R-Based Data Science Pipeline for Volleyball Match Data - Application

Giulia Raineri

2025-04-30

Contents

1 Overview

This R Markdown document is an extension of the file `rally_pipeline_function_HELP.rmd`, providing a practical example of how the functions work and what they return, using a sample dataset available in the `datavolley` package. The dataset refers to a match between Nicaragua and Cuba, held on August 23, 2022, which ended with a 0-3 result.

2 Required Packages

The script requires the following R packages to be installed:

```
# Install pacman if not already installed
if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")

# Use pacman to install and load CRAN packages
pacman::p_load(
  dplyr,          # For data manipulation
  tidyr,          # For reshaping and tidying data
  readxl,         # For reading Excel files
  tibble,         # Modern and user-friendly data frames
  stringr         # Consistent string manipulation
)

# Install and load 'datavolley' from custom repo if not already installed
if (!requireNamespace("datavolley", quietly = TRUE)) {
  install.packages("datavolley", repos = c("https://openvolley.r-universe.dev",
    ↪  "https://cloud.r-project.org"))
}
library(datavolley)

# Confirmation message
cat("All packages correctly installed and loaded.\n")
```

```
## All packages correctly installed and loaded.
```

3 File Input

Three input files are needed to make work the functions:

1. **Match file**: `.dvw` file containing detailed match information. They can be downloaded from the DataVolley software.
2. **Roster file**: A file containing information about the players on each team for the season of interest. Information on players should be complete with name and surname, uniform number, and role. To avoid player data mismatches, it's crucial to check that player numbers and teams match properly in both the roster and match files.
3. **Block statistics file**: A file containing players block statistics.

4 Data Flow

4.1 Step 1: Reading match files and creating the px dataset

The px dataset is the main component used to analyze play-by-play data from the volleyball matches. While the associated help document (rally_pipeline_function_HELP.rmd) shows also how to load match files directly from user computer, for the purpose of this document, it will be shown only how to load a sample match file available in the datavolley package.

This example file can be loaded as follows:

```
# Upload example file
x <- dv_read(dv_example_file(3), insert_technical_timeouts = FALSE)
px <- plays(x)
head(px)
```

```
## # A tibble: 6 x 85
##   match_id point_id time   video_file_number video_time code         team
##   <chr>       <int> <dttm>             <int>      <dbl> <chr>        <chr>
## 1 8984c1f~        0 NA                     1        724 *P06>LUp     NICARAGUA
## 2 8984c1f~        0 NA                     1        724 *z1>LUp      NICARAGUA
## 3 8984c1f~        0 NA                     1        724 aP24>LUp     Cuba Norce~
## 4 8984c1f~        0 NA                     1        724 az3>LUp      Cuba Norce~
## 5 8984c1f~        0 NA                     1        724 *06SH-~~~18D NICARAGUA
## 6 8984c1f~        0 NA                     1        726 a10RH+~~~18D Cuba Norce~
## # i 78 more variables: player_number <int>, player_name <chr>, player_id <chr>,
## #   skill <chr>, skill_type <chr>, evaluation_code <chr>, evaluation <chr>,
## #   attack_code <chr>, attack_description <chr>, set_code <chr>,
## #   set_description <chr>, set_type <chr>, start_zone <int>, end_zone <int>,
## #   end_subzone <chr>, end_cone <int>, skill_subtype <chr>, num_players <chr>,
## #   num_players_numeric <int>, special_code <chr>, timeout <lgl>,
## #   end_of_set <lgl>, substitution <lgl>, point <lgl>, ...
```

```
# View match summary:
summary(x)
```

```
## Match summary:
## Date: 2022-08-23
## League: NA
## Teams: NICARAGUA (QUINTANA RENE/NOGUERA CLAUDIA)
##        vs
##        Cuba Norceca 22 (Tomas Fernandez Arteaga/Joel Olazaba IHerrera)
## Result: 0-3 (7-25, 11-25, 14-25)
## Duration: 55 minutes
```

Output: The resulting px dataset will be a structured table containing variables related to each game event. It will include information on: match and rally details, player data, skill execution, court and spatial data, rotations, video and file number, as well as phases and touch context.

```
head(px)
```

```
## # A tibble: 6 x 85
##   match_id point_id time   video_file_number video_time code         team
##   <chr>       <int> <dttm>             <int>      <dbl> <chr>        <chr>
## 1 8984c1f~        0 NA                     1        724 *P06>LUp     NICARAGUA
## 2 8984c1f~        0 NA                     1        724 *z1>LUp      NICARAGUA
## 3 8984c1f~        0 NA                     1        724 aP24>LUp     Cuba Norce~
## 4 8984c1f~        0 NA                     1        724 az3>LUp      Cuba Norce~
## 5 8984c1f~        0 NA                     1        724 *06SH-~~~18D NICARAGUA
## 6 8984c1f~        0 NA                     1        726 a10RH+~~~18D Cuba Norce~
## # i 78 more variables: player_number <int>, player_name <chr>, player_id <chr>,
## #   skill <chr>, skill_type <chr>, evaluation_code <chr>, evaluation <chr>,
## #   attack_code <chr>, attack_description <chr>, set_code <chr>,
## #   set_description <chr>, set_type <chr>, start_zone <int>, end_zone <int>,
## #   end_subzone <chr>, end_cone <int>, skill_subtype <chr>, num_players <chr>,
## #   num_players_numeric <int>, special_code <chr>, timeout <lgl>,
## #   end_of_set <lgl>, substitution <lgl>, point <lgl>, ...
```

This dataset serves as the foundation for detailed analysis of match events and player performances.

