R Classes

Paul E. Johnson[1] [2]

[1]Department of Political Science

[2]Center for Research Methods and Data Analysis, University of Kansas

2015

Outline

- This is a LaTeX version of the presentation R_classes-1, which was written with R markdown in 2015
- I concluded I hate writing with that, converted over to LaTeX.

Recipe

- Here's the basic R analysis recipe
  - estimator function creates data structures (not presentable to humans) and
  - declares that thing to be of a certain class.
- programmer also provides summary() and other methods that beautify the presentation and re-structure the data.
  - display methods: print, plot, etc
- Behind the scene, information is passed around in a relatively ugly format
- We show the users a prettied-up version.

Outline

Same sequence plays itself out across the R source code

1. Estimate a model
2. Estimate more models
3. Apply follow-up procedures and plotters

Same sequence plays itself out across the R source code

- User runs a model-making function, such as lm()

```
dat <- data.frame(y = rnorm(100), x1 = rnorm(100), x2 = rnorm(100))
m1 <- lm(y ~ x1 + x2, data = dat)
```

- R supplies follow-up procedures.We run

```
summary(m1)
anova(m1)
plot(m1)
termplot(m1)
nobs(m1)
```

- summary() is a "generic function"
- work is actually provided by summary.lm, a **method**.

method: a class-specific function that implements work promised by a generic function

Same sequence plays itself out across the R source code ...

- If there's no class-specific method, then the work is given to a "default" method, such as summary.default()

Want examples I can explain?

Look in rockchalk

- meanCenter, residualCenter,standardize.
  - all are functions that receive regressions, re-assign their classes.
- pctable

Generic Function

Definition: a function that receives a request and sends it to another function

A generic function like

```
summary(m1)
```

```
Call:
lm(formula = y ~ x1 + x2, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.13533 -0.87134  0.06965  0.53335  2.53370

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.13009    0.10616   1.225    0.223
x1          -0.04814    0.11359  -0.424    0.673
x2           0.02185    0.10720   0.204    0.839

Residual standard error: 1.056 on 97 degrees of freedom
Multiple R^2:  0.002176,  Adjusted R^2:  -0.0184
F-statistic: 0.1058 on 2 and 97 DF,  p-value: 0.8997
```

Generic Function ...

calls a method (an R function) named summary.lm

- "Dispatching" in S3. R runtime looks at the class of the FIRST input argument, and then sends the work to the other function.
- It's possible to create separate methods, as we see below
  - A formula method "pctable.formula()", so users can run pctable(y ~ x, data = dat)
  - A character method "pctable.character()", so users can run pctable("y", "x", data = dat)

List out Methods

- R has tools to see methods that are available
- list available print methods

```
methods(print)
```

```
 [1] print.acf*                                  print.anova*
                                        print.aov*
 [4] print.aovlist*                              print.ar*
                                     print.Arima*
 [7] print.arima0*                               print.AsIs
                                   print.aspell*
[10] print.aspell_inspect_context*               print.bibentry*
                                print.Bibtex*
[13] print.browseVignettes*                      print.by
                                     print.changedFiles*
[16] print.check_code_usage_in_package*          print.check_compiled_code*
                           print.check_demo_index*
[19] print.check_depdef*                         print.checkDocFiles*
                                print.checkDocStyle*
[22] print.check_dotInternal*                    print.checkFF*
                                  print.check_make_vars*
```

List out Methods ...

```
[25] print.check_nonAPI_calls*
     print.check_package_code_assign_to_globalenv*
     print.check_package_code_attach*
[28] print.check_package_code_data_into_globalenv*
     print.check_package_code_startup_functions*
     print.check_package_code_syntax*
[31] print.check_package_code_unload_functions*
     print.check_package_compact_datasets*
     print.check_package_CRAN_incoming*
[34] print.check_package_datasets*                    print.check_package_depends*
                        print.check_package_description*
[37] print.check_package_description_encoding*        print.check_package_license*
                        print.check_packages_in_dir*
[40] print.check_packages_in_dir_changes*             print.check_packages_used*
                         print.check_po_files*
[43] print.checkRd*                                   print.check_Rd_contents*
                           print.check_Rd_line_widths*
[46] print.check_Rd_metadata*                         print.check_Rd_xrefs*
                              print.checkReplaceFuns*
[49] print.checkS3methods*                            print.check_so_symbols*
                            print.check_T_and_F*
[52] print.checkTnF*                                  print.check_url_db*
                               print.check_vignette_index*
[55] print.checkVignettes*                            print.citation*
                                  print.codoc*
[58] print.codocClasses*                              print.codocData*
                                print.colorConverter*
```

