Supplementary Documentation for guttemap v 1.0

Michael Reichenheim

10 October 2025

1. Function Header and Initial Setup

```
# Function Definition and Initial Setup
# This section defines the `guttemap` function with customizable arguments for data
# input, sampling weights, graphical parameters, and output preferences.

guttemap <- function(data, ..., sw = NULL, scl = 1, sclmsg = FALSE,
                     grtit = "Guttman Error Map",
                     grfont = "Times New Roman", grsave = FALSE,
                     grty = "svg", titsize = 18, xlabsize = 12,
                     ylabsize = 12, valsize = 3.5, valformat = "%.1f",
                     item_position = "BOTTOM", item_order = "decreasing",
                     legposit = "bottom", legtit = "% error", legsize = 100,
                     obs = FALSE, mat = FALSE, tith = FALSE, itemh = FALSE,
                     nowarn = FALSE, smooth = 0) {

  # Explanation of Arguments:
  # - `data`: The primary dataset to be used, which should contain the item variables
  #   intended for analysis.
  #
  # - `...`: Specifies which item variables within `data` should be included in the
  #   analysis. This argument is flexible, allowing the user to input any number of
  #   variables from the dataset.
  #
  # - `sw`: An optional sampling weights variable. If provided, these weights are
  #   used to adjust the sample representativeness, scaling the data proportionally.
  #   Defaults to `NULL`, meaning no weights are applied unless specified.
  #
  # - `scl`: A scaling factor for weights (if `sw` is provided). This argument
  #   enables additional refinement of the weights, with a default value of `1`,
  #   meaning no further scaling is applied. Higher values can increase the scaling
  #   effect, but too high a value may lead to memory issues with data expansion.
  #
  # - `sclmsg`: A logical argument (`TRUE` or `FALSE`) that controls whether detailed
  #   messages regarding scaling are displayed. If set to `TRUE`, detailed messages
  #   about the scaling process are shown in the console.
  #
  # - `grtit`: The title of the Guttman Error Map. This character string sets the
  #   main title for the generated plot.
  #
```

```
# - `grfont`: The font to be used for the plot text, with a default of
#   `"Times New Roman"`.
#
# - `grsave`: If a character string is provided, this argument specifies the
#   filename for saving the plot. If left as `FALSE`, the plot is not saved.
#
# - `grty`: Specifies the file format of the saved plot. The default is `"svg"`,
#   but other formats like `"png"` or `"pdf"` can be specified if compatible with
#   `ggsave`.
#
# - `titsize`: Controls the font size of the plot title. Default is `18`.
#
# - `xlabsize`: Controls the font size for x-axis labels, with a default value
#   of `12`.
#
# - `ylabsize`: Controls the font size for y-axis labels, with a default value
#   of `12`.
#
# - `valsize`: Sets the font size for cell values displayed within the plot,
#   defaulting to `3.5`.
#
# - `valformat`: Controls the format for cell values in the plot. The default is
#   `"%.1f"`, indicating one decimal place.
#
# - `item_position`: Specifies the position of item labels on the plot's x-axis.
#   Options are `"TOP"` (display at the top) or `"BOTTOM"` (display at the bottom).
#   Default is `"BOTTOM"`.
#
# - `item_order`: Specifies the ordering of items in the Guttman Error Map.
#   Options are `"decreasing"` (default), which orders items by endorsement
#   frequency, or `"custom"`, which respects the order of variables entered
#   by the user in `...`.
#
# - `legposit`: Specifies the position of the legend on the plot. Common options
#   include `"bottom"`, `"top"`, `"left"`, or `"right"`.
#
# - `legtit`: Sets the legend title, with the default text being `"% error"`.
#
# - `legsize`: A percentage scaling factor for the legend size. Defaults to `100`
#   (original size); values can be increased or decreased proportionately.
#
# - `obs`: A logical argument; if `TRUE`, displays the effective sample size
#   in the console.
#
# - `mat`: A logical argument; if `TRUE`, displays the calculated matrices in the
#   console, including the Guttman Error and Loevinger's H matrices.
#
# - `tith`: A logical argument; if `TRUE`, adds the overall Loevinger’s H value
#   to the plot title.
#
# - `itemh`: A logical argument; if `TRUE`, displays Loevinger’s H for each item
#   in brackets beneath x-axis labels.
#
```

```
  # - `nowarn`: A logical argument; if `TRUE`, suppresses warnings from `coefH`.
  #   These warnings are about variations in the item scores across items, which
  #   may be normal in some analyses. Default is `FALSE`, so warnings are displayed
  #   by default.
  #
  # - `smooth`: Controls the level of smoothing applied to the Guttman Error Map.
  #   - `smooth = 0` displays a standard discrete heatmap.
  #   - `smooth = 100` creates a fully smoothed, continuous gradient heatmap.
  #   - Values between 0 and 100 apply partial smoothing, blending discrete tiles
  #     with a continuous gradient effect.
```

2. Load Required Libraries and Capture Item Variables

