R Functions

```r
#' Check Compatibility of Two Datasets for Comparison
#' Verifies if two datasets are compatible for comparison,
#' focusing on dimensions and variable names.
#' @param df1 The first data frame to be compared.
#' @param df2 The second data frame to be compared.
#' @return A list containing details about the compatibility of the datasets,
#' including information on dimension equality and common columns.
#' @export
#' @examples
#' check_compatibility(df1, df2)
check_compatibility <- function(df1, df2) {
  compatibility_results <- list()

  # Check for null datasets
  if (is.null(df1) || is.null(df2)) {
    return(list(compatible = FALSE, reason = "One or both datasets are null"))
  }

  # Check for dimension equality
  if (!all(dim(df1) == dim(df2))) {
    compatibility_results$dimensions_equal <- FALSE
  } else {
    compatibility_results$dimensions_equal <- TRUE
  }

  # Compare column names
  compatibility_results$common_columns <- intersect(names(df1), names(df2))
  compatibility_results$extra_in_df1 <- setdiff(names(df1), names(df2))
  compatibility_results$extra_in_df2 <- setdiff(names(df2), names(df1))

  compatibility_results$compatible <- compatibility_results$dimensions_equal &&
    length(compatibility_results$extra_in_df1) == 0 &&
    length(compatibility_results$extra_in_df2) == 0

  compatibility_results
}
#' Clean Dataset
#' Cleans the given dataset by handling missing values, removing duplicates,
```

```r
#' and potentially standardizing formats for selected variables.
#' @param df A data frame to be cleaned.
#' @param variables Optional; a vector of variable names to specifically clean.
#' If NULL, applies cleaning to all variables.
#' @param remove_duplicates Logical; whether to remove duplicate rows.
#' @param standardize_formats Logical; whether to standardize the formats (e.g., to lower case for char
#' @return A cleaned data frame.
#' @export
#' @examples
#' clean_dataset(df, variables = c("var1", "var2"), remove_duplicates = TRUE, standardize_formats = TRU
clean_dataset <- function(df, variables = NULL, remove_duplicates = TRUE, standardize_formats = TRUE) {
  # If no specific variables are specified, apply to all columns
  if (is.null(variables)) {
    variables <- names(df)
  }

  for (var in variables) {
    if (var %in% names(df)) {
      if (remove_duplicates) {
        df[[var]] <- df[[var]][!duplicated(df[[var]]), , drop = FALSE]
      }

      if (standardize_formats) {
        # Apply standardization only to character columns
        if (is.character(df[[var]])) {
          df[[var]] <- tolower(df[[var]])
        }
        # Additional format standardizations can be added here
      }
    } else {
      warning(paste("Variable", var, "not found in the dataset."))
    }
  }

  df
}
#' Compare Datasets by Group
#' Performs dataset comparisons based on specified grouping variables.
#' Useful for BY-group analysis in datasets.
#' @param df1 First dataset for comparison.
#' @param df2 Second dataset for comparison.
#' @param group_vars Variables used for grouping the data.
```

```r
#' @return A list of comparison results for each group.

#' @export

#' @examples

#' compare_by_group(df1, df2, group_vars = c("group_var1", "group_var2"))
compare_by_group <- function(df1, df2, group_vars) {
  if (!all(group_vars %in% names(df1)) || !all(group_vars %in% names(df2))) {
    stop("Grouping variables must be present in both datasets.")
  }

  # Splitting datasets by groups
  df1_split <- split(df1, df1[, group_vars, drop = FALSE])
  df2_split <- split(df2, df2[, group_vars, drop = FALSE])

  # Identifying unique groups in both datasets
  all_groups <- union(names(df1_split), names(df2_split))

  # Comparing each group
  results <- lapply(all_groups, function(group) {
    group_df1 <- df1_split[[group]] %>% dplyr::bind_rows()
    group_df2 <- df2_split[[group]] %>% dplyr::bind_rows()
    compare_datasets(group_df1, group_df2)
  })

  names(results) <- all_groups
  return(results)
}

#' Compare Two Datasets

#' This function performs a high-level comparison of two datasets.

#' It checks for basic compatibility in terms of dimensions and variable names,

#' and then performs detailed comparisons of variables and observations.

#' @param df1 A data frame representing the first dataset.

#' @param df2 A data frame representing the second dataset.

#' @return A list containing the results of the comparison.

