Software quality analysis of fourteen hydrological models

Arnald Puy

Contents

1 Preliminary functions

```
#   PRELIMINARY FUNCTIONS ####################################################

sensobol::load_packages(c("data.table", "tidyverse", "openxlsx", "scales",
                          "cowplot", "readxl", "ggrepel", "tidytext", "here"))

# Create custom theme -----------------------------------------------------------

theme_AP <- function() {
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.background = element_rect(fill = "transparent", color = NA),
          legend.key = element_rect(fill = "transparent", color = NA),
          strip.background = element_rect(fill = "white"),
          legend.text = element_text(size = 7.3),
          axis.title = element_text(size = 10),
          legend.key.width = unit(0.4, "cm"),
          legend.key.height = unit(0.4, "cm"),
          legend.key.spacing.y = unit(0, "lines"),
          legend.box.spacing = unit(0, "pt"),
          legend.title = element_text(size = 7.3),
          axis.text.x = element_text(size = 7),
          axis.text.y = element_text(size = 7),
          axis.title.x = element_text(size = 7.3),
          axis.title.y = element_text(size = 7.3),
          plot.title = element_text(size = 8),
          strip.text.x = element_text(size = 7.4),
          strip.text.y = element_text(size = 7.4))
}

# Select color palette ---------------------------------------------------------

color_languages <- c("fortran" = "steelblue", "python" = "lightgreen")

# Source all .R files in the "functions" folder --------------------------------

r_functions <- list.files(path = here("functions"),
                          pattern = "\\.R$", full.names = TRUE)

lapply(r_functions, source)
```

```
## [[1]]
## [[1]]$value
## function (plot, legend = NULL)
## {
##     if (is.ggplot(plot)) {
```

```
##         gt <- ggplotGrob(plot)
##     }
##     else {
##         if (is.grob(plot)) {
##             gt <- plot
##         }
##         else {
##             stop("Plot object is neither a ggplot nor a grob.")
##         }
##     }
##     pattern <- "guide-box"
##     if (!is.null(legend)) {
##         pattern <- paste0(pattern, "-", legend)
##     }
##     indices <- grep(pattern, gt$layout$name)
##     not_empty <- !vapply(gt$grobs[indices], inherits, what = "zeroGrob",
##         FUN.VALUE = logical(1))
##     indices <- indices[not_empty]
##     if (length(indices) > 0) {
##         return(gt$grobs[[indices[1]]])
##     }
##     return(NULL)
## }
##
## [[1]]$visible
## [1] FALSE
##
##
## [[2]]
## [[2]]$value
## function (nodes_vec)
## {
##     node_df$risk_node[match(nodes_vec, node_df$name)]
## }
##
## [[2]]$visible
## [1] FALSE
##
##
## [[3]]
## [[3]]$value
## function (path_vertices)
## {
##     idx <- as.integer(path_vertices)
##     rn <- node_df$risk_node[idx]
##     tibble(path_nodes = list(node_df$name[idx]), path_str = paste(node_df$name[idx],
##         collapse = " → "), hops = length(idx) - 1, risk_sum = sum(rn),
##         risk_mean = mean(rn))
## }
##
## [[3]]$visible
## [1] FALSE
```

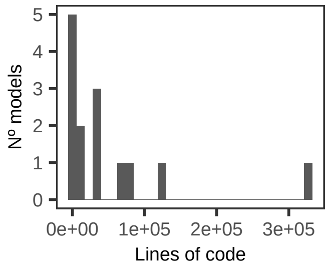
2 Results

```r
# READ IN DATASET #############################################################

# Get name of sheets -----------------------------------------------------------

sheets <- excel_sheets("./datasets/results_sqa.xlsx")

# Read all sheets --------------------------------------------------------------

dt <- lapply(sheets, function(x) data.table(read_excel("./datasets/results_sqa.xlsx",
                                                       sheet = x)))

# Name the slots ---------------------------------------------------------------

names(dt) <- sheets
```

2.1 Descriptive statistics

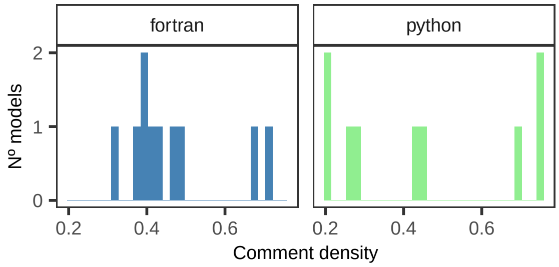
```r
# PLOT LINES OF CODE ###########################################################

plot_lines_code <- dt$descriptive_stats[, .(total_lines_code = sum(lines_code)), model] %>%
  ggplot(., aes(total_lines_code)) +
  geom_histogram() +
  labs(x = "Lines of code", y = "Nº models") +
  theme_AP()

plot_lines_code
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



