UNIVERSITÀ DEGLI STUDI DI BRESCIA

*DIPARTIMENTO DI ECONOMIA E MANAGEMENT*

*Corso di Laurea Magistrale*

*in ANALYTICS E DATA SCIENCE FOR ECONOMICS AND MANAGEMENT*

**Tesi di Laurea**

# Data Science for Volleyball Analytics: Preparing and Modelling Data for Performance Prediction

Relatore: Chiar.ma Prof.ssa Marica Manisera

Correlatore: Chiar.mo Prof. Riccardo Ricciardi

Laureanda:
Giulia Raineri

Matricola n. 731113

*Anno Accademico 2024/2025*

**To those I love,**

who never let me face the ups and downs alone,

and who made this ride unforgettable.

*“Nobody queues for a flat rollercoaster”*.

# Table of Contents

# INTRODUCTION

The advent of big data and advanced analytical techniques has changed how teams and coaches analyse and evaluate athletes' performances and use data to take informed decisions. In recent decades, data science has become a game-changer in various fields, including sports.

The integration of technology into sports analytics allowed teams to collect and analyse data in ways that were unimaginable. As technology advanced, the amount of data generated in sports have increased, making big data an essential tool for modern sports organizations [1,2]. The ability to analyse complex datasets enables teams to make more informed decisions and develop competitive strategies [3]. Big data not only allows team to conduct in-depth analysis of player performance but also helps coaches predict risks of injuries by analysing biometric data [4].

Elite sports now are using advanced technologies, such as high-definition cameras, sophisticated tracking systems, used to collect accurate real-time information in player movements, ball trajectories, and game dynamics [5,6]. Data are collected not only from post-game analysis but also processed in real-time during matches enabling coaches and analysts to make tactical live adjustments [7].

The growing importance of analytics created a culture of collaboration between coaches and data scientists, for instance teams began to recognize the value of integrating analytical insights into game strategic planning and this collaboration led to more informed decisions, allowing teams to optimize plan and strategies based on data, which resulted in a better-informed judgment [8].

Data science in sports is not only physical performance. Recent studies have begun exploring also psychological and cognitive factors, such as mental fatigue, or stress [9,10].

Today more than ever, the role of data science in sports analytics continues to expand, incorporating elements of machine learning (ML) and artificial intelligence (AI) to enhance forecast capabilities. These technologies allow teams to simulate game scenario, analyse opponent strategies and optimize training based on performance data.

Nevertheless, as is well known, the success of an analysis depends on the quality of the data to be used: reliable, consistent, and well-structured data are essential in order to extract meaningful information and actionable insights.

This work focuses on the application of data science to volleyball data, trying to develop a pipeline for processing and transforming raw match data into a structured dataset. The creation of the functions, then presented, is a crucial step for enabling both descriptive and predictive analytics in high-level volleyball performance and match analysis.

# DATA SCIENCE IN VOLLEYBALL: RESEARCH BACKGROUND

Despite the growing interest, data-driven volleyball research seems to remain little explored in respect to other team sports, like football or baseball. However, several are the recent key studies published among the various applications of data analytics in volleyball, including match scheduling, training monitoring, performance assessment, outcome prediction, injury prevention, tactical analysis, and other relevant areas.

Research to identify relevant studies published recently (up to May 2025) was conducted through the main databases, including PubMed, Journal of Sports Analytics, and Journal of Quantitative Analytics in Sports, among other. The research was carried out searching for keywords such as "volleyball analytics", "data science in sports", "volleyball performance assessment" and related terms. The articles analysed all concentrate on the main applications of data science and machine learning techniques to volleyball sport and have underlined the importance and potentiality of data to monitor and improve athlete and team performance and overall volleyball sport management.

In match scheduling area, optimization algorithms and statistical models have been developed to balance fairness, minimize travel, and maximize competitiveness in volleyball tournaments [11]. The biggest area is the one of training monitoring, in particular wearable technology devices used to collect data to offer reliable insights to coaches. These systems enable the tracking of parameters such as jump frequency, jump height, and acceleration, supporting the creation of customized training programs [12]. In the field of performance evaluation, machine learning techniques and real-time tracking systems, such as KINEXON, have facilitated a more detailed examination of player metrics during both training and competitions, providing deeper insights into neuromuscular demands and game-related movements [13,14]. A strong correlation was found between some indicators, for example, high-intensity jumps and cumulative workloads, with in-game performance, underlying the importance of balancing competitive preparation and load control [15].

Another relevant area of research concerns outcome forecast. Statistical models, including hierarchical Bayesian frameworks and more advanced but always interpretable AI models, have been used to predict match results with increasing accuracy, providing decision-support tools for coaches and analysts [16,17]. In the same direction, tactical analysis has been improved by the adoption of computer vision and social network analysis approaches. Recent developments, such as PathFinderPlus, allow for fine-grained classification of game strategies, improving the understanding of team dynamics and offensive/defensive patterns to predict or adjust the strategy of a team [18].

Another interesting area is the one of injury prevention, that has benefited from real-time monitoring and data-driven assessments of athletes' biomechanical and physiological metrics [19,20]. By identifying patterns associated with elevated injury risks, coaches and medical staff can be informed and they can work in advance on safter training loads and contribute to long-term athlete health.

The opportunities in volleyball analytics are vast, and the possibility of expansion in the analysis continues to grow with the introduction of new technologies and statistical techniques. Even if many studies emphasize the potential of predictive modelling and tactical analysis, in the literature there are few articles helping in the preprocessing and contextualization of raw data.

This contribution aims to introduce a systematic and reproducible R-based pipeline for data preparation, intended to facilitate the evaluation of performance and strategic planning in volleyball by using data science and machine learning methods [21]. This workflow is specifically designed to handle the most frequent used format of raw volleyball data, exported directly from the software DataVolley.

“DataVolley” developed by Data Project, an Italian software company specialized in sports data management. The software is quickly gained popularity in the world of volleyball, becoming the reference tool for match analysis [22]. It was designed specifically for volleyball scouting, prioritizing simplicity and usability to facilitate the scouting process. It is characterized by a user-friendly interface that allows scouts to efficiently input data in real-time.