```
# Purpose:
# This section loads essential libraries required for data manipulation,
# visualization, and Mokken scale analysis. It also captures specified item
# variables from the dataset, verifies their presence, and subsets the data
# for further analysis.

# Code:
# Load necessary libraries
# The following libraries are used for plotting, data manipulation,
# graphics layout, and Mokken scale analysis.
library(ggplot2)       # Plotting library for complex, customizable plots
library(RColorBrewer)  # Provides color palettes for color gradients in plots
library(dplyr)         # Simplifies data filtering, selecting, and transforming
library(grid)          # Provides low-level graphical layout functions for R
library(mokken)        # Enables Mokken scale analysis and calculation of
                       # Loevinger's H coefficients

# Capture item variables specified in the function call
# '...' represents items to be included in the analysis
items <- rlang::quos(...)           # Capture item variables as quosures
item_names <- sapply(items, rlang::as_name)  # Convert item quosures to character
                                             # strings

# Check if each item variable exists in the dataset
# 'missing_items' identifies any specified variables that aren't found in 'data'.
missing_items <- setdiff(item_names, names(data))
if (length(missing_items) > 0) {
  stop(paste("Error: The following items are not found in the dataset:",
             paste(missing_items, collapse = ", ")))
}

# Select only specified items from the dataset
# 'data_subset' is the subset of 'data' containing only the specified items for
# analysis.
data_subset <- data[item_names]

# Explanation:
# - **Load Libraries**:
```

```
#   - This block imports the libraries needed for data manipulation, plot creation,
#     and Mokken scale analysis.
#   - `ggplot2` and `RColorBrewer` provide tools for plotting and color gradients,
#     while `dplyr` facilitates data filtering and transformations.
#   - `grid` offers low-level control of graphical layouts, useful for customizing
#     plot aesthetics.
#   - The `mokken` package includes functions for calculating Loevinger's H
#     coefficients, which are essential for Mokken scale analysis in this function.
#
# - **Capture Item Variables**:
#   - `items <- rlang::quos(...)` captures specified item variables as quosures.
#     This lets the function reference the variables without immediately evaluating
#     them.
#   - `item_names <- sapply(items, rlang::as_name)` converts each quosure to a
#     character string, producing a list of variable names that will be easy to
#     process in later steps.
#   - **Purpose**: Capturing item variables as quosures provides flexibility,
#     letting users specify any number of item variables in `...`, which is
#     especially useful for multi-item scale analysis.
#
# - **Verify Variables in Dataset**:
#   - `missing_items <- setdiff(item_names, names(data))` identifies any item names
#     in `item_names` that are missing from the dataset `data`.
#   - `if (length(missing_items) > 0) { ... }` stops the function with an error
#     message if any specified items are missing. This check ensures that the
#     function only attempts to analyze items present in `data`.
#   - **Purpose**: Ensures data integrity by verifying that specified item
#     variables exist in the dataset before proceeding.
#
# - **Subset Data**:
#   - `data_subset <- data[item_names]` creates a subset of `data` containing only
#     the specified item variables, making it ready for further analysis.
#   - **Purpose**: This step isolates the selected items for analysis, ensuring
#     that only relevant data columns are included, simplifying the analysis
#     process.
```

3. Handling Sampling Weight

```
# Purpose:
# This section processes sampling weights (`sw`) if provided, using a proportional
# scaling transformation. It scales weights based on a specified factor (`scl`),
# prepares the data by expanding rows according to the scaled weights, and displays
# feedback messages about the applied weights.

# Code:
# Check if sampling weights (sw) are provided and process them if available
if (!is.null(sw)) {
  sw <- as.numeric(na.omit(sw))  # Convert weights to numeric, omitting NAs

  if (length(sw) == 0) stop("The sampling weight 'sw' is missing, undefined, or
                            contains only NA values.")
```

```r
  # Capture and round the first 10 original sampling weights for display
  sw_original_first_10 <- round(head(sw, 10), 3)  # Display with 3 decimal places

  # Apply proportional scaling transformation with an optional scaling factor `scl`
  sw_min <- min(sw)
  SW_internal <- round((sw / sw_min) * scl)

  # Round to the nearest integer for use in data expansion
  SW_internal <- round(SW_internal)

  # Expand the data based on the scaled weights
  data_subset <- data_subset[rep(1:nrow(data_subset), times = SW_internal), ]

  # Display messages on sampling weights applied
  message("\nSampling weights applied.\n")
  message("Original sampling weights (first 10): ",
          paste(sw_original_first_10, collapse = ", "), "\n")
  message("Scaled internal weights (first 10): ",
          paste(head(SW_internal, 10), collapse = ", "), "\n")

  # Display a structured summary of the scaled internal weights
  message("Summary of scaled internal weights: ")
  summary_output <- capture.output(summary(SW_internal))
  for (line in summary_output) {
    message(line)
  }

  message("\nNumber of rows after expansion: ", nrow(data_subset), "\n")
}