4.2 Step 2: Creating and loading additional files

block_stats.csv

The `block_stats` file dataset was generated using the `calculate_block_statistics()` function. While applying this function to a single match doesn't provide meaningful insights from a statistical point of view, it is shown here to demonstrate how the function works and what it returns. In a real analysis scenario, this function would be more appropriate when applied to data from an entire season, and if data from multiple matches are not available, would be more relevant to search statistics online.

The file will contain the block statistics for players, including the efficacy index, which is a performance measure of each player's blocking ability. This index is calculated based on the ratio of successful blocks to the total blocks attempted during various matches.

- **Total sets played**: The number of sets played by each player-
- **Total blocks attempted**: The number of block attempts made by each player.
- **Perfect blocks**: The number of successful blocks (blocks with point).
- **Efficacy index**: A performance measure calculated as the ratio of perfect blocks to the total blocks attempted.
- **Points per set block**: The number of points scored per set by blocking.

```r
calculate_block_stats <- function(px) {

  # Total number of sets played per player
  sets_played_per_player <- px %>%
    group_by(match_id, team, player_name, player_number) %>%
    summarise(total_sets_played = n_distinct(set_number), .groups = 'drop') %>%
    group_by(team, player_name, player_number) %>%
    summarise(total_sets_played = sum(total_sets_played), .groups = 'drop')

  # Number of attempted blocks per player
  total_blocks_per_player <- px %>%
    filter(skill == "Block") %>%
    group_by(team, player_name, player_number) %>%
    summarise(total_blocks_attempted = n(), .groups = 'drop')
```

```
  # Perfect blocks (with point) per player
  perfect_blocks_per_player <- px %>%
    filter(skill == "Block", evaluation_code == "#") %>%
    group_by(team, player_name, player_number) %>%
    summarise(perfect_blocks = n(), .groups = 'drop')

  # Merge all information in one table
  block_stats <- sets_played_per_player %>%
    left_join(total_blocks_per_player, by = c("team", "player_name", "player_number")) %>%
    left_join(perfect_blocks_per_player, by = c("team", "player_name", "player_number")) %>%
    left_join(px %>% select(team, player_name, player_number, player_id) %>% distinct(),
              by = c("team", "player_name", "player_number")) %>%
    mutate(
      total_blocks_attempted = replace_na(total_blocks_attempted, 0),
      perfect_blocks = replace_na(perfect_blocks, 0),
      efficacy_index = ifelse(total_blocks_attempted > 0, perfect_blocks / total_blocks_attempted,
       ↪  NA),
      points_per_set_block = perfect_blocks / total_sets_played
    ) %>%
    filter(!is.na(player_name)) %>%
    arrange(desc(efficacy_index))

  return(block_stats)
}
block_stats <- calculate_block_stats(px)
head(block_stats)
```

```
## # A tibble: 6 x 9
##   team        player_name player_number total_sets_played total_blocks_attempted
##   <chr>       <chr>               <int>             <int>                  <int>
## 1 Cuba Norce~ Dayana Mar~             5                 2                      1
## 2 Cuba Norce~ unknown pl~             9                 1                      1
## 3 Cuba Norce~ Claudia Ba~            14                 3                      4
## 4 Cuba Norce~ Jessica Ag~            15                 2                      4
## 5 Cuba Norce~ Yamisleydi~            13                 2                      3
## 6 Cuba Norce~ Ailama Ces~            12                 3                      8
## # i 4 more variables: perfect_blocks <int>, player_id <chr>,
## #   efficacy_index <dbl>, points_per_set_block <dbl>
```

roster_file

The `roster_file` is required to match player names with their corresponding roles, number and teams.