List out Methods ...

```
[61] print.compactPDF*                  print.condition
                                print.connection
[64] print.data.frame                   print.Date
                                    print.default
[67] print.dendrogram*                  print.density*
                                print.difftime
[70] print.dist*                        print.Dlist
                                    print.DLLInfo
[73] print.DLLInfoList                  print.DLLRegisteredRoutines
                        print.dummy_coef*
[76] print.dummy_coef_list*             print.ecdf*
                                    print.factanal*
[79] print.factor                       print.family*
                                print.fileSnapshot*
[82] print.findLineNumResult*           print.formula*
                                print.ftable*
[85] print.function                     print.getAnywhere*
                            print.glm*
[88] print.hclust*                      print.help_files_with_topic*
                        print.hexmode
[91] print.HoltWinters*                 print.hsearch*
                                print.hsearch_db*
[94] print.htest*                       print.infl*
                                    print.integrate*
[97] print.isoreg*                      print.kmeans*
                                print.Latex*
```

List out Methods ...

```
[100] print.LaTeX*                                  print.libraryIQR
                                  print.listof
[103] print.lm*                                     print.loadings*
                                   print.loess*
[106] print.logLik*                                 print.ls_str*
                                    print.medpolish*
[109] print.MethodsFunction*                        print.mtable*
                                    print.NativeRoutineList
[112] print.news_db*                                print.nls*
                                       print.noquote
[115] print.numeric_version                         print.object_size*
                                print.octmode
[118] print.packageDescription*                     print.packageInfo
                                 print.packageIQR*
[121] print.packageStatus*                          print.pairwise.htest*
                              print.PDF_Array*
[124] print.PDF_Dictionary*                         print.pdf_doc*
                                  print.pdf_fonts*
[127] print.PDF_Indirect_Reference*                 print.pdf_info*
                                print.PDF_Keyword*
[130] print.PDF_Name*                               print.PDF_Stream*
                                 print.PDF_String*
[133] print.person*                                 print.POSIXct
                                    print.POSIXlt
[136] print.power.htest*                            print.ppr*
                                       print.prcomp*
```

List out Methods ...

```
[139] print.princomp*                                  print.proc_time
                                          print.raster*
[142] print.Rd*                                        print.recordedplot*
                                    print.restart
[145] print.RGBcolorConverter*                         print.rle
                                               print.roman*
[148] print.sessionInfo*                               print.simple.list
                                     print.smooth.spline*
[151] print.socket*                                    print.srcfile
                                         print.srcref
[154] print.stepfun*                                   print.stl*
                                        print.StructTS*
[157] print.subdir_tests*
      print.summarize_CRAN_check_status*               print.summary.aov*
[160] print.summary.aovlist*                           print.summaryDefault
                                  print.summary.ecdf*
[163] print.summary.glm*                               print.summary.lm*
                                     print.summary.loess*
[166] print.summary.manova*                            print.summary.nls*
                                print.summary.packageStatus*
[169] print.summary.ppr*                               print.summary.prcomp*
                                print.summary.princomp*
[172] print.summary.table                              print.table
                                      print.tables_aov*
[175] print.terms*                                     print.ts*
                                        print.tskernel*
```

List out Methods ...

```
[178] print.TukeyHSD*                                print.tukeyline*
                                  print.tukeysmooth*
[181] print.undoc*                                   print.vignette*
                                   print.warnings
[184] print.xgettext*                                print.xngettext*
                                  print.xtabs*
see '?methods' for accessing help and source code
```

- list methods that can apply to objects of class "lm"

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

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

S3/S4

- S4 is a more elaborate object structure, more formal.
- Many packages still written in S3.
- Different accessor symbols for lists.
- S3 allows abbreviation dat$x for dat[, "x"]
- S4 uses @ as accessor.
- Difficult to write functions that can receive either S3 or S4, check for type, and then do the right thing.

Outline

Table example

- We Needed presentable cross tabulations
- R table output too sparse

```
table(dat$yf1, dat$xf1)
```

```
   Alberquerque Denver Santa Fe
a             7      8         7
b            12     13        17
c            12     10        13
d             9      7         4
e             4      9        11
```

What's wrong with that?

- no margins
- no percents
- Can add those things with follow-up functions (prop.table, addmargins), but that's tedious.

Clunky to use, too

- Compared to other R functions, there is no beauty in the table function.

```
table(dat$yf1, dat$xf1)
```

- I wanted something that looked more like a regression model

```
myTable(yf1 ~ xf1, data = dat)
```

Table-maker more like regression and plot

- I wanted to replace this style

```
table(dat$yf1, dat$xf1, dnn = list("My Great Row Var", "My Column Var"))
```

with this:

```
myTable(yf1 ~ xf1, data = dat, rvlab = "My Great Row Var", cvlab = "My Column
    Var")
```