# OBJECTIVE OF THE ANALYSIS

## Motivations

One of the biggest limitations of why data-driven approaches in volleyball are still underutilized, especially compared to other team sports, is that the data exported from the most widely used software for match recording, "DataVolley", are not immediately available in a readable format, but they need to be pre-processed. The software gives the possibility of making basic analyses directly in-platform, but for more sophisticated statistical analyses the user still has to download the data and manipulate them.

A big contribution was made by Ben Raymond, who developed the R package "datavolley" [23]. This package greatly improved the accessibility of "DataVolley" files by converting raw data, often difficult to interpret, into a structured dataset, where each row there is a single touch of the ball by a single player in the game, with all the specifications recorded by the scout. However, the structure focuses on individual actions, which are useful for skill-based performance evaluation at the player level but does not capture the dynamic of an entire rally.

To perform and evaluate a comprehensive situation of the game is important to highlight the rally, that can be defined as a sequence of consecutive actions starting with a serve ending when the point is won by one team. To do this, further transformations on the data are necessary.
In addition, context variables such as player roles, rotations or game phases are not yet specifically present. This justifies the need for a code that is useful to clean and add information to already existing data, making them suitable for both descriptive and predictive analytics.

## Scope

The core of this thesis is data preparation, as a necessary step toward the final development of a predictive model. The objective is to develop a pipeline of functions in R, which allow to transform data from raw to a structured dataset, where each row corresponds to a sequence of actions performed by the same team during a single rally phase.

The proposed functions are designed for:

- Add context variables (e.g. high-pressure situations, front-line attackers).
- Map the names and roles of players, to obtain the actual positions on the field, based on the roles.
- Aggregate data from action to sequence of passes.
- Maintain modularity and reproducibility for future extensions.

## Contribution

The main contribution of this work is the modularity and the possibility to replicate the pipeline to transform the volleyball data in R software, starting from the dataset processed through the R "datavolley" package.

The aim of the pipeline is to provide high-quality data, that can serve as a basis for building effective predictive models and conducting performance analysis. The data must be accurate and need to capture all the relevant contextual, tactical and technical characteristics that can influence the game. Without this quality data, also the more sophisticated model may fail leading to incorrect results or giving no information at all.

The structure and the logic implemented through the pipeline provides a solid foundation for future analytical work where sophisticated models can be tested on the enriched dataset with different aims: performance analysis, prediction of the outcome of a match or a tournament, tactical assessment, injury risk estimation, and study of the offensive and defensive strategy of opponents.

In summary, this thesis introduces a conceptual framework for managing data in volleyball, addressing a critical gap between the availability of raw data and the actionable analysis.

# VOLLEYBALL: GAME OVERVIEW

## General Notions

Volleyball is one of the most popular sports globally, with an estimated 900 million fans across the world.[1] It was invented in 1895 by William G. Morgan in the United States and in 1964 it becomes an Olympic event.

It can be played from an early age and unlike many other team sports, there is no direct contact with the opponent. The game’s outcome is based on how the individual player and the team reacts. It involves movements that require speed, fast reactions, great concentration, quick reflexes, coordination, high communication and good physical condition.

Two teams, typically of six players each, compete to score points by sending the ball over the net and into the opposing team’s court. The goal of volleyball is to win the match, played in sets, at the best of five. The first sets are won when a team reach 25 points, with a minimum lead of two points required to win. In case the teams reach the tie, winning two sets each, the fifth set is played (“tie break”), which is won at 15 points.

A point is scored when:

- Ball touches the ground in the opponent’s field.
- Ball gets off the court.
- Ball hits the net.

[1] **SportyTell**, "Top 10 Most Popular Sports in The World 2023." Accessed October 23, 2024. https://sportytell.com/sports/most-popular-sports-world/

- Infringement of a rule by a player.

Each team is allowed to touch the ball three times to return the ball over the net. These touches usually involve the reception, the set and the attack. Players must avoid a double hit or holding the ball for too long. Front-row players can try to block an opponent's attack by doing a blocking. If the block is successful, the ball may land on the attacking team's side. A block does not count as one of the three allowed touches.

## Volleyball Court

Volleyball is played on a court of 18 meters long and 9 meters wide, divided into two equal halves by a hight net (Figure 1). The net height varies; it is 2.43 meters for men's competitions and 2.24 meters for women's competitions. The court is divided in zones for serving, attacking and defending. The attack line is located 3 meters from the net. It divides front row players from the back row ones. Only front row players can attack the ball above the net from in front of the attack line. The service area is located behind the end line, if a player crosses this line during the service, it is a foul.

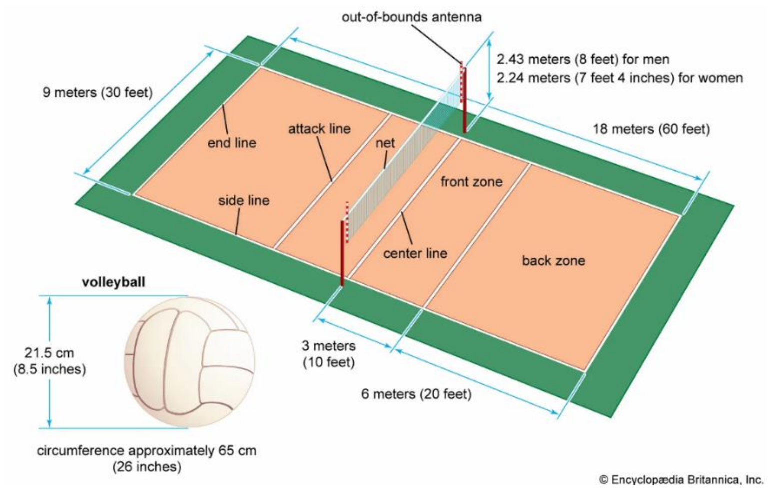

*Figure 1 - Volleyball Court's Dimensions. SOURCE: https://www.britannica.com/*

## Fundamentals

There are several fundamental skills known by volleyball players:

**SERVING:** The game always starts with one team serving the ball from behind the end line, the player must hit the ball over the net and into the opponent's court. There are different types of serves.

**RECEPTION (bump or forearm):** This is the first contact after the serve. It is important for the receiver to control the ball and pass it to the setter. Effective reception is crucial for building a solid attack.

**SET:** Setting is the technique to lift the ball and prepare it for the attack. This fundamental requires precision and ability to place the ball at the right height for each attacker.