# Explanation:
# - **Sampling Weight Check**:
#   - Checks if `sw` is provided. If not NULL, `sw` is processed; otherwise, it
#     is skipped.
#   - `sw <- as.numeric(na.omit(sw))` converts `sw` to numeric while removing any
#     NA values.
#   - If `sw` has only NA values or is empty, the function stops and provides an
#     error message.
#   - **Purpose**: Ensures that sampling weights, if specified, are in a usable
#     numeric format.

# - **Proportional Scaling Transformation**:
#   - Calculates `sw_min` and `sw_max` to capture the range of weights in `sw`.
#   - `R <- sw_max / sw_min` computes the ratio of `sw_max` to `sw_min`, which
#     is used to scale the weights.
#   - The scaling transformation `SW_internal <- (((R - 1) * sw) / (sw_max -
#     sw_min)) * scl` proportionally adjusts all weights, while setting the
#     smallest weight to 1.
#   - **Rounding**: The scaled weights are rounded to the nearest integer so they
#     can be used in data expansion.
#   - The `scl` argument provides further control over scaling. Larger values of
#     `scl` increase the scaling level; however, high values may cause excessive
#     data expansion, potentially leading to memory issues.
```

```
#   - **Purpose**: Retains the relative differences across observations in `sw`
#     while controlling for expansion size using `scl`.

# - **Data Expansion**:
#   - `data_subset <- data_subset[rep(1:nrow(data_subset), times = SW_internal),
#     ]` repeats each row in `data_subset` according to its corresponding weight
#     in `SW_internal`.
#   - **Purpose**: Expands the dataset to reflect the weighted representation of
#     each observation based on `sw`, thereby creating a dataset proportional to
#     the provided sampling weights.

# - **User Messages**:
#   - Provides feedback messages showing the first 10 original and scaled weights,
#     as well as a summary of the scaled weights.
#   - A message with the final number of rows in the expanded dataset is also
#     displayed.
#   - **Purpose**: Helps users understand the effect of the sampling weights, the
#     impact of scaling, and the final dataset size after expansion.
```

4. Handling Missing Data

```
# Purpose:
# This section removes rows with missing values from `data_subset`, ensuring that
# only complete data is used in subsequent analyses.

# Code:
# Remove any rows with missing values from the selected subset of data
data_subset <- na.omit(data_subset)

# Explanation:
# - **Purpose of Missing Data Handling**:
#   - This step removes rows with missing values to create a complete dataset,
#     avoiding potential issues in calculations that may arise from NA values.
#   - It ensures consistency and accuracy in the analysis, as Guttman error matrices
#     and Loevinger's H calculations depend on fully populated data.

# - **Functionality**:
#   - `na.omit(data_subset)` removes any row in `data_subset` that contains NA values.
#   - This function automatically omits incomplete rows, simplifying the data
#     preparation step and reducing the risk of NA-related issues during analysis.

# - **Outcome**:
#   - The resulting `data_subset` is a clean dataset free from missing values,
#     aligning with the requirements for generating reliable Guttman error matrices
#     and Loevinger's H coefficients.
```

5. Calculate and Order Item Endorsement Frequencies

```r
# Purpose:
# This section calculates the endorsement frequencies for each item in
# `data_subset`, then orders the items based on the `item_order` argument.
# Items can be ordered by decreasing endorsement frequency or retain their
# custom order, depending on user preference. This ordering enhances
# interpretability in the Guttman Error Map.

# Code:
# Compute the endorsement frequencies for each item in the dataset
endorsement_freq <- colMeans(data_subset, na.rm = TRUE)

# Determine item order based on the `item_order` argument
if (tolower(item_order) == "decreasing") {
  ordered_items <- names(sort(endorsement_freq, decreasing = TRUE))
} else if (tolower(item_order) == "custom") {
  ordered_items <- colnames(data_subset)  # Retain custom order
} else {
  stop("Invalid `item_order` argument. Use 'decreasing' or 'custom'.")
}

# Reorder data_subset based on the determined item order
data_subset <- data_subset[ordered_items]

# Explanation:
# - **Endorsement Frequency Calculation**:
#   - `endorsement_freq <- colMeans(data_subset, na.rm = TRUE)` calculates the
#     average endorsement (mean score) for each item across observations,
#     ignoring NA values.
#   - This metric helps identify items with higher average scores, showing
#     which items have stronger endorsement among respondents.
#   - **Purpose**: Calculating endorsement frequencies allows us to prioritize
#     items based on their endorsement strength when `item_order = "decreasing"`.

