Chapter 13: S3

Ryan Heslin

May 31, 2022

Basics

1.

t.test is the generic for the t.test function, which dispatches to the appropriate method. t.data.frame is the method of t for the data.frame class (it just coerces to matrix and invokes NextMethod).

2.

data.frame and many of the as.* and is.* family are major offenders.

3.

It coerces a data frame to a data frame, which means stripping classes inherited before data.frame and optionally adding row names. The overuse of dot separators makes it hard to understand that this is an S3 method, a problem that could have been solved by using snake case instead.

4.

The first dispatches to mean.Date, which coerces a date to integer, calls mean on it, and coerces back to Date. The second coerces to integer beforehand without doing this.

```
set.seed(1014)
some_days <- as.Date("2017-01-31") + sample(10, 5)
```

```
mean(some_days)
```

```
[1] "2017-02-06"
```

```
#> [1] "2017-02-06"
mean(unclass(some_days))
```

```
[1] 17203.4
```

```
#> [1] 17203
```

5.

It generates the ECDF function for a given vector. The object contains the function and preserves the call used to create it.

```
x <- ecdf(rpois(100, 10))
x
```

```
Empirical CDF 
Call: ecdf(rpois(100, 10))
 x[1:18] =      2,      3,      4,  ...,     18,     19
```

```
#> Empirical CDF
#> Call: ecdf(rpois(100, 10))
#>  x[1:18] =  2,  3,  4,  ..., 2e+01, 2e+01
```

6.

A `table` object is an atomic vector array. Like arrays in general, it has dimensions and a `dimnames` attribute. The class is mostly used for its print method.

```
x <- table(rpois(100, 5))
x
```

```

 1  2  3  4  5  6  7  8  9 10 
 7  5 18 14 15 15 14  4  5  3 
```

Classes

I elect not to enforce unique names, and I generate default column names the same way as `data.frame`.

1.

```
data.frame2 <- function(..., .row_names = NULL) {
  dots <- list(...)
  l_dots <- length(dots)
  if (l_dots == 0) {
    return(structure(list(),
      class = "data.frame",
      row.names = make_row_names(.row_names, 0)
    ))
  }
  dots_names <- names(dots)
  has_names <- !is.null(dots_names)
  col_names <- vector("character", l_dots)
  col_data <- vector("list", l_dots)
  for (i in seq_along(dots)) {
    el <- dots[[i]]
    el_name <- dots_names[[i]]
    if (length(dim(el)) > 1) { # Data frame or array
      if (is.array(el)) el <- as.data.frame(el)
      el_rows <- nrow(el)
      # Ignore element name and instead combine with existing column names
      if (has_names) {
        col_names[[i]] <- ""
        if (el_name != "") {
          colnames(el) <- paste(el_name, colnames(el,
            do.NULL = TRUE,
            prefix = "V"
          ),
          sep
          = "_"
          )
        }
      }
    } else if (!is.array(el) &&
      is.vector(el)) { # Vector case
      el_rows <- length(el)
      el <- list(el) # Ensure correct concatenation
      if (has_names && el_name != "") {
        col_names[[i]] <- el_name
      }
    } else {
      stop("Cannot coerce object")
    }
    if (i == 1) {
      n_rows <- el_rows
    } else if (n_rows != el_rows) {
      stop("Number of rows mismatch")
    }
    col_data[[i]] <- el
  }

  .row_names <- make_row_names(.row_names, n_rows)

  # Supply default column names for unnamed arguments
  if (has_names) {
    unnamed <- is.na(col_names)
    col_names[unnamed] <- paste0("V", seq_along(unnamed))
    names(col_data) <- col_names
  }
  full_data <- do.call(c, col_data)
  dnn <- c(.row_names, names(full_data))
  # class(full_data) <- "data.frame"

  structure(
    .Data = full_data, class = c("data.frame"),
    row.names = .row_names
  )
  # dim = c(n_rows, length(full_data))
  # )
}

# Check row names and create if necessary
make_row_names <- function(rn, n_rows) {
  if (!is.null(rn)) {
    if (length(rn) != n_rows) {
      stop("Length of row names does not match data length")
    }
    if (anyDuplicated(rn)) stop("Duplicate row names")
  } else {
    rn <- seq_len(n_rows)
  }
  as.character(rn)
}

library(testthat)

data.frame2(mtcars, x = mtcars$cyl)
```

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	x
21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	6
21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	6
22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	4
21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	6
18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	8
18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	6

