```r
# Load required libraries for survival analysis, machine learning, and visualizati
on
library(readxl)        # For reading Excel files
library(survival)      # For survival analysis functions
library(cmprsk)        # For competing risks analysis
library(randomForestSRC) # For Random Survival Forests
library(riskRegression) # For risk prediction models and evaluation
library(prodlim)       # For non-parametric survival estimation
library(ggplot2)       # For data visualization
library(DALEX)         # For model explainability and interpretation
library(tidyr)         # For data tidying and transformation
library(dplyr)         # For data manipulation
library(gridExtra)     # For arranging multiple plots in a grid
library(tibble)        # For modern data frames

# Function to create distribution plots for variables
create_distribution_plots <- function(data, plot_config, ncol = 3) {
  plot_list <- list()

  # Loop through each variable in the configuration
  for (var in names(plot_config)) {
    config <- plot_config[[var]]

    # Create histogram for continuous variables
    if (config$type == "histogram") {
      p <- ggplot(data, aes(x = .data[[var]])) +
        geom_histogram(bins = 30, fill = config$fill, alpha = 0.7, color = "white"
) +
        labs(title = config$title, x = config$xlab, y = "Frequency") +
        theme_minimal() +
        theme(plot.title = element_text(size = 10, face = "bold"))
    }
    # Create bar plot for categorical variables
    else if (config$type == "bar") {
      p <- ggplot(data, aes(x = factor(.data[[var]]))) +
        geom_bar(fill = config$fill, alpha = 0.7, color = "white") +
        labs(title = config$title, x = config$xlab, y = "Count") +
        theme_minimal() +
        theme(plot.title = element_text(size = 10, face = "bold"),
              axis.text.x = element_text(angle = 45, hjust = 1))
    }
```

```r
    plot_list[[var]] <- p
  }

  # Calculate grid dimensions and arrange plots
  n_plots <- length(plot_list)
  nrow <- ceiling(n_plots / ncol)

  grid.arrange(grobs = plot_list, ncol = ncol, nrow = nrow)
}

# Function to generate synthetic health data for simulation studies
generate_health_data <- function(n, seed = 1234) {
  set.seed(seed)

  # Generate covariates with realistic constraints
  age      <- pmax(30, pmin(rnorm(n, 60, 10), 90))
  bmi      <- pmax(18, pmin(rnorm(n, 28, 4), 45))
  smoking  <- sample(0:1, n, replace = TRUE, prob = c(0.7, 0.3))
  hba1c    <- pmax(4.5, pmin(rnorm(n, 7.0, 1.2), 12.0))

  # Define risk scores with U-shaped effects for BMI and HbA1c
  # Cause 1 (event of interest): liver-related death
  lp1 <- 0.002 * age +
    0.06 * (bmi - 28)^2 +
    0.15 * (hba1c - 7)^2 +
    0.5 * smoking

  # Cause 2 (competing event): death from other causes
  lp2 <- 0.017 * age +
    0.03 * abs(bmi - 26) +
    0.3 * smoking

  # Generate event times from Weibull distribution
  shape <- 2.0
  scale_factor <- 90

  time_1 <- rweibull(n, shape = shape, scale = scale_factor * exp(-lp1 / shape))
  time_2 <- rweibull(n, shape = shape, scale = scale_factor * exp(-lp2 / shape))

  # Determine observed time and event status
  true_time <- pmin(time_1, time_2)
  obs_time <- runif(n, min = 15, max = 80)
```

```r
  time <- pmin(true_time, obs_time)
  status <- ifelse(true_time <= obs_time,
                   ifelse(time_1 < time_2, 1, 2),
                   0)

  # Apply administrative censoring at 80 months
  time <- pmin(time, 80)
  time <- round(time, 0)

  # Return final dataset
  data.frame(
    time = time,
    status = as.integer(status),
    age = round(age, 0),
    bmi = round(bmi, 2),
    smoking = smoking,
    hba1c = round(hba1c, 2)
  )
}

# Comprehensive function to calculate and compare model performance
calculate_model_performance <- function(models,
                                        data,
                                        time_var = "SurvMonths",
                                        event_var = "Event",
                                        times = seq(6, 80, 6),