# - **Item Ordering**:
#   - `if (tolower(item_order) == "decreasing") { ... }` orders items by
#     decreasing endorsement frequency if the `item_order` argument is set to
#     `"decreasing"` (default).
#   - `else if (tolower(item_order) == "custom") { ... }` retains the order of
#     variables as provided by the user when `item_order = "custom"`.
#   - `ordered_items <- colnames(data_subset)` ensures the custom order is
#     preserved.
#   - `data_subset <- data_subset[ordered_items]` reorders `data_subset` to
#     match the determined order.
#   - **Purpose**: Allows flexibility in item ordering, enabling users to
#     choose the default endorsement-based order or retain a custom order.
```

6. Calculate Loevinger's H Coefficients and Guttman Error Matrix

```r
# Purpose:
# This section calculates Loevinger's H coefficients to assess item scalability
# and computes pairwise Guttman Error values for each item pair. Adjustments are
# made to ensure that any negative H values are set to zero and any values
# exceeding 1 are truncated to 1, thus keeping Guttman Error values within the
# 0-100% range. The results are used to generate the Guttman Error Map, with
# adjustments to the diagonal values for clarity.

# Code:
# Suppress console output temporarily while calculating Loevinger's H coefficients
if (nowarn) {  # Conditionally suppress warnings
  suppressWarnings({
    sink(tempfile())  # Suppress automatic printing of coefH result
    Hcoefs <- coefH(data_subset, se = FALSE)
    sink()  # Stop suppressing console output
  })
} else {
  sink(tempfile())  # Suppress automatic printing of coefH result
  Hcoefs <- coefH(data_subset, se = FALSE)
  sink()  # Stop suppressing console output
}

item_H_values <- Hcoefs$Hi  # Item-specific Loevinger's H

# Extract pairwise H coefficients (Hij)
pairwise_H <- Hcoefs$Hij

# Ensure no Hij values are negative by setting any negative values to 0
pairwise_H[pairwise_H < 0] <- 0

# Truncate any Hij values exceeding 1 by setting them to 1
pairwise_H[pairwise_H > 1] <- 1

# Calculate pairwise Guttman Error values based on adjusted H coefficients
pairwise_GE <- (1 - pairwise_H) * 100

# Set diagonal entries to NA to keep them blank in the matrix
diag(pairwise_GE) <- NA

# Explanation:
# - **Loevinger's H Calculation**:
#   - `Hcoefs <- coefH(data_subset, se = FALSE)` calculates Loevinger's H coefficients,
#     which measure the scalability of items in the dataset. This provides insight into
#     how well items align with a unidimensional hierarchical structure.
#   - `pairwise_H <- Hcoefs$Hij` extracts the pairwise H coefficients (Hij) from `Hcoefs`.
#   - **Purpose**: Calculating Loevinger's H coefficients helps assess item scalability,
#     a key component of the Guttman Error Map.

# - **Adjustment for Negative and Excessive H Values**:
#   - `pairwise_H[pairwise_H < 0] <- 0` ensures no negative values in the `pairwise_H`
#     matrix, preventing Guttman Error values from exceeding 100%.
```

```
#   - `pairwise_H[pairwise_H > 1] <- 1` truncates any values exceeding 1 to 1, keeping
#     the Guttman Error values within the 0-100 range.
#   - **Purpose**: These adjustments ensure that Guttman Error values remain within a
#     meaningful range (0-100), enhancing interpretability.

# - **Optional Warning Suppression (`nowarn` Argument)**:
#   - The new `nowarn` argument allows users to suppress certain warnings generated by
#     `coefH`.
#   - If `nowarn = TRUE`, any warnings about item score variations across categories are
#     suppressed by wrapping the `coefH` calculation in `suppressWarnings()`.
#   - **Purpose**: This option provides users with flexibility, letting them avoid
#     certain warnings if they are aware of or not concerned by minor data issues such
#     as varying item response categories.

# - **Guttman Error Matrix Calculation**:
#   - `pairwise_GE <- (1 - pairwise_H) * 100` converts Loevinger's H values to Guttman
#     Error percentages for each item pair. This matrix represents the degree of error
#     in item pair ordering.
#   - **Purpose**: The Guttman Error Matrix serves as the basis for the error map,
#     visually indicating item consistency and the overall hierarchy.

# - **Diagonal Adjustment**:
#   - `diag(pairwise_GE) <- NA` sets the diagonal values of `pairwise_GE` to `NA`, so
#     they appear blank in the matrix. Diagonal values represent self-comparisons and
#     are therefore irrelevant.
#   - **Purpose**: Setting diagonal entries to `NA` ensures the plot focuses only on
#     meaningful pairwise comparisons, improving readability and interpretability.
```

7. Display Effective Sample Size and Optional Matrices

```
# Purpose:
# This section displays the effective sample size if `obs = TRUE` and
# conditionally prints detailed matrix outputs if `mat = TRUE`. This
# includes the Pairwise Guttman Error Matrix, Pairwise Loevinger's H
# Matrix, item-specific H values, and overall scale H. These outputs
# provide insight into item scalability and hierarchy.

# Code:
# Display effective sample size if `obs = TRUE`
if (obs) {
  cat("\nThe effective sample size is N =", nrow(data_subset), "\n")
}

# Conditionally print matrices based on `mat` argument
  # Conditionally print matrices based on the 'mat' argument
  if (mat) {
    cat("\nGuttman Error Matrix (rounded to 1 decimal place):\n")
    print(round(pairwise_GE, 1)) # Rounded to 1 decimal

    cat("\nPairwise Loevinger's Hij Matrix (rounded to 3 decimal places):\n")
    print(round(pairwise_Hij, 3))    # Rounded to 3 decimals
```

```
    cat("\Item Loevinger's Hi Values (rounded to 3 decimal places):\n")
    print(round(Hcoefs$Hi, 3))     # Rounded to 3 decimals

    cat("\nOverall Scale Loevinger's H (rounded to 3 decimal places):\n")
    print(round(Hcoefs$H, 3))      # Rounded to 3 decimals
  }