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	x
14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	8
24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	4
22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	4
19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	6
17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	6
16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	8
17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	8
15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	8
10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	8
10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	8
14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	8
32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1	4
30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2	4
33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	4
21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	4
15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2	8
15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2	8
13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4	8
19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2	8
27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1	4
26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2	4
30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2	4
15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4	8
19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6	6
15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8	8
21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2	4

```r
data.frame2(mpg = mtcars$mpg)
```

mpg
21.0
21.0
22.8
21.4
18.7
18.1
14.3
24.4
22.8
19.2
17.8
16.4
17.3
15.2
10.4
10.4
14.7
32.4
30.4
33.9

mpg
21.5
15.5
15.2
13.3
19.2
27.3
26.0
30.4
15.8
19.7
15.0
21.4

```
data.frame2(mpg = mtcars$mpg, cyl = mtcars$cyl)
```

mpg	cyl
21.0	6
21.0	6
22.8	4
21.4	6
18.7	8
18.1	6
14.3	8
24.4	4
22.8	4
19.2	6
17.8	6
16.4	8
17.3	8
15.2	8
10.4	8
10.4	8
14.7	8
32.4	4
30.4	4
33.9	4
21.5	4
15.5	8
15.2	8
13.3	8
19.2	8
27.3	4
26.0	4
30.4	4
15.8	8
19.7	6
15.0	8
21.4	4

```
data.frame2(unname(as.matrix(mtcars)))
```

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
data.frame2(unname(as.matrix(mtcars)), .row_names = rownames(mtcars))
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
data.frame2(mtcars$cyl, y = as.matrix(mtcars$disp), mtcars[4:6], .row_names = rownames(mtcars))
```

		y_V1	hp	drat	wt
Mazda RX4	6	160.0	110	3.90	2.620
Mazda RX4 Wag	6	160.0	110	3.90	2.875
Datsun 710	4	108.0	93	3.85	2.320
Hornet 4 Drive	6	258.0	110	3.08	3.215
Hornet Sportabout	8	360.0	175	3.15	3.440
Valiant	6	225.0	105	2.76	3.460
Duster 360	8	360.0	245	3.21	3.570
Merc 240D	4	146.7	62	3.69	3.190
Merc 230	4	140.8	95	3.92	3.150
Merc 280	6	167.6	123	3.92	3.440
Merc 280C	6	167.6	123	3.92	3.440
Merc 450SE	8	275.8	180	3.07	4.070
Merc 450SL	8	275.8	180	3.07	3.730
Merc 450SLC	8	275.8	180	3.07	3.780
Cadillac Fleetwood	8	472.0	205	2.93	5.250
Lincoln Continental	8	460.0	215	3.00	5.424
Chrysler Imperial	8	440.0	230	3.23	5.345
Fiat 128	4	78.7	66	4.08	2.200
Honda Civic	4	75.7	52	4.93	1.615
Toyota Corolla	4	71.1	65	4.22	1.835
Toyota Corona	4	120.1	97	3.70	2.465
Dodge Challenger	8	318.0	150	2.76	3.520
AMC Javelin	8	304.0	150	3.15	3.435
Camaro Z28	8	350.0	245	3.73	3.840
Pontiac Firebird	8	400.0	175	3.08	3.845

		y_V1	hp	drat	wt
Fiat X1-9	4	79.0	66	4.08	1.935
Porsche 914-2	4	120.3	91	4.43	2.140
Lotus Europa	4	95.1	113	3.77	1.513
Ford Pantera L	8	351.0	264	4.22	3.170
Ferrari Dino	6	145.0	175	3.62	2.770
Maserati Bora	8	301.0	335	3.54	3.570
Volvo 142E	4	121.0	109	4.11	2.780

```
data.frame2()
```

|| || || ||

```
expect_error(data.frame2(NULL))
expect_error(data.frame2(mtcars$cyl, iris$Species))
```

2.

I replicate the base behavior of replacing values absent from the levels with `NA` and excluding `NA` from the levels.

```
new_factor <- function(x = integer(), levels = character(), contr = NULL, ...) {
  stopifnot(is.integer(x))
  stopifnot(is.character(levels))
  levels <- levels[!is.na(levels)]

  nlevel <- length(unique(levels))
  out_call <-
    if (!is.null(contr)) {
      if (is.character(contr)) contr <- match.fun(contr)
      if (is.function(contr)) {
        contr <- contr(nlevel, ...)
      }
      if (!is.matrix(contr)) stop("Contrasts must be a matrix")
      if (nrow(contr) != nlevel) {
        stop("If supplied, contrasts must have one row for each level of x")
      }
    }
  out <- structure(
    x,
    levels = levels,
    class = "factor"
  )
  if (!is.null(contr)) contrasts(out) <- contr
  out
}
```