Player roles and uniform numbers were taken from the official match report, available at the following link: http://www.norceca.net/2022%20Events/XIX%20Women%20Pan-American%20Cup/Calendars%20WSPAN22/P3%20WPANAM22%20-%20Match%2010.pdf

```
roster_data <- tribble(
  ~team, ~player_name, ~player_number, ~player_role,
  "Cuba Norceca 22", "Yamisleydis Viltres Pacheco", 13, "Middle Blocker",
  "Cuba Norceca 22", "Claudia Basilia Tarin", 14, "Spiker",
  "Cuba Norceca 22", "Dezirett De la Caridad Madan", 2, "Spiker",
  "Cuba Norceca 22", "Laura Beatriz Suarez", 4, "Middle Blocker",
  "Cuba Norceca 22", "Ellemay Santa Miranda", 10, "Libero",
  "Cuba Norceca 22", "Thalia Moreno", 24, "Setter",
  "Cuba Norceca 22", "Ailama Cese Montalvo", 12, "Spiker",
  "Cuba Norceca 22", "Jessica Aguilera", 15, "Middle Blocker",
  "Cuba Norceca 22", "Thainalien Castillo Leyva", 16, "Spiker",
  "Cuba Norceca 22", "Sulian Caridad Matienzo", 18, "Spiker",
  "Cuba Norceca 22", "Greisy Fine", 20, "Spiker",
  "Cuba Norceca 22", "Lianny Tamayo", 23, "Libero",
  "Cuba Norceca 22", "Dayana Martinez", 5, "Middle Blocker",
  "Cuba Norceca 22", "Elena Moreno", 11, "Setter",

  "NICARAGUA", "HELLEN TRAÑA", 4, "Spiker",
  "NICARAGUA", "MYRIAM BLANDINO", 1, "Libero",
  "NICARAGUA", "MARIA ARCIA", 2, "Spiker",
  "NICARAGUA", "SWAN MENDOZA", 3, "Spiker",
  "NICARAGUA", "NORMA BRENES", 5, "Setter",
  "NICARAGUA", "LOLETTE RODRIGUEZ", 6, "Setter",
  "NICARAGUA", "DALIA CONTO", 8, "Middle Blocker",
  "NICARAGUA", "MASIEL BRENES", 9, "Middle Blocker",
  "NICARAGUA", "JUNY AGUILERA", 11, "Libero",
  "NICARAGUA", "ERICKA CALERO", 15, "Spiker",
  "NICARAGUA", "BRITANNY FORBES", 17, "Middle Blocker",
  "NICARAGUA", "GILMARY SMITH", 20, "Spiker"
)
roster_file <- roster_data
head(roster_file)
```

```
## # A tibble: 6 x 4
##   team            player_name                   player_number player_role
##   <chr>           <chr>                                 <dbl> <chr>
## 1 Cuba Norceca 22 Yamisleydis Viltres Pacheco              13 Middle Blocker
## 2 Cuba Norceca 22 Claudia Basilia Tarin                    14 Spiker
## 3 Cuba Norceca 22 Dezirett De la Caridad Madan              2 Spiker
## 4 Cuba Norceca 22 Laura Beatriz Suarez                      4 Middle Blocker
## 5 Cuba Norceca 22 Ellemay Santa Miranda                    10 Libero
## 6 Cuba Norceca 22 Thalia Moreno                            24 Setter
```

4.3 Step 3: Data processing functions

In this section we define a series of functions designed to manipulate the `px` dataset. These functions extract and process key information from `px` with the objective to transform the dataset into a new refined version (`rally_data`) in which each row contains detailed information about an entire ball possession within each rally for each match.

4.3.1 Prepare match data

The `prepare_px_data()` function is used to prepare the match data, ensuring that actions are ordered chronologically and assigning unique identifiers for each rally (`rally_id`) and possession (`possession_id`). It also adds an `original_order` column to preserve the sequence of actions. These additions allow for a more structured analysis of the match, especially when looking at single rallies.

```
prepare_px_data <- function(px) {
  px %>%
    # Keep chronological order
    mutate(original_order = row_number()) %>%

    # Sort by original order
    arrange(original_order) %>%
```

```r
    # Create unique identifier for each rally
    mutate(rally_id = point_id) %>%

    # Group by match and rally
    group_by(match_id, rally_id) %>%

    # Track number of possessions (change of team)
    mutate(possession_id = cumsum(team != lag(team, default = first(team)))) %>%
    ungroup()
}
```

What to input: The dataframe containing the play-by-play data (px).

How to use the function:

```r
px <- prepare_px_data(px)
```

Output: Modified px with the additional variables: `original_order`, `rally_id` and `possession_id`.

```
## # A tibble: 5 x 3
##   original_order rally_id possession_id
##            <int>    <int>         <int>
## 1              1        0             0
## 2              2        0             0
## 3              3        0             1
## 4              4        0             1
## 5              5        0             2
```

4.3.2 Add context variables

The `add_context_variables()` function adds context variables to `px` dataset, providing key insights into the dynamics of the match. It identifies the team's home status, the number of hitters in the front row, key moments, and the current game phase.

```r
add_context_variables <- function(px, roster_file, high_pressure_threshold = 23) {

  # Check if roster_file is a dataframe
  if (!is.data.frame(roster_file)) {
    stop("roster_file must be a dataframe")
  }

  # Check that necessary columns are present in px
  required_columns <- c("player_name", "player_number", "team", "home_team",
↪   "home_setter_position", "visiting_setter_position", "point_phase", "set_number",
↪   "home_team_score", "visiting_team_score")
  missing_columns <- setdiff(required_columns, colnames(px))
  if (length(missing_columns) > 0) {
    stop(paste("Missing required columns in px dataset:", paste(missing_columns, collapse = ", ")))
  }

  # Join roster data with px
  px <- px %>%
    dplyr::left_join(
      roster_file %>%
        dplyr::select(player_name, player_number, team, player_role),
      by = c("player_name", "player_number", "team")
    )

  # Add context variables
  px <- px %>%
    dplyr::mutate(
      is_home = dplyr::if_else(team == home_team, 1L, 0L),

      # Number of hitters in the front row based on setter position
      hitters_front_row = dplyr::case_when(
        is_home == 1L & home_setter_position %in% c(1, 5, 6) ~ 3L,
        is_home == 1L & home_setter_position %in% c(2, 3, 4) ~ 2L,
        is_home == 0L & visiting_setter_position %in% c(1, 5, 6) ~ 3L,
        is_home == 0L & visiting_setter_position %in% c(2, 3, 4) ~ 2L,
        TRUE ~ NA_integer_
      ),

      # Identify key moments
      is_breakpoint = dplyr::if_else(point_phase == "Breakpoint", 1L, 0L, missing = 0L),
      is_sideout = dplyr::if_else(point_phase == "Sideout", 1L, 0L, missing = 0L),

      # High-pressure moments (set point, tight score situations)
      is_high_pressure = dplyr::if_else(
        # Case 1: both teams above the threshold
        (home_team_score >= high_pressure_threshold & visiting_team_score >=
↪  high_pressure_threshold) |

          # Case 2: one team is at 24 (set points)
          home_team_score == 24 | visiting_team_score == 24 |

          # Case 3: 1 point gap in the final set
          (
            abs(home_team_score - visiting_team_score) == 1 &
            (
              (set_number < 5 & (home_team_score >= 20 | visiting_team_score >= 20)) |
              (set_number == 5 & (home_team_score >= 12 | visiting_team_score >= 12))
            )
          ),
        1L, 0L
      ),

      # Game phase (serve or reception)
      phase = dplyr::case_when(
        skill == "Serve" ~ "Serve",
        TRUE ~ "Reception"
      )
    )

  return(px)
}
```

