STAT40830
Adv Data Programming with R (online)

Topic 5: Review on Functions and OOP

Isabella Gollini

Overview

1. Functions
2. Object Oriented Programming 2.1 Methods

1. Functions

1. Function components[1]

- A function has three parts:
  1. The `formals()`, the list of arguments that control how you call the function.
  2. The `body()`, the code inside the function.
  3. The `environment()`, the data structure that determines how the function finds the values associated with the names.

1. Function components

```
fun2_for_setup <- function(X){ # similar to fun1
  Z <- X # now Z is setup to be the same as X
  # do a loop taking each column separately
  for(j in 1:ncol(X)){
    # put the result in column j
    Z[,j] <- (X[,j] - mean(X[,j])) / sd(X[,j])
  }
  return(Z)
}
```

1. Function components

```
1 formals(fun2_for_setup)
```

```
$X
```

```
1 body(fun2_for_setup)
```

```
{
    Z <- X
    for (j in 1:ncol(X)) {
        Z[, j] <- (X[, j] - mean(X[, j]))/sd(X[, j])
    }
    return(Z)
}
```

```
1 environment(fun2_for_setup)
```

```
<environment: R_GlobalEnv>
```

1. Function components

```
1 formals(sum)
```

```
NULL
```

```
1 body(sum)
```

```
NULL
```

```
1 environment(sum)
```

```
NULL
```

```
1 sum
```

```
function (..., na.rm = FALSE)  .Primitive("sum")
```

- There is one exception to the rule that a function has three components. **Primitive functions**, like `sum()` and `[`, call C code directly.
- Notice that `.Primitive` is different from `.Internal`:
  - `.Primitive` functions are built in functions (from base R) calling C.
  - All the other functions call C using `.Internal`.[1]

1. Type of functions

```
1 typeof(sum)
```

```
[1] "builtin"
```

```
1 typeof(`[`)
```

```
[1] "special"
```

```
1 typeof(fun2_for_setup)
```

```
[1] "closure"
```

```
1 typeof(ggplot2::ggplot)
```

```
[1] "closure"
```

- *"special"* and *"builtin"* are for primitive functions and operators
- *"closure"* is for any function not from base R

1. Function components

```
1 formals(ggplot2::ggplot)
```

```
$data
NULL

$mapping
aes()

$...


$environment
parent.frame()
```

```
1 body(ggplot2::ggplot)
```

```
{
    UseMethod("ggplot")
}
```

```
1 environment(ggplot2::ggplot)
```

```
<environment: namespace:ggplot2>
```

2. Object Oriented Programming

2. Object Oriented Programming

- Everything we use in R (functions, vectors, data frames, etc) is an **object**.
- There are several important features of OOP that R promotes:
  - **Encapsulation** - data items are packaged into one class instance and makes keeping track of everything easier.
  - **Polymorphism** - the same function will have different actions on different classes.
  - **Inheritance** - objects of one class can inherit characteristics of another class.

2. Classes

- There are 3 classes provided in base R: **S3**, **S4**, RC and some other classes from contributed packages, the most commonly used is the class **R6** from the `R6` package. Today we'll focus on S3 class only.
- An S3 class consists of a **list** with a **class name** attribute (e.g. `glm`) and **dispatch capability**.
- The **dispatch capability** means that you can make use of generic functions (e.g. `print` or `summary`).

2. Generic functions

- We have met a number of functions that work on vectors, lists, matrices, etc: `print`, `plot`, `summary`.
- These are known as **generic functions**, in that they act differently for different types of objects given to them.
- We'll focus on **S3** classes and methods.
- To learn more about all classes in R look at the Object Oriented Programming chapter in Advanced R

2.1 Methods

2.1 the function `lm`

Let's start with a simple example:

```
1  res <- lm(Sepal.Length ~ Petal.Length + Sepal.Width, iris[1:5,])
2  res
```