tabl was the first try

```
## A new table making function
##' Creates a cross tabulation with counts and column percents
##'
##' .. content for \details{} ..
##' @param x row variable name
##' @param y col variable name
##' @param data dataframe
##' @param xlab Optional: row variable label
##' @param ylab Optional: col variable label
##' @param exclude Default NULL, all values displayed (incl missing)
##' @param rounded Default FALSE, rounds to 10's for privacy purposes
##' @return list of tables
##' @author pauljohn@@ku.edu
tabl <- function(x, y, data = parent.frame(), xlab = NULL, ylab = NULL, exclude
    = NULL, rounded = FALSE){
    xlabel <- if (!missing(x)) deparse(substitute(x))
    ylabel <- if (!missing(y)) deparse(substitute(y))
    xlab <- if (is.null(xlab)) xlabel else xlab
    ylab <- if (is.null(ylab)) ylabel else ylab
    t1 <- table(data[ ,xlabel], data[ , ylabel], dnn = c(xlab, ylab), exclude
        = exclude)
    rownames(t1)[is.na(rownames(t1))] <- "NA" ## symbol to letters
    colnames(t1)[is.na(colnames(t1))] <- "NA"
    if (rounded) t1 <- round(t1, -1)
    t2 <- addmargins(t1, c(1,2))
    t1p <- round(100*prop.table(t1, 2), 1)
    t3 <- t2
```

tabl was the first try ...

```
      for(j in rownames(t1p)){
          for(k in colnames(t1p)){
              t3[j, k] <- paste0(t2[j, k], "(", t1p[j, k], "%)")
          }
      }
  print(t3)
  invisible(list(t2, t1p, t3, call = match.call() ))
  }
```

```
tabl(yf1, xf1, data = dat)
```

```
     xf1
yf1   Alberquerque Denver     Santa Fe   NA      Sum
  a   7(15.6%)     8(16.7%)   7(13.2%)   1(25%)  23
  b   12(26.7%)    13(27.1%)  17(32.1%)  1(25%)  43
  c   12(26.7%)    10(20.8%)  13(24.5%)  0(0%)   35
  d   9(20%)       7(14.6%)   4(7.5%)    0(0%)   20
  e   4(8.9%)      9(18.8%)   11(20.8%)  2(50%)  26
  NA  1(2.2%)      1(2.1%)    1(1.9%)    0(0%)   3
  Sum 45           48         53         4       150
```

- Step in a good direction
  - data frame argument!

tabl was the first try ...

- Some argument magic here. Allows users to name symbols as input variables. Did not require quoted variable names "xf1". That gets sacrificed in the final version.
- Defaults to show missing values

pctable() in the rockchalk package 1.8.90+

- Use a formula interface

```
require(rockchalk)
t1 <- pctable(yf1 ~ xf1, dat)
```

```
Count (column %)
     xf1
yf1   Alberquerque Denver    Santa Fe  Sum
  a   7(15.9%)     8(17%)    7(13.5%)  22
  b   12(27.3%)    13(27.7%) 17(32.7%) 42
  c   12(27.3%)    10(21.3%) 13(25%)   35
  d   9(20.5%)     7(14.9%)  4(7.7%)   20
  e   4(9.1%)      9(19.1%)  11(21.2%) 24
  Sum 44           47        52        143
```

creates a data structure in t1 that can be used to spit out various summaries

- Or even use it the old way

```
pctable(dat$yf1, dat$xf1)
```

pctable() in the rockchalk package 1.8.90+ ...

```
Count (column %)
        dat$xf1
dat$yf1  Alberquerque Denver     Santa Fe  Sum
    a    7(15.9%)     8(17%)     7(13.5%)  22
    b    12(27.3%)    13(27.7%)  17(32.7%) 42
    c    12(27.3%)    10(21.3%)  13(25%)   35
    d    9(20.5%)     7(14.9%)   4(7.7%)   20
    e    4(9.1%)      9(19.1%)   11(21.2%) 24
    Sum  44           47         52        143
```

- It turns out it is not allowed/practical to have this interface,

```
pctable(yf1, xf1, dat)
```

- However, I found a way to implement this

```
pctable("yf1", "xf1", dat)
```

pctable() in the rockchalk package 1.8.90+ ...

which is almost as good.
- Found ways to export to LaTeX and Word compatible formats, so students can make tables that don't offend the reader
- Keep options open to include counts and percents in same table

pctable is pronounced "presentable"

```
library(rockchalk)
pctable(yf1 ~ xf1, data = dat, rvlab = "A Thing I Predict", cvlab = "A Mighty
    Predictor!")
```

```
Count (column %)
                  A Mighty Predictor!
A Thing I Predict Alberquerque Denver     Santa Fe  Sum
                a   7(15.9%)     8(17%)    7(13.5%)  22
                b   12(27.3%)    13(27.7%) 17(32.7%) 42
                c   12(27.3%)    10(21.3%) 13(25%)   35
                d   9(20.5%)     7(14.9%)  4(7.7%)   20
                e   4(9.1%)      9(19.1%)  11(21.2%) 24
              Sum   44           47        52        143
```

