```
library(ggplot2)
library(plotly)
library(copula)
library(patchwork)
library(dplyr)
library(tibble)
library(MASS)
library(randomForest)
library(mgcv)
library(ks)
library(tidyr)
library(evd)
library(mvtnorm)
library(quantreg)
library(splines)
library(nnet)
library(ggridges)
library(shiny)
library(shinythemes)
library(rlang)
library(gt)
library(rsconnect)


# Funkcie

compute_area <- function(y, f) {
  valid <- is.finite(y) & is.finite(f)
  y <- y[valid]
  f <- f[valid]

  if (length(y) < 2) return(NA)

  sum(diff(y) * (head(f, -1) + tail(f, -1)) / 2)
}

apply_dark_gt_theme <- function(gt_tbl, highlight_rows = NULL,
highlight_color = "#ffe0b2") {
  gt_tbl <- gt_tbl %>%
    # Základné nastavenia tmavého štýlu
    gt::tab_options(
      table.background.color = "#2b2b2b",
      column_labels.background.color = "#2b2b2b",
      heading.background.color = "#2b2b2b",
      table.font.color = "white",
      data_row.padding = gt::px(6),
      row.striping.background_color = "#3a3a3a"
    ) %>%

    # Štýl titulku
    gt::tab_style(
      style = list(gt::cell_text(color = "white")),
      locations = gt::cells_title(groups = "title")
    ) %>%

    # Štýl podtitulku
    gt::tab_style(
      style = list(gt::cell_text(color = "gray")),
      locations = gt::cells_title(groups = "subtitle")
    ) %>%

    # Štýl hlavičiek stĺpcov
    gt::tab_style(
      style = list(
        gt::cell_fill(color = "#2b2b2b"),
        gt::cell_text(color = "white", weight = "bold")
      ),
      locations = gt::cells_column_labels(columns =
gt::everything())
    )

  # Zvýraznenie vybraných riadkov, ak sú zadané
  if (!is.null(highlight_rows)) {
    gt_tbl <- gt_tbl %>%
      gt::tab_style(
        style = gt::cell_fill(color = highlight_color),
        locations = gt::cells_body(
          columns = gt::everything(),
          rows = highlight_rows
        )
      ) %>%
      gt::tab_style(
        style = gt::cell_text(weight = "bold", color = "black"),
        locations = gt::cells_body(
          columns = gt::everything(),
          rows = highlight_rows
        )
      )
  }

  return(gt_tbl)
}
```

```
standardize_gt_table <- function(gt_tbl) {
  gt_tbl %>%
    gt::tab_options(
      table.font.size = px(13),
      data_row.padding = px(6),
      table.width = pct(100),
      heading.title.font.size = px(14)
    )
}

identify_variables <- function(data) {
  variable_types <- sapply(data, function(col) {
    if (is.factor(col) || is.character(col) || (is.numeric(col)
&& length(unique(col)) < 10)) {
      return("Diskretna")
    } else {
      return("Spojita")
    }
  })

  discrete_vars <- names(variable_types[variable_types ==
"Diskretna"])
  continuous_vars <- names(variable_types[variable_types ==
"Spojita"])

  return(list(Diskretne = discrete_vars, Spojite =
continuous_vars))
}

printVariables <- function(data) {
  num_variables <- length(colnames(data))
  variables <- identify_variables(data)

  all_variables <- c(variables$Diskretne, variables$Spojite)

  variable_types <- c(rep("Discrete",
length(variables$Diskretne)),
                      rep("Continuous",
length(variables$Spojite)))

  counts <- sapply(all_variables, function(var)
sum(!is.na(data[[var]])))

  tibble(
    Index = seq_len(num_variables),
    Variable_Name = all_variables,
    Variable_Type = variable_types,
    Pocet_pozorovani = counts
  )
}

model_mixture_density <- function(data, discrete_vars,
continuous_vars, model_type = "kernel", bw = NULL) {

  all_levels <- sort(unique(data[[discrete_vars]]))
  data <- data %>% filter(!is.na(.data[[discrete_vars]]),
!is.na(.data[[continuous_vars]]))

  if (is.factor(data[[continuous_vars]]) ||
is.character(data[[continuous_vars]])) {
    if (all(grepl("^-?\\d+(\\.\\d+)?$",
as.character(data[[continuous_vars]])))) {
      data[[continuous_vars]] <-
as.numeric(as.character(data[[continuous_vars]]))
    } else {
      stop("Spojitá premenná obsahuje nečíselné hodnoty. Zmeň
alebo uprav dáta.")
    }
  }

  # Konverzia diskretnej premennej na faktor
  data[[discrete_vars]] <- factor(data[[discrete_vars]], levels =
all_levels)
  categories <- levels(data[[discrete_vars]])

  palette_size <- length(categories)
  palette_colors <- if (palette_size <= 9) {
    RColorBrewer::brewer.pal(palette_size, "Set1")
  } else {
    scales::hue_pal()(palette_size)
  }
  category_colors <- setNames(palette_colors, categories)

  # Pravdepodobnosti kategorii
  category_probs <- prop.table(table(data[[discrete_vars]]))

  bw_used_list <- list()

  if (is.factor(data[[continuous_vars]]) ||
is.character(data[[continuous_vars]])) {
    data[[continuous_vars]] <-
```

```
as.numeric(as.character(data[[continuous_vars]]))
  }

  x_global_vals <-
as.numeric(as.character(data[[continuous_vars]]))
  x_range <- range(x_global_vals, na.rm = TRUE)

  # Funkcia na vypocet hustoty podla model_type
  calculate_density <- switch(
    model_type,

    "kernel" = function(data, category) {
      sub_data <- dplyr::filter(data, .data[[discrete_vars]] ==
category)
      if (nrow(sub_data) > 1) {
        bw_use <- if (is.null(bw))
bw.nrd0(sub_data[[continuous_vars]]) else bw
        bw_used_list[[category]] <<- bw_use
        kde <- density(sub_data[[continuous_vars]], bw = bw_use)
        kde_fun <- approxfun(kde$x, kde$y, rule = 2)
        #x_range <- range(data[[continuous_vars]], na.rm = TRUE)
        x <- seq(x_range[1], x_range[2], length.out = 100)
        density <- kde_fun(x)
        weighted_density <- density * category_probs[[category]]
        tibble::tibble(Continuous_Var = x, Density =
weighted_density, Discrete_Var = as.character(category))
      } else {
        tibble::tibble(Continuous_Var = numeric(0), Density =
numeric(0), Discrete_Var = character(0))
      }
    },

    "normal" = function(data, category) {
      sub_data <- dplyr::filter(data, .data[[discrete_vars]] ==
category)
      x_vals <-
as.numeric(as.character(sub_data[[continuous_vars]]))
      x_vals <- x_vals[!is.na(x_vals)]

      if (length(x_vals) > 1 && !is.na(sd(x_vals)) && sd(x_vals)
> 0) {
        x_range <- range(x_global_vals, na.rm = TRUE)
        mu <- mean(x_vals)
        sigma <- sd(x_vals)
        x <- seq(x_range[1], x_range[2], length.out = 100)
        density <- dnorm(x, mean = mu, sd = sigma)
        weighted_density <- density *
category_probs[[as.character(category)]]
        message("Category: ", category, ", Length: ",
length(x_vals), ", sd: ", sd(x_vals))
        return(tibble::tibble(
          Continuous_Var = x,
          Density = weighted_density,
          Discrete_Var = as.character(category)
        ))
      } else {
        return(tibble::tibble(
          Continuous_Var = numeric(0),
          Density = numeric(0),
          Discrete_Var = character(0)
        ))
      }
    },

    "t" = function(data, category) {
      sub_data <- dplyr::filter(data, .data[[discrete_vars]] ==
category)
      x_vals <-
as.numeric(as.character(sub_data[[continuous_vars]]))
      x_vals <- x_vals[!is.na(x_vals)]

      if (length(x_vals) > 2 && !is.na(sd(x_vals)) && sd(x_vals)
> 0) {
        x_range <- range(x_global_vals, na.rm = TRUE)
        mu <- mean(x_vals)
        sigma <- sd(x_vals)
        df <- length(x_vals) - 1
        x <- seq(x_range[1], x_range[2], length.out = 100)
        density <- dt((x - mu) / sigma, df = df) / sigma
        weighted_density <- density *
category_probs[[as.character(category)]]
        tibble::tibble(Continuous_Var = x, Density =
weighted_density, Discrete_Var = as.character(category))
        message("Category: ", category, ", Length: ",
length(x_vals), ", sd: ", sd(x_vals))

        return(tibble::tibble(
          Continuous_Var = x,
          Density = weighted_density,
          Discrete_Var = as.character(category)
        ))
      } else {
        return(tibble::tibble(
          Continuous_Var = numeric(0),
          Density = numeric(0),
          Discrete_Var = character(0)
        ))
      }
    }
  )

  valid_categories <- categories[sapply(categories, function(cat)
{
    sub_data <- dplyr::filter(data, .data[[discrete_vars]] ==
cat)
    x_vals <-
as.numeric(as.character(sub_data[[continuous_vars]]))
    x_vals <- x_vals[!is.na(x_vals)]
    length(x_vals) > 2 && !is.na(sd(x_vals)) && sd(x_vals) > 0
  })]

  # Vypocet hustot pre vsetky kategorie
  density_data <- dplyr::bind_rows(lapply(valid_categories,
function(cat) calculate_density(data, cat)))
  missing_categories <- setdiff(categories,
unique(density_data$Discrete_Var))

  if (length(missing_categories) > 0) {
    empty_rows <- dplyr::bind_rows(lapply(missing_categories,
function(cat) {
      tibble::tibble(
        Continuous_Var = mean(x_global_vals, na.rm = TRUE),
        Density = 0,
        Discrete_Var = as.character(cat)
      )
    }))
    density_data <- dplyr::bind_rows(density_data, empty_rows)
  }

  # Vypocet celkoveho integralu pre kontrolu
  total_integral <- sum(sapply(categories, function(cat) {
    sub_density <- density_data[density_data$Discrete_Var == cat,
]
    if (nrow(sub_density) > 1) {
      dx_values <- diff(sub_density$Continuous_Var)
      dx <- if (length(dx_values) > 0) dx_values[1] else NA
      if (!is.na(dx)) {
        return(sum(sub_density$Density) * dx)
      }
    }
    return(0)
  }))

  # Vystup
  bw_detail <- if (model_type == "kernel" && is.null(bw)) {
    bw_vals <- unlist(bw_used_list)
    if (length(bw_vals) > 0) {
      paste(paste0(names(bw_vals), ": ", round(bw_vals, 4)),
collapse = "; ")
    } else {
      "Nepodarilo sa vypočítať rozsahy vyhladzovania."