                                        title_suffix = "",
                                        fgr_model = NULL,
                                        rsf_model = NULL,
                                        csc_model1 = NULL,
                                        csc_model2 = NULL,
                                        B = 0,
                                        split.method = "none") {

  # Create formula for risk regression
  formula_str <- paste0("Hist(", time_var, ", ", event_var, ") ~ 1")
  formula_obj <- as.formula(formula_str)
  print(formula_obj)

  # Calculate performance scores using riskRegression package
  score_result <- Score(
```

```
  object = models,
  formula = formula_obj,
  data = data,
  times = times,
  plots = "calibration",
  summary = "risks",
  se.fit = TRUE,
  B = B,
  split.method = split.method
)

# Identify available models and assign colors for plotting
available_models <- c()
model_colors <- c()

if (!is.null(fgr_model)) {
  available_models <- c(available_models, "Fine-Gray")
  model_colors <- c(model_colors, "Fine-Gray" = "#E41A1C")
}
if (!is.null(rsf_model)) {
  available_models <- c(available_models, "RSF Model")
  model_colors <- c(model_colors, "RSF Model" = "#377EB8")
}
if (!is.null(csc_model1) && !is.null(csc_model2)) {
  available_models <- c(available_models, "CSC Model")
  model_colors <- c(model_colors, "CSC Model" = "#4DAF4A")
}

# Always include Aalen-Johansen as reference
available_models <- c(available_models, "Aalen-Johansen")
model_colors <- c(model_colors, "Aalen-Johansen" = "#000000")

# Calculate Cumulative Incidence Functions (CIF) for each model
cif_data_list <- list()

# --- AALEN-JOHANSEN CIF (Non-parametric reference) ---
aj_formula <- as.formula(paste0("Hist(", time_var, ", ", event_var, ") ~ 1"))
aj <- prodlim::prodlim(aj_formula, data = data)

# Cause 1 CIF
aj_cause1 <- data.frame(
  Time = aj$time,
```

```
    CIF = aj$cuminc$`1`,
    Model = "Aalen-Johansen",
    Cause = "Cause 1"
  )
  aj_cause1_interp <- approx(aj_cause1$Time, aj_cause1$CIF, xout = times, rule = 2
)$y

  # Cause 2 CIF
  aj_cause2 <- data.frame(
    Time = aj$time,
    CIF = aj$cuminc$`2`,
    Model = "Aalen-Johansen",
    Cause = "Cause 2"
  )
  aj_cause2_interp <- approx(aj_cause2$Time, aj_cause2$CIF, xout = times, rule = 2
)$y

  cif_data_list$AJ <- data.frame(
    Time = rep(times, 2),
    CIF = c(aj_cause1_interp, aj_cause2_interp),
    Model = "Aalen-Johansen",
    Cause = rep(c("Cause 1", "Cause 2"), each = length(times))
  )

  # --- RSF CIF ---
  if (!is.null(rsf_model)) {
    pred_rsf <- predict(rsf_model, newdata = data)
    time_points_rsf <- pred_rsf$time.interest
    cif_cause1_rsf <- apply(pred_rsf$cif[,,1], 2, mean)
    cif_cause2_rsf <- apply(pred_rsf$cif[,,2], 2, mean)

    cif_cause1_rsf_interp <- approx(time_points_rsf, cif_cause1_rsf, xout = times,
rule = 2)$y
    cif_cause2_rsf_interp <- approx(time_points_rsf, cif_cause2_rsf, xout = times,
rule = 2)$y

    cif_data_list$RSF <- data.frame(
      Time = rep(times, 2),
      CIF = c(cif_cause1_rsf_interp, cif_cause2_rsf_interp),
      Model = "RSF Model",
      Cause = rep(c("Cause 1", "Cause 2"), each = length(times))
    )
```

```
  }

  # --- FGR CIF ---
  if (!is.null(fgr_model)) {
    original_formula <- formula(fgr_model)
    fgr1 <- FGR(formula = original_formula, data = data, cause = 1)
    fgr2 <- FGR(formula = original_formula, data = data, cause = 2)

    pred_fgr1 <- predict(fgr1, newdata = data, times = times, cause = 1)
    pred_fgr2 <- predict(fgr2, newdata = data, times = times, cause = 2)
    cif_cause1_fgr <- colMeans(pred_fgr1)
    cif_cause2_fgr <- colMeans(pred_fgr2)

    cif_data_list$FGR <- data.frame(
      Time = rep(times, 2),
      CIF = c(cif_cause1_fgr, cif_cause2_fgr),
      Model = "Fine-Gray",
      Cause = rep(c("Cause 1", "Cause 2"), each = length(times))
    )
  }

  # --- CSC CIF ---
  if (!is.null(csc_model1) && !is.null(csc_model2)) {
    # CSC CIF estimation for cause 1