```r
# PLOT COMMENT DENSITY #########################################################

plot_comment_density <- dt$descriptive_stats[, .(total_lines_code = sum(lines_code),
      total_lines_comments = sum(lines_comments)), .(model, language)] %>%
  .[, comment_density:= total_lines_comments / total_lines_code] %>%
  ggplot(., aes(comment_density, fill = language)) +
  geom_histogram() +
  facet_wrap(~language) +
  scale_y_continuous(breaks = breaks_pretty(n = 3)) +
  scale_fill_manual(values = color_languages) +
  labs(x = "Comment density", y = "Nº models") +
  theme_AP() +
  theme(legend.position = "none")

plot_comment_density
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

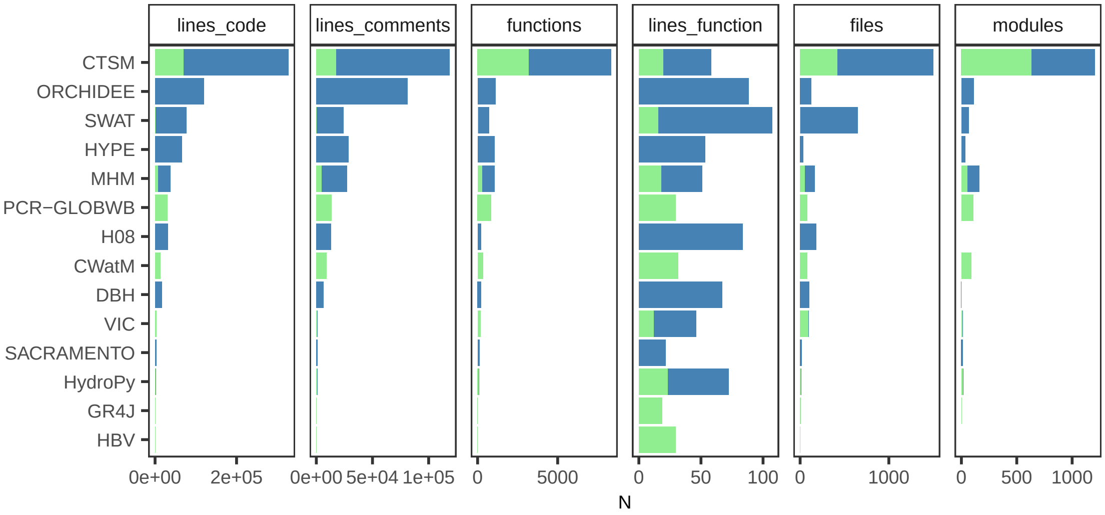


```
# PLOT PER MODEL ###############################################################

# Sort by model ----------------------------------------------------------------

model_ordered <- dt$descriptive_stats[, sum(lines), model] %>%
  .[order(V1)]

# Print ------------------------------------------------------------------------

model_ordered
```

```
##          model    V1
##         <char> <num>
##  1:        HBV   180
##  2:       GR4J   423
##  3:    HydroPy  3739
##  4: SACRAMENTO  5294
##  5:        VIC  5952
##  6:        DBH 24334
##  7:      CWatM 27745
##  8:        H08 42917
##  9: PCR-GLOBWB 52686
## 10:        MHM 76286
## 11:       HYPE 89137
## 12:       SWAT 99976
## 13:   ORCHIDEE 211871
## 14:       CTSM 491592
```

```r
# Extract column names ----------------------------------------------------------

col_names <- colnames(dt$descriptive_stats)

# Order facets ------------------------------------------------------------------

facet_order <- c("lines", "lines_code", "lines_comments", "functions",
                 "lines_function", "files", "modules")


# Plot --------------------------------------------------------------------------

plot_per_model <- melt(dt$descriptive_stats, measure.vars = col_names[-c(1, length(col_names))]
  .[, variable:= factor(variable, levels = facet_order)] %>%
  .[, model:= factor(model, levels = model_ordered[, model])] %>%
  .[!variable == "lines"] %>%
  ggplot(., aes(model, value, fill = language)) +
  geom_col() +
  coord_flip() +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  scale_fill_manual(values = color_languages) +
  facet_wrap(~ variable, ncol = 7, scales = "free_x") +
  labs(x = "", y = "N") +
  theme_AP() +
  theme(legend.position = "none")

plot_per_model
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale range
## (`geom_col()`).
```



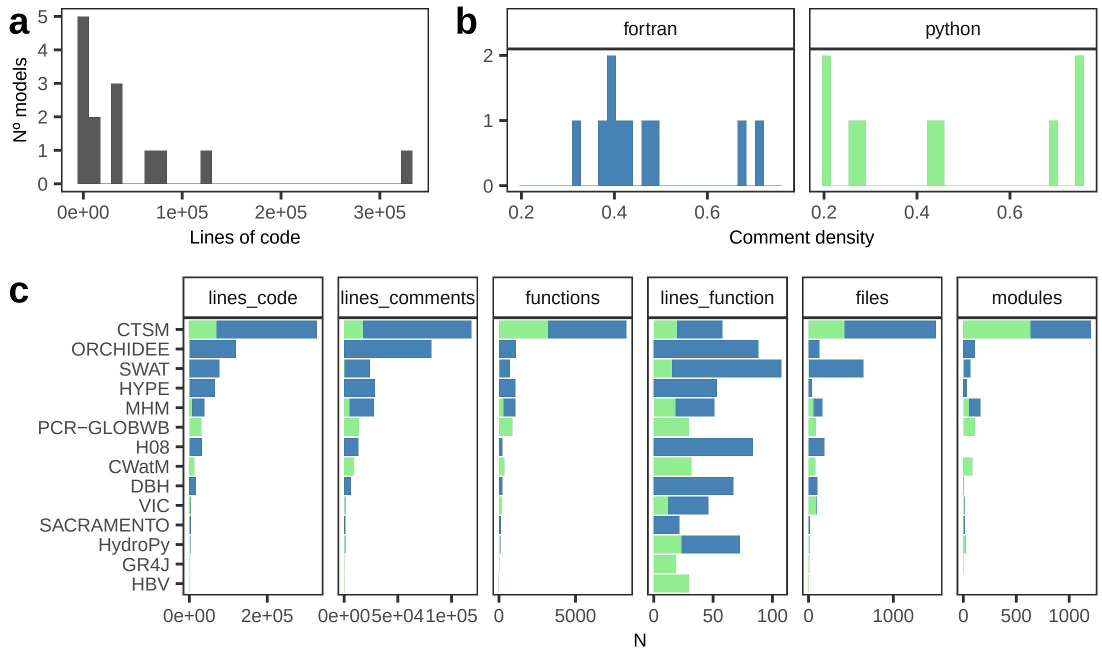
```
# MERGE PLOTS ##################################################################

top <- plot_grid(plot_lines_code, plot_comment_density + labs(x = "Comment density", y  = ""),
          labels = "auto", rel_widths = c(0.4, 0.6))
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