    }
  } else if (model_type == "kernel" && !is.null(bw)) {
    as.character(bw)
  } else {
    "Nepoužité"
  }

  return(list(
    density_data = density_data,
    category_probs = category_probs,
    summary_info = list(
      n_categories = length(categories),
      categories = categories,
      model_type = model_type,
      bandwidth = bw_detail,
      total_integral = total_integral
    ),
    discrete_var = discrete_vars,
    continuous_var = continuous_vars,
    category_colors = category_colors,
    data = data,
    vector_type = "mix"
  ))
}

render_mixture_density <- function(model_output, plot_type =
"2D") {

  data <- model_output$density_data
  discrete_var <- model_output$discrete_var
  continuous_var <- model_output$continuous_var
  colors <- model_output$category_colors
```

```
  categories <- levels(as.factor(data$Discrete_Var))

  if (plot_type == "3D") {
    plot <- plot_ly(
      data,
      x = ~Continuous_Var,
      y = ~Discrete_Var,
      z = ~Density,
      type = "scatter3d",
      mode = "lines",
      color = ~Discrete_Var,
      colors = colors,
      line = list(width = 4)
    ) %>%
      layout(scene = list(
        xaxis = list(title = continuous_var),
        yaxis = list(title = discrete_var, type = "category"),
        zaxis = list(title = "Hustota")
      ))

  } else { # plot_type == "2D"
    raw_data <- model_output$data
    raw_data[[discrete_var]] <- factor(raw_data[[discrete_var]],
levels = categories)

    reference_bw <- bw.nrd0(raw_data[[continuous_var]])
    adjust_factor <-
as.numeric(model_output$summary_info$bandwidth) / reference_bw

    # Scatter plot
    scatter_plot <- ggplot(raw_data, aes_string(x =
continuous_var, y = discrete_var, color = discrete_var)) +
      geom_point(size = 3, alpha = 0.7, show.legend = FALSE) +
      labs(x = continuous_var, y = discrete_var) +
      scale_color_manual(values = colors, guide = "none") +
      theme_minimal() +
      theme(legend.position = "right")

    data_density <- model_output$density_data

    x_density <- ggplot(data_density, aes(x = Continuous_Var, y =
Density, color = Discrete_Var)) +
      geom_line(size = 1) +
      scale_color_manual(values = colors) +
      labs(x = NULL, y = paste("Hustota (", continuous_var, ")",
sep = ""), color = discrete_var) +
      theme_minimal() +
      theme(axis.text.x = element_blank(), axis.ticks.x =
element_blank(), legend.position = "right")

    y_bar <- ggplot(raw_data, aes_string(x = discrete_var, fill =
discrete_var)) +
      geom_bar(alpha = 0.7) +
      coord_flip() +
      scale_fill_manual(values = colors) +
      labs(x = NULL, y = paste("P(", discrete_var, ")", sep =
"")) +
      theme_minimal() +
      theme(axis.text.y = element_blank(), axis.ticks.y =
element_blank(), legend.position = "right")

    plot <- (x_density + patchwork::plot_spacer()) /
      (scatter_plot + y_bar) +
      patchwork::plot_layout(widths = c(4, 1), heights = c(1, 4))
  }

  summary_tbl <- tibble::tibble(
    Name = c(
      "Number of Categories", "Names of Categories",
      "Model Type", "Bandwidth", "Total Integral"
    ),
    Value = c(
      model_output$summary_info$n_categories,
      paste(model_output$summary_info$categories, collapse = ",
"),
      model_output$summary_info$model_type,
      model_output$summary_info$bandwidth,
      round(model_output$summary_info$total_integral, 4)
    )
  )

  summary_table <- gt::gt(summary_tbl) %>%
    gt::tab_header(
      title = gt::md("**Model Summary**")
    ) %>%
    gt::cols_width(Name ~ px(220), Value ~ px(420)) %>%
    gt::opt_row_striping() %>%
    gt::opt_table_font(font = "Arial")

  summary_table <- apply_dark_gt_theme(
    gt_tbl = summary_table,
    highlight_rows = NULL,
    highlight_color = NULL
  )

  return(list(
    plot = plot,
    summary = summary_table
  ))
}

model_continuous_density <- function(data, continuous_vars,
model_type = "kernel") {

  result <- list()

  if (model_type == "kernel") {

    kde_result <- MASS::kde2d(
      x = data[[continuous_vars[2]]],
      y = data[[continuous_vars[1]]],
      n = 100
    )

    result$x_vals <- kde_result$x
    result$y_vals <- kde_result$y
    result$z_matrix <- t(kde_result$z)
    result$model_type <- model_type
    result$continuous_vars <- continuous_vars

    result$summary_info <- tibble::tibble(
      Name = c("Variables", "Z-matrix Dimensions"),
      Value = c(
        paste(continuous_vars, collapse = ", "),
        paste(dim(result$z_matrix), collapse = " x ")
      )
    )

  } else if (model_type == "normal" || model_type == "t") {

    mean_x <- mean(data[[continuous_vars[2]]], na.rm = TRUE)
    sd_x <- sd(data[[continuous_vars[2]]], na.rm = TRUE)
    mean_y <- mean(data[[continuous_vars[1]]], na.rm = TRUE)
    sd_y <- sd(data[[continuous_vars[1]]], na.rm = TRUE)
    cor_val <- cor(data[[continuous_vars[2]]],
data[[continuous_vars[1]]], method = "pearson", use =
"complete.obs")

    x_vals <- seq(min(data[[continuous_vars[2]]], na.rm = TRUE),
max(data[[continuous_vars[2]]], na.rm = TRUE), length.out = 100)
    y_vals <- seq(min(data[[continuous_vars[1]]], na.rm = TRUE),
max(data[[continuous_vars[1]]], na.rm = TRUE), length.out = 100)
    grid <- expand.grid(x = x_vals, y = y_vals)

    if (model_type == "normal") {
      rho <- cor_val
      bivariate_density <- function(x, y) {
        z_x <- (x - mean_x) / sd_x
        z_y <- (y - mean_y) / sd_y
        exponent <- -1 / (2 * (1 - rho^2)) * (z_x^2 + z_y^2 - 2 *
rho * z_x * z_y)
        (1 / (2 * pi * sd_x * sd_y * sqrt(1 - rho^2))) *
exp(exponent)
      }
    } else { # model_type == "t"
      Sigma <- matrix(c(sd_x^2, cor_val * sd_x * sd_y,
                        cor_val * sd_x * sd_y, sd_y^2), nrow = 2)
      Sigma_inv <- solve(Sigma)
      det_Sigma <- det(Sigma)
      df <- nrow(data) - 1
      bivariate_density <- function(x, y) {
        z <- c(x - mean_x, y - mean_y)
        quad_form <- t(z) %*% Sigma_inv %*% z
        coeff <- gamma((df + 2) / 2) / (gamma(df / 2) * (df * pi)
* sqrt(det_Sigma))
        exponent <- (1 + quad_form / df)^(-(df + 2) / 2)
        as.numeric(coeff * exponent)
      }
    }

    grid$z <- mapply(bivariate_density, grid$x, grid$y)
    z_matrix <- matrix(grid$z, nrow = 100, byrow = FALSE)

    result$x_vals <- x_vals
    result$y_vals <- y_vals
    result$z_matrix <- t(z_matrix)
    result$model_type <- model_type
    result$continuous_vars <- continuous_vars
    result$params <- list(mean_x = mean_x, sd_x = sd_x, mean_y =
mean_y, sd_y = sd_y, cor_val = cor_val, df = if (model_type ==
"t") df else NULL)

    # Summary
    names_vec <- if (model_type == "normal") {
```

```
      c(
        paste0("Mean (", continuous_vars[1], ")"),
        paste0("SD (", continuous_vars[1], ")"),
        paste0("Mean (", continuous_vars[2], ")"),
        paste0("SD (", continuous_vars[2], ")"),
        "Pearson correlation"
      )
    } else {
      c(
        paste0("Mean (", continuous_vars[1], ")"),
        paste0("SD (", continuous_vars[1], ")"),
        paste0("Mean (", continuous_vars[2], ")"),
        paste0("SD (", continuous_vars[2], ")"),
        "Pearson correlation",
        "Degrees of Freedom"
      )
    }

    values_vec <- if (model_type == "normal") {
      c(mean_x, sd_x, mean_y, sd_y, cor_val)
    } else {
      c(mean_x, sd_x, mean_y, sd_y, cor_val, df)
    }

    result$summary_info <- tibble::tibble(
      Name = names_vec,
      Value = values_vec
    )
  }

  result$vector_type <- "continuous"
  return(result)
}

render_continuous_density <- function(model_output, data,
plot_type = "2D") {

  x_vals <- model_output$x_vals
  y_vals <- model_output$y_vals
  z_matrix <- model_output$z_matrix
  vars <- model_output$continuous_vars

  if (plot_type == "3D") {
    plot <- plot_ly(
      x = ~x_vals, y = ~y_vals, z = ~z_matrix,
      type = "surface",
      colors = colorRamp(c("blue", "cyan", "yellow", "red")),
      opacity = 0.7,
      showscale = TRUE
    ) %>%
      layout(
        title = paste("Združená hustota", vars[1], "a", vars[2],
ifelse(model_output$model_type == "kernel", "(jadrové
vyhladzovanie)", ifelse(model_output$model_type == "normal",
"(bivariátne normálne)", "(bivariátne t-rozdelenie)"))),
        scene = list(
          xaxis = list(title = vars[1]),
          yaxis = list(title = vars[2]),
          zaxis = list(title = "Hustota")
        )
      ) %>% event_register("plotly_click")

  } else { # 2D

    contour_df <- expand.grid(x = x_vals, y = y_vals)
    contour_df$z <- as.vector(t(z_matrix))

    scatter_plot <- ggplot(data, aes_string(x = vars[1], y =
vars[2])) +
      geom_point(size = 3, alpha = 0.7, aes_string(color =
vars[2])) +
      geom_contour(data = contour_df, aes(x = x, y = y, z = z),
color = "black") +
      labs(
        x = vars[1],
        y = vars[2],
        title = paste("Scatter plot s vrstevnicami",
ifelse(model_output$model_type == "kernel", "(jadrové
vyhladzovanie)", ifelse(model_output$model_type == "normal",
"(bivariátne normálne)", "(bivariátne t-rozdelenie)"))
      ) +
      scale_color_gradient(low = "blue", high = "red") +
      guides(color = "none") +
      theme_minimal()

    if (model_output$model_type == "t") {
      mean_x <- model_output$params$mean_x
      sd_x <- model_output$params$sd_x
      mean_y <- model_output$params$mean_y
      sd_y <- model_output$params$sd_y
      df <- model_output$params$df

      density_x <- ggplot(data, aes_string(x = vars[1])) +
        stat_function(fun = function(x) dt((x - mean_x) / sd_x,
df = df) / sd_x, fill = "blue", geom = "area", alpha = 0.5) +
        labs(x = NULL, y = paste("Hustota (", vars[1], ")", sep =
"")) +
        theme_minimal() +
        theme(axis.text.x = element_blank(), axis.ticks.x =
element_blank())

      density_y <- ggplot(data, aes_string(x = vars[2])) +
        stat_function(fun = function(y) dt((y - mean_y) / sd_y,
df = df) / sd_y, fill = "red", geom = "area", alpha = 0.5) +
        coord_flip() +
        labs(x = NULL, y = paste("Hustota (", vars[2], ")", sep =
"")) +
        theme_minimal() +
        theme(axis.text.y = element_blank(), axis.ticks.y =
element_blank())

    } else {
      density_x <- ggplot(data, aes_string(x = vars[1])) +
        geom_density(fill = "blue", alpha = 0.5) +
        labs(x = NULL, y = paste("Hustota (", vars[1], ")", sep =
"")) +
        theme_minimal() +
        theme(axis.text.x = element_blank(), axis.ticks.x =
element_blank())

      density_y <- ggplot(data, aes_string(x = vars[2])) +
        geom_density(fill = "red", alpha = 0.5) +
        coord_flip() +
        labs(x = NULL, y = paste("Hustota (", vars[2], ")", sep =
"")) +
        theme_minimal() +
        theme(axis.text.y = element_blank(), axis.ticks.y =
element_blank())
    }

    plot <- (density_x + plot_spacer()) /
      (scatter_plot + density_y) +
      patchwork::plot_layout(widths = c(4, 1), heights = c(1, 4))
  }

  # Summary
  summary_table <- gt::gt(model_output$summary_info) %>%
    gt::tab_header(
      title = gt::md("**Model Summary**")
    ) %>%
    gt::opt_row_striping() %>%
    gt::opt_table_font(font = "Arial") %>%
    standardize_gt_table()

  summary_table <- apply_dark_gt_theme(
    gt_tbl = summary_table,
    highlight_rows = NULL,
    highlight_color = NULL
  )

  return(list(
    plot = plot,
    summary = summary_table
  ))
}

model_continuous_density_copula <- function(data,
continuous_vars, model_type = "nonparametric", copula_type =
"empirical (beta)", marginal_densities = NULL) {

  # Modelovanie zdruzenej hustoty cez kopulovu funkciu a
marginalne rozdelenie (vsetko neparametricke)
  if (model_type == "nonparametric") {
    kde_x <- density(data[[continuous_vars[1]]], n = 512)
    kde_y <- density(data[[continuous_vars[2]]], n = 512)

    bw_x <- kde_x$bw
    bw_y <- kde_y$bw

    kde_x_density <- approxfun(kde_x$x, kde_x$y, rule = 2)
    kde_x_cdf <- approxfun(kde_x$x, cumsum(kde_x$y) /
sum(kde_x$y), rule = 2)

    kde_y_density <- approxfun(kde_y$x, kde_y$y, rule = 2)
    kde_y_cdf <- approxfun(kde_y$x, cumsum(kde_y$y) /
sum(kde_y$y), rule = 2)

    u1 <- kde_x_cdf(data[[continuous_vars[1]]])
    u2 <- kde_y_cdf(data[[continuous_vars[2]]])
    empirical_data <- pobs(cbind(u1, u2))

    if (copula_type == "empirical (beta)") {
      copula_model_fitted <- empCopula(empirical_data, smoothing
= "beta")
```

```
    } else {
      stop("Unsupported copula_type. Only 'empirical (beta)' is
allowed here.")
    }

    x_vals <- seq(min(data[[continuous_vars[1]]], na.rm = TRUE),
max(data[[continuous_vars[1]]], na.rm = TRUE), length.out = 100)
    y_vals <- seq(min(data[[continuous_vars[2]]], na.rm = TRUE),
max(data[[continuous_vars[2]]], na.rm = TRUE), length.out = 100)
    grid <- expand.grid(x = x_vals, y = y_vals)

    copula_density_function <- function(x, y) {

      u1 <- kde_x_cdf(x)
      u2 <- kde_y_cdf(y)
      copula_density <- dCopula(cbind(u1, u2), copula =
copula_model_fitted)
      marginal_x <- kde_x_density(x)
      marginal_y <- kde_y_density(y)

      copula_density * marginal_x * marginal_y
    }

    grid$z <- mapply(copula_density_function, grid$x, grid$y)
    z_matrix <- matrix(grid$z, nrow = 100, byrow = FALSE)

    return(list(
      x_vals = x_vals,
      y_vals = y_vals,
      z_matrix = t(z_matrix),
      copula_type = copula_type,
      model_type = model_type,
      marginal_densities = c("KDE", "KDE"),
      continuous_vars = continuous_vars,
      bw_x = bw_x,
      bw_y = bw_y,
      vector_type = "continuous_copula"
    ))
  }

  # Modelovanie zdruzenej hustoty cez kopulovu funkciu a
marginalne rozdelenie (vsetko parametricke)
  if (model_type == "parametric") {

    mean_x <- mean(data[[continuous_vars[1]]], na.rm = TRUE)
    sd_x <- sd(data[[continuous_vars[1]]], na.rm = TRUE)
    mean_y <- mean(data[[continuous_vars[2]]], na.rm = TRUE)
    sd_y <- sd(data[[continuous_vars[2]]], na.rm = TRUE)

    marginal_cdf_function <- function(value, mean, sd, index) {
      density_type <- marginal_densities[index]
      if (density_type == "normal" || density_type ==
"log_normal") {
        return(pnorm(value, mean = mean, sd = sd))
      } else if (density_type == "t") {
        df <- max(nrow(data) - 1, 2)
        return(pt((value - mean) / sd, df = df))
      } else {
        stop(paste("Neznáma hustota:", density_type))
      }
    }

    u1 <- marginal_cdf_function(data[[continuous_vars[1]]],
mean_x, sd_x, 1)
    u2 <- marginal_cdf_function(data[[continuous_vars[2]]],
mean_y, sd_y, 2)

    copula_model <- switch(
      copula_type,
      "Clayton" = claytonCopula(param = 2, dim = 2),
      "Gumbel"  = gumbelCopula(param = 2, dim = 2),
      "Frank"   = frankCopula(param = 5, dim = 2),
      "Joe"     = joeCopula(param = 2, dim = 2),
      "t"       = tCopula(dim = 2, df = 4, df.fixed = FALSE),
      stop("Zadaný 'copula_type' nie je podporovaný. Použi:
'Clayton', 'Gumbel', 'Frank', 'Joe', 't'.")
