

Package ‘immundata’

August 26, 2026

Title A Unified Data Layer for Large-Scale Single-Cell, Spatial and Bulk Immunomics

Version 0.1.0

Contact support@immunomind.com

Description Provides a unified data layer for single-cell, spatial and bulk T-cell and B-cell immune receptor repertoire data. Think AnnData or SeuratObject, but for AIRR data, a.k.a. Adaptive Immune Receptor Repertoire, VDJ-seq, RepSeq, or VDJ sequencing data.

License Apache License (>= 2)

URL <https://immunomind.github.io/docs/>,
<https://github.com/immunomind/immundata>

BugReports <https://github.com/immunomind/immundata/issues>

Encoding UTF-8

Depends R (>= 4.1.0), dplyr (>= 1.2.1), duckplyr (>= 1.2.1)

Imports checkmate, cli, dbplyr, glue, jsonlite, lifecycle, R6, readr, rlang, tibble, tools, utils

Suggests duckdb, rmarkdown, testthat (>= 3.0.0), Seurat

Config/Needs/website altdoc

Config/testthat/edition 3

Config/testthat/parallel true

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Vadim I. Nazarov [aut, cre] (ORCID:
<<https://orcid.org/0000-0003-3659-2709>>)

Maintainer Vadim I. Nazarov <support@immunomind.com>

Repository CRAN

Date/Publication 2026-08-26 10:00:02 UTC

Contents

agg_receptors	2
agg_repertoires	4
agg_strata	7
annotate_anndata	9
annotate_immundata	10
annotate_seurat	15
collect.ImmunData	16
compute.ImmunData	17
count.ImmunData	17
dimnames.ImmunData	19
dimnames<-.ImmunData	19
downsample_immundata	20
filter_immundata	22
from_immunarch	26
imd_rename_cols	27
imd_schema	28
ImmunData	29
make_default_preprocessing	33
make_receptor_schema	36
make_seq_options	38
mutate_immundata	39
names<-.ImmunData	44
print.ImmunData	44
read_immundata	45
read_manifest	48
read_repertoires	49
rename_strata	54
write_immundata	56
Index	60

agg_receptors	<i>Group AIRR sequence rows into receptors</i>
---------------	--

Description

agg_receptors() is a low-level function used during AIRR data ingestion. It decides which sequence rows represent the same biological receptor and adds package-standard identifiers and counts to the input table.

A receptor can be one chain or a pair of chains from the same cell. The schema argument defines which sequence features and loci make two receptors identical.

This function works with a prepared duckplyr table and returns a duckplyr table. It does not accept or return an [ImmunData](#) object. Most analysis workflows should provide the same arguments to [read_repertoires\(\)](#), which calls agg_receptors() during import.

Usage

```
agg_receptors(
  dataset,
  schema,
  barcode_col = NULL,
  count_col = NULL,
  locus_col = NULL,
  umi_col = NULL,
  verbose = getOption("immundata.verbose", TRUE)
)
```

Arguments

dataset	A duckplyr table containing AIRR sequence data, with one row per chain or bulk clonotype. It must contain the columns named in schema and in any of barcode_col, count_col, locus_col, and umi_col that are supplied.
schema	Definition of receptor identity. Supply either: • A character vector naming the features that must match, such as <code>c("v_call", "j_call", "junction_aa")</code>. • An object created by <code>make_receptor_schema()</code>. Its features define chain identity, while its optional chains select one locus or define a pair of loci. A schema can contain at most two chain entries. Use syntax such as <code>c("IGH", "IGL IGK")</code> to accept either IGH-IGL or IGH-IGK pairs.
barcode_col	Name of the column containing cell barcodes. Supply this for single-cell data. umi_col is then also required, and count_col cannot be supplied. If imd_filename is present, identical barcode values from different source files are treated as different cells. The default is NULL.
count_col	Name of the column containing non-negative abundance values in bulk repertoire data. These values are copied to imd_n_chains. count_col cannot be used together with barcode_col. The default is NULL.
locus_col	Name of the column containing loci such as "TRA", "TRB", or "IGH". It is required when schema specifies one or more chains. The column is renamed to the standard name locus when necessary. The default is NULL.
umi_col	Name of the column containing per-chain UMI or read counts. It is required when barcode_col is supplied and is used to choose one chain when a cell contains several chains from the same locus. The default is NULL.
verbose	Whether to print information about the selected processing mode and loci. Defaults to <code>getOption("immundata.verbose", TRUE)</code> .

Details

The receptor features are the columns that define the identity of one chain. Two chains with the same values in all feature columns receive the same receptor identity in a single-chain analysis. Common features include V gene, J gene, and CDR3 amino acid sequence.

The function supports three input modes:

- **Uncounted sequence table:** If neither `barcode_col` nor `count_col` is supplied, every input row is treated as one observed chain. A synthetic barcode is created for each row, and `imd_n_chains` is set to 1.
- **Bulk repertoire:** If `count_col` is supplied, every input row receives a synthetic barcode and its abundance is copied to `imd_n_chains`.
- **Single-cell repertoire:** If `barcode_col` is supplied, rows are grouped by cell. `umi_col` is required and `imd_n_chains` is set to 1 for every retained cell-chain observation.

When one chain is specified in schema, only that locus is retained. If a cell contains several chains from that locus, the row with the highest value in `umi_col` is retained. If the highest values are tied, the first row is retained.

When two chains are specified, only cells containing both requested loci are retained. The selected chains are paired by barcode, and both rows receive the same `imd_receptor_id`. Cells with incomplete pairs are excluded.

A relaxed pair such as `c("IGH", "IGL|IGK")` requires IGH and exactly one of the two alternative light-chain loci. Cells containing both IGL and IGK are excluded.

Numeric `imd_receptor_id` values identify receptors within the returned table. The particular number assigned to a receptor is not a biological identifier and may change when the data are aggregated again.

Value

A duckplyr table containing the retained input rows and these package-standard columns:

- `imd_receptor_id`: links rows that belong to the same receptor.
- `imd_barcode`: contains the input cell barcode, or a synthetic row-level barcode for uncounted and bulk data.
- `imd_chain_id`: identifies an individual retained chain row.
- `imd_n_chains`: contains 1 for uncounted and single-cell data, or the value from `count_col` for bulk data.
- `imd_count`: initialized to 0; receptor counts are calculated later by `agg_repertoires()`.

