Summary Plot for Faba Bean Correction Version 4

Hao Nan Tobey Wang – haonantobey.wang@agr.gc.ca

2025-12-16

```
rm(list=ls())
```

```
# --- 0. Load Libraries and Setup ---
library(dplyr)
library(tidyr)
library(ggplot2)
library(gridExtra)
library(boot)
library(readxl)
library(scales) # For alpha()
library(here) # For robust path management
library(rlang) # For Non-Standard Evaluation (NSE) fixes in ggplot
library(writexl) # For saving the summary table
library(data.table) # For efficient data manipulation (Outlier detection)

# --- Global Definitions ---
# 1. Define ALL methods (for Length and Width)
methods_full <- c("GT-MM", "GT-DM", "CP", "SVD", "min", "colorbox")

# 2. Define Area methods (Excluding GT-DM and colorbox)
methods_area <- c("GT-MM", "CP", "SVD", "min")

measures <- c("Length", "Width", "Area")

# Update Palette to include all potential keys
palette <- c(
  "GT-MM" = "#6a3d9a",
  "GT-DM" = "#956",
  "CP" = "#1b9e77",
  "SVD" = "#d95f02",
  "min" = "#7570b3",
  "colorbox" = "#a6761d"
  # "CP-Affine" = "#33a02c",
  # "SVD-Affine" = "#553300",
  # "min-Affine" = "#880000"
)
unit_labels <- c("Length" = " (mm)", "Width" = " (mm)", "Area" = " (mm\u00B2)")

# --- Output Directory Setup ---
current_date <- format(Sys.Date(), "%Y-%m-%d")
output_dir <- here::here(paste0("plots_output_section1_", current_date))
```

```r
# Placeholder for removal tracking
removed_seeds_log <- list()
outlier_threshold_data <- list() # To store IQR/SD datax`

# ------------------------------------------------
# 1. Read and Prepare Data (INITIAL CLEANING ONLY)
# ------------------------------------------------
# Ensure file exists
file_path <- "Summary_Faba Bean Correction.xlsx"
if (!file.exists(file_path)) stop("Excel file not found!")

df <- read_excel(file_path, sheet = "Sheet7")

# Extract Group
df <- df %>%
    mutate(Group2 = sub("-(\\d+)$", "", Code))

# Remove specific groups (Manual exclusion)
groups_to_remove <- c("460", "117", "198", "619", "283", "90", "615", "456", "280", "139", "620")
manual_exclusion_pattern <- paste0("Faba-Seed-CC_Vf(", paste(groups_to_remove, collapse="|"), ")")

# Log manual exclusions
manually_removed_groups <- df %>%
    filter(grepl(manual_exclusion_pattern, Group2)) %>%
    select(Code, Group2) %>%
    distinct(Code, .keep_all = TRUE) %>%
    mutate(Reason = "Manual Group Exclusion", Threshold = NA_character_)

removed_seeds_log[[1]] <- manually_removed_groups

df <- df %>%
    filter(!grepl(manual_exclusion_pattern, Group2))

# --- 1.1 DYNAMIC COLUMN SELECTION ---
ideal_cols_length <- paste0("Length-", methods_full)
ideal_cols_width <- paste0("Width-", methods_full)
ideal_cols_area <- paste0("Area-", methods_area)

ideal_cols_all <- c(ideal_cols_length, ideal_cols_width, ideal_cols_area)

# Intersect with what actually exists in the Excel file
cols_to_convert <- intersect(ideal_cols_all, names(df))

# 1.2 Convert to numeric and REPLACE 0 with NA_real_
df_clean <- df %>%
    mutate(across(all_of(cols_to_convert), ~ as.numeric(trimws(as.character(.))))) %>%
    mutate(across(all_of(cols_to_convert), ~ ifelse(. == 0, NA_real_, .)))

message(paste("Initial data cleaning complete. Manual exclusions applied, 0s converted to NA."))

# ------------------------------------------------
# 1.3 Outlier Detection (Standard Boxplot Method: 1.5 * IQR)
# ------------------------------------------------
```

```r
outlier_log <- list()
for (col in cols_to_convert) {
    if (col %in% names(df_clean)) {
        data <- df_clean[[col]]
        # Use boxplot.stats for standard outlier detection (1.5 * IQR rule)
        stats <- boxplot.stats(data)$stats
        q1 <- stats[2]
        q3 <- stats[4]
        iqr <- q3 - q1

        lower_bound <- q1 - 1.5 * iqr
        upper_bound <- q3 + 1.5 * iqr

        # Store threshold data
        outlier_threshold_data[[col]] <- data.frame(
            Column = col,
            Q1 = q1,
            Q3 = q3,
            IQR_x_1_5 = 1.5 * iqr,
            Lower_Bound = lower_bound,
            Upper_Bound = upper_bound
        )

        # Identify outliers
        outliers_indices <- which(data < lower_bound | data > upper_bound)

        if (length(outliers_indices) > 0) {
            outliers_df <- df_clean[outliers_indices, ] %>%
                select(Code) %>%
                mutate(
                    Reason = paste("Outlier (1.5xIQR) in column:", col),
                    Threshold = paste0("[", round(lower_bound, 3), ", ", round(upper_bound, 3), "]")
                )
            outlier_log[[col]] <- outliers_df
        }
    }
}
removed_seeds_log[[2]] <- do.call(bind_rows, outlier_log)
message("Outlier detection complete.")

# ------------------------------------------------
# 2. Sorting Function (Ensures seed alignment within group)
# ------------------------------------------------
sort_within_group <- function(df, cols) {
    # Only sort columns that exist in the dataframe
    existing_cols <- intersect(cols, names(df))

    if(length(existing_cols) == 0) return(df)

    # Drop rows where ALL sorting columns are NA within the group before sorting
    df_no_na_groups <- df %>%
        group_by(Group2) %>%
        filter(sum(!is.na(c_across(all_of(existing_cols)))) > 0) %>%
```

```
        ungroup()

    df_sorted <- df_no_na_groups %>%
        group_by(Group2) %>%
        mutate(across(all_of(existing_cols), ~ sort(.x, na.last = TRUE))) %>%
        ungroup()

    return(df_sorted)
}

df_sorted <- df_clean %>%
    sort_within_group(ideal_cols_length) %>%
    sort_within_group(ideal_cols_width) %>%
    sort_within_group(ideal_cols_area)

message("Applied within-group sorting.")

# ------------------------------------------------
# 3. Localized Filtering Function (Group-wise NA removal per comparison)
# ------------------------------------------------
localized_group_filter <- function(df, target_cols) {
    if (length(target_cols) == 0) return(df)

    # Ensure target cols exist
    target_cols <- intersect(target_cols, names(df))
    if(length(target_cols) == 0) return(df)

    # 1. Find groups that have ANY NA in ANY of the target columns
    na_groups_to_remove <- df %>%
        rowwise() %>%
        mutate(has_na_in_cols = any(is.na(c_across(all_of(target_cols))))) %>%
        ungroup() %>%
        filter(has_na_in_cols) %>%
        pull(Group2) %>%
        unique()
    # 2. Filter the original dataframe
    df_filtered_local <- df %>%
        filter(!(Group2 %in% na_groups_to_remove))
    return(df_filtered_local)
}


# ------------------------------------------------
# 4. Regression metrics function (Bias, RMSE, R2, Slope, Intercept with CIs)
# ------------------------------------------------
regression_metrics_full <- function(gt, pred, nboot=2000){
    df_tmp <- data.frame(GT=gt, Pred=pred) %>% drop_na()

    if (nrow(df_tmp) < 3) { # Need at least 3 points for confidence intervals/bootstrapping
        return(list(
            R2 = NA_real_, Bias = NA_real_, RMSE = NA_real_, Slope = NA_real_, Intercept = NA_real_,
            Reg_Slope_CI_Low = NA_real_, Reg_Slope_CI_High = NA_real_,
            Reg_Intercept_CI_Low = NA_real_, Reg_Intercept_CI_High = NA_real_,
```

```
            Boot_Slope_CI_Low = NA_real_, Boot_Slope_CI_High = NA_real_,
            Boot_Intercept_CI_Low = NA_real_, Boot_Intercept_CI_High = NA_real_,
            Boot_R2_CI_Low = NA_real_, Boot_R2_CI_High = NA_real_
        ))
    }

    diff <- df_tmp$Pred - df_tmp$GT
    bias <- mean(diff)
    rmse <- sqrt(mean(diff^2))

    fit <- lm(Pred ~ GT, data=df_tmp)
    sum_fit <- summary(fit)
    reg_ci <- confint(fit, level = 0.95)
    r2 <- sum_fit$r.squared

    boot_fun <- function(data, indices){
        d <- data[indices,]
        fit_boot <- lm(Pred ~ GT, data=d)
        c(coef(fit_boot), summary(fit_boot)$r.squared)
    }

    boot_res <- tryCatch(boot(df_tmp, boot_fun, R=nboot), error=function(e) NULL)

    if (is.null(boot_res)) {
          return(list(
            R2 = r2, Bias = bias, RMSE = rmse, Slope = coef(fit)[2], Intercept = coef(fit)[1],
            Reg_Slope_CI_Low = reg_ci[2, 1], Reg_Slope_CI_High = reg_ci[2, 2],
            Reg_Intercept_CI_Low = reg_ci[1, 1], Reg_Intercept_CI_High = reg_ci[1, 2],
            Boot_Slope_CI_Low = NA_real_, Boot_Slope_CI_High = NA_real_,
            Boot_Intercept_CI_Low = NA_real_, Boot_Intercept_CI_High = NA_real_,
            Boot_R2_CI_Low = NA_real_, Boot_R2_CI_High = NA_real_
        ))
    }

    # Helper to safely extract CI
    get_ci <- function(b, idx) {
        tryCatch(boot.ci(b, type="perc", index=idx)$percent[4:5], error=function(e) c(NA, NA))
    }

    boot_ci_intercept <- get_ci(boot_res, 1)
    boot_ci_slope <- get_ci(boot_res, 2)
    boot_ci_r2 <- get_ci(boot_res, 3)

    list(
        R2 = r2, Bias = bias, RMSE = rmse, Slope = coef(fit)[2], Intercept = coef(fit)[1],
        Reg_Slope_CI_Low = reg_ci[2, 1], Reg_Slope_CI_High = reg_ci[2, 2],
        Reg_Intercept_CI_Low = reg_ci[1, 1], Reg_Intercept_CI_High = reg_ci[1, 2],
        Boot_Slope_CI_Low = boot_ci_slope[1], Boot_Slope_CI_High = boot_ci_slope[2],
        Boot_Intercept_CI_Low = boot_ci_intercept[1], Boot_Intercept_CI_High = boot_ci_intercept[2],
        Boot_R2_CI_Low = boot_ci_r2[1], Boot_R2_CI_High = boot_ci_r2[2]
    )
}
```

```r
# ------------------------------------------------------------------------
# 5. Apply regression metrics and build summary table
# ------------------------------------------------------------------------
message("\n--- Generating Regression Summary ---")
reg_results <- list()
for(meas in measures){
    gt_col <- paste0(meas,"-GT-MM") # GT-MM is the ground truth standard

    # SELECT THE CORRECT METHOD LIST BASED ON MEASURE
    if(meas == "Area") {
        current_methods <- methods_area
    } else {
        current_methods <- methods_full
    }

# Loop over prediction methods only (exclude GT-MM itself)
    for(m in current_methods[!current_methods %in% c("GT-MM")]){
        pred_col <- paste0(meas,"-",m)

        if (!(gt_col %in% names(df_sorted) && pred_col %in% names(df_sorted))) {
            next
        }

         # --- LOCALIZED FILTERING FOR THIS PAIR ---
        df_pair_filtered <- localized_group_filter(df_sorted, c(gt_col, pred_col))

        gt_data <- df_pair_filtered[[gt_col]]
        pred_data <- df_pair_filtered[[pred_col]]

        N_final <- sum(!is.na(gt_data) & !is.na(pred_data))

        if (N_final < 2) {
            warning(paste("Skipping regression for", meas, m, "due to insufficient data. N =", N_final))
            metrics <- regression_metrics_full(c(NA, NA), c(NA, NA))
        } else {
            metrics <- regression_metrics_full(gt_data, pred_data)
        }
        format_ci <- function(low, high) {
            if (is.na(low) | is.na(high)) return("[NA, NA]")
            return(paste0("[", round(low, 3), ", ", round(high, 3), "]"))
        }

        reg_results[[paste(meas,m,sep="_")]] <- data.frame(
            Measure = meas, Method = m,
            N = N_final,
            R2 = round(metrics$R2, 3),
            R2_Boot_CI = format_ci(metrics$Boot_R2_CI_Low, metrics$Boot_R2_CI_High),
            Bias = round(metrics$Bias, 3),
            RMSE = round(metrics$RMSE, 3),
            Slope = round(metrics$Slope, 3),
            Slope_Reg_CI = format_ci(metrics$Reg_Slope_CI_Low, metrics$Reg_Slope_CI_High),
            Slope_Boot_CI = format_ci(metrics$Boot_Slope_CI_Low, metrics$Boot_Slope_CI_High),
            Intercept = round(metrics$Intercept, 3),
```

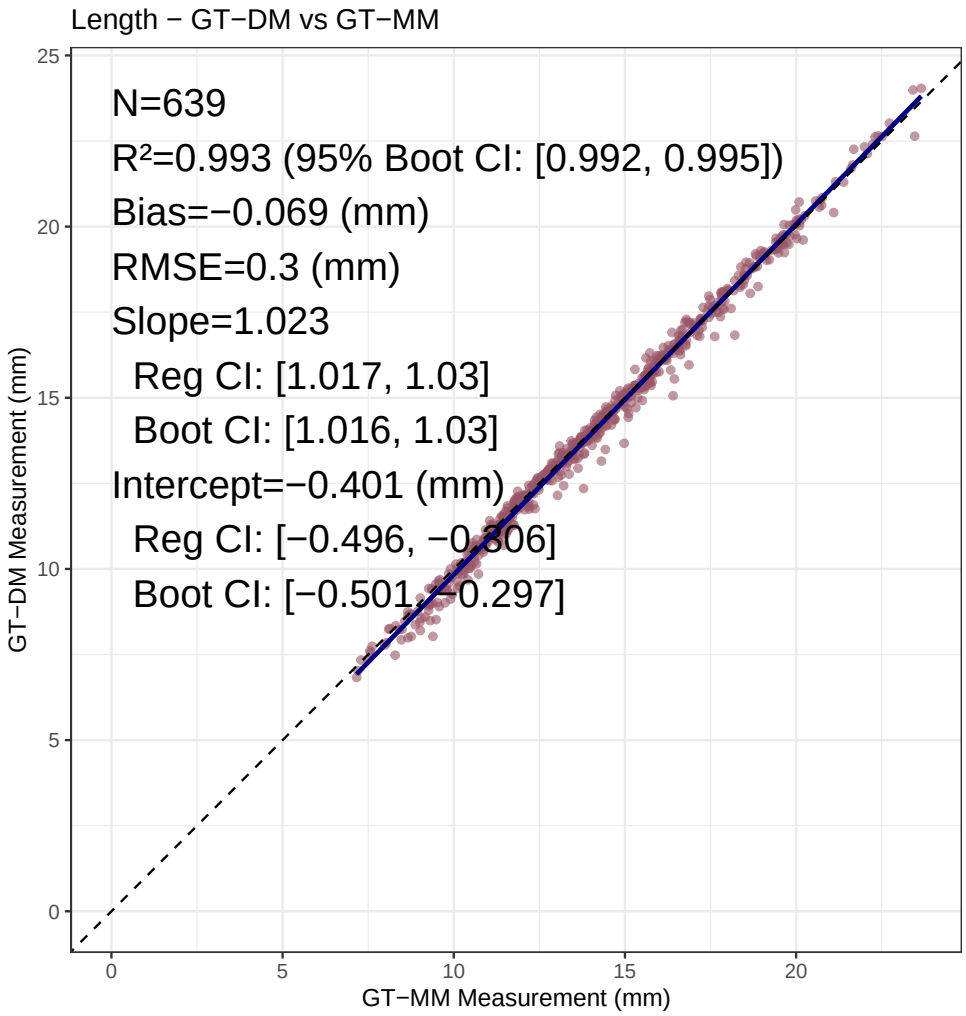
```
            Intercept_Reg_CI = format_ci(metrics$Reg_Intercept_CI_Low, metrics$Reg_Intercept_CI_High),
            Intercept_Boot_CI = format_ci(metrics$Boot_Intercept_CI_Low, metrics$Boot_Intercept_CI_High)
        )
    }
}
reg_summary_full <- do.call(rbind, reg_results)
rownames(reg_summary_full) <- NULL

# ------------------------------------------------------------------
# 6. Scatter plots (UPDATED LABELS: Uses GT-MM)
# ------------------------------------------------------------------
plot_list <- list()
for(meas in measures){
    gt_col <- paste0(meas,"-GT-MM")
    unit <- unit_labels[meas]

    # SELECT THE CORRECT METHOD LIST BASED ON MEASURE
    if(meas == "Area") {
        current_methods <- methods_area
    } else {
        current_methods <- methods_full
    }

    for(m in current_methods[!current_methods %in% c("GT-MM")]){
        pred_col <- paste0(meas,"-",m)

        if (!(gt_col %in% names(df_sorted) && pred_col %in% names(df_sorted))) next

        # --- LOCALIZED FILTERING FOR THIS PAIR ---
        df_plot_filtered <- localized_group_filter(df_sorted, c(gt_col, pred_col))

        tmp <- df_plot_filtered %>% select(Code, all_of(gt_col), all_of(pred_col)) %>%
            rename(GT = all_of(gt_col), Pred = all_of(pred_col)) %>% drop_na()

        if (nrow(tmp) == 0) next

        metrics <- reg_summary_full %>% filter(Measure==meas, Method==m)
        annotation_text <- paste0(
            "N=", metrics$N, "\n",
            "R\u00B2=", metrics$R2, " (95% Boot CI: ", metrics$R2_Boot_CI, ")\n",
            "Bias=", metrics$Bias, unit, "\n",
            "RMSE=", metrics$RMSE, unit, "\n",
            "Slope=", metrics$Slope, "\n",
            "  Reg CI: ", metrics$Slope_Reg_CI, "\n",
            "  Boot CI: ", metrics$Slope_Boot_CI, "\n",
            "Intercept=", metrics$Intercept, unit, "\n",
            "  Reg CI: ", metrics$Intercept_Reg_CI, "\n",
            "  Boot CI: ", metrics$Intercept_Boot_CI
        )

        p <- ggplot(tmp, aes(x=GT, y=Pred)) +
            geom_point(color=palette[m], alpha=0.6, size=1.5) +
            geom_smooth(method="lm", se=TRUE, color="navy", fill=alpha("navy",0.2)) +
```