#' @export

#' @examples

#' compare_datasets(df1, df2)
compare_datasets <- function(df1, df2) {
  if (is.null(df1) || is.null(df2)) {
    return("One or both datasets are null.")
  }

  dimensions_equal <- all(dim(df1) == dim(df2))
  dimension_message <- if (dimensions_equal) {
    "Both datasets have the same dimensions."
```

```
  } else {
    "Datasets have different dimensions."
  }

  common_cols <- intersect(names(df1), names(df2))
  extra_df1 <- setdiff(names(df1), names(df2))
  extra_df2 <- setdiff(names(df2), names(df1))

  col_name_message <- if (length(extra_df1) == 0 && length(extra_df2) == 0) {
    "Both datasets have the same column names."
  } else {
    paste(
      "Common Columns:", paste(common_cols, collapse = ", "),
      "\nExtra Columns in Dataset 1:", paste(extra_df1, collapse = ", "),
      "\nExtra Columns in Dataset 2:", paste(extra_df2, collapse = ", ")
    )
  }

  data_type_comparison <- sapply(common_cols, function(col) {
    type_df1 <- class(df1[[col]])
    type_df2 <- class(df2[[col]])

    if (type_df1 != type_df2) {
      return(paste("Column '", col, "' has different data types in the two datasets."))
    } else {
      return(NULL)
    }
  })

  # New section to check for missing values
  missing_values_comparison <- sapply(common_cols, function(col) {
    na_df1 <- sum(is.na(df1[[col]]))
    na_df2 <- sum(is.na(df2[[col]]))

    if (na_df1 > 0 || na_df2 > 0) {
      return(paste("Column '", col, "' has missing values. Dataset1:", na_df1, "missing. Dataset2:", na
    } else {
      return(NULL)
    }
  })

  result_messages <- c(
    dimension_message,
    col_name_message,
    data_type_comparison[!sapply(data_type_comparison, is.null)],
    missing_values_comparison[!sapply(missing_values_comparison, is.null)]
  )
  result_string <- paste(result_messages, collapse = "\n")

  return(result_string)
}

#' Compare Observations of Two Datasets
```

```r
#' Performs row-wise comparison to identify differences in data values between two datasets.
#' Highlights rows with differences.
#' @param df1 A data frame representing the first dataset.
#' @param df2 A data frame representing the second dataset.
#' @return A data frame showing the rows where the datasets differ.
#' @export
#' @examples
#' compare_observations(df1, df2)
compare_observations <- function(df1, df2) {
  if (nrow(df1) != nrow(df2)) {
    stop("The datasets have different numbers of rows.")
  }

  common_cols <- intersect(names(df1), names(df2))

  discrepancy_counts <- integer(length = length(common_cols))
  names(discrepancy_counts) <- common_cols
  row_differences <- list()

  for (col in common_cols) {
    if (is.factor(df1[[col]]) || is.factor(df2[[col]])) {
      # Convert factors to characters to compare
      df1_col <- as.character(df1[[col]])
      df2_col <- as.character(df2[[col]])
    } else {
      df1_col <- df1[[col]]
      df2_col <- df2[[col]]
    }

    differences <- which(df1_col != df2_col)
    discrepancy_counts[col] <- length(differences)

    names(discrepancy_counts) <- common_cols
    print(discrepancy_counts)

    if (length(differences) > 0) {
      row_differences[[col]] <- data.frame(
        Row = differences,
        Value_in_df1 = df1_col[differences],
        Value_in_df2 = df2_col[differences]
      )
    }
  }

  return(list(discrepancies = discrepancy_counts, details = row_differences))

  # Just before returning the result in compare_observations
  print("Final discrepancy counts:")
```

```r
  print(discrepancy_counts)

}

#' Compare Variables of Two Datasets

#' Compares the variables (columns) of two datasets in terms of names and data types.

#' Reports discrepancies in variables between the datasets.

#' @param df1 A data frame representing the first dataset.

#' @param df2 A data frame representing the second dataset.

#' @return A data frame containing the comparison results of the variables.