See Also

`read_repertoires()`, `make_receptor_schema()`, `agg_repertoires()`, `ImmunData`

Description

Use `agg_repertoires()` to define which receptor observations belong to the same biological repertoire and calculate receptor abundance within each repertoire.

Use this function after importing data without repertoire definitions, or when you want to redefine repertoires using sample information. One repertoire usually represents one biological sample. It can also represent one sample and time-point combination. The columns in schema define these groups.

The unit being defined is the repertoire. The function does not remove chain rows or redefine cells or receptors. It returns a new [ImmunData](#) object. The original object is not changed.

Usage

```
agg_repertoires(
  idata,
  schema = "repertoire_id",
  verbose = getOption("immundata.verbose", TRUE)
)
```

Arguments

<code>idata</code>	An ImmunData object containing receptor observations and the columns named in schema. This is usually created by read_repertoires() or read_immundata() .
<code>schema</code>	A non-empty character vector. One or more column names that together define a repertoire. For example, "Sample" creates one repertoire per sample, and <code>c("Sample", "TimePoint")</code> creates one repertoire per sample and time-point combination. The default is "repertoire_id"; this column must exist if the default is used.
<code>verbose</code>	A logical value. Accepted for consistency with other aggregation functions. It currently does not change the output. Defaults to <code>getOption("immundata.verbose", TRUE)</code> .

Details

The function calculates summaries at repertoire and receptor levels while keeping the original chain rows.

Value

A new [ImmunData](#) object with repertoire definitions and abundance statistics. Its repertoire summary contains the schema columns, `imd_repertoire_id`, `n_barcode`s, and `n_receptor`s. Its chain rows also contain `imd_repertoire_id`, `imd_count`, `imd_proportion`, and `n_repertoires`.

What the function calculates

The returned repertoire summary contains one row for each repertoire:

- `imd_repertoire_id`: a new integer identifier for the repertoire.

- `n_barcodes`: the number of observed cells for single-cell data, or the total abundance for bulk data.
- `n_receptors`: the number of distinct receptors in the repertoire.

The function also adds these values to each chain row:

- `imd_repertoire_id`: the repertoire containing the row.
- `imd_count`: the number of cells carrying that receptor in single-cell data, or its summed abundance in bulk data, within the repertoire.
- `imd_proportion`: the receptor's fraction of the repertoire, calculated as `imd_count / n_barcodes`.
- `n_repertoires`: the number of repertoires in which the receptor occurs.

Values calculated for a receptor are repeated on all chain rows belonging to that receptor in the same repertoire.

Calling `agg_repertoires()` again replaces previous repertoire definitions, receptor counts, proportions, and related strata summaries.

Backend and storage

Large-table calculations run on the duckplyr annotation table. The annotation data remain lazy when the input is lazy. The small repertoire summary is collected and stored in the returned object.

Aggregation can be expensive for a large dataset. After checking the result, consider saving it so later analyses do not repeat the calculation. Use `write_immundata(idata, tag = "by-sample")` to create a managed snapshot in the object's project home. Managed snapshots are versioned, so another write with the same tag creates a new version and keeps the earlier version.

Use `write_immundata(idata, output_folder = "path/to/result")` when you need a standalone saved state in a specific folder, for example to share it or to choose a new storage location. Unlike a managed snapshot, writing to an existing explicit folder replaces the ImmunData files in that folder. Both forms materialize pending duckplyr calculations and return a disk-backed object that can be reopened with `read_immundata()`.

See Also

[read_repertoires\(\)](#), [agg_strata\(\)](#), [write_immundata\(\)](#), [ImmunData](#)

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Create a small bulk T-cell receptor dataset from two biological samples
bulk_data <- tibble(
  Sample = c("Tumor", "Tumor", "Blood", "Blood"),
  cdr3_aa = c("CASSA", "CASSB", "CASSA", "CASSC"),
  v_call = c("TRBV1", "TRBV2", "TRBV1", "TRBV3"),
  abundance = c(20L, 5L, 4L, 6L)
)
```

```

bulk_file <- tempfile(fileext = ".tsv")
readr::write_tsv(bulk_data, bulk_file)

# Import receptors without defining repertoires
idata <- read_repertoires(
  path = bulk_file,
  schema = c("cdr3_aa", "v_call"),
  count_col = "abundance",
  repertoire_schema = NULL,
  output_folder = tempfile("immundata-example-")
)

# Define one repertoire for each biological sample
sample_repertoires <- idata |>
  agg_repertoires(schema = "Sample")

sample_repertoires$repertoires |>
  select(Sample, n_barcode, n_receptors) |>
  arrange(Sample)
# Expected result:
#   Sample n_barcode n_receptors
#   Blood      10         2
#   Tumor      25         2

# For example, CASSA forms 80% of the Tumor repertoire and 40% of the
# Blood repertoire. It occurs in two repertoires.

# For a large dataset, save the result as a managed snapshot so this
# aggregation does not need to run again.
saved_repertoires <- write_immundata(
  sample_repertoires,
  tag = "by-sample"
)

```

agg_strata

Group repertoires into biological strata

Description

Use `agg_strata()` to place sample repertoires into biological comparison groups, such as treatment arms, tissues, or disease groups.

Use this function after `agg_repertoires()` when several repertoires should be analysed as one group. A *stratum* contains every repertoire with the same value, or the same combination of values, in schema.

The unit being grouped is a whole repertoire. The function returns a new `ImmunData` object. The original object is not changed.

Usage