    )

    copula_fit <- fitCopula(copula_model, pobs(cbind(u1, u2)),
method = "ml")
    copula_model_fitted <- copula_fit@copula

    copula_params <- copula_fit@estimate
    rho_fitted <- copula_params[1]
    df_fitted <- if (inherits(copula_model_fitted, "tCopula"))
copula_params[2] else NA

    x_vals <- seq(min(data[[continuous_vars[1]]], na.rm = TRUE),
max(data[[continuous_vars[1]]], na.rm = TRUE), length.out = 100)
    y_vals <- seq(min(data[[continuous_vars[2]]], na.rm = TRUE),
max(data[[continuous_vars[2]]], na.rm = TRUE), length.out = 100)
    grid <- expand.grid(x = x_vals, y = y_vals)
```

```
    marginal_density_function <- function(value, mean, sd, index)
{
      density_type <- marginal_densities[index]
      if (density_type == "normal") {
        return(dnorm(value, mean = mean, sd = sd))
      } else if (density_type == "log_normal") {
        return(exp(dnorm(value, mean = mean, sd = sd, log =
TRUE)))
      } else if (density_type == "t") {
        df <- max(nrow(data) - 1, 2)
        return(dt((value - mean) / sd, df = df) / sd)
      } else {
        stop(paste("Neznáma hustota:", density_type))
      }
    }

    copula_density_function <- function(x, y) {
      u1 <- marginal_cdf_function(x, mean_x, sd_x, 1)
      u2 <- marginal_cdf_function(y, mean_y, sd_y, 2)
      copula_part <- dCopula(cbind(u1, u2), copula =
copula_model_fitted)
      marginal_x <- marginal_density_function(x, mean_x, sd_x, 1)
      marginal_y <- marginal_density_function(y, mean_y, sd_y, 2)
      copula_part * marginal_x * marginal_y
    }

    grid$z <- mapply(copula_density_function, grid$x, grid$y)
    z_matrix <- matrix(grid$z, nrow = 100, byrow = FALSE)

    return(list(
      x_vals = x_vals,
      y_vals = y_vals,
      z_matrix = z_matrix,
      mean_x = mean_x,
      sd_x = sd_x,
      mean_y = mean_y,
      sd_y = sd_y,
      copula_model_fitted = copula_model_fitted,
      continuous_vars = continuous_vars,
      marginal_densities = marginal_densities,
      copula_type = copula_type,
      model_type = model_type,
      rho_copula = rho_fitted,
      df_copula = df_fitted,
      vector_type = "continuous_copula"
    ))
  }

  # Modelovanie zdruzenej hustoty cez kopulovu funkciu a
marginalne rozdelenie (Lubovolny vyber)
  if (model_type == "hybrid") {

    mean_x <- mean(data[[continuous_vars[1]]], na.rm = TRUE)
    sd_x <- sd(data[[continuous_vars[1]]], na.rm = TRUE)
    mean_y <- mean(data[[continuous_vars[2]]], na.rm = TRUE)
    sd_y <- sd(data[[continuous_vars[2]]], na.rm = TRUE)

    marginal_cdf_function <- function(value, mean, sd, index) {
      density_type <- marginal_densities[index]
      if (density_type == "normal" || density_type ==
"log_normal") {
        return(pnorm(value, mean = mean, sd = sd))
      } else if (density_type == "t") {
        df <- max(nrow(data) - 1, 2)
        return(pt((value - mean) / sd, df = df))
      } else if (density_type == "KDE") {
        dens <- density(data[[continuous_vars[index]]], n = 512)
        approx_fun <- approxfun(dens$x,
cumsum(dens$y)/sum(dens$y), rule = 2)
        return(approx_fun(value))
      } else {
        stop(paste("Neznáma hustota:", density_type))
      }
    }

    bw_x <- if (marginal_densities[1] == "KDE") {
      density(data[[continuous_vars[1]]], n = 512)$bw
    } else {
      NA_real_
    }
    bw_y <- if (marginal_densities[2] == "KDE") {
      density(data[[continuous_vars[2]]], n = 512)$bw
    } else {
      NA_real_
    }

    u1 <- marginal_cdf_function(data[[continuous_vars[1]]],
mean_x, sd_x, 1)
    u2 <- marginal_cdf_function(data[[continuous_vars[2]]],
mean_y, sd_y, 2)
```

```
    if (copula_type == "empirical (beta)") {
      copula_model_fitted <- empCopula(pobs(cbind(u1, u2)),
smoothing = "beta")
      rho_fitted <- NA
      df_fitted <- NA
    } else {
      copula_model <- switch(copula_type,
                             "Clayton" = claytonCopula(param = 2,
dim = 2),
                             "Gumbel" = gumbelCopula(param = 2,
dim = 2),
                             "Frank" = frankCopula(param = 5, dim
= 2),
                             "Joe" = joeCopula(param = 2, dim =
2),
                             "t" = tCopula(dim = 2, df = 4,
df.fixed = FALSE),
                             NULL
      )

      if (is.null(copula_model)) {
        stop("Neznámy typ kopuly.")
      }

      copula_fit <- fitCopula(copula_model, pobs(cbind(u1, u2)),
method = "ml")
      copula_model_fitted <- copula_fit@copula

      copula_params <- copula_fit@estimate
      rho_fitted <- copula_params[1]
      df_fitted <- if (inherits(copula_model_fitted, "tCopula"))
copula_params[2] else NA
    }

    marginal_density_function <- function(value, mean, sd, index)
{
      density_type <- marginal_densities[index]
      if (density_type == "normal") {
        return(dnorm(value, mean = mean, sd = sd))
      } else if (density_type == "log_normal") {
        return(exp(dnorm(value, mean = mean, sd = sd, log =
TRUE)))
      } else if (density_type == "t") {
        df <- max(nrow(data) - 1, 2)
        return(dt((value - mean) / sd, df = df) / sd)
      } else if (density_type == "KDE") {
        dens <- density(data[[continuous_vars[index]]], n = 512)
        approx_fun <- approxfun(dens$x, dens$y, rule = 2)
        return(approx_fun(value))
      } else {
        stop(paste("Neznáma hustota:", density_type))
      }
    }

    x_vals <- seq(min(data[[continuous_vars[1]]], na.rm = TRUE),
max(data[[continuous_vars[1]]], na.rm = TRUE), length.out = 100)
    y_vals <- seq(min(data[[continuous_vars[2]]], na.rm = TRUE),
max(data[[continuous_vars[2]]], na.rm = TRUE), length.out = 100)

    grid <- expand.grid(x = x_vals, y = y_vals)

    copula_density_function <- function(x, y) {
      u1 <- marginal_cdf_function(x, mean_x, sd_x, 1)
      u2 <- marginal_cdf_function(y, mean_y, sd_y, 2)
      copula_part <- dCopula(cbind(u1, u2), copula =
copula_model_fitted)
      marginal_x <- marginal_density_function(x, mean_x, sd_x, 1)
      marginal_y <- marginal_density_function(y, mean_y, sd_y, 2)
      copula_part * marginal_x * marginal_y
    }

    fx_vals <- sapply(y_vals, function(xi)
marginal_density_function(xi, mean_y, sd_y, 2))

    grid$z <- mapply(copula_density_function, grid$x, grid$y)
    #z_matrix <- matrix(grid$z, nrow = 100, ncol = 100, byrow =
FALSE)
    z_matrix <- matrix(grid$z, nrow = 100, ncol = 100, byrow =
FALSE)

    dx <- diff(x_vals[1:2])
    dy <- diff(y_vals[1:2])
    area_total <- sum(z_matrix) * dx * dy

    if (!is.na(area_total) && area_total > 0 && abs(area_total -
1) > 0.01) {
      z_matrix <- z_matrix / area_total
      message(sprintf("Znormalizovaná hustota – pôvodná plocha
bola %.8f", area_total))
    }

    df_value <- tryCatch({
      if (inherits(copula_model_fitted, "tCopula")) {
        copula_model_fitted@df
      } else {
        NA
      }
    }, error = function(e) NA)

    return(list(
      x_vals = x_vals,
      y_vals = y_vals,
      z_matrix = z_matrix,
      fx_vals = fx_vals,
      continuous_vars = continuous_vars,
      copula_type = copula_type,
      marginal_densities = marginal_densities,
      mean_x = mean_x,
      sd_x = sd_x,
      mean_y = mean_y,
      sd_y = sd_y,
      bw_x = bw_x,
      bw_y = bw_y,
      copula_model_fitted = copula_model_fitted,
      rho_copula = rho_fitted,
      df_copula = df_fitted,
      vector_type = "continuous_copula"
    ))

  }
}

render_continuous_density_copula <- function(model_output, data,
model_type = "nonparametric", plot_type = "2D") {

  x_vals <- model_output$x_vals
  y_vals <- model_output$y_vals
  z_matrix <- model_output$z_matrix
  continuous_vars <- model_output$continuous_vars
  copula_type <- model_output$copula_type
  marginal_densities <- model_output$marginal_densities

  if (is.null(model_type)){
    stop("Chýba typ modelu pre kopulové modelovanie.")
  }

  if (model_type == "nonparametric") {
    if (plot_type == "3D") {
    plot <- plot_ly(
      x = ~x_vals, y = ~y_vals, z = ~z_matrix,
      type = "surface",
      colors = colorRamp(c("blue", "cyan", "yellow", "red")),
      opacity = 0.7,
      showscale = TRUE
    ) %>% layout(
      title = paste0("Združená hustota ", continuous_vars[1], " a
", continuous_vars[2], " (KDE + ", copula_type, " copula)"),
      scene = list(
        xaxis = list(title = continuous_vars[1]),
        yaxis = list(title = continuous_vars[2]),
        zaxis = list(title = "Hustota")
      )
    )

  } else { # plot_type == "2D"
    contour_df <- expand.grid(x = x_vals, y = y_vals)
    contour_df$z <- as.vector(t(z_matrix))

    scatter_plot <- ggplot(data, aes_string(x =
continuous_vars[1], y = continuous_vars[2])) +
      geom_point(size = 3, alpha = 0.7, aes_string(color =
continuous_vars[2])) +
      geom_contour(data = contour_df, aes(x = x, y = y, z = z),
color = "black", size = 0.6) +
      labs(
        x = continuous_vars[1],
        y = continuous_vars[2],
        title = paste("Scatter plot s vrstevnicami (",
marginal_densities[1], " +", copula_type, " kopula)")
      ) +
      scale_color_gradient(low = "blue", high = "red") +
      theme_minimal()

    density_x <- ggplot(data, aes_string(x = continuous_vars[1]))
+
      geom_density(fill = "blue", alpha = 0.5) +
      labs(x = NULL, y = paste("Hustota (", continuous_vars[1],
")", sep = "")) +
      theme_minimal() +
      theme(axis.text.x = element_blank(), axis.ticks.x =
element_blank())

    density_y <- ggplot(data, aes_string(x = continuous_vars[2]))
+
      geom_density(fill = "red", alpha = 0.5) +
```

```
      coord_flip() +
      labs(x = NULL, y = paste("Hustota (", continuous_vars[2],
")", sep = "")) +
      theme_minimal() +
      theme(axis.text.y = element_blank(), axis.ticks.y =
element_blank())

    plot <- (density_x + plot_spacer()) /
      (scatter_plot + density_y) +
      patchwork::plot_layout(widths = c(4, 1), heights = c(1, 4))
  }

    bw_x <- round(model_output$bw_x, 4)
    bw_y <- round(model_output$bw_y, 4)

    summary_tbl <- tibble::tibble(
      Name = c(
        paste0("Marginal Density X (", continuous_vars[1], ")"),
        paste0("Bandwidth X (", continuous_vars[1], ")"),
        paste0("Marginal Density Y (", continuous_vars[2], ")"),
        paste0("Bandwidth Y (", continuous_vars[2], ")"),
        "Copula Type",
        "Smoothing Method",
        "Copula Model"
      ),
      Value = c(
        marginal_densities[1], bw_x,
        marginal_densities[2], bw_y,
        copula_type, "beta", "empCopula (empirical)"
      )
    )

    summary_table <- gt::gt(summary_tbl) %>%
      gt::tab_header(
        title = gt::md("**Model Summary**")
      ) %>%
      gt::cols_width(Name ~ px(260), Value ~ px(460)) %>%
      gt::opt_row_striping() %>%
      gt::opt_table_font(font = "Arial") %>%
      standardize_gt_table()

    summary_table <- apply_dark_gt_theme(
      gt_tbl = summary_table,
      highlight_rows = NULL,
      highlight_color = NULL
    )

    return(list(
      plot = plot,
      summary = summary_table
    ))
  }

  if (model_type == "parametric") {

    if (plot_type == "3D") {
      plot <- plot_ly(
        x = ~x_vals, y = ~y_vals, z = ~z_matrix,
        type = "surface",
        opacity = 0.7,
        colors = colorRamp(c("blue", "cyan", "yellow", "red")),
        showscale = TRUE
      ) %>%
        layout(
          title = paste0("Združená hustota ", continuous_vars[1],
" a ", continuous_vars[2], " (", marginal_densities[1], " + ",
copula_type, " kopula)"),
          scene = list(
            xaxis = list(title = continuous_vars[1]),
            yaxis = list(title = continuous_vars[2]),
            zaxis = list(title = "Hustota")
          )
        )

    } else { # plot_type == "2D"

      contour_df <- expand.grid(x = x_vals, y = y_vals)
      contour_df$z <- as.vector(t(z_matrix))

      scatter_plot <- ggplot(data, aes_string(x =
continuous_vars[1], y = continuous_vars[2])) +
        geom_point(size = 3, alpha = 0.7, aes_string(color =
continuous_vars[2])) +
        geom_contour(data = contour_df, aes(x = x, y = y, z = z),
color = "black", bins = 10) +
        labs(
          x = continuous_vars[1],
          y = continuous_vars[2],
          title = paste("Scatter plot s vrstevnicami (",
marginal_densities[1], " +", copula_type, " kopula)")
        ) +
        scale_color_gradient(low = "blue", high = "red") +
        theme_minimal()

      density_x <- ggplot(data, aes_string(x =
continuous_vars[1])) +
        stat_function(fun = dnorm, args = list(mean =
model_output$mean_x, sd = model_output$sd_x), fill = "blue", geom
= "area", alpha = 0.5) +
        labs(x = NULL, y = paste("Hustota (", continuous_vars[1],
")")) +
        theme_minimal() +
        theme(axis.text.x = element_blank(), axis.ticks.x =
element_blank())

      density_y <- ggplot(data, aes_string(x =
continuous_vars[2])) +
        stat_function(fun = dnorm, args = list(mean =
model_output$mean_y, sd = model_output$sd_y), fill = "red", geom
= "area", alpha = 0.5) +
        coord_flip() +
        labs(x = NULL, y = paste("Hustota (", continuous_vars[2],
")")) +
        theme_minimal() +
        theme(axis.text.y = element_blank(), axis.ticks.y =
element_blank())

      plot <- (density_x + patchwork::plot_spacer()) /
        (scatter_plot + density_y) +
        patchwork::plot_layout(widths = c(4, 1), heights = c(1,
4))
    }

    copula_rho <- round(model_output$rho_copula, 4)
    copula_df <- if (!is.na(model_output$df_copula))
round(model_output$df_copula, 4) else NA

    summary_tbl <- tibble::tibble(
      Name = c(
        paste0("Marginal Model X (", continuous_vars[1], ")"),
        paste0("Mean (", continuous_vars[1], ")"),
        paste0("SD (", continuous_vars[1], ")"),
        paste0("Marginal Model Y (", continuous_vars[2], ")"),
        paste0("Mean (", continuous_vars[2], ")"),
        paste0("SD (", continuous_vars[2], ")"),
        "Copula Type",
        "Fitted Copula Family",
        "Fitted Copula Rho"
      ),
      Value = as.character(c(
        marginal_densities[1],
        round(model_output$mean_x, 4),
        round(model_output$sd_x, 4),
        marginal_densities[2],
        round(model_output$mean_y, 4),
        round(model_output$sd_y, 4),
        copula_type,
        class(model_output$copula_model_fitted),
        copula_rho
      ))
    )

    if (!is.na(copula_df)) {
      summary_tbl <- dplyr::add_row(
        summary_tbl,
        Name = "Degrees of Freedom (t-copula)",
        Value = as.character(copula_df)
      )
    }