#' @export

#' @examples

#' compare_variables(df1, df2)
compare_variables <- function(df1, df2) {
  # Initialize a list to hold the results
  variable_comparisons <- list()

  # Compare column names
  variable_comparisons$common_columns <- intersect(names(df1), names(df2))
  variable_comparisons$extra_in_df1 <- setdiff(names(df1), names(df2))
  variable_comparisons$extra_in_df2 <- setdiff(names(df2), names(df1))

  # Compare data types for common columns
  common_cols <- variable_comparisons$common_columns
  data_type_comparisons <- lapply(common_cols, function(col) {
    list(column = col,
         type_df1 = class(df1[[col]]),
         type_df2 = class(df2[[col]]))
  })

  # Add data type comparisons to the results
  variable_comparisons$data_type_comparisons <- data_type_comparisons

  # Return the results
  variable_comparisons

  # Calculate the discrepancy count
  discrepancy_count <- length(variable_comparisons$extra_in_df1) + length(variable_comparisons$extra_in

  # Return the results
  list(discrepancies = discrepancy_count, details = variable_comparisons)

}

#' Convert Data Types of Variables in a Dataset

#' Converts the data types of specified variables in a dataset.
```

```r
#' This is useful for ensuring that variables are in the correct format
#' for analysis or comparison.
#' @param df A data frame containing the variables to be converted.
#' @param conversions A named list where names correspond to variable names
#' in the dataset, and values are the desired data types (e.g., 'numeric', 'factor').
#' @return A data frame with converted variable types.
#' @export
#' @examples
#' convert_data_types(df, conversions = list(var1 = 'numeric', var2 = 'factor'))
convert_data_types <- function(df, target_type) {
  # This is a simplistic version. A more robust version would handle various cases and data types.
  if (target_type == "character") {
    df[] <- lapply(df, as.character)
  } else if (target_type == "numeric") {
    df[] <- lapply(df, as.numeric)
  }
  df
}
#' Generate Visualization for Data Comparison
#' Creates a visual representation of data comparison results.
#' This can be particularly useful for showing discrepancies in variable distributions or counts.
#' @param comparison_results A list containing the results of dataset comparisons.
#' @return A plot object visualizing the comparison results.
#' @export
#' @examples
#' generate_comparison_visualization(comparison_results)
generate_comparison_visualization <- function(comparison_results) {
  if (!"ggplot2" %in% installed.packages()) {
    install.packages("ggplot2")
  }
  library(ggplot2)

  if (nrow(comparison_results) == 0) {
    warning("Empty dataset provided.")
    return(ggplot())  # Return an empty ggplot object
  }

  # Check if necessary columns are present
  required_columns <- c("Variable", "Discrepancies")
  if (!all(required_columns %in% names(comparison_results))) {
    warning("Data does not contain the required columns: Variable, Discrepancies")
    return(NULL)
```

```r
  }

  ggplot(data = comparison_results, aes(x = Variable, y = Discrepancies)) +
    geom_bar(stat = "identity") +
    theme_minimal() +
    labs(title = "Discrepancies per Variable", x = "Variable", y = "Count of Discrepancies")
}

#' Generate a Detailed Report of Dataset Comparison

#' Creates a detailed report outlining all the differences found in the comparison,

#' including variable differences, observation differences, and group-based discrepancies.

#' @param comparison_results A list containing the results of dataset comparisons.

#' @param output_format Format of the output ('text', 'html', or 'pdf').

#' @param file_name Name of the file to save the report to (applicable for 'html' and 'pdf' formats).

#' @return The detailed report, output format depends on 'output_format' parameter.

#' @export

#' @examples

#' generate_detailed_report(comparison_results, output_format = "text")
generate_detailed_report <- function(comparison_results, output_format = "text", file_name = "detailed_
  detailed_report <- paste("Detailed Comparison Report\n", "======================\n\n", sep = "")

  # Generate report content
  if (!is.null(comparison_results$VariableDifferences)) {
    detailed_report <- paste0(detailed_report, "Variable Differences:\n")
    detailed_report <- paste0(detailed_report, format(comparison_results$VariableDifferences), "\n\n")
  }

  if (!is.null(comparison_results$ObservationDifferences)) {
    detailed_report <- paste0(detailed_report, "Observation Differences:\n")
    for (col in names(comparison_results$ObservationDifferences)) {
      detailed_report <- paste0(detailed_report, "Column: ", col, "\n")
      detailed_report <- paste0(detailed_report, format(comparison_results$ObservationDifferences[[col]]
    }
  }

  # Output the report based on the specified format
  if (output_format == "text") {
    cat(detailed_report)
  } else if (output_format == "html") {
    rmarkdown::render(input = detailed_report, output_format = "html_document", output_file = paste0(fi
  } else if (output_format == "pdf") {
    rmarkdown::render(input = detailed_report, output_format = "pdf_document", output_file = paste0(fil
  } else {
    stop("Unsupported output format")
  }
}
```

```r
#' Generate a Summary Report of Dataset Comparison

#' Provides a summary of the comparison results, highlighting key points such as the number of differin

#' @param comparison_results A list containing the results of dataset comparisons.