```
p1 <- plot_grid(top, plot_per_model, ncol = 1, labels = c("", "c"), rel_heights = c(0.4, 0.6))
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale range
## (`geom_col()`).
```

```
p1
```



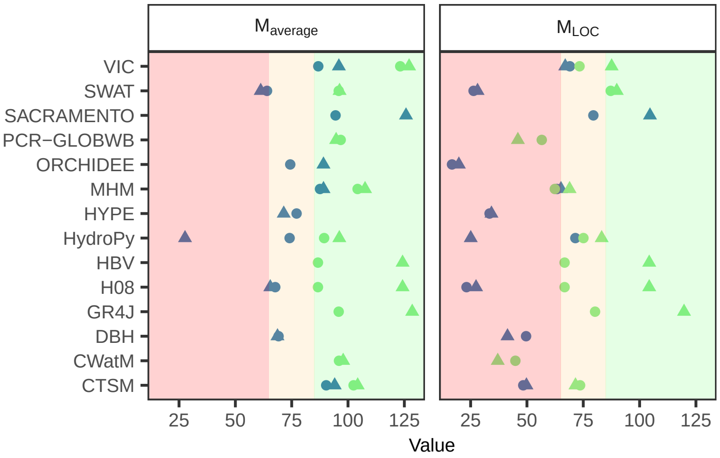
2.2 Maintainability index

```
# CALCULATE INTERPRETATIBILITY OF MAINTAINABILITY INDEX 3######################

# Define vector of interpretation ---------------------------------------------

vec_interpretation <- c("low", "moderate", "high")

# Calculate -------------------------------------------------------------------

dt$maintainability_index %>%
  melt(., measure.vars = c("M_loc", "M_average")) %>%
  .[, interpretativity:= ifelse(value > 85, vec_interpretation[3],
                                ifelse(value <=85 & value >= 65, vec_interpretation[2],
                                       vec_interpretation[1]))] %>%
  .[, .N, .(language, interpretativity, variable)] %>%
  dcast(., variable + language ~ interpretativity, value.var = "N") %>%
  .[, total:= rowSums(.SD, na.rm = TRUE), .SDcols = vec_interpretation] %>%
  .[, paste(vec_interpretation, "prop", sep = "_"):= lapply(.SD, function(x)
    x / total), .SDcols = vec_interpretation] %>%
  print()
```

```
## Key: <variable, language>
##     variable language  high   low moderate total low_prop moderate_prop
##       <fctr>   <char> <int> <int>    <int> <num>    <num>         <num>
## 1:     M_loc  fortran     1    14        5    20     0.70          0.25
## 2:     M_loc   python     6     5        9    20     0.25          0.45
## 3: M_average  fortran     9     3        8    20     0.15          0.40
## 4: M_average   python    20    NA       NA    20       NA            NA
##    high_prop
##        <num>
## 1:      0.05
## 2:      0.30
## 3:      0.45
## 4:      1.00
```

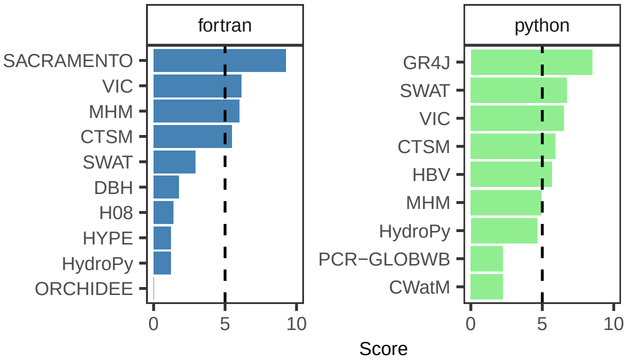
By combining the classic and extended versions of the maintainability index, our analysis reveals differences between Fortran and Python implementations. Using the weighted measure (M_{LOC}), 70% of Fortran code falls into the "low" maintainability category, compared with only 15% when using the unweighted average (M_{average}). This discrepancy indicates that a few complex, poorly maintainable routines dominate the overall profile of the Fortran codebase. In contrast, Python routines present a more favorable profile: 27% achieve high maintainability under M_{LOC}, and all are classified as "highly maintainable" under M_{average}.