    summary_table <- gt::gt(summary_tbl) %>%
      gt::tab_header(
        title = gt::md("**Model Summary**")
      ) %>%
      gt::cols_width(Name ~ px(260), Value ~ px(460)) %>%
      gt::opt_row_striping() %>%
      gt::opt_table_font(font = "Arial") %>%
      standardize_gt_table()

    summary_table <- apply_dark_gt_theme(
      gt_tbl = summary_table,
      highlight_rows = NULL,
      highlight_color = NULL
    )

    return(list(
      plot = plot,
      summary = summary_table
    ))
  }

  if (model_type == "hybrid") {

    if (plot_type == "3D") {
```

```
      plot <- plot_ly(
        x = ~x_vals, y = ~y_vals, z = ~z_matrix,
        type = "surface",
        opacity = 0.7,
        colors = colorRamp(c("blue", "cyan", "yellow", "red")),
        showscale = TRUE
      ) %>%
        layout(
          title = paste0("Združená hustota ", continuous_vars[1],
" a ", continuous_vars[2], " (hybridné marginály + ",
copula_type, " kopula)"),
          scene = list(
            xaxis = list(title = continuous_vars[1]),
            yaxis = list(title = continuous_vars[2]),
            zaxis = list(title = "Hustota")
          )
        )
    } else {
      contour_df <- expand.grid(x = x_vals, y = y_vals)
      contour_df$z <- as.vector(t(z_matrix))

      scatter_plot <- ggplot(data, aes_string(x =
continuous_vars[1], y = continuous_vars[2])) +
        geom_point(size = 3, alpha = 0.7, aes_string(color =
continuous_vars[2])) +
        geom_contour(data = contour_df, aes(x = x, y = y, z = z),
color = "black", size = 0.6) +
        labs(
          x = continuous_vars[1],
          y = continuous_vars[2],
          title = paste("Scatter + vrstevnice (hybrid +",
copula_type, "kopula)")
        ) +
        scale_color_gradient(low = "blue", high = "red") +
        theme_minimal()

      density_x <- ggplot(data, aes_string(x =
continuous_vars[1])) +
        geom_density(fill = "blue", alpha = 0.5) +
        labs(x = NULL, y = paste("Hustota (", continuous_vars[1],
")")) +
        theme_minimal() +
        theme(axis.text.x = element_blank(), axis.ticks.x =
element_blank())

      density_y <- ggplot(data, aes_string(x =
continuous_vars[2])) +
        geom_density(fill = "red", alpha = 0.5) +
        coord_flip() +
        labs(x = NULL, y = paste("Hustota (", continuous_vars[2],
")")) +
        theme_minimal() +
        theme(axis.text.y = element_blank(), axis.ticks.y =
element_blank())

      plot <- (density_x + plot_spacer()) / (scatter_plot +
density_y) +
        patchwork::plot_layout(widths = c(4, 1), heights = c(1,
4))
    }

    ## Summary Table:
    marginal_summary <- function(var_name, density_type,
mean_val, sd_val, bw_val, index) {
      df <- max(nrow(data) - 1, 2)
      if (density_type == "normal") {
        return(tibble::tibble(
          Component = paste("Marginal", var_name),
          Type = "Normal",
          Parameters = paste0("Mean = ", round(mean_val, 4), ";
SD = ", round(sd_val, 4))
        ))
      } else if (density_type == "log_normal") {
        return(tibble::tibble(
          Component = paste("Marginal", var_name),
          Type = "Log-Normal",
          Parameters = paste0("Mean = ", round(mean_val, 4), ";
SD = ", round(sd_val, 4))
        ))
      } else if (density_type == "t") {
        return(tibble::tibble(
          Component = paste("Marginal", var_name),
          Type = "Student-t",
          Parameters = paste0("Mean = ", round(mean_val, 4), ";
SD = ", round(sd_val, 4), "; df = ", df)
        ))
      } else if (density_type == "KDE") {
        return(tibble::tibble(
          Component = paste("Marginal", var_name),
          Type = "Kernel Density Estimate",
          Parameters = paste0("bw = ", round(bw_val, 4))
        ))
      } else {
        return(tibble::tibble(
          Component = paste("Marginal", var_name),
          Type = density_type,
          Parameters = "Unknown"
        ))
      }
    }

    marg1 <- marginal_summary(continuous_vars[1],
marginal_densities[1], model_output$mean_x, model_output$sd_x,
model_output$bw_x, 1)
    marg2 <- marginal_summary(continuous_vars[2],
marginal_densities[2], model_output$mean_y, model_output$sd_y,
model_output$bw_y, 2)

    copula_parameters <- tryCatch({
      if (tolower(model_output$copula_type) == "t") {
        paste0("rho = ", round(model_output$rho_copula, 4),
               "; df = ", round(model_output$df_copula, 4))
      } else {
        sub(".*Parameters:\\s*", "",

paste(capture.output(show(model_output$copula_model_fitted)),
collapse = "<br>")
        )
      }
    }, error = function(e) {
      "Empirical copula (no parameters)"
    })

    copula_summary <- tibble::tibble(
      Component = "Copula",
      Type = model_output$copula_type,
      Parameters = copula_parameters
    )

    summary_tbl <- dplyr::bind_rows(marg1, marg2, copula_summary)

    summary_table <- gt::gt(summary_tbl) %>%
      gt::tab_header(
        title = gt::md("**Model Summary**"),
        subtitle = "Hybrid Copula Model"
      ) %>%
      gt::cols_width(Component ~ px(200), Type ~ px(200),
Parameters ~ px(420)) %>%
      gt::fmt_markdown(columns = "Parameters") %>%
      gt::opt_row_striping() %>%
      gt::opt_table_font(font = "Arial") %>%
      standardize_gt_table()

    summary_table <- apply_dark_gt_theme(
      gt_tbl = summary_table,
      highlight_rows = NULL,
      highlight_color = NULL
    )

    return(list(
      plot = plot,
      summary = summary_table
    ))
  }
}

model_joint_pmf <- function(data, discrete_vars) {

  tab <- as.data.frame(table(data[, discrete_vars]))
  tab$Probability <- tab$Freq / sum(tab$Freq)

  tab[[discrete_vars[1]]] <- factor(tab[[discrete_vars[1]]])
  tab[[discrete_vars[2]]] <- factor(tab[[discrete_vars[2]]])

  tab$x_pos <- as.numeric(tab[[discrete_vars[1]]])
  tab$y_pos <- as.numeric(tab[[discrete_vars[2]]])

  x_labels <- levels(tab[[discrete_vars[1]]])
  y_labels <- levels(tab[[discrete_vars[2]]])

  total_prob <- sum(tab$Probability)

  joint_values_lines <- paste(
    apply(tab[, discrete_vars], 1, paste, collapse = " | "),
    ": ",
    round(tab$Probability, 4)
  )

  joint_values_md <- paste0("```\n", paste(joint_values_lines,
collapse = "\n"), "\n```")

  summary_tbl <- tibble::tibble(
    Name = c(
      "Model Type",
```

```
      "Variable X", "Variable Y",
      "Levels in X", "Levels in Y",
      "Number of States (X,Y)",
      "PMF Values",
      "Total Probability"
    ),
    Value = c(
      "discrete",
      discrete_vars[1],
      discrete_vars[2],
      length(x_labels),
      length(y_labels),
      nrow(tab),
      joint_values_md,
      round(total_prob, 4)
    )
  )

  base_table <- summary_tbl %>%
    gt::gt() %>%
    gt::tab_header(
      title = gt::md("**Model Summary**"),
      subtitle = "Probability Mass Function"
    ) %>%
    gt::cols_width(Name ~ px(200), Value ~ px(500)) %>%
    gt::fmt_markdown(columns = "Value") %>%
    gt::opt_row_striping() %>%
    gt::opt_table_font(font = "Arial") %>%
    standardize_gt_table()

  summary_table_gt <- apply_dark_gt_theme(
    gt_tbl = base_table,
    highlight_rows = NULL,
    highlight_color = NULL
  )

  return(list(
    tab = tab,
    x_labels = x_labels,
    y_labels = y_labels,
    discrete_vars = discrete_vars,
    summary = summary_table_gt,
    vector_type = "discrete"
  ))
}

render_joint_pmf <- function(model_output, plot_type = "2D") {

  tab <- model_output$tab
  x_labels <- model_output$x_labels
  y_labels <- model_output$y_labels
  discrete_vars <- model_output$discrete_vars

  if (plot_type == "2D") {

    marginal_x <- tab %>%
      group_by(!!sym(discrete_vars[1])) %>%
      summarise(Prob_X = sum(Probability))

    marginal_y <- tab %>%
      group_by(!!sym(discrete_vars[2])) %>%
      summarise(Prob_Y = sum(Probability))

    scatter_plot <- ggplot(tab, aes_string(x = discrete_vars[1],
y = discrete_vars[2], fill = "Probability")) +
      geom_tile(color = "white") +
      scale_fill_gradient(low = "blue", high = "red") +
      labs(x = discrete_vars[1], y = discrete_vars[2], fill =
"Pravdepodobnosť") +
      guides(fill = "none") +
      theme_minimal()

    marginal_x_plot <- ggplot(marginal_x, aes_string(x =
discrete_vars[1], y = "Prob_X")) +
      geom_bar(stat = "identity", fill = "blue", alpha = 0.6) +
      labs(x = NULL, y = "P(X)") +
      theme_minimal()

    marginal_y_plot <- ggplot(marginal_y, aes_string(x =
discrete_vars[2], y = "Prob_Y")) +
      geom_bar(stat = "identity", fill = "red", alpha = 0.6) +
      coord_flip() +
      labs(x = NULL, y = "P(Y)") +
      theme_minimal()

    plot <- (marginal_x_plot + patchwork::plot_spacer()) /
      (scatter_plot + marginal_y_plot) +
      patchwork::plot_layout(widths = c(4, 1), heights = c(1, 4))

  } else if (plot_type == "3D") {

    fig_3d <- plot_ly()

    for (i in 1:nrow(tab)) {
      fig_3d <- fig_3d %>% add_trace(
        x = rep(tab$x_pos[i], 2),
        y = rep(tab$y_pos[i], 2),
        z = c(0, tab$Probability[i]),
        type = "scatter3d",
        mode = "lines",
        line = list(color = "blue"),
        showlegend = FALSE
      )
    }

    fig_3d <- fig_3d %>% add_trace(
      x = tab$x_pos,
      y = tab$y_pos,
      z = tab$Probability,
      type = "scatter3d",
      mode = "markers",
      marker = list(size = 5, color = "red"),
      name = "Pravdepodobnosti"
    )

    plot <- fig_3d %>% layout(
      scene = list(
        xaxis = list(title = paste0(discrete_vars[1], " (X)"),
                     tickvals = 1:length(x_labels),
                     ticktext = x_labels),
        yaxis = list(title = paste0(discrete_vars[2], " (Y)"),
                     tickvals = 1:length(y_labels),
                     ticktext = y_labels),
        zaxis = list(title = "P(X, Y)")
      )
    )
  }

  list(
    plot = plot,
    summary = model_output$summary
  )
}

model_joint_distribution_density <- function(data,
selected_variables, model_type = NULL, bw = NULL, use_copula =
FALSE, copula_type = NULL, marginal_densities = c("dnorm",
"dnorm")) {

  # Identifikacia typov premennych
  variable_types <- identify_variables(data)
  discrete_vars <- intersect(variable_types$Diskretne,
selected_variables)
  continuous_vars <- intersect(variable_types$Spojite,
selected_variables)
  variables_count <- length(discrete_vars) +
length(continuous_vars)

  if (length(selected_variables) == 2 && length(continuous_vars)
== 0) {
    result <- model_joint_pmf(data, discrete_vars)
    return(result)
  } else if (length(selected_variables) == 2 &&
length(discrete_vars) == 0) {
    if (use_copula == FALSE) {
      result <- model_continuous_density(data, continuous_vars,
model_type)
      return(result)
    }
    else {
      if (is.null(copula_type)) {
        stop("Ak je 'use_copula' = 'TRUE', musi byt specifikovany
aj parameter 'copula_type'.")
      }
      else {
        result <- model_continuous_density_copula(data,
continuous_vars, model_type, copula_type, marginal_densities)
        return(result)
      }
    }
  } else if (length(selected_variables) == 2 &&
length(discrete_vars) == 1) {
    result <- model_mixture_density(data, discrete_vars,
continuous_vars, model_type, bw)
    return(result)
  }
}

render_joint_distribution_density <- function(model_output =
NULL, data = NULL, plot_type = NULL, model_type = NULL) {

  if (is.null(plot_type)) {
    stop("Nebol zadany typ grafu na vykreslenie.")
  }
```

```r
  if (is.null(model_output)) {
    stop("Neboli poskytnuté žiadne dáta na vizualizáciu.")
  }

  vector_type <- model_output$vector_type

  result <- switch(vector_type,
                   "mix" =
render_mixture_density(model_output, plot_type),
                   "continuous" =
render_continuous_density(model_output, data, plot_type),
                   "continuous_copula" =
render_continuous_density_copula(model_output, data, model_type,
plot_type),
                   "discrete" =
render_joint_pmf(model_output, plot_type),
                   NULL
  )

  if (is.null(result)) {
    stop("Neznámy typ vektora premenných.")
  }

  return(result)
}

model_conditional_mean <- function(data, selected_variables,
mean_method = "linear", poly_mean_degree = NULL, specific_x =
NULL) {
  response_name <- selected_variables[1]
  predictor_name <- selected_variables[2]

  response <- data[[response_name]]
  predictor <- data[[predictor_name]]
  x_seq <- seq(min(predictor), max(predictor), length.out = 200)

  fit <- NULL
  r_squared <- NULL
  mse <- NULL
  model_formula <- NULL
  param_summary <- NULL
  basis_function <- NA
  hyperparams <- NA

  # Fitovanie modelu podla zvolenej metody
  if (mean_method == "loess") {
    fit <- loess(response ~ predictor, span = 0.75)
    fitted_values <- predict(fit)
    ss_res <- sum((response - fitted_values)^2)
    ss_tot <- sum((response - mean(response))^2)
    r_squared <- 1 - ss_res / ss_tot
    mse <- mean((response - fitted_values)^2)
    basis_function <- "LOESS"
    hyperparams <- "span = 0.75"

  } else if (mean_method == "gam") {
    fit <- mgcv::gam(response ~ s(predictor))
    r_squared <- summary(fit)$r.sq
    mse <- mean((response - predict(fit))^2)
    basis_function <- "Spline smoother (s())"

  } else if (mean_method == "spline") {
    fit <- lm(response ~ bs(predictor, df = 5))
    summary_fit <- summary(fit)
    r_squared <- summary_fit$r.squared
    mse <- mean((response - predict(fit))^2)
    basis_function <- "B-spline"
    hyperparams <- "df = 5"
    param_summary <- broom::tidy(fit)

  } else if (mean_method == "linear") {
    fit <- lm(response ~ predictor)
    summary_fit <- summary(fit)
    r_squared <- summary_fit$r.squared
    mse <- mean((response - predict(fit))^2)
    basis_function <- "Linear"
    param_summary <- broom::tidy(fit)

  } else if (mean_method == "poly") {
    fit <- lm(response ~ poly(predictor, degree =
poly_mean_degree, raw = TRUE))
    summary_fit <- summary(fit)
    r_squared <- summary_fit$r.squared
    mse <- mean((response - predict(fit))^2)
    basis_function <- "Polynomial"
    hyperparams <- paste("degree =", poly_mean_degree)
    param_summary <- broom::tidy(fit)

  } else if (mean_method == "exp") {
    fit <- nls(response ~ a * exp(b * predictor), start = list(a
= 1, b = 0.01))
    fitted_values <- predict(fit)
    ss_res <- sum((response - fitted_values)^2)
    ss_tot <- sum((response - mean(response))^2)
    r_squared <- 1 - ss_res / ss_tot
    mse <- mean((response - fitted_values)^2)
    basis_function <- "Exponential"
    param_summary <- broom::tidy(fit)

  } else {
    stop("Neplatný mean_method!")