#' @param detail_level The level of detail ('high', 'medium', 'low') for the summary.

#' @param output_format Format of the output ('text', 'html', or 'pdf').

#' @param file_name Name of the file to save the report to (applicable for 'html' and 'pdf' formats).

#' @return The summary report, output format depends on 'output_format' parameter.

#' @export

#' @examples

#' generate_summary_report(comparison_results, detail_level = "high", output_format = "text")
generate_summary_report <- function(comparison_results, detail_level = "high", output_format = "text", f
  summary_report <- paste("Summary Comparison Report\n", "======================\n\n", sep = "")

  # Generate summary based on the detail level
  if (!is.null(comparison_results$VariableDifferences)) {
    num_var_diffs <- length(comparison_results$VariableDifferences)
    summary_report <- paste0(summary_report, "Number of Variable Differences: ", num_var_diffs, "\n")
  }

  if (!is.null(comparison_results$ObservationDifferences)) {
    num_obs_diffs <- sum(sapply(comparison_results$ObservationDifferences, nrow))
    summary_report <- paste0(summary_report, "Total Number of Observation Differences: ", num_obs_diffs
  }
  # Output the summary based on the specified format
  if (output_format == "text") {
    cat(summary_report)
  } else if (output_format == "html") {
    rmarkdown::render(input = summary_report, output_format = "html_document", output_file = paste0(fil
  } else if (output_format == "pdf") {
    rmarkdown::render(input = summary_report, output_format = "pdf_document", output_file = paste0(file
  } else {
    stop("Unsupported output format")
  }
}

#' Handle Missing Values in Dataset

#' Applies specified method for handling missing values in the dataset.

#' Options include excluding rows with missing values, replacing them, or flagging them.

#' @param df A data frame with potential missing values.

#' @param method Method for handling missing values ('exclude', 'replace', 'mean', 'median', 'flag').

#' @param replace_with Optional; a value or named list to replace missing values with (used with 'repla

#' @return A data frame after handling missing values.
```

```r
#' @export
```

```r
#' @examples
```

```r
#' handle_missing_values(df, method = "exclude")
handle_missing_values <- function(df, method = "exclude", replace_with = NULL) {
  if (method == "exclude") {
    df <- na.omit(df)
  } else if (method == "replace") {
    if (is.null(replace_with)) {
      stop("Please specify a value to replace missing data with using 'replace_with' parameter.")
    }
    replace_list <- setNames(replicate(ncol(df), replace_with, simplify = FALSE), names(df))
    df <- tidyr::replace_na(df, replace_list)
  } else if (method == "mean") {
    numeric_cols <- sapply(df, is.numeric)
    df[, numeric_cols] <- lapply(df[, numeric_cols, drop = FALSE], function(col) {
      replace(col, is.na(col), mean(col, na.rm = TRUE))
    })
  } else if (method == "median") {
    numeric_cols <- sapply(df, is.numeric)
    df[, numeric_cols] <- lapply(df[, numeric_cols, drop = FALSE], function(col) {
      replace(col, is.na(col), median(col, na.rm = TRUE))
    })
  } else if (method == "flag") {
    df$missing_flag <- apply(df, 1, function(x) any(is.na(x)))
  } else {
    stop("Invalid method specified. Choose from 'exclude', 'replace', 'mean', 'median', or 'flag'.")
  }

  df
}
```

```r
#' Initialize Settings for Data Comparison
```

```r
#' Sets up initial settings or parameters for a data comparison session.
```

```r
#' This includes setting default tolerance levels and methods for handling missing values.
```

```r
#' @param tolerance Default tolerance level for numeric comparisons.
```

```r
#' @param missing_value_method Default method for handling missing values in data comparison.
```

```r
#' @return None; this function sets global options and does not return a value.
```

```r
#' @export
```

```r
#' @examples
```

```r
#' initialize_comparison_settings(tolerance = 0.01, missing_value_method = "exclude")
initialize_comparison_settings <- function(tolerance = 0, missing_value_method = "ignore") {
  options(comparison_tolerance = tolerance)
  options(missing_value_handling = missing_value_method)
  message("Comparison settings initialized. Tolerance: ", tolerance, ", Missing Value Handling: ", missi
}
```

```r
#' Prepare Datasets for Comparison

#' Prepares datasets for comparison by performing steps like filtering, sorting,

#' or handling missing values based on predefined criteria.

