2,]
  age male.wt male.ht
1   8    27.8   129.1
2  10    35.5   139.4

> setMethod(f='[', signature='student1',
+           definition=function(x,i,j) {x@data[i,j]})
[1] "["

> # Replace method
> x[1,] = c(8, 25, 130)
Error in x[1, ] = c(8, 25, 130) : object of type 'S4' is
not subsettable
```

Example

```
> setMethod(f = '[<-', signature = 'student1',
+           definition = function(x,i,j,value) {
+             x@data[i,j] = value;  validObject(x)
+             return(x)
+           })
[1] "[<-"

> x[1,] = c(8, 25, 130);  x
ID = 1
Sex = male
  age male.wt male.ht
1   8    25.0   130.0
2  10    35.5   139.4
3  12    45.5   151.8
4  14    55.4   165.5
5  16    62.4   171.8
6  18    65.8   173.4

> methods(class='student1')
[1] [    [<-  show
```

Example

```
> # Check the validity of an object from student1
> validity.student1 = function(object)
+ {
+   if(!all(sapply(object@data, is.numeric)))
+   {
+     return('data not numeric')
+   } else return(TRUE)
+ }

> setClass('student1', representation(ID= 'numeric',
+          sex= 'character', data= 'data.frame'),
+          validity=validity.student1)
>
> x = new('student1', data=data.frame(age, male.wt, male.ht),
+         sex='male', ID=1)

> z = new('student1', data=data.frame(c('0year'),c(49.99),c(3.53)),
+         sex="male", ID=1)
Error in validObject(.Object) :
  invalid class \student1" object: data not numeric
```

Inheritance of S4 Class

- Possible to create a new class that extends the first class. That is, create a new class that contains all of the information from the existing class, plus additional slots.
- Methods defined for the contained class can also be used for the new class.
- Validity requirements of the contained class also apply to the new class.
- Use the argument contains in `setClass()` to set a superclass. Set *contains = 'name of class being extend'*.

Example

```
> # 'student1.Order' is a subclass of 'student1'.
> setClass('student1.Order',
+          representation(Order='numeric'),
+          contains='student1')

> setMethod(f='show', signature ='student1.Order',
+           definition = function(object) {
+             cat('ID =', object@ID, '\nSex =',
+                 object@sex,'\nOrder =',object@Order,'\n')
+           print(object@data)})
[1] "show"

> x.more = new('student1.Order', Order=1,
+              data=data.frame(age,male.wt,male.ht),
+              sex="male", ID=1)
```

Example

```
> x.more
ID = 1
Sex = male
Order = 1
  age male.wt male.ht
1   8    27.8   129.1
2  10    35.5   139.4
3  12    45.5   151.8
4  14    55.4   165.5
5  16    62.4   171.8
6  18    65.8   173.4

> class(x.more)
[1] "student1.Order"
attr(,"package")
[1] ".GlobalEnv"
```

Example

```
> is(x.more)
[1] "student1.Order" "student1"

> x.more[1:2,]
  age male.wt male.ht
1   8    27.8   129.1
2  10    35.5   139.4

> # Although we did NOT define '[' function for 'student1.Order'
> # class, it is still available because 'student1' class is
> # the super class of 'student1.Order'.
```