# Explanation:
# - **Effective Sample Size**:
#   - `if (obs) { ... }` checks if `obs = TRUE`. If so, the function outputs
#     the effective sample size, which is the number of rows in `data_subset`
#     after any adjustments.
#   - **Purpose**: Displaying the sample size is useful for understanding the
#     scope of the data after expansion or filtering steps.

# - **Conditional Matrix Display**:
#   - `if (mat) { ... }` conditionally displays detailed matrix outputs if
#     `mat = TRUE`.
#   - **Matrices Printed**:
#     - Pairwise Guttman Error Matrix (`pairwise_GE`): Displays pairwise
#       error percentages.
#     - Pairwise Loevinger's H Matrix (`pairwise_H`): Shows scalability (H)
#       values for each item pair.
#     - Item-specific Loevinger's H values (`Hcoefs$Hi`): Lists H values for
#       individual items.
#     - Overall Scale Loevinger's H (`Hcoefs$H`): Displays the overall
#       scalability for the entire scale.
#   - **Purpose**: These matrices offer in-depth detail about item
#     scalability and consistency, supporting a thorough analysis of the
#     data when required.
#   - **Use Case**: Displaying these matrices conditionally allows users to
#     access detailed information without cluttering the console if it's
#     not needed.
```

8. Preparing Item Names and Axis Positioning

```
# Purpose:
# This section prepares the item names and their positioning for the heatmap plot.
# Items are ordered based on the `item_order` argument, ensuring consistency
# between the dataset and the plot. Additionally, axis positioning is set based
# on the `item_position` argument, enabling customization of label placement.

# Code:
# Determine item order based on the `item_order` argument
if (tolower(item_order) == "decreasing") {
  ordered_items <- names(sort(endorsement_freq, decreasing = TRUE))
} else if (tolower(item_order) == "custom") {
  ordered_items <- colnames(data_subset)  # Retain custom order
} else {
  stop("Invalid `item_order` argument. Use 'decreasing' or 'custom'.")
```

```
}

# Set up factor levels for correct ordering in the plot
heatmap_data$X <- factor(heatmap_data$X, levels = ordered_items)
heatmap_data$Y <- factor(heatmap_data$Y, levels = rev(ordered_items))

# Define axis positioning based on `item_position` argument
axis_position <- if (toupper(item_position) == "TOP") {
  list(
    theme(
      axis.text.x.top = element_text(size = xlabsize, angle = 0, vjust = 0.5),
      axis.title.x.top = element_blank(),
      axis.title.x = element_blank()
    ),
    scale_x_discrete(position = "top")
  )
} else {
  list(
    theme(
      axis.text.x = element_text(size = xlabsize, angle = 0, hjust = 0.5),
      axis.title.x = element_blank()
    )
  )
}

# Explanation:
# - **Ordering Items**:
#   - `if (tolower(item_order) == "decreasing") { ... }` orders items by
#     decreasing endorsement frequency if `item_order = "decreasing"`.
#   - `else if (tolower(item_order) == "custom") { ... }` retains the order of
#     variables as provided by the user when `item_order = "custom"`.
#   - `ordered_items <- colnames(data_subset)` ensures the custom order is
#     preserved.
#   - `heatmap_data$X <- factor(heatmap_data$X, levels = ordered_items)` and
#     `heatmap_data$Y <- factor(heatmap_data$Y, levels = rev(ordered_items))`
#     set the correct item order for the plot axes.
#   - **Purpose**: Ensures the plot reflects the specified item ordering,
#     aligning with user preference and the dataset.

# - **Axis Positioning**:
#   - `axis_position` is defined based on the `item_position` argument, which can
#     be set to "TOP" to display labels at the top or "BOTTOM" to place them at
#     the bottom.
#   - The `theme` settings adjust text size, alignment, and remove unnecessary
#     axis titles.
#   - **Purpose**: Provides flexibility in label placement, improving readability
#     and alignment with user preferences for plot aesthetics.
```

9. Creating the Heatmap Plot