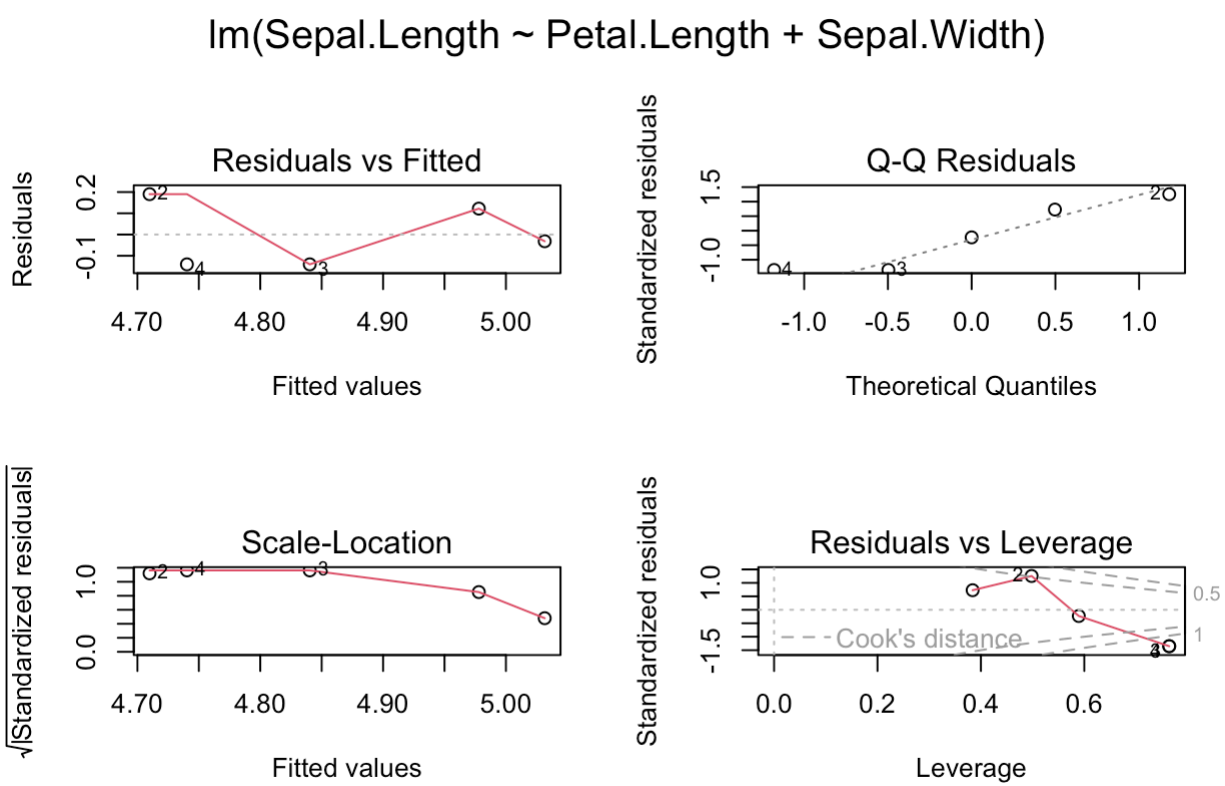
```
Call:
lm(formula = Sepal.Length ~ Petal.Length + Sepal.Width, data = iris[1:5,
    ])

Coefficients:
 (Intercept)  Petal.Length   Sepal.Width
      3.4262       -0.2319        0.5361
```

- We all now that `lm` is used to fit linear models.
- When we print the object it's just showing the model and the coefficient.

2.1 the function `lm`

But when we plot the object it's showing more information:

```
1  plot(res, ask = FALSE)
```



- and the plot are not the default scatterplot!

2.1 the function `lm`

If we use the function `summary()` we see even information

```
1 summary(res)
```

```
Call:
lm(formula = Sepal.Length ~ Petal.Length + Sepal.Width, data = iris[1:5,
    ])

Residuals:
       1        2        3        4        5
 0.12205  0.19011 -0.14030 -0.14030 -0.03156

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    3.4262     2.6909   1.273    0.331
Petal.Length  -0.2319     1.5260  -0.152    0.893
Sepal.Width    0.5361     0.4169   1.286    0.327

Residual standard error: 0.2138 on 2 degrees of freedom
Multiple R-squared:  0.4686,    Adjusted R-squared:  -0.06287
F-statistic: 0.8817 on 2 and 2 DF,  p-value: 0.5314
```

2.1 the function `lm`

Let's have a look at the function:

```
lm # let's have a look at the function lm
```

```
function (formula, data, subset, weights, na.action, method = "qr",
    model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
    contrasts = NULL, offset, ...)
{
    ret.x <- x
    ret.y <- y
    cl <- match.call()
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data", "subset", "weights", "na.action",
        "offset"), names(mf), 0L)
```

- We can see that in line 56 they assigned `class(z) <- c(if (mlm) "mlm", "lm")` where `z` is the object storing the output
- Notice that objects can have more than one class. The generic functions will then look for methods by looping through the classes. This is called *inheritance*.

