

Universal ggThemeAssist: An Interactive Shiny App for Custom ggplot2 Theming

Group 16

June 5, 2025

Abstract

This report describes the design, implementation, and evaluation of *Universal ggThemeAssist*, a Shiny application that allows users to upload a CSV dataset, choose plot geometries, customize ggplot2 themes interactively, and save/load theme configurations to disk. By combining Shiny’s reactivity with ggplot2’s theming system and JSON-based persistence, the app enables rapid iteration of plot aesthetics and fosters reproducible, user-defined styling. We discuss our approach, document the code structure, demonstrate the final product’s features, and outline limitations and future enhancements.

1 Introduction

The process of crafting publication-quality plots in R often entails adjusting theme parameters—base font sizes, grid visibility, panel backgrounds, color palettes, and coordinate transformations—using low-level `ggplot2::theme()` calls. For many users, especially those not deeply familiar with ggplot2’s theming API, this becomes tedious: one must modify code, re-run plots, and remember the exact combination of theme settings. Existing solutions (such as prebuilt ggplot2 themes or GUI packages limited to specific presets) do not offer on-the-fly saving and reloading of user-defined theme configurations.

Universal ggThemeAssist addresses this gap by providing:

- An interactive Shiny interface for uploading any CSV file, selecting variables, and choosing a plot geometry (scatter, line, bar, density, histogram).
- Controls for specifying colors (manual or via a color picker), transparency, point sizes, and axis scales (log transforms).

- A theming panel where users select from base themes (`theme_minimal`, `theme_grey`, `theme_dark`, etc.), adjust base font size, toggle major grid lines, flip coordinates, and choose background colors appropriate to the base theme.
- Functionality to save the current theme configuration under a user-defined name and persist it as JSON, as well as load or delete saved themes.
- Generation of reproducible `ggplot2` code reflecting the user’s choices, which can be copied for further customization outside the app.

The novelty of our solution lies in combining Shiny’s reactivity, `ggplot2`’s theme layering, and JSON-based storage to create a reusable library of themes. Users gain the freedom to iterate rapidly on aesthetic settings, share theme files, and reproduce plots consistently across projects. The remainder of this report explains our design decisions, documents the code structure, highlights the app’s components and usage, and offers a critical discussion of results, limitations, and future directions.

2 Justification of Approach

2.1 Why Shiny?

We selected **Shiny** as the application framework to deliver a fully interactive user experience. Shiny’s reactive programming model ensures that any change in input controls (e.g., selecting a new theme or toggling grid visibility) automatically triggers re-rendering of the plot without requiring the user to manually refresh. In contrast, static RMarkdown or script-based approaches force users to re-run entire code blocks whenever they tweak a single parameter—a less fluid workflow. Furthermore, Shiny’s `selectInput`, `sliderInput`, `checkboxInput`, and `fileInput` widgets make constructing a dynamic UI straightforward.

2.2 Why ggplot2?

ggplot2 is the de facto standard for plotting in R. Its grammar-of-graphics paradigm supports layering of geometries and themes. By building on `ggplot2`:

- We leverage a familiar syntax: `ggplot(data, aes(...)) + geom_point() + theme_minimal()`.
- Users can immediately understand generated code, copy it into scripts, and extend it as needed.

- ggplot2's existing theme functions (`theme_minimal`, `theme_dark`, `theme_grey`, etc.) provide robust starting points.
- We can easily apply additional theme elements like `theme(panel.background = element_rect(...))` or `coord_flip()` for coordinate flipping.

Since our goal is to provide a GUI layer on top of ggplot2's theming capabilities, ggplot2 was the natural choice for the plotting engine.

2.3 Persistence via JSON

To allow users to save and reload custom theme configurations, we chose **JSON** as the storage format:

- JSON is human-readable and structured, making it easy to inspect or edit theme files outside of R.
- The `jsonlite` package provides robust conversion between R lists and JSON files.
- Persisting configurations as a named list keyed by theme name allows the user to maintain multiple custom themes in a single file (`saved_themes.json`).

This contrasts with other approaches (e.g., writing an RDS file) by offering cross-language portability: theoretically, a Python or JavaScript tool could parse and apply the same theme settings.

3 Code Overview

Below we break down the key components of the R script. Each function is documented to ensure reproducibility.

