User Guide for Updating the GEMMA Toolbox

The **GEMMA toolbox** main script is organized into two key sections: the **User Interface (UI)** and the **Server**. These two sections collaborate to build an interactive application. External scripts should be added in the script sub-folder.

---

1. Overview of the GEMMA Toolbox Structure

UI Function:
The UI function determines the application's layout and appearance. It defines input widgets (e.g., sliders, text boxes, dropdown menus) and output components (e.g., plots, tables, text). Using layout functions such as *fluidPage()*, *sidebarLayout()*, or *navbarPage()*, you can structure the interface effectively.

Server Function:
The server function handles the application's logic and computations. It processes user inputs and dynamically updates outputs using reactive expressions and render functions like *renderPlot()* or *renderTable()*.

Together, the UI and server functions create a seamless interaction loop: the UI gathers inputs from users, while the server computes result and displays them back via the UI.

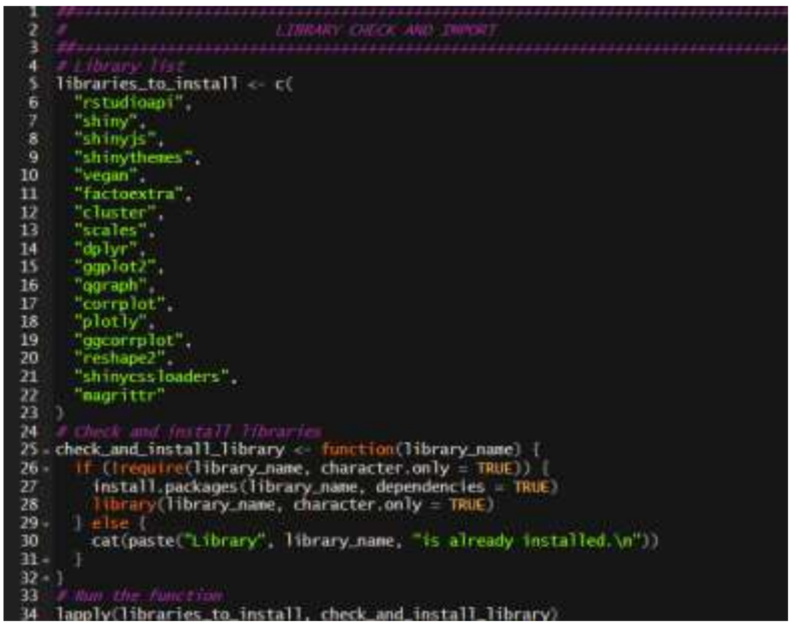
---

2. Adding a New Section to the GEMMA Toolbox

This guide provides instructions for adding a new section to the GEMMA toolbox. While advanced structural implementations are recommended for experienced programmers, minor customizations (e.g., styling, adding statistical tests) can be performed by any R user.

Step 1: Update the Library List

If your new section requires additional R libraries, ensure they are added to the GEMMA toolbox library list. Refer to the figure below for guidance.