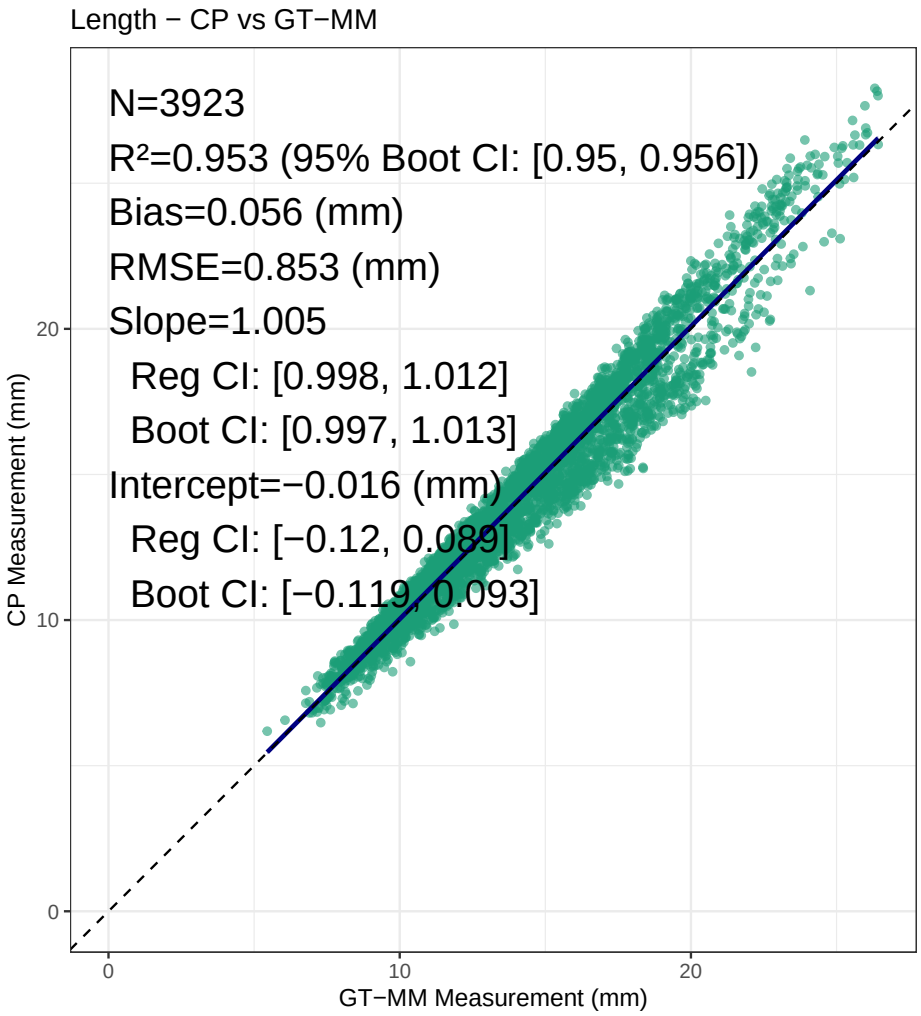
```
            geom_abline(slope=1, intercept=0, linetype="dashed") +
            annotate("text",
                     x = 0,
                     y = max(tmp$Pred, na.rm=TRUE),
                     label=annotation_text, hjust=0, vjust=1, size=6.525) +
            labs(title=paste0(meas," - ",m," vs GT-MM"),
                x=paste0("GT-MM Measurement", unit),
                y=paste0(m," Measurement", unit)) +
            theme_bw(base_size = 12) +
            coord_fixed(ratio=1) +
            expand_limits(x=0, y=0)
        plot_list[[paste0("Scatter_", meas,"_",m)]] <- p
    }
}

# ------------------------------------------------------------------
# 7. Boxplots
# ------------------------------------------------------------------
create_boxplot_v3 <- function(df_long, y_var, y_label, all_methods_in_order, original_df_clean) {

    # 1. Filter to only include methods present in the data/plot list
    present_methods <- intersect(unique(df_long$Method), all_methods_in_order)

    boxplot_data_list <- list()
    stats_list <- list()

    for(m in present_methods) {
        col_name <- paste0(y_var, "-", m)
        if (!col_name %in% names(original_df_clean)) next

        # --- LOCALIZED FILTERING ---
        df_filtered_local <- localized_group_filter(original_df_clean, c(col_name))

        data_for_method <- df_filtered_local %>%
            select(Method = "Code", Value = all_of(col_name)) %>%
            mutate(Method = m) %>%
            drop_na(Value)
        if (nrow(data_for_method) == 0) next

        stats_tmp <- data_for_method %>%
            summarise(
                median_val = median(Value, na.rm = TRUE),
                n = sum(!is.na(Value)),
                upper_whisker = tryCatch(boxplot.stats(Value)$stats[5], error = function(e) max(Value,
                .groups = 'drop'
            ) %>%
            mutate(Method = m)

        boxplot_data_list[[m]] <- data_for_method
        stats_list[[m]] <- stats_tmp
    }

    if (length(boxplot_data_list) == 0) return(NULL)
```

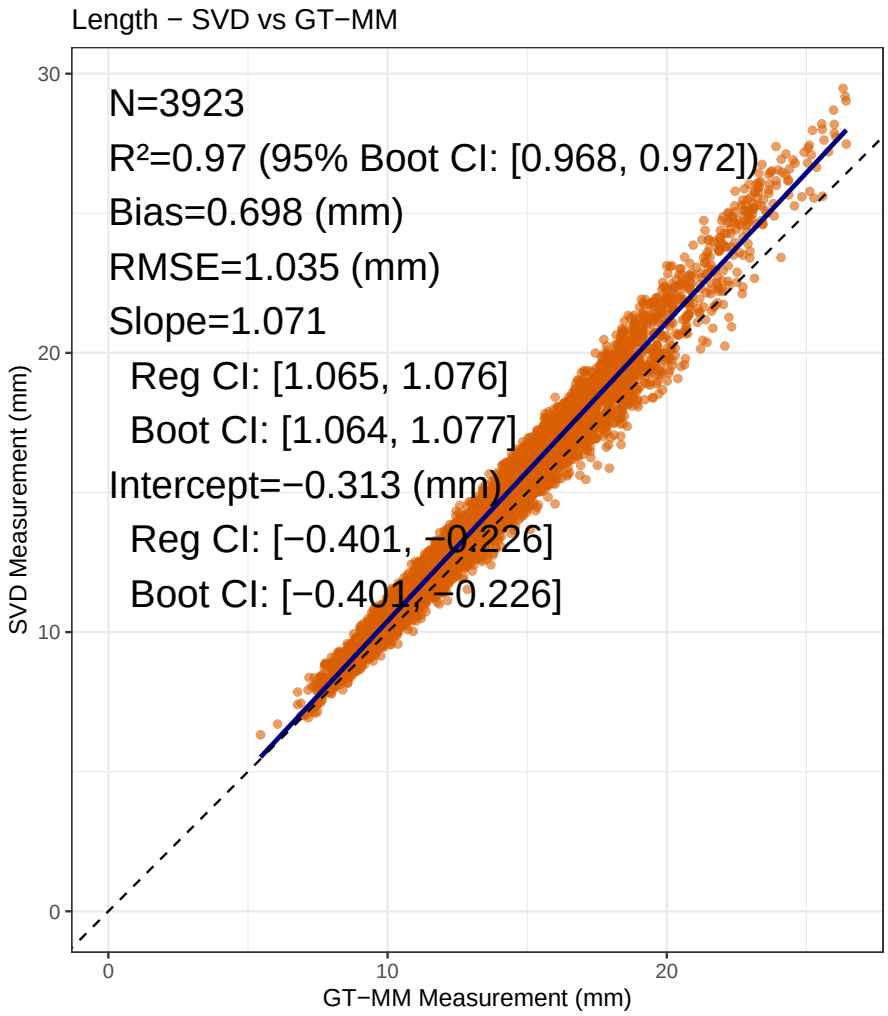
```
    df_long_filtered <- do.call(bind_rows, boxplot_data_list)
    stats <- do.call(bind_rows, stats_list)

    df_long_filtered$Method <- factor(df_long_filtered$Method, levels = present_methods)
    stats$Method <- factor(stats$Method, levels = present_methods)

    min_y_global <- min(df_long_filtered$Value, na.rm = TRUE)
    max_y_global <- max(stats$upper_whisker, na.rm = TRUE)

    stats <- stats %>%
        mutate(
            text_y_pos = upper_whisker + 0.05 * (max_y_global - min_y_global)
        )

    present_palette <- palette[present_methods]
    present_palette <- present_palette[!is.na(names(present_palette))] # Remove NA if key missing

    p <- ggplot(df_long_filtered, aes(x = Method, y = Value, fill = Method)) +
        geom_boxplot(outlier.shape = 21, alpha = 0.7) +
        scale_fill_manual(values = present_palette) +
        theme_minimal(base_size = 16) +
        labs(title = paste0(y_label, " Comparison Across Methods"),
             x = "Measurement Method", y = y_label) +
        theme(legend.position = "none",
              axis.text.x = element_text(angle = 45, hjust = 1)) +
        geom_text(
            data = stats,
            aes(x = Method,
                y = text_y_pos,
                label = paste0("Median: ", round(median_val, 2), "\nN: ", n)),
            vjust = 0,
            size = 3.5,
            fontface = "bold",
            color = "darkgrey"
        )
    return(p)
}

# Construct Long DFs (Filtering out non-existent columns safely)
length_cols <- intersect(names(df_clean), paste0("Length-", methods_full))
width_cols <- intersect(names(df_clean), paste0("Width-", methods_full))
area_cols <- intersect(names(df_clean), paste0("Area-", methods_area))

length_df_long_base <- df_clean %>% select(Code, Group2, all_of(length_cols)) %>%
    pivot_longer(cols = starts_with("Length-"), names_to="Method_Col", values_to="Value") %>%
    mutate(Method = sub("Length-", "", Method_Col))

width_df_long_base <- df_clean %>% select(Code, Group2, all_of(width_cols)) %>%
    pivot_longer(cols = starts_with("Width-"), names_to="Method_Col", values_to="Value") %>%
    mutate(Method = sub("Width-", "", Method_Col))

area_df_long_base <- df_clean %>% select(Code, Group2, all_of(area_cols)) %>%
    pivot_longer(cols = starts_with("Area-"), names_to="Method_Col", values_to="Value") %>%
```

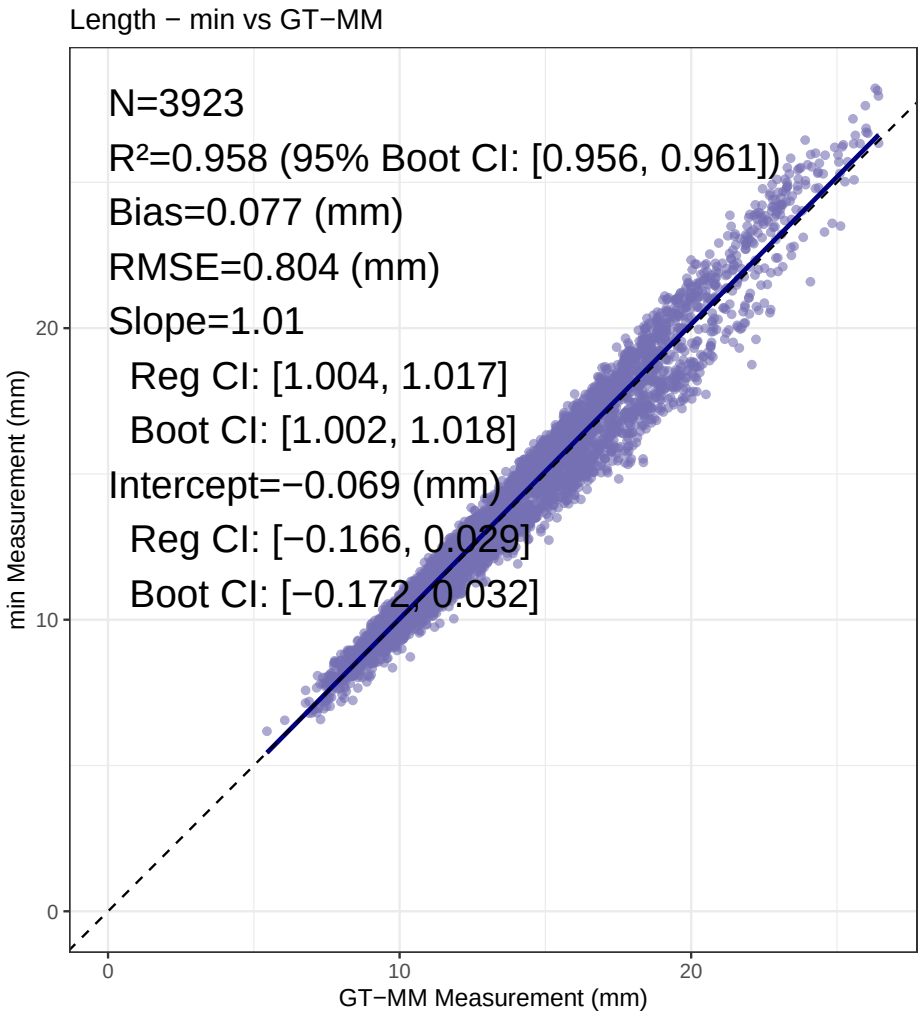
```
    mutate(Method = sub("Area-", "", Method_Col))

# Create boxplots
p_length_v3 <- create_boxplot_v3(length_df_long_base, "Length", paste0("Length", unit_labels["Length"])
p_width_v3 <- create_boxplot_v3(width_df_long_base, "Width", paste0("Width", unit_labels["Width"]), meth
p_area_v3 <- create_boxplot_v3(area_df_long_base, "Area", paste0("Area", unit_labels["Area"]), methods_a

boxplot_list <- list()
if(!is.null(p_length_v3)) boxplot_list[["Boxplot_Length"]] <- p_length_v3
if(!is.null(p_width_v3)) boxplot_list[["Boxplot_Width"]] <- p_width_v3
if(!is.null(p_area_v3)) boxplot_list[["Boxplot_Area"]] <- p_area_v3


# ---------------------------------------------------------------------
# 8. Bland-Altman Plots (UPDATED LABELS: Uses GT-MM and df_sorted)
# ---------------------------------------------------------------------
altman_list <- list()
for(meas in measures){
    gt_col <- paste0(meas,"-GT-MM")
    unit <- unit_labels[meas]
    # SELECT THE CORRECT METHOD LIST BASED ON MEASURE
    if(meas == "Area") {
        current_methods <- methods_area
    } else {
        current_methods <- methods_full
    }

    for(m in current_methods[!current_methods %in% c("GT-MM")]){
        pred_col <- paste0(meas,"-",m)

        # Check against df_sorted (to use aligned seeds)
        if (!(gt_col %in% names(df_sorted) && pred_col %in% names(df_sorted))) next

        # --- LOCALIZED FILTERING (Applied to SORTED data) ---
        df_altman_filtered <- localized_group_filter(df_sorted, c(gt_col, pred_col))

        tmp <- df_altman_filtered %>% select(all_of(gt_col), all_of(pred_col)) %>%
            rename(GT = all_of(gt_col), Pred = all_of(pred_col)) %>% drop_na()

        if (nrow(tmp) < 2) next

        tmp <- tmp %>% mutate(
            Mean = (GT + Pred)/2,
            Diff = Pred - GT
        )

        mean_diff <- mean(tmp$Diff)
        sd_diff <- sd(tmp$Diff)
        loa_upper <- mean_diff + 1.96*sd_diff
        loa_lower <- mean_diff - 1.96*sd_diff

        max_mean <- max(tmp$Mean, na.rm=TRUE)
        label_x_pos <- max_mean * 0.8
```

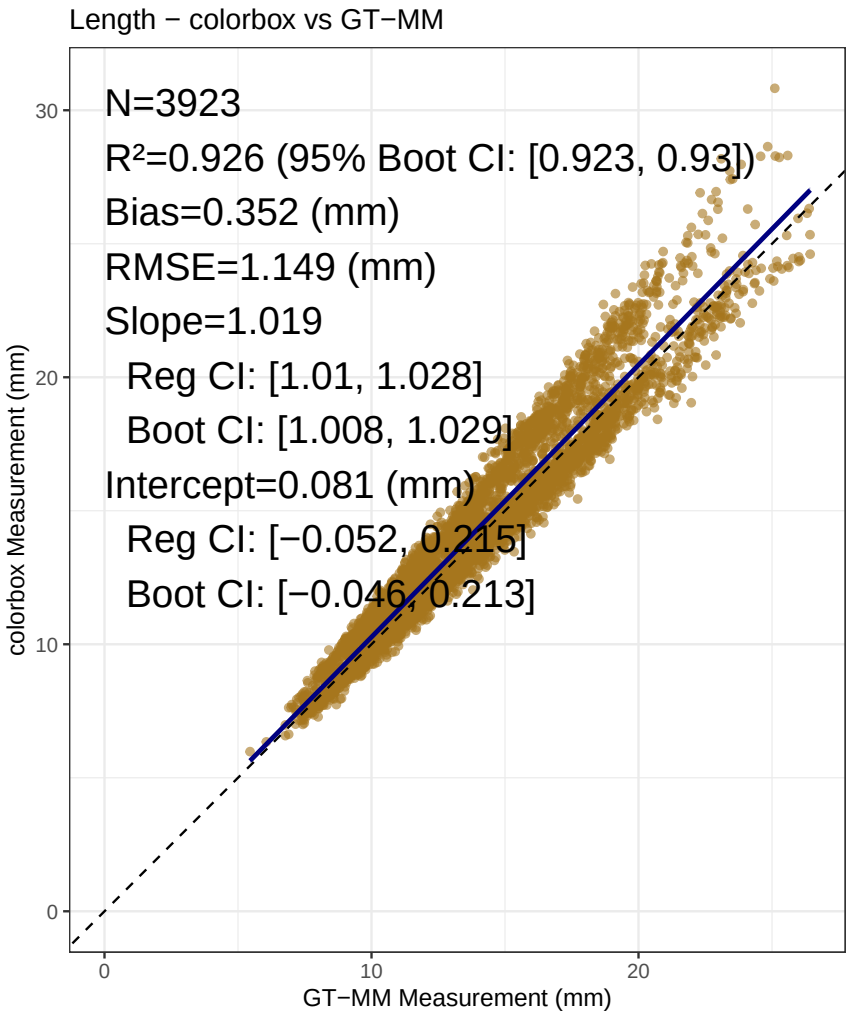
```
        p <- ggplot(tmp, aes(x=Mean, y=Diff)) +
            geom_point(color=palette[m], alpha=0.6, size=1.5) +
            geom_hline(yintercept = mean_diff, color="blue", linetype="solid", size=1) +
            geom_hline(yintercept = loa_upper, color="red", linetype="dashed", size=1) +
            geom_hline(yintercept = loa_lower, color="red", linetype="dashed", size=1) +

            annotate("text", x=label_x_pos, y=mean_diff,
                    label=paste("Mean Diff:", round(mean_diff, 3), unit, " (N=", nrow(tmp), ")"),
                    color="blue", size=6.2, vjust=-0.5) +
            annotate("text", x=label_x_pos, y=loa_upper,
                    label=paste("+1.96 SD:", round(loa_upper, 3), unit),
                    color="red", size=6.2, vjust=-0.5) +
            annotate("text", x=label_x_pos, y=loa_lower,
                    label=paste("-1.96 SD:", round(loa_lower, 3), unit),
                    color="red", size=6.2, vjust=1.5) +
            labs(title=paste0(meas," -- ", m, " VS GT-MM"),
                 x=paste0("Mean of GT-MM and Pred", unit),
                 y=paste0("Difference (Pred - GT-MM)", unit)) +
            theme_bw(base_size = 14)

        altman_list[[paste0("Altman_", meas,"_",m)]] <- p
    }
}
# ------------------------------------------------------------------
# 9. Save Plots and Summary
# ------------------------------------------------------------------

if (!dir.exists(output_dir)) {
    dir.create(output_dir, recursive = TRUE)
    message(paste("\nCreated output directory:", output_dir))
} else {
    message(paste("\nOutput directory already exists:", output_dir))
}

save_plots <- function(plot_list, folder) {
      if (is.null(plot_list) || length(plot_list) == 0) {
        message("Plot list is empty, skipping save operation.")
        return(NULL)
    }

    for (plot_name in names(plot_list)) {
        filename <- file.path(folder, paste0(plot_name, ".png"))

        ggsave(filename, plot_list[[plot_name]], width=10, height=8, units="in", dpi=300)

        invisible(print(plot_list[[plot_name]]))

        message(paste("Saved plot:", filename))
    }
}

message("\n--- Starting Plot Saving ---")
save_plots(plot_list, output_dir)
```