2.1 the function `lm`

Let's see the output if we `unclass` the object:

```
1 unclass(res)
```

```
$coefficients
 (Intercept) Petal.Length  Sepal.Width
   3.4262357   -0.2319392    0.5361217

$residuals
          1           2           3           4           5
 0.12205323  0.19011407 -0.14030418 -0.14030418 -0.03155894

$effects
 (Intercept) Petal.Length  Sepal.Width
-10.86729037  -0.07071068  -0.27494210  -0.29649795   0.05912538

$rank
[1] 3

$fitted.values
       1        2        3        4        5
4.977947 4.709886 4.840304 4.740304 5.031559

$assign
[1] 0 1 2
```

It's a list with a lot of data!

2.1 the `print()` method

So, we see two different output if we `print(res)` or we `print(unclass(res))`. Let's check the `print()`!

```
1 print
```

```
function (x, ...)
UseMethod("print")
<bytecode: 0x13bca8428>
<environment: namespace:base>
```

The body of the function is only one command! `UseMethod("print")`

- `print()` is using generic methods to print objects of a specific class!

2.1 `print.lm()`

Let's have a look at the `print()` method for an object of class `lm`:

```
1 print.lm # try to look at print.lm
```

```
Error in eval(expr, envir, enclos): object 'print.lm' not found
```

```
1 isS3method("print.lm")
```

```
[1] TRUE
```

```
1 getS3method("print", "lm") # get access to the print method for class lm
```

```
function (x, digits = max(3L, getOption("digits") - 3L), ...)
{
    cat("\nCall:\n", paste(deparse(x$call), sep = "\n", collapse = "\n"),
        "\n\n", sep = "")
    if (length(coef(x))) {
        cat("Coefficients:\n")
        print.default(format(coef(x), digits = digits), print.gap = 2L,
            quote = FALSE)
    }
    else cat("No coefficients\n")
    cat("\n")
    invisible(x)
}
<bytecode: 0x12af4e7c0>
<environment: namespace:stats>
```

2.1 `plot.lm()`

Let's have a look at the `plot()` method for an object of class `lm`:

```
plot.lm # try to look at plot.lm
```

```
Error in eval(expr, envir, enclos): object 'plot.lm' not found
```

```
isS3method("plot.lm")
```

```
[1] TRUE
```

```
# getS3method("print", "lm")  # get access to the plot method for class lm
stats:::plot.lm # or we can use ::: to access internal functions of a package
```

```
function (x, which = c(1, 2, 3, 5), caption = list("Residuals vs Fitted",
    "Q-Q Residuals", "Scale-Location", "Cook's distance", "Residuals vs Leverage",
    expression("Cook's dist vs Leverage* " * h[ii]/(1 - h[ii]))),
    panel = if (add.smooth) function(x, y, ...) panel.smooth(x,
        y, iter = iter.smooth, ...) else points, sub.caption = NULL,
    main = "", ask = prod(par("mfcol")) < length(which) && dev.interactive(),
    ..., id.n = 3, labels.id = names(residuals(x)), cex.id = 0.75,
    qqline = TRUE, cook.levels = c(0.5, 1), cook.col = 8, cook.lty = 2,
    cook.legendChanges = list(), add.smooth = getOption("add.smooth"),
    iter.smooth = if (isGlm) 0 else 3, label.pos = c(4, 2), cex.caption = 1,
```

2.1 Built in functions

Let's have a look at the `summary()` method for an object of class `lm`:

```
1 summary.lm # let's have a look at summary.lm
```

```
function (object, correlation = FALSE, symbolic.cor = FALSE,
    ...)
{
    z <- object
    p <- z$rank
    rdf <- z$df.residual
    if (p == 0) {
        r <- z$residuals
        n <- length(r)
        w <- z$weights
        if (is.null(w)) {
            rss <- sum(r^2)
        }
        else {
            rss <- sum(w * r^2)
            r <- sqrt(w) * r
        }
        resvar <- rss/rdf
        ans <- z[c("call", "terms", if (!is.null(z$weights)) "weights")]
        class(ans) <- "summary.lm"
        ans$aliased <- is.na(coef(object))
        ans$residuals <- r
```

- The output is of class `summary.lm`. You can check and see the S3 print method for that class: `stats:::print.summary.lm`.