**ATTACK:** The spike usually is the third touch. A player jumps and hits the ball over the net in the opponent's court. There are several types of attacks made by different spikers, depending on the hight and trajectory of the ball. The attack requires excellent coordination between the setter and the attacker.

**DIG:** It is a defensive move used to defend against a spike. Players use a forearm pass to keep the ball from hitting the ground after an attack.

**BLOCK:** Blocking is a defensive move, where a front-row players jump to prevent the attack from the opponent's team crossing the net.

## Players’ roles

Each player has a specialized role, crucial for a team’s overall performance. Here’s a breakdown of the main positions and responsibilities:

**SETTER:** the primary responsibility is to set up offensive plays by placing accurately the ball for the hitters. The setter needs to be very precise, have an excellent ball control and quick decision-making to decide which hitter to set up. The setter is always in court, occupying both front row and back row line.

**OUTSIDE HITTER:** it is a very versatile role, because the two main responsibilities are both the defence and the attack, both front-row and back-row from the left side of the court.

**OPPOSITE HITTER:** The main responsibility of this role is to attack from the right-side of the court. Opposite hitters usually don’t have reception duties.

**MIDDLE BLOCKER:** Player specialized in blocking the attack from the opposing team’s hitters, and perform quick attacks close to setter, like the fast. The middle blocker, when in second line during rotation, generally exchanges position with the libero to optimize defensive strategy. This role is characterized by the ability to read the opponent’s attack, be strong, very speed and agile. The position in court usually is the number 3.

**LIBERO:** the libero is a defensive specialist, and it is easily identifiable by wearing a different coloured jersey. They only play in second row and their role is to receive serve and digging attacks.

# R-driven Volleyball Data Pipeline – Core functions

## Introduction to the Help R Markdown

This section presents the help for functions created in R to support transformation and enrichment of volleyball data exported by DataVolley software.

The key objective of these functions is to re-structure data at action-level, passing to a team sequence level, adding contextual key variables such as game phase, player rotation, role responsibilities and press conditions.

The idea behind this development is the need to move from the ability to develop simple statistical analysis to a broader game perspective, where it is possible to represent performance at its tactical context level.

The following is an overview of each function, its purpose, inputs and outputs required. Each function has been carefully documented to ensure modularity and future re-use possibilities.

## Overview

The objective of this R script is to create a set of functions that allow for processing volleyball match data in “dvw” format, which are downloaded from the “DataVolley” software.

This document provides a step-by-step breakdown of the process for reading and processing the data.

## Required Packages

The script requires the following R packages to be installed:

```
#For reading and pre-processing volleyball match data
install.packages("datavolley", repos =
                   c("<https://openvolley.r-universe.dev>","<https://c
loud.r-project.org>"))

# For data manipulation
install.packages("dplyr")

# For tidying and reshaping data
install.packages("tidyr")

# For reading Excel or csv files
install.packages("readxl")
```

Once the packages are installed, the user can load them into the R environment using the `library()` function.

```
library(datavolley)
library(readxl)
library(dplyr)
library(tidyr)
```

Remember to install the packages only once, after installation the user can always load them with the `library()` function.

## File Input

Three input files are needed to make the functions work:

1. **Match files**: `.dvw` files containing detailed match information. They can be downloaded from the DataVolley software.

2. **Roster file**: An `.xlsx` file containing information about the players on each team for the season of interest. Information on players should be completed with name and surname, uniform number, and role. For matches of Superlega[2] volley, the roster is published on the official website[3]. 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 `.csv` file containing players block statistics.

[2] The Superlega is the top professional men's volleyball league in Italy, organized by the Italian Volleyball League (Lega Pallavolo Serie A).
[3] https://www.legavolley.it/

# Data Flow

## Step 1: Reading match files and creating the `px` dataset

The “px” dataset is the key component in analyzing the play-by-play data from the matches. It is created in two steps:

1. **Reading the match files**: The `dvw` match files are read using the `datavolley` package. This package structure match data into a more readable format.

2. **Processing the match data**: The `px` dataset is created by extracting the play-by-play component of each match.

The **datavolley** package is designed to decode match data stored in `dvw` format. The decoding process extracts crucial information such as game events, players involved, actions performed and organizes data into a structured format.

The original data downloaded from DataVolley software appears as lines of code like the following:

```
a10SQ-
\~\~\~15\~\~\~\~R1;;;;;;;;20.32.49;1;2;6;1;470;;18;6;15;4;11;5;10;4;15;
22;11;7;
```

If the user has only one match file available, it can be loaded with the following command:

```r
# Specify directory and match file
match_file <- "path_to_your_dvw_file/match_file.dvw"

# Read match file
px <- dv_read(match_file)

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

If the user has multiple files, they can be decoded to create the px dataset, using the following code:

```r
# Specify the directory where the `.dvw` are located
dir_path <- "path_to_your_dvw_files"

# Load match files into a list
d <- dir(dir_path, pattern = "dvw$", full.names = TRUE)
lx <- lapply(seq_along(d), function(i) {
  message(paste("Reading file", i, "of", length(d)))
  dv_read(d[i], insert_technical_timeouts = FALSE)
})

# Combine play-by-play data from all matches into a single dataset
px <- bind_rows(lapply(lx, plays))
```

If the user does not have any match file, the datavolley package provides an example file that can be used to start exploring its features. This example file can be loaded as follows:

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

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

**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.

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

## Step 2: Creating and loading additional files

***block_stats.csv***

If the user has available the entire season, it can use it to calculate block statistics and generate the `block_stats.csv` file. This file contains 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.

The function `calculates_block_stats()` calculates the following statistics for each player:

- **Total sets played**: The number of sets each player played.
- **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 ration 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 = 'd
rop') %>%
    group_by(team, player_name, player_number) %>%
    summarise(total_sets_played = sum(total_sets_played), .groups = 'd
rop')

  # 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_i
d) %>% 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_bloc
ks / 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)
}

# Save the block statistics as a CSV file
write.csv(block_stats, "block_stats.csv", row.names = FALSE)
```

**Output**: The output of the code will be a dataset structured as follows:

| team | player_ name | player_ number | total_ sets_ played | total_ blocks_ attempted | perfect_ blocks | player_ id | efficacy_ index | points_ per_set_ block |
|---|---|---|---|---|---|---|---|---|
| Trentino | Marko Podrascanin | 18 | 73 | 216 | 50 | 116525 | 0.23 | 0.68 |
| Perugia | Wilfredo Leon Venero | 9 | 31 | 33 | 8 | 119646 | 0.24 | 0.26 |
| Piacenza | Robertlandy Simon Aties | 13 | 86 | 277 | 52 | 114422 | 0.19 | 0.60 |

**Alternative approach**: If the user does not have access to data from multiple matches to calculate the **efficacy index**, some statistics can be found, for the SuperLega, on the official website[4].