```
agg_strata(idata, schema, prefix = "Strata")
```

Arguments

idata	An ImmunData object with repertoires already defined. Use agg_repertoires() first if the object does not contain repertoires.
schema	A non-empty character vector. One or more repertoire-level columns that define a stratum. For example, use "Therapy" for treatment arms or c("Tissue", "Disease") for each tissue and disease combination. The columns must be present in <code>idata\$repertoires</code> .
prefix	A non-empty character string. Prefix for the automatic stratum labels. The default, "Strata", produces labels such as "Strata1" and "Strata2". You can use rename_strata() to assign meaningful labels later instead.

Details

If schema contains several columns, a separate stratum is created for each observed combination. For example, c("Tissue", "Therapy") can define separate blood and tumour strata within each treatment arm.

Calling `agg_strata()` again replaces the existing strata with groups defined by the new schema.

Value

A new [ImmunData](#) object in which every repertoire belongs to one stratum. The `$strata` table lists the strata, their defining biological values, and their automatic labels. Repertoire definitions and summary statistics are preserved.

Identifiers and storage

`imd_strata_id` is an internal identifier and can change when strata are rebuilt. It is added to the repertoire table and to the underlying chain annotations. `strata_name` is stored only in the smaller repertoire and strata tables.

Calling [agg_repertoires\(\)](#) again rebuilds the repertoires, so it removes the existing strata. Call `agg_strata()` again after redefining repertoires.

See Also

[agg_repertoires\(\)](#), [rename_strata\(\)](#), [ImmunData](#)

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Define sample repertoires using the biological metadata in the test data
idata <- get_test_idata() |>
```

```
agg_repertoires(c("Response", "Therapy"))

# Group the sample repertoires into treatment arms
treatment_groups <- idata |>
  agg_strata(schema = "Therapy")

treatment_groups$repertoires |>
  select(Therapy, Response, imd_strata_id, strata_name) |>
  arrange(imd_strata_id)
# Expected result:
#   Therapy Response imd_strata_id strata_name
#   CAR-T   PR           1 Strata1
#   ICI     FR           2 Strata2

# Each repertoire is now assigned to its treatment stratum. Any additional
# repertoire with the same Therapy value would receive the same stratum ID.
```

annotate_anndata	<i>Annotate an AnnData object from ImmunData (by barcode)</i>
------------------	---

Description

Copy selected columns from `idata$annotations` to `adata$obs`, matching by cell barcode (`adata$obs_names`).

Usage

```
annotate_anndata(idata, adata, cols)
```

Arguments

<code>idata</code>	An ImmunData object.
<code>adata</code>	An anndataR::AbstractAnnData object.
<code>cols</code>	Character vector with column names to transfer from <code>idata\$annotations</code> .

Value

The updated AnnData object.

annotate_immundata	<i>Add external information to ImmunData</i>
--------------------	--

Description

Use the `annotate_*`() functions to add information stored in another data frame to an [ImmunData](#) object. For example, you can add cell types from a single-cell analysis, antigen labels for receptors, or clinical information for samples.

When matching identifiers are unique, the functions keep every row in `idata`. When a row has no match in annotations, the new columns contain NA. Each function returns a new [ImmunData](#) object. The original object is not changed.

Usage

```
annotate_immundata(  
  idata,  
  annotations,  
  by,  
  keep_repertoires = TRUE,  
  remove_limit = FALSE,  
  conflicts = c("error", "replace")  
)  
  
annotate(  
  idata,  
  annotations,  
  by,  
  keep_repertoires = TRUE,  
  remove_limit = FALSE,  
  conflicts = c("error", "replace")  
)  
  
annotate_receptors(  
  idata,  
  annotations,  
  annot_col = imd_schema("receptor"),  
  keep_repertoires = TRUE,  
  remove_limit = FALSE,  
  conflicts = c("error", "replace")  
)  
  
annotate_barcode(  
  idata,  
  annotations,  
  annot_col = "<rownames>",  
  keep_repertoires = TRUE,
```

```

    remove_limit = FALSE,
    conflicts = c("error", "replace")
  )

  annotate_chains(
    idata,
    annotations,
    annot_col = imd_schema("chain"),
    keep_repertoires = TRUE,
    remove_limit = FALSE,
    conflicts = c("error", "replace")
  )

```

Arguments

<code>idata</code>	An ImmunData object.
<code>annotations</code>	A data frame containing the information to add. It must contain the columns used for matching and at most one row for each matching identifier or combination of identifiers.
<code>by</code>	A named character vector describing how columns are matched. Names are columns in <code>idata</code> ; values are the corresponding columns in <code>annotations</code> . For example, <code>c(Response = "response_code")</code> .
<code>keep_repertoires</code>	Whether to preserve existing repertoire and strata summaries without recalculation. The default is <code>TRUE</code> . If <code>FALSE</code> , these summaries and their derived annotation columns are removed.
<code>remove_limit</code>	Whether to allow an annotation table with 100 or more columns. The default is <code>FALSE</code> , which stops the operation for such tables. Set to <code>TRUE</code> only when the wide join is intentional.
<code>conflicts</code>	How to handle new annotation columns whose names already exist in <code>idata</code> . "error", the default, stops the operation. "replace" replaces existing columns that are not protected by <code>ImmunData</code> .
<code>annot_col</code>	Name of the identifier column in <code>annotations</code> . For <code>annotate_receptors()</code> and <code>annotate_chains()</code> , the default is the standard <code>ImmunData</code> receptor or chain identifier. For <code>annotate_barcode()</code> , the default "<rownames>" uses the row names of <code>annotations</code> . Supplying an explicit barcode column is usually clearer.

Details

The functions differ in how they select the columns used for matching. The rules for duplicate identifiers, column conflicts, and preserved summaries are the same for all functions.

Value

A new [ImmunData](#) object containing the added annotation columns. Existing repertoire and strata summaries are preserved when `keep_repertoires = TRUE`.

Choose a function

Use the function that matches the type of information you want to add:

- `annotate_barcodes()` matches cell or barcode identifiers.
- `annotate_receptors()` matches receptor identifiers. All rows belonging to a matched receptor receive the new information.
- `annotate_chains()` matches chain identifiers.
- `annotate()` matches any one or more columns that you specify in `by`.

The first three functions select the correct ImmunData identifier for you. `annotate_immundata()` is an alternative name for `annotate()`.

How matching works

For `annotate_barcodes()`, `annotate_receptors()`, and `annotate_chains()`, `annot_col` names the identifier column in annotations. For example, `annot_col = "barcode"` matches the barcode column in annotations with the barcode identifier in `idata`.