  }

  mean_pred <- predict(fit, newdata = data.frame(predictor =
x_seq))

  if (is.null(specific_x)) specific_x <- median(predictor, na.rm
= TRUE)
  specific_mean <- predict(fit, newdata = data.frame(predictor =
specific_x))

  # Vystupna tabulka
  summary_tbl <- tibble::tibble(
    Name = c(
      "Model Type", "Basis Function", "Hyperparameters",
      "R-squared", "MSE", "X (specific)", "E[Y|X = x]"
    ),
    Value = c(
      mean_method,
      basis_function,
      ifelse(is.na(hyperparams), "-", hyperparams),
      round(r_squared, 4),
      round(mse, 4),
      round(specific_x, 2),
      round(specific_mean, 2)
    )
  )

  if (!is.null(param_summary)) {
    param_tbl <- param_summary %>%
      dplyr::transmute(
        Name = paste0("β_", term),
        Value = paste0(round(estimate, 4), " ± ",
round(std.error, 4))
      )
    summary_tbl <- bind_rows(summary_tbl, param_tbl)
  }

  summary_table_gt <- apply_dark_gt_theme(
    gt::gt(summary_tbl) %>%
      gt::tab_header(
        title = gt::md(paste0("**Mean Model Summary**")),
        subtitle = "Conditional Mean Function"
      ) %>%
      gt::cols_width(Name ~ px(240), Value ~ px(460)) %>%
      gt::opt_row_striping() %>%
      gt::opt_table_font(font = "Arial") %>%
      standardize_gt_table()
  )

  return(list(
    conditional_mean = data.frame(X = x_seq, E_Y_given_X =
mean_pred),
    specific_x = specific_x,
    specific_mean = specific_mean,
    r_squared = r_squared,
    summary = summary_table_gt
  ))
}

model_conditional_quantiles <- function(data, selected_variables,
quantile_method = "linear", poly_quant_degree = NULL, quantiles =
0.5, specific_x = NULL) {
  response_name <- selected_variables[1]
  predictor_name <- selected_variables[2]

  response <- data[[response_name]]
  predictor <- data[[predictor_name]]

  x_seq <- seq(min(predictor), max(predictor), length.out = 200)

  quantile_preds <- list()
  specific_quantiles <- list()
  summary_tables <- list()

  for (q in quantiles) {
    if (quantile_method == "linear") {
      rq_fit <- rq(response ~ predictor, tau = q)
      formula_str <- "Y ~ X"
      params <- broom::tidy(rq_fit)
      metrics <- broom::glance(rq_fit)
    } else if (quantile_method == "poly") {
      rq_fit <- rq(response ~ poly(predictor, degree =
```

```
poly_quant_degree), tau = q)
      formula_str <- paste0("Y ~ poly(X, degree = ",
poly_quant_degree, ")")
      params <- broom::tidy(rq_fit)
      metrics <- broom::glance(rq_fit)
    } else if (quantile_method == "spline") {
      rq_fit <- rq(response ~ ns(predictor, df = 4), tau = q)
      formula_str <- "Y ~ ns(X, df = 4)"
      params <- broom::tidy(rq_fit)
      metrics <- broom::glance(rq_fit)
    } else {
      stop("Neplatný quantile_method!")
    }

    # Predikcie
    quantile_preds[[as.character(q)]] <- predict(rq_fit, newdata
= data.frame(predictor = x_seq))
    specific_quantiles[[as.character(q)]] <- predict(rq_fit,
newdata = data.frame(predictor = specific_x))

    # Suhrnna tabulka
    param_summary <- tibble::tibble(
      Name = paste0("β_", params$term),
      Value = if ("std.error" %in% colnames(params)) {
        paste0(round(params$estimate, 4), " ± ",
round(params$std.error, 4))
      } else {
        paste0(round(params$estimate, 4), " ± N/A")
      }
    )

    eval_metrics <- tibble::tibble(
      Name = c("Null deviance", "Residual deviance", "AIC"),
      Value = as.character(round(c(
        metrics$null.deviance %||% NA,
        metrics$deviance %||% NA,
        metrics$AIC %||% NA
      ), 4))
    )

    model_info <- tibble::tibble(
      Name = c("Quantile Level", "Method", "Basis Function"),
      Value = as.character(c(q, quantile_method, formula_str))
    )

    combined_tbl <- bind_rows(model_info, param_summary,
eval_metrics)

    summary_tables[[as.character(q)]] <- list(
      data = combined_tbl,
      gt = apply_dark_gt_theme(
        combined_tbl %>%
          gt::gt() %>%
          gt::tab_header(
            title = gt::md(paste0("**Quantile Model Summary (τ =
", q, ")**")),
            subtitle = "Conditional Quantile Function"
          ) %>%
          gt::cols_width(Name ~ gt::px(240), Value ~ gt::px(460))
%>%
          gt::opt_row_striping() %>%
          gt::opt_table_font(font = "Arial") %>%
          standardize_gt_table()
      )
    )
  }

  plot_data <- data.frame(X = x_seq)
  for (q in quantiles) {
    plot_data[[paste0("Quantile_", q)]] <-
quantile_preds[[as.character(q)]]
  }

  return(list(
    conditional_quantiles = plot_data,
    specific_x = specific_x,
    specific_quantiles = specific_quantiles,
    summaries = summary_tables
  ))
}

model_discrete_predictor <- function(data, selected_variables,
discrete_model_type = "lm") {
  response_name <- selected_variables[1]
  predictor_name <- selected_variables[2]

  response <- data[[response_name]]
  predictor <- data[[predictor_name]]

  # Prevedieme na faktor a ulozime si jeho urovne
  if (!is.factor(predictor)) {
    predictor <- factor(predictor)
  }
  levels_pred <- levels(predictor)

  # Fit model
  df <- data.frame(response = response, predictor = predictor)
  names(df) <- c(response_name, predictor_name)

  if (!is.null(discrete_model_type) && discrete_model_type ==
"lm") {
    fit <- lm(as.formula(paste(response_name, "~",
predictor_name)), data = df)
    r_squared <- summary(fit)$r.squared
  } else if (!is.null(discrete_model_type) && discrete_model_type
== "glm_log") {
    fit <- glm(as.formula(paste(response_name, "~",
predictor_name)),
               data = df, family = gaussian(link = "log"))
    r_squared <- 1 - fit$deviance / fit$null.deviance
  } else {
    stop("Neplatný model_type. Použi 'lm' alebo 'glm_log'.")
  }

  # Zakladne odhady pre kazdu kategoriu
  est <- broom::tidy(fit)
  param_tbl <- tibble::tibble(
    Name = paste0("β_", est$term),
    Value = paste0(round(est$estimate, 4), " ± ",
                   ifelse("std.error" %in% names(est),
                          round(est$std.error, 4), "N/A"))
  )

  # Sumarizacna tabulka
  summary_tbl <- tibble::tibble(
    Name = c("Model Type", "Link Function", "R-squared"),
    Value = c(
      discrete_model_type,
      if (discrete_model_type == "glm_log") "log link" else
"identity link",
      round(r_squared, 4)
    )
  )

  summary_table_gt <- apply_dark_gt_theme(
    gt::gt(summary_tbl) %>%
      gt::tab_header(
        title = gt::md("**Discrete Predictor Summary**"),
        subtitle = "Conditional Mean Function"
      ) %>%
      gt::cols_width(Name ~ gt::px(240), Value ~ gt::px(460)) %>%
      gt::opt_row_striping() %>%
      gt::opt_table_font(font = "Arial") %>%
      standardize_gt_table()
  )

  # Boxplot
  plot <- ggplot(df, aes(x = factor(!!as.name(predictor_name)), y
= !!as.name(response_name))) +
    geom_boxplot(outlier.shape = NA, fill = "gold", alpha = 0.4)
+
    stat_summary(fun = mean, geom = "point", color = "darkblue",
size = 3) +
    stat_summary(fun.data = mean_se, geom = "errorbar", color =
"darkblue", width = 0.2) +
    theme_minimal() +
    labs(
      title = paste0("Boxplot s odhadmi E[Y | X = kategória] (",
discrete_model_type, ")"),
      x = predictor_name,
      y = response_name
    )

  return(list(
    model = fit,
    r_squared = r_squared,
    plot = plot,
    summary = summary_table_gt
  ))
}

combine_conditional_models <- function(data, selected_variables,
mean_method = "linear", quantile_method = "linear",
poly_mean_degree = NULL, poly_quant_degree = NULL, quantiles =
0.5, specific_x = NULL, discrete_model_type = "lm") {
  if (length(selected_variables) != 2) {
    stop("Zadavaju sa dve premenne: odozva a prediktor.")
  }

  response_name <- selected_variables[1]
  predictor_name <- selected_variables[2]

  if (!is.null(mean_method) && mean_method == "poly" &&
is.null(poly_mean_degree)) {
```

```
    stop("Pri mean_method = 'poly' je treba zadat aj
poly_mean_degree.")
  }

  if (!is.null(poly_mean_degree) && (is.null(mean_method) ||
mean_method != "poly")) {
    stop("poly_mean_degree sa pouziva iba pri mean_method =
'poly'.")
  }

  if (!is.null(quantile_method) && quantile_method == "poly" &&
is.null(poly_quant_degree)) {
    stop("Pri quantile_method = 'poly' je treba zadat
poly_quant_degree.")
  }

  if (!is.null(poly_quant_degree) && (is.null(quantile_method) ||
quantile_method != "poly")) {
    stop("poly_quant_degree sa pouziva iba pri quantile_method =
'poly'.")
  }

  variable_types <- identify_variables(data)

  if (!(response_name %in% colnames(data))) {
    stop(paste("Premenna", response_name, "nie je v datach."))
  }

  if (!(predictor_name %in% colnames(data))) {
    stop(paste("Premenna", predictor_name, "nie je v datach."))
  }

  if (response_name %in% variable_types$Diskretne) {

    # Ak je odozva typu faktor alebo character => pretypovanie na
cislo
    if (is.factor(data[[response_name]])) {
      message(paste("Premenna", response_name, "je faktor.
Pretypovanie na cislo."))
      data[[response_name]] <- as.numeric(data[[response_name]])
    }

    if (is.character(data[[response_name]])) {
      message(paste("Premenna", response_name, "je character.
Pretypovanie na cislo."))
      data[[response_name]] <-
as.numeric(as.factor(data[[response_name]]))
    }
  }

  if (predictor_name %in% variable_types$Diskretne) {
    message(paste("Prediktor", predictor_name, "je diskretny -
pouzitie model_discrete_predictor."))