In my opinion, nobody should ever want row percents, but if you do, I will still be your friend:

```
pctable(yf1 ~ xf1, data = dat, rowpct = TRUE, colpct = FALSE)
```

```
Count (row %)
     xf1
yf1   Alberquerque Denver     Santa Fe  Sum
  a   7(31.8%)     8(36.4%)   7(31.8%)  22
  b   12(28.6%)    13(31%)    17(40.5%) 42
  c   12(34.3%)    10(28.6%)  13(37.1%) 35
  d   9(45%)       7(35%)     4(20%)    20
  e   4(16.7%)     9(37.5%)   11(45.8%) 24
  Sum 44           47         52        143
```

Make a high quality table

```
Count (column %)
      xf1
yf1    Alberquerque Denver     Santa Fe  NA      Sum
  a    7(15.6%)     8(16.7%)   7(13.2%)  1(25%)  23
  b    12(26.7%)    13(27.1%)  17(32.1%) 1(25%)  43
  c    12(26.7%)    10(20.8%)  13(24.5%) 0(0%)   35
  d    9(20%)       7(14.6%)   4(7.5%)   0(0%)   20
  e    4(8.9%)      9(18.8%)   11(20.8%) 2(50%)  26
  NA   1(2.2%)      1(2.1%)    1(1.9%)   0(0%)   3
  Sum  45           48         53        4       150
```

Latex output

	xf1				
yf1	Alberquerque	Denver	Santa Fe	NA	Sum
a	8(19%)	12(22.6%)	13(25.5%)	0(0%)	33
b	9(21.4%)	7(13.2%)	11(21.6%)	3(75%)	30
c	8(19%)	11(20.8%)	7(13.7%)	1(25%)	27
d	11(26.2%)	17(32.1%)	9(17.6%)	0(0%)	37
e	5(11.9%)	5(9.4%)	10(19.6%)	0(0%)	20
NA	1(2.4%)	1(1.9%)	1(2%)	0(0%)	3
Sum	42	53	51	4	150

What's required to make this work? pctable

- From rockchalk source code, this is it!

```
pctable <- function(rv, ...) {
    UseMethod("pctable")
}
NULL
pctable.default <-
function(rv, cv,rvlab = NULL, cvlab = NULL, colpct = TRUE,
rowpct = FALSE, rounded = FALSE, ...)
{
    rvlabel <- if (!missing(rv)) deparse(substitute(rv))
    cvlabel <- if (!missing(cv)) deparse(substitute(cv))
    rvlab <- if (is.null(rvlab)) rvlabel else rvlab
    cvlab <- if (is.null(cvlab)) cvlabel else cvlab

    dots <- list(...)
    dotnames <- names(dots)

    tableargs <- list(rv, cv, dnn = c(rvlab, cvlab))
    newargs <- modifyList(tableargs, dots, keep.null = TRUE)

    t1 <- do.call("table", newargs)
    rownames(t1)[is.na(rownames(t1))] <- "NA" ## symbol to letters
    colnames(t1)[is.na(colnames(t1))] <- "NA"
    if (rounded) t1 <- round(t1, -1)
```

What's required to make this work? pctable ...

```
      t2 <- addmargins(t1, c(1,2))
      t1colpct <- round(100*prop.table(t1, 2), 1)
      t1rowpct <- round(100*prop.table(t1, 1), 1)
      t1colpct <- apply(t1colpct, c(1,2), function(x) gsub("NaN", "", x))
      t1rowpct <- apply(t1rowpct, c(1,2), function(x) gsub("NaN", "", x))
      res <- list("count" = t2, "colpct" = t1colpct, "rowpct" = t1rowpct,
                  call = match.call())
      class(res) <- "pctable"
      print.pctable(res, colpct = colpct, rowpct = rowpct)
      invisible(res)
  }
NULL
## Creates a cross tabulation with counts and column percents
##
## The formula method is the recommended method for users. Run
## \code{pctable(myrow ~ mycol, data = dat)}. In an earlier version,
## I gave different advice, so please adjust your usage.
##
## @param formula A two sided formula.
## @param data A data frame.
## @examples
##   [omitted]
## @rdname pctable
## @method pctable formula
## @export
pctable.formula <-
```

What's required to make this work? pctable ...

```
function(formula, data = NULL,  rvlab = NULL, cvlab = NULL, colpct = TRUE,
     rowpct = FALSE, rounded = FALSE, ...)

{
    if (missing(formula) || (length(formula) != 3L))
        stop("pctable requires a two sided formula")
    dots <- list(...)
    dotnames <- names(dots)

    mt <- terms(formula, data = data)
    if (attr(mt, "response")==0L) stop("response variable is required")
    mf <- match.call(expand.dots = TRUE)
    mfnames <- c("formula", "data", "subset", "xlev", "na.action", "
        drop.unused.levels")
    keepers <- match(mfnames, names(mf), 0L)
    mf <- mf[c(1L, keepers)]

    if (!"na.action" %in% dotnames) mf$na.action <- na.pass
    mf[[1L]] <- quote(stats::model.frame)

    ## remove used arguments from dots, otherwise errors happen
    ## when unexpected arguments pass through. Don't know why
    for (i in c("subset", "xlev", "na.action", "drop.unused.levels")) dots[[i
        ]] <- NULL

    mf <- eval(mf, parent.frame())
    mfnames <- names(mf)
```