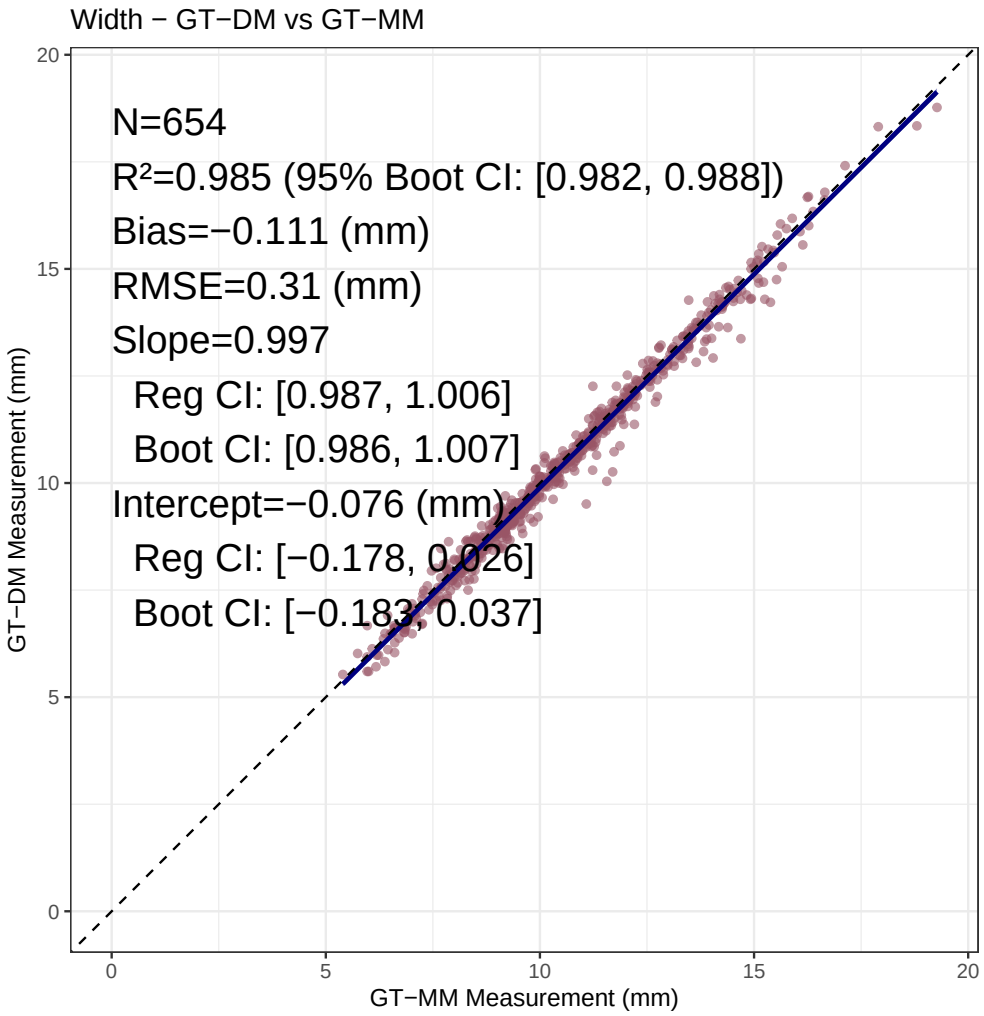

Length − GT−DM vs GT−MM
N=639
R²=0.993 (95% Boot CI: [0.992, 0.995])
Bias=−0.069 (mm)
RMSE=0.3 (mm)
Slope=1.023
Reg CI: [1.017, 1.03]
Boot CI: [1.016, 1.03]
Intercept=−0.401 (mm)
Reg CI: [−0.496, −0.306]
Boot CI: [−0.501, −0.297]
GT−DM Measurement (mm)
GT−MM Measurement (mm)
0
5
10
15
20
25

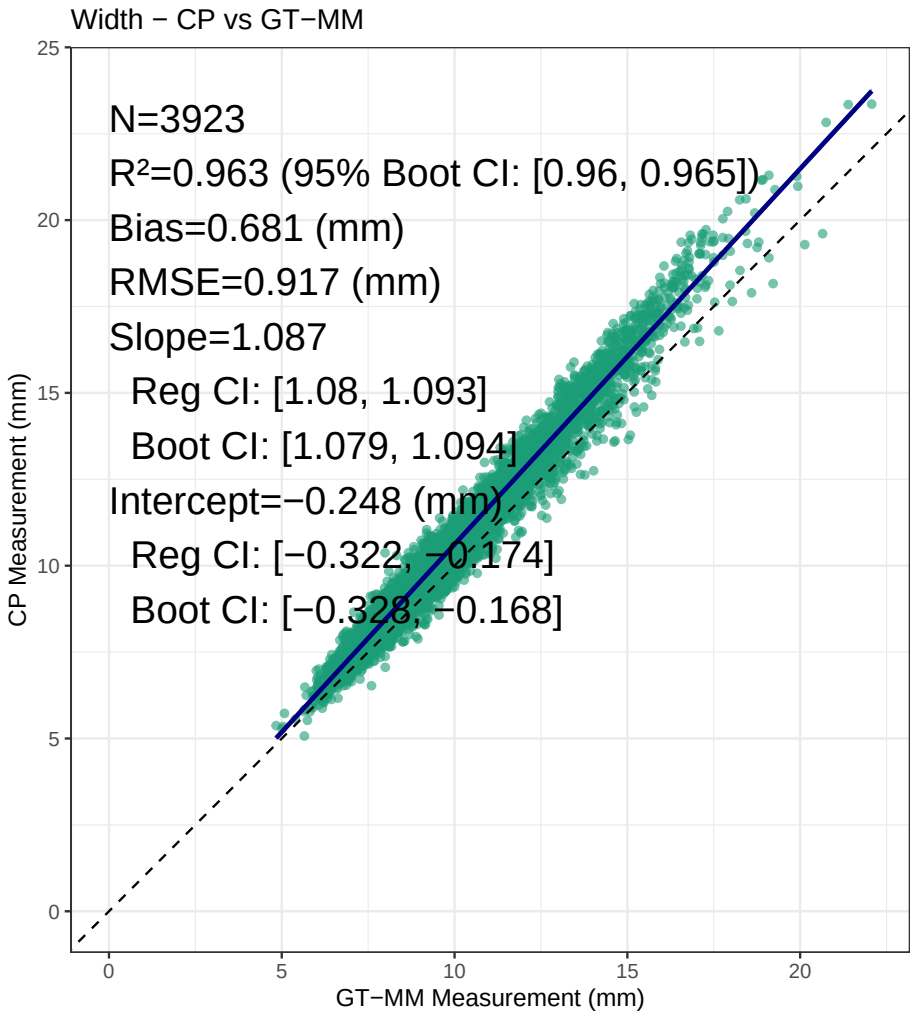

Length − CP vs GT−MM
N=3923
R²=0.953 (95% Boot CI: [0.95, 0.956])
Bias=0.056 (mm)
RMSE=0.853 (mm)
Slope=1.005
Reg CI: [0.998, 1.012]
Boot CI: [0.997, 1.013]
Intercept=−0.016 (mm)
Reg CI: [−0.12, 0.089]
Boot CI: [−0.119, 0.093]
CP Measurement (mm)
GT−MM Measurement (mm)
0
10
20

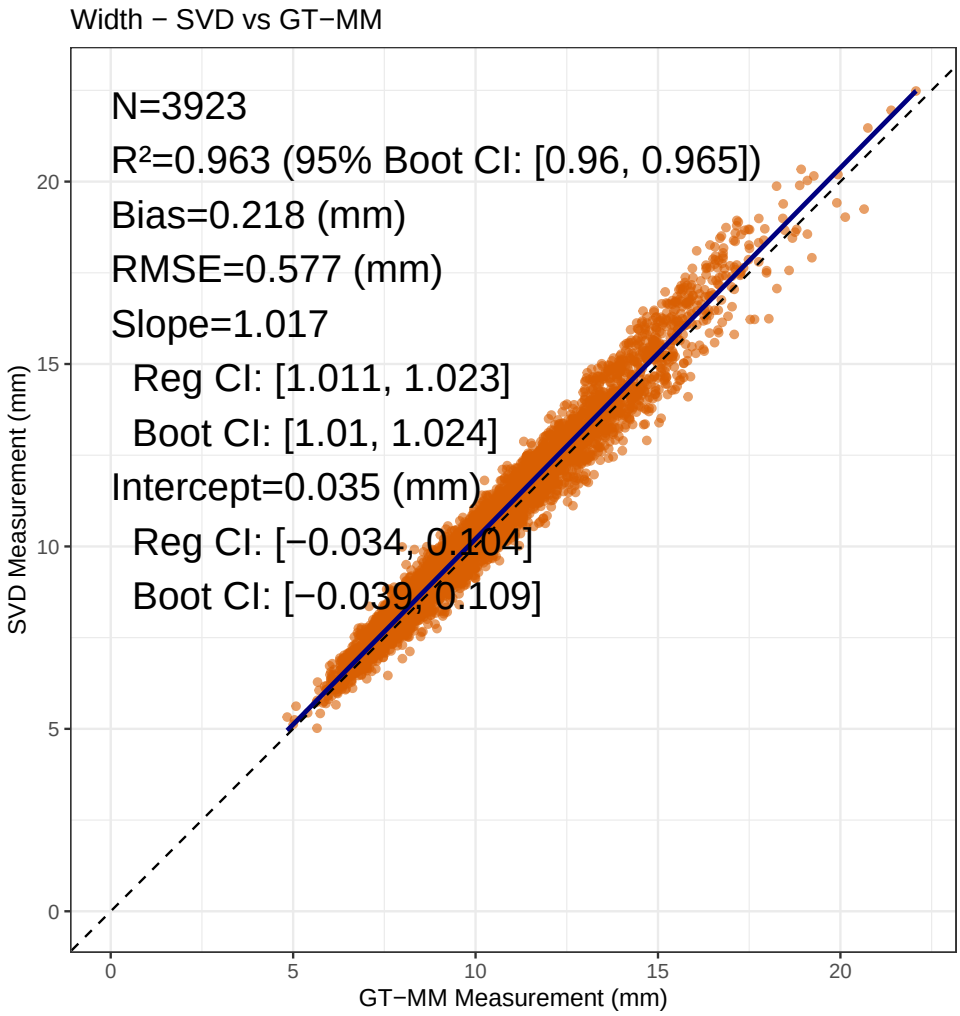

Length − SVD vs GT−MM
N=3923
R²=0.97 (95% Boot CI: [0.968, 0.972])
Bias=0.698 (mm)
RMSE=1.035 (mm)
Slope=1.071
Reg CI: [1.065, 1.076]
Boot CI: [1.064, 1.077]
Intercept=−0.313 (mm)
Reg CI: [−0.401, −0.226]
Boot CI: [−0.401, −0.226]
SVD Measurement (mm)
GT−MM Measurement (mm)
0
10
20
30

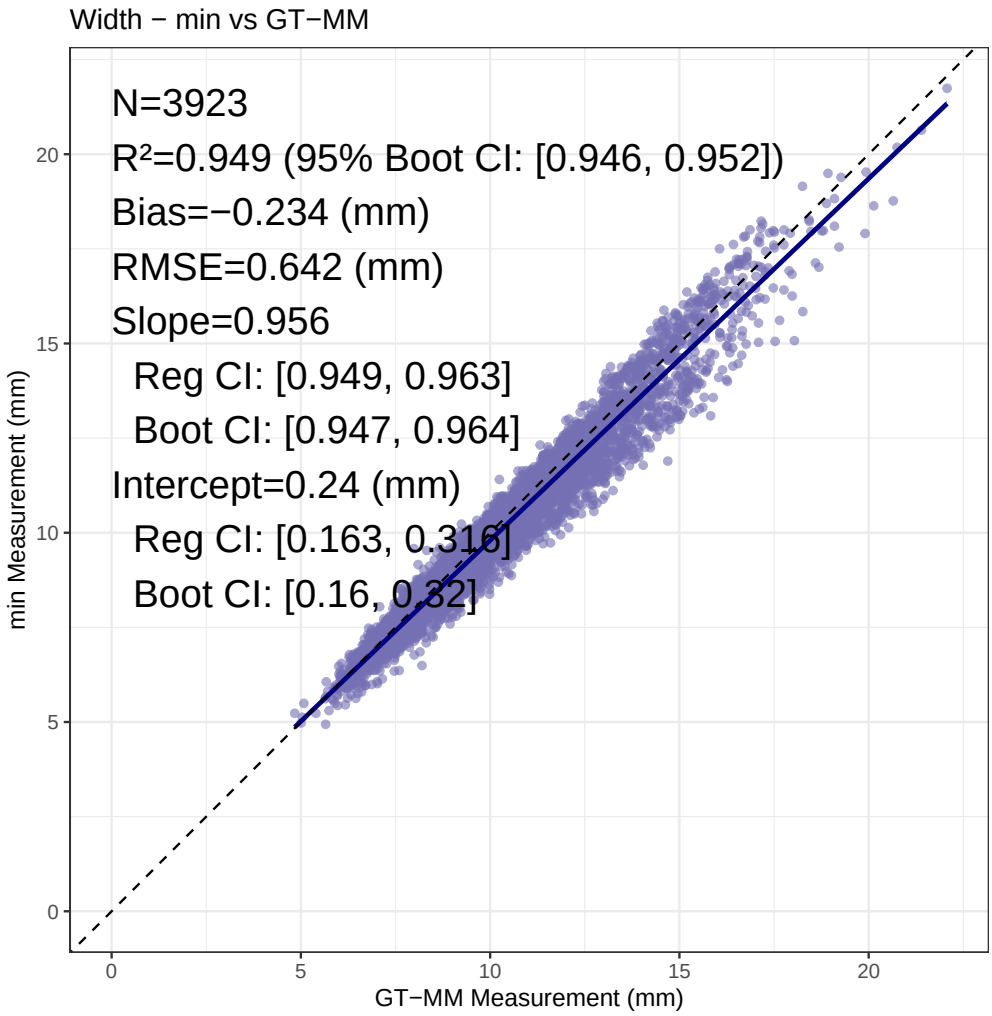

Length − min vs GT−MM
N=3923
R²=0.958 (95% Boot CI: [0.956, 0.961])
Bias=0.077 (mm)
RMSE=0.804 (mm)
Slope=1.01
Reg CI: [1.004, 1.017]
Boot CI: [1.002, 1.018]
Intercept=−0.069 (mm)
Reg CI: [−0.166, 0.029]
Boot CI: [−0.172, 0.032]
min Measurement (mm)
GT−MM Measurement (mm)
0
10
20

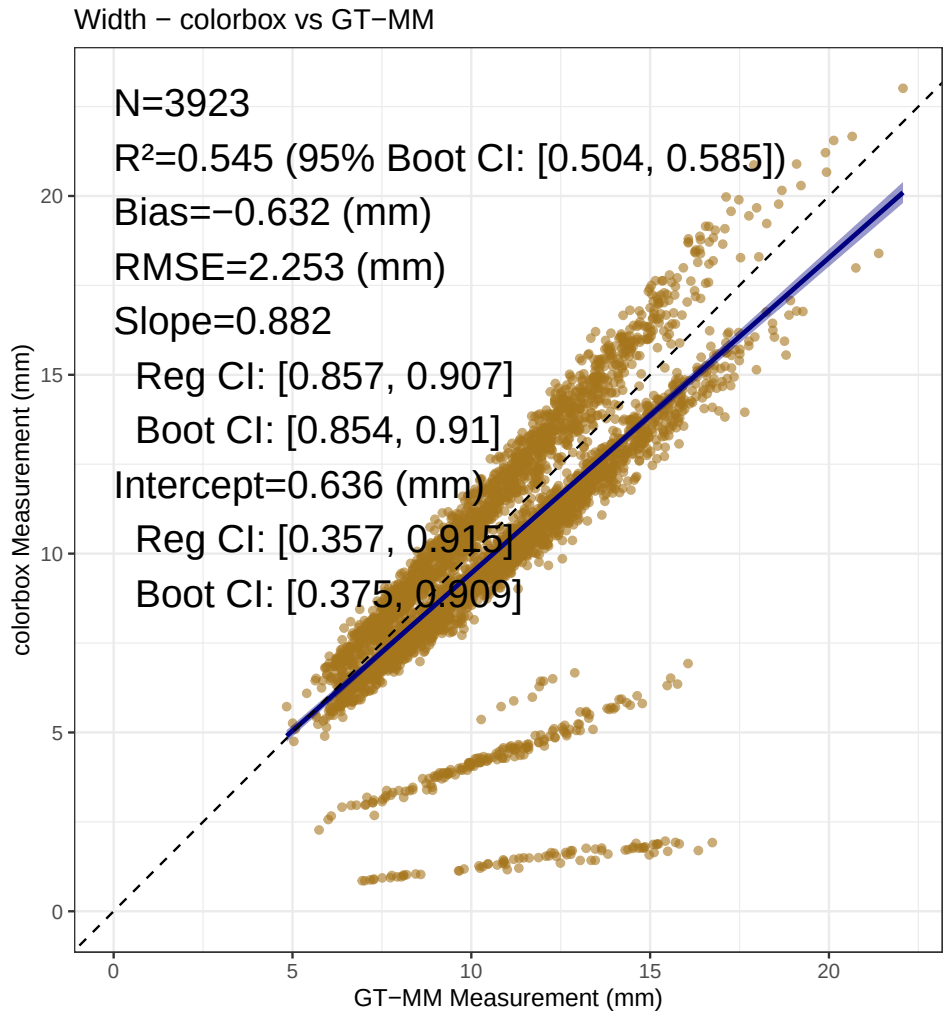

Length − colorbox vs GT−MM
N=3923
R²=0.926 (95% Boot CI: [0.923, 0.93])
Bias=0.352 (mm)
RMSE=1.149 (mm)
Slope=1.019
Reg CI: [1.01, 1.028]
Boot CI: [1.008, 1.029]
Intercept=0.081 (mm)
Reg CI: [−0.052, 0.215]
Boot CI: [−0.046, 0.213]
colorbox Measurement (mm)
GT−MM Measurement (mm)
0
10
20
30

