R programming: generic programming, tidy evaluation

2024-09-02

- **M1 MIDS/MFA**
- **Université Paris Cité**
- Année 2024-2025
- Course Homepage

- Moodle

```
stopifnot(
  require(broom),
  require(devtools),
  require(ggforce),
  require(ggfortify),
  require(glue),
  require(Hmisc),
  require(lobstr),
  require(patchwork),
  require(rlang),
  require(skimr),
  require(testthat),
  require(tidyverse),
  require(usethis)
)

tidymodels::tidymodels_prefer(quiet = TRUE)

old_theme <-theme_set(
  theme_minimal(base_size=9,
                base_family = "Helvetica")
)
```

```
gc <- options(ggplot2.discrete.colour="viridis")
gc <- options(ggplot2.discrete.fill="viridis")
gc <- options(ggplot2.continuous.fill="viridis")
gc <- options(ggplot2.continuous.colour="viridis")
```

Objectives

Generics and S3 classes

OO in Advanced R Programming

Programming with `dplyr` and `ggplot2`

We aim at programming a function that takes as input a dataframe `df`, a column name `col`, and that, depending on the type of the column denoted by `col`, plots a histogram (for numerical column), a barplot (for factors), or raise an error of the column is neither categorical, nor numerical.

The function should return a ggplot object.

Here is a first attempt.

```
tb <- tibble(
  col_num = rnorm(100),
  col_fac = as_factor(sample(letters, 100, replace = T)),
  col_ts = Sys.time() + duration(sample(1:20, 100, replace=T),units="days")
)

tb |>
  head()
```

```
# A tibble: 6 x 3
  col_num col_fac col_ts
    <dbl> <fct>   <dttm>
1 -0.468  c       2024-09-06 09:01:11
2 -0.406  e       2024-09-10 09:01:11
3  0.979  n       2024-09-09 09:01:11
4 -0.0479 z       2024-09-06 09:01:11
5 -0.886  e       2024-09-06 09:01:11
6 -0.570  q       2024-09-06 09:01:11
```

```
gg_obj <-  function(df, col){

  vct <- df[[col]]
  tp <- class(vct)

  if (tp != "numeric" & tp !="factor") {
    stop(paste0(col, " is of wrong type!"))
  }

  p <- ggplot(df) +
    aes(x=.data[[col]])

  if (tp=="numeric") {
```

```
    p <- p + geom_histogram()
  } else {
    p <- p + geom_bar()
  }

  p
}
```

- pass more optional arguments
- avoid quoting the column name

```
gg_obj_2 <-  function(df, col, ...){

  vct <- pull(df, {{col}})
  tp <- class(vct)[1]

  if (tp != "numeric" & tp !="factor") {
    stop("column is of wrong type!")
    return
  }

  p <- ggplot(df) +
    aes(x={{col}})

  if (tp=="numeric") {
    p <- p + geom_histogram(...)
  } else {
    p <- p + geom_bar(...)
  }

  p
}
```

Inside `lm()`

> **Question**
>
> In classes like `lm`, `prcomp`, … we have a member called `call`. What does it represent? How is it constructed?
> Read the code of `lm`.

```
> 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)
    mf <- mf[c(1L, m)]
    mf$drop.unused.levels <- TRUE
    mf[[1L]] <- quote(stats::model.frame)
    mf <- eval(mf, parent.frame())
    if (method == "model.frame")
        return(mf)
    else if (method != "qr")
        warning(gettextf("method = '%s' is not supported. Using 'qr'",
            method), domain = NA)
    mt <- attr(mf, "terms")
    y <- model.response(mf, "numeric")
    w <- as.vector(model.weights(mf))
    if (!is.null(w) && !is.numeric(w))
        stop("'weights' must be a numeric vector")
    offset <- model.offset(mf)
    mlm <- is.matrix(y)
    ny <- if (mlm)
        nrow(y)
    else length(y)
    if (!is.null(offset)) {
        if (!mlm)
            offset <- as.vector(offset)
        if (NROW(offset) != ny)
            stop(gettextf("number of offsets is %d, should equal %d (number of observations)",
                NROW(offset), ny), domain = NA)
    }
    if (is.empty.model(mt)) {
        x <- NULL
        z <- list(coefficients = if (mlm) matrix(NA_real_, 0,
            ncol(y)) else numeric(),
                  residuals = y,
                  fitted.values = 0 * y,
                  weights = w,
                  rank = 0L,
                  df.residual = if (!is.null(w)) sum(w != 0) else ny
              )
        if (!is.null(offset)) {
            z$fitted.values <- offset
            z$residuals <- y - offset
        }
    }
```

```
    else {
        x <- model.matrix(mt, mf, contrasts)
        z <- if (is.null(w))
            lm.fit(x, y, offset = offset, singular.ok = singular.ok,
                ...)
        else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,
            ...)
    }
    class(z) <- c(if (mlm) "mlm", "lm")
    z$na.action <- attr(mf, "na.action")
    z$offset <- offset
    z$contrasts <- attr(x, "contrasts")
    z$xlevels <- .getXlevels(mt, mf)
    z$call <- cl
    z$terms <- mt
    if (model)
        z$model <- mf
    if (ret.x)
        z$x <- x
    if (ret.y)
        z$y <- y
    if (!qr)
        z$qr <- NULL
    z
}
<bytecode: 0x55564224e930>
<environment: namespace:stats>
```

Data masking and environments

> **Question**

Tidy evaluation

> **Question**
>
> What is *quasi-quotation*?
> Keep the `rlang` cheatsheet around.

> **Question**
>
> Explain the difference between an *expression* and a *quosure*

Question

References