In this case, the efficacy index will not be available. The user can use the **point per set statistic** as an alternative metric. While this can provide a general indication of performance, it does not provide the same level of detail as the efficacy index.

***superlega_roster_23_24.xlsx***

The `roster_file` is required to match player names with their corresponding roles, number and teams. In this case, the file corresponds to the rosters of the 2023/2024 regular season published on the official LegaVolley website[5]. Proper matching between player number and team in both the roster file and the match files is crucial to avoid mismatches in player data. It must include the following columns, as shown in the example table below:

[4]https://ww.legavolley.it/statistiche/?TipoStat=1.1&Serie=1&AnnoInizio=2023&Fase=1&Giornata=0&Squadra=TN-ITAS
[5]https://www.legavolley.it/

| player_name | player_role | player_number | team |
|---|---|---|---|
| Riccardo Sbertoli | Setter | 6 | Itas Trentino |
| Matteo Piano | Middle Blocker | 11 | Allianz Milano |
| Mattia Bottolo | Spiker | 21 | Cucine Lube Civitanova |

This information can be collected from the official team rosters, published by each Superlega team.

***Loading***

Once the `block_stats.csv` and `superlega_roster_23_24.xlsx` files have been generated, users can load them using `read.csv` and `readxl()` functions:

```
# Load roster file
roster_file <- read_excel("path/to/superlega_roster_23_24.xlsx")

# Load block statistics
block_stats <- read.csv("path/to/block_stats.csv")

# Check the structure
str(block_stats)
str(roster_file)

# View the first few rows
head(block_stats)
head(roster_file)
```

## 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 of transforming 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.

### 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`) within each rally. 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) %>%

    # 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(te
am)))) %>%
    ungroup()
}
```

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

**How to use the function**:

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

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

**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.

```
add_context_variables <- function(px, roster_file, high_pressure_thres
hold = 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_phas
e", "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(missin
g_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, mi
ssing = 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_sc
ore >= 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_tea
m_score >= 12))
            )
          ),
        1L, 0L
      ),

      # Game phase (serve or reception)
```

```r
      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 be 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: Binary variables 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.

**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.

```
add_skill_variables <- function(px) {

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

  # Add skill-related variables
  px <- px %>%
    dplyr::mutate(
      # Serve
      serve_player_name     = dplyr::case_when(skill == "Serve" ~ play
er_name, TRUE ~ NA_character_),
      serve_evaluation_code = dplyr::case_when(skill == "Serve" ~ eval
uation_code, TRUE ~ NA_character_),
      serve_start_zone      = dplyr::case_when(skill == "Serve" ~ star
t_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 == "Receptio
n" ~ player_name, TRUE ~ NA_character_),
      reception_evaluation_code   = dplyr::case_when(skill == "Receptio
n" ~ evaluation_code, TRUE ~ NA_character_),
      reception_serve_start_zone = dplyr::case_when(skill == "Receptio
n" ~ start_zone, TRUE ~ NA_integer_),
      reception_start_zone        = dplyr::case_when(skill == "Receptio
n" ~ end_zone, TRUE ~ NA_integer_),

      # Set
      set_player_name     = dplyr::case_when(skill == "Set" ~ player_n
ame, TRUE ~ NA_character_),
      set_evaluation_code = dplyr::case_when(skill == "Set" ~ evaluati
on_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" ~ pl
ayer_name, TRUE ~ NA_character_),
      attack_evaluation_code = dplyr::case_when(skill == "Attack" ~ ev
aluation_code, TRUE ~ NA_character_),
      attack_start_zone      = dplyr::case_when(skill == "Attack" ~ st
art_zone, TRUE ~ NA_integer_),
      attack_end_zone        = dplyr::case_when(skill == "Attack" ~ en
d_zone, TRUE ~ NA_integer_),

      # Block
      block_player_name     = dplyr::case_when(skill == "Block" ~ play
er_name, TRUE ~ NA_character_),
      block_evaluation_code = dplyr::case_when(skill == "Block" ~ eval
uation_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" ~ evalua
tion_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_zo
ne, 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**:

```
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: .

### 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.

```
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_s
erve_start_zone", "reception_start_zone",
    "set_player_name", "set_evaluation_code", "set_start_zone", "set_e
nd_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_star
t_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
```

```r
      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, n
a.rm = TRUE),
      visiting_score_start_of_point = max(visiting_score_start_of_poin
t, 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)
```