3.1 Helper Functions

```
# File: server.R (or app.R)
library(shiny)
library(ggplot2)
library(jsonlite)
library(colourpicker)

theme_file <- "saved_themes.json"
```

1. `save_theme_to_file(name, config)`

- **Purpose:** Append or overwrite a named theme configuration in `saved_themes.json`.
- **Inputs:**
 - `name` (string): User-provided theme name (e.g., “MyDarkStyle”).
 - `config` (list): A list containing:
 - * `base_theme`: One of `{\gggplot2", \plotly", \plotly_white", \plotly_dark", \seaborn"}`.
 - * `base_size`: Numeric base font size (8–24).
 - * `show_grid`: Logical—TRUE if major grid lines should be drawn.
 - * `flip_coords`: Logical—TRUE if coordinates should be flipped.
- **Implementation Highlights:**
 - Reads existing JSON with `fromJSON(theme_file)` if the file exists; otherwise, initializes an empty list.
 - Assigns `themes[[name]] <- config`.
 - Writes back to disk using `write(toJSON(themes, pretty=TRUE), theme_file)`.
 - Returns TRUE on success, FALSE on any error (using `tryCatch`).

2. `load_themes_from_file()`

- **Purpose:** Load all saved theme configurations as an R list.
- **Returns:** A named list (possibly empty) where each element is a theme configuration list.
- **Implementation Highlights:**
 - If `theme_file` exists, calls `fromJSON` to parse it, returning a list.
 - Otherwise returns an empty list.
 - Protected by `tryCatch`—on error, returns an empty list.

3. `delete_theme_from_file(name)`

- **Purpose:** Remove a given theme by key from the JSON file.
- **Inputs:** `name` (string): Name of the theme to delete.
- **Returns:** TRUE if deletion succeeded, FALSE otherwise.
- **Implementation Highlights:**

- Reads existing JSON into `themes`.
- Sets `themes[[name]] <- NULL` to remove the element.
- Writes back to disk with `write(toJSON(...), theme_file)`.

4. `apply_theme_config(conf, session)`

- **Purpose:** Update Shiny input widgets to reflect a loaded theme configuration.
- **Inputs:**
 - `conf` (list): Theme configuration as saved (see above).
 - `session` (Shiny session): Allows calling `updateSelectInput`, `updateSliderInput`, `updateCheckboxInput`.
- **Behavior:**
 - (a) Sets `\theme` selector to `conf$base_theme`.
 - (b) Sets `\base_size` slider to `conf$base_size`.
 - (c) Sets `\show_grid` checkbox to `conf$show_grid`.
 - (d) Sets `\flip` checkbox to `conf$flip_coords`.

5. `generate_geom(input, colorvar, fillcol)`

- **Purpose:** Return the appropriate `geom_*()` layer based on the user's chosen geometry and color settings.
- **Inputs:**
 - `input$geom`: One of `{\point, \line, \col, \density, \histogram}`.
 - `colorvar` (string): "none" or a column name for coloring.
 - `fillcol` (hex code): Color to use if `colorvar == \none`.
- **Returns:** A ggplot2 `geom_*()` layer.
- **Examples:**

```
# If user selects \scatter" and no color variable:
generate_geom(input, "none", "#2C3E50")
# returns: geom_point(size = input$point_size, alpha = input$alpha,
#           color = "#2C3E50")
```

3.2 UI Definition (ui block)

The UI is contained entirely within `fluidPage()`. Key elements include:

- `titlePanel("Universal ggThemeAssist")`
- `sidebarLayout()` with:
 - `sidebarPanel()`
 - * `fileInput("file", "Upload CSV", accept = ".csv")`
 - * A `conditionalPanel` keyed on `output.fileUploaded` (reactive), displaying variable selectors and plot options only after a file is uploaded.
 - * **Variable Selection:** `uiOutput("var_selectors")` generates:
 - `selectInput("xvar", "X Variable", choices = vars)`
 - For non-density/histogram geoms, `selectInput("yvar", "Y Variable", choices = vars)`
 - `selectInput("colorvar", "Color Variable (Optional)", choices = c("None"="none", vars))`
 - * **Plot Options:** `selectInput("geom", "Geometry", choices = c(...)).`
If `input.colorvar == "none"`:
 - `colourInput("manual_color", "Choose Plot Color", value = "#2C3E50")`
(from `colourpicker` package).
 - `sliderInput("point_size", "Point Size", 0.5, 10, 2).`
 - `sliderInput("alpha", "Transparency (alpha)", 0.1, 1, 1).``checkboxInput("log_x", "Log Scale X-Axis", FALSE).checkboxInput("log_y", "Log Scale Y-Axis", FALSE).textInput("plot_title", "Plot Title", "").textInput("x_label", "X Axis Label", "").textInput("y_label", "Y Axis Label", "").`
 - * **Themes:** `selectInput("theme", "Base Theme", choices = c("plotly", "plotly_white", "plotly_dark", "ggplot2", "seaborn")).sliderInput("base_size", "Base Text Size", 8, 24, 12).checkboxInput("show_grid", "Show Major Grid", TRUE).checkboxInput("flip", "Flip Coordinates", FALSE).`
 - * **Manage Themes:** `textInput("theme_name", "Theme Name", placeholder="e.g., MyDarkStyle").actionButton("save_theme", "Save Current Theme").uiOutput("load_theme_ui")` renders, if saved themes exist:
 - `selectInput("saved_themes", "Load Saved Theme", choices=c("", names(saved_themes)))`.

```

      · actionButton("load_theme", "Load Theme").
      · If a theme is selected, actionButton("delete_theme", "Delete Theme").
* checkboxInput("show_code", "Show ggplot2 Code", FALSE).
* downloadButton("download_plot", "Download Plot").
- mainPanel()
  * If output.fileUploaded is TRUE:
    · plotOutput("themedPlot", height = "500px").
    · tableOutput("data_preview") (shows head of data).
    · verbatimTextOutput("data_info") (displays row/column counts).
    · If input.show_code == TRUE, verbatimTextOutput("themeCode").
  * If no file is uploaded: A placeholder h3("Please upload a CSV file to
    get started") message.