2.1 Look at all methods for a class

```
1 methods(class = "lm")
```

```
 [1] add1             alias            anova            case.names       coerce
 [6] confint          cooks.distance   deviance         dfbeta           dfbetas
[11] drop1            dummy.coef       effects          extractAIC       family
[16] formula          hatvalues        influence        initialize       kappa
[21] labels           logLik           model.frame      model.matrix     nobs
[26] plot             predict          print            proj             qr
[31] residuals        rstandard        rstudent         show             simulate
[36] slotsFromS3      summary          variable.names   vcov
see '?methods' for accessing help and source code
```

```
1 methods(class = "summary.lm")
```

```
[1] print vcov
see '?methods' for accessing help and source code
```

2.1 Let's build our methods

Consider a function to standardise columns of a matrix or data.frame

```
my_fun <- function(X){ # function to standardise col. of a matrix or data.frame
  stopifnot(is.matrix(X) | apply(X, 2, is.numeric)) # check the input
  Z <- X # now Z is setup to be the same as X
  for(j in 1:ncol(X)){   # do a loop taking each column separately
    # put the result in column j
    Z[,j] <- (X[,j] - mean(X[,j])) / sd(X[,j])
  }
  class(Z) <- c("myclass", class(X))
  return(Z)
}
```

2.1 Let's build our methods

Let's test it:

```
set.seed(10)
n <- 5
X <- matrix(rnorm(n*2, 2, 2), n, 2)
Z <- my_fun(X)
Z
```

```
           [,1]       [,2]
[1,]  0.5967845  1.2494964
[2,]  0.2837753 -0.7414304
[3,] -1.5466124  0.3106817
[4,] -0.3559936 -1.2629964
[5,]  1.0220463  0.4442486
attr(,"class")
[1] "myclass" "matrix"  "array"
```

It's printing a matrix with the class as an attribute.

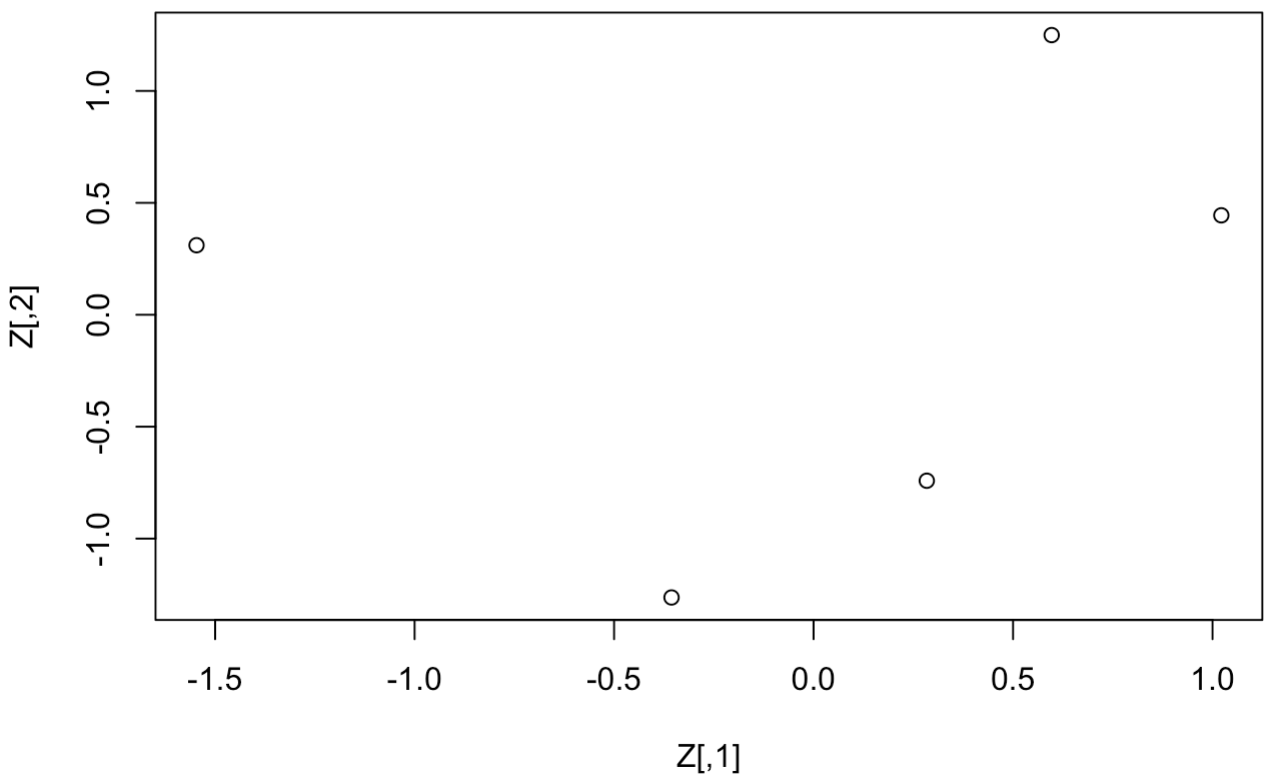
2.1 `print` method for `myclass`

```r
print.myclass <- function(x){
  cat("Standardised Object\n")
  obj <- x
  class(obj) <- class(x)[-1]
  print(obj)
  cat("\n")
  v <- apply(my_fun(x), 2, function(x) c(mean(x), sd(x)))
  rownames(v) <- c("mean", "variance")
  cat("Check Standardisation:\n")
  print(v)
  cat("\n")
  invisible(x)
}
Z <- my_fun(X)
Z
```

```
Standardised Object
           [,1]       [,2]
[1,]  0.5967845  1.2494964
[2,]  0.2837753 -0.7414304
[3,] -1.5466124  0.3106817
[4,] -0.3559936 -1.2629964
[5,]  1.0220463  0.4442486

Check Standardisation:
         [,1] [,2]
mean        0    0
variance    1    1
```