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

**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 replaced in the correct positions. Moreover, it updates the roles of the spiker in the opposite position to the player with the role of "Opposite" (based on the setter's location on the court).

```
assign_info_to_positions <- function(rally_data, roster_file) {
  join_and_rename <- function(data, number_col, team_col, new_name, ne
w_role) {
    data %>%
      left_join(
        roster_file %>% select(player_name, player_number, team, playe
r_role),
        by = setNames(c("player_number", "team"), c(number_col, team_c
ol))
      ) %>%
      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_r
ole") %>%
    join_and_rename("home_p2", "home_team", "home_p2_name", "home_p2_r
ole") %>%
    join_and_rename("home_p3", "home_team", "home_p3_name", "home_p3_r
ole") %>%
    join_and_rename("home_p4", "home_team", "home_p4_name", "home_p4_r
ole") %>%
    join_and_rename("home_p5", "home_team", "home_p5_name", "home_p5_r
ole") %>%
    join_and_rename("home_p6", "home_team", "home_p6_name", "home_p6_r
ole") %>%
    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 & visiti
ng_p1_role == "Spiker", "Opposite", visiting_p1_role),
      visiting_p2_role = ifelse(visiting_setter_position == 5 & visiti
ng_p2_role == "Spiker", "Opposite", visiting_p2_role),
      visiting_p3_role = ifelse(visiting_setter_position == 6 & visiti
ng_p3_role == "Spiker", "Opposite", visiting_p3_role),
      visiting_p4_role = ifelse(visiting_setter_position == 1 & visiti
ng_p4_role == "Spiker", "Opposite", visiting_p4_role),
      visiting_p5_role = ifelse(visiting_setter_position == 2 & visiti
ng_p5_role == "Spiker", "Opposite", visiting_p5_role),
      visiting_p6_role = ifelse(visiting_setter_position == 3 & visiti
ng_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".

### Assign block statistics to opponent

The `assign_opponent_block_efficacy()` function assigns efficacy block efficacy 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, ou
r_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$efficacy_index[match(paste(visiting_p2, visi
ting_team), paste(block_stats_clean$player_number, block_stats_clean$t
eam))],
        block_stats_clean$efficacy_index[match(paste(home_p2, home_tea
m), paste(block_stats_clean$player_number, block_stats_clean$team))]
      ),

      opponent_efficacy_index_p3 = ifelse(
        touching_team_is_home == 1,
        block_stats_clean$efficacy_index[match(paste(visiting_p3, visi
ting_team), paste(block_stats_clean$player_number, block_stats_clean$t
eam))],
        block_stats_clean$efficacy_index[match(paste(home_p3, home_tea
m), paste(block_stats_clean$player_number, block_stats_clean$team))]
      ),

      opponent_efficacy_index_p4 = ifelse(
        touching_team_is_home == 1,
        block_stats_clean$efficacy_index[match(paste(visiting_p4, visi
ting_team), paste(block_stats_clean$player_number, block_stats_clean$t
eam))],
        block_stats_clean$efficacy_index[match(paste(home_p4, home_tea
m), paste(block_stats_clean$player_number, block_stats_clean$team))]
      )
    ) %>%
    # Replace NA with 0
    mutate(
      opponent_efficacy_index_p2 = ifelse(is.na(opponent_efficacy_inde
x_p2), 0, opponent_efficacy_index_p2),
      opponent_efficacy_index_p3 = ifelse(is.na(opponent_efficacy_inde
x_p3), 0, opponent_efficacy_index_p3),
      opponent_efficacy_index_p4 = ifelse(is.na(opponent_efficacy_inde
x_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 = "Our Team")
```

**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 availalbe, the value will be set to 0.

### Assign block statistics to opponent effective positions

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

```
remap_opponent_block_to_effective_positions <- function(rally_data, ou
r_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, phas
e),
      eff_pos_p3 = mapply(get_effective_position, opp_p3_role, 3, phas
e),
      eff_pos_p4 = mapply(get_effective_position, opp_p4_role, 4, phas
e)
    ) %>%
    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 opponenet players' roles and nominal positions, team side indicator, phase of the game and pre-assigned opponenet block indices (`opponent_efficacy_index_p`).

**How to use**:

```
rally_data <- remap_opponent_block_to_effective_positions(rally_data,
our_team = "Our Team")
```

**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.

## Conclusion

These R-based data science pipelines that process volleyball match data offer a comprehensive approach for analyzing and understanding team and player performance from match data extracted from DataVolley software.

By integrating several key functions, this pipeline processes play-by-play match data, calculates player statistics, and assigns various metrics to positions, roles, and phases of the game. Each function focuses on a specific aspect, with the aim of returning a comprehensive picture of the game's dynamics. These functions help transform raw match data into a structured dataset, which allows for detailed analysis of rallies.

The user can customize the analysis by changing easily team or leagues, making this pipeline flexible and adaptable.

To ensure optimal results, users should check that the input data are complete and accurately formatted.

# R-driven Volleyball Data Pipeline – Working Example

## Introduction to the Application R Markdown

After having introduced the help functions to transform data, an application on real data of a match of 23 August 2022 between Nicaragua and Cuba is proposed below.

This section shows how the pipeline is executed sequentially to produce the dataset with all the transformations, useful for producing further statistical or predictive analyses.

In the working example file, there are some duplications that were present also in the help file, however, this choice is intentional. The idea is to make each document independently readable and usable. The help file provides a detailed explanation of the purpose and logic of each function, while the application file shows how to apply them through concrete examples.
By maintaining both files self-contained, users can either understand and implement the functions by consulting the application directly or refer to the help file for a deeper understanding, without having to switch between the two.

## 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.

## Required Packages

The script requires the following R packages to be installed:

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

# 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 instal
led
if (!requireNamespace("datavolley", quietly = TRUE)) {
  install.packages("datavolley", repos = c("https://openvolley.r-unive
rse.dev", "https://cloud.r-project.org"))
}
library(datavolley)

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

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

## File Input

Three input files are needed to make the functions work:

3. **Match file**: `.dvw` file containing detailed match information. They can be downloaded from the DataVolley software.

4. **Roster file**: A file containing information about the players on each team for the season of interest. Information on players should be completed 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.

5. **Block statistics file**: A file containing players block statistics.

# Data Flow

## 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 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 × 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-~~~18
D NICARAGUA
## 6 8984c1f…        0 NA                     1        726 a10RH+~~~18
D Cuba Norce…
## # ℹ 78 more variables: player_number <int>, player_name <chr>, play
er_id <chr>,
## #   skill <chr>, skill_type <chr>, evaluation_code <chr>, evaluatio
n <chr>,
## #   attack_code <chr>, attack_description <chr>, set_code <chr>,
## #   set_description <chr>, set_type <chr>, start_zone <int>, end_zo
ne <int>,
## #   end_subzone <chr>, end_cone <int>, skill_subtype <chr>, num_pla
yers <chr>,
## #   num_players_numeric <int>, special_code <chr>, timeout <lgl>,
## #   end_of_set <lgl>, substitution <lgl>, point <lgl>, home_team_sc
ore <int>, …

# 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 IHerre
ra)
## 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 × 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-~~~18
D NICARAGUA
## 6 8984c1f…        0 NA                     1        726 a10RH+~~~18
D Cuba Norce…
## # ℹ 78 more variables: player_number <int>, player_name <chr>, play
er_id <chr>,
## #   skill <chr>, skill_type <chr>, evaluation_code <chr>, evaluatio
n <chr>,
## #   attack_code <chr>, attack_description <chr>, set_code <chr>,
## #   set_description <chr>, set_type <chr>, start_zone <int>, end_zo
ne <int>,
## #   end_subzone <chr>, end_cone <int>, skill_subtype <chr>, num_pla
yers <chr>,
## #   num_players_numeric <int>, special_code <chr>, timeout <lgl>,
## #   end_of_set <lgl>, substitution <lgl>, point <lgl>, home_team_sc
ore <int>, …
```

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

## 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, it 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.

```
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 = 'd
rop') %>%
    group_by(team, player_name, player_number) %>%
    summarise(total_sets_played = sum(total_sets_played), .groups = 'd
rop')

  # 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_i
d) %>% 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_bloc
ks / 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 × 9
##   team         player_name player_number total_sets_played total_blo
cks_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 Blocke
r",
  "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 × 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
```

## 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 of transforming 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.