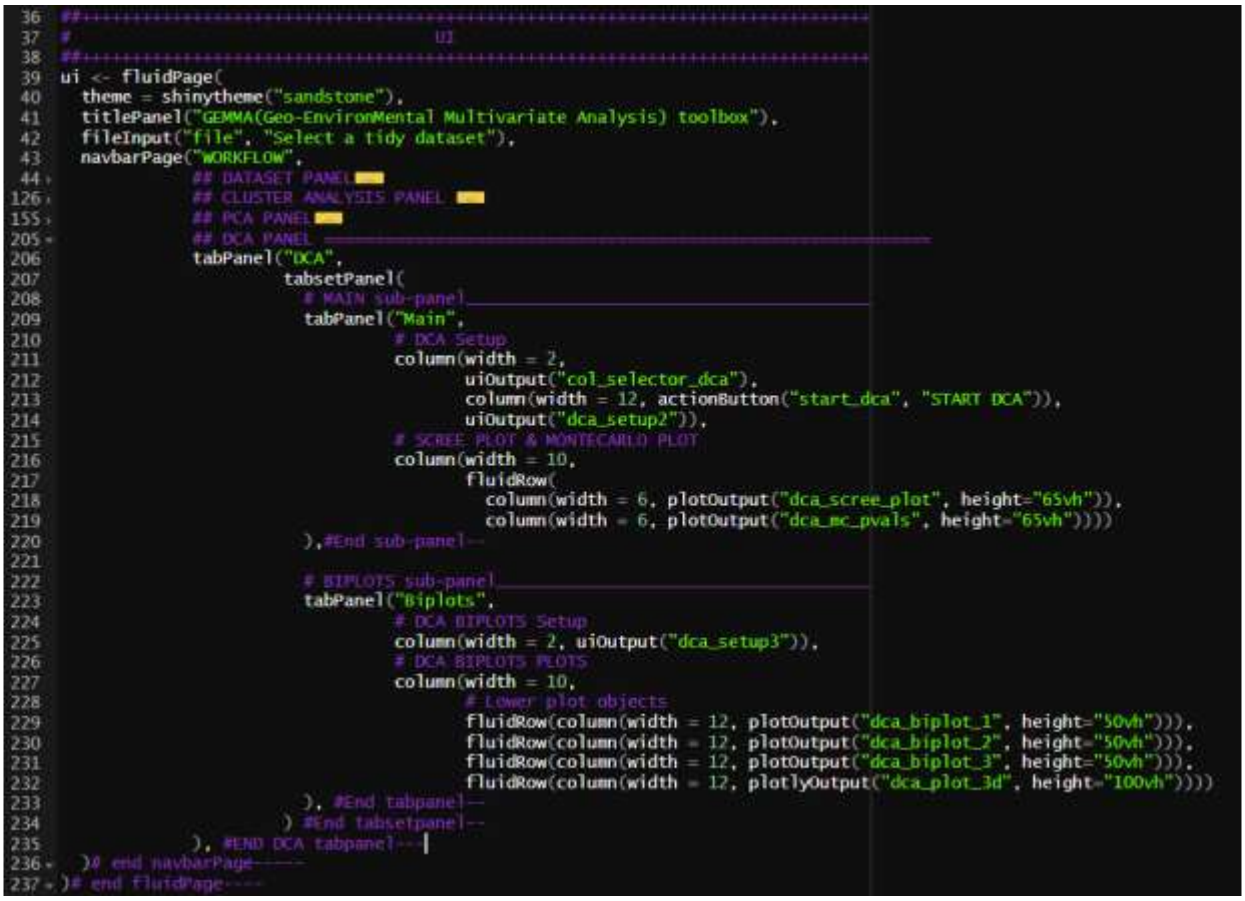
```
##############################################################################
#                         LIBRARY CHECK AND IMPORT
##############################################################################
# Library list
libraries_to_install <- c(
  "rstudioapi",
  "shiny",
  "shinyjs",
  "shinythemes",
  "vegan",
  "factoextra",
  "cluster",
  "scales",
  "dplyr",
  "ggplot2",
  "qgraph",
  "corrplot",
  "plotly",
  "ggcorrplot",
  "reshape2",
  "shinycssloaders",
  "magrittr"
)
# Check and install libraries
check_and_install_library <- function(library_name) {
  if (!require(library_name, character.only = TRUE)) {
    install.packages(library_name, dependencies = TRUE)
    library(library_name, character.only = TRUE)
  } else {
    cat(paste("Library", library_name, "is already installed.\n"))
  }
}
# Run the function
lapply(libraries_to_install, check_and_install_library)
```

Step 2: Update the UI Function

In this example, we'll add a new panel called **"DCA"** with two subpanels: **"MAIN"** and **"BIPLOTS"**.

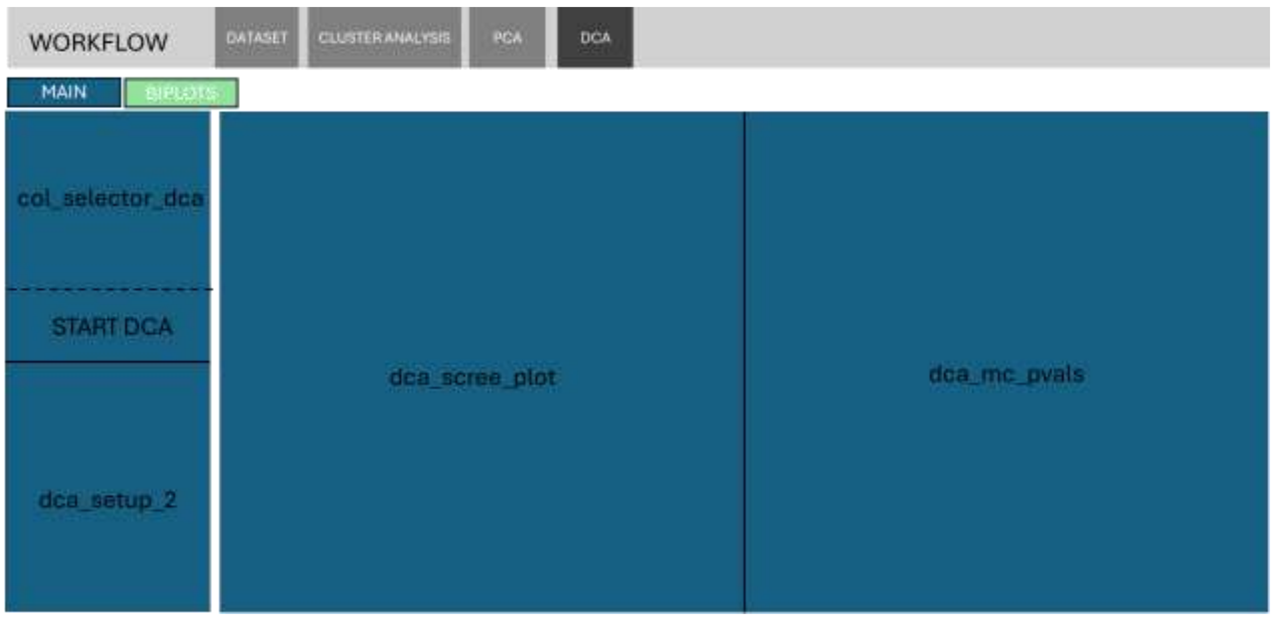
- Panels and subpanels can be created using the *tabPanel()* function.
- Use columns and rows to organize the graphical user interface (GUI).
- Incorporate dynamic elements using the *uiOutput()* function, which allows for GUI updates during the workflow.