For general matching, supply `by` in the form `c(immunodata_column = "annotation_column")`. For example, `by = c(Response = "response_code")` matches the `Response` column in `idata` with the `response_code` column in annotations. To match columns with the same name, use a value such as `by = c(Response = "Response")`. You can include more than one pair of columns in `by`.

Columns from annotations that are not used for matching are added to the result. Rows in `idata` without a match receive `NA`. Rows in annotations without a match are ignored.

Annotation identifiers must be unique

annotations must contain at most one row for each identifier, or each combination of identifiers when matching several columns. For example, a barcode annotation table must contain at most one row per barcode.

The function does not check this rule because the annotation table may be very large. If an identifier occurs several times, the corresponding rows in `idata` are repeated. This can make receptor counts, proportions, and other summaries incorrect.

Existing annotation columns

By default, the function stops if a new annotation column has the same name as a column already present in `idata`. This prevents accidental replacement.

Use `conflicts = "replace"` to replace existing annotation columns. Columns that define receptors, repertoires, strata, or other ImmunData state are protected and cannot be replaced. The old column is removed before matching, so rows without a new match receive `NA`.

Repertoire and strata summaries

With the default `keep_repertoires = TRUE`, existing repertoire and strata summaries are copied to the new object without recalculation. Use this option when you are only adding information and the matching identifiers in annotations are unique.

Set `keep_repertoires = FALSE` when you plan to filter rows or define new repertoires using the added information. This removes existing repertoire and strata summaries and their derived columns. After annotation and filtering, use `agg_repertoires()` to define the new repertoires.

Very wide annotation tables

By default, the function stops when annotations contains 100 or more columns. Adding a very wide table, such as a complete gene-expression matrix, can be slow and require a large amount of memory. If you understand this cost, set `remove_limit = TRUE` to allow the operation.

See Also

[dplyr::left_join\(\)](#), [agg_repertoires\(\)](#), [filter_immundata\(\)](#), [mutate_immundata\(\)](#), [ImmunData](#)

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Load data included with immundata
idata <- get_test_idata()

# Add cell types by matching barcode identifiers
cell_labels <- tibble(
  barcode = c("S1_1", "S1_2"),
  cell_type = c("CD8 T cell", "CD4 T cell")
)

idata_with_cells <- idata |>
  annotate_barcodes(
    annotations = cell_labels,
    annot_col = "barcode"
  )

idata_with_cells |>
  collect() |>
  filter(imd_barcode %in% c("S1_1", "S1_2", "S1_3")) |>
  select(imd_barcode, cell_type) |>
  arrange(imd_barcode)
# Expected result:
#   imd_barcode cell_type
#   S1_1       CD8 T cell
#   S1_2       CD4 T cell
#   S1_3       NA

# Add antigen labels to selected receptors
receptor_labels <- tibble(
  receptor_id = c(738L, 1567L),
  antigen = c("CMV", "CMV")
)
```

```

)

idata_with_antigens <- idata |>
  annotate_receptors(
    annotations = receptor_labels,
    annot_col = "receptor_id"
  )

idata_with_antigens |>
  collect() |>
  filter(!is.na(antigen)) |>
  distinct(imd_receptor_id, cdr3_aa, antigen) |>
  arrange(imd_receptor_id)
# Expected result:
#   imd_receptor_id cdr3_aa      antigen
#               738 ASRAGAGTGELF  CMV
#               1567 ASFPVLSPYNEQF CMV

# Match columns with different names
response_info <- tibble(
  response_code = c("FR", "PR"),
  response_label = c("Full response", "Partial response")
)

idata_with_response <- idata |>
  annotate(
    annotations = response_info,
    by = c(Response = "response_code")
  )

idata_with_response |>
  collect() |>
  distinct(Response, response_label) |>
  arrange(Response)
# Expected result:
#   Response response_label
#   FR       Full response
#   PR       Partial response

# Replace an annotation column intentionally
revised_cell_labels <- tibble(
  barcode = c("S1_1", "S1_2"),
  cell_type = c("Cytotoxic T cell", "Helper T cell")
)

idata_with_revised_cells <- idata_with_cells |>
  annotate_barcodes(
    annotations = revised_cell_labels,
    annot_col = "barcode",
    conflicts = "replace"
  )

# Remove old repertoire summaries before defining repertoires by cell type

```

```
cell_repertoires <- idata |>
  annotate_barcodes(
    annotations = cell_labels,
    annot_col = "barcode",
    keep_repertoires = FALSE
  ) |>
  filter(!is.na(cell_type)) |>
  agg_repertoires(schema = "cell_type")

cell_repertoires$repertoires |>
  arrange(cell_type)
# Expected result:
#   imd_repertoire_id cell_type  n_barcodes n_receptors
#               1 CD4 T cell           1           1
#               2 CD8 T cell           1           1
```

annotate_seurat	<i>Annotate a Seurat object from ImmunData (by barcode)</i>
-----------------	---

Description

Copy selected columns from `idata$annotations` to Seurat metadata using the cell barcode. This is the simplest way to transfer data from `immundata` to Seurat object, e.g., for plotting data on UMAP.

Usage

```
annotate_seurat(idata, sdata, cols)
```

Arguments

<code>idata</code>	An ImmunData object.
<code>sdata</code>	A Seurat object (cells are columns; barcodes are <code>colnames(sdata)</code>).
<code>cols</code>	Character vector with column names to transfer from <code>idata\$annotations</code> . Typical choices: <code>"clonal_prop_bin"</code> or <code>"clonal_rank_bin"</code> .

Details

See functions `annotate_clonality_rank` and `annotate_clonality_prop` in `immunarch` package.

Value

The updated Seurat object with new metadata columns.

See Also

[ImmunData](#), [SeuratObject::AddMetaData](#)

Examples

```
## Not run:
# After annotating receptors:
idata <- annotate_clonality_prop(idata)

# Transfer the clonality bin to Seurat and plot:
sdata <- annotate_seurat(idata, sdata, cols = "clonal_prop_bin")
Seurat::DimPlot(sdata, reduction = "umap", group.by = "clonal_prop_bin", shuffle = TRUE)

# Alternative: rank bins
idata <- annotate_clonality_rank(idata, bins = c(10, 100))
sdata <- annotate_seurat(idata, sdata, cols = "clonal_rank_bin")