```
# PLOT MAINTAINABILITY INDEX ###################################################

plot_maintainability_index <- dt$maintainability_index %>%
  melt(., measure.vars = c("M_loc", "M_average")) %>%
  .[, variable:= factor(variable, levels = c("M_average", "M_loc"))] %>%
  ggplot(., aes(model, value, color = language, shape = type)) +
  geom_point() +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = 65,
           fill = "red", alpha = 0.18) +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = 65, ymax = 85,
           fill = "orange", alpha = 0.1) +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = 85, ymax = Inf,
           fill = "green", alpha = 0.1) +
  facet_wrap(~variable, labeller = as_labeller(c(M_loc = "M[LOC]",
                                                 M_average = "M[average]"),
                                               default = label_parsed)) +
  labs(x = "", y = "Value") +
  scale_color_manual(values = color_languages, guide = "none") +
  theme_AP() +
  theme(legend.position = "none") +
  coord_flip()

plot_maintainability_index
```

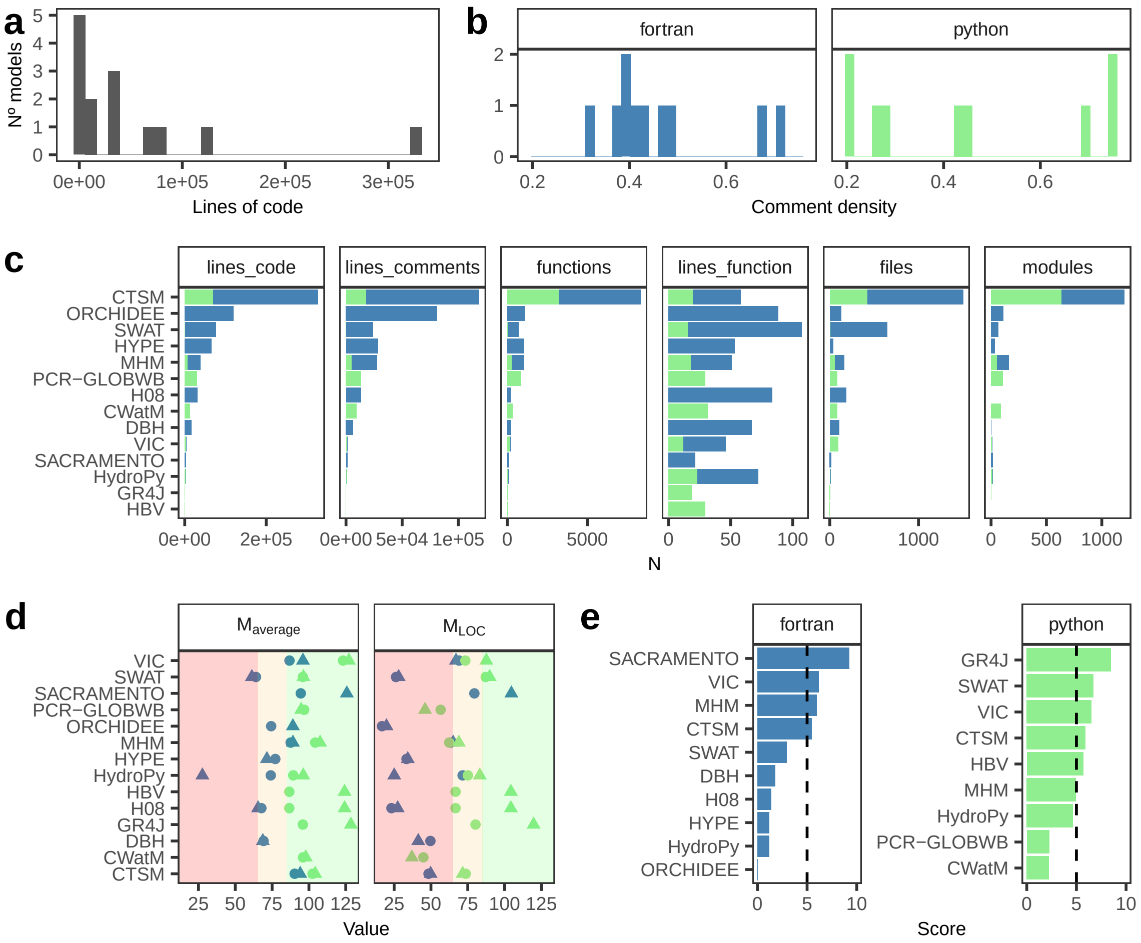


2.3 Score

```
# PLOT SCORE ###################################################################

plot_score <- dt$score %>%
  ggplot(aes(x = reorder_within(model, score, language), y = score, fill = language)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ language, scales = "free_y") +
  labs(x = "", y = "Score") +
  scale_fill_manual(values = color_languages) +
  geom_hline(yintercept = 5, lty = 2) +
  scale_x_reordered() +
  scale_y_continuous(limits = c(0, 10), breaks = c(0, 5, 10)) +
  coord_flip() +
  theme_AP() +
  theme(legend.position = "none")

plot_score
```



```
# MERGE PLOTS ##################################################################

bottom <- plot_grid(plot_maintainability_index, plot_score, ncol = 2, labels = c("d", "e"))

plot_grid(p1, bottom, ncol = 1, rel_heights = c(0.62, 0.38))
```



2.4 Metrics at the function level

```
# METRICS AT THE FILE AND FUNCTION LEVEL #######################################

folder <- "./datasets/results_function"

# Get names of files -----------------------------------------------------------

csv_files <- list.files(path = folder, pattern = "\\.csv$", full.names = TRUE)