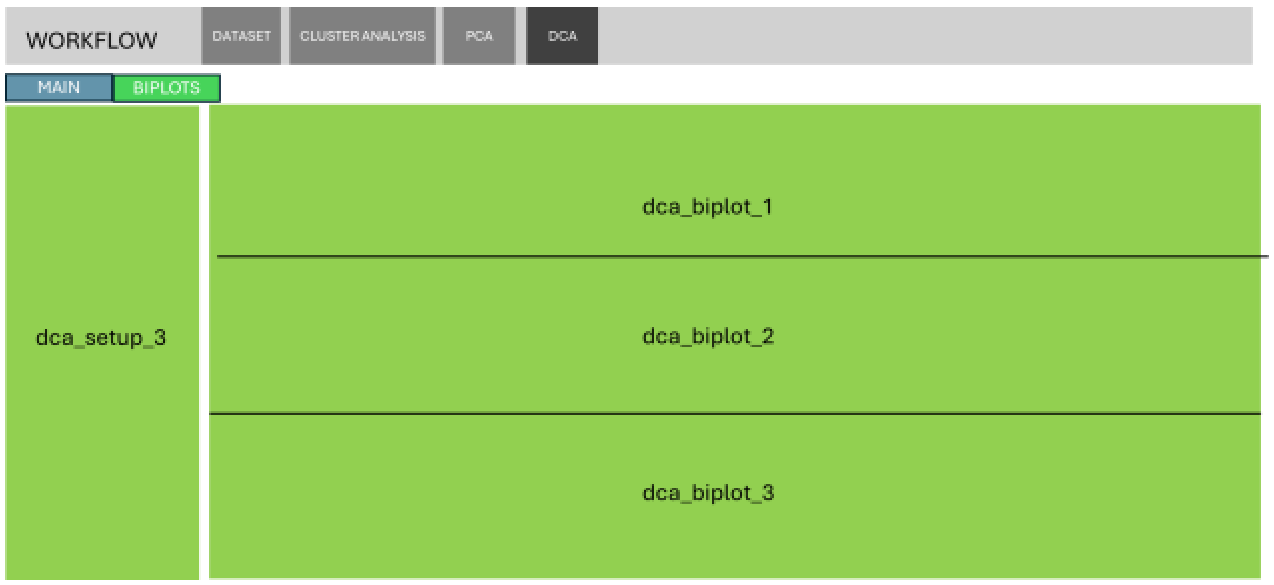
The figures below illustrate the structure of the new subpanels.



```r
##++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#                                    UI
##++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ui <- fluidPage(
  theme = shinytheme("sandstone"),
  titlePanel("GEMMA(Geo-EnvironMental Multivariate Analysis) toolbox"),
  fileInput("file", "Select a tidy dataset"),
  navbarPage("WORKFLOW",
             ## DATASET PANEL
             ## CLUSTER ANALYSIS PANEL
             ## PCA PANEL
             ## DCA PANEL ==========================================================
             tabPanel("DCA",
                      tabsetPanel(
                        # MAIN sub-panel____________________________________________
                        tabPanel("Main",
                                 # DCA Setup
                                 column(width = 2,
                                        uiOutput("col_selector_dca"),
                                        column(width = 12, actionButton("start_dca", "START DCA")),
                                        uiOutput("dca_setup2")),
                                 # SCREE PLOT & MONTECARLO PLOT
                                 column(width = 10,
                                        fluidRow(
                                          column(width = 6, plotOutput("dca_scree_plot", height="65vh")),
                                          column(width = 6, plotOutput("dca_mc_pvals", height="65vh"))))
                        ),#End sub-panel--

                        # BIPLOTS sub-panel_________________________________________
                        tabPanel("Biplots",
                                 # DCA BIPLOTS Setup
                                 column(width = 2, uiOutput("dca_setup3")),
                                 # DCA BIPLOTS PLOTS
                                 column(width = 10,
                                        # Lower plot objects
                                        fluidRow(column(width = 12, plotOutput("dca_biplot_1", height="50vh"))),
                                        fluidRow(column(width = 12, plotOutput("dca_biplot_2", height="50vh"))),
                                        fluidRow(column(width = 12, plotOutput("dca_biplot_3", height="50vh"))),
                                        fluidRow(column(width = 12, plotlyOutput("dca_plot_3d", height="100vh"))))
                        ), #End tabpanel--
                      ) #End tabsetpanel--
             ), #END DCA tabpanel---
  )# end navbarPage-----
)# end fluidPage----
```