## End(Not run)
```

collect.ImmunData	<i>Collect ImmunData annotations</i>
-------------------	--------------------------------------

Description

Collects annotations from an ImmunData object and returns them as a tibble. Factor columns are converted to character.

Usage

```
## S3 method for class 'ImmunData'
collect(x, ...)
```

Arguments

x	ImmunData object.
...	Additional arguments passed to <code>dplyr::collect()</code> for <code>x\$annotations</code> .

Value

A tibble with collected annotations.

compute.ImmunData	<i>Compute ImmunData annotations</i>
-------------------	--------------------------------------

Description

Materializes the annotation table of an ImmunData object via `dplyr::compute()` and returns a new ImmunData.

Usage

```
## S3 method for class 'ImmunData'
compute(x, ...)
```

Arguments

x	ImmunData object.
...	Additional arguments passed to <code>dplyr::compute()</code> for x\$annotations.

Value

A new ImmunData object with computed annotations and the input repertoire, strata, schema, and provenance state preserved.

count.ImmunData	<i>Count chain rows in ImmunData</i>
-----------------	--------------------------------------

Description

Use `count()` to find how many chain rows are stored in an `ImmunData` object.

Use this method for a quick check of dataset size. The unit counted is one retained chain row. Each retained cell with a paired receptor usually contributes two rows, one for each chain. The same receptor can therefore contribute two rows for every cell carrying it. For bulk data with an abundance column, this method counts table rows rather than the summed sequence abundance.

The function returns a one-row duckplyr table. The original object is not changed.

Usage

```
## S3 method for class 'ImmunData'
count(x, ..., wt = NULL, sort = FALSE, name = NULL)
```

Arguments

x	An ImmunData object.
...	Additional arguments. Accepted for compatibility with dplyr::count() , but currently ignored.
wt	Any value or NULL. Accepted for compatibility with dplyr::count() , but currently ignored.
sort	A logical value. Accepted for compatibility with dplyr::count() , but currently ignored.
name	A character string or NULL. Accepted for compatibility with dplyr::count() , but currently ignored. The result column is always named n.

Details

This method currently provides only the total row count. The grouping, weighting, sorting, and result-name arguments of [dplyr::count\(\)](#) are accepted for method compatibility but are not applied.

The calculation runs on the duckplyr annotation table and can remain in DuckDB. Use [dplyr::pull\(\)](#) or [dplyr::collect\(\)](#) to bring the small result into R.

Value

A one-row duckplyr table with an integer column named n. This value is the number of rows in the chain-level annotation table.

See Also

[dplyr::count\(\)](#), [dplyr::collect\(\)](#), [ImmunData](#)

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)
idata <- get_test_idata()

idata |> count()
# Expected result:
#       n
#   1902

# The result means that the object contains 1,902 retained chain rows.
# It does not mean that it contains 1,902 unique receptors.
```

dimnames.ImmunData	<i>Get Annotation Dimnames from ImmunData</i>
--------------------	---

Description

Returns dimension names for an ImmunData object so that colnames(idata) maps to annotation column names.

Usage

```
## S3 method for class 'ImmunData'  
dimnames(x)
```

Arguments

x	ImmunData object.
---	-------------------

Value

A list with NULL row names and annotation column names.

dimnames<-.ImmunData	<i>Prevent Renaming ImmunData via dimnames</i>
----------------------	--

Description

Disallows replacing dimension names on ImmunData.

Usage

```
## S3 replacement method for class 'ImmunData'  
dimnames(x) <- value
```

Arguments

x	ImmunData object.
value	Not used.

downsample_immundata *Reduce repertoires to a common sampling depth*

Description

Use `downsample_immundata()` to reduce every repertoire to the same number or fraction of observed cells or bulk sequence counts before comparing repertoires. So, it is just a downsampling.

Use this function when different sequencing depths could affect a comparison of repertoire diversity or composition. In single-cell data, the sampling unit is a cell barcode and all selected chains from that cell stay together. In bulk data with abundance values, the sampling unit is one sequence count.

The function returns a new [ImmunData](#) object. The original object is not changed.

Usage

```
downsample_immundata(idata, n, seed = NULL)
```

Arguments

<code>idata</code>	An ImmunData object. For comparisons between repertoires, its repertoires should already be defined with <code>read_repertoires()</code> or <code>agg_repertoires()</code> .
<code>n</code>	A number. Sampling depth. Use a value strictly between 0 and 1 for a fraction, or a whole number greater than or equal to 1 for an absolute number of cells or bulk sequence counts.
<code>seed</code>	A non-negative integer or NULL. Used to reproduce the same random sample. The default is NULL.

Value

A new [ImmunData](#) object containing the sampled chain observations. If the input has repertoires or strata, their summaries are recalculated for the sampled data.

Meaning of `n` for single-cell data

- $0 < n < 1$ keeps $\text{floor}(n * \text{number of cells})$ cells from each repertoire.
- $n \geq 1$ keeps n cells from each repertoire.

Cell barcodes are sampled without replacement. For paired receptors, all retained chains belonging to a selected cell stay together.

Meaning of `n` for bulk data

- $0 < n < 1$ keeps $\text{floor}(n * \text{total abundance})$ sequence counts from each repertoire.
- $n \geq 1$ keeps a total abundance of n from each repertoire.

Counts are sampled without replacement according to their observed abundance. A retained receptor can therefore have a smaller abundance than it had before downsampling. For example, $n = 1000$ makes the total retained abundance equal to 1000 in every repertoire that originally contained at least 1000 counts.

If a requested whole-number n is larger than a repertoire, that repertoire is returned unchanged and the function gives a warning. If a fraction is so small that it selects zero units in any repertoire, the function stops and asks for a larger value.

Repertoire and strata summaries

When repertoires are defined, the function recalculates receptor counts, proportions, repertoire sizes, and the number of repertoires containing each receptor. Existing strata are also rebuilt, and their labels are retained. When repertoires are not defined, the complete dataset is treated as one sampling group and no repertoire summary is added.

Backend and storage

Chain-level selection and reconstruction use the duckplyr annotation table. The small table of sampling units is collected into R for random sampling. The function does not overwrite the stored input object. Use `write_immundata()` to save the returned object.