What to input:

- `px`: The play-by-play dataset.
- `roster_file`: The dataframe containing the players information (e.g. name, number, role).
- `high_pressure_threshold`: A numeric threshold for defining high-pressure situations. The default is 23.

How to use the function:

```r
px <- add_context_variables(px, roster_file, high_pressure_threshold = 20)
```

Output: Modified `px` with new context variables:

- `is_home`: A binary variable indicating if the team is playing at home (1 for home, 0 for away.
- `hitters_front_row`: The number of hitters in front row based on `setter_position` (it can 2 or 3)
- `is_breakpoint`: A binary variable indicating if the current point is a breakpoint (1 if true, 0 if false).
- `is_sideout`: A binary variable indicating if the current point is a sideout (1 if true, 0 if false).
- `is_high_pressure`: A binary variable indicating if the current point is a high-pressure situation, based on conditions like set points, tight score gaps, and final set phase (1 if true, 0 if false).
- `phase`: A variable indicating the current phase of the game for each team, such as "Serve" or "Reception".

These context variables help provide a deeper understanding of the match dynamics, allowing for a more detailed analysis.

```
## # A tibble: 5 x 6
##   is_home hitters_front_row is_breakpoint is_sideout is_high_pressure phase
##     <int>             <int>         <int>      <int>            <int> <chr>
## 1       1                 3             0          0                0 Reception
## 2       1                 3             0          0                0 Reception
## 3       0                 2             0          0                0 Reception
## 4       0                 2             0          0                0 Reception
## 5       1                 3             0          0                0 Serve
```

4.3.3 Add skill variables

The `add_skill_variables()` function creates new columns in the `px` dataset to track key volleyball skills, including Serve, Reception, Set, Attack, Block, Dig, and Freeball. For each skill type, it captures the relevant details.

```r
add_skill_variables <- function(px) {

  # Check if necessary columns are present
  required_columns <- c("skill", "player_name", "evaluation_code", "start_zone", "end_zone")
  missing_columns <- setdiff(required_columns, colnames(px))
  if (length(missing_columns) > 0) {
    stop(paste("Missing required columns in px dataset:", paste(missing_columns, collapse = ", ")))
  }

  # Add skill-related variables
  px <- px %>%
    dplyr::mutate(
      # Serve
      serve_player_name     = dplyr::case_when(skill == "Serve" ~ player_name, TRUE ~
       ↪  NA_character_),
      serve_evaluation_code = dplyr::case_when(skill == "Serve" ~ evaluation_code, TRUE ~
       ↪  NA_character_),
      serve_start_zone      = dplyr::case_when(skill == "Serve" ~ start_zone, TRUE ~ NA_integer_),
      serve_end_zone        = dplyr::case_when(skill == "Serve" ~ end_zone, TRUE ~ NA_integer_),

      # Reception
      reception_player_name      = dplyr::case_when(skill == "Reception" ~ player_name, TRUE ~
       ↪ NA_character_),
      reception_evaluation_code  = dplyr::case_when(skill == "Reception" ~ evaluation_code, TRUE ~
       ↪ NA_character_),
      reception_serve_start_zone = dplyr::case_when(skill == "Reception" ~ start_zone, TRUE ~
       ↪ NA_integer_),
      reception_start_zone       = dplyr::case_when(skill == "Reception" ~ end_zone, TRUE ~
       ↪ NA_integer_),

      # Set
      set_player_name     = dplyr::case_when(skill == "Set" ~ player_name, TRUE ~ NA_character_),
      set_evaluation_code = dplyr::case_when(skill == "Set" ~ evaluation_code, TRUE ~
       ↪ NA_character_),
      set_start_zone      = dplyr::case_when(skill == "Set" ~ end_zone, TRUE ~ NA_integer_),
      set_end_zone        = dplyr::case_when(skill == "Attack" ~ start_zone, TRUE ~ NA_integer_),

      # Attack
      attack_player_name     = dplyr::case_when(skill == "Attack" ~ player_name, TRUE ~
       ↪ NA_character_),
      attack_evaluation_code = dplyr::case_when(skill == "Attack" ~ evaluation_code, TRUE ~
       ↪ NA_character_),
      attack_start_zone      = dplyr::case_when(skill == "Attack" ~ start_zone, TRUE ~
       ↪ NA_integer_),
      attack_end_zone        = dplyr::case_when(skill == "Attack" ~ end_zone, TRUE ~ NA_integer_),

      # Block
      block_player_name     = dplyr::case_when(skill == "Block" ~ player_name, TRUE ~
       ↪ NA_character_),
      block_evaluation_code = dplyr::case_when(skill == "Block" ~ evaluation_code, TRUE ~
       ↪ NA_character_),
      block_start_zone      = dplyr::case_when(skill == "Block" ~ end_zone, TRUE ~ NA_integer_),

      # Dig
      dig_player_name       = dplyr::case_when(skill == "Dig" ~ player_name, TRUE ~ NA_character_),
      dig_evaluation_code   = dplyr::case_when(skill == "Dig" ~ evaluation_code, TRUE ~
       ↪ NA_character_),
      dig_attack_start_zone = dplyr::case_when(skill == "Dig" ~ start_zone, TRUE ~ NA_integer_),
      dig_end_zone          = dplyr::case_when(skill == "Dig" ~ end_zone, TRUE ~ NA_integer_),

      # Freeball
      freeball_player_name     = dplyr::case_when(skill == "Freeball" ~ player_name, TRUE ~
       ↪ NA_character_),
      freeball_evaluation_code = dplyr::case_when(skill == "Freeball" ~ evaluation_code, TRUE ~
       ↪ NA_character_),
      freeball_start_zone      = dplyr::case_when(skill == "Freeball" ~ start_zone, TRUE ~
       ↪ NA_integer_),
      freeball_end_zone        = dplyr::case_when(skill == "Freeball" ~ end_zone, TRUE ~
       ↪ NA_integer_)
    )

  return(px)
}
```

