Extending atable

Alan Haynes, Armin Ströbel

August 29, 2024

Contents

‘atable‘ has been designed for flexibility in mind. If you don’t like the defaults, you can define your own summary statistics, tests and effect measures. You can even define your own methods for classes not supported natively. This vignette gives some details and examples on how to go about these tasks.

In this vignette we will use the ‘mtcars‘ dataset as an example. Load it and prepare factors and other variables. We also set the format_to option to ‘Latex‘ for nicer printing in the vignette.

```
data(mtcars)
# factors
mtcars$am <- factor(mtcars$am, c(0, 1), c("Automatic", "Manual"))
mtcars$vs <- factor(mtcars$vs, c(0, 1), c("V-shaped", "straight"))
# ordered
mtcars$cyl <- ordered(mtcars$cyl)
# set format_to
atable_options(format_to = "Latex")
```

```
Hmisc::latex(atable(vs + cyl + hp + disp ~ am, mtcars),
             file = "",
             title = "",
             rowname = NULL,
             table.env = FALSE)
```

Group	Automatic	Manual	p	stat	Effect Size (CI)
Observations					
	19	13			
vs					
V-shaped	63% (12)	46% (6)	0.56	0.35	2 (0.38; 11)
straight	37% (7)	54% (7)			
missing	0% (0)	0% (0)			
cyl					
4	16% (3)	62% (8)	NaN	NaN	0.57 (0.18; 0.81)
6	21% (4)	23% (3)			
8	63% (12)	15% (2)			
missing	0% (0)	0% (0)			
hp					
Mean (SD)	160 (54)	127 (84)	0.023	0.51	0.49 (-0.25; 1.2)
valid (missing)	19 (0)	13 (0)			
disp					
Mean (SD)	290 (110)	144 (87)	<0.001	0.69	1.4 (0.62; 2.3)
valid (missing)	19 (0)	13 (0)			

1 Methods for other classes

'atable' only support numeric, factor and ordered classes by default. If you want to use unsupported classes, e.g. 'Date' or 'surv', you can define methods for them reasonably easily.

1.1 Example methods for 'Date's

There are no methods for 'Date's in 'atable'. We can define them easily though. If we want the minimum, median and maximum dates, we could define the statistics function as follows. The class of the output here is important - it is used to choose the appropriate formatting function.

```
statistics.Date <- function(x, ...){
  out <- list(min = min(x, na.rm = TRUE),
              med = median(x, na.rm = TRUE),
              max = max(x, na.rm = TRUE))
  class(out) <- c("statistics_date", class(out))
  out
}
```

The suitable formatting function for that might be the following to put minimum and maximum on one line followed by the median on the next. The factor is required to avoid reordering the rows.

```
format_statistics.statistics_date <- function(x, ...){
  z <- c("Min ; Max", "Median")
  out <- data.frame(tag = factor(z, z),
                    value = c(paste(x$min, x$max, sep = " ; "),
                              as.character(x$med)),
                    stringsAsFactors = FALSE)
  return(out)
}
```

```
# add a date variable to mtcars
mtcars$date <- as.Date(runif(nrow(mtcars), 0, 365*10), "1990-01-01")

Hmisc::latex(atable(mtcars, "date"),
             file = "",
             title = "",
             rowname = NULL,
             table.env = FALSE)
```

Group	value
Observations	
	32
date	
Min ; Max	1991-03-01 ; 1999-03-05
Median	1994-05-24

If comparing two or more groups, then suitable ‘two_sample_htest‘ and ‘multi_sample_htest‘ functions should also be defined.

1.2 Example methods for ‘surv‘ objects

Probably more useful than the ‘Date‘ methods would be ‘surv‘ objects, as defined by the ‘survival‘ package.

First we add a ‘surv‘ object to ‘mtcars‘ by creating an observation time point approximately 10 years after the date we defined previously. We then calculate the time between these two time points and define an indicator whether an event occured, in this case the car no longer being road-worthy.

```
# add some survival data (use 'date' as the timepoint)
if (requireNamespace("survival", quietly = TRUE)) {
  mtcars$date2 <- mtcars$date + round(rnorm(nrow(mtcars), 10, 4)) # end date
  mtcars$time <- as.numeric(mtcars$date2 - mtcars$date) # time
  mtcars$not_road_worthy <- rbinom(nrow(mtcars), 1, .2) # 'survived'?
```

```
  mtcars$surv <- with(mtcars, survival::Surv(time, not_road_worthy))
} else {
  ## do nothing
}
```

Now we need the appropriate methods for 'atable'. Mean survival time is a common choice for time-to-event analyses. Similarly, the Mantel-Haenszel test is a used to compare two curves.

```
if (requireNamespace("survival", quietly = TRUE)) {
  # statistics function
  statistics.Surv <- function(x, ...){
    survfit_object <- survival::survfit(x ~ 1)
    # copied from survival::print.survfit
    out <- survival:::survmean(survfit_object, rmean = "common")
    return(list(mean_survival_time = out$matrix["*rmean"],
                SE = out$matrix["*se(rmean)"]))
  }
  # testing function
  two_sample_htest.Surv <- function(value, group, ...){
    survdiff_result <- survival::survdiff(value~group, rho=0)
    # copy from survival::print.survdiff
    etmp <- survdiff_result$exp
    df <- (sum(1 * (etmp > 0))) - 1
    p <- 1 - stats::pchisq(survdiff_result$chisq, df)
    return(list(p = p,stat = survdiff_result$chisq))
  }
} else {
  ## do nothing
}
```