```
# Purpose:
# This section generates the heatmap plot based on the Guttman Error Matrix, using a
# color gradient to represent error percentages. The plot aesthetics, including axis
# positions, background, and legends, are customized for clarity and visual appeal.

# Code:
# Determine plot layers based on `smooth` parameter
plot_layer <- if (smooth >= 100) {
  # Maximum smooth: continuous gradient
  geom_raster(interpolate = TRUE)
} else if (smooth > 0) {
  # Partial smooth: blend using transparent tiles
  list(
    geom_tile(color = NA, alpha = 1 - smooth / 100),
    geom_raster(interpolate = TRUE, alpha = smooth / 100)
  )
} else {
  # No smooth: discrete boxes
  geom_tile(color = "white")
}

# Create the heatmap plot with color gradient and themes
p <- ggplot(heatmap_data, aes(X, Y, fill = Value)) +
  plot_layer +  # Apply smoothing logic based on `smooth` parameter
  geom_text(aes(label = ifelse(is.na(Value), "",
            sprintf(valformat, Value))), size = valsize) +
  scale_fill_gradientn(
    colors = colorRampPalette(c("#63BE7B", "#D1DA92", "#FFEB84", "#F8696B"))(100),
    limits = c(0, 100),
    breaks = seq(0, 100, by = 10),
    na.value = "white"  # Set diagonal boxes to white
  ) +
  theme_minimal(base_family = grfont) +
  axis_position +  # Add axis positioning setup
  theme(
    plot.background = element_rect(fill = "#f0f0f0", color = NA),  # Light grey bg
    panel.border = element_rect(color = "black", fill = NA, linewidth = 1),  # Outer frame
    plot.margin = margin(20, 20, 20, 20),  # Set wider margins for balance
    axis.text.y = element_text(size = ylabsize),
    axis.title.y = element_blank(),
    plot.title = element_text(size = titsize, face = "bold", hjust = 0.5),
    legend.position = legposit,  # Updated to use legposit
    legend.key.height = if (legposit %in% c("top", "bottom")) {
      unit(0.75 * legsize, "cm")  # Adjust legend key height based on legsize
    } else {
      unit(2.5 * legsize, "cm")  # Adjust for right or left positions
    },
    legend.key.width = if (legposit %in% c("top", "bottom")) {
      unit(2.5 * legsize, "cm")  # Adjust for top or bottom positions
    } else {
      unit(0.75 * legsize, "cm")  # Adjust for right or left positions
```

```
    },
    legend.text = element_text(size = 12 * legsize),  # Scale legend text
    legend.title = element_text(size = 12 * legsize)  # Scale legend title
  ) +
  labs(title = grtit, fill = legtit)  # Updated legend title

# Explanation:
# - **Color Gradient and Theme**:
#   - `scale_fill_gradientn` applies a color gradient from green to red to represent
#     error percentages, providing an intuitive visual cue. Lower errors are shown in
#     green, transitioning to red for higher errors.
#   - `theme_minimal` is applied for a clean plot background, and additional `theme`
#     settings further customize the appearance for readability and aesthetics.
#   - **Purpose**: The color gradient enhances the interpretability of error percentages
#     by providing a clear visual distinction across the range of values.

# - **Smoothing**:
#   - `plot_layer` determines the appearance of the heatmap based on the `smooth` parameter.
#   - `smooth = 0`: Displays a discrete heatmap with each cell individually colored.
#   - `smooth = 100`: Produces a fully smoothed, continuous gradient heatmap.
#   - `smooth` values between 0 and 100 apply a blended effect where cells are partially
#     smooth, balancing between discrete tiles and a continuous gradient effect.
#   - **Purpose**: The `smooth` parameter allows flexibility in the plot's appearance,
#     enabling users to select a display style that best fits their interpretative needs.

# - **Axis Position and Margins**:
#   - `axis_position` applies the customized axis positioning from Section 8, allowing
#     labels to appear at the top or bottom as specified by `item_position`.
#   - `plot.margin = margin(20, 20, 20, 20)` adjusts the margins to ensure proportional
#     spacing between plot elements.
#   - **Purpose**: Customizing axis positioning and margins enhances plot balance and
#     readability.

# - **Legend and Titles**:
#   - `labs(title = grtit, fill = legtit)` sets the main plot title and the updated
#     legend title for the color gradient.
#   - `legend.position = legposit`, `legend.key.height`, and `legend.key.width` are
#     modified to adjust legend placement and proportions, ensuring the legend is
#     appropriately sized and positioned.
#   - **Purpose**: Titles and legend customization make the heatmap informative and
#     balanced, helping users interpret the plot effectively.
```

10. Save the Heatmap Plot

```
# Purpose:
# This section saves the heatmap plot to a specified file if the `grsave` argument
# is provided as a string, allowing users to store the plot in various formats.
# Error handling ensures that any issues during saving are reported.

# Code:
# Save the plot if `grsave` is specified
```

```r
if (is.character(grsave)) {
  save_path <- paste0(grsave, ".", grty)
  tryCatch({
    # Save the plot to the specified file path and format
    ggsave(filename = save_path, plot = p, width = 10, height = 8)
    message("Plot saved successfully!")