Step 3: Modify the Server Function

The server function is the most complex part of the application, handling:

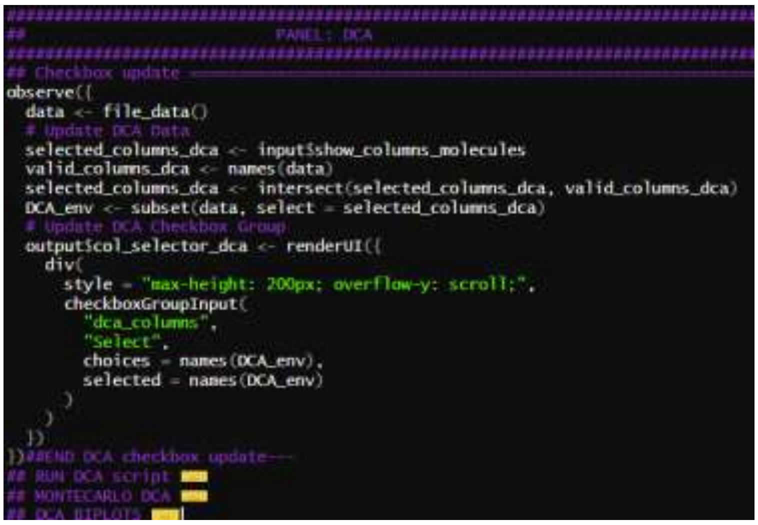
- Iterative processes
- Data analysis
- Script calls
- Plot updates
- Data export

Important: When modifying the server function:

- Avoid conflicts between global and local variables.
- Ensure the software remains functional under iterative, multi-analysis scenarios.

Server Function Features:

1. **Dynamic Variable Updates:**
   Use the *observe()* function to monitor variables in **MATRIX A** and update the checkbox list for DCA analysis using the *renderUI()* function.