We can then use them with the variables we defined in mtcars...

```
if (requireNamespace("survival", quietly = TRUE)) {
  Hmisc::latex(atable(surv ~ am, mtcars),
               file = "",
               title = "",
               rowname = NULL,
               table.env = FALSE)
} else {
  ## do nothing
}
```

Group	Automatic	Manual	p	stat
Observations				
	19	13		
surv				
mean_survival_time	NA	NA	0.52	0.41
SE	NA	NA		

An appropriate formatting function could be defined as above for ‘Date‘s.

2 Different statistics for variables of a single class

In the ‘mtcars‘ example, suppose we want to summarize ‘hp‘ by mean and SD and ‘disp‘ by median and quartiles. Mean and SD are the default statistics for numeric variables in ‘atable‘ so we only have to worry about ‘disp‘. To accomplish this, we can use the same method as we used above for ‘Date‘ variables - we will define new functions for a new class. We will assign the new class, which we will call ‘numeric2‘, to ‘disp‘ and define new functions to handle it.

```
# add numeric2 to the class of disp
class(mtcars$disp) <- c("numeric2", class(mtcars$disp))

# subsetting function for numeric2 class
'[.numeric2' <- function(x, i, j, ...){
  y <- unclass(x)[i, ...]
  class(y) <- c("numeric2", class(y))
  y
}
```

The subsetting function is used to retain the class of the variable (otherwise it reverts to a numeric in this case). We didn’t need to do this above as the relevant function for the ‘Date‘ and ‘surv‘ classes already exist.

Next we define functions to calculate the statistics that we want to use. These both have to return named lists.

```
# statistics function
statistics.numeric2 <- function(x, ...){
  statistics_out <- list(Median = median(x, na.rm = TRUE),
                         p25 = quantile(x, 0.25, na.rm = TRUE),
                         p75 = quantile(x, 0.75, na.rm = TRUE))
  class(statistics_out) <- c("statistics_numeric2", class(statistics_out))
  # We will need this new class later to specify the format
  return(statistics_out)
}
```

```r
# testing function
two_sample_htest.numeric2 <- function(value, group, ...){
  d <- data.frame(value = value, group = group)
  test_out <- stats::wilcox.test(value ~ group, d)
  return(test_out)
}
```

Now we can test to see if our new class has been identified and used correctly.

```r
Hmisc::latex(atable(vs + cyl + hp + disp ~ am, mtcars),
             file = "",
             title = "",
             rowname = NULL,
             table.env = FALSE)
```

Group	Automatic	Manual	p	stat	Effect Size (CI)
Observations					
	19	13			
vs					
V-shaped	63% (12)	46% (6)	0.56	0.35	2 (0.38; 11)
straight	37% (7)	54% (7)			
missing	0% (0)	0% (0)			
cyl					
4	16% (3)	62% (8)	NaN	NaN	0.57 (0.18; 0.81)
6	21% (4)	23% (3)			
8	63% (12)	15% (2)			
missing	0% (0)	0% (0)			
hp					
Mean (SD)	160 (54)	127 (84)	0.023	0.51	0.49 (-0.25; 1.2)
valid (missing)	19 (0)	13 (0)			
disp					
Median	276	120	<0.001		
p25	196	79			
p75	360	160			

We probably don't want to have the quartiles beneath the median so we can also define a formatting function. The 'format_statistics' function should return a dataframe with variable tag (as a factor to retain ordering) and value (most likely a string). The class is no longer 'numeric2' but 'statistics_numeric2' as defined in the 'statistics.numeric2' function.

```r
format_statistics.statistics_numeric2 <- function(x, ...){
  out <- data.frame(
```

```
    tag = factor(c("Median [Quartiles]")),
    value = sprintf("%2.1f [%2.1f ; %2.1f]", x$Median, x$p25, x$p75),
    stringsAsFactors = FALSE)
  return(out)
}
```

```
Hmisc::latex(atable(vs + cyl + hp + disp ~ am, mtcars),
             file = "",
             title = "",
             rowname = NULL,
             table.env = FALSE)
```

Group	Automatic	Manual	p	stat	Effect Size (CI)
Observations					
	19	13			
vs					
V-shaped	63% (12)	46% (6)	0.56	0.35	2 (0.38; 11)
straight	37% (7)	54% (7)			
missing	0% (0)	0% (0)			
cyl					
4	16% (3)	62% (8)	NaN	NaN	0.57 (0.18; 0.81)
6	21% (4)	23% (3)			
8	63% (12)	15% (2)			
missing	0% (0)	0% (0)			
hp					
Mean (SD)	160 (54)	127 (84)	0.023	0.51	0.49 (-0.25; 1.2)
valid (missing)	19 (0)	13 (0)			
disp					
Median [Quartiles]	275.8 [196.3 ; 360.0]	120.3 [79.0 ; 160.0]	<0.001		