# Split into file_metrics and func_metrics -------------------------------------

file_metric_files <- grep("file_metrics", csv_files, value = TRUE)
func_metric_files <- grep("func_metrics", csv_files, value = TRUE)
```

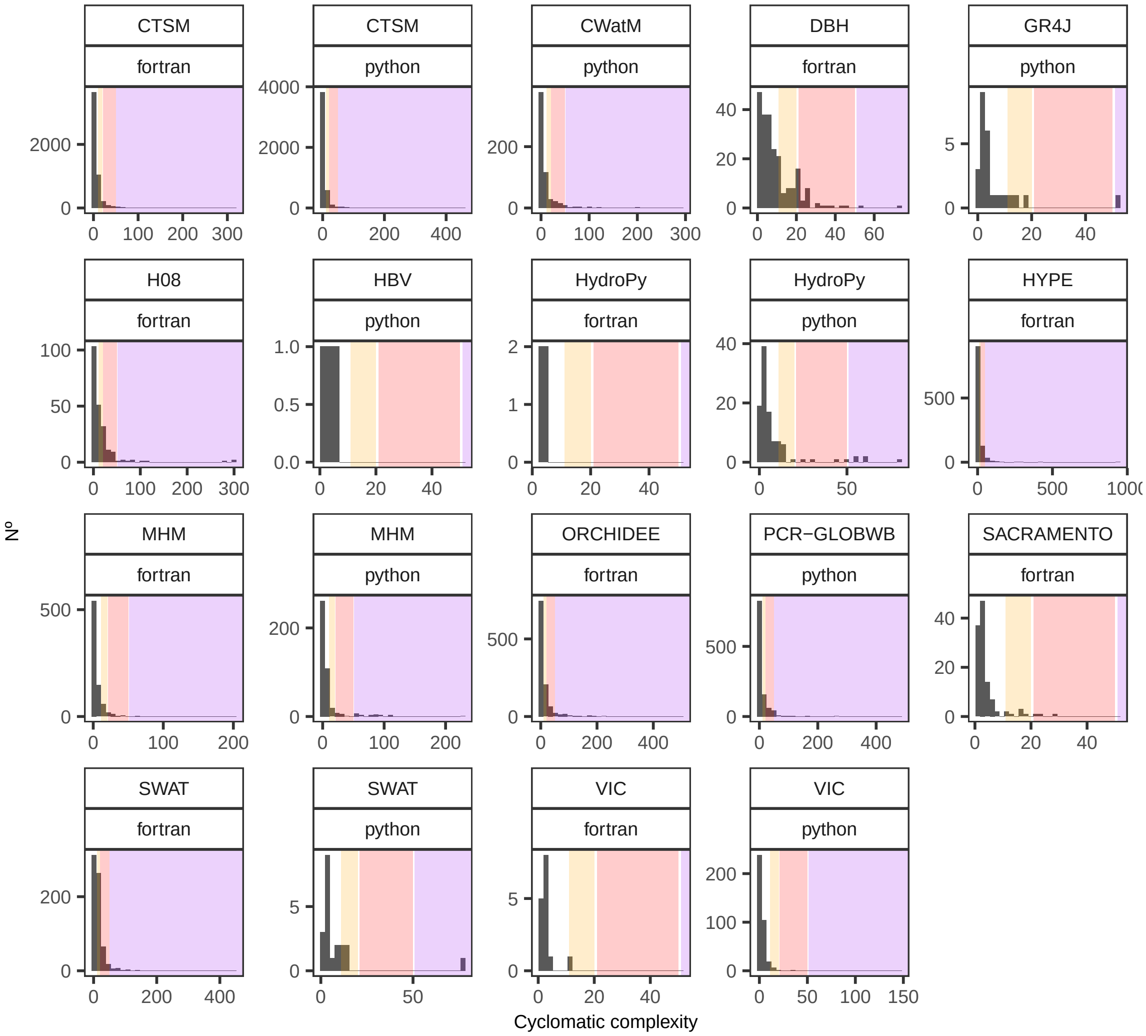
```r
# Build one named list ---------------------------------------------------------

list_metrics <- list(file_metrics = setNames(lapply(file_metric_files, fread),
                                             basename(file_metric_files)),
                     func_metrics = setNames(lapply(func_metric_files, fread),
                                             basename(func_metric_files)))

# Create function to combine files ---------------------------------------------

make_combined <- function(subset_list, pattern) {
  rbindlist(subset_list[grep(pattern, names(subset_list))], idcol = "source_file")
}

# Combine files ----------------------------------------------------------------

metrics_combined <- list(file_fortran = make_combined(list_metrics$file_metrics,  "fortran"),
                         file_python = make_combined(list_metrics$file_metrics,  "python"),
                         func_fortran = make_combined(list_metrics$func_metrics, "fortran"),
                         func_python = make_combined(list_metrics$func_metrics, "python"))

# Functions to extract name of model and language from file --------------------

extract_model <- function(x)
  sub("^(file|func)_metrics_\\d+_([A-Za-z0-9-]+)_(fortran|python).*", "\\2", x)

extract_lang  <- function(x)
  sub("^(file|func)_metrics_\\d+_([A-Za-z0-9-]+)_(fortran|python).*", "\\3", x)

# Extract name of model and language -------------------------------------------

metrics_combined <- lapply(metrics_combined, function(dt) {
  dt[, source_file:= sub("\\.csv$", "", basename(source_file))]
  dt[, model:= extract_model(source_file)]
  dt[, language:= extract_lang(source_file)]
  dt
})