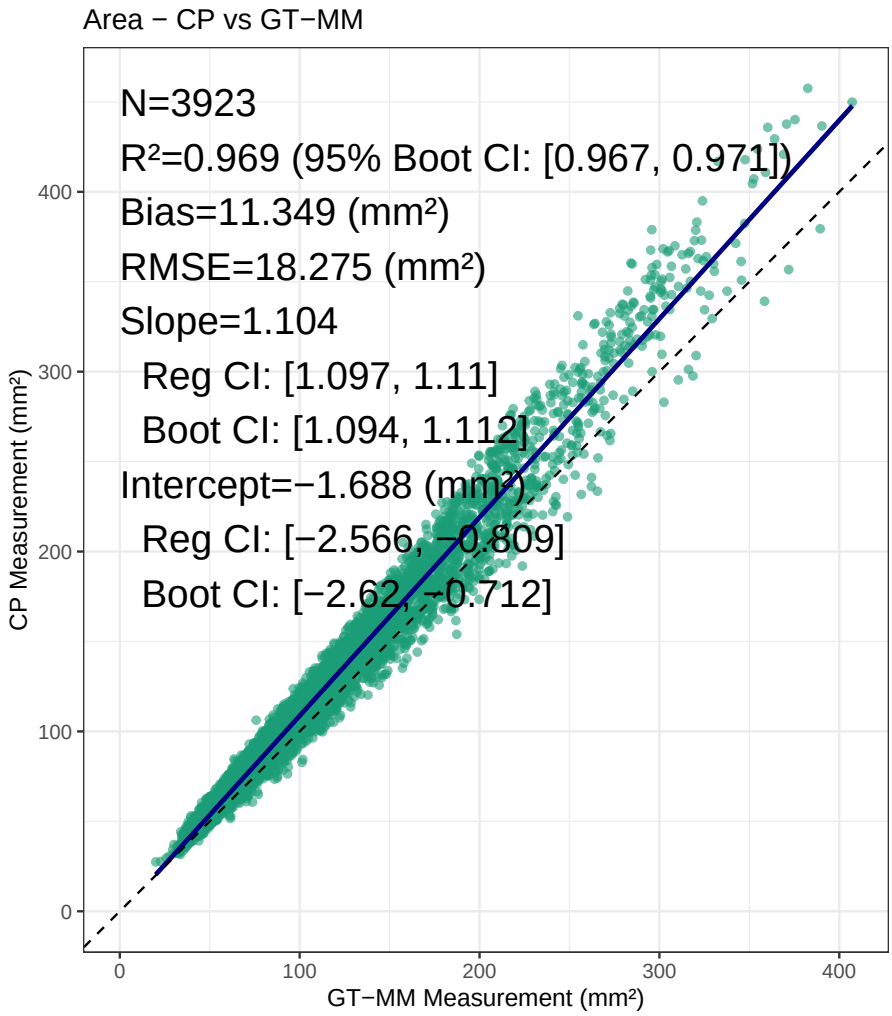

Width − GT−DM vs GT−MM
N=654
R²=0.985 (95% Boot CI: [0.982, 0.988])
Bias=−0.111 (mm)
RMSE=0.31 (mm)
Slope=0.997
Reg CI: [0.987, 1.006]
Boot CI: [0.986, 1.007]
Intercept=−0.076 (mm)
Reg CI: [−0.178, 0.026]
Boot CI: [−0.183, 0.037]
GT−DM Measurement (mm)
GT−MM Measurement (mm)
0
5
10
15
20

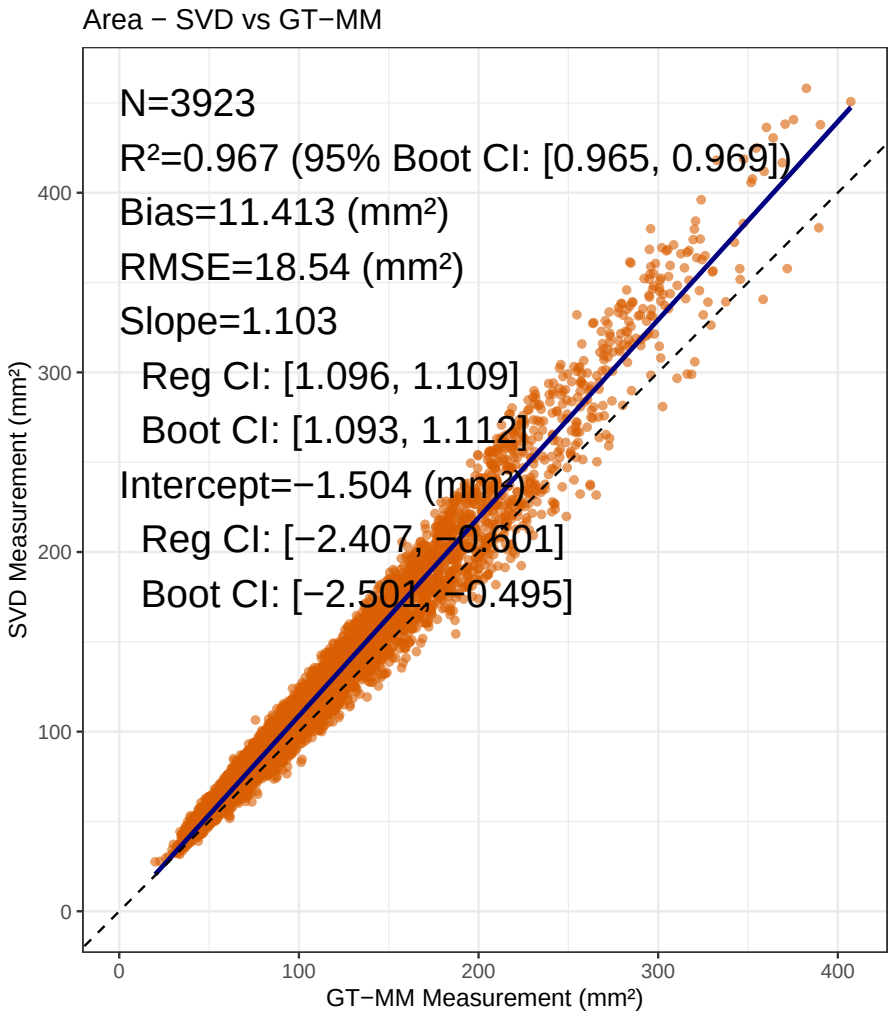

Width − CP vs GT−MM
N=3923
R²=0.963 (95% Boot CI: [0.96, 0.965])
Bias=0.681 (mm)
RMSE=0.917 (mm)
Slope=1.087
Reg CI: [1.08, 1.093]
Boot CI: [1.079, 1.094]
Intercept=−0.248 (mm)
Reg CI: [−0.322, −0.174]
Boot CI: [−0.328, −0.168]
CP Measurement (mm)
GT−MM Measurement (mm)
0
5
10
15
20
25

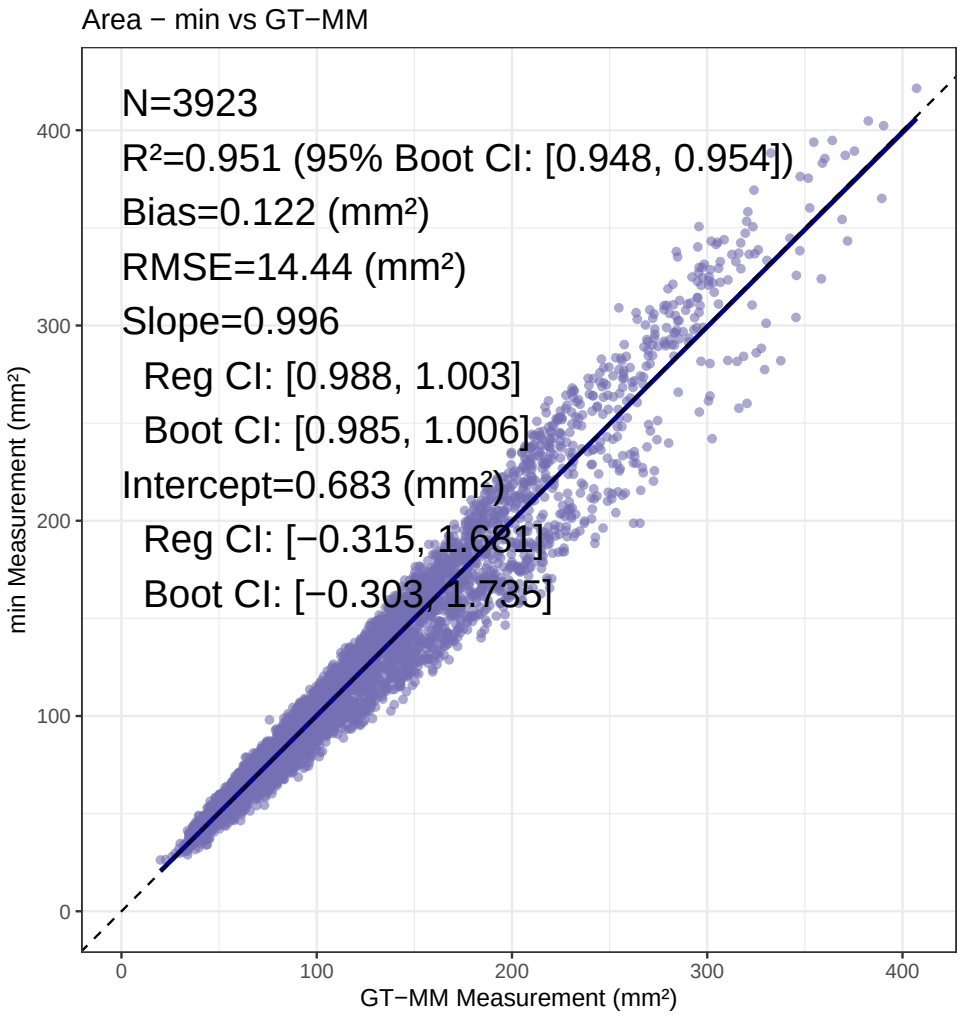

Width − SVD vs GT−MM
N=3923
R²=0.963 (95% Boot CI: [0.96, 0.965])
Bias=0.218 (mm)
RMSE=0.577 (mm)
Slope=1.017
Reg CI: [1.011, 1.023]
Boot CI: [1.01, 1.024]
Intercept=0.035 (mm)
Reg CI: [−0.034, 0.104]
Boot CI: [−0.039, 0.109]
SVD Measurement (mm)
GT−MM Measurement (mm)
0
5
10
15
20


Width − min vs GT−MM
N=3923
R²=0.949 (95% Boot CI: [0.946, 0.952])
Bias=−0.234 (mm)
RMSE=0.642 (mm)
Slope=0.956
Reg CI: [0.949, 0.963]
Boot CI: [0.947, 0.964]
Intercept=0.24 (mm)
Reg CI: [0.163, 0.316]
Boot CI: [0.16, 0.32]
min Measurement (mm)
GT−MM Measurement (mm)
0
5
10
15
20

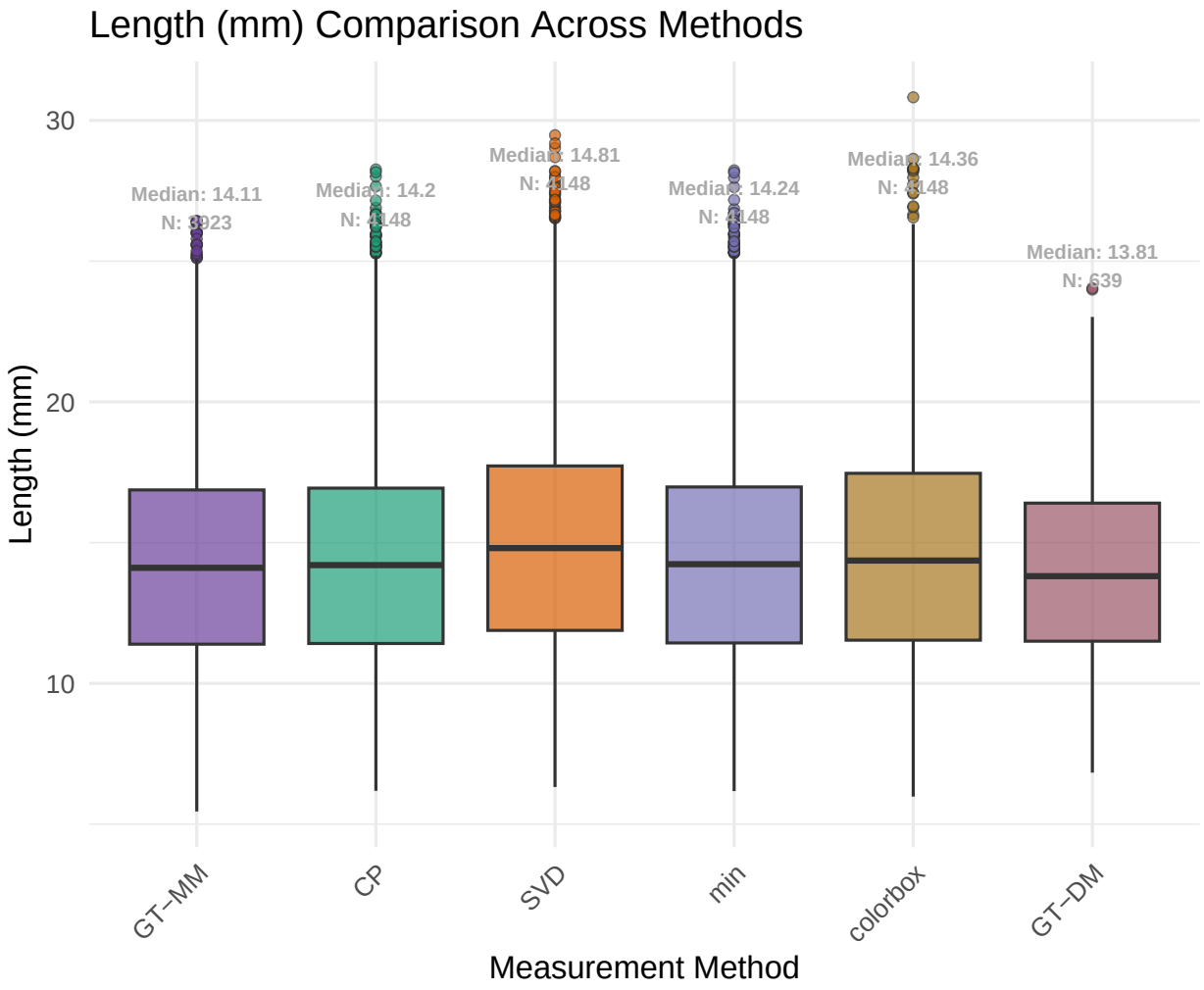

Width − colorbox vs GT−MM
N=3923
R²=0.545 (95% Boot CI: [0.504, 0.585])
Bias=−0.632 (mm)
RMSE=2.253 (mm)
Slope=0.882
Reg CI: [0.857, 0.907]
Boot CI: [0.854, 0.91]
Intercept=0.636 (mm)
Reg CI: [0.357, 0.915]
Boot CI: [0.375, 0.909]
colorbox Measurement (mm)
GT−MM Measurement (mm)
0
5
10
15
20

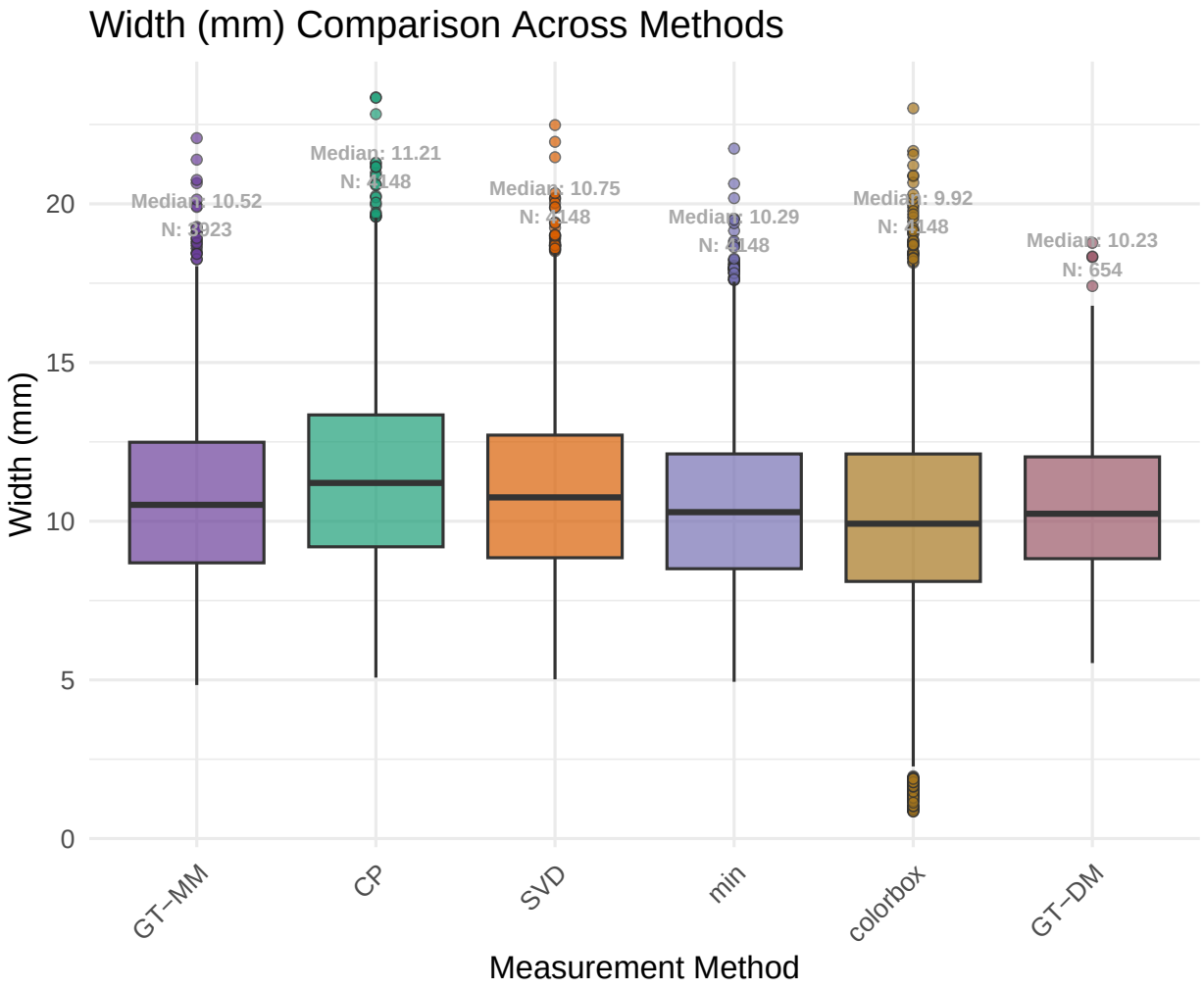

Area − CP vs GT−MM
N=3923
R²=0.969 (95% Boot CI: [0.967, 0.971])
Bias=11.349 (mm²)
RMSE=18.275 (mm²)
Slope=1.104
Reg CI: [1.097, 1.11]
Boot CI: [1.094, 1.112]
Intercept=−1.688 (mm²)
Reg CI: [−2.566, −0.809]
Boot CI: [−2.62, −0.712]
CP Measurement (mm²)
GT−MM Measurement (mm²)
0
100
200
300
400