What's required to make this work? pctable ...

```
      response <- attr(attr(mf, "terms"), "response")
      ## response is column 1
      rvname <- mfnames[response]
      cvname <- mfnames[-response][1] ##just take 2?
      rvlab <- if (missing(rvlab)) rvname else rvlab
      cvlab <- if (missing(cvlab)) cvname else cvlab

      arglist <- list(rv = mf[[rvname]], cv = mf[[cvname]],
                      rvlab = rvlab, cvlab = cvlab,
                      colpct = colpct, rowpct = rowpct,
                      rounded = rounded)
      arglist <- modifyList(arglist, dots, keep.null = TRUE)

      res <- do.call(pctable.default, arglist)
      invisible(res)
  }
NULL
##' Method for variable names as character strings
##'
##' The character method exists only for variety.  It accepts
##' character strings rather than a formula to define the columns that
##' should be plotted.  The method used most often for most users should
##' be the formula method.
##'
##' When using character arguments, the row variable rv rowvar must be
##' a quoted string if the user intends the method pctable.character
##' to be dispatched. The column variable cv may be a string or just a
```

What's required to make this work? pctable ...

```
##' variable name (which this method will coerce to a string).
##' @examples omitted
##' @rdname pctable
##' @method pctable character
##' @export
pctable.character <-
 function(rv, cv, data = NULL, rvlab = NULL,
     cvlab = NULL, colpct = TRUE, rowpct = FALSE,
     rounded = FALSE, ...)
 {
     if (missing(data) || !is.data.frame(data)) stop("pctable requires a data
         frame")
     cv <- as.character(substitute(cv))[1L]

     rvlab <- if (missing(rvlab)) rv else rvlab
     cvlab <- if (missing(cvlab)) cv else cvlab
     res <- pctable.formula(formula(paste(rv, " ~ ", cv)), data = data,
                            rvlab = rvlab, cvlab = cvlab, colpct = colpct,
                            rowpct = rowpct, rounded = rounded, ...)

     invisible(res)
 }
NULL
##' Extract presentation from a pctable object
##'
##' Creates a column and/or row percent display of a pctable
##' result
```

What's required to make this work? pctable ...

```
##' @param object A pctable object
##' @param colpct Default TRUE: should column percents be included
##' @param rowpct Default FALSE: should row percents be included
##' @param ... Other arguments, currently unused
##' @return An object of class summary.pctable
##' @author Paul Johnson <pauljohn@@ku.edu>
##' @method summary pctable
##' @export
summary.pctable <-
 function(object, ..., colpct = TRUE, rowpct = FALSE)
 {
     colpct <- if (missing(colpct)) object$call[["colpct"]] else colpct
     rowpct <- if (missing(rowpct)) object$call[["rowpct"]] else rowpct

     count <- object[["count"]]

     t3 <- count
     attr(t3, which = "colpct") <- colpct
     attr(t3, which = "rowpct") <- rowpct
     class(t3) <- c("summary.pctable", "table")
     if (colpct && !rowpct) {
         cpct <- object[["colpct"]]
         for(j in rownames(cpct)){
             for(k in colnames(cpct)){
                 t3[j, k] <- paste0(count[j, k], "(", cpct[j, k], "%)")
             }
         }
```

What's required to make this work? pctable ...

```
        return(t3)
    }

    ## rowpct == TRUE else would have returned
    rpct <- object[["rowpct"]]
    for(j in rownames(rpct)){
        for(k in colnames(rpct)){
            t3[j, k] <- paste0(count[j, k], "(", rpct[j, k], "%)")
        }
    }

    if (!colpct) {
        return(t3)
    } else {
        cpct <- object[["colpct"]]
        t4 <- array("", dim = c(1, 1) + c(2,1)*dim(object$colpct))
        t4[seq(1, NROW(t4), 2), ] <- t3
        rownames(t4)[seq(1, NROW(t4), 2)] <- rownames(t3)
        rownames(t4)[is.na(rownames(t4))] <- ""
        colnames(t4) <- colnames(t3)
        for(j in rownames(object[["colpct"]])) {
            for(k in colnames(object[["colpct"]])){
                t4[1 + which(rownames(t4)==j) ,k] <- paste0(object[["colpct"
                    ]][j, k], "%")
            }

        }
```

What's required to make this work? pctable ...

```
        t4 <- as.table(t4)
        names(dimnames(t4)) <- names(dimnames(count))
        attr(t4, which = "colpct") <- colpct
        attr(t4, which = "rowpct") <- rowpct
        class(t4) <- c("summary.pctable", "table")
        return(t4)
    }
}
##' print method for summary.pctable objects
##'
##' prints pctab objects. Needed only to deal properly with quotes
##'
##' @param x a summary.pctable object
##' @param ... Other arguments to print method
##' @return Nothing is returned
##' @author Paul Johnson <pauljohn@@ku.edu>
##' @method print summary.pctable
##' @export
print.summary.pctable <- function(x, ...){
    colpct <- attr(x, "colpct")
    rowpct <- attr(x, "rowpct")

    if (colpct && !rowpct) {
        cat("Count (column %)\n")
    } else if (!colpct && rowpct) {
        cat("Count (row %)\n")
    } else {
```