# Add column of complexity category --------------------------------------------

metrics_combined <- lapply(names(metrics_combined), function(nm) {
  dt <- as.data.table(metrics_combined[[nm]])
  if (grepl("^func_", nm) && "cyclomatic_complexity" %in% names(dt)) {
    dt[, complexity_category := cut(
      cyclomatic_complexity,
      breaks = c(-Inf, 10, 20, 50, Inf),
      labels = c("b1","b2","b3","b4")
    )]
  }
  dt
}) |> setNames(names(metrics_combined))

# Define labels ----------------------------------------------------------------

lab_expr <- c(
  b1 = expression(C %in% "(" * 0 * ", 10" * "]"),
  b2 = expression(C %in% "(" * 10 * ", 20" * "]"),
  b3 = expression(C %in% "(" * 20 * ", 50" * "]"),
  b4 = expression(C %in% "(" * 50 * ", " * infinity * ")")
)

# EXPORT DATA TO .CSV ##########################################################

# set output folder inside "datasets" ------------------------------------------

outdir <- file.path("datasets", "merged_results")

# write each slot to its own CSV -----------------------------------------------

lapply(names(metrics_combined), function(nm) {
  out_file <- file.path(outdir, paste0(nm, ".csv"))
  fwrite(metrics_combined[[nm]], out_file)
})
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```

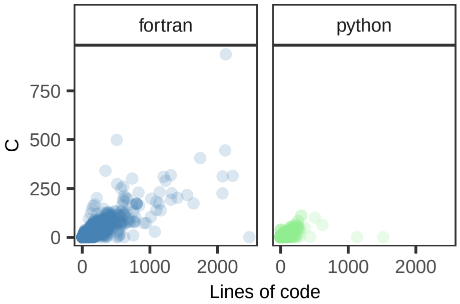
```
# Define vector of interest for cyclomatic complexity

vector_cyclomatic <- c("SUBROUTINE", "FUNCTION")
```

```
# PLOT #########################################################################

# Cyclomatic complexity at the model level -------------------------------------

metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(cyclomatic_complexity, model, language)]) %>%
  rbindlist() %>%
  ggplot(., aes(cyclomatic_complexity)) +
  geom_histogram() +
  annotate("rect",
           xmin = 11, xmax = 20,
           ymin = -Inf, ymax = Inf,
           fill = "orange", alpha = 0.2) +
  annotate("rect",
           xmin = 21, xmax = 50,
           ymin = -Inf, ymax = Inf,
           fill = "red", alpha = 0.2) +
  annotate("rect",
           xmin = 51, xmax = Inf,
           ymin = -Inf, ymax = Inf,
           fill = "purple", alpha = 0.2) +
  facet_wrap(model ~ language, scales = "free") +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  labs(x = "Cyclomatic complexity", y = "Nº") +
  theme_AP()
```

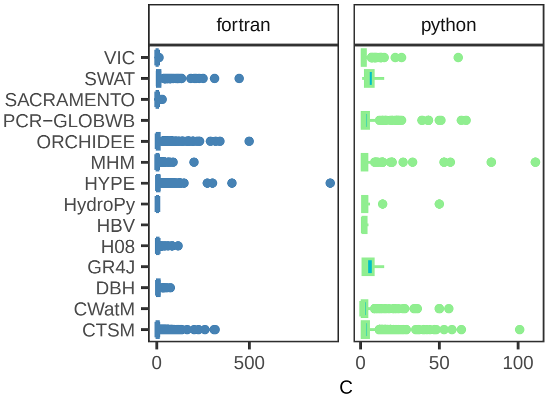
```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



```r
plot_scatterplot <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, `function`, cyclomatic_complexity, loc, bugs, type)]) %>%
  rbindlist() %>%
  .[type %in% vector_cyclomatic] %>%
  ggplot(., aes(cyclomatic_complexity, loc, color = language)) +
  geom_point(alpha = 0.2) +
  scale_color_manual(values = color_languages) +
  coord_flip() +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  facet_wrap(~language) +
  labs(x = "C", y = "Lines of code") +
  theme_AP() +
  theme(legend.position = "none")

plot_scatterplot
```

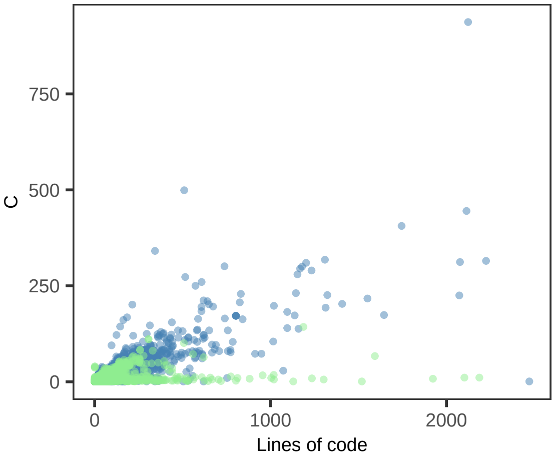


```
plot_c_model <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, `function`, cyclomatic_complexity, loc, bugs, type)]) %>%
  rbindlist() %>%
  .[type %in% vector_cyclomatic] %>%
  ggplot(., aes(model, cyclomatic_complexity, fill = language, color = language)) +
  geom_boxplot(outlier.size = 1) +
  coord_flip() +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 2)) +
  facet_wrap(~language, scales = "free_x") +
  labs(x = "", y = "C") +
  theme_AP() +
  scale_color_manual(values = color_languages) +
  theme(legend.position = "none")

