

# Package ‘vald.extractor’

May 8, 2026

**Type** Package

**Title** Robust Pipeline for 'VALD' 'ForceDecks' Data Extraction and Analysis

**Version** 0.1.1

**Description** Provides a robust and reproducible pipeline for extracting, cleaning, and analyzing athlete performance data generated by 'VALD' 'ForceDecks' systems. The package supports batch-oriented data processing for large datasets, standardized data transformation workflows, and visualization utilities for sports science research and performance monitoring. It is designed to facilitate reproducible analysis across multiple sports with comprehensive documentation and error handling.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Imports** readxl, httr, jsonlite, data.table, ggplot2, dplyr, tidyr, stringr, lubridate, valdr, stats, utils

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**URL** <https://github.com/praveenmaths89/vald.extractor>

**BugReports** <https://github.com/praveenmaths89/vald.extractor/issues>

**NeedsCompilation** no

**Author** Praveen D Chougale [aut, cre],  
Usha Ananthakumar [aut]

**Maintainer** Praveen D Chougale <praveenmaths89@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-02-01 17:10:02 UTC

## Contents

classify_sports . . . . .	2
fetch_vald_batch . . . . .	3
fetch_vald_metadata . . . . .	4
patch_metadata . . . . .	5
plot_vald_compare . . . . .	6
plot_vald_trends . . . . .	8
split_by_test . . . . .	9
standardize_vald_metadata . . . . .	10
summary_vald_metrics . . . . .	11
<b>Index</b>	<b>13</b>

---

classify_sports	<i>Automated Sports Taxonomy Mapping</i>
-----------------	--

---

## Description

Applies regex-based pattern matching to standardize inconsistent sport/team naming conventions into a clean categorical variable. This is the core "value-add" for multi-sport organizations where team names may vary (e.g., "Football", "Soccer", "FSI" all map to "Football").

## Usage

```
classify_sports(
  data,
  group_col = "all_group_names",
  output_col = "sports_clean"
)
```

## Arguments

data	Data frame containing athlete metadata.
group_col	Character. Name of the column containing group/team names. Default is "all_group_names".
output_col	Character. Name for the new standardized sports column. Default is "sports_clean".

## Details

Classify Sports from Group Names

## Value

Data frame with an additional column containing standardized sports categories.

## Examples

```
if (FALSE) {  
  metadata <- standardize_vald_metadata(profiles, groups)  
  metadata <- classify_sports(metadata)  
  table(metadata$sports_clean)  
}
```

---

fetch\_vald\_batch

*Robust Batch Extraction of VALD Trials*

---

## Description

Implements chunked trial extraction from VALD ForceDecks API with fault-tolerant error handling. This function prevents timeout errors and memory issues when working with large datasets by processing data in manageable chunks.

## Usage

```
fetch_vald_batch(start_date, chunk_size = 100, verbose = TRUE)
```

## Arguments

start_date	Character string in ISO 8601 format (e.g., "2020-01-01T00:00:00Z"). The starting date for data extraction.
chunk_size	Integer. Number of tests to process per batch. Default is 100. Reduce this value if you experience timeout errors.
verbose	Logical. If TRUE, prints progress messages. Default is TRUE.

## Details

Fetch VALD ForceDecks Data in Batches

This function first retrieves all test metadata, then iterates through tests in chunks to fetch associated trial data. Each chunk is wrapped in a tryCatch block to ensure that errors in one chunk do not halt the entire extraction process.

The chunking strategy is essential for large organizations with thousands of tests, as it prevents API timeout errors and reduces memory pressure.

## Value

A list containing two data frames:

**tests** Data frame of all tests metadata

**trials** Data frame of all trials (individual repetitions) data

## Examples

```
if (FALSE) {  
  # Set VALD credentials first  
  valdr::set_credentials(  
    client_id = "your_client_id",  
    client_secret = "your_client_secret",  
    tenant_id = "your_tenant_id",  
    region = "aue"  
  )  
  
  # Fetch data from 2020 onwards in chunks of 100  
  vald_data <- fetch_vald_batch(  
    start_date = "2020-01-01T00:00:00Z",  
    chunk_size = 100  
  )  
  
  # Access tests and trials  
  tests_df <- vald_data$tests  
  trials_df <- vald_data$trials  
}
```

---

fetch\_vald\_metadata    *Retrieve Athlete Profiles and Group Assignments*

---

## Description

Authenticates with VALD API using OAuth2 client credentials flow and retrieves complete athlete profile and group membership data. This function handles token management, pagination, and robust JSON parsing.

## Usage

```
fetch_vald_metadata(  
  client_id,  
  client_secret,  
  tenant_id,  
  region = "aue",  
  verbose = TRUE  
)
```

## Arguments

client_id	Character. Your VALD API client ID.
client_secret	Character. Your VALD API client secret.
tenant_id	Character. Your VALD tenant ID.
region	Character. VALD region code (e.g., "aue" for Australia East). Default is "aue".
verbose	Logical. If TRUE, prints progress messages. Default is TRUE.