    discrete_result <- model_discrete_predictor(data,
selected_variables, discrete_model_type = discrete_model_type)

    return(list(
      combined_plot = discrete_result$plot,
      mean_result = list(
        model = discrete_result$model,
        r_squared = discrete_result$r_squared,
        summary = discrete_result$summary
      ),
      quantile_result = NULL
    ))
  }

  mean_result <- model_conditional_mean(
    data = data,
    selected_variables = selected_variables,
    mean_method = mean_method,
    poly_mean_degree = poly_mean_degree,
    specific_x = specific_x
  )

  quantile_result <- model_conditional_quantiles(
    data = data,
    selected_variables = selected_variables,
    quantile_method = quantile_method,
    poly_quant_degree = poly_quant_degree,
    quantiles = quantiles,
    specific_x = specific_x
  )

  p <- ggplot(data, aes_string(x = predictor_name, y =
response_name)) +
    geom_point(alpha = 0.4, color = "orange") +
    geom_line(
      data = mean_result$conditional_mean, aes(x = X, y =
E_Y_given_X),
      color = "blue", size = 1.2, linetype = "solid"
    ) +
    geom_point(
      data = data.frame(specific_x = mean_result$specific_x,
specific_mean = mean_result$specific_mean),
      mapping = aes(x = specific_x, y = specific_mean),
      color = "blue", size = 3
    ) +
    annotate("text",
             x = mean_result$specific_x,
             y = mean_result$specific_mean,
             label = paste0("E[Y|X=",
round(mean_result$specific_x, 2), "]=",
round(mean_result$specific_mean, 2)),
             hjust = -0.1, color = "blue") +
    labs(
      title = "Podmienená stredná hodnota a vybrané kvantilové
funkcie",
      x = paste0(predictor_name, " (Prediktor)"),
      y = paste0(response_name, " (Odozva)")
    ) +
    theme_minimal()

  for (q in quantiles) {
    quant_data <- quantile_result$conditional_quantiles
    col_name <- paste0("Quantile_", q)

    p <- p + geom_line(
      data = quant_data,
      aes_string(x = "X", y = col_name),
      size = 1,
      linetype = "dashed",
      color = "red"
    )

    # Bod na konkretnom specific_x
    p <- p + geom_point(data = data.frame(specific_x =
quantile_result$specific_x,
                                          y_val =
quantile_result$specific_quantiles[[as.character(q)]]),
                        mapping = aes(x = specific_x, y = y_val),
                        color = "red", size = 3)

    p <- p + annotate("text",
                      x = quantile_result$specific_x,
                      y =
quantile_result$specific_quantiles[[as.character(q)]],
                      label = paste0("Q_", q, "(Y|X=",
round(quantile_result$specific_x, 2), ")=",
round(quantile_result$specific_quantiles[[as.character(q)]], 2)),
                      hjust = -0.1, color = "red")
  }

  return(list(
    combined_plot = p,
    mean_result = mean_result,
    quantile_result = quantile_result
  ))
}

plot_decision_boundary <- function(data, response_name,
predictor_names, model, method) {

  if (!is.factor(data[[response_name]])) {
    data[[response_name]] <- as.factor(data[[response_name]])
  }

  # Automaticke vytvorenie gridu podla typu prediktorov
  grid_list <- list()

  for (pred in predictor_names) {
    var <- data[[pred]]
    if (is.factor(var)) {
      grid_list[[pred]] <- levels(var)
    } else {
      grid_list[[pred]] <- seq(min(var, na.rm = TRUE), max(var,
na.rm = TRUE), length.out = 200)
    }
  }

  grid <- expand.grid(grid_list)

  # Ak niektore premenne maju byt faktor v gride, zabezpec
spravne urovne
  for (pred in predictor_names) {
    if (is.factor(data[[pred]])) {
      grid[[pred]] <- factor(grid[[pred]], levels =
levels(data[[pred]]))
    }
  }

  # Predikcia modelu na grid
  if (method == "logistic") {
```

```
    response <- data[[response_name]]

    if (length(levels(as.factor(response))) == 2) {
      # Binarna logisticka regresia
      probs <- predict(model, newdata = grid, type = "response")
      grid$pred <- ifelse(probs > 0.5,
                          levels(as.factor(response))[2],
                          levels(as.factor(response))[1])
    } else {
      # Multinomicka logisticka regresia
      grid$pred <- predict(model, newdata = grid, type = "class")
    }

  } else if (method == "lda") {
    grid$pred <- predict(model, newdata = grid)$class
  } else if (method == "qda") {
    grid$pred <- predict(model, newdata = grid)$class
  } else if (method == "knn") {
    grid$pred <- class::knn(
      train = model$train_data,
      test = grid,
      cl = model$train_response,
      k = model$k
    )
  }

  p <- ggplot() +
    # Rozhodovacie oblasti (predikcie modelu)
    geom_tile(data = grid, aes_string(x = predictor_names[1], y =
predictor_names[2], fill = "pred"), alpha = 0.3) +

    # Skutocne data (odozvy)
    geom_point(data = data, aes_string(x = predictor_names[1], y
= predictor_names[2], color = response_name, shape =
response_name), size = 3) +

    labs(
      title = paste0("Predikčný model metódy: ", method),
      subtitle = paste("Prediktory:", paste(predictor_names,
collapse = " a "), "| Odozva:", response_name),
      x = paste0(predictor_names[1], " (Prediktor)"),
      y = paste0(predictor_names[2], " (Prediktor)"),
      fill = "Predikcia modelu",
      color = "Skutočná odozva",
      shape = "Skutočná odozva"
    ) +

    theme_minimal() +
    theme(
      plot.title = element_text(size = 14, face = "bold"),
      plot.subtitle = element_text(size = 10)
    )

  return(p)
}

plot_classification_1D_combined <- function(data, response_name,
predictor_name, model, method) {

  p1 <- NULL

  if (!is.factor(data[[response_name]])) {
    data[[response_name]] <- as.factor(data[[response_name]])
  }

  # Ulozenie originalnych levels
  if (!is.factor(data[[response_name]])) {
    data[[response_name]] <- factor(data[[response_name]])
  }

  response_levels <-
sort(unique(as.character(data[[response_name]])))
  data[[response_name]] <- factor(data[[response_name]], levels =
response_levels)

  classes <- levels(data[[response_name]])
  n_classes <- length(classes)

  # Paleta farieb a tvarov
  my_colors <- c("#F8766D", "#DAA520", "#00BA38", "#619CFF",
"#C77CFF", "#FF61C3")[1:n_classes]
  my_shapes <- c(16, 17, 15, 3, 4, 8)[1:n_classes]

  # Urcenie typu prediktora
  predictor_type <- if (is.numeric(data[[predictor_name]]) &&
length(unique(data[[predictor_name]])) > 5) {
    "continuous"
  } else {
    "discrete"
  }

  # Vytvorenie gridu
  grid_x <- if (predictor_type == "continuous") {
    seq(min(data[[predictor_name]], na.rm = TRUE),
        max(data[[predictor_name]], na.rm = TRUE),
        length.out = 300)
  } else {
    sort(unique(data[[predictor_name]]))
  }

  grid <- setNames(data.frame(grid_x), predictor_name)

  probs_long <- NULL

  if (method == "logistic" && length(classes) > 2) {
    probs <- predict(model, newdata = grid, type = "probs") %>%
as.data.frame()

    true_class_names <- if (!is.null(model$lev)) {
      model$lev
    } else {
      response_levels
    }

    if (all(colnames(probs) %in%
as.character(seq_along(true_class_names)))) {
      colnames(probs) <- true_class_names
    }

    grid$pred_class <- apply(probs, 1, function(row) {
      names(row)[which.max(row)]
    })
    grid$pred_class <- factor(grid$pred_class, levels = classes)

    probs[[predictor_name]] <- grid[[predictor_name]]
    probs_long <- probs %>%
      tidyr::pivot_longer(cols = all_of(classes), names_to =
"Hodnota_odozvy", values_to = "Pravdepodobnost")
    probs_long$Hodnota_odozvy <-
factor(probs_long$Hodnota_odozvy, levels = classes)

  } else if (method == "logistic" && length(classes) == 2) {
    p <- predict(model, newdata = grid, type = "response")

    probs <- data.frame(grid_x)
    colnames(probs) <- predictor_name
    probs[[as.character(classes[1])]] <- 1 - p
    probs[[as.character(classes[2])]] <- p

    grid$pred_class <- ifelse(p > 0.5, as.character(classes[2]),
as.character(classes[1])) %>%
      factor(levels = classes)
    probs_long <- probs %>%
      tidyr::pivot_longer(cols = all_of(classes), names_to =
"Hodnota_odozvy", values_to = "Pravdepodobnost")
    probs_long$Hodnota_odozvy <-
factor(probs_long$Hodnota_odozvy, levels = classes)

  } else if (method %in% c("lda", "qda")) {
    probs <- predict(model, newdata = grid)$posterior %>%
as.data.frame()
    probs <- probs[, classes, drop = FALSE]
    grid$pred_class <- apply(probs, 1, function(row)
names(row)[which.max(row)])
    grid$pred_class <- factor(grid$pred_class, levels = classes)
    probs[[predictor_name]] <- grid[[predictor_name]]
    probs_long <- probs %>%
      tidyr::pivot_longer(cols = all_of(classes), names_to =
"Hodnota_odozvy", values_to = "Pravdepodobnost")
    probs_long$Hodnota_odozvy <-
factor(probs_long$Hodnota_odozvy, levels = classes)

  } else if (method == "knn") {
    grid_matrix <- as.matrix(grid[, predictor_name, drop =
FALSE])
    preds <- class::knn(
      train = model$train_data,
      test = grid_matrix,
      cl = model$train_response,
      k = model$k
    )
    grid$pred_class <- factor(preds, levels = classes)
  } else {
    stop("Neznama metoda!")
  }

  present_levels <- levels(droplevels(grid$pred_class))
  present_indices <- match(present_levels, classes)
  colors_used <- my_colors[present_indices]

  # Pravdepodobnosti
  if (!is.null(probs_long)) {
    probs_long <- probs_long %>% filter(Hodnota_odozvy %in%
present_levels)
```

```
    if (predictor_type == "continuous") {
      p1 <- ggplot(probs_long, aes(x = .data[[predictor_name]], y
= Pravdepodobnost, color = Hodnota_odozvy)) +
        geom_line(linewidth = 1) +
        scale_color_manual(values = setNames(colors_used,
present_levels)) +
        labs(
          title = paste0("Pravdepodobnosti predikcie - ",
method),
          x = paste0(predictor_name, " (Prediktor)"),
          y = "Pravdepodobnosť",
          color = "Hodnota odozvy"
        ) +
        theme_minimal()
    } else {
      p1 <- ggplot(probs_long, aes(x =
factor(.data[[predictor_name]]), y = Pravdepodobnost, fill =
Hodnota_odozvy)) +
        geom_col(position = "dodge") +
        scale_fill_manual(values = setNames(colors_used,
present_levels)) +
        labs(
          title = paste0("Pravdepodobnosti predikcie - ",
method),
          x = predictor_name,
          y = "Pravdepodobnosť",
          fill = "Hodnota odozvy"
        ) +
        theme_minimal()
    }
  }

  #data[[response_name]] <- factor(data[[response_name]], levels
= sort(as.numeric(levels(data[[response_name]]))))
  # Skutocne hodnoty + rozhodovacia hranica
  p2 <- ggplot() +
    scale_y_discrete(limits = classes) +
    geom_tile(
      data = grid,
      aes(x = .data[[predictor_name]], y = pred_class, fill =
pred_class),
      height = 0.5, alpha = 0.3
    ) +
    geom_jitter(
      data = data,
      aes(x = .data[[predictor_name]], y =
.data[[response_name]],
          color = .data[[response_name]],
          shape = .data[[response_name]]),
      width = 0.1, height = 0.1, size = 2
    ) +
    scale_fill_manual(values = setNames(colors_used,
present_levels)) +
    scale_color_manual(values = my_colors) +
    scale_shape_manual(values = my_shapes) +
    labs(
      title = "Hodnoty odozvy + rozhodovacia hranica",
      x = paste0(predictor_name, " (Prediktor)"),
      y = paste0(response_name, " (Odozva)"),
      fill = "Predikcia",
      color = "Skutočná hodnota",
      shape = "Skutočná hodnota"
    ) +
    theme_minimal()

  if (!is.null(p1)) {
    return(p1 / p2)
  } else {
    return(p2)
  }
}

classification_model <- function(data, response_name,
predictor_names, method = "logistic", k = NULL) {

  if (length(predictor_names) > 2) {
    stop("Pocet prediktorov nesmie byt vacsi ako 2.")
  }

  if (!(response_name %in% colnames(data))) {
    stop(paste("Premenna", response_name, "nie je v datach!"))
  }

  predictors_exist <- predictor_names %in% colnames(data)
  if (!all(predictors_exist)) {
    stop(paste("Tieto prediktory chybaju v datach:",
paste(predictor_names[!predictors_exist], collapse = ", ")))
  }

  response <- data[[response_name]]
  if (!is.factor(response)) {
    unique_values <- length(unique(response))
    if (unique_values <= 10) {
      message(paste("Odozva", response_name, "ma", unique_values,
"unikatnych hodnot. Konverzia na faktor..."))
      response <- factor(response)
      data[[response_name]] <- response
    } else {
      stop(paste("Odozva", response_name, "nie je diskretna! Musi
byt faktor alebo mat konecny pocet kategorii."))
    }
  }

  if (is.numeric(response) && length(unique(response)) == 2) {
    message("Konvertovanie ciselnej odozvy na faktor.")
    response <- as.factor(response)
    data[[response_name]] <- response
  }

  # Konverzia diskretnych prediktorov na faktor
  for (pred in predictor_names) {
    if (!is.numeric(data[[pred]]) && !is.factor(data[[pred]])) {
      data[[pred]] <- as.factor(data[[pred]])
    }

    # Ak je prediktor číselny ale ma malo unikatnych hodnot
    if (is.numeric(data[[pred]]) && length(unique(data[[pred]]))
<= 10) {
      data[[pred]] <- as.factor(data[[pred]])
    }
  }

  # Osetrenie kategorii s malym poctom pozorovani pre QDA
  if (method == "qda") {
    class_sizes <- table(response)
    too_small_classes <- names(class_sizes[class_sizes < 4])

    if (length(too_small_classes) > 0) {
      warning(paste("Tieto triedy maju menej ako 4 pozorovania a
budu odstranene:", paste(too_small_classes, collapse = ", ")))

      data <- data[!(response %in% too_small_classes), ]
      response <- data[[response_name]]
    }
  }

  # Zostavenie formuly pre modelovanie
  formula_str <- paste(response_name, "~", paste(predictor_names,
collapse = " + "))
  formula <- as.formula(formula_str)

  # Trenovanie modelu
  if (method == "logistic") { # logisticka regresia

    if (length(levels(response)) == 2) {
      # Klasicka binarna logisticka regresia
      model <- glm(formula, data = data, family = binomial)
      probs <- predict(model, newdata = data, type = "response")
      preds <- ifelse(probs > 0.5, levels(response)[2],
levels(response)[1])
    } else {
      # Multinomicka logisticka regresia pre viac ako dve triedy
      model <- nnet::multinom(formula, data = data, trace =
FALSE)
      preds <- predict(model, newdata = data)
    }

  } else if (method == "lda") { # linearna diskriminacna analyza

    model <- MASS::lda(formula, data = data)

  } else if (method == "qda") { # kvadraticka diskriminacna
analyza

    model <- MASS::qda(formula, data = data)

  } else if (method == "knn") { # metoda k-najblizsich susedov
    train_data <- as.matrix(data[, predictor_names, drop =
FALSE])
    train_response <- data[[response_name]]

    if (is.null(k)) {
      message("Optimalizujem parameter 'k' pre k-NN...")

      accuracy_scores <- numeric(20)

      for (i in 1:20) {
        preds_i <- class::knn(
          train = train_data,
          test = train_data,
          cl = train_response,
          k = i
```

```
        )
        acc_i <- mean(preds_i == train_response)
        accuracy_scores[i] <- acc_i
      }

      # Najlepsie k (ak je viac rovnakych, berieme najmensie)
      k <- which.max(accuracy_scores)
      message(paste("Optimalne k je:", k, "s presnostou:",
round(accuracy_scores[k], 4)))
    }

    model <- list(
      train_data = train_data,
      train_response = train_response,
      k = k
    )

  }
  else {
    stop("Neplatna metoda. Vyber: 'logistic', 'lda', 'qda' alebo
'knn'.")