  }, error = function(e) {
    # Handle any error that occurs during saving
    message("Failed to save the plot: ", e$message)
  })
}

# Explanation:
# - **Saving the Plot**:
#   - `if (is.character(grsave))` checks if `grsave` is specified as a string. If so,
#     the function saves the plot to a file.
#   - `save_path <- paste0(grsave, ".", grty)` constructs the full filename by
#     combining `grsave` (the base filename) with `grty` (the file format).
#   - `ggsave(filename = save_path, plot = p, width = 10, height = 8)` saves the plot
#     with specified dimensions (10x8 units).
#   - **Purpose**: Saving the plot allows users to preserve the output in various
#     formats for sharing or further analysis.

# - **Error Handling**:
#   - `tryCatch` manages any errors that occur during saving. If successful, it
#     displays a confirmation message (`"Plot saved successfully!"`).
#   - If an error occurs, an error message is shown, including the specific reason
#     for the failure (`e$message`).
#   - **Purpose**: Error handling improves user experience by confirming a successful
#     save or reporting any issues during the save process.
```

11. Return Statement

```r
# Purpose:
# This section returns the Guttman Error Matrix and the generated heatmap plot.
# Providing these as outputs enables further analysis or visualization by the user.

# Code:
# Only return the Guttman Error (GE) matrix and the plot
return(list(matrix = pairwise_GE, plot = p))

# Explanation:
# - **Return Output**:
#   - `return(list(matrix = pairwise_GE, plot = p))` provides both the
#     `pairwise_GE` matrix (Guttman Error Matrix) and the `p` plot (heatmap
#     plot) as a list.
#   - This enables users to access the computed matrix and the plot directly,
#     allowing further programmatic manipulation, analysis, or re-visualization
#     if desired.
#   - **Purpose**: By returning these outputs, the function is versatile,
```

```
#     supporting not only immediate visualization but also further use of the
#     computed data and plot.
```

Appendix: complete code

```
guttemap <- function(data, ..., sw = NULL, scl = 1, sclmsg = FALSE,
                     grtit = "Guttman Error Map", grfont = "Times New Roman",
                     grsave = FALSE, grty = "svg", titsize = 18, xlabsize = 12,
                     ylabsize = 12, valsize = 3.5, valformat = "%.1f",
                     item_position = "BOTTOM", item_order = "decreasing",
                     legposit = "bottom", legtit = "% error", legsize = 100,
                     obs = FALSE, mat = FALSE, tith = FALSE, itemh = FALSE,
                     nowarn = FALSE, smooth = 0) {

  # Load required libraries
  library(ggplot2)
  library(ggtext)
  library(RColorBrewer)
  library(dplyr)
  library(grid)
  library(mokken)
  library(rlang)
  library(extrafont)

  # Capture item variables and check if they exist in the dataset
  items <- rlang::quos(...)
  item_names <- sapply(items, rlang::as_name)

  # Check for missing variables in the dataset
  missing_items <- setdiff(item_names, names(data))
  if (length(missing_items) > 0) {
    stop(paste("Error: The following items are not found in the dataset:",
               paste(missing_items, collapse = ", ")))
  }

  # Select variables from the dataset
  data_subset <- data[item_names]
  original_sample_size <- nrow(data_subset)

  # Handle sampling weights if provided
  if (!is.null(sw)) {
    sw <- as.numeric(na.omit(sw))
    if (length(sw) == 0) stop("The sampling weight 'sw' is missing, undefined, or contains only NA va

    sw_original_first_10 <- round(head(sw, 10), 3)
    sw_min <- min(sw)
    SW_internal <- round((sw / sw_min) * scl)
    data_subset <- data_subset[rep(1:nrow(data_subset), times = SW_internal), ]

    if (sclmsg) {
      message("\nSampling weights applied. Provide 'sclmsg=TRUE' for details.\n")
```

```
    message("Original sampling weights (first 10): ", paste(sw_original_first_10, collapse = ", "),
    message("Scaled internal weights (first 10): ", paste(head(SW_internal, 10), collapse = ", "),
    message("Summary of original weights: \n")
    message(paste(capture.output(summary(sw)), collapse = "\n"))
    message("\nSummary of scaled internal weights: \n")
    message(paste(capture.output(summary(SW_internal)), collapse = "\n"))
  }
}

# Handle missing data by omitting rows with missing values
data_subset <- na.omit(data_subset)
expanded_sample_size <- nrow(data_subset)

# Display sample size if `obs = TRUE`
if (obs) {
  if (is.null(sw)) {
    message("\nThe data sample size is N = ", original_sample_size)
  } else {
    message("\nThe data sample size is N = ", original_sample_size,
            ". Since sw has been requested, the expanded N = ", expanded_sample_size)
  }
}

# Compute Loevinger's H coefficients, optionally suppressing warnings
if (nowarn) {
  suppressWarnings({
    sink(tempfile())  # Suppress automatic printing of coefH result
    Hcoefs <- coefH(data_subset, se = FALSE)
    sink()  # Stop redirecting output
  })
} else {
  sink(tempfile())
  Hcoefs <- coefH(data_subset, se = FALSE)
  sink()
}

item_H_values <- Hcoefs$Hi  # Item-specific Loevinger's H

# Compute item endorsement frequencies and order items based on item_order
if (item_order == "decreasing") {
  endorsement_freq <- colMeans(data_subset, na.rm = TRUE)
  ordered_items <- names(sort(endorsement_freq, decreasing = TRUE))
} else if (item_order == "custom") {
  ordered_items <- item_names
} else {
  stop("Invalid value for 'item_order'. Use 'decreasing' or 'custom'.")