What's required to make this work? pctable ...

```
            cat("Count (row %)\n")
            cat("column %\n")
        }
        NextMethod(generic = "print", object = x, quote = FALSE, ...)
 }
##' Display pctable objects
##'
##' This is not very fancy. Note that the saved pctable object
##' has the information inside it that is required to write both
##' column and row percentages. The arguments colpct and rowpct
##' are used to ask for the two types.
##'
##' @param x A pctable object
##' @param colpct Default TRUE: include column percentages?
##' @param rowpct Default FALSE: include row percentages?
##' @param ... Other arguments passed through to print method
##' @return A table object for the final printed table.
##' @author Paul Johnson <pauljohn@@ku.edu>
##' @method print pctable
##' @export
print.pctable <- function(x, colpct = TRUE, rowpct = FALSE, ...)
 {
     colpct <- if (missing(colpct)) x$call[["colpct"]] else colpct
     rowpct <- if (missing(rowpct)) x$call[["rowpct"]] else rowpct
     tab <- summary(x, colpct = colpct, rowpct = rowpct)
     print(tab, ...)
     invisible(tab)
```

What's required to make this work? pctable ...

```
  }
NULL
```

Why does this work? R class

- pctable is designed with the R secret recipe in mind.
- pctable()
  - creates a set of data structures
- summary.pctable can create a summary view, which picks through the structures to make the desired information.
- print.summary.pctable displays the summary to the user

Attributes

- Object is created with an attribute named "class"
- R runtime system depends on the class attribute.
  See all attributes

```
attributes(t1)
```

- Or just the one attribute we want to know

```
class(t1)
```

R Functions set Attributes as class strings

- Check any of the major model fitting functions in the R source code.
- Toward the end, as in the lm() function, one will find:

```
class(z) <- c(if (is.matrix(y)) "mlm", "lm")
```

- Meaning: If the model is multivariate, the class vector will be c("mlm", "lm"), otherwise just "lm".
- More specialized classes are appended *on the front* of the list.
  - Something classed as c("mlm", "lm") can have specialized functions for "mlm" objects where needed, but can use functions intended for "lm" objects when suitable.

Outline

The S3 system is very informal

- Can set class attribute to anything you want
- Unlike more structured languages like C++ or Java, there is no type-checking to make sure this makes sense.
- No need to "register" the class with the runtime system in any more elaborate way.
- For existing R generic functions like summary, plot, print, table, and so forth, one need only create functions named summary.mynewclass, print.mynewclass, table.mynewclass, etc.

Create some mischief, version 1.

- Lie to the R runtime (make a false claim.

```
m1 <- c(1, 2, 3)
class(m1) <- "lm"
```

```
summary(m1)
```

```
Error: $ operator is invalid for atomic vectors
```

Create some mischief, version 2.

- Here's a real lm regression object

```
dat <- data.frame(x = rnorm(100), y = rnorm(100))
m1 <- lm(y ~ x, data = dat)
```

- Add a new class on the front.

```
class(m1) <- c("pjgreatest", class(m1))
class(m1)
```

```
[1] "pjgreatest" "lm"
```

- The "pjgreatest" class has no methods, so anything we run will "still work" because R looks at the class list. It finds no method "summary.pjgreatest", but it finds the Next Method "summary.lm". See?

Create some mischief, version 2. ...

```
summary(m1)
```

```
Call:
lm(formula = y ~ x, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.56919 -0.54544  0.00144  0.56054  2.59574

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.06187    0.09580   0.646    0.520
x           -0.06282    0.10114  -0.621    0.536

Residual standard error: 0.9522 on 98 degrees of freedom
Multiple R^2:  0.003921,  Adjusted R^2:  -0.006243
F-statistic: 0.3858 on 1 and 98 DF,  p-value: 0.536
```

- Now lets provide a new summary method

Create some mischief, version 2. ...

```
summary.pjgreatest <- function(object, ...){
      newres <- NextMethod(generic = "summary", object = object, ...)
      newres$Rsquare <- runif(1, min = 0.90, max = 0.999)
      class(newres) <- c("summary.pjgreatest", class(newres))
      newres
 }
```

Create some mischief, version 2.