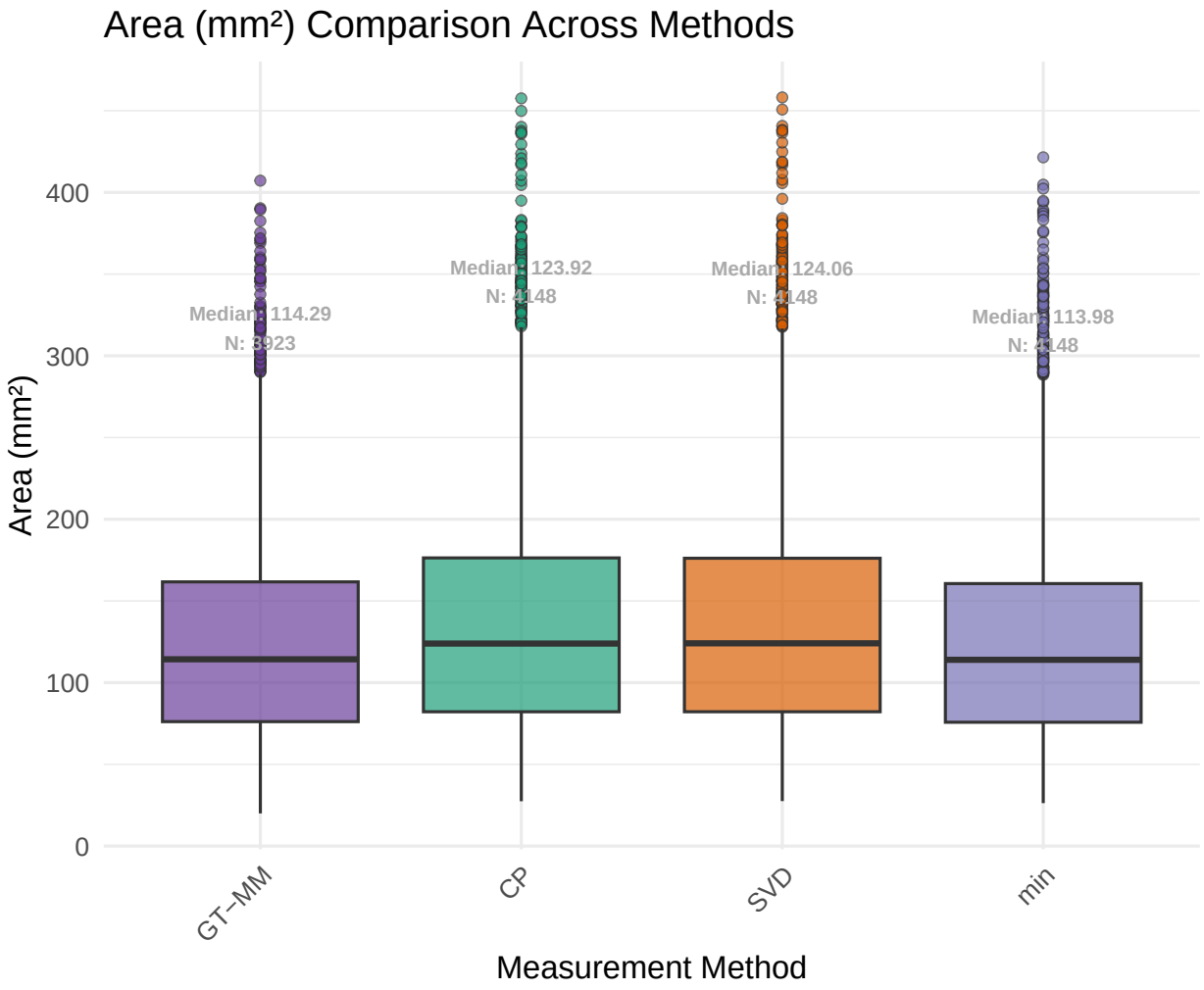

Area − SVD vs GT−MM
N=3923
R²=0.967 (95% Boot CI: [0.965, 0.969])
Bias=11.413 (mm²)
RMSE=18.54 (mm²)
Slope=1.103
Reg CI: [1.096, 1.109]
Boot CI: [1.093, 1.112]
Intercept=−1.504 (mm²)
Reg CI: [−2.407, −0.601]
Boot CI: [−2.501, −0.495]
SVD Measurement (mm²)
GT−MM Measurement (mm²)
0
100
200
300
400



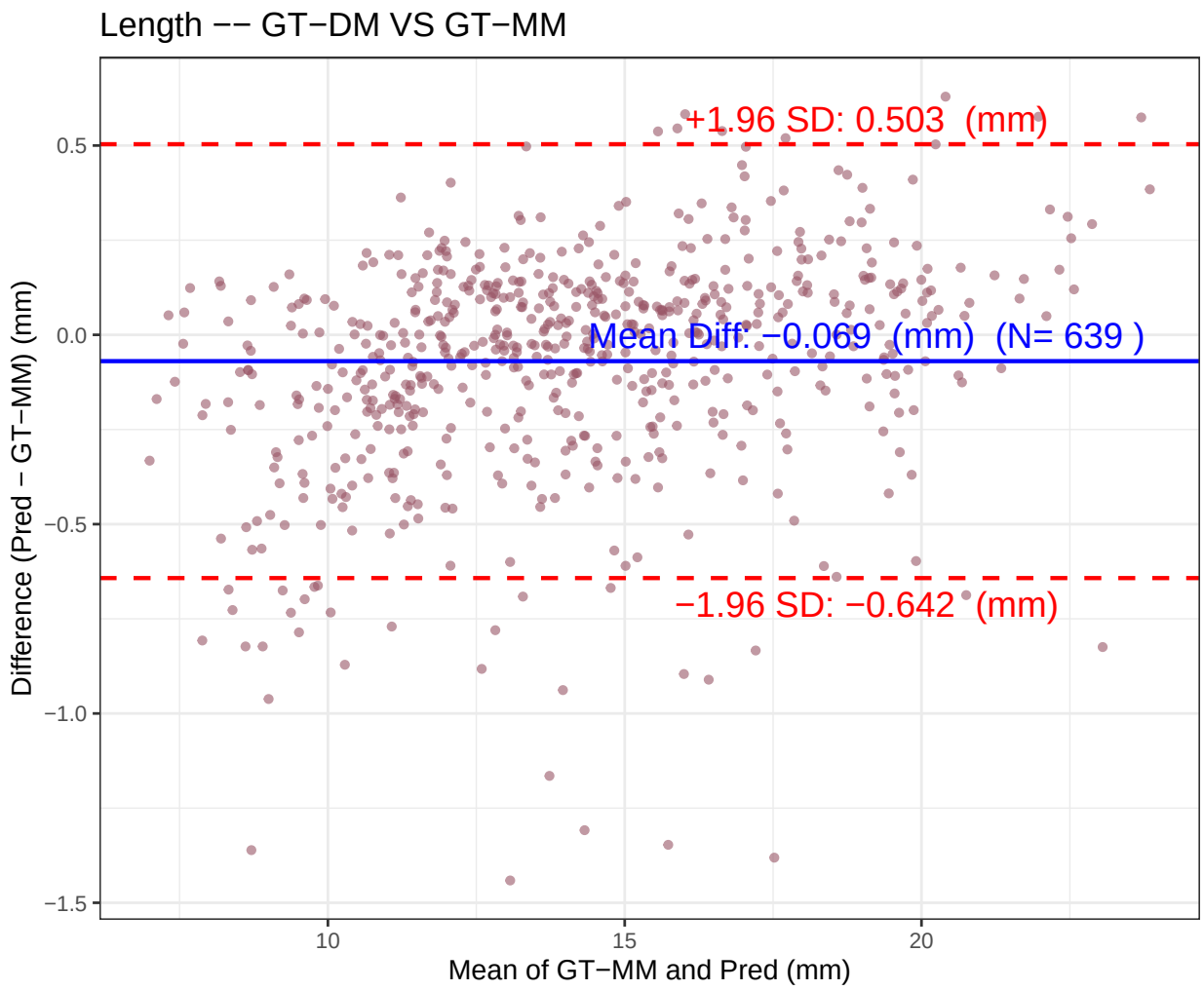

```
save_plots(boxplot_list, output_dir)
```

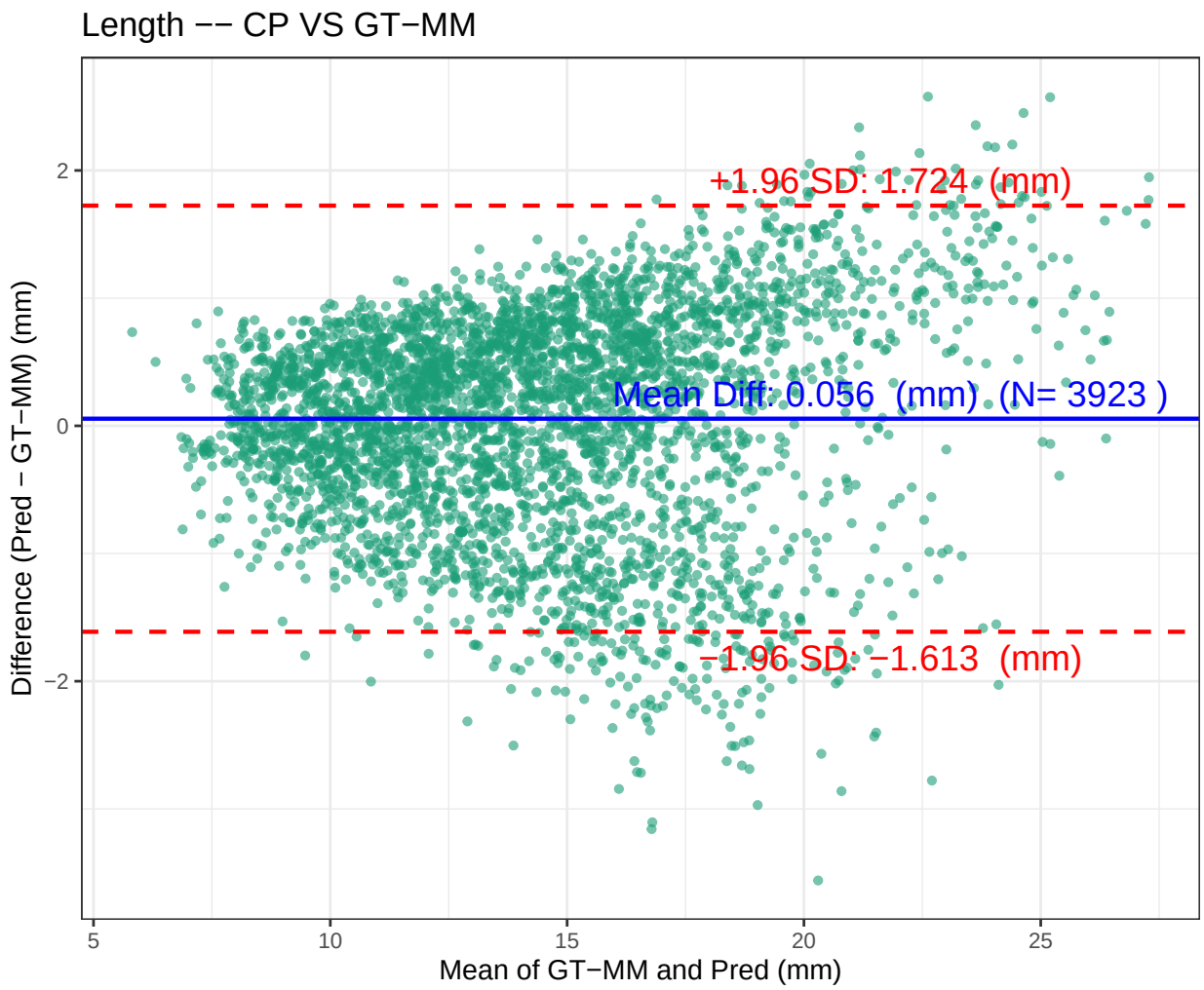

Length (mm) Comparison Across Methods
30
20
10
Length (mm)
Median: 14.11
N: 3923
Median: 14.2
N: 4148
Median: 14.81
N: 4148
Median: 14.24
N: 4148
Median: 14.36
N: 4148
Median: 13.81
N: 639
GT−MM
CP
SVD
min
colorbox
GT−DM
Measurement Method

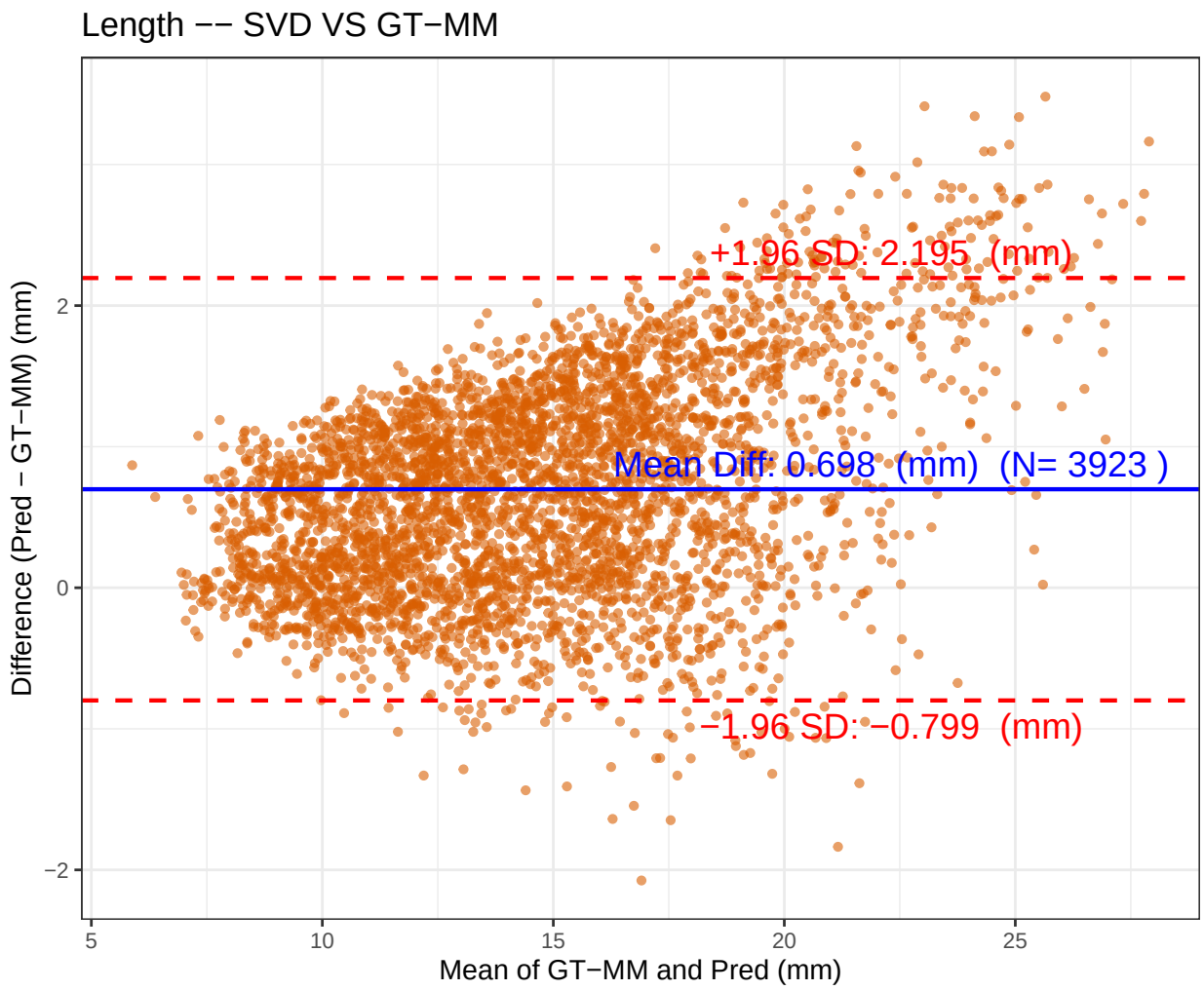

Width (mm) Comparison Across Methods
Median: 10.52
N: 3923
Median: 11.21
N: 4148
Median: 10.75
N: 4148
Median: 10.29
N: 4148
Median: 9.92
N: 4148
Median: 10.23
N: 654
Width (mm)
0
5
10
15
20
GT−MM
CP
SVD
min
colorbox
GT−DM
Measurement Method