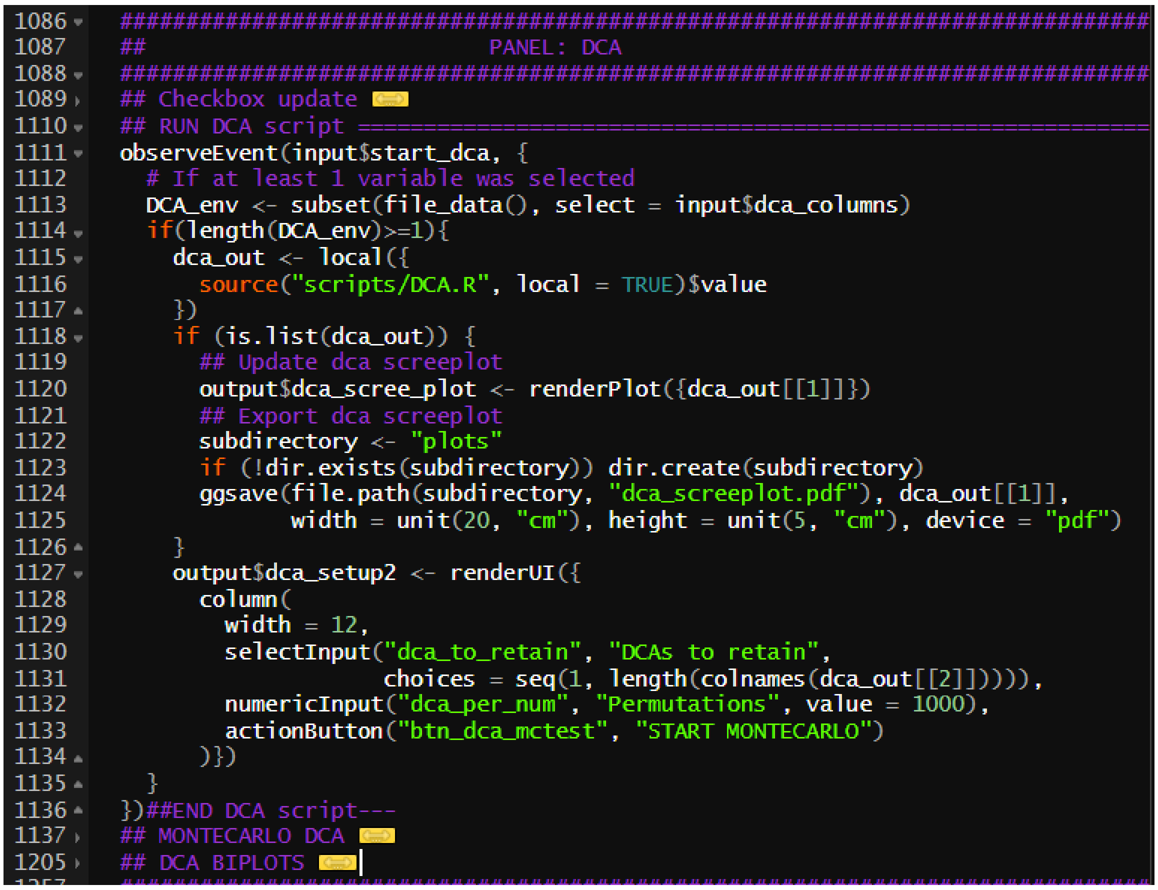


```
##############################################################################
##                           PANEL: DCA
##############################################################################
## Checkbox update ===========================================================
observe({
  data <- file_data()
  # Update DCA Data
  selected_columns_dca <- input$show_columns_molecules
  valid_columns_dca <- names(data)
  selected_columns_dca <- intersect(selected_columns_dca, valid_columns_dca)
  DCA_env <- subset(data, select = selected_columns_dca)
  # Update DCA Checkbox Group
  output$col_selector_dca <- renderUI({
    div(
      style = "max-height: 200px; overflow-y: scroll;",
      checkboxGroupInput(
        "dca_columns",
        "Select",
        choices = names(DCA_env),
        selected = names(DCA_env)
      )
    )
  })
})##END DCA checkbox update---
## RUN DCA script 
## MONTECARLO DCA 
## DCA BIPLOTS 
```

2. **Triggering Events:**
   The observeEvent() function detects when the **"start_dca"** button is clicked. Once triggered, the DCA process begins.