```
validate_factor <- function(x) {
  values <- unclass(x)
  levels <- attr(x, "levels")

  if (!all(values[!is.na(values)] > 0)) {
    stop(
      "All ‘x‘ values must be greater than zero",
      call. = FALSE
    )
  }


  if (length(levels) < max(values, na.rm = TRUE)) {
    stop(
      "There must be at least as many ‘levels‘ as possible values in ‘x‘",
      call. = FALSE
    )
  }

  x
}


factor <- function(x = character(), levels = unique(x), contr = NULL, ...) {
  levels <- as.character(levels)
  ind <- match(x, levels)
  validate_factor(new_factor(ind, levels, contr = contr, ...))
}

factor(c("a", "a", "b"))
```

```
[1] a a b
Levels: a b
```

```
factor(1:3)
```

```
[1] 1 2 3
Levels: 1 2 3
```

```
factor(1:3, levels = c(1, 3))
```

```
[1] 1    <NA> 3
Levels: 1 3
```

```
factor(1:3, levels = "a")
```

```
[1] <NA> <NA> <NA>
Levels: a
```

```
factor(1:3, contr = "contr.helmert")
```

```
[1] 1 2 3
attr(,"contrasts")
  [,1] [,2]
1   -1   -1
2    1   -1
3    0    2
Levels: 1 2 3
```

```
expect_error(factor(1:3, levels = 1, contr = "contr.poly"))
expect_error(factor(1:3, contr = 1:5))
#> [1] a a b
#> Levels: a b
```

3.

The base `factor` has the additional features of mapping different labels to the same levels and ordering the factors. More saliently, it assigns values that do not appear in the levels `NA` instead of throwing an error.

4.

Contrasts refer to different ways of encoding categorical variables in models in order numerically express the effects of different levels. Above, I rewrote `factor` to use this attribute if supplied.

5.

The validator should confirm that inputs are integer vectors whose elements are all in [1, 3899], the range of valid Roman numerals, or character vectors of such valid Roman numerals. A constructor would then convert the input to integer, if necessary, then just set its class to `roman`, enabling the specialized methods to do their work.

Generics and Methods

1.

It works correctly because `UseMethod` ultimately dispatches to `t.default`, since there is no `test` method for `t`.

```
library(sloop)

x <- structure(1:10, class = "test")
t(x)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    2    3    4    5    6    7    8    9    10
attr(,"class")
[1] "test"
```

2.

```
s3_methods_class("table")
```

generic	class	visible	source
[	table	TRUE	base
aperm	table	TRUE	base
as.data.frame	table	TRUE	base
Axis	table	FALSE	registered S3method
lines	table	FALSE	registered S3method
plot	table	FALSE	registered S3method
points	table	FALSE	registered S3method
print	table	TRUE	base
summary	table	TRUE	base
tail	table	FALSE	registered S3method

3.

```
s3_methods_class("ecdf")
```

generic	class	visible	source
plot	ecdf	TRUE	stats
print	ecdf	FALSE	registered S3method
quantile	ecdf	FALSE	registered S3method
summary	ecdf	FALSE	registered S3method

4.

`print`, naturally

```r
generics <- lsf.str("package:base")
generics <- generics[sapply(generics, isS3stdGeneric)]
names(generics) <- generics
sapply(generics, \(x) nrow(s3_methods_generic(x))) |>
  sort(decreasing = TRUE) |>
  head()
```

```
    print    format   summary      plot   as.list all.equal 
      283       134        41        30        17        12 
```

5.

```r
g <- function(x) {
  x <- 10
  y <- 10
  UseMethod("g")
}
g.default <- function(x) c(x = x, y = y)

x <- 1
y <- 1
g(x)
```

```
 x  y 
 1 10 
```

```
#>  x  y
#>  1 10
```

`UseMethod` constructs a call by matching arguments in the generic's execution environment *as they came in*, forwarding them, then matching arguments defined in the execution environment. These are then forwarded to the method that is matched. So the redefinition of `x` is ignored because only the value passed is read. This ignores the usual rule for argument lookup.

`NextMethod` skips the first method matched by `UseMethod` and continues searching, potentially to internal generics. Arguments are passed as promises to evaluate in the caller environment of NextMethod. This makes it possible to force use of a default or internal method by placing a call to `NextMethod` in a class-specific method.

6.

The answer differs for the many different methods implemented.

Object Styles

1.

Vector : `factor`, `as.Date`, `ordered`

Record: `as.POSIXct`

Data frame: arguably `table` is a generalization

Scalar: `lm`, `ecdf`, `I`

2.

`lm` is a list of information related to the model. First, the call should be captured and stored. If requested, data like model weights or the QR matrix used in model fitting should be retained. Once the model is fitted, the following should be recorded for each observation:

- Observed response
- Fitted value
- Residual

Beyond that, the model matrix, and some details like factor contrasts and unique `x` levels, have to be stored as well.

Inheritance