- -The strong (strong!) recommendation from R Core is to write summary methods that return data structures, and then we write a print method to display that structure.

```
print.summary.pjgreatest <- function(x, ...){
 cat("You are now in the twilight zone. \n")
 cat("All regressions fitted here are nearly perfect. \n")
 cat(paste("The Rsquare is", x$Rsquare, "\n\n"))
 cat(paste("That's a great Rsquare, just like you deserve! \n\n"))
 cat("But if you want to believe those R guys, the R square is only", x$
      r.square, "\n\n")
 cat("See?\n")
 NextMethod(generic = "print", object = x, ...)
 }
```

```
summary(m1)
```

Create some mischief, version 2. ...

```
You are now in the twilight zone.
All regressions fitted here are nearly perfect.
The Rsquare is 0.933691663353704

That's a great Rsquare, just like you deserve!

But if you want to believe those R guys, the R square is only 0.003920863

See?

Call:
lm(formula = y ~ x, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.56919 -0.54544  0.00144  0.56054  2.59574

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.06187    0.09580   0.646    0.520
x           -0.06282    0.10114  -0.621    0.536

Residual standard error: 0.9522 on 98 degrees of freedom
Multiple R^2:  0.003921,  Adjusted R^2:  -0.006243
F-statistic: 0.3858 on 1 and 98 DF,  p-value: 0.536
```

Generic functions almost always are simple with a minimal declaration

- "..." is a legal word. Meaning "user can add named arguments and if they don't match the ones we expect, we'll toss all of them into "...". Comments about processing "..." included in new Rchaeology.
- The generic print

```
print <- function(x, ...) UseMethod("print")
```

- This preserves flexibility, puts weight of work onto methods, which
- can have more named arguments.
- must have arguments x and ...

... is a three letter word

- Literally, three periods is a word!
- There's not much manual info on "...". Study of the R source code. See how they do it, then do it like they do.
- Explanation inserted into new Rchaeology vignette.
- Inside a function, we almost always either grab and inspect the dots

```
dots <- list(...)
dotnames <- names(dots)
```

- Often, the dots have argument that are intended for a particular purpose. We often have to separate the arguments, by name (dotnames).

Outline

Building an S3 class around an Mplus Automation result

- A project becomes chaotic if we let researchers rename all of the functions to suit their tastes.

- Likely more helpful to users if we can reshape functions in to the familiary R sequence

```
m1 <- fitter(y ~ x1 + x2, data = dat)
summary(m1)
coef(m1) ## method displays brief
coef(summary(m1))
anova(m1)
plot(m1)
```

We began with an interace with idiosyncratic names

```
runEverything <- function(dFrame = NULL, formString = NULL, missValue = -999,
      patternV = patternVector, mScript = mplusSkel, mplusFile = "mplus.csv",
      isMissing = FALSE, numPerLine = 5, isWeighted = FALSE, mplusDir = "../../
      NLTS2-data/Mplus/", subPop = FALSE) {
      ## snip
 }
summarizeObj <- function(List = NULL, r.VGAM = FALSE, dFrame = NULL){
      ## snip
 }
```

Here's what we end up with

```
stages <- function(formula = NULL, data = NULL, missValue = -999, skel =
      mplusSkel, mplusFile = "mplus.csv", isMissing = FALSE, numPerLine = 5,
      isWeighted = FALSE, mplusDir = "../../NLTS2-data/Mplus/", subPop = FALSE)
 {
      require(MplusAutomation)
      if (!file.exists(mplusDir)) dir.create(mplusDir, showWarnings = FALSE,
            recursive = TRUE)
      if (!plyr::is.formula(formula)) formula <- as.formula(formula)
      ## Create the matrix and recoding for Mplus
      mMat <- makeMplusMat(formula, data, missValue)
      ## Make the number of stages
      nstages <- length( unique( mMat[mMat[, 1] != -999, 1] ) )
      ## Check if nstages is 2, 3, or 4
      if(nstages < 2 || nstages > 4){
          stop('The number of stages must be 2, 3, or 4')
      }
      ## Add in code about 2 stages aka logistic regression
      if(nstages > 2){
          mMat <- recodeMplus(mMat, nstages, missValue)
      }
      ## Makes the replacement vector along with the mplus Key
      ## for conversion later returns list
      rList <- makeReplacementVector(mplusFile = mplusFile,
                  data = mMat, skel = skel, isMissing = isMissing, numPerLine =
                        numPerLine,
                  isWeighted = isWeighted, nstages = nstages,
                  missValue = as.character(missValue),
```

Here's what we end up with ...

```
                subPop = subPop)
mplusKey <- rList[[2]]
outList <- runMplus(data = mMat, replaceV = rList[[1]], skel, mplusDir,
     mplusFile)
## more direct way to rename all values in param column, misses Thresholds
     , however
pjkey <- mplusKey[ , "predictors"]
names(pjkey) <- mplusKey[ , "mplus"]
## Now beautify names of variables and columns in parameter objects
newheaders <- c(pval = "Pr(>|z|)", est = "Estimate", se = "Std Error",
     est_se = "z value")
for(aname in names(outList$parameters)){
    param <- outList[["parameters"]][[aname]][["param"]]
    outList[["parameters"]][[aname]][["param"]] <-
        ifelse( param %in% names(pjkey), pjkey[param], param)
    ## PJ. need to catch thresholds named V$x. make second pass
    param <- sapply(strsplit(param, "\\$"), function(xxx) xxx[1])
    outList[["parameters"]][[aname]][["param"]] <-
        ifelse( param %in% names(pjkey), pjkey[param], param)
    paramHeader <- outList[["parameters"]][[aname]][["paramHeader"]]
    phlist <- strsplit(paramHeader, "\\.")
    for(i in seq_along(phlist)){
        phlist[[i]][1] <- ifelse(phlist[[i]][1] %in% names(pjkey),
        pjkey[phlist[[i]][1]],
        phlist[[i]][1])
    }
    outList[["parameters"]][[aname]][["paramHeader"]] <-
```