See Also

`agg_repertoires()`, `filter_immundata()`, `write_immundata()`

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Create two small bulk T-cell repertoires with different total abundances.
bulk_data <- tibble(
  Sample = c("Tumor", "Tumor", "Blood", "Blood"),
  cdr3_aa = c("CASSA", "CASSB", "CASSA", "CASSC"),
  v_call = c("TRBV1", "TRBV2", "TRBV1", "TRBV3"),
  abundance = c(20L, 5L, 4L, 6L)
)
bulk_file <- tempfile(fileext = ".tsv")
readr::write_tsv(bulk_data, bulk_file)

idata <- read_repertoires(
  path = bulk_file,
  schema = c("cdr3_aa", "v_call"),
  count_col = "abundance",
  repertoire_schema = "Sample",
  preprocess = NULL,
  postprocess = NULL,
  rename_columns = NULL,
  output_folder = tempfile("immundata-downsample-")
)

before <- idata$repertoires |>
  select(Sample, n_barcode) |>
  rename(before = n_barcode)
```

```

sampled <- downsample_immundata(idata, n = 5, seed = 42)

before |>
  left_join(
    sampled$repertoires |>
      select(Sample, n_barcodes) |>
      rename(after = n_barcodes),
    by = "Sample"
  ) |>
  arrange(Sample)
# Expected result:
#   Sample before after
#   Blood      10      5
#   Tumor      25      5

# Each returned repertoire has five sequence counts. `idata` still has its
# original repertoire sizes of 10 and 25.

```

filter_immundata	<i>Keep selected rows or receptors in ImmunData</i>
------------------	---

Description

Use `filter()` to keep selected rows in an [ImmunData](#) object. For example, you can keep rows from one response group, rows using a selected V gene, or receptors containing a CDR3 sequence similar to a reference sequence.

The function returns a new [ImmunData](#) object. The original object is not changed.

This function is a direct implementation of [dplyr::filter](#). Alternative function name is `filter_immundata`.

Use `filter_barcodes()` to keep selected cell barcodes and `filter_receptors()` to keep selected receptor identifiers.

Usage

```

filter_immundata(idata, ..., seq_options = NULL, keep_repertoires = TRUE)

## S3 method for class 'ImmunData'
filter(
  .data,
  ...,
  .by = NULL,
  .preserve = FALSE,
  seq_options = NULL,
  keep_repertoires = TRUE
)

```

```
filter_barcodes(idata, barcodes, keep_repertoires = TRUE)
```

```
filter_receptors(idata, receptors, keep_repertoires = TRUE)
```

Arguments

<code>idata, .data</code>	An ImmunData object.
<code>...</code>	One or more conditions used to keep rows. Refer to annotation columns directly by name. Multiple conditions are combined with <code>&</code> . Conditions are applied before sequence matching.
<code>seq_options</code>	Options for matching sequences with reference sequences or patterns. Create these options with make_seq_options() . If <code>NULL</code> , the default, no sequence matching is performed.
<code>keep_repertoires</code>	If <code>TRUE</code> , the default, existing repertoire and strata summaries are recalculated from the filtered data. If <code>FALSE</code> , the returned object does not contain these summaries.
<code>.by, .preserve</code>	Accepted for compatibility with dplyr::filter() , but currently not used for ImmunData objects.
<code>barcodes</code>	A character, integer, or numeric vector of cell barcodes to keep with <code>filter_barcodes()</code> .
<code>receptors</code>	A character, integer, or numeric vector of receptor identifiers to keep with <code>filter_receptors()</code> .

Details

You can filter an [ImmunData](#) object in three ways:

- Supply conditions in `...` to filter using annotation columns. Refer to columns directly by name. For example, `Response == "FR"` keeps rows from the FR response group.
- Supply `seq_options`, created with [make_seq_options\(\)](#), to find receptors containing a sequence that matches one or more reference sequences or patterns.
- Use `filter_barcodes()` or `filter_receptors()` when you already have the identifiers that you want to keep.

Conditions in `...` are applied before sequence matching. Sequence matching then identifies receptors from the remaining rows. When one chain matches, all remaining chains belonging to the same receptor are kept. A chain removed by a condition in `...` is not added back by sequence matching.

Sequence matching methods are:

- `"exact"`: the sequence must be identical to one of the references.
- `"regex"`: the sequence must match a regular-expression pattern. This is an advanced option for matching text patterns.
- `"lev"`: the Levenshtein distance counts the substitutions, insertions, or deletions needed to change one sequence into the other.
- `"hamm"`: the Hamming distance counts different positions between sequences of the same length. Sequences of different lengths do not match.

For "lev" and "hamm", provide `max_dist`. A sequence is accepted when its distance from at least one reference is less than or equal to this value. A distance of 0 means an exact match, and smaller values mean more similar sequences.

By default, existing repertoire summaries are recalculated from the filtered data. Existing strata are also rebuilt, and their labels are retained. Set `keep_repertoires = FALSE` to return an object without repertoire or strata summaries.

Value

A new [ImmunData](#) object containing the selected rows and receptors. If requested, repertoire and strata summaries are recalculated for the selected data.

See Also

[dplyr::filter\(\)](#), [make_seq_options\(\)](#), [mutate_immundata\(\)](#), [agg_repertoires\(\)](#), [ImmunData](#)

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Load data included with immundata
idata <- get_test_idata()

# Keep rows from one response group
fr_response <- idata |>
  filter(Response == "FR")

fr_response |>
  collect() |>
  summarise(
    n_rows = n(),
    n_receptors = n_distinct(imd_receptor_id)
  )
# Expected result:
#   n_rows n_receptors
#     955         871

# Keep receptors containing one reference CDR3 sequence
reference_cdr3 <- "ASFPVLSPYNEQF"

exact_match <- idata |>
  filter(
    seq_options = make_seq_options(
      query_col = "cdr3_aa",
      patterns = reference_cdr3,
      method = "exact"
    )
  )
```

```

exact_match |>
  collect() |>
  select(cdr3_aa, v_call, Response)
# Expected result:
#   cdr3_aa      v_call    Response
#   ASFPVLSPYNEQF TRBV28*01 FR

# Keep receptors within four sequence changes of the reference
similar_sequences <- idata |>
  filter(
    seq_options = make_seq_options(
      query_col = "cdr3_aa",
      patterns = reference_cdr3,
      method = "lev",
      max_dist = 4
    )
  )

similar_sequences |>
  collect() |>
  distinct(cdr3_aa) |>
  arrange(cdr3_aa)
# Expected result:
#   cdr3_aa
#   ASFPVLSPYNEQF
#   ASSPDSPSYNEQF
#   ASSPGLAAYNEQF
#   ASSPTLYNEQF

# Keep two selected cell barcodes
selected_barcodes <- c("S1_1", "S1_2")

selected_cells <- idata |>
  filter_barcodes(selected_barcodes)

selected_cells |>
  collect() |>
  distinct(imd_barcode)
# Expected result:
#   imd_barcode
#   S1_1
#   S1_2

# The same approach can keep selected receptor identifiers
selected_receptors <- idata |>
  collect() |>
  distinct(imd_receptor_id) |>
  slice_head(n = 2) |>
  pull(imd_receptor_id)

selected_receptors_data <- idata |>
  filter_receptors(selected_receptors)