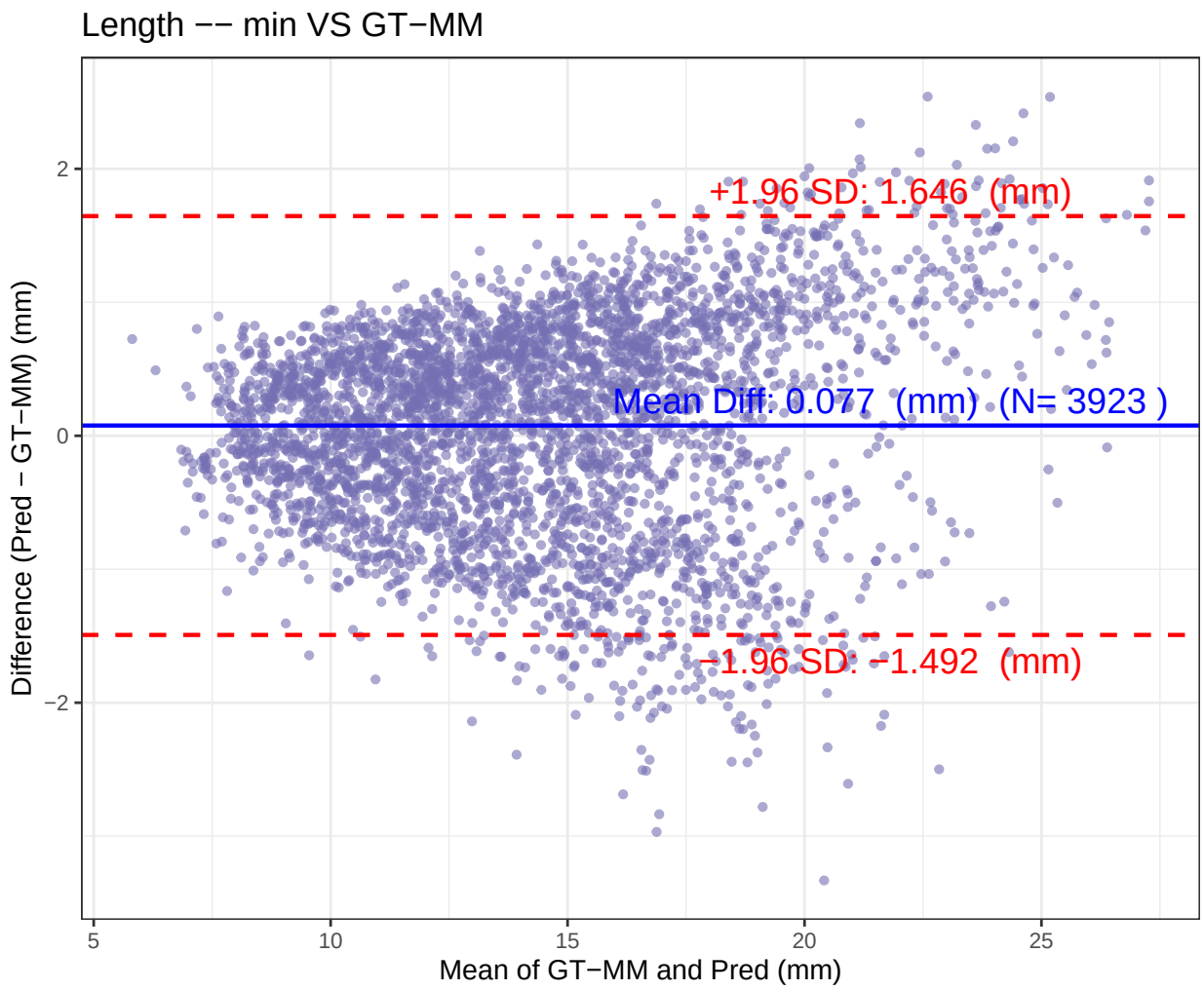


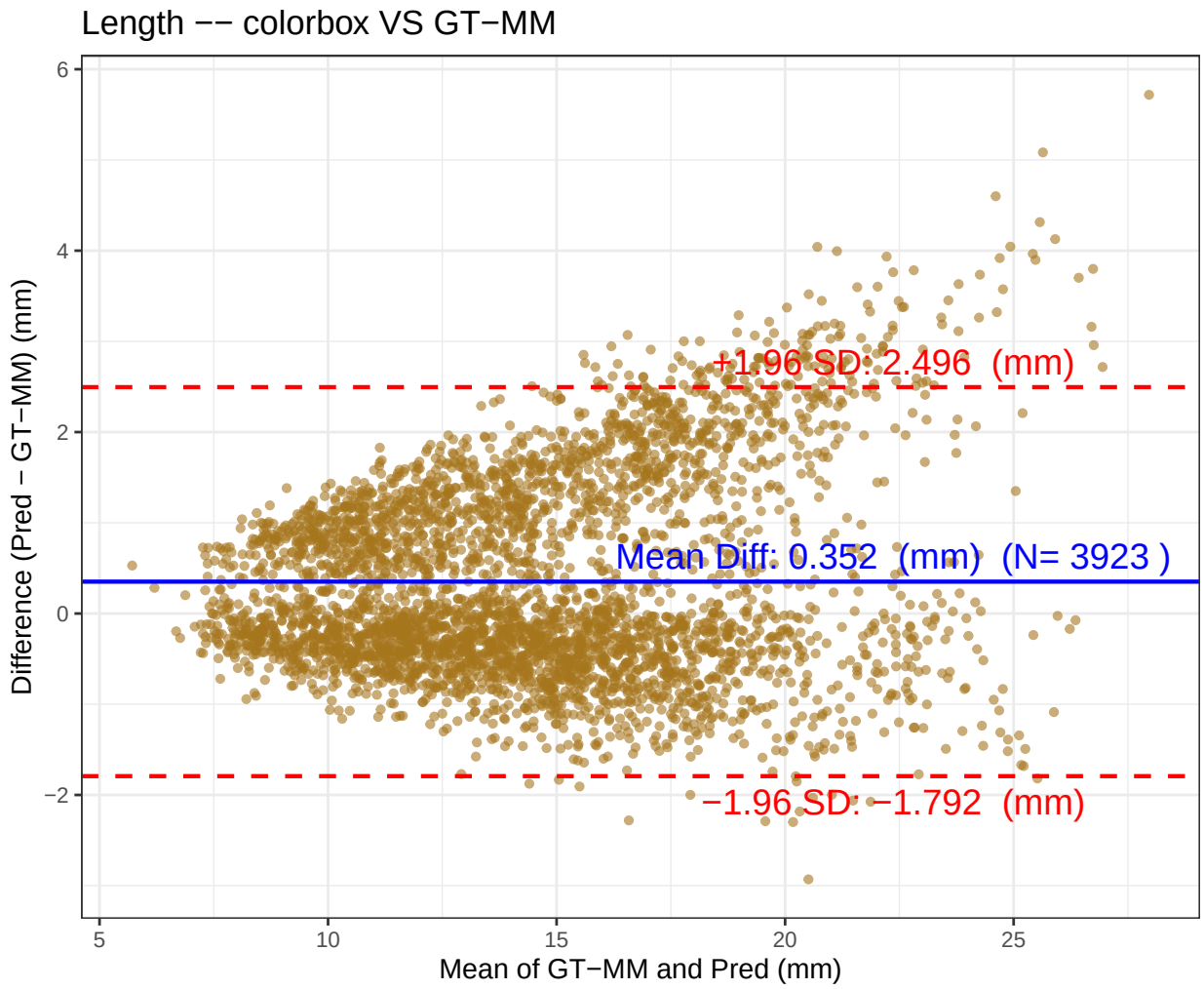

```
save_plots(altman_list, output_dir)
```

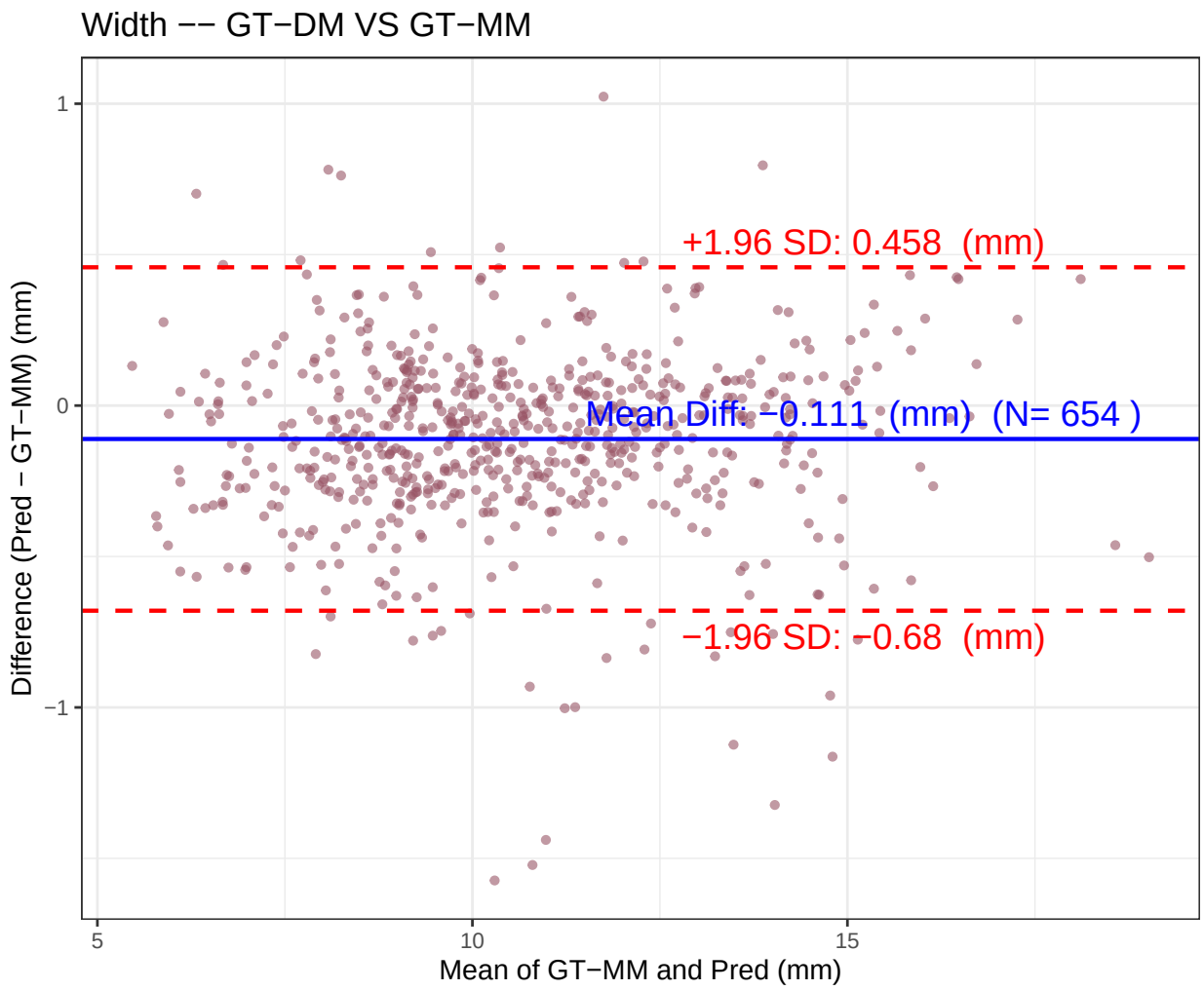

Length −− GT−DM VS GT−MM
+1.96 SD: 0.503 (mm)
Mean Diff: −0.069 (mm) (N= 639)
−1.96 SD: −0.642 (mm)
Difference (Pred − GT−MM) (mm)
Mean of GT−MM and Pred (mm)
0.5
0.0
−0.5
−1.0
−1.5
10
15
20

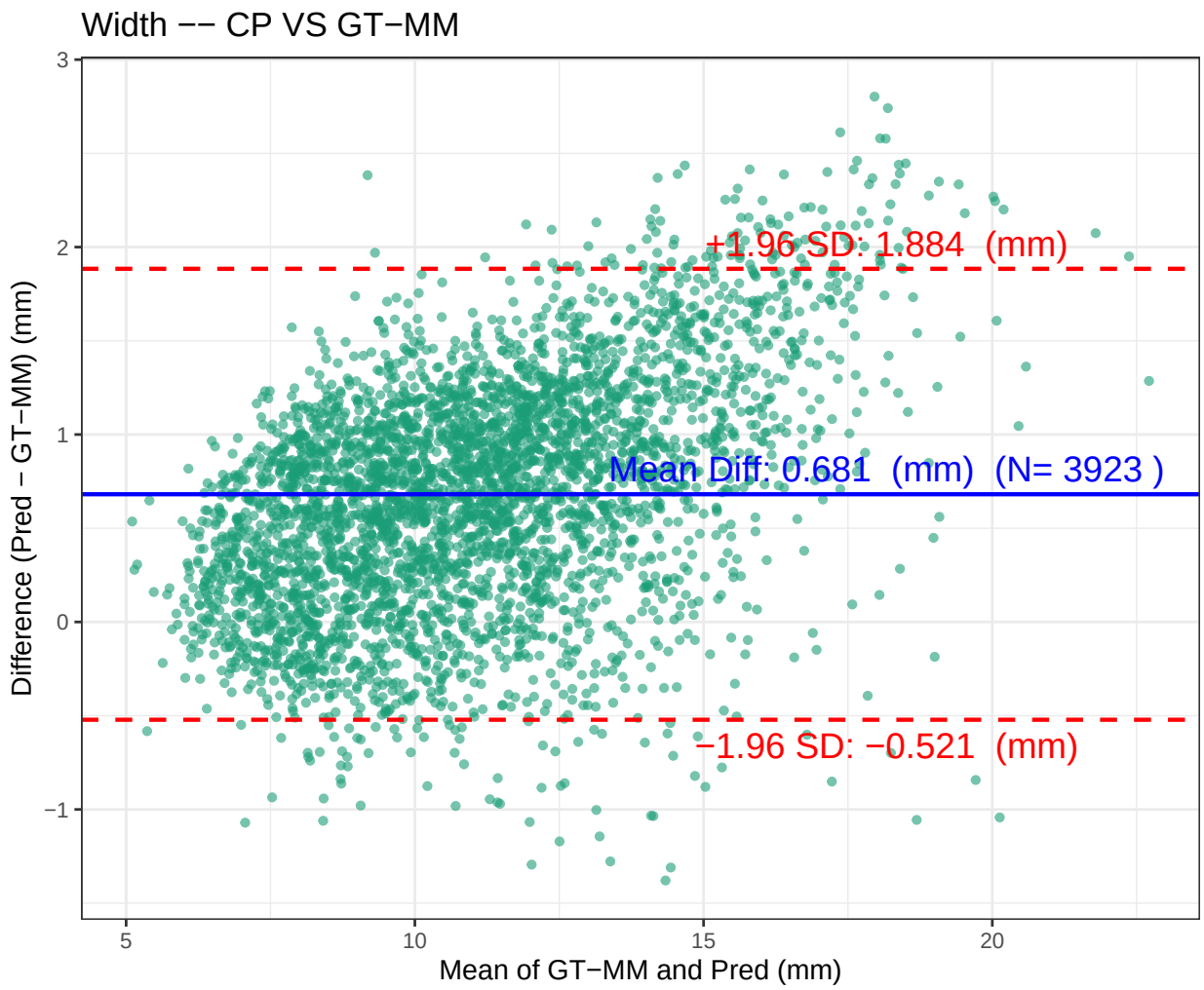

Length −− CP VS GT−MM
+1.96 SD: 1.724 (mm)
Mean Diff: 0.056 (mm) (N= 3923)
−1.96 SD: −1.613 (mm)
Difference (Pred − GT−MM) (mm)
Mean of GT−MM and Pred (mm)
2
0
−2
5
10
15
20
25

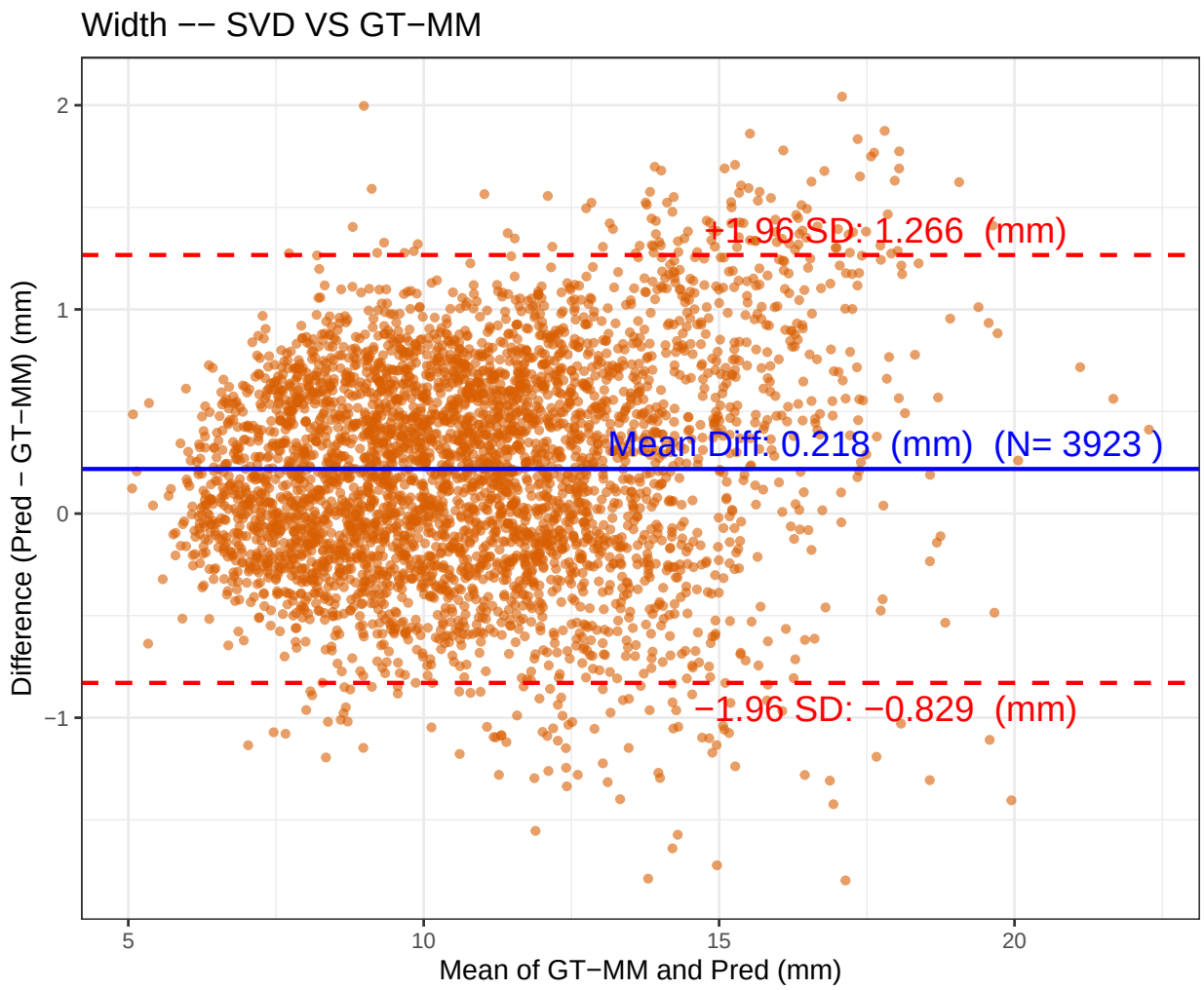

Length −− SVD VS GT−MM
+1.96 SD: 2.195 (mm)
Mean Diff: 0.698 (mm) (N= 3923)
−1.96 SD: −0.799 (mm)
Difference (Pred − GT−MM) (mm)
2
0
−2
5
10
15
20
25
Mean of GT−MM and Pred (mm)

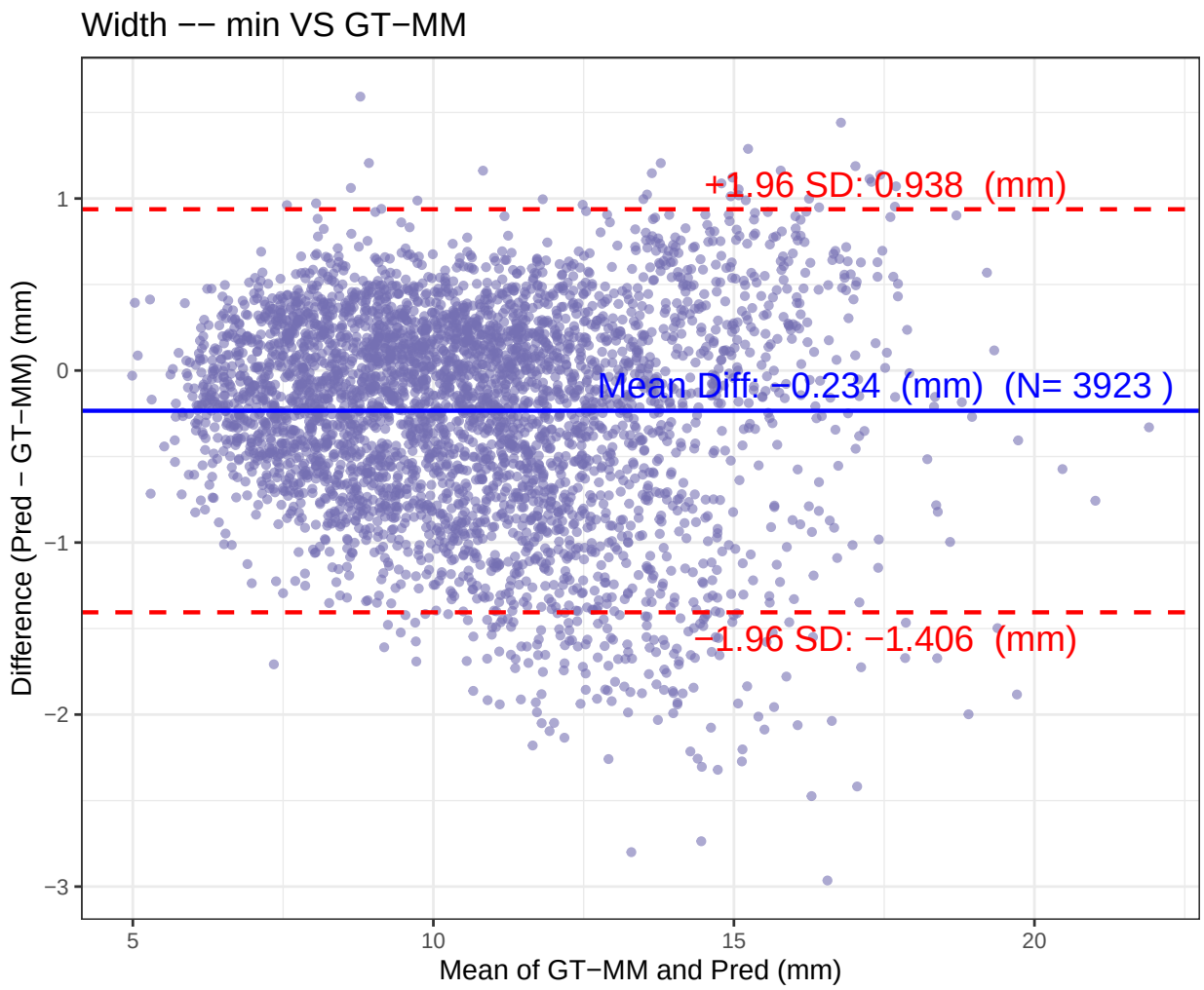

Length −− min VS GT−MM
+1.96 SD: 1.646 (mm)
Mean Diff: 0.077 (mm) (N= 3923)
−1.96 SD: −1.492 (mm)
Difference (Pred − GT−MM) (mm)
2
0
−2
5
10
15
20
25
Mean of GT−MM and Pred (mm)