Here's what we end up with ...

```
            lapply(phlist, function(xx) paste0(xx, collapse = "."))
        oldcols <- colnames(outList[["parameters"]][[aname]])
        colnames(outList[["parameters"]][[aname]]) <- ifelse(oldcols %in%
            names(newheaders), newheaders[oldcols], oldcols)
    }
    res <- list(output = outList, mplusKey = mplusKey, formula = formula,
                mplusFile = paste0(mplusDir, "mplus.out"))
    class(res) <- "stages"
    invisible(res)
}
summary.stages <- function(object = NULL, type = "unstandardized", trimrows =
    TRUE, ...)
{
    ## Check for type as character
    if(is.character(type)){
        type <- match.arg(tolower(type),
            c('unstandardized', 'std.standardized',
              'stdy.standardized', 'stdyx.standardized'))
    } else {
        stop(paste("type must equal unstandardized,",
        "stdyx.standardized, stdy.standardized,",
        "or std.standardized."))
    }
    formula <- object$formula
    coefs <- object[["output"]][["parameters"]][[type]]
    noise <- grep("Means$|Variances$|.*WITH$", coefs$paramHeader)
    paramHeader <- coefs$paramHeader
```

Here's what we end up with ...

```
if (trimrows & length(noise) > 0){
    coefs <- coefs[-noise, ]
}
## only if trimrows can we tighten up predictor names
if (trimrows){
    paramHeader <- substr(coefs$paramHeader, 1, 2)
    paramHeader <- gsub("C", "", paramHeader)
}
## Make rownames
rownames(coefs) <- paste0(coefs$param, ";", paramHeader)
## JH 2/4/2015
## create Th;1, ...., Th;n
```

```
charVec <- paste0('^', rownames(object$mplusKey)[grep('^C\\d{1,}_',
     rownames(object$mplusKey))], ';.+')
newThVec <- paste0('Threshold;', 1:length(charVec))
## Final rownames
rownames(coefs) <- multiGsub(charVec, newThVec, rownames(coefs))
coefs$param <- NULL
coefs$paramHeader <- NULL
coefs <- as.matrix(coefs)
res <- list(coefficients = coefs, N = object[[1]]$summaries$Observations,
            LL = object[[1]]$summaries$LL ,
            AIC = object[[1]]$summaries$AIC,
            BIC = object[[1]]$summaries$BIC,
            aBIC = object[[1]]$summaries$aBIC,
            AICC = object[[1]]$summaries$AICC, formula = formula)
class(res) <- "summary.stages"
```

Here's what we end up with ...

```
    attr(res, "trimrows") <- trimrows
    attr(res, "type") <- type
    res
}
print.summary.stages <- function(x, digits = max(3L, getOption("digits") - 3L),
                         signif.stars = getOption("show.signif.stars"), ....)
{
    coefs <- x$coefficients
    cat(paste("This is", attr(x, "type") ,
        "output from Mplus, reformatted similar to VGAM", "\n"))
    if (attr(x, "trimrows")) cat(paste("Note: Omitting output rows for WITH,",
        "Variances, Means", "\n"))
    printCoefmat(coefs, digits = digits, signif.stars = signif.stars,
                 na.print = "NA")
    cat(paste("N:" , formatC(x$N, digits = digits), "\n"))
    cat(paste("LL:" , formatC(x$LL, digits = digits), "\n"))
    cat(paste("AIC:" , formatC(x$AIC, digits = digits), "\n"))
    cat(paste("BIC:" , formatC(x$BIC, digits = digits), "\n"))
    cat(paste("aBIC:" , formatC(x$aBIC, digits = digits), "\n"))
    cat(paste("AICC:" , formatC(x$AICC, digits = digits), "\n"))
    invisible(coefs)
}
nobs.stages <- function(object, ...){
    object[[1]]$summaries$Observations
}
logLik.stages <- function(object, ...){
    myll <- object[[1]]$summaries$LL
```

Here's what we end up with ...

```
      attr(myll, "df") <- object[[1]]$summaries$Parameters
      myll
 }
coef.summary.stages <- function(object, ...){
      ggg <- object$coefficients
 }
coef.stages <- function(object, ...){
      objsum <- summary(object)$coefficients
 }
```