    pred_csc1 <- predict(csc_model1, newdata = data, times = times, type = "absRis
k")

    if (inherits(pred_csc1, "riskRegression")) {
      pred_csc1_mat <- as.matrix(pred_csc1)
      cif_cause1_csc <- colMeans(pred_csc1_mat, na.rm = TRUE)
    } else if (is.list(pred_csc1) && "absRisk" %in% names(pred_csc1)) {
      pred_csc1_vec <- as.numeric(unlist(pred_csc1$absRisk))
      n_obs <- nrow(data)
      n_times <- length(times)
      expected_length <- n_obs * n_times
      if (length(pred_csc1_vec) == expected_length) {
        pred_csc1_mat <- matrix(pred_csc1_vec, nrow = n_obs, ncol = n_times, byrow
= FALSE)
        cif_cause1_csc <- colMeans(pred_csc1_mat, na.rm = TRUE)
      } else {
        pred_csc1_mat <- matrix(pred_csc1_vec[1:expected_length], nrow = n_obs, nc
ol = n_times, byrow = FALSE)
```

```
        cif_cause1_csc <- colMeans(pred_csc1_mat, na.rm = TRUE)
      }
    } else if (is.matrix(pred_csc1) || is.data.frame(pred_csc1)) {
      cif_cause1_csc <- colMeans(as.matrix(pred_csc1), na.rm = TRUE)
    } else {
      cif_cause1_csc <- rep(NA, length(times))
    }

    # Repeat for cause 2
    pred_csc2 <- predict(csc_model2, newdata = data, times = times, type = "absRis
k")

    if (inherits(pred_csc2, "riskRegression")) {
      pred_csc2_mat <- as.matrix(pred_csc2)
      cif_cause2_csc <- colMeans(pred_csc2_mat, na.rm = TRUE)
    } else if (is.list(pred_csc2) && "absRisk" %in% names(pred_csc2)) {
      pred_csc2_vec <- as.numeric(unlist(pred_csc2$absRisk))
      n_obs <- nrow(data)
      n_times <- length(times)
      expected_length <- n_obs * n_times
      if (length(pred_csc2_vec) == expected_length) {
        pred_csc2_mat <- matrix(pred_csc2_vec, nrow = n_obs, ncol = n_times, byrow
= FALSE)
        cif_cause2_csc <- colMeans(pred_csc2_mat, na.rm = TRUE)
      } else {
        pred_csc2_mat <- matrix(pred_csc2_vec[1:expected_length], nrow = n_obs, nc
ol = n_times, byrow = FALSE)
        cif_cause2_csc <- colMeans(pred_csc2_mat, na.rm = TRUE)
      }
    } else if (is.matrix(pred_csc2) || is.data.frame(pred_csc2)) {
      cif_cause2_csc <- colMeans(as.matrix(pred_csc2), na.rm = TRUE)
    } else {
      cif_cause2_csc <- rep(NA, length(times))
    }

    cif_cause1_csc[1] <- ifelse(is.na(cif_cause1_csc[1]), 0, cif_cause1_csc[1])
    cif_cause2_csc[1] <- ifelse(is.na(cif_cause2_csc[1]), 0, cif_cause2_csc[1])

    cif_data_list$CSC <- data.frame(
      Time = rep(times, 2),
      CIF = c(cif_cause1_csc, cif_cause2_csc),
      Model = "CSC Model",
```

```
      Cause = rep(c("Cause 1", "Cause 2"), each = length(times))
    )
  }

  # Combine all CIF data
  if (length(cif_data_list) == 0) {
    stop("No CIF data generated. Check model predictions.")
  }

  cif_data <- do.call(rbind, cif_data_list) %>%
    mutate(Cause_Model = paste(Cause, Model, sep = "-"))

  # 1. AUC PLOT - Show only available models
  auc_plot <- ggplot(score_result$AUC$score, aes(x = times, y = AUC, color = model
)) +
    geom_line(linewidth = 1.2) +
    geom_ribbon(aes(ymin = lower, ymax = upper, fill = model), alpha = 0.2, linety
pe = 0) +
    labs(x = "Time (Months)", y = "Area Under Curve (AUC)") +
    theme_minimal(base_size = 14) +
    theme(legend.position = "top",
          panel.grid.minor = element_blank(),
          legend.title = element_blank()) +
    scale_color_manual(values = model_colors) +
    scale_fill_manual(values = model_colors) +
    ylim(0, 1) + xlim(0, max(times))

  # 2. BRIER SCORE PLOT - Show only available models
  brier_plot <- ggplot(score_result$Brier$score, aes(x = times, y = Brier, color =
model)) +
    geom_line(linewidth = 1.2) +
    geom_ribbon(aes(ymin = lower, ymax = upper, fill = model), alpha = 0.2, linety
pe = 0) +
    labs(x = "Time (Months)", y = "Brier Score") +
    theme_minimal(base_size = 14) +
    theme(legend.position = "top",
          legend.title = element_blank()) +
    scale_color_manual(values = model_colors) +
    scale_fill_manual(values = model_colors) +
    ylim(0, max(score_result$Brier$score$Brier, na.rm = TRUE) * 1.1) +