What to input: The play-by-play dataframe (px).

How to use the function:

```r
px <- add_skill_variables(px)
```

Output: Modified `px` with new skill-related variables:

- Serve actions: `serve_player_name`, `serve_evaluation_code`, `serve_start_zone`, `serve_end_zone`.
- Reception actions: `reception_player_name`,`reception_evaluation_code`, `reception_serve_start_zone`,`reception_start_zone`.
- Set actions: `set_player_name`, `set_evaluation_code`, `set_start_zone`, `set_end_zone`.
- Attack actions: `attack_player_name`, `attack_evaluation_code`, `attack_start_zone`, `attack_end_zone`.
- Block actions: `block_player_name`, `block_evaluation_code`, `block_start_zone`.
- Dig actions: `dig_player_name`, `dig_evaluation_code`, `dig_attack_start_zone`, `dig_end_zone`.
- Freeball actions: `freeball_player_name`, `freeball_evaluation_code`, `freeball_start_zone`, `freeball_end_zone`.

Example of output for serve actions:

```
## # A tibble: 5 x 4
##   serve_player_name serve_evaluation_code serve_start_zone serve_end_zone
##   <chr>             <chr>                            <int>          <int>
## 1 LOLETTE RODRIGUEZ -                                    1              8
## 2 HELLEN TRAÑA      -                                    6              6
## 3 BRITANNY FORBES   -                                    1              8
## 4 MARIA ARCIA       =                                    1              5
## 5 Dayana Martinez   +                                    5              4
```

4.3.4 Aggregate rally data

The `aggregate_rally_data()` function aggregates the data by rally and possession, ensuring that only relevant actions are included. For instance, it removes rows where all skill variables are `NA` (indicating no action occurred). Each row represents a unique rally with relevant aggregated data for analysis.

```r
aggregate_rally_data <- function(px) {
  skill_columns <- c(
    "serve_player_name", "serve_evaluation_code", "serve_start_zone", "serve_end_zone",
    "reception_player_name", "reception_evaluation_code", "reception_serve_start_zone",
     ↪  "reception_start_zone",
    "set_player_name", "set_evaluation_code", "set_start_zone", "set_end_zone",
    "attack_player_name", "attack_evaluation_code", "attack_start_zone", "attack_end_zone",
    "block_player_name", "block_evaluation_code", "block_start_zone",
    "dig_player_name", "dig_evaluation_code", "dig_attack_start_zone", "dig_end_zone",
    "freeball_player_name", "freeball_evaluation_code", "freeball_start_zone", "freeball_end_zone"
  )
  rally_data <- px %>%
    group_by(match_id, rally_id, possession_id, team) %>%
    arrange(original_order) %>%
    summarise(
      across(all_of(skill_columns), ~ first(.[!is.na(.)]), .names = "{.col}"),

      # Additional context variables
      phase                   = first(phase),
      is_breakpoint           = first(is_breakpoint),
      is_sideout              = first(is_sideout),
      is_high_pressure       = first(is_high_pressure),
      hitters_front_row      = first(hitters_front_row),
      home_score_start_of_point     = max(home_score_start_of_point, na.rm = TRUE),
      visiting_score_start_of_point = max(visiting_score_start_of_point, na.rm = TRUE),
      touching_team_is_home  = first(is_home),
      home_team              = first(home_team),
      visiting_team          = first(visiting_team),
      home_setter_position   = first(home_setter_position),
      visiting_setter_position = first(visiting_setter_position),
      serving_team           = first(serving_team),
      is_home     = first(is_home),

      # Players in court
      home_p1     = first(home_p1[!is.na(home_p1)]),
      home_p2     = first(home_p2[!is.na(home_p2)]),
      home_p3     = first(home_p3[!is.na(home_p3)]),
      home_p4     = first(home_p4[!is.na(home_p4)]),
      home_p5     = first(home_p5[!is.na(home_p5)]),
      home_p6     = first(home_p6[!is.na(home_p6)]),
      visiting_p1 = first(visiting_p1[!is.na(visiting_p1)]),
      visiting_p2 = first(visiting_p2[!is.na(visiting_p2)]),
      visiting_p3 = first(visiting_p3[!is.na(visiting_p3)]),
      visiting_p4 = first(visiting_p4[!is.na(visiting_p4)]),
      visiting_p5 = first(visiting_p5[!is.na(visiting_p5)]),
      visiting_p6 = first(visiting_p6[!is.na(visiting_p6)])
    ) %>%
    ungroup() %>%
    filter(rowSums(!is.na(pick(all_of(skill_columns)))) > 0) %>%
    group_by(match_id, rally_id) %>%
    mutate(possession_id = row_number() - 1) %>%
    ungroup() %>%
    arrange(match_id, rally_id, possession_id)

  return(rally_data)
}
```