### 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) %>%

    # 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(te
am)))) %>%
    ungroup()
}
```

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

**How to use the function**:

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

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

```
## # A tibble: 5 × 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
```

### 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.

```
add_context_variables <- function(px, roster_file, high_pressure_thres
hold = 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_phas
e", "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(missin
g_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, mi
ssing = 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_sc
ore >= 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_tea
m_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**:

```
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 be 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 × 6
##   is_home hitters_front_row is_breakpoint is_sideout is_high_pressu
re phase
##     <int>             <int>         <int>      <int>             <in
t> <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
```

**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.

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

  # Add skill-related variables
  px <- px %>%
    dplyr::mutate(
      # Serve
      serve_player_name      = dplyr::case_when(skill == "Serve" ~ play
er_name, TRUE ~ NA_character_),
      serve_evaluation_code = dplyr::case_when(skill == "Serve" ~ eval
uation_code, TRUE ~ NA_character_),
      serve_start_zone       = dplyr::case_when(skill == "Serve" ~ star
t_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 == "Receptio
n" ~ player_name, TRUE ~ NA_character_),
      reception_evaluation_code   = dplyr::case_when(skill == "Receptio
n" ~ evaluation_code, TRUE ~ NA_character_),
      reception_serve_start_zone = dplyr::case_when(skill == "Receptio
n" ~ start_zone, TRUE ~ NA_integer_),
      reception_start_zone        = dplyr::case_when(skill == "Receptio
n" ~ end_zone, TRUE ~ NA_integer_),

      # Set
      set_player_name      = dplyr::case_when(skill == "Set" ~ player_n
ame, TRUE ~ NA_character_),
      set_evaluation_code = dplyr::case_when(skill == "Set" ~ evaluati
on_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" ~ pl
ayer_name, TRUE ~ NA_character_),
      attack_evaluation_code = dplyr::case_when(skill == "Attack" ~ ev
aluation_code, TRUE ~ NA_character_),
      attack_start_zone       = dplyr::case_when(skill == "Attack" ~ st
art_zone, TRUE ~ NA_integer_),
      attack_end_zone         = dplyr::case_when(skill == "Attack" ~ en
d_zone, TRUE ~ NA_integer_),

      # Block
      block_player_name      = dplyr::case_when(skill == "Block" ~ play
er_name, TRUE ~ NA_character_),
      block_evaluation_code = dplyr::case_when(skill == "Block" ~ eval
uation_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" ~ evalua
tion_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_zo
ne, 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**:

```
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 × 4
##   serve_player_name serve_evaluation_code serve_start_zone serve_en
d_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
```

### 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.

```
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_s
erve_start_zone", "reception_start_zone",
    "set_player_name", "set_evaluation_code", "set_start_zone", "set_e
nd_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_star
t_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, n
a.rm = TRUE),
      visiting_score_start_of_point = max(visiting_score_start_of_poin
t, 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**:

```
rally_data <- aggregate_rally_data(px)

## `summarise()` has grouped output by 'match_id', 'rally_id', 'posses
sion_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.

```
head(rally_data)

## # A tibble: 6 × 57
##   match_id  rally_id possession_id team  serve_player_name serve_ev
aluation_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_zon
e <int>,
## #   set_end_zone <int>, attack_player_name <chr>, attack_evaluation
_code <chr>,
## #   attack_start_zone <int>, attack_end_zone <int>, block_player_na
me <chr>,
## #   block_evaluation_code <chr>, block_start_zone <int>, …
```

**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, ne
w_role) {
    data %>%
      left_join(
        roster_file %>% select(player_name, player_number, team, playe
r_role),
        by = setNames(c("player_number", "team"), c(number_col, team_c
ol))
      ) %>%
      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_r
ole") %>%
    join_and_rename("home_p2", "home_team", "home_p2_name", "home_p2_r
ole") %>%
    join_and_rename("home_p3", "home_team", "home_p3_name", "home_p3_r
ole") %>%
    join_and_rename("home_p4", "home_team", "home_p4_name", "home_p4_r
ole") %>%
    join_and_rename("home_p5", "home_team", "home_p5_name", "home_p5_r
ole") %>%
    join_and_rename("home_p6", "home_team", "home_p6_name", "home_p6_r
ole") %>%
    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 & visiti
ng_p1_role == "Spiker", "Opposite", visiting_p1_role),
      visiting_p2_role = ifelse(visiting_setter_position == 5 & visiti
ng_p2_role == "Spiker", "Opposite", visiting_p2_role),
      visiting_p3_role = ifelse(visiting_setter_position == 6 & visiti
ng_p3_role == "Spiker", "Opposite", visiting_p3_role),
      visiting_p4_role = ifelse(visiting_setter_position == 1 & visiti
ng_p4_role == "Spiker", "Opposite", visiting_p4_role),
      visiting_p5_role = ifelse(visiting_setter_position == 2 & visiti
ng_p5_role == "Spiker", "Opposite", visiting_p5_role),
      visiting_p6_role = ifelse(visiting_setter_position == 3 & visiti
ng_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".