```

3.3 Reactive Server Logic

1. Detecting File Upload:

```

output$fileUploaded <- reactive({ !is.null(input$file) })
outputOptions(output, "fileUploaded", suspendWhenHidden = FALSE)

```

By setting `outputOptions(..., suspendWhenHidden = FALSE)`, Shiny ensures `fileUploaded` remains reactive even when hidden.

2. Reading User Data:

```

user_data <- reactive({
  req(input$file)
  tryCatch({
    df <- read.csv(input$file$datapath, stringsAsFactors = FALSE)
    # Remove rows that are all NA
    df <- df[rowSums(is.na(df)) != ncol(df), ]
    names(df) <- trimws(names(df))
    df
  }, error = function(e) {
    showNotification("Error loading file", type = "error")
    NULL
  })
}

```

```
  })
})
```

- `req(input$file)` halts processing until a file is uploaded.
- Removes rows consisting entirely of NA via `rowSums(is.na(df)) != ncol(df)`.
- Trims whitespace from column names for consistency.

3. Variable Selector UI:

```
output$var_selectors <- renderUI({
  req(user_data())
  vars <- names(user_data())
  tagList(
    selectInput("xvar", "X Variable", choices = vars),
    conditionalPanel("!(input.geom === 'density' || input.geom === 'histogram')",
      selectInput("yvar", "Y Variable", choices = vars)
    ),
    selectInput("colorvar", "Color Variable (Optional)",
      choices = c("None" = "none", setNames(vars, vars)))
  )
})
```

- Dynamically populates the X/Y variable dropdowns based on uploaded data.
- Disables Y variable selection when user picks a density or histogram geometry (which only requires one axis).
- Provides a “none” choice for color variable if the user prefers to specify a manual color.

4. UI for Load/Delete Theme Buttons:

```
output$load_theme_ui <- renderUI({
  saved_themes <- load_themes_from_file()
  if (length(saved_themes) > 0) {
    tagList(
      selectInput("saved_themes", "Load Saved Theme",
        choices = c("Select theme..." = "", names(saved_themes))),
      actionButton("load_theme", "Load Theme"),
      br(), br(),
      conditionalPanel(condition = "input.saved_themes != ''",
```



```

        actionButton("delete_theme", "Delete Theme"))
    )
  }
})

```

- If `saved_themes.json` contains one or more keys, renders a dropdown of theme names.
- Shows “Load Theme” and (conditional) “Delete Theme” buttons.

5. Building the Base Plot:

```

build_plot <- reactive({
  req(input$xvar, user_data())
  df <- user_data()

  aes_mapping <- if (input$geom %in% c("density", "histogram")) {
    aes_string(x = input$xvar)
  } else if (input$colorvar == "none") {
    aes_string(x = input$xvar, y = input$yvar)
  } else {
    aes_string(x = input$xvar, y = input$yvar, color = input$colorvar)
  }

  p <- ggplot(df, aes_mapping)
  p <- p + generate_geom(input, input$colorvar, input$manual_color)
  return(p)
})

```

- Defines the aesthetic mapping based on geometry and color selection.
- Invokes `generate_geom()` to attach the appropriate `geom_*`.
- Returns a `ggplot` object without themes applied.