What to input: The play-by-play dataframe px.

How to use the function:

```r
rally_data <- aggregate_rally_data(px)
```

```
## `summarise()` has grouped output by 'match_id', 'rally_id', 'possession_id'.
## You can override using the `.groups` argument.
```

Output: A dataframe where each row corresponds to a rally and contains aggregated data for that rally.

```r
head(rally_data)
```

```
## # A tibble: 6 x 57
##   match_id  rally_id possession_id team  serve_player_name serve_evaluation_code
##   <chr>        <int>         <dbl> <chr> <chr>             <chr>
## 1 8984c1f4~        0             0 NICA~ LOLETTE RODRIGUEZ -
## 2 8984c1f4~        0             1 Cuba~ <NA>              <NA>
## 3 8984c1f4~        0             2 NICA~ <NA>              <NA>
## 4 8984c1f4~        0             3 Cuba~ <NA>              <NA>
## 5 8984c1f4~        1             0 Cuba~ Yamisleydis Vilt~ -
## 6 8984c1f4~        1             1 NICA~ <NA>              <NA>
## # i 51 more variables: serve_start_zone <int>, serve_end_zone <int>,
## #   reception_player_name <chr>, reception_evaluation_code <chr>,
## #   reception_serve_start_zone <int>, reception_start_zone <int>,
## #   set_player_name <chr>, set_evaluation_code <chr>, set_start_zone <int>,
## #   set_end_zone <int>, attack_player_name <chr>, attack_evaluation_code <chr>,
## #   attack_start_zone <int>, attack_end_zone <int>, block_player_name <chr>,
## #   block_evaluation_code <chr>, block_start_zone <int>, ...
```

4.3.5 Assign player name and role to positions

The `assign_info_to_positions()` function assigns player names and player roles to the corresponding court positions for both home and visiting teams, by joining the `roster_file` with `rally_data` . It ensures that player information is correctly placed in the appropriate positions. Additionally, it updates the roles of the Spiker who is in the opposite position of the Setter, changing it to the Opposite role.

```
assign_info_to_positions <- function(rally_data, roster_file) {
  join_and_rename <- function(data, number_col, team_col, new_name, new_role) {
    data %>%
      left_join(
        roster_file %>% select(player_name, player_number, team, player_role),
        by = setNames(c("player_number", "team"), c(number_col, team_col))
      ) %>%
      rename(!!new_name := player_name, !!new_role := player_role) %>%
      select(-starts_with("player_name"), -starts_with("player_role"), everything())
  }

  rally_data <- rally_data %>%
    join_and_rename("home_p1", "home_team", "home_p1_name", "home_p1_role") %>%
    join_and_rename("home_p2", "home_team", "home_p2_name", "home_p2_role") %>%
    join_and_rename("home_p3", "home_team", "home_p3_name", "home_p3_role") %>%
    join_and_rename("home_p4", "home_team", "home_p4_name", "home_p4_role") %>%
    join_and_rename("home_p5", "home_team", "home_p5_name", "home_p5_role") %>%
    join_and_rename("home_p6", "home_team", "home_p6_name", "home_p6_role") %>%
    join_and_rename("visiting_p1", "visiting_team", "visiting_p1_name", "visiting_p1_role") %>%
    join_and_rename("visiting_p2", "visiting_team", "visiting_p2_name", "visiting_p2_role") %>%
    join_and_rename("visiting_p3", "visiting_team", "visiting_p3_name", "visiting_p3_role") %>%
    join_and_rename("visiting_p4", "visiting_team", "visiting_p4_name", "visiting_p4_role") %>%
    join_and_rename("visiting_p5", "visiting_team", "visiting_p5_name", "visiting_p5_role") %>%
    join_and_rename("visiting_p6", "visiting_team", "visiting_p6_name", "visiting_p6_role") %>%

    # Substitution Spiker with Opposite
    mutate(
      # home team
      home_p1_role = ifelse(home_setter_position == 4 & home_p1_role == "Spiker", "Opposite",
       ↪  home_p1_role),
      home_p2_role = ifelse(home_setter_position == 5 & home_p2_role == "Spiker", "Opposite",
       ↪  home_p2_role),
      home_p3_role = ifelse(home_setter_position == 6 & home_p3_role == "Spiker", "Opposite",
       ↪  home_p3_role),
      home_p4_role = ifelse(home_setter_position == 1 & home_p4_role == "Spiker", "Opposite",
       ↪  home_p4_role),
      home_p5_role = ifelse(home_setter_position == 2 & home_p5_role == "Spiker", "Opposite",
       ↪  home_p5_role),
      home_p6_role = ifelse(home_setter_position == 3 & home_p6_role == "Spiker", "Opposite",
       ↪  home_p6_role),
      # visiting team
      visiting_p1_role = ifelse(visiting_setter_position == 4 & visiting_p1_role == "Spiker",
       ↪  "Opposite", visiting_p1_role),
      visiting_p2_role = ifelse(visiting_setter_position == 5 & visiting_p2_role == "Spiker",
       ↪  "Opposite", visiting_p2_role),
      visiting_p3_role = ifelse(visiting_setter_position == 6 & visiting_p3_role == "Spiker",
       ↪  "Opposite", visiting_p3_role),
      visiting_p4_role = ifelse(visiting_setter_position == 1 & visiting_p4_role == "Spiker",
       ↪  "Opposite", visiting_p4_role),
      visiting_p5_role = ifelse(visiting_setter_position == 2 & visiting_p5_role == "Spiker",
       ↪  "Opposite", visiting_p5_role),
      visiting_p6_role = ifelse(visiting_setter_position == 3 & visiting_p6_role == "Spiker",
       ↪  "Opposite", visiting_p6_role)
    )

  return(rally_data)
}
```