    xlim(0, max(times))
```

```r
  # 3. CIF PLOT - Show only available models
  cif_plot <- ggplot(cif_data, aes(x = Time, y = CIF, color = Model, linetype = Ca
use)) +
    geom_line(size = 1.2) +
    labs(x = "Time (Months)", y = "Cumulative Incidence") +
    theme_minimal(base_size = 14) +
    theme(legend.position = "top",
          legend.box = "vertical") +
    scale_color_manual(values = model_colors) +
    scale_linetype_manual(values = c("Cause 1" = "solid", "Cause 2" = "dashed")) +
    scale_x_continuous(limits = c(0, max(times)), breaks = seq(0, max(times), 10))
+
    scale_y_continuous(limits = c(0, 1)) +
    guides(color = guide_legend(title = "Model"),
           linetype = guide_legend(title = "Cause"))

  # 4. PERFORMANCE TABLE - Include only available models
  auc_df <- score_result$AUC$score
  brier_df <- score_result$Brier$score

  performance_table <- data.frame(Time = times)

  # Add AUC and Brier scores for available models
  if ("Fine-Gray" %in% available_models && "Fine-Gray" %in% auc_df$model) {
    performance_table$FG_AUC <- round(auc_df$AUC[auc_df$model == "Fine-Gray"], 3)
    performance_table$FG_Brier <- round(brier_df$Brier[brier_df$model == "Fine-Gra
y"], 3)
  }

  if ("RSF Model" %in% available_models && "RSF Model" %in% auc_df$model) {
    performance_table$RSF_AUC <- round(auc_df$AUC[auc_df$model == "RSF Model"], 3)
    performance_table$RSF_Brier <- round(brier_df$Brier[brier_df$model == "RSF Mod
el"], 3)
  }

  if ("CSC Model" %in% available_models && "CSC Model" %in% auc_df$model) {
    performance_table$CSC_AUC <- round(auc_df$AUC[auc_df$model == "CSC Model"], 3)
    performance_table$CSC_Brier <- round(brier_df$Brier[brier_df$model == "CSC Mod
el"], 3)
  }

  # Add Aalen-Johansen CIF values
```

```r
  performance_table$AJ_CIF1 <- round(cif_data_list$AJ$CIF[cif_data_list$AJ$Cause ==
= "Cause 1"], 3)
  performance_table$AJ_CIF2 <- round(cif_data_list$AJ$CIF[cif_data_list$AJ$Cause ==
= "Cause 2"], 3)

  # Add model CIF values for available models
  if (!is.null(fgr_model)) {
    performance_table$FG_CIF1 <- round(cif_data_list$FGR$CIF[cif_data_list$FGR$Cau
se == "Cause 1"], 3)
    performance_table$FG_CIF2 <- round(cif_data_list$FGR$CIF[cif_data_list$FGR$Cau
se == "Cause 2"], 3)
  }

  if (!is.null(rsf_model)) {
    performance_table$RSF_CIF1 <- round(cif_data_list$RSF$CIF[cif_data_list$RSF$Ca
use == "Cause 1"], 3)
    performance_table$RSF_CIF2 <- round(cif_data_list$RSF$CIF[cif_data_list$RSF$Ca
use == "Cause 2"], 3)
  }

  if (!is.null(csc_model1) && !is.null(csc_model2)) {
    performance_table$CSC_CIF1 <- round(cif_data_list$CSC$CIF[cif_data_list$CSC$Ca
use == "Cause 1"], 3)
    performance_table$CSC_CIF2 <- round(cif_data_list$CSC$CIF[cif_data_list$CSC$Ca
use == "Cause 2"], 3)
  }

  # Calibration plot function
  calib_plot <- function() {
    plotCalibration(score_result)
  }

  # Print used models (optional)
  cat("Used models:", paste(available_models, collapse = ", "), "\n")

  # Return comprehensive results
  list(score = score_result, auc_plot = auc_plot, brier_plot = brier_plot,
       cif_plot = cif_plot, calib_plot = calib_plot, performance_table = performan
ce_table,
       cif_data = cif_data, used_models = available_models)
}
```

```r
# Comprehensive DALEX analysis with VIMP interactions
complete_dalex_analysis_with_interactions <- function(rsf_model, data, n_features
= 15, time_point = 365) {

  # Create DALEX explainer for RSF model
  predict_function <- function(model, newdata) {
    pred <- predict(model, newdata = newdata)
    time_idx <- which.min(abs(pred$time.interest - time_point))
    return(pred$cif[, time_idx, 1])
  }

  explainer_rsf <- DALEX::explain(
    model = rsf_model,
    data = data[predictor_vars],
    y = predict_function(rsf_model, data),
    predict_function = predict_function,
    label = "RSF Model",
    verbose = FALSE
  )

  cat("DALEX explainer created...\n")

  # 1. VARIABLE IMPORTANCE (DALEX permutation-based)
  cat("Calculating Variable Importance...\n")