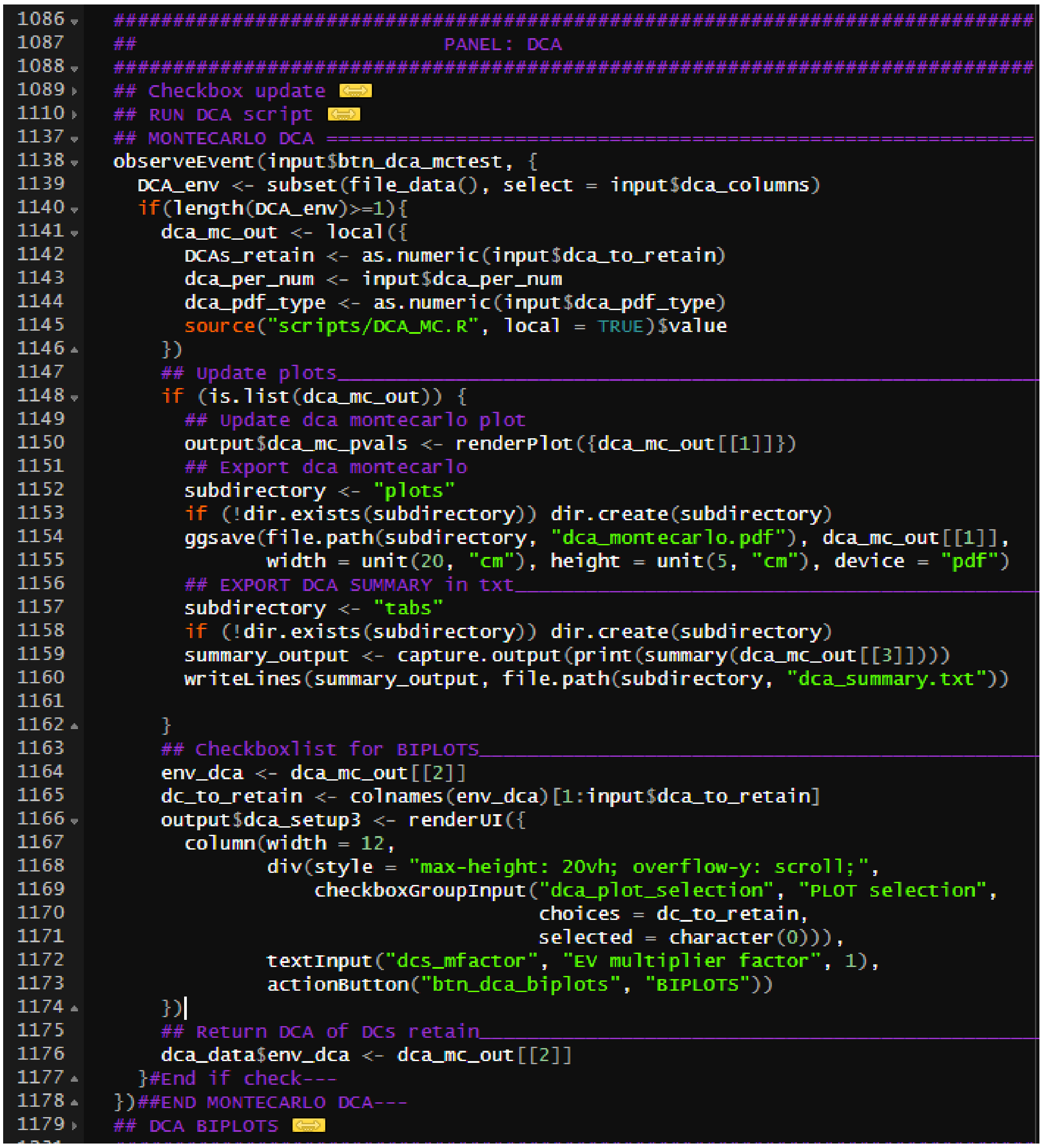
   ○ Scripts (e.g., RDA.R) can be executed using the local() function for a modular design.

   ○ Outputs, such as scree plots, can be saved using the ggsave() function.



```
##############################################################################
##                           PANEL: DCA
##############################################################################
## Checkbox update 
## RUN DCA script ============================================================
observeEvent(input$start_dca, {
  # If at least 1 variable was selected
  DCA_env <- subset(file_data(), select = input$dca_columns)
  if(length(DCA_env)>=1){
    dca_out <- local({
      source("scripts/DCA.R", local = TRUE)$value
    })
    if (is.list(dca_out)) {
      ## Update dca screeplot
      output$dca_scree_plot <- renderPlot({dca_out[[1]]})
      ## Export dca screeplot
      subdirectory <- "plots"
      if (!dir.exists(subdirectory)) dir.create(subdirectory)
      ggsave(file.path(subdirectory, "dca_screeplot.pdf"), dca_out[[1]],
             width = unit(20, "cm"), height = unit(5, "cm"), device = "pdf")
    }
    output$dca_setup2 <- renderUI({
      column(
        width = 12,
        selectInput("dca_to_retain", "DCAs to retain",
                    choices = seq(1, length(colnames(dca_out[[2]])))),
        numericInput("dca_per_num", "Permutations", value = 1000),
        actionButton("btn_dca_mctest", "START MONTECARLO")
      )})
  }
})##END DCA script---
## MONTECARLO DCA 
## DCA BIPLOTS 
```