#' @param df1 First dataset to be prepared.

#' @param df2 Second dataset to be prepared.

#' @param sort_columns Columns to sort the datasets by.

#' @param filter_criteria Criteria for filtering the datasets.

#' @return A list containing two prepared datasets.

#' @export

#' @examples

#' prepare_datasets(df1, df2, sort_columns = "variable", filter_criteria = "variable > 5")
prepare_datasets <- function(df1, df2, sort_columns = NULL, filter_criteria = NULL) {
  if (!is.null(sort_columns)) {
    if (all(sort_columns %in% names(df1)) && all(sort_columns %in% names(df2))) {
      df1 <- df1 %>% arrange(!!!syms(sort_columns))
      df2 <- df2 %>% arrange(!!!syms(sort_columns))
    } else {
      warning("Some sorting columns are not present in the datasets.")
    }
  }

  if (!is.null(filter_criteria)) {
    df1 <- df1 %>% filter(!!rlang::parse_expr(filter_criteria))
    df2 <- df2 %>% filter(!!rlang::parse_expr(filter_criteria))
  }

  list(df1 = df1, df2 = df2)
}

#' Generate a Report of Differences Found in Dataset Comparison

#' Summarizes the differences found between two datasets after comparison.

#' This function creates a report detailing variable and observation differences.

#' @param variable_diffs A data frame or list detailing the differences found in variables.

#' @param observation_diffs A data frame or list detailing the differences found in observations.

#' @return A structured report of the differences, typically a list or a data frame.

#' @export

#' @examples

#' # Assuming variable_diffs and observation_diffs are obtained from comparison functions

#' report_differences(variable_diffs, observation_diffs)
report_differences <- function(variable_diffs, observation_diffs) {
```

```
  report <- list()

  if (!is.null(variable_diffs)) {
    report$variable_differences <- variable_diffs
  }

  if (!is.null(observation_diffs)) {
    report$observation_differences <- observation_diffs
  }

  report
}
```

```
#' Reset Comparison Settings to Defaults
```

```
#' Resets the comparison settings to their default values.
```

```
#' This includes resetting the tolerance level and the method for handling missing values.
```

```
#' @return None; this function resets global options and does not return a value.
```

```
#' @export
```

```
#' @examples
```

```
#' reset_comparison_settings()
reset_comparison_settings <- function() {
  options(comparison_tolerance = 0)
  options(missing_value_handling = "ignore")
  message("Comparison settings have been reset to default values.")
}
```

```
#' Set Tolerance Level for Comparisons
```

```
#' Defines the tolerance level for numeric comparisons.
```

```
#' This is useful for dealing with floating-point arithmetic issues.
```

```
#' @param tolerance A non-negative numeric value specifying the tolerance level.
```

```
#' @return None; this function sets an option and does not return a value.
```

```
#' @export
```

```
#' @examples
```

```
#' set_tolerance(0.001)
set_tolerance <- function(tolerance = 0) {
  if (!is.numeric(tolerance) || tolerance < 0) {
    stop("Tolerance must be a non-negative numeric value.")
  }
  options(comparison_tolerance = tolerance)
  message("Tolerance set to ", tolerance)
}

get_tolerance <- function() {
  return(getOption("comparison_tolerance", default = 0))
}
```

```r
#' Transform Variables in a Dataset

#' Applies specified transformations to variables in a dataset.

#' Useful for standardizing data or converting variables to a consistent format or scale.

#' @param df A data frame containing the variables to be transformed.

#' @param transformations A list of functions for transforming the variables.

#' The names of the list should correspond to the variable names in the dataset.

#' @return A data frame with transformed variables.

#' @export

#' @examples

#' transform_variables(df, list(var1 = as.numeric, var2 = as.factor))
transform_variables <- function(df, transformations) {
  for (var in names(transformations)) {
    if (var %in% names(df)) {
      # Applying the transformation
      transform_function <- transformations[[var]]
      df[[var]] <- transform_function(df[[var]])
    } else {
      warning(paste("Variable", var, "not found in the dataset."))
    }
  }
  df
}
```