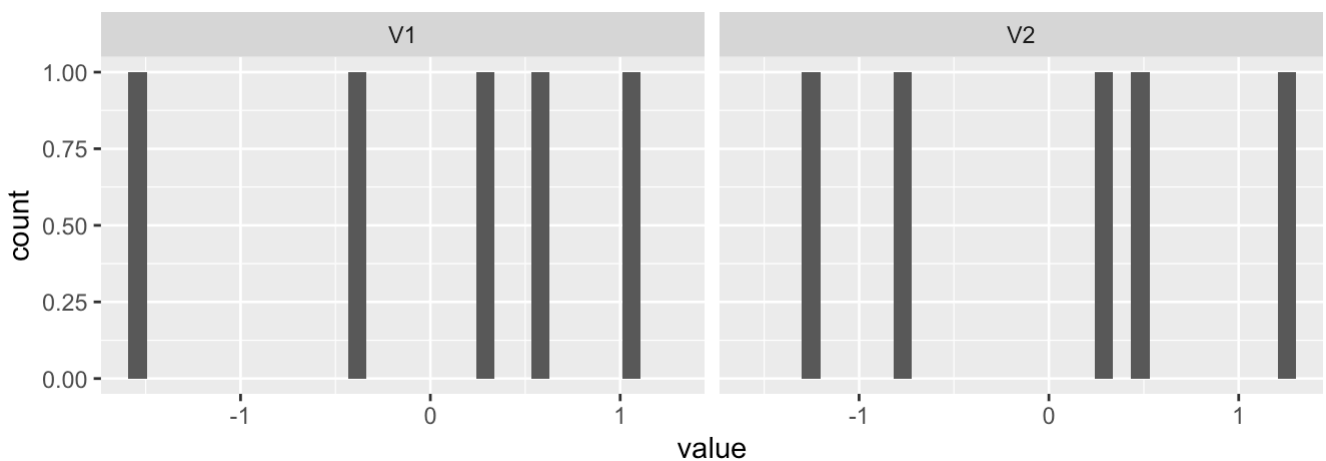
2.1 Let's check the plot

```
1 plot(Z)
```



2.1 `plot` method for `myclass`

```
plot.myclass <- function(x, ...){
  obj <- x
  if(!is.data.frame(obj)) obj <- as.data.frame(obj)
  obj <- tidyr::pivot_longer(obj, 1:ncol(obj))
  p <- ggplot2::ggplot(obj, ggplot2::aes(value)) +
    ggplot2::geom_histogram() +
    ggplot2::facet_wrap(~ name) +
    ggplot2::guides(color = "none")
  print(p)
  invisible(x)
}
plot(Z)
```



- See the *Using ggplot2 in packages* vignette from the `ggplot2` package.

2.1 `isS3method()`

Let's check if the function we created are S3 methods:

```
1 isS3method("print.myclass")
```

```
[1] TRUE
```

```
1 isS3method("plot.myclass")
```

```
[1] TRUE
```

Lessons from this lecture:

- Be careful if you used `.` in the name of a function, it may be confused with a method.
  - Even if there are a lot of functions in R using `.` see for example `as.data.frame()`, `read.csv()`, `t.test()`, etc.)
- To learn more see: Advanced R

Short Homework 2 (2% final grade)

- There are two marks available for this exercise.
- **On Brightspace you will find a discussion thread with all the details.**
- Pick an R package which you have not studied in the Data Programming courses and:
  - provide a link to the package on CRAN, and copy the title and description of the package.
  - write the name of a function in that package giving an output of a new class.
  - paste one method for that class.
- *You cannot use the same package already used in this thread by another student!*