What to input:

- `rally_data`: The aggregated rally data frame.
- `roster_file`: The dataframe containing player information (`player_name`, `player_number`, `team`, and `player_role`).

How to use the function:

```
rally_data <- assign_info_to_positions(rally_data, roster_file)
```

Output: Modified `rally_data` with player names and roles assigned to each court position. Additionally, players in the opposite position of the setter, are substituted with the role "Opposite".

4.3.6 Assign block statistics to opponent team

The `assign_opponent_block_efficacy()` function assigns efficacy block index values to the opponent players, in front row positions (p2, p3, p4), from the `block_stats.csv` file. To determine whether to look at `home_p*` or `visiting_p*`, the function checks if the team of interest (`our_team`) is playing at home or away, using the `touching_team_is_home` variable.

```
assign_opponent_block_efficacy <- function(rally_data, block_stats, our_team) {

  # Remove possible duplicates in block_stats
  block_stats_clean <- block_stats %>%
    distinct(player_number, player_name, .keep_all = TRUE)

  # Assign efficacy_index to opponent players in front-row positions (p2, p3, p4)
  rally_data <- rally_data %>%
    mutate(
      opponent_efficacy_index_p2 = ifelse(
        touching_team_is_home == 1,
        block_stats_clean$efficacy_index[match(paste(visiting_p2, visiting_team),
 ↪  paste(block_stats_clean$player_number, block_stats_clean$team))],
        block_stats_clean$efficacy_index[match(paste(home_p2, home_team),
 ↪  paste(block_stats_clean$player_number, block_stats_clean$team))]
```

```
      ),

      opponent_efficacy_index_p3 = ifelse(
        touching_team_is_home == 1,
        block_stats_clean$efficacy_index[match(paste(visiting_p3, visiting_team),
 ↪  paste(block_stats_clean$player_number, block_stats_clean$team))],
        block_stats_clean$efficacy_index[match(paste(home_p3, home_team),
 ↪  paste(block_stats_clean$player_number, block_stats_clean$team))]
      ),

      opponent_efficacy_index_p4 = ifelse(
        touching_team_is_home == 1,
        block_stats_clean$efficacy_index[match(paste(visiting_p4, visiting_team),
 ↪  paste(block_stats_clean$player_number, block_stats_clean$team))],
        block_stats_clean$efficacy_index[match(paste(home_p4, home_team),
 ↪  paste(block_stats_clean$player_number, block_stats_clean$team))]
      )
    ) %>%
    # Replace NA with 0
    mutate(
      opponent_efficacy_index_p2 = ifelse(is.na(opponent_efficacy_index_p2), 0,
       ↪  opponent_efficacy_index_p2),
      opponent_efficacy_index_p3 = ifelse(is.na(opponent_efficacy_index_p3), 0,
       ↪  opponent_efficacy_index_p3),
      opponent_efficacy_index_p4 = ifelse(is.na(opponent_efficacy_index_p4), 0,
       ↪  opponent_efficacy_index_p4)
    )

  return(rally_data)
}
```

What to input:

- `rally_data`: The aggregated rally data frame.
- `block_stats`: The dataframe containing the block statistics. It is important that it containis only one row per unique combination of `player_number` and `player_name` for each team. The function will automatically remove duplicate rows if necessary.
- `our_team`: A string specifying the name of the team of interest. This parameter helps identify the opposing team.