}

data_subset <- data_subset[ordered_items]

# Create x_labels with slightly smaller font size for H values if itemh = TRUE
if (itemh) {
  x_labels <- paste0(
```

```
    ordered_items, "<br><span style='font-size:",
    xlabsize * 0.85, "pt;'>(", formatC(item_H_values[ordered_items],
                                       format = "f", digits = 3), ")</span>"
  )
} else {
  x_labels <- ordered_items
}

# Set the overall Loevinger's H value in the title if tith = TRUE
if (tith) {
  grtit <- paste0(grtit, " (H = ", sprintf("%.3f", Hcoefs$H), ")")
}

# Prepare matrices for console display
pairwise_Hij <- Hcoefs$Hij
pairwise_Hij[pairwise_Hij < 0] <- 0

# Adjust Hij values greater than 1 to exactly 1
pairwise_Hij[pairwise_Hij > 1] <- 1

# Compute Guttman Error percentages
pairwise_GE <- (1 - pairwise_Hij) * 100
diag(pairwise_GE) <- NA

# Conditionally print matrices based on the 'mat' argument
if (mat) {
  cat("\nGuttman Error Matrix (rounded to 1 decimal place):\n")
  print(round(pairwise_GE, 1)) # Rounded to 1 decimal

  cat("\nPairwise Loevinger's Hij Matrix (rounded to 3 decimal places):\n")
  print(round(pairwise_Hij, 3))   # Rounded to 3 decimals

  cat("\Item Loevinger's Hi Values (rounded to 3 decimal places):\n")
  print(round(Hcoefs$Hi, 3))    # Rounded to 3 decimals

  cat("\nOverall Scale Loevinger's H (rounded to 3 decimal places):\n")
  print(round(Hcoefs$H, 3))     # Rounded to 3 decimals
}

# Prepare data for plotting
heatmap_data <- as.data.frame(as.table(pairwise_GE))
colnames(heatmap_data) <- c("X", "Y", "Value")

# Set factor levels for correct order in the plot
heatmap_data$X <- factor(heatmap_data$X, levels = ordered_items)
heatmap_data$Y <- factor(heatmap_data$Y, levels = rev(ordered_items))

# Modify x-axis based on item position
axis_position <- if (toupper(item_position) == "TOP") {
  list(
    theme(
      axis.text.x.top = element_markdown(size = xlabsize, angle = 0, vjust = 0.5),
      axis.title.x.top = element_blank(),
```

```
      axis.title.x = element_blank(),
      plot.margin = margin(40, 20, 20, 20)
    ),
    scale_x_discrete(position = "top", labels = x_labels)
  )
} else {
  list(
    theme(
      axis.text.x = element_markdown(size = xlabsize, angle = 0, hjust = 0.5),
      axis.title.x = element_blank()
    ),
    scale_x_discrete(labels = x_labels)
  )
}

# Adjust legend size and position
legend_scale <- legsize / 100
legend_dimensions <- if (legposit %in% c("right", "left")) {
  list(legend.key.height = unit(1.5 * legend_scale, "cm"),
       legend.key.width = unit(0.6 * legend_scale, "cm"))
} else {
  list(legend.key.height = unit(0.75 * legend_scale, "cm"),
       legend.key.width = unit(2.5 * legend_scale, "cm"))
}

# Determine plot layers based on smooth parameter
plot_layer <- if (smooth >= 100) {
  geom_raster(interpolate = TRUE)
} else if (smooth > 0) {
  list(
    geom_tile(color = NA, alpha = 1 - smooth / 100),
    geom_raster(interpolate = TRUE, alpha = smooth / 100)
  )
} else {
  geom_tile(color = "white")
}

# Create the heatmap plot
p <- ggplot(heatmap_data, aes(X, Y, fill = Value)) +
  plot_layer +
  geom_text(aes(label = ifelse(is.na(Value), "", sprintf(valformat, Value))),
            size = valsize, color = "black") +
  scale_fill_gradientn(
    colors = colorRampPalette(c("#63BE7B", "#D1DA92", "#FFEB84", "#F8696B"))(100),
    limits = c(0, 100),
    breaks = seq(0, 100, by = 10),
    na.value = "white"
  ) +
  theme_minimal(base_family = grfont) +
  axis_position +
  theme(
    plot.background = element_rect(fill = "#f0f0f0", color = NA),
    panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
```

```
      plot.margin = margin(20, 20, 20, 20),
      axis.text.y = element_text(size = ylabsize),
      axis.title.y = element_blank(),
      plot.title = element_text(size = titsize, face = "bold", hjust = 0.5),
      legend.position = legposit,
      legend.text = element_text(size = 12 * legend_scale),
      legend.title = element_text(size = 12 * legend_scale),
      !!!legend_dimensions
    ) +
    labs(title = grtit, fill = legtit)

  # Display and save plot
  print(p)
  if (is.character(grsave)) {
    save_path <- paste0(grsave, ".", grty)
    tryCatch({
      ggsave(filename = save_path, plot = p, width = 10, height = 8)
      message("Plot saved successfully!")
    }, error = function(e) {
      message("Failed to save the plot: ", e$message)
    })
  }

  return(list(matrix = pairwise_GE, plot = p))
}
```