## Details

Fetch VALD Metadata via OAuth2

## Value

A list containing two data frames:

**profiles** Complete athlete profile data

**groups** Group/team membership data

## Examples

```
if (FALSE) {  
  metadata <- fetch_vald_metadata(  
    client_id = "your_client_id",  
    client_secret = "your_client_secret",  
    tenant_id = "your_tenant_id"  
  )  
  
  profiles <- metadata$profiles  
  groups <- metadata$groups  
}
```

---

patch\_metadata

*Fix Missing or Incorrect Athlete Demographics*

---

## Description

Allows users to provide an external Excel or CSV file containing corrected demographic information (e.g., sex, date of birth) for athletes with missing or incorrect data in the VALD system. This function merges the corrections and updates the master metadata.

## Usage

```
patch_metadata(  
  data,  
  patch_file,  
  patch_sheet = 1,  
  id_col = "profileId",  
  fields_to_patch = c("sex", "dateOfBirth"),  
  verbose = TRUE  
)
```

**Arguments**

data	Data frame. Master metadata or analysis dataset.
patch_file	Character. Path to Excel (.xlsx) or CSV (.csv) file containing corrections.
patch_sheet	Character or integer. For Excel files, which sheet to read. Default is 1 (first sheet).
id_col	Character. Name of the ID column in both data and patch_file. Default is "profileId".
fields_to_patch	Character vector. Column names to update from the patch file. Default is c("sex", "dateOfBirth").
verbose	Logical. If TRUE, prints progress messages. Default is TRUE.

**Details**

Patch Missing Metadata from External File

**Value**

Data frame with patched metadata.

**Examples**

```
if (FALSE) {
  # Create an Excel file with columns: profileId, sex, dateOfBirth
  # Then patch the metadata
  patched_data <- patch_metadata(
    data = athlete_metadata,
    patch_file = "corrections.xlsx",
    fields_to_patch = c("sex", "dateOfBirth")
  )

  # Check results
  table(patched_data$sex)
}
```

---

plot\_vald\_compare

*Boxplot Comparison of Metrics by Sport, Sex, or Team*

---

**Description**

Creates boxplots to compare performance metrics across different groups (e.g., sports, sex, teams). Useful for benchmarking and identifying performance differences between populations.

**Usage**

```
plot_vald_compare(  
  data,  
  metric_col,  
  group_col = "sports",  
  fill_col = "sex",  
  title = NULL,  
  y_label = NULL  
)
```

**Arguments**

data	Data frame. Test data with grouping variables and metrics.
metric_col	Character. Name of the metric to plot.
group_col	Character. Primary grouping variable (x-axis). Default is "sports".
fill_col	Character. Optional fill color grouping (e.g., "sex"). Default is "sex".
title	Character. Plot title. If NULL, auto-generates from metric name.
y_label	Character. Y-axis label. If NULL, uses metric_col.

**Details**

Compare Performance Across Groups

**Value**

A ggplot2 object.

**Examples**

```
if (FALSE) {  
  test_datasets <- split_by_test(final_analysis_data)  
  
  # Compare CMJ peak force across sports and sex  
  plot_vald_compare(  
    data = test_datasets$CMJ,  
    metric_col = "PEAK_FORCE_Both",  
    group_col = "sports",  
    fill_col = "sex",  
    title = "Peak Force Comparison by Sport and Sex"  
  )  
}
```

---

plot\_vald\_trends      *Time-Series Visualization of Performance Metrics*

---

### Description

Creates professional line plots showing how performance metrics change over time for individual athletes or groups. Useful for tracking training adaptations, injury recovery, and seasonal trends.

### Usage

```
plot_vald_trends(  
  data,  
  date_col = "Testdate",  
  metric_col,  
  group_col = NULL,  
  facet_col = NULL,  
  title = NULL,  
  smooth = FALSE  
)
```

### Arguments

data	Data frame. Test data with a date column and at least one metric.
date_col	Character. Name of the date column. Default is "Testdate".
metric_col	Character. Name of the metric to plot.
group_col	Character. Optional grouping variable (e.g., "profileId", "sports"). If provided, separate lines are drawn for each group.
facet_col	Character. Optional faceting variable (e.g., "sex"). Creates separate panels for each level.
title	Character. Plot title. If NULL, auto-generates from metric name.
smooth	Logical. If TRUE, adds a smoothed trend line. Default is FALSE.

### Details

Plot Longitudinal Trends for VALD Metrics

### Value

A ggplot2 object.