How to use the function:

```
rally_data <- assign_opponent_block_efficacy(rally_data, block_stats, our_team = "NICARAGUA")
```

Output: The function returns an updated version of `rally_data` with new columns (`opponent_efficacy_index_p2`, `opponent_efficacy_index_p3`, and `opponent_efficacy_index_p4`). These columns contain the block efficacy index values for the opposing players in the front-row positions. If no efficacy index is available, the value will be set to 0.

```
## # A tibble: 5 x 3
##   opponent_efficacy_index_p2 opponent_efficacy_index_p3 opponent_efficacy_inde~1
##                        <dbl>                      <dbl>                    <dbl>
## 1                      0.333                          0                        0
## 2                           0                              0.2                            0
## 3                           0.333                          0                              0
## 4                           0                              0.2                            0
## 5                           0                              0.2                            0
## # i abbreviated name: 1: opponent_efficacy_index_p4
```

4.3.7 Assign block statistics to opponent effective positions

The `assign_opponent_effective_block_positions()` functions assigns the opponent's block efficacy index to their effective court positions based on player's role. It dynamically adjust their positions according to the rally context. It enables a more accurate performance analysis of the opponent's defensive performance during the rally.

The rules for assigning effective positions based on players' roles are:

- The **Opposite** and the **Setter**, when in the front row, are always assigned to position **2**, except when they are in position **4** and the phase of rally is **"Reception"**.
- The **Spiker**, when in front row, is always in position **4**, except when player is in position **2** and the phase of rally is **"Reception"**.
- The **Middle Blocker** is always assigned to position **3**.

```r
remap_opponent_block_to_effective_positions <- function(rally_data, our_team) {

  get_effective_position <- function(role, position, phase) {
    if (is.na(role)) return(NA_integer_)

    if (role == "Middle Blocker") {
      return(3)
    } else if (role == "Spiker") {
      if (position == 2 && phase == "Reception") return(2) else return(4)
    } else if (role %in% c("Opposite", "Setter")) {
      if (position == 4 && phase == "Reception") return(4) else return(2)
    } else {
      return(NA_integer_)
    }
  }

  rally_data <- rally_data %>%
    mutate(
      is_home = home_team == our_team,

      opp_p2_role = if_else(is_home, visiting_p2_role, home_p2_role),
      opp_p3_role = if_else(is_home, visiting_p3_role, home_p3_role),
      opp_p4_role = if_else(is_home, visiting_p4_role, home_p4_role),

      eff_p2_val = opponent_efficacy_index_p2,
      eff_p3_val = opponent_efficacy_index_p3,
      eff_p4_val = opponent_efficacy_index_p4,

      eff_pos_p2 = mapply(get_effective_position, opp_p2_role, 2, phase),
      eff_pos_p3 = mapply(get_effective_position, opp_p3_role, 3, phase),
      eff_pos_p4 = mapply(get_effective_position, opp_p4_role, 4, phase)
    ) %>%
    rowwise() %>%
    mutate(
      opponent_effective_efficacy_index_p2 = sum(
        ifelse(eff_pos_p2 == 2, eff_p2_val, 0),
        ifelse(eff_pos_p3 == 2, eff_p3_val, 0),
        ifelse(eff_pos_p4 == 2, eff_p4_val, 0),
        na.rm = TRUE
      ),
      opponent_effective_efficacy_index_p3 = sum(
        ifelse(eff_pos_p2 == 3, eff_p2_val, 0),
        ifelse(eff_pos_p3 == 3, eff_p3_val, 0),
        ifelse(eff_pos_p4 == 3, eff_p4_val, 0),
        na.rm = TRUE
      ),
      opponent_effective_efficacy_index_p4 = sum(
        ifelse(eff_pos_p2 == 4, eff_p2_val, 0),
        ifelse(eff_pos_p3 == 4, eff_p3_val, 0),
        ifelse(eff_pos_p4 == 4, eff_p4_val, 0),
        na.rm = TRUE
      )
    ) %>%
    ungroup() %>%
    select(-starts_with("eff_"), -starts_with("opp_"), -is_home)

  return(rally_data)
}
```

What to input:

- `rally_data` containing opponent players' roles and nominal positions, team side indicator, phase of the game and pre-assigned opponent block indices (`opponent_efficacy_index_p`).

How to use:

```
rally_data <- remap_opponent_block_to_effective_positions(rally_data, our_team = "NICARAGUA")
```

Output: The function returns the updated `rally_data` dataframe with three new variables: `opponent_effective_efficacy_index_p2`, `opponent_effective_efficacy_index_p3`, `opponent_effective_efficacy_index_p4`. They reflect the efficacy values based on the effective defending position of each opponent player, considering the in-play shifts related to their role and game phase.

```
## # A tibble: 5 x 3
##   opponent_effective_efficacy_in~1 opponent_effective_e~2 opponent_effective_e~3
##                              <dbl>                  <dbl>                  <dbl>
## 1                              0                    0.333                    0
## 2                              0.2                  0                        0
## 3                              0                    0.333                    0
## 4                              0.2                  0                        0
## 5                              0                    0                        0.2
## # i abbreviated names: 1: opponent_effective_efficacy_index_p2,
## #   2: opponent_effective_efficacy_index_p3,
## #   3: opponent_effective_efficacy_index_p4
```

5 Conclusion

This application demonstrates how the functions provided in the `rally_pipeline_functions_HELP` document can be used in a real-case scenario, emphasizing their flexibility and modular design.

Throughout this extension, user can see how each function contributes to the transformation of raw DataVolley input into a clean, analysis-ready dastaset.