plot_c_model
```

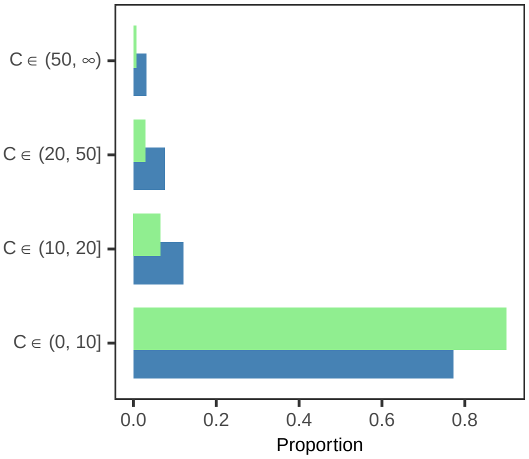


```
# Scatterplot cyclomatic vs lines of code --------------------------------------

plot_c_vs_loc <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(loc, cyclomatic_complexity, language)]) %>%
  rbindlist() %>%
  ggplot(., aes(loc, cyclomatic_complexity, color = language)) +
  geom_point(alpha = 0.5, size = 0.7) +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = "Lines of code", y = "C") +
  scale_color_manual(values = color_languages) +
  theme_AP() +
  theme(legend.position = "none")

plot_c_vs_loc
```

```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



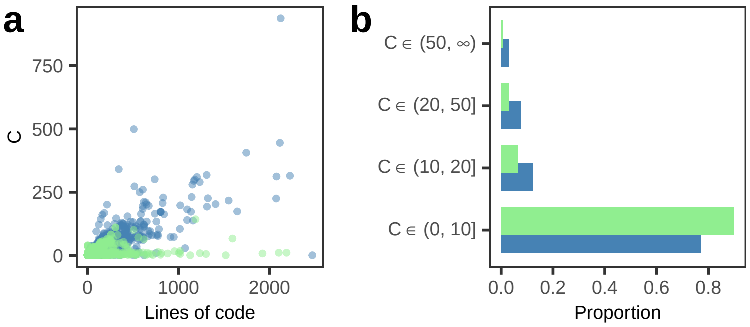
```
# Count & proportion -----------------------------------------------------------

plot_bar_cyclomatic <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(complexity_category, language, type)]) %>%
  rbindlist() %>%
  .[type %in% vector_cyclomatic] %>%
  .[, .N, .(complexity_category, language)] %>%
  .[, proportion := N / sum(N), language] %>%
  ggplot(., aes(complexity_category, proportion, fill = language)) +
  geom_bar(stat = "identity", position = position_dodge(0.6)) +
  scale_fill_manual(values = color_languages) +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 4)) +
  scale_x_discrete(labels = lab_expr) +
  labs(x = "", y = "Proportion") +
  coord_flip() +
  theme_AP() +
  theme(legend.position = "none")

plot_bar_cyclomatic
```



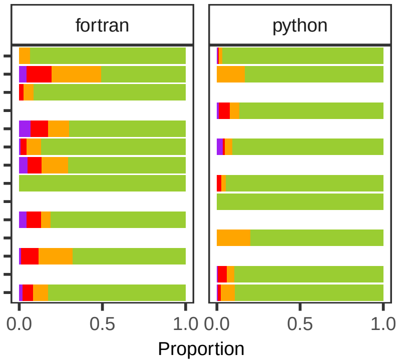
```
# MERGE ########################################################################


 plot_cyclomatic <- plot_grid(plot_c_vs_loc, plot_bar_cyclomatic, ncol = 2, labels = "auto",
          rel_widths = c(0.45, 0.55))
```

```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