```

from_immunarch

Convert an immunarch Object into an ImmunData Dataset

Description

The `from_immunarch()` function takes an **immunarch** object (as returned by `immunarch::repLoad()`), writes each repertoire to a TSV file with an added internal filename column in a specified folder, and then imports those files into an **ImmunData** object via `read_repertoires()`.

Usage

```
from_immunarch(
  imm,
  output_folder,
  schema = c("CDR3.aa", "V.name"),
  temp_folder = file.path(tempdir(), "temp_folder")
)
```

Arguments

<code>imm</code>	A list returned by <code>immunarch::repLoad()</code> , typically containing: • <code>data</code>: a named list of <code>data.frames</code>, one per repertoire. • <code>meta</code>: (optional) a <code>data.frame</code> of sample metadata.
<code>output_folder</code>	Path to the output directory where the resulting ImmunData Parquet files will be stored. This directory will be created if it does not already exist.
<code>schema</code>	Character vector of column names that together define unique receptors (for example, <code>c("CDR3.aa", "V.name", "J.name")</code>).
<code>temp_folder</code>	Path to a directory where intermediate TSV files will be written. Defaults to <code>file.path(tempdir(), "temp_folder")</code> .

Value

An **ImmunData** object containing all repertoires from the input **immunarch** object, with data saved under `output_folder`.

See Also

[read_repertoires\(\)](#), [read_immundata\(\)](#), [ImmunData](#)

Examples

```
## Not run:
imm <- immunarch::repLoad("/path/to/your/files")
idata <- from_immunarch(imm,
  schema = c("CDR3.aa", "V.name"),
  temp_folder = tempdir(),
  output_folder = "/path/to/immundata_out"
)

## End(Not run)
```

imd_rename_cols	<i>Get input-column presets</i>
-----------------	---------------------------------

Description

Use these helpers to inspect or customize the column renaming and removal presets used by [read_repertoires\(\)](#).

`imd_rename_cols()` returns mappings from standard output names to source names. `imd_drop_cols()` returns technical columns that can usually be removed before receptors are defined. These functions return definitions only; they do not change input files or an [ImmunData](#) object.

Usage

```
imd_rename_cols(format = "default")
```

```
imd_drop_cols(format = "airr")
```

Arguments

format	A character string. The input format preset. For <code>imd_rename_cols()</code> , use "default" or "10x"; the default is "default". For <code>imd_drop_cols()</code> , use "universal", "airr", or "10x"; the default is "airr".
--------	--

Value

`imd_rename_cols()` returns a named character vector in the form `c(new_name = "source_name")`.
`imd_drop_cols()` returns a character vector of source columns to remove.

See Also

[read_repertoires\(\)](#), [make_default_preprocessing\(\)](#), [imd_schema\(\)](#)

Examples

```
imd_rename_cols("10x")
# Includes c(v_call = "v_gene", locus = "chain").

head(imd_drop_cols("10x"), 3)
# Expected result:
#   "full_length" "is_cell" "contig_id"

# Keep the 10x `contig_id` column while dropping the other default columns.
columns_to_drop <- setdiff(imd_drop_cols("10x"), "contig_id")
custom_preprocessing <- list(
  exclude_columns = make_exclude_columns(columns_to_drop),
  filter_nonproductive = make_productive_filter(
    truthy = c("TRUE", "true", "1")
  )
)
```

imd_schema

Get a standard ImmunData column name

Description

Use `imd_schema()` when code needs the standard column name for an `ImmunData` identifier or calculated value, such as the cell barcode, receptor identifier, repertoire identifier, count, or proportion.

Use this helper in reusable analysis code or package extensions instead of writing an internal name such as `"imd_barcode"` directly. It only returns names; it does not inspect or change an [ImmunData](#) object.

Usage

```
imd_schema(key = NULL)
```

Arguments

key	A character string or <code>NULL</code> . One schema key, for example <code>"barcode"</code> , <code>"receptor"</code> , <code>"repertoire"</code> , <code>"count"</code> , or <code>"proportion"</code> . Use <code>NULL</code> , the default, to return all available keys and column names.
-----	--

Value

If key is supplied, one character string containing the standard column name. If `key = NULL`, a named list of all schema keys and column names.

See Also

[make_receptor_schema\(\)](#), [imd_rename_cols\(\)](#), [ImmunData](#)

Examples

```
imd_schema("barcode")
# Expected result: "imd_barcode"

imd_schema("receptor")
# Expected result: "imd_receptor_id"

# Use a returned name for programmatic selection.
barcode_column <- imd_schema("barcode")
get_test_idata() |>
  dplyr::collect() |>
  dplyr::select(dplyr::all_of(barcode_column)) |>
  head(2)
```

ImmunData

ImmunData: A data structure for storing adaptive immune receptor repertoire data

Description

ImmunData stores adaptive immune receptor repertoire (AIRR) data and the rules used to turn observed sequences or cells into data units for analysis. Think AnnData or SeuratObject, but for immune repertoires.