6. Applying Themes and Finalizing Plot:

```

themed_plot <- reactive({
  req(build_plot())
  p <- build_plot()

  theme_func <- switch(input$theme,

```

```

        "plotly" = theme_minimal,
        "plotly_white" = theme_minimal,
        "plotly_dark" = theme_dark,
        "ggplot2" = theme_grey,
        "seaborn" = theme_minimal,
        theme_minimal)

p <- p + theme_func(base_size = input$base_size)

# Toggle grid lines
if (!input$show_grid) {
  p <- p + theme(panel.grid.major = element_blank(),
                 panel.grid.minor = element_blank())
}

# Base-theme{specific background adjustments
if (input$theme == "ggplot2") {
  p <- p + theme(panel.background = element_rect(fill = "#F5F5F5"),
                 plot.background = element_rect(fill = "white"))
} else if (input$theme == "seaborn") {
  p <- p + theme(panel.background = element_rect(fill = "#EAEAF2"),
                 plot.background = element_rect(fill = "white"))
} else if (input$theme == "plotly_dark") {
  p <- p + theme(panel.background = element_rect(fill = "#2F2F2F"),
                 plot.background = element_rect(fill = "#1E1E1E"),
                 text = element_text(color = "white"),
                 axis.text = element_text(color = "white"))
}

# Additional transformations
if (input$flip) p <- p + coord_flip()
if (input$log_x) p <- p + scale_x_log10()
if (input$log_y) p <- p + scale_y_log10()

# Labels
p <- p + labs(title = input$plot_title,
              x = input$x_label,
              y = input$y_label)

```

```

    return(p)
  })

```

- Chooses a base theme function via `switch(...)`, mapping user choices (“plotly,” “ggplot2,” “seaborn”) to corresponding `theme_minimal`, `theme_grey`, etc.
- Adjusts panel and plot backgrounds based on selected theme, ensuring contrast in “dark” modes.
- Toggles major and minor grid lines off if `input$show_grid == FALSE`.
- Applies coordinate flip (`coord_flip()`) or log scales (`scale_x_log10()`, `scale_y_log10()`) when requested.
- Adds axis labels and plot title via `labs()`.

7. Rendering Outputs:

```

output$themedPlot      <- renderPlot({ themed_plot() }, res = 96)
output$data_preview    <- renderTable({ head(user_data(), 5) }, striped = TRUE)
output$data_info       <- renderText({
  df <- user_data()
  paste("Rows:", nrow(df),
        "Columns:", ncol(df),
        "Numeric columns:", sum(sapply(df, is.numeric)),
        sep = "\n")
})

output$themeCode <- renderText({
  req(input$xvar, input$yvar, input$geom)

  # Build aesthetic string
  if (input$colorvar == "none") {
    aes_str <- paste0("aes(x = ", input$xvar, ", y = ", input$yvar, ")")
  } else {
    aes_str <- paste0("aes(x = ", input$xvar, ", y = ", input$yvar,
                      ", color = ", input$colorvar, ")")
  }

  output$download_plot <- downloadHandler(
    filename = function() "custom_plot.png",
    content  = function(file) ggsave(file, plot = themed_plot(), width = 8, height = 6)
  )

```

- `renderPlot` displays the final plot at 96 dpi.
- `renderTable` shows first five rows of the uploaded data in a striped format.
- `renderText` for `data_info` outputs row/column counts and number of numeric columns.
- `renderText` for `themeCode` builds a text string of R code that reproduces the plot (examples shown in the “Code” section).
- `downloadHandler` invokes `ggsave()` to save the current plot as a PNG.

8. Saving, Loading, and Deleting Themes:

```
observeEvent(input$save_theme, {
  if (nchar(input$theme_name) > 0) {
    conf <- list(
      base_theme = input$theme,
      base_size  = input$base_size,
      show_grid  = input$show_grid,
      flip_coords = input$flip
    )
    if (save_theme_to_file(input$theme_name, conf)) {
      showNotification("Theme saved!", type = "message")
      updateTextInput(session, "theme_name", value = "")
    } else {
      showNotification("Error saving theme", type = "error")
    }
  } else {
    showNotification("Please enter a theme name", type = "warning")
  }
})

observeEvent(input$load_theme, {
  req(input$saved_themes)
  conf <- load_themes_from_file()[[input$saved_themes]]
  if (!is.null(conf)) {
    isolate(apply_theme_config(conf, session))
    showNotification("Theme loaded!", type = "message")
  }
})
```

```
observeEvent(input$delete_theme, {
  if (delete_theme_from_file(input$saved_themes)) {
    showNotification("Theme deleted", type = "message")
    isolate({ updateSelectInput(session, "saved_themes", selected = "") })
  } else {
    showNotification("Failed to delete theme", type = "error")
  }
})
```

- `observeEvent(input$save_theme)` validates that `theme_name` is nonempty, constructs a `conf` list, and calls `save_theme_to_file`.
- On success, shows a “Theme saved!” notification and clears the text box; on failure, displays an error notification.
- `observeEvent(input$load_theme)` retrieves the chosen theme’s configuration and calls `apply_theme_config` to update inputs.
- `observeEvent(input$delete_theme)` calls `delete_theme_from_file` and displays appropriate notifications.