  vi <- DALEX::model_parts(explainer_rsf,
                           type = "variable_importance",
                           B = 20)

  # Process importance results
  vi_result <- as.data.frame(vi)
  vi_filtered <- vi_result[vi_result$variable != "_full_model_" & vi_result$variab
le != "_baseline_", ]
  vi_summary <- aggregate(dropout_loss ~ variable, data = vi_filtered, FUN = mean)
  vi_summary <- vi_summary[order(-vi_summary$dropout_loss), ]
  top_features <- head(vi_summary$variable, min(n_features, nrow(vi_summary)))

  importance_plot <- plot(vi) +
    labs(title = "DALEX Variable Importance",
         subtitle = "Permutation-based Importance") +
    theme_minimal()

  # 2. BREAK DOWN PLOTS with CIF curves for individual predictions
```

```r
cat("Creating Break Down plots with CIF curves...\n")
break_down_plots <- list()
cif_break_down_plots <- list()

for (i in 1:min(3, nrow(data))) {
  # Break Down plot for individual prediction explanation
  bd <- DALEX::predict_parts(explainer_rsf,
                             new_observation = data[i, predictor_vars],
                             type = "break_down")
  break_down_plots[[paste("Observation", i)]] <- plot(bd) +
    labs(title = paste("Break Down - Observation", i)) +
    theme_minimal()

  # CIF curve for the same observation
  pred_individual <- predict(rsf_model, newdata = data[i, predictor_vars])

  cif_data <- data.frame(
    Time = pred_individual$time.interest,
    CIF1 = pred_individual$cif[1, , 1],  # Event 1
    CIF2 = pred_individual$cif[1, , 2]   # Event 2
  )

  max_cif <- max(cif_data$CIF1, cif_data$CIF2)

  cif_plot <- ggplot(cif_data, aes(x = Time)) +
    geom_line(aes(y = CIF1, color = "Event 1"), size = 1) +
    geom_line(aes(y = CIF2, color = "Event 2"), size = 1) +
    geom_vline(xintercept = time_point, linetype = "dashed", color = "red",
               size = 1, alpha = 0.7) +
    annotate("text", x = time_point, y = max_cif * 0.9,
             label = paste("Time =", time_point), hjust = -0.1, color = "red") +
    scale_color_manual(values = c("Event 1" = "blue", "Event 2" = "red")) +
    labs(title = paste("CIF Curves - Observation", i),
         x = "Time",
         y = "Cumulative Incidence Function",
         color = "Event Type") +
    theme_minimal() +
    theme(legend.position = "bottom")

  cif_break_down_plots[[paste("Observation", i)]] <- cif_plot
}
```

```r
# 3. PARTIAL DEPENDENCE PLOTS for marginal effects
cat("Creating Partial Dependence Plots...\n")
pdp_plots <- list()

for (feature in head(top_features, 4)) {
  pdp <- DALEX::model_profile(explainer_rsf, variables = feature)
  pdp_plots[[feature]] <- plot(pdp) +
    labs(title = paste("PDP -", feature)) +
    theme_minimal()
}

# 4. VIMP INTERACTIONS using randomForestSRC
cat("Calculating VIMP Interactions...\n")

# Use existing VIMP code for interaction detection
int_features <- top_features[1:min(6, length(top_features))]

vimp_interactions <- randomForestSRC::find.interaction(
  rsf_model,
  xvar.names = int_features,
  method = "vimp",
  importance = "permute",
  nrep = 5,
  na.action = "na.omit",
  seed = -1,
  verbose = FALSE
)

# Process VIMP interaction results
vimp_event1 <- vimp_interactions$event.1

vimp_df <- as.data.frame(vimp_event1) %>%
  rownames_to_column("Pair") %>%
  mutate(
    Var1 = sapply(strsplit(Pair, ":"), `[`, 1),
    Var2 = sapply(strsplit(Pair, ":"), `[`, 2),
    Interaction_Strength = Difference,
    Interaction_Type = case_when(
      Difference > 0.02 ~ "Strong Synergy",
      Difference > 0.01 ~ "Medium Synergy",
      Difference > 0 ~ "Weak Synergy",
      Difference > -0.01 ~ "Neutral",
```

```
      Difference > -0.02 ~ "Weak Redundancy",
      TRUE ~ "Strong Redundancy"
    )
  ) %>%
  arrange(desc(abs(Difference)))

# VIMP Heatmap visualization
vimp_heatmap <- ggplot(vimp_df, aes(x = Var2, y = Var1, fill = Difference)) +
  geom_tile(color = "white", linewidth = 0.5) +