  }

  # Predikcie
  if (method %in% c("logistic", "lda", "qda")) {

    if (method == "logistic") {

      if (length(levels(response)) == 2) {
        # Klasicka binarna logisticka regresia
        probs <- predict(model, newdata = data, type =
"response")
        preds <- ifelse(probs > 0.5, levels(response)[2],
levels(response)[1])

      } else {
        # Multinomicka logisticka regresia
        preds <- predict(model, newdata = data, type = "class")
      }

    } else if (method == "lda") {
      preds <- predict(model, newdata = data)$class

    } else if (method == "qda") {
      preds <- predict(model, newdata = data)$class
    }

  } else if (method == "knn") {

    preds <- class::knn(
      train = model$train_data,
      test = model$train_data,
      cl = model$train_response,
      k = model$k
    )
  }

  # Vypocet presnosti
  accuracy <- sum(preds == response) / length(preds)
  confusion_mat <- table(Predikovane = preds, Skutocne =
response)

  # Vystupna sumarizacna tabulka
  param_rows <- list()
  metrics <- tibble::tibble(
    Name = c("Model Type", "Accuracy"),
    Value = as.character(c(method, round(accuracy, 4)))
  )

  if (method == "logistic") {
    metrics <- add_row(metrics, Name = "Response Type", Value =
ifelse(length(levels(response)) == 2, "Binary", "Multinomial"))

    if (length(levels(response)) == 2) {
      coefs <- broom::tidy(model)
      param_rows <- coefs %>%
        dplyr::transmute(
          Name = paste0("β_", term),
          Value = paste0(round(estimate, 4), " ± ",
round(std.error, 4))
        )
    } else {
      coefs <- broom::tidy(model)
      param_rows <- coefs %>%
        dplyr::transmute(
          Name = paste0("β_", term, " (", y.level, ")"),
          Value = paste0(round(estimate, 4), " ± ",
ifelse(is.na(std.error), "N/A", round(std.error, 4)))
        )
    }

  } else if (method %in% c("lda", "qda")) {
    metrics <- add_row(metrics, Name = "Response Type", Value =
"Multiclass")
    if (!is.null(model$df)) {
      metrics <- add_row(metrics, Name = "Degrees of Freedom",
Value = as.character(model$df))
    }

    for (cls in rownames(model$means)) {
      for (pred in predictor_names) {
        mean_val <- model$means[cls, pred]
        class_data <- data[data[[response_name]] == cls, pred,
drop = TRUE]
        sd_val <- sd(class_data, na.rm = TRUE)

        param_rows <- dplyr::bind_rows(
          param_rows,
          tibble::tibble(Name = paste0("Mean(", pred, ", class ",
cls, ")"), Value = as.character(round(mean_val, 4))),
          tibble::tibble(Name = paste0("SD(", pred, ", class ",
cls, ")"), Value = as.character(round(sd_val, 4)))
        )
      }
    }

  } else if (method == "knn") {
    metrics <- add_row(metrics, Name = "Hyperparameter (k)",
Value = as.character(model$k))
  }

  if (length(param_rows) == 0) {
    param_rows <- tibble::tibble(Name = character(), Value =
character())
  }

  summary_tbl <- dplyr::bind_rows(metrics, param_rows)

  summary_gt <- apply_dark_gt_theme(
    gt::gt(summary_tbl) %>%
      gt::tab_header(
        title = gt::md("**Classification Model Summary**"),
        subtitle = paste("Method:", toupper(method))
      ) %>%
      gt::cols_width(Name ~ gt::px(260), Value ~ gt::px(460)) %>%
      gt::opt_row_striping() %>%
      gt::opt_table_font(font = "Arial") %>%
      standardize_gt_table()
  )

  # Vystup
  result <- list(
    model = model,
    predictions = preds,
    accuracy = accuracy,
    confusion_matrix = confusion_mat,
    summary_gt = summary_gt
  )

  if (length(predictor_names) == 1) {
    result$decision_plot <- plot_classification_1D_combined(
      data, response_name, predictor_names[1], model, method
    )
  } else if (length(predictor_names) == 2) {
    result$decision_plot <- plot_decision_boundary(
      data, response_name, predictor_names, model, method
    )
  }

  return(result)
}

model_conditional_continuous_densities <- function(df, n_breaks,
density_scaling, mean_curve, quantiles,

mean_poly_degree, quantile_poly_degree,

mixture_model_outputs = list(),

model_output_kernel, model_output_normal, model_output_t,
model_output_copula) {

  response_name <- attr(df, "response_var")
  predictor_name <- attr(df, "predictor_var")

  x <- df$predictor
  y <- df$response

  # Zistenie typu prediktora
  var_types <- identify_variables(df)
  predictor_is_discrete <- "predictor" %in% var_types$Diskretne

  valid_idx <- if (is.numeric(x)) is.finite(x) & is.finite(y)
else !is.na(x) & !is.na(y)
```

```
  df <- df[valid_idx, ]
  x <- x[valid_idx]
  y <- y[valid_idx]

  if (!predictor_is_discrete) {
    if (!is.null(model_output_copula)) {
      dens2d <- list(
        x = model_output_copula$y_vals,
        y = model_output_copula$x_vals,
        z = t(model_output_copula$z_matrix)
      )
    } else {
      dens2d <- MASS::kde2d(x, y, n = 100)
    }

    dens_x <- density(x, bw = "nrd0", n = 100)

    x_seq <- dens2d$x
    y_range <- range(y, na.rm = TRUE)
    y_seq <- seq(y_range[1], y_range[2], length.out = 300)

    df$predictor_numeric <- df$predictor
  }

  if (predictor_is_discrete) {
    x_factor_original <- factor(df$predictor)
    category_levels <- levels(x_factor_original)
    category_numeric <- seq_along(category_levels)
    names(category_numeric) <- category_levels

    df$predictor_numeric <- as.numeric(x_factor_original)
    x_numeric <- df$predictor_numeric
    x <- x_numeric

    breaks <- x_numeric[match(category_levels,
as.character(df$predictor))]
    x_seq <- sort(unique(x_numeric))

  } else {
    eps <- 0.001 * diff(range(x, na.rm = TRUE))
    breaks <- seq(min(x, na.rm = TRUE) + eps, max(x, na.rm =
TRUE) - eps, length.out = n_breaks)
  }

  density_data <- list()
  mean_curve_data <- NULL
  quantile_data <- list()

  if (predictor_is_discrete) {

    for (model_type in names(mixture_model_outputs)) {
      safe_approx <- function(x, y, xout, rule = 2) {
        x <- as.numeric(x)
        y <- as.numeric(y)

        valid <- is.finite(x) & is.finite(y)
        x <- x[valid]
        y <- y[valid]

        if (length(x) < 2 || length(unique(x)) < 2 || length(y)
!= length(x)) {
          warning("Skipping approx(): insufficient or invalid
data")
          return(rep(0, length(xout)))
        }

        tryCatch({
          approx(x = x, y = y, xout = xout, rule = rule)$y
        }, error = function(e) {
          warning(paste("approx() failed:", e$message))
          rep(0, length(xout))
        })
      }

      output <- mixture_model_outputs[[model_type]]
      y_min <-
suppressWarnings(min(output$density_data$Continuous_Var, na.rm =
TRUE))
      y_max <-
suppressWarnings(max(output$density_data$Continuous_Var, na.rm =
TRUE))

      if (!is.finite(y_min) || !is.finite(y_max) || y_min ==
y_max) {
        warning(paste("Invalid y range for model:", model_type))
        next
      }

      y_seq <- seq(y_min, y_max, length.out = 100)

      fade_density <- density(rep(0, 100), bw = 0.3, n = 100,
from = -1, to = 1)
      fade_factor <- fade_density$y / max(fade_density$y)

      observed_levels <-
sort(unique(as.character(output$density_data$Discrete_Var)))
      category_numeric <- setNames(seq_along(observed_levels),
observed_levels)

      for (cat in observed_levels) {
        sub_df <-
output$density_data[output$density_data$Discrete_Var == cat, ]
%>%
          dplyr::group_by(Continuous_Var) %>%
          dplyr::summarise(Density = mean(Density), .groups =
"drop") %>%
          dplyr::arrange(Continuous_Var)

        if (nrow(sub_df) >= 2 &&
length(unique(sub_df$Continuous_Var)) >= 2 &&
length(sub_df$Density) >= 2) {

          sub_df <- sub_df[order(sub_df$Continuous_Var), ]

          interpolated <- safe_approx(
            x = sub_df$Continuous_Var,
            y = sub_df$Density,
            xout = y_seq
          )

          area <- compute_area(y_seq, interpolated)
          message(sprintf("Plocha pod f_cond pre kategóriu = %s
(model: %s): %.6f", cat, model_type, area))

          # f_scaled <- interpolated * fade_factor
          f_scaled <- interpolated
          f_scaled[!is.finite(f_scaled)] <- 0

          if (max(f_scaled) > 0) {
            cat <- as.character(cat)
            #f_scaled <- f_scaled / max(f_scaled)

            if (max(f_scaled) < 0.01) {
              f_scaled <- f_scaled * 50
            }

            if (!is.na(category_numeric[cat])) {
              xi_numeric <- as.numeric(category_numeric[cat])

              density_data[[length(density_data) + 1]] <-
data.frame(
                x = xi_numeric,
                y = y_seq,
                width = f_scaled,
                section = cat,
                type = model_type,
                category_numeric = xi_numeric,
                category_label = cat
              )
            }
          }
        }
      }
    }

    df$predictor_numeric <- x

  } else {

    epsilon <- 0.02 * diff(range(x))
    max_n_local <- max(sapply(breaks, function(xi) sum(abs(x -
xi) <= epsilon)))

    for (xi in breaks) {
      subset_y <- y[abs(x - xi) <= epsilon]
      n_local <- length(subset_y)

      section_label <- as.factor(round(xi, 2))
      fade_density <- density(rep(0, 100), bw = 0.3, n =
length(y_seq), from = -1, to = 1)
      fade_factor <- fade_density$y / max(fade_density$y)

      if (!is.null(model_output_copula)) {

        col_idx <- which.min(abs(model_output_copula$y_vals -
xi))
        fxy <- model_output_copula$z_matrix[, col_idx]
        fx  <- model_output_copula$fx_vals[col_idx]

        if (!is.na(fx) && fx > .Machine$double.eps) {
          f_cond <- fxy / fx
          f_cond[is.na(f_cond)] <- 0

          interpolated <- approx(x = model_output_copula$x_vals,
```

```
y = f_cond, xout = y_seq, rule = 2)

          area_before_scaling <- compute_area(y_seq,
interpolated$y)

          print(area_before_scaling)

          if (!is.na(area_before_scaling) &&
abs(area_before_scaling - 1) > 0.05) {
            warning(sprintf("Plocha pod f_cond pre x = %.2f
(model: %s) = %.4f", xi, paste0("copula[",
model_output_copula$copula_type, "]"), area_before_scaling))
          }

          f_scaled <- interpolated$y * density_scaling

          density_data[[length(density_data) + 1]] <- data.frame(
            x = xi, y = y_seq, width = f_scaled,
            section = section_label, type = paste0("copula[",
model_output_copula$copula_type, "]")
          )
        }
      }

      if (!is.null(model_output_kernel)) {
        col_idx <- which.min(abs(model_output_kernel$x_vals -
xi))
        fxy <- model_output_kernel$z_matrix[, col_idx]

        delta_y <- diff(model_output_kernel$y_vals[1:2])
        fx <- sum(fxy) * delta_y

        if (!is.na(fx) && fx > .Machine$double.eps) {
          f_cond <- fxy / fx
          f_cond[is.na(f_cond)] <- 0

          interpolated <- approx(
            x = model_output_kernel$y_vals,
            y = f_cond,
            xout = y_seq,
            rule = 2
          )

          area_before_scaling <- compute_area(y_seq,
interpolated$y)

          message(sprintf("Plocha pod f_cond pre x = %.2f (model:
%s): %.6f",
                          xi, "KDE", area_before_scaling))

          if (!is.na(area_before_scaling) &&
abs(area_before_scaling - 1) > 0.05) {
            warning(sprintf("Plocha sa výrazne líši od 1 pre x =
%.2f (KDE): %.6f",
                            xi, area_before_scaling))
          }

          # f_scaled <- interpolated$y * fade_factor
          # scale_factor <- if (max_n_local > 0) sqrt(n_local /
max_n_local) else 1
          # f_scaled <- f_scaled / max(f_scaled) *
density_scaling * scale_factor
          f_scaled <- interpolated$y * density_scaling

          density_data[[length(density_data) + 1]] <- data.frame(
            x = xi,
            y = y_seq,
            width = f_scaled,
            section = section_label,
            type = "KDE"
          )
        }
      }

      if (!is.null(model_output_normal)) {
        row_idx <- which.min(abs(model_output_normal$x_vals -
xi))
        fxy <- model_output_normal$z_matrix[row_idx, ]

        fx_vals <- rowSums(model_output_normal$z_matrix) *
diff(model_output_normal$y_vals[1:2])
        fx <- fx_vals[row_idx]

        if (!is.na(fx) && fx > .Machine$double.eps) {
          f_cond <- fxy / fx
          f_cond[is.na(f_cond)] <- 0
          interpolated <- approx(
            x = model_output_normal$y_vals,
            y = f_cond,
            xout = y_seq,
            rule = 2
          )

          # f_scaled <- interpolated$y * fade_factor
          # scale_factor <- if (max_n_local > 0) sqrt(n_local /
max_n_local) else 1
          # f_scaled <- f_scaled / max(f_scaled) *
density_scaling * scale_factor
          f_scaled <- interpolated$y * density_scaling

          density_data[[length(density_data) + 1]] <- data.frame(
            x = xi, y = y_seq, width = f_scaled,
            section = section_label, type = "normal"
          )
        }
      }

      if (!is.null(model_output_t)) {
        row_idx <- which.min(abs(model_output_t$x_vals - xi))
        fxy <- model_output_t$z_matrix[row_idx, ]

        fx_vals <- rowSums(model_output_t$z_matrix) *
diff(model_output_t$y_vals[1:2])
        fx <- fx_vals[row_idx]

        if (!is.na(fx) && fx > .Machine$double.eps) {
          f_cond <- fxy / fx
          f_cond[is.na(f_cond)] <- 0
          interpolated <- approx(x = model_output_t$y_vals, y =
f_cond, xout = y_seq, rule = 2)

          # f_scaled <- interpolated$y * fade_factor
          # scale_factor <- if (max_n_local > 0) sqrt(n_local /
max_n_local) else 1
          # f_scaled <- f_scaled / max(f_scaled) *
density_scaling * scale_factor
          f_scaled <- interpolated$y * density_scaling

          density_data[[length(density_data) + 1]] <- data.frame(
            x = xi, y = y_seq, width = f_scaled,
            section = section_label, type = "t"
          )
        }
      }
    }
  }

  if (!predictor_is_discrete && mean_curve) {
    fit <- lm(response ~ poly(predictor_numeric,
mean_poly_degree, raw = TRUE), data = df)
    mean_pred <- predict(fit, newdata =
data.frame(predictor_numeric = x_seq))
    newdata = data.frame(predictor_numeric = x_seq)
    mean_curve_data <- data.frame(x = x_seq, y = mean_pred)
  }

  if (!predictor_is_discrete && !is.null(quantiles)) {
    for (q in quantiles) {
      rq_fit <- quantreg::rq(response ~ poly(predictor_numeric,
quantile_poly_degree, raw = TRUE), tau = q, data = df)
      q_pred <- predict(rq_fit, newdata =
data.frame(predictor_numeric = x_seq))
      quantile_data[[as.character(q)]] <- data.frame(x = x_seq, y
= q_pred)
    }
  }

  if (length(x) >= 2 && length(y) >= 2 && sum(complete.cases(x,
y)) >= 2) {
    mu_x <- mean(x, na.rm = TRUE)
    mu_y <- mean(y, na.rm = TRUE)
    sd_x <- sd(x, na.rm = TRUE)
    sd_y <- sd(y, na.rm = TRUE)
    cor_xy <- cor(x, y, use = "complete.obs")
  } else {
    warning("Nedostatok platných párov (x, y) na výpočet
štatistiky.")