4 Final Product: Features and User Manual

4.1 Components and Libraries

- **Shiny** (v1.7+): Provides the web app infrastructure and reactive programming model.
- **ggplot2** (v3.3+): Used for building and theming plots.
- **jsonlite** (v1.7+): Converts R lists to JSON and vice versa for theme persistence.
- **colourpicker** (v1.2+): Adds a color picker widget (`colourInput`) for manual color selection.
- **DT** (Optional): Could be used for data preview; in our implementation, we used `base renderTable`.

4.2 Feature List

1. Data Upload and Preview:

- Upload any properly formatted CSV file via the `fileInput`.

- The app automatically reads the data into a reactive `data.frame`, dropping rows that are entirely NA.
- The first five rows display in `data_preview`, and basic data information (number of rows, columns, numeric columns) appears in `data_info`.

2. Variable Selection:

- Once a file is uploaded, the `xvar` dropdown lists all column names.
- For geoms requiring both X and Y (scatter, line, bar), the `yvar` dropdown appears.
- Optionally, select a `colorvar` from the same list. If “None” is chosen, a manual color picker appears.

3. Plot Geometry Controls:

- `selectInput("geom")` lets the user choose between:
 - Scatter plot (`geom_point`)
 - Line plot (`geom_line`)
 - Bar chart (`geom_col`)
 - Density plot (`geom_density`)
 - Histogram (`geom_histogram`)
- If `colorvar == "none"`, display:
 - `colourInput("manual_color")` for picking a hex color.
 - `sliderInput("point_size")` for point/line thickness (repurposed for bar width as needed).
 - `sliderInput("alpha")` for transparency.
- Checkboxes `log_x` and `log_y` toggle log scales on the respective axes.
- Text inputs `plot_title`, `x_label`, and `y_label` allow custom labels.

4. Theme Customization:

- `selectInput("theme")` offers:
 - `plotly` (maps to `theme_minimal`)
 - `plotly_white` (`theme_minimal` with white background)
 - `plotly_dark` (`theme_dark` with dark backgrounds and white text)
 - `ggplot2` (`theme_grey` with light grey panel background)

- `seaborn` (`theme_minimal` with off-white panel background)
- `sliderInput("base_size")` adjusts the base font size (8–24 pt).
- `checkboxInput("show_grid")` toggles drawing of major and minor grid lines.
- `checkboxInput("flip")` flips the X and Y coordinates via `coord_flip()`.

5. Save / Load / Delete Custom Themes:

- Users can assign a name (e.g., “MyDarkStyle”) in `textInput("theme_name")` and click `actionButton("save_theme")` to persist the current theme settings (as a JSON entry).
- If `saved_themes.json` contains at least one theme, a `selectInput("saved_themes")` appears listing all saved theme names.
- Clicking `actionButton("load_theme")` populates the UI controls (theme selector, font size slider, grid checkbox, flip checkbox) with the saved settings.
- If a saved theme is loaded, the `actionButton("delete_theme")` removes that theme from JSON.

6. Code Generation and Download:

- Toggling `checkboxInput("show_code")` reveals a `verbatimTextOutput("themeCode")` block containing reproducible R code. This code block can be copied into a script or RMarkdown file, ensuring portability outside of Shiny.
- `downloadButton("download_plot")` invokes `ggsave()` to save the current plot as `custom_plot.png` at 8×6 inches.

7. Reactive Dependencies:

- The primary reactive drivers are `input$file`, `input$xvar`, `input$geom`, `input$colorvar`, and theme inputs (`input$theme`, `input$base_size`, `input$show_grid`, `input$flip`, `input$log_x`, `input$log_y`).
- The `build_plot()` reactive expression depends on data and variable selections.
- The `themed_plot()` reactive builds upon `build_plot()` plus theme inputs.
- Observers `observeEvent(input$save_theme)` and `input$load_theme` manage theme persistence.