```
plot_cyclomatic
```



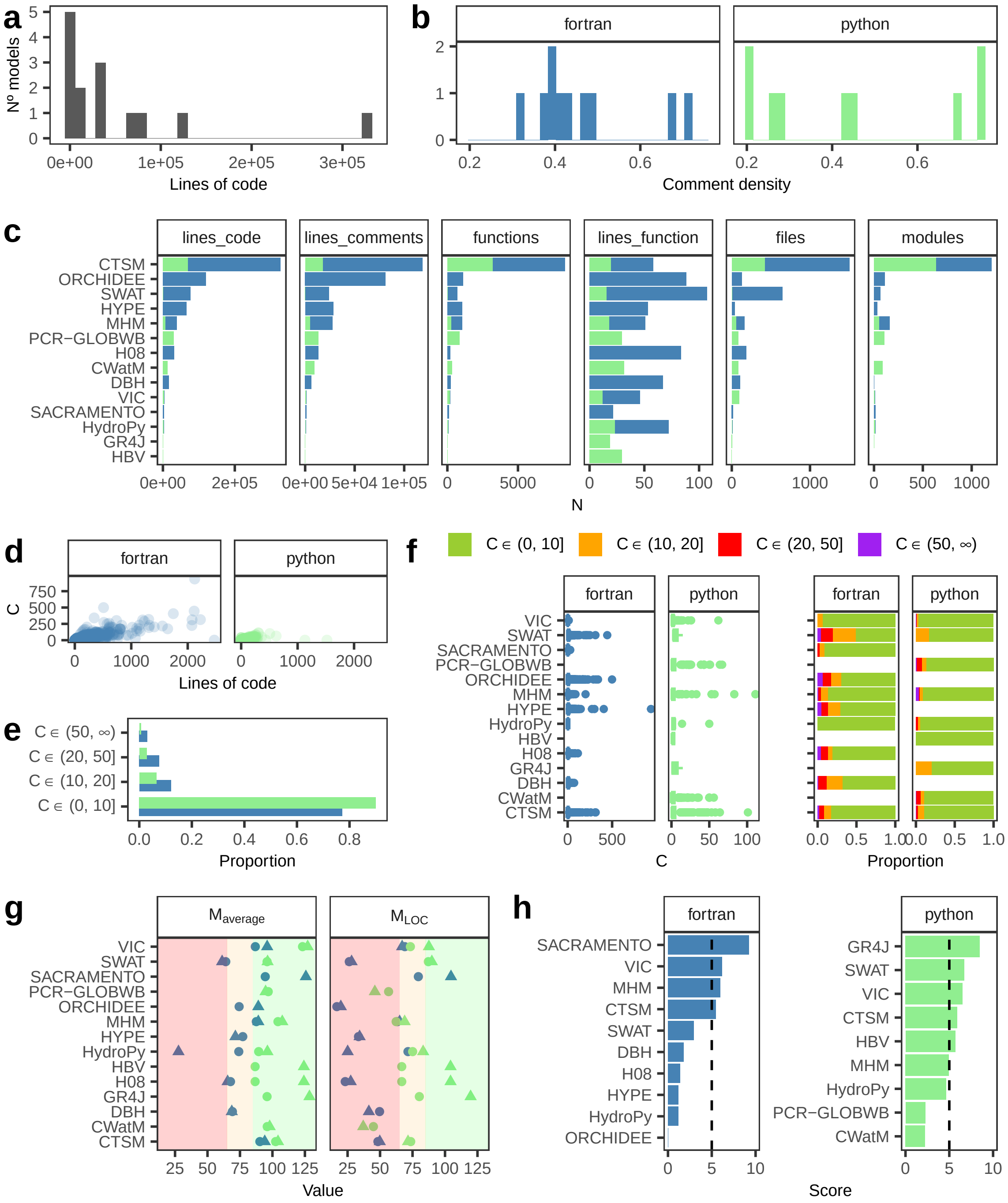
```
plot_bar_category <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, complexity_category, type)]) %>%
  rbindlist() %>%
  .[type %in% vector_cyclomatic] %>%
  .[, .N, .(model, language, complexity_category)] %>%
  .[, proportion := N / sum(N), .(language, model)] %>%
  ggplot(., aes(model, proportion, fill = complexity_category)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("yellowgreen", "orange", "red", "purple"),
                    labels = lab_expr,
                    name = "") +
  facet_wrap(~language) +
  labs(x = "", y = "Proportion") +
  coord_flip() +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 3)) +
  theme_AP() +
  theme(legend.position = "none") +
  theme(axis.text.y = element_blank(),
        legend.text = element_text(size = 7))

plot_bar_category
```



```
di <- plot_grid(plot_scatterplot, plot_bar_cyclomatic, ncol = 1, labels = c("d", "e"))
legend <- get_legend_fun(plot_bar_category + theme(legend.position = "top"))
```

```
## Warning: `is.ggplot()` was deprecated in ggplot2 3.5.2.
## i Please use `is_ggplot()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
dada <- plot_grid(plot_c_model, plot_bar_category, ncol = 2, rel_widths = c(0.61, 0.39))
dada2 <- plot_grid(legend, dada, ncol = 1, rel_heights = c(0.1, 0.9), labels = "f")
dada3 <- plot_grid(di, dada2, ncol = 2, rel_widths = c(0.4, 0.6))
dada4 <- plot_grid(plot_maintainability_index, plot_score, ncol = 2, labels = c("g", "h"))
dada5 <- plot_grid(p1, dada3, ncol = 1, rel_heights = c(0.6, 0.4))
plot_grid(dada5, dada4, rel_heights = c(0.73, 0.27), ncol = 1)
```


a
Nº models
Lines of code
0e+00
1e+05
2e+05
3e+05
b
fortran
python
Comment density
c
lines_code
lines_comments
functions
lines_function
files
modules
CTSM
ORCHIDEE
SWAT
HYPE
MHM
PCR−GLOBWB
H08
CWatM
DBH
VIC
SACRAMENTO
HydroPy
GR4J
HBV
N
d
fortran
python
C
Lines of code
e
C ∈ (50, ∞)
C ∈ (20, 50]
C ∈ (10, 20]
C ∈ (0, 10]
Proportion
f
🟩 C ∈ (0, 10]
🟧 C ∈ (10, 20]
🟥 C ∈ (20, 50]
🟪 C ∈ (50, ∞)
fortran
python
VIC
SWAT
SACRAMENTO
PCR−GLOBWB
ORCHIDEE
MHM
HYPE
HydroPy
HBV
H08
GR4J
DBH
CWatM
CTSM
C
Proportion
g
M_average
M_LOC
Value
h
fortran
SACRAMENTO
VIC
MHM
CTSM
SWAT
DBH
H08
HYPE
HydroPy
ORCHIDEE
python
GR4J
SWAT
VIC
CTSM
HBV
MHM
HydroPy
PCR−GLOBWB
CWatM
Score