    mu_x <- mu_y <- sd_x <- sd_y <- cor_xy <- NA
  }

  epsilon <- 0.02 * diff(range(x))
  summary_table <- lapply(breaks, function(xi) {
    subset_y <- df$response[abs(df$predictor_numeric - xi) <=
epsilon]
    if (length(subset_y) == 0) {
      return(data.frame(Break = round(xi, 2), Count = 0, Mean =
NA, SD = NA, Min = NA, Max = NA))
    }
    data.frame(
      Break = round(xi, 2),
      Count = length(subset_y),
      Mean = round(mean(subset_y, na.rm = TRUE), 4),
      SD = round(sd(subset_y, na.rm = TRUE), 4),
      Min = round(min(subset_y, na.rm = TRUE), 4),
```

```
      Max = round(max(subset_y, na.rm = TRUE), 4)
    )
  }) %>% dplyr::bind_rows()

  return(list(
    df = df,
    density_data = density_data,
    breaks = breaks,
    x_seq = x_seq,
    mean_curve_data = mean_curve_data,
    quantile_data = quantile_data,
    summary_table = summary_table,
    meta = list(
      response_name = response_name,
      predictor_name = predictor_name,
      predictor_is_discrete = predictor_is_discrete,
      epsilon = epsilon,
      mu_x = mu_x,
      mu_y = mu_y,
      sd_x = sd_x,
      sd_y = sd_y,
      cor_xy = cor_xy
    )
  ))
}

render_conditional_continuous_densities <- function(model_output)
{
  df <- model_output$df
  density_data <- model_output$density_data
  breaks <- model_output$breaks
  x_seq <- model_output$x_seq
  mean_curve_data <- model_output$mean_curve_data
  quantile_data <- model_output$quantile_data
  summary_table <- model_output$summary_table
  meta <- model_output$meta

  if (meta$predictor_is_discrete) {
    density_df <- dplyr::bind_rows(density_data)

    category_levels <- sort(unique(density_df$category_label))

    category_levels <- sort(unique(density_df$category_label))
    category_levels <- as.character(category_levels)

    df$category_label <- as.character(df$predictor)
    df$category_numeric <- as.numeric(factor(df$category_label,
levels = category_levels))

    density_df$category_numeric <-
as.numeric(factor(density_df$category_label, levels =
category_levels))

    p <- ggplot(df, aes(x = category_numeric, y = response)) +
      geom_point(alpha = 0.6, color = "darkorange")

    # Hustoty
    if (length(density_data) > 0) {
      p <- p +
        geom_path(
          data = density_df,
          aes(x = category_numeric - width, y = y, group =
interaction(section, type), color = type),
          linewidth = 1,
          inherit.aes = FALSE
        )
    }

    # Osy a popis
    p <- p +
      scale_x_continuous(
        breaks = seq_along(category_levels),
        labels = category_levels,
        expand = expansion(add = 0.3)
      ) +
      labs(
        x = paste0(meta$predictor_name, " (Prediktor)"),
        y = paste0(meta$response_name, " (Odozva)")
      )
  }
  else {
    p <- ggplot(df, aes(x = predictor, y = response)) +
      geom_point(alpha = 0.6, color = "darkorange") +
      geom_vline(xintercept = breaks, linetype = "dashed", color
= "grey50")

    if (length(density_data) > 0) {
      density_df <- dplyr::bind_rows(density_data)

      type_levels <- unique(density_df$type)
      colors <- scales::hue_pal()(length(type_levels))
      names(colors) <- type_levels

      p <- p + geom_path(
        data = density_df,
        aes(x = x - width, y = y, group = interaction(section,
type), color = type),
        linewidth = 1
      ) +
        scale_color_manual(values = colors)
    }
  }

  if (!is.null(mean_curve_data)) {
    p <- p + geom_line(
      data = mean_curve_data,
      aes(x = x, y = y),
      color = "blue", linewidth = 1.2
    )
  }

  if (length(quantile_data) > 0) {
    for (qdat in quantile_data) {
      p <- p + geom_line(
        data = qdat,
        aes(x = x, y = y),
        color = "purple", linetype = "dashed"
      )
    }
  }

  p <- p + theme_bw() +
    labs(
      title = paste("Podmienené hustoty pre", meta$response_name,
"podľa", meta$predictor_name),
      x = paste0(meta$predictor_name, " (Prediktor)"),
      y = paste0(meta$response_name, " (Odozva)")
    )

  summary_gt <- apply_dark_gt_theme(
    gt::gt(summary_table) %>%
      gt::tab_header(
        title = gt::md("**Estimated Conditional Densities**"),
        subtitle = paste("Local stats in ±", round(meta$epsilon,
2), "around each cut for", meta$response_name)
      ) %>%
      gt::cols_width(everything() ~ gt::px(130)) %>%
      gt::opt_row_striping() %>%
      gt::opt_table_font(font = "Arial") %>%
      standardize_gt_table()
  )

  return(list(
    plot = p,
    summary_gt = summary_gt
  ))
}

model_conditional_discrete_densities <- function(df, n_breaks,
density_scaling, normal_density, kernel_density) {
  response_name <- attr(df, "response_var")
  predictor_name <- attr(df, "predictor_var")

  df$response_cat <- df$response
  response_levels <- levels(factor(df$response_cat))
  df$response <- as.numeric(factor(df$response_cat, levels =
response_levels))

  var_types <- identify_variables(df)
  predictor_is_discrete <- "predictor" %in% var_types$Diskretne

  density_data <- list()

  if (predictor_is_discrete) {

    df$predictor_cat <- factor(df$predictor)
    predictor_levels <- levels(df$predictor_cat)
    df$predictor_numeric <- as.numeric(df$predictor_cat)

    for (i in seq_along(predictor_levels)) {
      level_label <- predictor_levels[i]
      x_val <- i
      sub_df <- df[df$predictor_cat == level_label, ]
      if (nrow(sub_df) < 1) next

      prob_table <- prop.table(table(sub_df$response))
      for (d in names(prob_table)) {
        d_val <- as.numeric(d)
        prob <- as.numeric(prob_table[[d]])
        density_data[[length(density_data) + 1]] <- data.frame(
          x_center = x_val,
          x_center_numeric = x_val,
          section = level_label,
          d = d_val,
```

```
          Category = response_levels[d_val],
          prob = prob,
          y = d_val,
          method = "Empirical"
        )
      }
    }
  } else {
    breaks <- seq(min(df$predictor, na.rm = TRUE),
max(df$predictor, na.rm = TRUE), length.out = n_breaks + 1)
    centers <- (head(breaks, -1) + tail(breaks, -1)) / 2
    center_labels <- paste0("(", round(head(breaks, -1), 1), ",
", round(tail(breaks, -1), 1), "]")

    prior_probs <- prop.table(table(df$response))

    if (kernel_density) {
      f_C <- density(df$predictor)
      f_C_fun <- approxfun(f_C$x, f_C$y, rule = 2)
    } else if (normal_density) {
      mu_c <- mean(df$predictor, na.rm = TRUE)
      sigma_c <- sd(df$predictor, na.rm = TRUE)
      f_C_fun <- function(c) dnorm(c, mean = mu_c, sd = sigma_c)
    }

    for (d in names(prior_probs)) {
      d_val <- as.numeric(d)
      sub_df <- df[df$response == d_val, ]
      if (nrow(sub_df) < 2) next

      if (kernel_density) {
        f_c_given_d <- density(sub_df$predictor)
        f_c_given_d_fun <- approxfun(f_c_given_d$x,
f_c_given_d$y, rule = 2)
      }
      if (normal_density) {
        mu <- mean(sub_df$predictor)
        sigma <- sd(sub_df$predictor)
      }

      for (j in seq_along(centers)) {
        c <- centers[j]
        section_label <- center_labels[j]
        denominator <- f_C_fun(c)

        if (kernel_density) {
          numerator_kde <- prior_probs[d] * f_c_given_d_fun(c)
          cond_prob_kde <- ifelse(denominator > 0, numerator_kde
/ denominator, 0)
          density_data[[length(density_data) + 1]] <- data.frame(
            x_center = c,
            x_center_numeric = c,
            section = section_label,
            d = d_val,
            Category = response_levels[d_val],
            prob = cond_prob_kde,
            y = d_val + runif(1, -0.1, 0.1),
            method = "KDE"
          )
        }

        if (normal_density && is.finite(mu) && is.finite(sigma)
&& sigma > 0) {
          f_c_given_d_norm <- dnorm(c, mean = mu, sd = sigma)
          numerator_norm <- prior_probs[d] * f_c_given_d_norm
          cond_prob_norm <- ifelse(denominator > 0,
numerator_norm / denominator, 0)
          density_data[[length(density_data) + 1]] <- data.frame(
            x_center = c,
            x_center_numeric = c,
            section = section_label,
            d = d_val,
            Category = response_levels[d_val],
            prob = cond_prob_norm,
            y = d_val + runif(1, -0.1, 0.1),
            method = "Normal"
          )
        }
      }
    }
    df$predictor_numeric <- df$predictor
  }

  density_df <- dplyr::bind_rows(density_data)
  density_df$d <- factor(density_df$d, labels = response_levels)
  density_df$x_end <- density_df$x_center + density_scaling *
density_df$prob

  return(list(
    df = df,
    density_df = density_df,
    response_levels = response_levels,
    predictor_is_discrete = predictor_is_discrete,
    response_name = response_name,
    predictor_name = predictor_name
  ))
}

render_conditional_discrete_densities <- function(model_output,
ordinal = FALSE) {
  df <- model_output$df
  density_df <- model_output$density_df
  response_levels <- model_output$response_levels
  predictor_is_discrete <- model_output$predictor_is_discrete
  response_name <- model_output$response_name
  predictor_name <- model_output$predictor_name

  if (predictor_is_discrete) {
    x_vals <- df$predictor_numeric
    x_labels <- levels(factor(df$predictor))
  } else {
    x_vals <- df$predictor
    x_labels <- waiver()
  }

  p <- ggplot(df, aes(x = predictor_numeric, y =
as.numeric(response))) +
    geom_jitter(width = 0.1, height = 0.1, alpha = 0.5, color =
"orange") +
    theme_bw() +
    labs(title = paste("Podmienené pravdepodobnosti pre",
response_name, "podľa", predictor_name),
         x = paste(predictor_name, "(Prediktor)"),
         y = paste(response_name, "(Odozva)")) +
    scale_y_continuous(breaks = seq_along(response_levels),
labels = response_levels) +
    scale_x_continuous(breaks = if (predictor_is_discrete)
seq_along(x_labels) else waiver(),
                       labels = if (predictor_is_discrete)
x_labels else waiver())

  if (!predictor_is_discrete) {
    centers <- unique(density_df$x_center)
    p <- p + geom_vline(xintercept = centers, linetype =
"dashed", color = "grey50")
  }

  if (ordinal) {
    p <- p + geom_segment(
      data = density_df,
      aes(x = x_center_numeric, xend = x_end, y = y, yend = y,
color = d, linetype = method),
      size = 1,
      inherit.aes = FALSE
    ) +
      geom_point(
        data = density_df,
        aes(x = x_end, y = y, color = d, shape = method),
        size = 2,
        inherit.aes = FALSE
      )

    sections <- unique(density_df$section)
    methods <- unique(density_df$method)

    for (s in sections) {
      for (m in methods) {
        sub_density <- density_df %>% dplyr::filter(section == s,
method == m) %>% dplyr::arrange(y)
        if (nrow(sub_density) >= 2) {
          p <- p + geom_path(
            data = sub_density,
            aes(x = x_end, y = y, group = interaction(section,
method)),
            color = "darkorchid",
            linewidth = 1,
            inherit.aes = FALSE
          )
        }
      }
    }
  } else {
    p <- p + geom_segment(
      data = density_df,
      aes(x = x_center_numeric, xend = x_end, y = y, yend = y,
color = d, linetype = method),
      size = 1,
      inherit.aes = FALSE
    ) +
      geom_point(
        data = density_df,
        aes(x = x_end, y = y, color = d, shape = method),
        size = 2,
        inherit.aes = FALSE
```

```
      )
  }

  p <- p +
    scale_linetype_manual(values = c("KDE" = "solid", "Normal" =
"dashed", "Empirical" = "solid")) +
    scale_shape_manual(values = c("KDE" = 15, "Normal" = 17,
"Empirical" = 15)) +
    guides(
      linetype = guide_legend(title = "Metóda"),
      shape = guide_legend(title = "Metóda")
    )

  summary_gt <- NULL
  if (!is.null(density_df) && nrow(density_df) > 0) {
    probability_summary <- density_df %>%
      dplyr::mutate(
        Section = as.character(section),
        Category = as.character(Category),
        Probability = round(prob, 4)
      ) %>%
      dplyr::select(Section, Category, Probability) %>%
      tidyr::pivot_wider(
        names_from = Category,
        values_from = Probability,
        values_fill = as.list(setNames(rep(0,
length(unique(density_df$Category))),
unique(density_df$Category)))
      )

    colnames(probability_summary) <-
as.character(colnames(probability_summary))

    summary_gt <- apply_dark_gt_theme(
      gt::gt(probability_summary) %>%
        gt::tab_header(
          title = gt::md("**Estimated Conditional
Probabilities**"),
          subtitle = "Across predictor sections"
        ) %>%
        gt::cols_label(.list = setNames(
          paste0("Y = ", colnames(probability_summary)[-1]),
          colnames(probability_summary)[-1]
        )) %>%
        gt::cols_width(everything() ~ gt::px(140)) %>%
        gt::opt_table_font(font = "Arial") %>%
        standardize_gt_table()
    )
  }

  return(list(
    plot = p,
    summary_gt = summary_gt
  ))
}

model_conditional_densities <- function(data, selected_variables,
n_breaks = 5, density_scaling = 2000, ordinal = FALSE, mean_curve
= TRUE, quantiles = NULL, mean_poly_degree = 1,
quantile_poly_degree = 1, normal_density = TRUE, kernel_density =
TRUE, bw_scale = NULL) {

  response_var <- selected_variables[1]
  predictor_var <- selected_variables[2]

  if (!(response_var %in% names(data)) || !(predictor_var %in%
names(data))) {
    stop("Premenne nie su v datach!")
  }

  df <- data[, c(predictor_var, response_var)]
  var_types <- identify_variables(df)
  colnames(df) <- c("predictor", "response")

  if (is.character(df$predictor)) {
    df$predictor <- factor(df$predictor)
  }

  # Atributy pre prediktor a odozvu
  attr(df, "response_var") <- response_var
  attr(df, "predictor_var") <- predictor_var

  if (response_var %in% var_types$Diskretne) {

    message(paste("Odozva", response_var, "bola identifikovana
ako diskretna."))
    df$response <- as.factor(df$response)
    model_conditional_discrete_densities(df, n_breaks,
density_scaling, ordinal)

  } else if (response_var %in% var_types$Spojite) {

    message(paste("Odozva", response_var, "bola identifikovana
ako spojita."))
    model_conditional_continuous_densities(df, n_breaks,
density_scaling, mean_curve, quantiles, mean_poly_degree,
quantile_poly_degree, normal_density, kernel_density, bw_scale)


    } else {

    stop("Nebolo mozne identifikovat typ odozvy.")
  }
}
```