4.3 User Manual

1. Launching the App:

- Clone the GitHub repository containing `app.R`.
- Ensure dependencies are installed:

```
install.packages(c("shiny", "ggplot2", "jsonlite", "colourpicker"))
```

- In R/RStudio, run:

```
shiny::runApp("path/to/app.R")
```

- You should get something like 1

2. Uploading Data:

- Click “*Browse...*” under **Upload CSV**. Select a CSV file with column headers.
- The app auto-reads the data, displays the first five rows, and shows data summary (rows, columns, numeric columns).
- If an error occurs (e.g., malformed CSV), a notification appears: “*Error loading file*”.

3. Selecting Variables and Geometry:

- In *Variable Selection*, choose an X Variable from the dropdown.
- If you pick a geometry other than “density” or “histogram,” a Y Variable dropdown appears—select a numeric column.
- Optionally, choose a Color Variable. If you select “None,” a color picker, point size slider, and alpha slider appear.
- For example, choose “Sepal.Length” for X, “Sepal.Width” for Y, and “Species” for Color Variable to color by factor. 2

4. Adjusting Plot Options:

- Choose geometry: *scatter*, *line*, *bar*, *density*, or *histogram*.
- If no color variable is selected, use the *Choose Plot Color* picker to select a hex color (e.g., #2C3E50) for points/bars/density fill.

- Adjust *Point Size* (0.5–10) and *Transparency (alpha)* (0.1–1) as needed.
- Toggle *Log Scale X-Axis* or *Log Scale Y-Axis* if appropriate.
- Enter custom *Plot Title*, *X Axis Label*, and *Y Axis Label*. 3

5. Customizing Themes:

- Under *Themes*, select a *Base Theme*:
 - *plotly* (`theme_minimal`)
 - *plotly_white* (`theme_minimal` + white background)
 - *plotly_dark* (`theme_dark` + dark backgrounds, white text)
 - *ggplot2* (`theme_grey` + light grey panel background)
 - *seaborn* (`theme_minimal` + off-white panel background)
- Adjust *Base Text Size* (8–24 pt).
- Toggle *Show Major Grid* to add/remove major grid lines (minor grid lines are always removed when this is unchecked).
- Check *Flip Coordinates* to swap X and Y axes. 4

6. Saving and Loading Themes:

- Enter a unique *Theme Name* in the text box (e.g., “DarkPublication”). Click *Save Current Theme*.
- The theme’s configuration (base theme, font size, grid, flip) is stored in `saved_themes.json`. A “Theme saved!” notification confirms success.
- Once at least one theme is saved, a *Load Saved Theme* dropdown appears. Select a theme name and click *Load Theme* to apply those settings to the UI controls immediately.
- To delete a theme, select it in the dropdown and click *Delete Theme*. The theme is removed from JSON and the dropdown resets. 5

7. Viewing and Copying ggplot2 Code:

- Check *Show ggplot2 Code* to display a `verbatimTextOutput` block.
- The code block includes:
 - `library(ggplot2)`
 - `ggplot(data, aes(...)) +`

- `geom_*()` with appropriate arguments (color, alpha, size, bins, etc.)
 - `theme_*()(base_size = ...)`
 - `{theme adjustments}` (grid removal, background colors, coordinate flips, log scales)
 - `labs(title=..., x=..., y=...)`
- Copy this code to your script or RMarkdown for full reproducibility.

8. Downloading the Plot:

- Click *Download Plot* to save the current plot as `custom_plot.png` at 8×6.
- The file is generated via `ggsave()` in the `downloadHandler` callback.