  geom_text(aes(label = round(Difference, 3)), size = 3, color = "black") +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red",
                       midpoint = 0, name = "VIMP Difference") +
  labs(title = "VIMP-based Feature Interactions",
       subtitle = "Red = Synergy, Blue = Redundancy") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# 5. SHAPLEY VALUES for local explanations with CIF curves
cat("Calculating Shapley Values with CIF curves...\n")
shapley_plots <- list()
cif_shapley_plots <- list()

for (i in 1:min(2, nrow(data))) {
  # Shapley values for local interpretability
  shap <- DALEX::predict_parts(explainer_rsf,
                               new_observation = data[i, predictor_vars],
                               type = "shap",
                               B = 10)
  shapley_plots[[paste("Shapley", i)]] <- plot(shap) +
    labs(title = paste("Shapley Values - Observation", i)) +
    theme_minimal()

  # CIF curve for the same observation (for Shapley)
  pred_individual <- predict(rsf_model, newdata = data[i, predictor_vars])

  cif_data <- data.frame(
    Time = pred_individual$time.interest,
    CIF1 = pred_individual$cif[1, , 1],  # Event 1
    CIF2 = pred_individual$cif[1, , 2]   # Event 2
  )

  max_cif <- max(cif_data$CIF1, cif_data$CIF2)
```

```
  cif_plot_shap <- ggplot(cif_data, aes(x = Time)) +
    geom_line(aes(y = CIF1, color = "Event 1"), size = 1) +
    geom_line(aes(y = CIF2, color = "Event 2"), size = 1) +
    geom_vline(xintercept = time_point, linetype = "dashed", color = "red",
               size = 1, alpha = 0.7) +
    annotate("text", x = time_point, y = max_cif * 0.9,
             label = paste("Time =", time_point), hjust = -0.1, color = "red") +
    scale_color_manual(values = c("Event 1" = "blue", "Event 2" = "red")) +
    labs(title = paste("CIF Curves (Shapley) - Observation", i),
         x = "Time",
         y = "Cumulative Incidence Function",
         color = "Event Type") +
    theme_minimal() +
    theme(legend.position = "bottom")

  cif_shapley_plots[[paste("Observation", i)]] <- cif_plot_shap
}

# Print all plots
print(importance_plot)
print(vimp_heatmap)

for (plot in break_down_plots) print(plot)
for (plot in cif_break_down_plots) print(plot)
for (plot in pdp_plots) print(plot)
for (plot in shapley_plots) print(plot)
for (plot in cif_shapley_plots) print(plot)

# Generate detailed analysis report
cat("\n")
cat("================================================\n")
cat("DALEX ANALYSIS WITH VIMP INTERACTIONS\n")
cat("================================================\n")

cat("Top 5 Feature Importance:\n")
print(head(vi_summary, 5))

significant_vimp <- vimp_df %>% filter(abs(Difference) > 0.01)
cat("\nSignificant VIMP Interactions (|Difference| > 0.01):\n")
if(nrow(significant_vimp) > 0) {
```

```
    print(head(significant_vimp[, c("Pair", "Difference", "Interaction_Type")], 5)
)
  } else {
    cat("No significant interactions found.\n")
  }

  # Return comprehensive analysis results
  return(list(
    explainer = explainer_rsf,
    variable_importance = vi,
    vimp_interactions = list(
      event1 = vimp_event1,
      significant = significant_vimp,
      all = vimp_df
    ),
    plots = list(
      importance = importance_plot,
      vimp_heatmap = vimp_heatmap,
      break_down = break_down_plots,
      cif_break_down = cif_break_down_plots,
      pdp = pdp_plots,
      shapley = shapley_plots,
      cif_shapley = cif_shapley_plots
    )
  ))
}

# Define common variables for all sample sizes
survival_vars <- c("time", "status")
time_var <- survival_vars[1]
event_var <- survival_vars[2]

continuous_vars <- c("age", "bmi", "hba1c")
categorical_vars <- c("smoking")
predictor_vars <- c(continuous_vars, categorical_vars)
formula_rhs <- paste(predictor_vars, collapse = " + ")
hist_formula <- as.formula(paste("Hist(", time_var, ", ", event_var, ") ~", formul
a_rhs))
surv_formula <- as.formula(paste("Surv(", time_var, ", ", event_var, ") ~", formul
a_rhs))

# === ANALYSIS FOR n=500 ===
```

```r
set.seed(1234)
n_500 <- 500