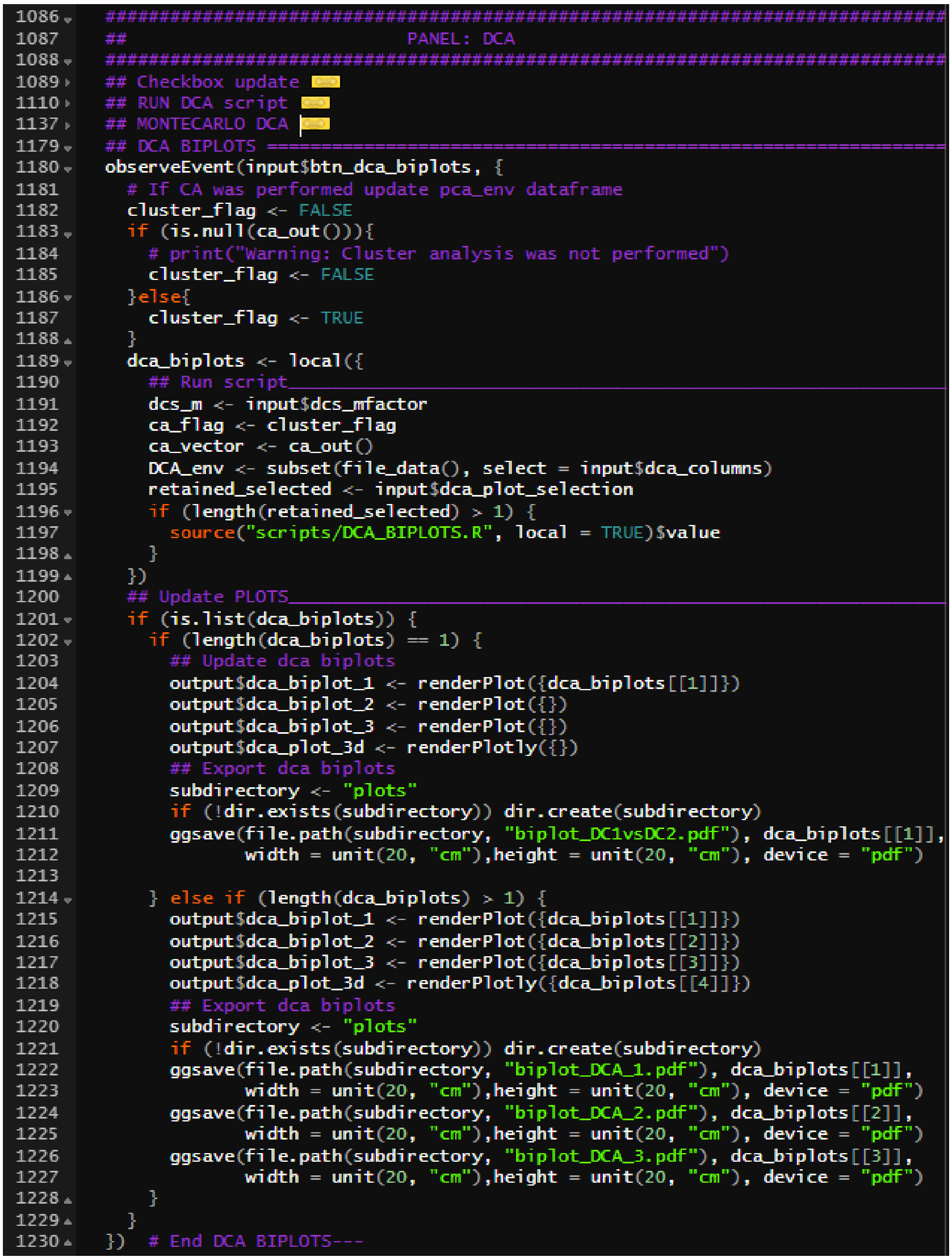
3. **Monte Carlo Test Integration:**
   - A dedicated section runs the **DCA_MC** script, updates the dca_mc_pvals plot area, and allows PDF export.
   - The **"BIPLOTS"** subpanel dynamically updates with retained DCA axes and provides a button for running the biplot script.



```
################################################################################
##                          PANEL: DCA
################################################################################
## Checkbox update
## RUN DCA script
## MONTECARLO DCA =============================================================
observeEvent(input$btn_dca_mctest, {
  DCA_env <- subset(file_data(), select = input$dca_columns)
  if(length(DCA_env)>=1){
    dca_mc_out <- local({
      DCAs_retain <- as.numeric(input$dca_to_retain)
      dca_per_num <- input$dca_per_num
      dca_pdf_type <- as.numeric(input$dca_pdf_type)
      source("scripts/DCA_MC.R", local = TRUE)$value
    })
    ## Update plots_______________________________________________________________
    if (is.list(dca_mc_out)) {
      ## Update dca montecarlo plot
      output$dca_mc_pvals <- renderPlot({dca_mc_out[[1]]})
      ## Export dca montecarlo
      subdirectory <- "plots"
      if (!dir.exists(subdirectory)) dir.create(subdirectory)
      ggsave(file.path(subdirectory, "dca_montecarlo.pdf"), dca_mc_out[[1]],
             width = unit(20, "cm"), height = unit(5, "cm"), device = "pdf")
      ## EXPORT DCA SUMMARY in txt_____________________________________________
      subdirectory <- "tabs"
      if (!dir.exists(subdirectory)) dir.create(subdirectory)
      summary_output <- capture.output(print(summary(dca_mc_out[[3]])))
      writeLines(summary_output, file.path(subdirectory, "dca_summary.txt"))

    }
    ## Checkboxlist for BIPLOTS___________________________________________________
    env_dca <- dca_mc_out[[2]]
    dc_to_retain <- colnames(env_dca)[1:input$dca_to_retain]
    output$dca_setup3 <- renderUI({
      column(width = 12,
             div(style = "max-height: 20vh; overflow-y: scroll;",
                 checkboxGroupInput("dca_plot_selection", "PLOT selection",
                                    choices = dc_to_retain,
                                    selected = character(0))),
             textInput("dcs_mfactor", "EV multiplier factor", 1),
             actionButton("btn_dca_biplots", "BIPLOTS"))
    })
    ## Return DCA of DCs retain___________________________________________________
    dca_data$env_dca <- dca_mc_out[[2]]
  }#End if check---
})##END MONTECARLO DCA---
## DCA BIPLOTS
```