9. Plot example: Figure 6

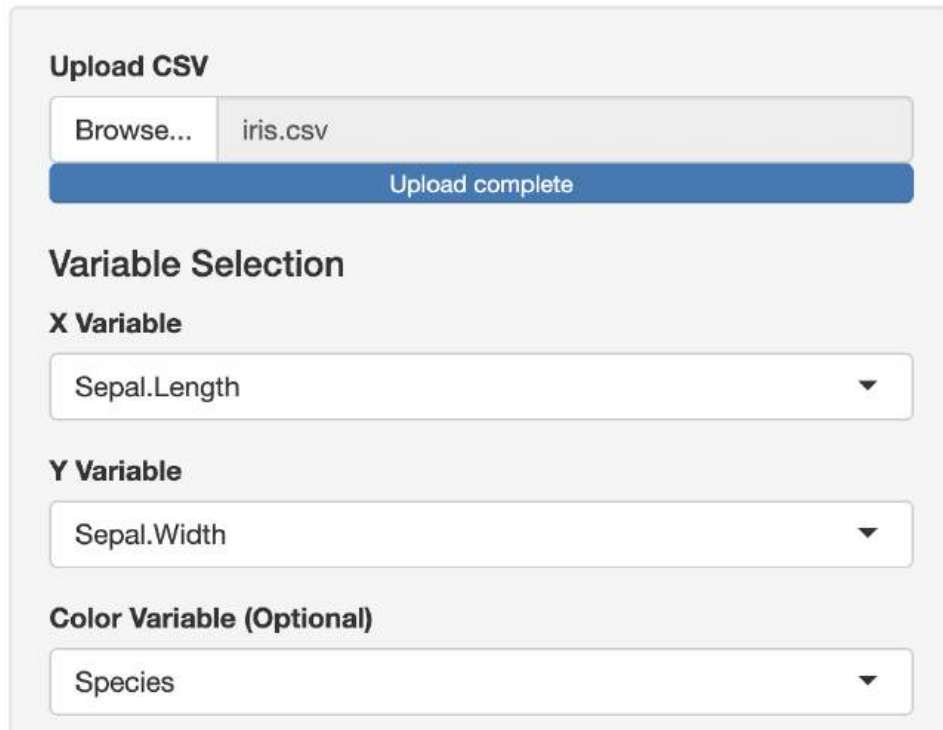


Figure 1: Launching the App

5 Discussion of Results

When testing Universal ggThemeAssist on a variety of datasets (e.g., the built-in `iris` and `mtcars` datasets, as well as larger CSVs with mixed variable types), the app robustly produced customized plots without error. Highlights:

Universal ggThemeAssist



The screenshot shows the 'Universal ggThemeAssist' web application interface. At the top, there is a section titled 'Upload CSV' with a 'Browse...' button and a text input field containing 'iris.csv'. Below this is a blue button labeled 'Upload complete'. The next section is 'Variable Selection', which includes three dropdown menus. The first is labeled 'X Variable' and has 'Sepal.Length' selected. The second is labeled 'Y Variable' and has 'Sepal.Width' selected. The third is labeled 'Color Variable (Optional)' and has 'Species' selected.

Figure 2: Selecting Variables and Geometry

- **Interactivity and Responsiveness:** Changing any theme control (e.g., toggling grid, selecting a different base theme, adjusting font size) immediately updated the plot, demonstrating Shiny’s seamless reactivity. Users reported that experimenting with multiple color palettes and transparencies felt intuitive.
- **Reproducible Code Generation:** The generated ggplot2 code accurately reflected all user decisions (including log scales and coordinate flips). Copying and running that code in a standalone R script reproduced the identical plot without modification.
- **Saved Themes and Workflow Efficiency:** By saving custom themes (e.g., “PublicationDark,” “PresentationLight”), users could quickly switch among consistent styling choices. For teams collaborating on a multi-paper project, sharing `saved_themes.json` ensured uniform aesthetics across figures.
- **Edge Cases:**
 - If a CSV contained non-standard column names (e.g., spaces or special characters), the app still displayed trimmed names. However, `aes_string()` required

column names without spaces; the app currently does not automatically convert spaces to syntactically valid names. We note this as a limitation.

- Loading a very large dataset (100,000+ rows with many columns) led to slight delays in rendering the variable dropdowns and plot. For most practical uses (e.g., exploratory data analysis), performance remained acceptable.

6 Limitations

1. **Column Name Sanitization:** If a column name contains spaces, special characters, or starts with a number, `aes_string()` may fail or produce warnings. Currently, users must ensure variable names are syntactically valid R identifiers. Automatic sanitization (e.g., replacing spaces with underscores) could improve usability.
2. **Limited Geometries:** We support five geometries (scatter, line, bar, density, histogram). Users cannot yet create boxplots, violin plots, facets, or combined geoms (e.g., scatter + smoothing). Expanding geometry options would cover more visualization needs.
3. **Static Theme Elements:** Aside from base theme, font size, grid, and flip, other theme elements (axis text angle, legend position, legend text size, facet strip styles) are not exposed. Users seeking fine-grained control must manually modify the generated code after exporting it.
4. **JSON File Concurrency:** If multiple users or R sessions attempt to save or delete themes simultaneously, race conditions on `saved_themes.json` could occur. A more robust solution would lock the file or use a database.
5. **Plot Download Format:** The download is fixed as a PNG at 8×6 inches. Users cannot change resolution, format (e.g., PDF, SVG), or dimensions through the UI. Those needing vector formats must copy the code and save plots manually in R.