### Assign block statistics to the 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, ou
r_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$efficacy_index[match(paste(visiting_p2, visi
ting_team), paste(block_stats_clean$player_number, block_stats_clean$t
eam))],
        block_stats_clean$efficacy_index[match(paste(home_p2, home_tea
m), paste(block_stats_clean$player_number, block_stats_clean$team))]
      ),

      opponent_efficacy_index_p3 = ifelse(
        touching_team_is_home == 1,
        block_stats_clean$efficacy_index[match(paste(visiting_p3, visi
ting_team), paste(block_stats_clean$player_number, block_stats_clean$t
eam))],
        block_stats_clean$efficacy_index[match(paste(home_p3, home_tea
m), paste(block_stats_clean$player_number, block_stats_clean$team))]
      ),

      opponent_efficacy_index_p4 = ifelse(
        touching_team_is_home == 1,
        block_stats_clean$efficacy_index[match(paste(visiting_p4, visi
ting_team), paste(block_stats_clean$player_number, block_stats_clean$t
eam))],
        block_stats_clean$efficacy_index[match(paste(home_p4, home_tea
m), paste(block_stats_clean$player_number, block_stats_clean$team))]
      )
    ) %>%
    # Replace NA with 0
    mutate(
      opponent_efficacy_index_p2 = ifelse(is.na(opponent_efficacy_inde
x_p2), 0, opponent_efficacy_index_p2),
      opponent_efficacy_index_p3 = ifelse(is.na(opponent_efficacy_inde
x_p3), 0, opponent_efficacy_index_p3),
      opponent_efficacy_index_p4 = ifelse(is.na(opponent_efficacy_inde
x_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 × 3
##   opponent_efficacy_index_p2 opponent_efficacy_index_p3 opponent_ef
ficacy_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: ¹opponent_efficacy_index_p4
```

**Assign block statistics to opponent effective positions**

The `assign_opponent_effective_block_positions()` function assigns the opponent's block efficacy index to their effective court positions based on player's role. It dynamically adjusts 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**.