```
new_secret <- function(x, ..., class = character()) {
  stopifnot(is.double(x))

  structure(
    x,
    ...,
    class = c(class, "secret")
  )
}
new_supersecret <- function(x) {
  new_secret(x, class = "supersecret")
}

print.supersecret <- function(x, ...) {
  print(rep("xxxxx", length(x)))
  invisible(x)
}

x2 <- new_supersecret(c(15, 1, 456))
x2
```

```
[1] "xxxxx" "xxxxx" "xxxxx"
```

```
x <- structure(1:10, class = "test")
t(x)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    2    3    4    5    6    7    8    9    10
attr(,"class")
[1] "test"
```

1.

[.Date delegates to NextMethod, passing the most specific class of the argument x as determined by oldClass. This means a Date subclass foo is dispatched to [.foo.

```
library(sloop)
s3_methods_generic("[")
```

generic	class	visible	source
[	acf	FALSE	registered S3method

generic	class	visible	source
[	AsIs	TRUE	base
[	bibentry	FALSE	registered S3method
[	check_details_changes	FALSE	registered S3method
[	cli_doc	FALSE	registered S3method
[	data.frame	TRUE	base
[	Date	TRUE	base
[	difftime	TRUE	base
[	Dlist	TRUE	base
[	DLLInfoList	TRUE	base
[	factor	TRUE	base
[	formula	FALSE	registered S3method
[	fs_bytes	FALSE	registered S3method
[	fs_path	FALSE	registered S3method
[	fs_perms	FALSE	registered S3method
[	fseq	FALSE	registered S3method
[	GenericSummary	FALSE	registered S3method
[	getAnywhere	FALSE	registered S3method
[	glue	FALSE	registered S3method
[	hexmode	TRUE	base
[	listof	TRUE	base
[	news_db	FALSE	registered S3method
[	noquote	TRUE	base
[	numeric_version	TRUE	base
[	octmode	TRUE	base
[	person	FALSE	registered S3method
[	POSIXct	TRUE	base
[	POSIXlt	TRUE	base
[	quosure	FALSE	registered S3method
[	quosures	FALSE	registered S3method
[	raster	FALSE	registered S3method
[	rlang_ctxt_pronoun	FALSE	registered S3method
[	rlang_data_pronoun	FALSE	registered S3method
[	rlang_envs	FALSE	registered S3method
[	rlang:::list_of_conditions	FALSE	registered S3method
[	rlib_bytes	FALSE	registered S3method
[	roman	FALSE	registered S3method
[	simple.list	TRUE	base
[	table	TRUE	base
[	tbl_df	FALSE	registered S3method
[	terms	FALSE	registered S3method
[	ts	FALSE	registered S3method
[	tskernel	FALSE	registered S3method
[	vctrs_rcrd	FALSE	registered S3method
[	vctrs_sclr	FALSE	registered S3method
[	vctrs_unspecified	FALSE	registered S3method
[	vctrs_vctr	FALSE	registered S3method
[	warnings	TRUE	base

2.

It looks like `POSIXct` methods are more verbose and do more elaborate checking involving timezones. `print` is naturally the same for both.

3.

`generic2` dispatches on the class of `x`. `generic2.b` is called first, the class reassignment is ignored, then `NextMethod` dispatches to `generic.a2`.

```
generic2 <- function(x) UseMethod("generic2")
generic2.a1 <- function(x) "a1"
generic2.a2 <- function(x) "a2"
generic2.b <- function(x) {
  class(x) <- "a1"
  NextMethod()
}

generic2(structure(list(), class = c("b", "a2")))
```

```
[1] "a2"
```

Dispatch Details

1.

Internal methods dispatch only on implicit class (what `1:5` has), not explicit class set by `class`.

2.

`Math.data.frame` checks types before using `lapply` to compute the operation. `Math.difftime` records units before forwarding to `NextMethod`. The `factor` and `PosixLT` methods warn the user that calling them is nonsensical.

```
sloop::s3_methods_generic("Math")
```

generic	class	visible	source
Math	data.frame	TRUE	base
Math	Date	TRUE	base
Math	difftime	TRUE	base
Math	factor	TRUE	base
Math	POSIXt	TRUE	base
Math	quosure	FALSE	registered S3method
Math	vctrs_sclr	FALSE	registered S3method
Math	vctrs_vctr	FALSE	registered S3method

3.

It tracks units and throws an error for unsupported operations.