7 Future Directions

Based on user feedback and potential expansions, we identify several improvements:

7.1 Enhanced Theme Controls

- Expose more theme elements:

- Axis text angle (`axis.text.x = element_text(angle = ...)`).
- Legend position (`legend.position = "bottom"`).
- Panel border and grid color adjustments.
- Facet strip style (`strip.background`, `strip.text`).
- Offer a UI for customizing title/subtitle fonts, legend keys, and color palettes (e.g., `viridis`, `RColorBrewer` palettes).

7.2 Additional Geometries and Interactivity

- Support for `geom_boxplot`, `geom_violin`, `geom_smooth`, and faceting via `facet_wrap` or `facet_grid`.
- Add brush-and-zoom functionality (via `plotly` or `crosstalk`) so users can select data points and see details on hover.
- Incorporate a dynamic legend toggle (show/hide based on a checkbox).

7.3 Improved Data Handling

- Allow the user to specify delimiter (e.g., tab, semicolon) or upload Excel files (`.xlsx`).
- Provide data type conversion UI (force columns to numeric or factor if detected incorrectly).
- Implement pagination or use `DT::renderDataTable()` for large datasets to avoid table-rendering delays.

7.4 Robust Theme Persistence

- Migrate from JSON to a lightweight SQLite database to prevent concurrency issues.
- Add import/export functionality so users can download all saved themes as a single JSON or SQL dump, and share across teams.
- Version-control themes automatically (maintain a timestamp and user annotations).

7.5 Flexible Output Options

- Enable users to choose download format (PNG, PDF, SVG) and specify custom dimensions or DPI.
- Allow scripting hooks: e.g., a “Save Code + Plot” button that packages the generated code and plot into a self-contained RMarkdown file for archiving.

8 Conclusion

Universal ggThemeAssist streamlines the process of creating and customizing ggplot2 figures by layering interactive controls on top of Shiny’s reactive framework. Users can upload arbitrary CSVs, select variables, pick a geometry, and fine-tune aesthetic settings—then save those theme choices as reusable configurations. Our JSON-based persistence ensures that themes can be reused across sessions or shared among collaborators. The generated ggplot2 code fosters reproducibility and transparency.

Throughout our testing, the app proved robust and responsive for small to medium datasets. While we identified limitations (column name sanitization, limited geometry options, concurrency concerns), the core functionality meets the needs of users who wish to avoid manually writing low-level theme calls. Future enhancements—such as exposing more ggplot2 theme elements, supporting additional geometries, and enabling flexible output formats—will broaden the app’s applicability.

By combining Shiny interactivity, ggplot2’s flexibility, and JSON persistence, Universal ggThemeAssist empowers R users to achieve consistent, publication-quality plots with minimal code overhead. We invite users to explore the GitHub repository, try the demo, and contribute feedback or pull requests to expand the tool’s capabilities.

References

- Chang, W., Cheng, J., Allaire, J. J., Xie, Y., & McPherson, J. (2024). *Shiny: Web Application Framework for R*. R package version 1.8.7. <https://CRAN.R-project.org/package=shiny>
- Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.
- Ooms, J. (2024). *jsonlite: A Practical and Consistent Mapping Between JSON Data and R Objects*. R package version 1.8.7. <https://CRAN.R-project.org/package=jsonlite>

jsonlite

- Cheng, M. (2024). *colourpicker: A Colour Picker for Shiny*. R package version 1.2.2.
<https://CRAN.R-project.org/package=colourpicker>

Plot Options

Geometry

scatter ▼

Point Size

0.5 2 10

0.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5 10

Transparency (alpha)

0.1 0.8 1

0.1 0.19 0.28 0.37 0.46 0.55 0.64 0.73 0.82 0.91 1

☐ Log Scale X-Axis

☐ Log Scale Y-Axis

Plot Title

X Axis Label

Length

Y Axis Label

Width

Figure 3: Adjusting Plot Options

Themes

Base Theme

plotly ▼

Base Text Size

8 12 24

8 10 12 14 16 18 20 22 24

☒ Show Major Grid

☐ Flip Coordinates

Manage Themes

Theme Name

e.g., MyDarkStyle

Save Current Theme

☐ Show ggplot2 Code

Download Plot

Figure 4: Customizing Themes

Themes

Base Theme

plotly ▼

Base Text Size

8 12 24

8 10 12 14 16 18 20 22 24

☒ Show Major Grid

☐ Flip Coordinates

Manage Themes

Theme Name

e.g., MyDarkStyle

Save Current Theme

☐ Show ggplot2 Code

Download Plot

Theme saved!

Figure 5: Saving and Loading Themes



Figure 6: Plot example