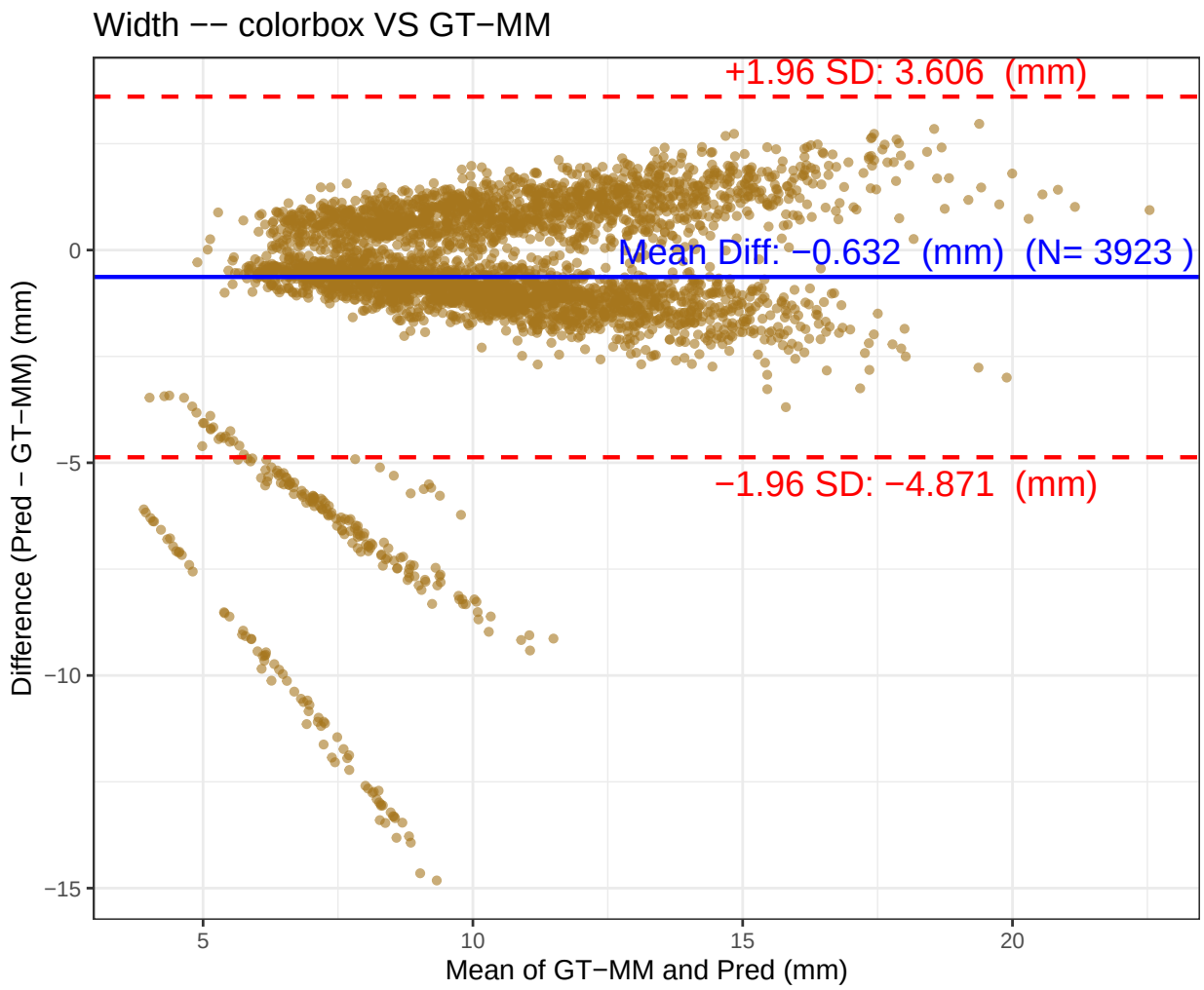

Length −− colorbox VS GT−MM
Difference (Pred − GT−MM) (mm)
Mean of GT−MM and Pred (mm)
+1.96 SD: 2.496 (mm)
Mean Diff: 0.352 (mm) (N= 3923)
−1.96 SD: −1.792 (mm)
6
4
2
0
−2
5
10
15
20
25

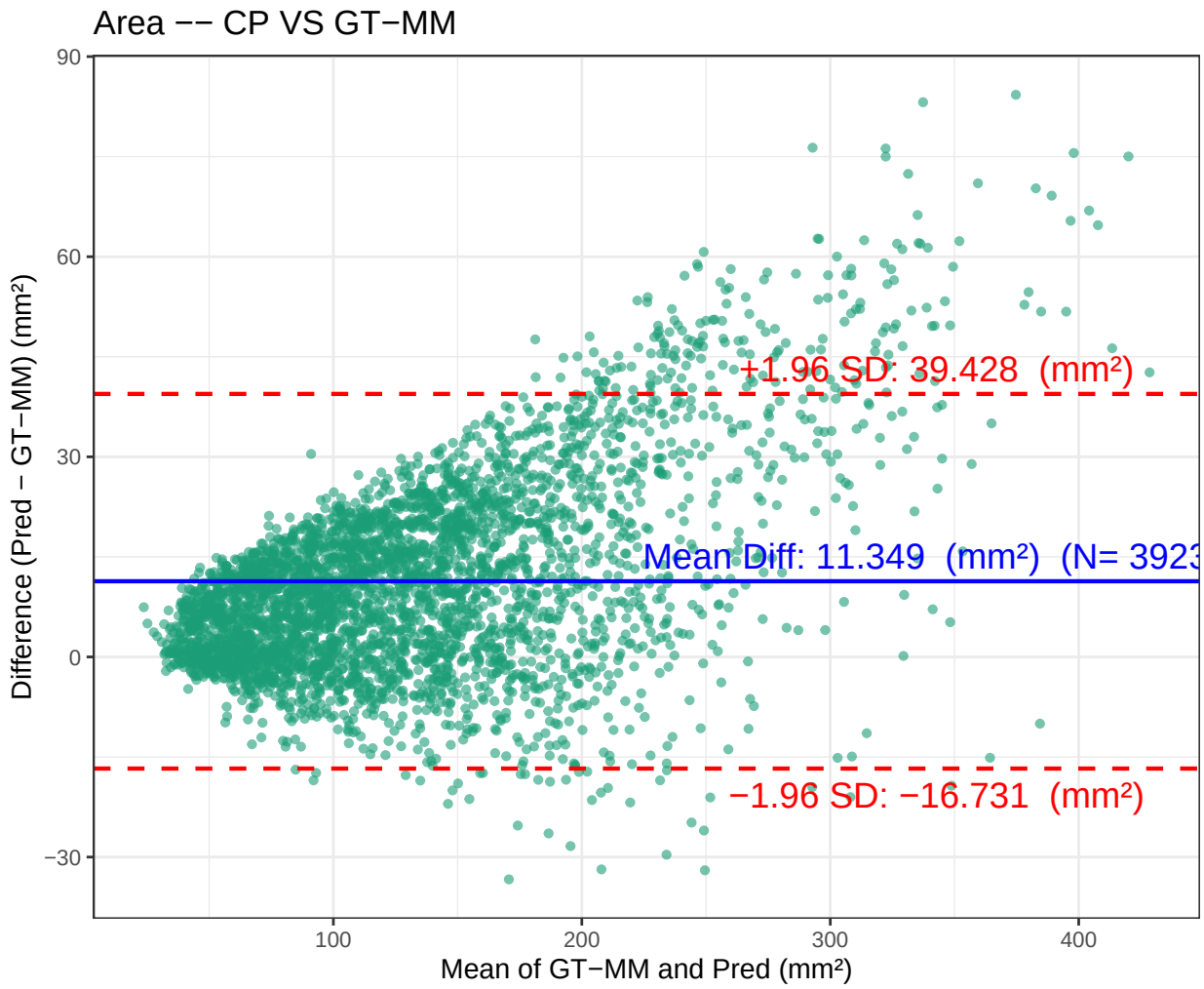

Width −− GT−DM VS GT−MM
+1.96 SD: 0.458 (mm)
Mean Diff: −0.111 (mm) (N= 654)
−1.96 SD: −0.68 (mm)
Difference (Pred − GT−MM) (mm)
Mean of GT−MM and Pred (mm)
1
0
−1
5
10
15

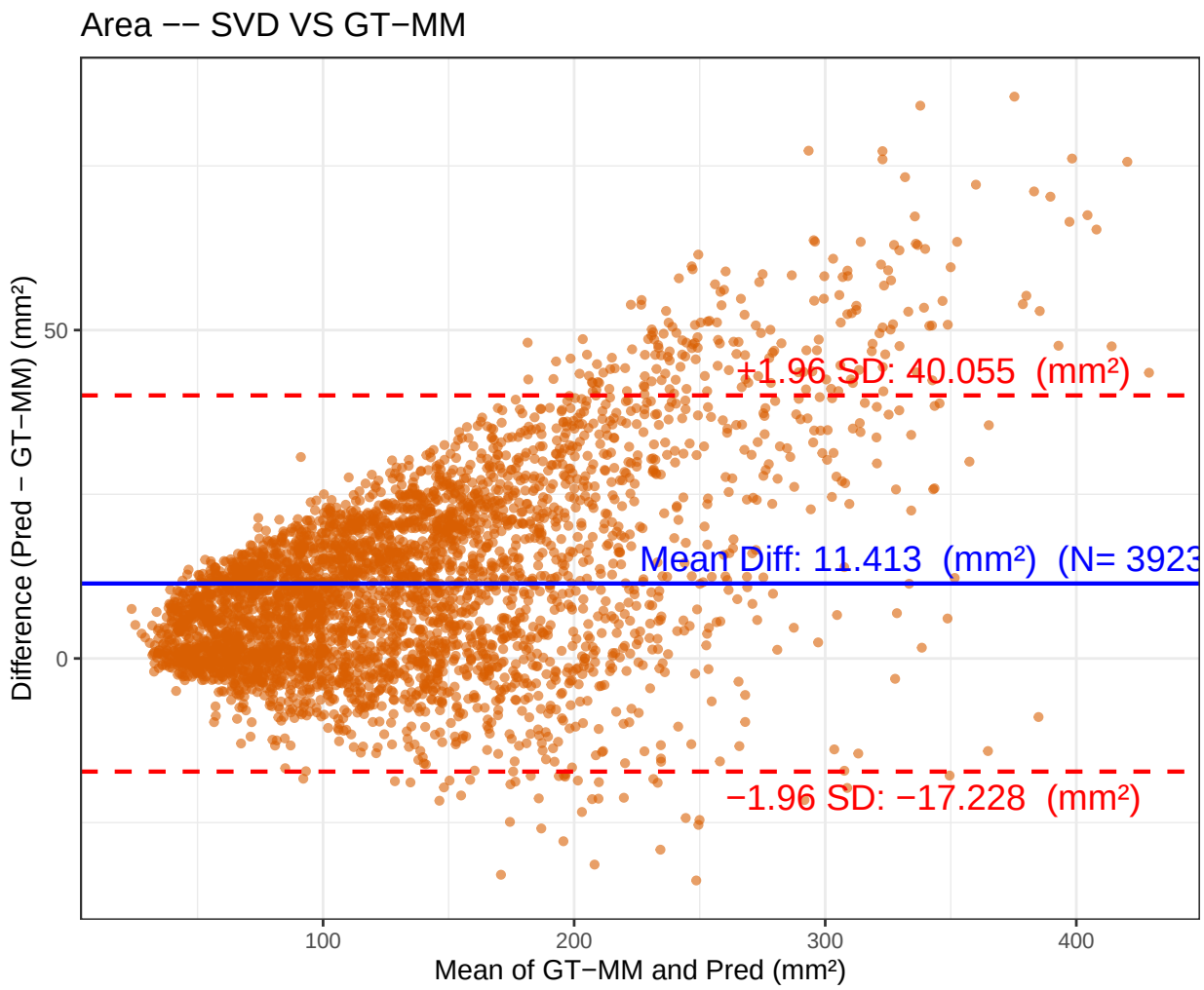

Width −− CP VS GT−MM
+1.96 SD: 1.884 (mm)
Mean Diff: 0.681 (mm) (N= 3923)
−1.96 SD: −0.521 (mm)
Difference (Pred − GT−MM) (mm)
Mean of GT−MM and Pred (mm)
3
2
1
0
−1
5
10
15
20

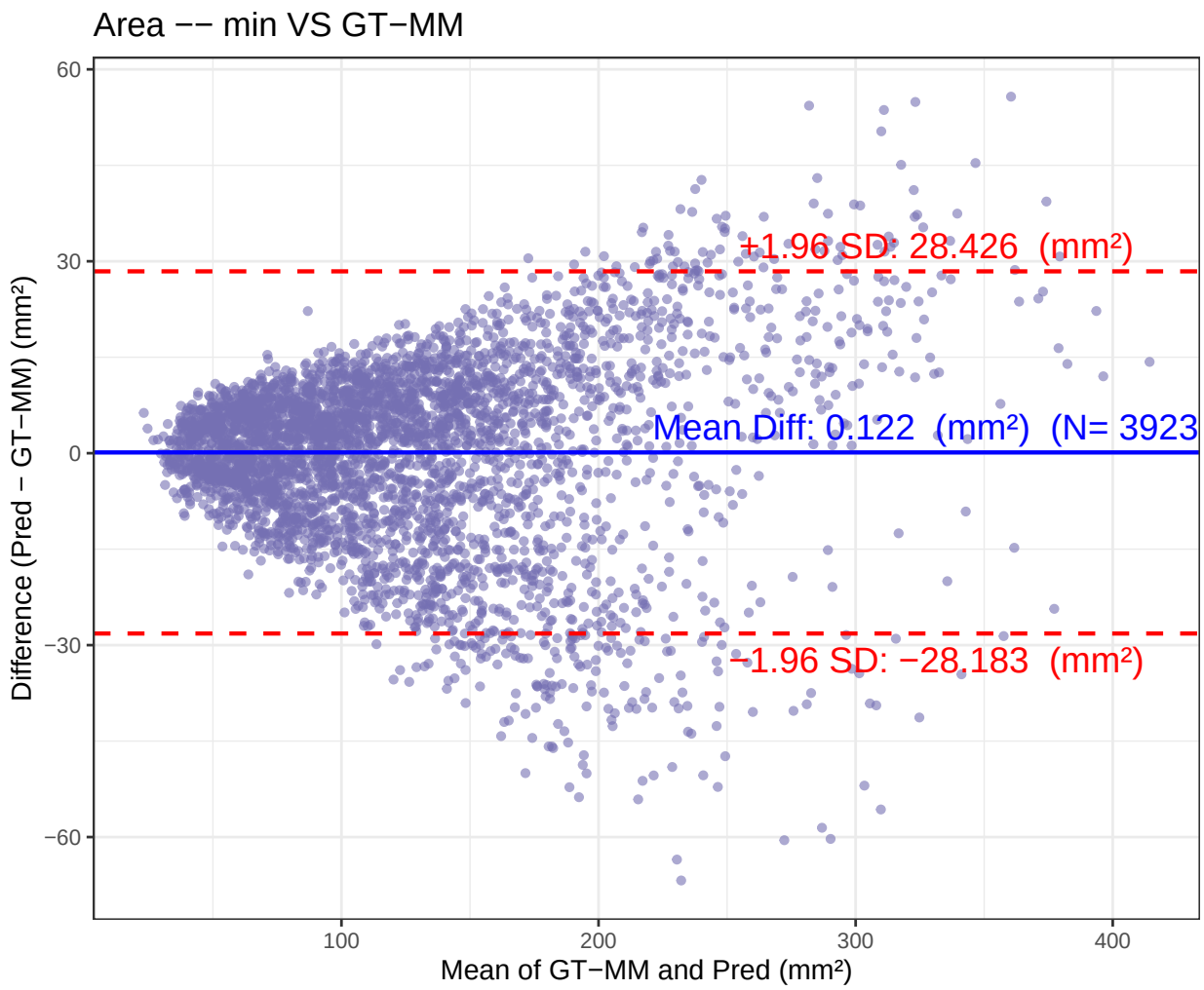

Width −− SVD VS GT−MM
+1.96 SD: 1.266 (mm)
Mean Diff: 0.218 (mm) (N= 3923)
−1.96 SD: −0.829 (mm)
Difference (Pred − GT−MM) (mm)
Mean of GT−MM and Pred (mm)
2
1
0
−1
5
10
15
20


Width −− min VS GT−MM
+1.96 SD: 0.938 (mm)
Mean Diff: −0.234 (mm) (N= 3923)
−1.96 SD: −1.406 (mm)
Difference (Pred − GT−MM) (mm)
Mean of GT−MM and Pred (mm)
1
0
−1
−2
−3
5
10
15
20


Width −− colorbox VS GT−MM
+1.96 SD: 3.606 (mm)
Mean Diff: −0.632 (mm) (N= 3923)
−1.96 SD: −4.871 (mm)
Difference (Pred − GT−MM) (mm)
0
−5
−10
−15
5
10
15
20
Mean of GT−MM and Pred (mm)


Area −− CP VS GT−MM
90
60
30
0
−30
Difference (Pred − GT−MM) (mm²)
+1.96 SD: 39.428 (mm²)
Mean Diff: 11.349 (mm²) (N= 3923
−1.96 SD: −16.731 (mm²)
100
200
300
400
Mean of GT−MM and Pred (mm²)