```
remap_opponent_block_to_effective_positions <- function(rally_data, ou
r_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, phas
e),
      eff_pos_p3 = mapply(get_effective_position, opp_p3_role, 3, phas
e),
      eff_pos_p4 = mapply(get_effective_position, opp_p4_role, 4, phas
e)
    ) %>%
    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 × 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: ¹opponent_effective_efficacy_index_p2,
## #   ²opponent_effective_efficacy_index_p3,
## #   ³opponent_effective_efficacy_index_p4
```

## 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, users can see how each function contributes to the transformation of raw DataVolley input into a clean, analysis-ready dataset.

# PREDICTIVE MODELLING

## Estimating the set target zone

In addition to the construction of the pipeline to modify and get the dataset enriched and divided by sequence of three or less touches by the same team, a test was done to develop a predictive model of the final setting zone.

The response variable to be predicted is “set_end_zone”, that indicates with a number in which area the ball has been set to allow the spiker to carry out the attack.

The prediction of the final setting zone is very useful to allow to understand offensive patterns of the opponent team and consequently adapt tactical game schemes to better manage defensive situations.

Before building a predictive model to predict the set end zone, a preparatory phase was necessary to ensure the quality and completeness of the input data. This phase included:

1. Defining the relevant features to include in the model.
2. Handling missing values.
3. Transforming variables into suitable formats for modelling.

## Selection of Predictors

To predict the set target zone and create the predictive model, a careful selection and preparation of the predictive variables (X) was carried out following logics guided by the knowledge of the volleyball game and by analysing contextual factors that could influence the setter's decision.

The selected features included are:

- setter_position → Initial position of the setter at the beginning of the rally (a position that can range from 1 to 6), which directly impacts the movement of the setter and the number of the hitters available in the front line.

- hitters_front_row → Number of hitters in front row based on the position of the setter.
- ball_quality → A variable that includes the evaluation of the action before the set, either it is reception or dig. The evaluation could be:
    - = → Error: team can't receive or defence.
    - / → Very poor: the ball is sent directly in the opponent court side.
    - - →Poor: it allows only one mandatory attack.
    - ! →Good: the ball is received out of the 3-meter line, not all attack combination can be performed.
    - + → Positive: the ball is received in the 3-meter line, not all attack combination can be performed.
    - # → Perfect: it is possible to perform all attack combinations.
- set_start_zone → Initial position on the court of the setter at the time of the set.
- is_sideout, is_breakpoint, is_high_pressure → Binary variables that indicate whether the game phase can be difficult on a psychological level.
- opponent_effective_efficacy_index_p2, p3, p4 → Statistics on the opponent's block after each player has gone to the position of competence according to the role (middle blocker in place 3, opposite in place 2, spiker in place 4, disregarding exceptions).

These variables have been chosen to reflect not only the situation in a part of the field (player positions or the number of attackers in the front line), but also external conditions (opponent's wall, reception or defence quality, pressure situations) that can influence the setter’s choices.

## Handling missing values

The biggest challenge of this prediction model was the high number of missing values (NA) to be managed. Rather than eliminating the entire row, which would have led to a reduction in the training set, a logic for imputing values in some cases was followed.

The key steps included:

- Rows where the independent variable (set_end_zone) had no values were excluded.
- Missing values in the opponent's block variables (opponent_effective_efficacy_index_p2/p3/p4) have been replaced with 0, assuming neutral impact of the block.
- The rows in which hitters_front_row had NA were eliminated because they were considered essential to set decision-making.
- Missing values in the new ball_quality variable were handled with some considerations:
  - If the block made by the team results in an error (=) or in an invasion (/) the row was eliminated.
  - If set_start_zone was central (zone 3), it was assumed that the ball_quality was excellent (#).
- In cases where the variable set_start_zone had missing values, some logic was followed to impute them:
  - If ball_quality was an error (=) or very poor (/), not allowing to set, but only to do a freeball, the row was eliminated.

    - If ball_quality was poor (!) or sent out of the 3-meter line (-) set_start_zone became 10, suggesting a difficult situation.
    - If ball_quality was excellent (#), the central zone 3 was assigned, assuming the setter had full control, having the possibility to set wherever he wants.

## Formatting selected variables

To ensure compatibility with classification models (e.g. Random Forest) the selected variables have been recoded and factored where necessary:

- set_end_zone, is_sideout, is_breakpoint, is_high_pressure have been converted into factors.
- ball_quality has been treated as an ordered factor with increasing quality (=, /, -,!, +, #).

This preparation ensured that all variables could be seen correctly by the model and used in training of the predictive model.

## Future research directions

Starting from the data transformed using the developed functions, and after having identified which variables to take as predictors and how to manage the missing values appropriately, various are the possible future developments for the research. A possibility could involve the implementation of machine learning models to predict the final set zone. A technique that could be explored may be the random forest model, it could be a good starting point, thanks to the robustness, interpretability and ability to manage both categorical and numerical variables.
Another approach that can be used are the neural networks, especially in cases where the

aim is to model complex and nonlinear relationships that may not be easily captured by simpler and traditional models. For instance, feedforward neural networks could be trained to detect hidden patterns in high-dimensional feature spaces, especially when the number of matches grows over time. If sequential data will be introduced, such as rotational chronologies, recurrent neural networks (RNNs) or long-short-term memory (LSTM) could be explored to incorporate temporal dependencies into the predictions.

Before applying these models, it must always be remembered that it is essential to have quality data and that it is necessary to address the possible imbalance of classes, for example if some areas are much more frequent than others, and to adopt strict validation to assess whether the model generalizes in the right way.

# CONCLUSIONS

This thesis has addressed a critical gap in volleyball data analytics by developing an R-driven pipeline working to transform action-level data into a structured dataset divided by a sequence of passages by row. Starting from the foundational work of Ben Raymond and his R “datavolley” package, this work creates new functions to facilitate comprehensive analysis allowing insights into entire sequences of play.

The core of the developed pipeline helps create a robust and flexible framework, which can improve the usability of data, helping coaches and analysts to explore more advanced metrics, tactical patterns and team dynamics with greater accuracy.

Adding and enriching the dataset, the approach tries to overcome the limitations of the previous data arrangement and opens to new possibilities for meaningful statistical analysis in volleyball.

The working practical example proposed in this work demonstrates the effectiveness and versatility of the pipeline through different scenarios and datasets, laying the foundation for future developments and applications in volleyball sport analysis.

Overall, this work helps to reduce the gap between raw data downloaded from "DataVolley" and advanced performance analysis. By making the dataset structured and more accessible, it gives the possibility to analysts to explore better match dynamics, study game performance and tactics, giving them the possibility to provide coaches with actionable insights to support tactical and strategic decisions.

# FUTURE WORKS

Beginning from the results of this thesis, many are the possible directions that can be pursued for further advanced statistical analysis:

- Integrate more comprehensive data: improve data collection to avoid and reduce missing values, capture and add information on specific player positions or psychological data.
- Improve the management of missing data: by integrating and exploring new techniques or probabilistic modelling to improve the management of NA values, especially for variables of interest such as set_end_zone.
- Develop more complex methods for predictive models: integrate and apply advanced modelling techniques, including Random Forests, Neural Networks or ensemble learning to improve both accuracy and interpretability. These models could handle complex volleyball data, and the results could assist the coach in anticipating opponents' strategies and support decisions.
- Integrate and improve real-time analytics: Enrich the functions to support real-time analytics helping coaches make tactical decisions live during the match.

These future steps aim to make the analysis of volleyball data more accurate, accessible, but also more applicable, supporting teams to improve offensive and defensive strategies, with the possibility to benefits meaningfully from the analysis.

# REFERENCES

1. Wang, W. (2023). The role of big data in enhancing sports performance analytics. International IT Journal of Research.
2. Wang, R. (2025). Application and Analysis of Big Data Analysis in Sports. Applied Mathematics and Nonlinear Sciences.
3. Xarles, A, Escalera, S., Moeslund, T.B. & Clapés A. (2025), Action valuation in sports: A survey. arXiv preprint.
4. Reis, F. J. J., Alaiti, R. K., Vallio, C. S., & Hespanhol, L. (2024). Artificial intelligence and machine learning approaches in sports: Concepts, applications, challenges, and future perspectives. *Brazilian Journal of Physical Therapy.*
5. Ambasht, A. (2023). *Real-time data integration and analytics: Empowering data-driven decision making*. International Journal of Computer Trends and Technology.
6. Tang, X., Long, B., & Zhou, L. (2024). *Real-time monitoring and analysis of track and field athletes based on edge computing and deep reinforcement learning algorithm*. arXiv.
7. *How real-time data analysis in sport shapes critical coaching decisions*. (2025, September 2). *Sports Tech Today*. https://www.sportstechtoday.com/how-real-time-data-analysis-in-sport-shapes-critical-coaching-decisions.
8. Mateus, N., Abade, E., Coutinho, D., Gómez, M.-Á., Lago-Peñas, C., & Sampaio, J. (2024). Empowering the Sports Scientist with Artificial Intelligence in Training, Performance, and Health Management.
9. Russell, S., Jenkins, D., Rynne, S., & Halson, S. (2019). What is mental fatigue in elite sport? Perceptions from athletes and staff. *European Journal of Sport Science*

10. Yuan, J., Zhang, L., Zhou, X., & Chen, H. (2021). Mental fatigue and sports performance of athletes: Theoretical explanation, influencing factors, and intervention methods. *Behavioral Sciences.*
11. Lambers, R., et al. (2023). *How to schedule the Volleyball Nations League*. *Journal of Scheduling.*
12. García-Pinillos, F., et al. (2023). *Assessing and Monitoring Physical Performance Using Wearable Technologies in Volleyball Players: A Systematic Review*. *Applied Sciences.*
13. KINEXON Sports (2025). *Innovative Load & Performance Tracking for Volleyball*. https://kinexon-sports.com/sports/volleyball.
14. KINEXON Sports (2025). *The Most Accurate Volleyball Analytics in the Game*. https://kinexon-sports.com/blog/volleyball-analytics.
15. Sanders, G. J., et al. (2025). *Impact of Early Season Jump Loads on Neuromuscular Performance in Division I Volleyball: Analyzing Force, Velocity, and Power from Countermovement Jump Tests*. *Translational Sports Medicine*.
16. Gabrio, A. (2020). *Bayesian hierarchical models for the prediction of volleyball results*. *Journal of Applied Statistics.*
17. Lalwani, A., et al. (2022). *Machine Learning in Sports: A Case Study on Using Explainable Models for Predicting Outcomes of Volleyball Matches*.
18. Xia, H., et al. (2023). *Advanced Volleyball Stats for All Levels: Automatic Setting Tactic Detection and Classification with a Single Camera.*
19. Rossi, A., et al. (2023). Wearable-based biomechanical monitoring in volleyball. *Journal of Biomechanics in Sport.*
20. Fuller, D., et al. (2024). Predictive models for sports injury prevention. *British Journal of Sports Medicine.*

21. Kohn, M., & van der Kamp, J. (2025). Mental fatigue indicators in team sports. *Psychophysiology of Performance*.

22. Data Project, "FIVB Men's World Championship 2018. https://dataproject.com/it/News.

23. Raymond, B., Ickowicz, A., & Widdison, T. Datavolley R package. *OpenVolley Project*. https://datavolley.openvolley.org/articles/datavolley.html.