train_data_500 <- generate_health_data(n_500 * 0.8)
test_data_500 <- generate_health_data(n_500 * 0.2)
max_time_500 <- min(72, max(test_data_500[[time_var]], na.rm = TRUE))

# Train models for n=500
fg_model_500 <- FGR(hist_formula, data = train_data_500, cause = 1)
rsf_model_500 <- rfsrc(surv_formula, data = train_data_500, ntree = 100, cause = 1
, importance = TRUE)
csc_model1_500 <- CSC(hist_formula, data = train_data_500, cause = 1)
csc_model2_500 <- CSC(hist_formula, data = train_data_500, cause = 2)

# Calculate performance for n=500
results_500 <- calculate_model_performance(
  models = list(
    "Fine-Gray" = fg_model_500,
    "RSF Model" = rsf_model_500,
    "CSC Model" = csc_model1_500
  ),
  data = test_data_500,
  time_var = time_var,
  event_var = event_var,
  times = seq(2, 72, 2),
  title_suffix = "Test Set - n=500",
  fgr_model = fg_model_500,
  rsf_model = rsf_model_500,
  csc_model1 = csc_model1_500,
  csc_model2 = csc_model2_500,
  split.method = "none"
)

# Print results for n=500
print(results_500$auc_plot)
print(results_500$brier_plot)
print(results_500$cif_plot)
results_500$calib_plot()
kable(results_500$performance_table)
dalex_with_vimp_500 <- complete_dalex_analysis_with_interactions(rsf_model_500, te
st_data_500, time_point = 24)

# === ANALYSIS FOR n=2500 ===
```

```r
set.seed(1234)
n_2500 <- 2500
train_data_2500 <- generate_health_data(n_2500 * 0.8)
test_data_2500 <- generate_health_data(n_2500 * 0.2)
max_time_2500 <- min(72, max(test_data_2500[[time_var]], na.rm = TRUE))

# Train models for n=2500
fg_model_2500 <- FGR(hist_formula, data = train_data_2500, cause = 1)
rsf_model_2500 <- rfsrc(surv_formula, data = train_data_2500, ntree = 100, cause =
1, importance = TRUE)
csc_model1_2500 <- CSC(hist_formula, data = train_data_2500, cause = 1)
csc_model2_2500 <- CSC(hist_formula, data = train_data_2500, cause = 2)

# Calculate performance for n=2500
results_2500 <- calculate_model_performance(
  models = list(
    "Fine-Gray" = fg_model_2500,
    "RSF Model" = rsf_model_2500,
    "CSC Model" = csc_model1_2500
  ),
  data = test_data_2500,
  time_var = time_var,
  event_var = event_var,
  times = seq(2, 66, 2),
  title_suffix = "Test Set - n=2500",
  fgr_model = fg_model_2500,
  rsf_model = rsf_model_2500,
  csc_model1 = csc_model1_2500,
  csc_model2 = csc_model2_2500,
  split.method = "none"
)

# Print results for n=2500
print(results_2500$auc_plot)
print(results_2500$brier_plot)
print(results_2500$cif_plot)
results_2500$calib_plot()
kable(results_2500$performance_table)
dalex_with_vimp_2500 <- complete_dalex_analysis_with_interactions(rsf_model_2500,
test_data_2500, time_point = 24)

# === ANALYSIS FOR n=5000 ===
```

```r
set.seed(1234)
n_5000 <- 5000
train_data_5000 <- generate_health_data(n_5000 * 0.8)
test_data_5000 <- generate_health_data(n_5000 * 0.2)
max_time_5000 <- min(72, max(test_data_5000[[time_var]], na.rm = TRUE))

# Train models for n=5000
fg_model_5000 <- FGR(hist_formula, data = train_data_5000, cause = 1)
rsf_model_5000 <- rfsrc(surv_formula, data = train_data_5000, ntree = 100, cause =
1, importance = TRUE)
csc_model1_5000 <- CSC(hist_formula, data = train_data_5000, cause = 1)
csc_model2_5000 <- CSC(hist_formula, data = train_data_5000, cause = 2)

# Calculate performance for n=5000
results_5000 <- calculate_model_performance(
  models = list(
    "Fine-Gray" = fg_model_5000,
    "RSF Model" = rsf_model_5000,
    "CSC Model" = csc_model1_5000
  ),
  data = test_data_5000,
  time_var = time_var,
  event_var = event_var,
  times = seq(2, 66, 2),
  title_suffix = "Test Set - n=5000",
  fgr_model = fg_model_5000,
  rsf_model = rsf_model_5000,
  csc_model1 = csc_model1_5000,
  csc_model2 = csc_model2_5000,
  split.method = "none"
)

# Print results for n=5000
print(results_5000$auc_plot)
print(results_5000$brier_plot)
print(results_5000$cif_plot)
results_5000$calib_plot()
kable(results_5000$performance_table)
dalex_with_vimp_5000 <- complete_dalex_analysis_with_interactions(rsf_model_5000,