Area −− SVD VS GT−MM
Difference (Pred − GT−MM) (mm²)
50
0
+1.96 SD: 40.055 (mm²)
Mean Diff: 11.413 (mm²) (N= 3923
−1.96 SD: −17.228 (mm²)
100
200
300
400
Mean of GT−MM and Pred (mm²)




```r
message("--- Plot Saving Complete ---\n")

# Combine all removed seeds into one table
all_removed_seeds <- do.call(bind_rows, removed_seeds_log) %>%
    distinct(Code, .keep_all = TRUE) # Remove duplicates if a seed was both manually excluded AND an ou

# Set Output filenames
summary_filename <- file.path(output_dir, "Regression_Summary_Metrics_Localized_Filtering.xlsx")
outlier_filename <- file.path(output_dir, "Removed_and_Outlier_Seeds.csv")
threshold_filename <- file.path(output_dir, "Outlier_Thresholds_1_5_IQR.xlsx")

# Save Regression Summary to XLSX
tryCatch({
    writexl::write_xlsx(
        x = list("Regression_Summary" = reg_summary_full),
        path = summary_filename
    )
    message(paste0("\nSaved Regression Summary Table to: ", summary_filename))
}, error = function(e) {
    warning(paste0("Could not save XLSX file. Error: ", e$message))
```

```
})

# Save outlier list to CSV
tryCatch({
    write.csv(all_removed_seeds, file = outlier_filename, row.names = FALSE)
    message(paste0("Saved Removed Seed and Outlier List to: ", outlier_filename))
}, error = function(e) {
    warning(paste0("Could not save Removed Seed CSV file. Error: ", e$message))
})

# Save outlier thresholds to XLSX
tryCatch({
    outlier_threshold_df <- do.call(bind_rows, outlier_threshold_data)
    writexl::write_xlsx(
        x = list("Outlier_Thresholds" = outlier_threshold_df),
        path = threshold_filename
    )
    message(paste0("Saved Outlier Thresholds to: ", threshold_filename))
}, error = function(e) {
    warning(paste0("Could not save Threshold XLSX file. Error: ", e$message))
})


cat("\n\n*** Regression Summary Table ***\n")
```

```
##
##
## *** Regression Summary Table ***
```

```
print(reg_summary_full)
```

```
##    Measure   Method    N    R2     R2_Boot_CI   Bias   RMSE Slope
## 1   Length    GT-DM  639 0.993 [0.992, 0.995] -0.069  0.300 1.023
## 2   Length       CP 3923 0.953  [0.95, 0.956]  0.056  0.853 1.005
## 3   Length      SVD 3923 0.970 [0.968, 0.972]  0.698  1.035 1.071
## 4   Length      min 3923 0.958 [0.956, 0.961]  0.077  0.804 1.010
## 5   Length colorbox 3923 0.926  [0.923, 0.93]  0.352  1.149 1.019
## 6    Width    GT-DM  654 0.985 [0.982, 0.988] -0.111  0.310 0.997
## 7    Width       CP 3923 0.963  [0.96, 0.965]  0.681  0.917 1.087
## 8    Width      SVD 3923 0.963  [0.96, 0.965]  0.218  0.577 1.017
## 9    Width      min 3923 0.949 [0.946, 0.952] -0.234  0.642 0.956
## 10   Width colorbox 3923 0.545 [0.504, 0.585] -0.632  2.253 0.882
## 11    Area       CP 3923 0.969 [0.967, 0.971] 11.349 18.275 1.104
## 12    Area      SVD 3923 0.967 [0.965, 0.969] 11.413 18.540 1.103
## 13    Area      min 3923 0.951 [0.948, 0.954]  0.122 14.440 0.996
##      Slope_Reg_CI  Slope_Boot_CI Intercept Intercept_Reg_CI Intercept_Boot_CI
## 1   [1.017, 1.03]  [1.016, 1.03]    -0.401 [-0.496, -0.306]  [-0.501, -0.297]
## 2  [0.998, 1.012] [0.997, 1.013]    -0.016   [-0.12, 0.089]   [-0.119, 0.093]
## 3  [1.065, 1.076] [1.064, 1.077]    -0.313 [-0.401, -0.226]  [-0.401, -0.226]
## 4  [1.004, 1.017] [1.002, 1.018]    -0.069  [-0.166, 0.029]   [-0.172, 0.032]
## 5   [1.01, 1.028] [1.008, 1.029]     0.081  [-0.052, 0.215]   [-0.046, 0.213]
## 6  [0.987, 1.006] [0.986, 1.007]    -0.076  [-0.178, 0.026]   [-0.183, 0.037]
```

```
## 7   [1.08, 1.093] [1.079, 1.094]    -0.248 [-0.322, -0.174]  [-0.328, -0.168]
## 8  [1.011, 1.023]  [1.01, 1.024]     0.035  [-0.034, 0.104]   [-0.039, 0.109]
## 9  [0.949, 0.963] [0.947, 0.964]     0.240   [0.163, 0.316]      [0.16, 0.32]
## 10 [0.857, 0.907]  [0.854, 0.91]     0.636   [0.357, 0.915]    [0.375, 0.909]
## 11  [1.097, 1.11] [1.094, 1.112]    -1.688 [-2.566, -0.809]   [-2.62, -0.712]
## 12 [1.096, 1.109] [1.093, 1.112]    -1.504 [-2.407, -0.601]  [-2.501, -0.495]
## 13 [0.988, 1.003] [0.985, 1.006]     0.683  [-0.315, 1.681]   [-0.303, 1.735]
```

```
cat("\n\n*** Removed and Outlier Seeds (First 8) ***\n")
```

```
##
##
## *** Removed and Outlier Seeds (First 8) ***
```

```
print(head(all_removed_seeds, 8))
```

```
## # A tibble: 8 x 4
##   Code                      Group2                 Reason                Threshold
##   <chr>                     <chr>                  <chr>                 <chr>
## 1 Faba-Seed-CC_Vf198-1-2-0 Faba-Seed-CC_Vf198-1-2 Manual Group Exclus~ <NA>
## 2 Faba-Seed-CC_Vf198-1-2-1 Faba-Seed-CC_Vf198-1-2 Manual Group Exclus~ <NA>
## 3 Faba-Seed-CC_Vf198-1-2-2 Faba-Seed-CC_Vf198-1-2 Manual Group Exclus~ <NA>
## 4 Faba-Seed-CC_Vf198-1-2-3 Faba-Seed-CC_Vf198-1-2 Manual Group Exclus~ <NA>
## 5 Faba-Seed-CC_Vf198-1-2-4 Faba-Seed-CC_Vf198-1-2 Manual Group Exclus~ <NA>
## 6 Faba-Seed-CC_Vf198-1-2-5 Faba-Seed-CC_Vf198-1-2 Manual Group Exclus~ <NA>
## 7 Faba-Seed-CC_Vf198-1-2-6 Faba-Seed-CC_Vf198-1-2 Manual Group Exclus~ <NA>
## 8 Faba-Seed-CC_Vf198-1-2-7 Faba-Seed-CC_Vf198-1-2 Manual Group Exclus~ <NA>
```

```
# ------------------------------------------------------------------------
# 1. Load Required Libraries
# ------------------------------------------------------------------------
library(openxlsx)
library(tidyverse)
library(GGally)

# ------------------------------------------------------------------------
# 2. Read and Prepare Data
# ------------------------------------------------------------------------

file_path <- "Faba_Seed_Analyzer_Data_August_2024.xlsx"
sheet_name <- "Individual S2 Seed Data"
data_columns_all <- c(1, 2, 3, 4, 7, 8)

df <- read.xlsx(
  xlsxFile = file_path,
  sheet = sheet_name,
  startRow = 1,
  cols = data_columns_all,
  colNames = TRUE,
  detectDates = FALSE
)
```

```r
colnames(df) <- c("ID", "Length_mm", "Width_mm", "Area_mm2", "Circularity", "Aspect_Ratio")

df_filtered <- df %>% filter(ID != "Vf620-1-1")

df_clean <- df_filtered %>%
  select(Length_mm, Width_mm, Area_mm2, Circularity, Aspect_Ratio) %>%
  mutate(across(everything(), as.numeric)) %>%
  drop_na()

# ---------------------------------------------------------------------
# 3. Helper Function: Regression Equation Only
# ---------------------------------------------------------------------

lm_eqn <- function(df){
  m <- lm(y ~ x, data = df)

  # Extract coefficients as numeric scalars
  intercept <- as.numeric(coef(m)[1])
  slope <- as.numeric(coef(m)[2])

  # Build equation as plotmath expression
  eq <- substitute(italic(y) == a + b * italic(x),
                   list(a = intercept, b = slope))

  return(eq)
}

# ---------------------------------------------------------------------
# 4. Lower Panel: Scatter + Regression + Equation
# ---------------------------------------------------------------------

custom_lower_with_eq <- function(data, mapping, ...) {

  plot_df <- data.frame(
    x = data[[as_label(mapping$x)]],
    y = data[[as_label(mapping$y)]]
  )

  eq_label <- lm_eqn(plot_df)

  GGally::ggally_points(
    data = data,
    mapping = mapping,
    color = "darkblue",
    size = 1.5,
    alpha = 0.6
  ) +
    geom_smooth(method = "lm", se = FALSE, color = "red") +

    # Add equation
    annotate(
      "text",
      x = 0.05, y = 0.95,
```

```
      label = paste(deparse(eq_label), collapse = ""),
      parse = TRUE,
      size = 3,
      hjust = 0, vjust = 1,
      x.unit = "npc", y.unit = "npc"
    ) +

    theme_bw()
}

# ---------------------------------------------------------------------
# 5. Upper Panel: Correlation
# ---------------------------------------------------------------------

custom_upper_cor <- function(data, mapping, ...) {
  GGally::ggally_cor(
    data = data,
    mapping = mapping,
    size = 4,
    color = "darkred"
  )
}

# ---------------------------------------------------------------------
# 6. Generate Pair Plot
# ---------------------------------------------------------------------

pair_plot <- ggpairs(
  data = df_clean,
  columns = 1:ncol(df_clean),
  title = "Pair Plot of Ground-Truth Faba Seed Traits with Regression Equations",
  diag = list(continuous = "densityDiag"),
  upper = list(continuous = custom_upper_cor),
  lower = list(continuous = custom_lower_with_eq),
  columnLabels = colnames(df_clean)
)

# ---------------------------------------------------------------------
# 7. Save Plot
# ---------------------------------------------------------------------

ggsave(
  filename = "Faba_Seed_PairPlot_EquationOnly.png",
  plot = pair_plot,
  width = 12,
  height = 12,
  dpi = 300
)

print("Pair plot saved as Faba_Seed_PairPlot_EquationOnly.png")
```

```
## [1] "Pair plot saved as Faba_Seed_PairPlot_EquationOnly.png"
```

```r
# --- R Code Chunk: Width Measurement Pair Plot (ALL LOWER PLOTS ANNOTATED) ---
message("\n--- Generating Width Pair Plot (Section 1 Data - ALL Lower Plots Annotated) ---")
```

```
##
## --- Generating Width Pair Plot (Section 1 Data - ALL Lower Plots Annotated) ---
```

```r
# 1. Define the specific columns for the Width Pair Plot
width_cols_for_plot <- paste0("Width-", methods_full)
gt_col_w <- "Width-GT-MM"

# CRITICAL: Select the columns from df_sorted BUT DO NOT drop_na() globally.
df_width_plot_local_base <- df_sorted %>%
    select(Group2, all_of(width_cols_for_plot))

# Check if enough data remains
if (nrow(df_width_plot_local_base) < 2) {
    stop("Insufficient data rows remaining after sorting for Width Pair Plot.")
}

# --- Helper Functions (No Change to Calculation Logic) ---

# Helper function to perform localized filtering and calculate string metrics
lm_stats_strings_local <- function(df_input, x_col, y_col){

    # 1. LOCALIZED GROUP FILTERING (Replicates main chunk's filtering logic)
    df_pair_filtered <- localized_group_filter(df_input, c(x_col, y_col))

    # 2. Extract and Drop NA (Local to this pair)
    df_tmp <- df_pair_filtered %>%
        select(x = all_of(x_col), y = all_of(y_col)) %>%
        drop_na()

    N_final <- nrow(df_tmp)

    if (N_final < 2) {
        return(list(N = 0, r = "r = NA", eq = "y = NA", rmse = "RMSE = NA"))
    }

    # 3. Calculate Metrics
    m <- lm(y ~ x, data=df_tmp);
    res <- residuals(m)
    rmse_val <- sqrt(mean(res^2))
    r_val <- cor(df_tmp$x, df_tmp$y)

    # 4. Format components as simple strings
    a_fmt <- format(coef(m)[1], digits = 3)
    b_fmt <- format(coef(m)[2], digits = 4)
    r_fmt <- format(r_val, digits = 3)
    rmse_fmt <- format(rmse_val, digits = 3)

    # Simple text strings for display
    n_str <- paste0("N = ", N_final)
    eq_str <- paste0("y = ", a_fmt, " + ", b_fmt, "x")
```

```
    r_str <- paste0("r = ", r_fmt)
    rmse_str <- paste0("RMSE = ", rmse_fmt)

    return(list(N = n_str, r = r_str, eq = eq_str, rmse = rmse_str))
}


# Custom Lower Panel: Scatter plot with regression line and STATISTIC STRINGS
# CRITICAL FIX: The conditional check has been REMOVED.
custom_lower_points_stats_local_all <- function(data, mapping, ...) {
    # 1. Identify the column names
    x_col_name <- rlang::as_label(mapping$x)
    y_col_name <- rlang::as_label(mapping$y)

    # 2. CRITICAL: Recalculate metrics using the LOCAL filtering logic on the full base data
    # This runs for EVERY panel in the lower triangle.
    stats <- lm_stats_strings_local(
        df_input = df_width_plot_local_base, # Use the base, non-globally-filtered data
        x_col = x_col_name,
        y_col = y_col_name
    )

    # 3. Define data anchor for text
    plot_df <- data.frame(x = data[[x_col_name]], y = data[[y_col_name]])
    x_range <- max(plot_df$x, na.rm = TRUE) - min(plot_df$x, na.rm = TRUE)
    y_range <- max(plot_df$y, na.rm = TRUE) - min(plot_df$y, na.rm = TRUE)

    x_pos <- min(plot_df$x, na.rm = TRUE) + 0.05 * x_range
    y_pos <- max(plot_df$y, na.rm = TRUE) - 0.05 * y_range

    # 4. Use the stable ggally_points wrapper for the scatter plot
    p <- GGally::ggally_points(
        data = data,
        mapping = mapping,
        # Ensure points are colored
        color = "darkblue",
        size = 1.5,
        alpha = 0.6
    ) +
        # Ensure regression line is drawn
        geom_smooth(method = "lm", se = FALSE, color = "red") +

        # 5. Annotate with STATIC metrics (using simple strings and data coordinates)

        # Add N value (Top)
        annotate(
            "text", x = x_pos, y = y_pos, label = stats$N, size = 3,
            hjust = 0, vjust = 1
        ) +

        # Add R value (Second down)
        annotate(
            "text", x = x_pos, y = y_pos - 0.08 * y_range,
```

```
            label = stats$r, size = 3,
            hjust = 0, vjust = 1
        ) +

        # Add Equation (Third down)
        annotate(
            "text", x = x_pos, y = y_pos - 0.16 * y_range,
            label = stats$eq, size = 3,
            hjust = 0, vjust = 1
        ) +

        # Add RMSE (Fourth down)
        annotate(
            "text", x = x_pos, y = y_pos - 0.24 * y_range,
            label = stats$rmse, size = 3,
            hjust = 0, vjust = 1
        ) +

        theme_bw()

    return(p)
}


# Custom Upper Panel: Correlation Value (No change needed)
custom_upper_cor <- function(data, mapping, ...) {
    GGally::ggally_cor(
        data = data,
        mapping = mapping,
        size = 4,
        color = "darkred"
    )
}

# --- 4. Generate the Pair Plot ---
# Note: Use the minimally filtered data here
width_pair_plot <- ggpairs(
    data = df_width_plot_local_base %>% select(-Group2) %>% drop_na(any_of(width_cols_for_plot)),
    columns = 1:length(width_cols_for_plot),
    title = "Pair Plot: Correlation Between All Width Measurement Methods",

    diag = list(continuous = "densityDiag"),
    upper = list(continuous = custom_upper_cor),
    # Use the new function that annotates ALL panels
    lower = list(continuous = custom_lower_points_stats_local_all),

    columnLabels = sub("Width-", "", width_cols_for_plot)
)


# --- 5. Save the Plot ---
width_pair_plot_filename <- here::here(output_dir, paste0("PairPlot_Width_AllMethods_AllAnnotated.png"))
ggsave(
```

```
    filename = width_pair_plot_filename,
    plot = width_pair_plot,
    width = 12,
    height = 12,
    dpi = 300
)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```

```
message(paste("Width Pair Plot successfully saved to:", width_pair_plot_filename))
```

```
## Width Pair Plot successfully saved to: C:/Users/wanghao/OneDrive - AGR-AGR/Desktop/FABA BEAN/output
```

Summary Table

```
# ------------------------------------------------------------------------
# 1. Load Required Libraries
# ------------------------------------------------------------------------
library(openxlsx)
library(dplyr)
library(tidyr)
library(gt)
library(scales)
library(webshot2)

# ------------------------------------------------------------------------
# 2. Read Feature Names from taubinSVD (all)
# ------------------------------------------------------------------------
file_path <- "Summary_Faba Bean Correction.xlsx"
sheet_name <- "taubinSVD (all)"

all_headers <- read.xlsx(file_path, sheet = sheet_name, rowNames = FALSE, colNames = TRUE, rows = 1)
feature_cols <- c(3:6, 13:25, 27:30)  # C:F, M:Y, AA:AD
feature_names <- names(all_headers)[feature_cols]
```

```r
# ------------------------------------------------------------------------
# 3. Read full data
# ------------------------------------------------------------------------
df_features <- read.xlsx(file_path, sheet = sheet_name, colNames = TRUE)
id_col <- "Class"
df_features <- df_features %>% select(all_of(id_col), all_of(feature_names))

# ------------------------------------------------------------------------
# 4. Clean Class names
# ------------------------------------------------------------------------
df_features <- df_features %>% mutate(Class = sub("^Faba-Seed-CC_", "", Class))

# ------------------------------------------------------------------------
# 5. Remove unwanted groups
# ------------------------------------------------------------------------
groups_to_remove <- c("460", "117", "198", "619", "283", "90", "615", "456", "280", "139", "620")
df_features <- df_features %>%
  filter(!sapply(Class, function(x) any(sapply(groups_to_remove, function(g) grepl(g, x)))))

# ------------------------------------------------------------------------
# 6. Prepare Table Data
# ------------------------------------------------------------------------
table_list <- lapply(feature_names, function(feat) {

  seed_min <- df_features %>% filter(!!sym(feat) == min(!!sym(feat), na.rm = TRUE))
  seed_max <- df_features %>% filter(!!sym(feat) == max(!!sym(feat), na.rm = TRUE))

  df_class <- df_features %>%
    group_by(Class) %>%
    summarize(avg_feat = mean(!!sym(feat), na.rm = TRUE), .groups = "drop")

  class_min <- df_class %>% filter(avg_feat == min(avg_feat))
  class_max <- df_class %>% filter(avg_feat == max(avg_feat))

  tibble(
    `S.No.` = NA,
    `Feature Name` = feat,
    `Class (Min)` = seed_min$Class[1],
    `Min Value` = formatC(seed_min[[feat]][1], format = "f", digits = 6),
    `Class (Max)` = seed_max$Class[1],
    `Max Value` = formatC(seed_max[[feat]][1], format = "f", digits = 6),
    `Class Avg (Min)` = class_min$Class[1],
    `Min Avg` = formatC(class_min$avg_feat[1], format = "f", digits = 6),
    `Class Avg (Max)` = class_max$Class[1],
    `Max Avg` = formatC(class_max$avg_feat[1], format = "f", digits = 6)
  )
})

table_df <- bind_rows(table_list)
table_df$`S.No.` <- 1:nrow(table_df)

# ------------------------------------------------------------------------
# 7. Generate gt Table with Multi-level Headers and Colors
```

```r
# ----------------------------------------------------------------------
gt_table <- table_df %>%
  gt() %>%

  # Main title
  tab_header(title = "Faba Seed Features Summary (TaubinSVD Correction)") %>%

  # Spanner (second header) - Assigning the names needed for styling
  tab_spanner(label = "Features",
              columns = c("S.No.", "Feature Name")) %>%
  tab_spanner(label = "Data at Seed Level",
              columns = c("Class (Min)", "Min Value", "Class (Max)", "Max Value")) %>%
  tab_spanner(label = "Average Data at Class Level",
              columns = c("Class Avg (Min)", "Min Avg", "Class Avg (Max)", "Max Avg")) %>%

  # Right-align numeric columns
  cols_align(
    align = "right",
    columns = c("Min Value", "Max Value", "Min Avg", "Max Avg")
  ) %>%

  # Format numbers
  fmt_number(
    columns = c("Min Value", "Max Value", "Min Avg", "Max Avg"),
    decimals = 6
  ) %>%

  # ------------------------------------------------------------
  # Fix: Background colors for SPANNER HEADER
  # ------------------------------------------------------------
  tab_style(
    style = cell_fill(color = "#DFF0D8"),
    locations = cells_column_spanners(spanners = "Features")
  ) %>%
  tab_style(
    style = cell_fill(color = "#D9EDF7"),
    locations = cells_column_spanners(spanners = "Data at Seed Level")
  ) %>%
  tab_style(
    style = cell_fill(color = "#FCF8E3"),
    locations = cells_column_spanners(spanners = "Average Data at Class Level")
  ) %>%

  # Smaller font and padding (Incompatible argument removed)
  tab_options(
    table.font.size = px(9),
    data_row.padding = px(2)
  )

# ----------------------------------------------------------------------
# 8. Save Table as Single-Page PDF
# ----------------------------------------------------------------------
gtsave(gt_table, "Faba_Seed_Feature_Summary.pdf")
```