**Examples**

```

if (FALSE) {
  test_datasets <- split_by_test(final_analysis_data)

  # Plot individual athlete trends
  plot_vald_trends(
    data = test_datasets$CMJ,
    metric_col = "PEAK_FORCE_Both",
    group_col = "profileId",
    facet_col = "sex"
  )

  # Plot sport-level averages
  sport_avg <- test_datasets$CMJ %>%
    group_by(Testdate, sports) %>%
    summarise(avg_force = mean(PEAK_FORCE_Both, na.rm = TRUE))

  plot_vald_trends(
    data = sport_avg,
    date_col = "Testdate",
    metric_col = "avg_force",
    group_col = "sports"
  )
}

```

---

split\_by\_test

*Generic Test-Type Splitting with Suffix Removal*


---

**Description**

Takes a master wide-format dataset and returns a named list of data frames, one per test type (e.g., CMJ, DJ, ISO). Crucially, this function automatically strips the test-type suffix from column names within each sub-dataframe, enabling generic analysis code that works across all test types.

This implements the "DRY" (Don't Repeat Yourself) principle by allowing users to write one analysis function that works for any test type.

**Usage**

```
split_by_test(data, metadata_cols = NULL, verbose = TRUE)
```

**Arguments**

data	Data frame. Wide-format test data with columns ending in test type suffixes (e.g., "PEAK_FORCE_Both_CMJ").
metadata_cols	Character vector. Column names to retain as metadata in each split dataset. Default includes common identifiers and demographics.
verbose	Logical. If TRUE, prints progress messages. Default is TRUE.

**Details**

Split Wide-Format Data by Test Type

**Value**

Named list of data frames, one per test type. Each data frame contains:

- All metadata columns
- Test-specific metrics with suffixes removed (e.g., "PEAK\_FORCE\_Both")

**Examples**

```
if (FALSE) {
  # After joining tests, trials, and metadata into wide format
  test_datasets <- split_by_test(
    data = final_analysis_data,
    metadata_cols = c("profileId", "sex", "Testdate", "age", "sports")
  )

  # Access individual test datasets
  cmj_data <- test_datasets$CMJ
  dj_data <- test_datasets$DJ

  # Note: Column names are now generic (e.g., "PEAK_FORCE_Both" not "PEAK_FORCE_Both_CMJ")
  # This allows you to write one function that works for all test types
}
```

---

standardize\_vald\_metadata

*Create Unified Athlete Metadata with Group Assignments*

---

**Description**

Processes raw profile and group data to create a clean, analysis-ready metadata table. Unnests group memberships, concatenates group names, and applies sports classification logic.

**Usage**

```
standardize_vald_metadata(profiles, groups, verbose = TRUE)
```

**Arguments**

profiles	Data frame. Raw profile data from <code>fetch_vald_metadata()</code> .
groups	Data frame. Raw group data from <code>fetch_vald_metadata()</code> .
verbose	Logical. If TRUE, prints progress messages. Default is TRUE.

**Details**

Standardize VALD Metadata

**Value**

A data frame with one row per athlete containing:

**profileId** Unique athlete identifier

**givenName, familyName** Athlete names

**dateOfBirth, sex** Demographic information

**all\_group\_names** Comma-separated list of all group memberships

**all\_group\_ids** Comma-separated list of all group IDs

**Examples**

```
if (FALSE) {  
  metadata <- fetch_vald_metadata(client_id, client_secret, tenant_id)  
  clean_metadata <- standardize_vald_metadata(  
    profiles = metadata$profiles,  
    groups = metadata$groups  
  )  
}
```

---

summary\_vald\_metrics *Dynamic Summary Table for Performance Metrics*

---

**Description**

Creates a comprehensive summary table showing mean, standard deviation, coefficient of variation, and sample size for all numeric performance metrics. Can be grouped by test type, sex, sport, or any combination thereof.

**Usage**

```
summary_vald_metrics(  
  data,  
  group_vars = c("sex", "sports"),  
  exclude_cols = c("profileId", "athleteId", "testId", "Testdate", "dateofbirth", "age",  
    "Weight_on_Test_Day"),  
  digits = 2  
)
```

**Arguments**

<code>data</code>	Data frame. Test data (typically from <code>split_by_test()</code> ).
<code>group_vars</code>	Character vector. Variables to group by. Default is <code>c("sex", "sports")</code> .
<code>exclude_cols</code>	Character vector. Column names to exclude from summary (typically metadata columns). Default includes common ID and date fields.
<code>digits</code>	Integer. Number of decimal places for rounding. Default is 2.

**Details**

Generate Summary Statistics for VALD Metrics

**Value**

Data frame with summary statistics (Mean, SD, CV, N) for each metric and grouping combination.

**Examples**

```
if (FALSE) {  
  test_datasets <- split_by_test(final_analysis_data)  
  cmj_summary <- summary_vald_metrics(  
    data = test_datasets$CMJ,  
    group_vars = c("sex", "sports")  
  )  
  print(cmj_summary)  
}
```

# Index

`classify_sports`, [2](#)

`fetch_vald_batch`, [3](#)

`fetch_vald_metadata`, [4](#)

`patch_metadata`, [5](#)

`plot_vald_compare`, [6](#)

`plot_vald_trends`, [8](#)

`split_by_test`, [9](#)

`standardize_vald_metadata`, [10](#)

`summary_vald_metrics`, [11](#)