test_data_5000, time_point = 24)

# Compare results across all sample sizes
```

```r
cat("=== n=500 RESULTS ===\n")
print(results_500$performance_table)
cat("\n=== n=2500 RESULTS ===\n")
print(results_2500$performance_table)
cat("\n=== n=5000 RESULTS ===\n")
print(results_5000$performance_table)
```

SEER DATA

```r
# Read and prepare the hepatocellular carcinoma dataset
df <- read_excel("C:/notebook/yarisanrisk/karaciger1.xlsx")
df <- data.frame(df)

# Data cleaning: remove unknown tumor grades and recent years for complete follow-
up
df <- df[df$SEERTumorGrade != 99 & !(df$Year_of_diagnosis %in% c(2021, 2022)), ]

# Remove diagnosis year column as it's no longer needed
df$Year_of_diagnosis <- NULL

# Define and convert categorical variables to factors
categorical_vars <- c("Sex",
                      "Race",
                      "MaritalStatus",
                      "Radiation",
                      "Mets_at_DX_bone",
                      "SEERStage",
                      "Chemotherapy",
                      "SEERTumorGrade")

for (var in categorical_vars) {
  df[[var]] <- as.factor(df[[var]])
}

# Check data structure
str(df)

# Convert numerical variables to appropriate format
df$Event <- as.numeric(df$Event)
df$SurvMonths <- as.numeric(df$SurvMonths)
df$Treatment_delay_days <- as.numeric(df$Treatment_delay_days)
df$TumorSize <- as.numeric(df$TumorSize)
df$Age <- as.numeric(df$Age)
```

```r
# Split data into training (80%) and testing (20%) sets
set.seed(123)
ind.split <- sample(1:nrow(df), round(nrow(df)*4/5), replace = FALSE)
train_data <- df[ind.split, ]
test_data <- df[-ind.split, ]

# Define survival analysis variables
survival_vars <- c("SurvMonths", "Event")
time_var <- survival_vars[1]  # Survival time in months
event_var <- survival_vars[2] # Event indicator (0=censored, 1=liver death, 2=othe
r death)

# Define predictor variables (EXCLUDING survival variables from predictors)
continuous_vars <- c("Treatment_delay_days", "TumorSize", "Age")
predictor_vars <- c(continuous_vars, categorical_vars)
formula_rhs <- paste(predictor_vars, collapse = " + ")

# Create survival formulas for different models
hist_formula <- as.formula(paste("Hist(", time_var, ", ", event_var, ") ~", formul
a_rhs))
surv_formula <- as.formula(paste("Surv(", time_var, ", ", event_var, ") ~", formul
a_rhs))

# Train competing risk models
# Fine-Gray model for subdistribution hazards
fg_model <- FGR(hist_formula, data = train_data, cause = 1)

# Random Survival Forest for flexible nonlinear modeling
rsf_model <- rfsrc(surv_formula, data = train_data, ntree = 100, cause = 1, import
ance = TRUE)

# Cause-Specific Cox models for each event type
csc_model1 <- CSC(hist_formula, data = train_data, cause = 1)  # Liver-related dea
th
csc_model2 <- CSC(hist_formula, data = train_data, cause = 2)  # Other causes deat
h

# Determine maximum evaluation time (capped at 72 months)
max_time <- min(72, max(test_data[[time_var]], na.rm = TRUE))

# Calculate comprehensive model performance
```

```r
resultsHCC <- calculate_model_performance(
  models = list(
    "Fine-Gray" = fg_model,
    "RSF Model" = rsf_model,
    "CSC Model" = csc_model1
  ),
  data = test_data,
  time_var = time_var,
  event_var = event_var,
  times = seq(2, 72, 2),  # Evaluate every 2 months up to 72 months
  title_suffix = "Test Set",
  fgr_model = fg_model,
  rsf_model = rsf_model,
  csc_model1 = csc_model1,
  csc_model2 = csc_model2,
  split.method = "none"
)

# Display performance results
print(resultsHCC$auc_plot)      # Time-dependent AUC plot
print(resultsHCC$brier_plot)    # Brier score plot
print(resultsHCC$cif_plot)      # Cumulative incidence functions
resultsHCC$calib_plot()         # Calibration plot
kable(resultsHCC$performance_table)  # Performance metrics table

# Conduct comprehensive model interpretability analysis
dalex_with_vimp <- complete_dalex_analysis_with_interactions(
  rsf_model,
  test_data,
  time_point = 24  # Analyze predictions at 24 months
)

dalex_with_vimp
```