4. **Biplot Creation:**
   - After completing the Monte Carlo test, users can access the biplot subsection.
   - DCA axes can be selected via a checkbox list.
   - The DCA_BIPLOT.R script generates biplots, and areas like dca_biplot_1, dca_biplot_2, and dca_biplot_3 are updated accordingly.

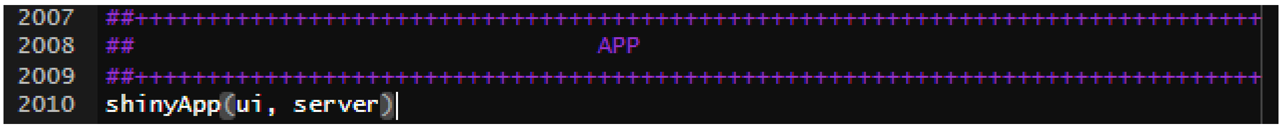


```
##############################################################################
##                          PANEL: DCA
##############################################################################
## Checkbox update 
## RUN DCA script 
## MONTECARLO DCA 
## DCA BIPLOTS ===============================================================
observeEvent(input$btn_dca_biplots, {
  # If CA was performed update pca_env dataframe
  cluster_flag <- FALSE
  if (is.null(ca_out())){
    # print("Warning: Cluster analysis was not performed")
    cluster_flag <- FALSE
  }else{
    cluster_flag <- TRUE
  }
  dca_biplots <- local({
    ## Run script
    dcs_m <- input$dcs_mfactor
    ca_flag <- cluster_flag
    ca_vector <- ca_out()
    DCA_env <- subset(file_data(), select = input$dca_columns)
    retained_selected <- input$dca_plot_selection
    if (length(retained_selected) > 1) {
      source("scripts/DCA_BIPLOTS.R", local = TRUE)$value
    }
  })
  ## Update PLOTS
  if (is.list(dca_biplots)) {
    if (length(dca_biplots) == 1) {
      ## Update dca biplots
      output$dca_biplot_1 <- renderPlot({dca_biplots[[1]]})
      output$dca_biplot_2 <- renderPlot({})
      output$dca_biplot_3 <- renderPlot({})
      output$dca_plot_3d <- renderPlotly({})
      ## Export dca biplots
      subdirectory <- "plots"
      if (!dir.exists(subdirectory)) dir.create(subdirectory)
      ggsave(file.path(subdirectory, "biplot_DC1vsDC2.pdf"), dca_biplots[[1]],
             width = unit(20, "cm"),height = unit(20, "cm"), device = "pdf")

    } else if (length(dca_biplots) > 1) {
      output$dca_biplot_1 <- renderPlot({dca_biplots[[1]]})
      output$dca_biplot_2 <- renderPlot({dca_biplots[[2]]})
      output$dca_biplot_3 <- renderPlot({dca_biplots[[3]]})
      output$dca_plot_3d <- renderPlotly({dca_biplots[[4]]})
      ## Export dca biplots
      subdirectory <- "plots"
      if (!dir.exists(subdirectory)) dir.create(subdirectory)
      ggsave(file.path(subdirectory, "biplot_DCA_1.pdf"), dca_biplots[[1]],
             width = unit(20, "cm"),height = unit(20, "cm"), device = "pdf")
      ggsave(file.path(subdirectory, "biplot_DCA_2.pdf"), dca_biplots[[2]],
             width = unit(20, "cm"),height = unit(20, "cm"), device = "pdf")
      ggsave(file.path(subdirectory, "biplot_DCA_3.pdf"), dca_biplots[[3]],
             width = unit(20, "cm"),height = unit(20, "cm"), device = "pdf")
    }
  }
})  # End DCA BIPLOTS---
```

Step 4: Create the App

The final step is to build the app using the shinyApp() function. See the figure below for an example.



```
##++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
##                                    APP
##++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
shinyApp(ui, server)
```

Additional Notes

- For advanced customizations, maintain code modularity and perform extensive testing to ensure stability.
- Minor updates (e.g., styling or additional statistical features) can be made without altering the core structure.

This guide should help you extend the GEMMA toolbox while preserving its functionality and integrity.