You work with an ImmunData object after importing bulk or single-cell AIRR-seq data. The major idea behind ImmunData is that because sequencing provides only information about sequences and, for single-cell data, cell barcodes, the your responsibility is to determine, which sequences you want to treat as the same receptor, repertoire, or stratum (group of repertoires). You define these analysis units with schemas. A schema is a stored set of column names and chain-selection rules that tells ImmunData how to group observations. Those definitions are kept inside ImmunData to ensure that downstream functions count, filter, and compare the same units consistently. Repertoire and strata schemas can be changed later to re-aggregate repertoires differently, e.g., merge receptors from different clusters into per-patient clusters. Receptor schema is fixed once and for all, so if you want to work with a different receptor definition, e.g., use "CDR3aa + V gene" instead of just "CDR3aa" as a definition for a unique receptor, you will need to create a separate ImmunData object.

ImmunData is immutable, meaning that functions that transform an ImmunData object return a new object, and the original object is not changed. Due to multiple optimisations on the backend, it does not mean that you re-create the whole dataset each time you run a, let's say, a filter. However, it does affect analysis workflow significantly. You can read about it more on the website and in tutorials.

From observed data to analysis units

ImmunData connects observed records to user-defined analysis units:

- A **chain observation** is an observed receptor-chain sequence, such as a TRA, TRB, or IGH sequence. Chain observations form the main table.

- A **barcode** is an observed identifier for a cell in single-cell data. It links chains found in the same cell.
- A **receptor** is a virtual analysis unit that you define. For example, you may define it by CDR3 sequence alone, by CDR3 and V gene, or as a paired TRA-TRB receptor. The receptor schema records which chain features and loci must match for observations to receive the same receptor identifier.
- A **repertoire** is a virtual collection of receptors that you define from annotation columns. For example, one repertoire may contain all receptors from one sample, or from one donor at one time point.
- A **stratum** is a virtual collection of repertoires for a comparison. For example, one stratum may contain all repertoires from one treatment arm.

These definitions do not change the observed sequences. They determine how observations are grouped and counted during analysis. The resulting hierarchy is chain observations and barcodes -> receptors -> repertoires -> strata.

Inspect and transform an object

Print an object for a compact overview. Use `$receptors` for the receptor table, `$repertoires` for one summary row per repertoire, and `$strata` for one row per stratum. Most analysis functions accept the complete `ImmunData` object directly.

Common transformations include:

- `filter_immundata()` to keep selected chains, cells, or receptors;
- `mutate_immundata()` to calculate annotation columns;
- `annotate()` to add external biological information;
- `agg_repertoires()` to define repertoires; and
- `agg_strata()` to group repertoires into strata.

Create an object

Create an `ImmunData` object with `read_repertoires()`, or reopen a saved object with `read_immundata()`. Do not call the `$new()` constructor in analysis code. Direct construction is reserved for package developers.

Lazy data and storage

The chain-level table uses `duckplyr` and can remain on disk. Filtering, mutation, and aggregation stay lazy when possible, so large datasets do not need to be loaded fully into R memory. Downstream analysis functions in the `immunarch` package are designed to accept lazy `ImmunData` objects. Pass the object directly; you usually do not need to call `dplyr::collect()`. Collect data only when another function explicitly requires an in-memory data frame or when you want to inspect a small table in R.

Objects created by `read_repertoires()` are backed by files in their output folder. Keep that folder while you use the object. Use `write_immundata()` to save a transformed object and `read_immundata()` to reopen it.

Public fields

`schema_receptor` A named list defining the virtual receptor unit. The features element names the chain columns used to group observations, such as CDR3 sequence and V gene. The chains element selects one chain or a paired set of chains.

`schema_repertoire` A character vector naming annotation columns whose unique combinations define one repertoire, such as `sample_id` or `c("donor_id", "timepoint")`. It is NULL when repertoires have not been defined.

`schema_strata` A character vector naming repertoire-level columns whose unique combinations define one stratum, such as `treatment`. It is NULL when strata have not been defined.

Active bindings

`receptors` A derived duckplyr table of distinct receptors. For a paired receptor, the selected chain features are shown side by side.

`annotations` The lazy duckplyr table of retained chain observations and their biological annotations. For most tasks, pass the complete ImmunData object to a transformation function or use `collect(idata)` to inspect this table in memory.

`repertoires` A small table with one row per repertoire, the columns that define it, and summary statistics such as `n_barcode`s and `n_receptors`. It is NULL when repertoires have not been defined.

`strata` A small table with one row per stratum, its label, and the repertoire-level columns that define it. It is NULL when strata have not been defined.

`provenance` Read-only named list describing the snapshot origin and storage context carried by this object. Retrieve the complete list with `idata$provenance`, or one field with, for example, `idata$provenance$current_path`. The fields are:

- `home_path`: project home used for managed snapshots and artifacts. The original ingestion snapshot is stored directly in this folder; it is NULL for an object with no persisted home.
- `current_path`: exact folder of the most recently loaded or written snapshot. Transformations preserve this source path until the transformed object is written as another snapshot; it is NULL for an object that has never been loaded from or written to disk.
- `snapshot_root`: derived managed-snapshot root, `home_path/snapshots`, or NULL when `home_path` is NULL.
- `artifacts_root`: derived project-level root for optional external tool outputs, `home_path/artifacts`, or NULL when `home_path` is NULL.
- `artifacts_path`: derived namespace for artifacts associated with the most recently loaded or written snapshot. It is `artifacts_root/root` for the original ingestion, `artifacts_root/<tag>/vNNN` for a managed snapshot, and `artifacts_root/by-id/<snapshot_id>` for a detached explicit snapshot. External tools can append `<tool>/<run>` and create that directory; artifact contents are not part of ImmunData. Write a transformed object as a new snapshot before storing artifacts that should be associated with the transformed data.
- `snapshot_id`: unique identifier generated when the snapshot is written; NULL for an in-memory object that has never been written.
- `lineage`: ordered list of ingestion and snapshot events leading to the current snapshot.

The accessor is read-only; assigning to `idata$provenance` is an error.

Methods

Public methods:

- [ImmunData\\$new\(\)](#)
- [ImmunData\\$clone\(\)](#)

[ImmunData\\$new\(\)](#): Low-level constructor for package developers. Analysis code must create an [ImmunData](#) object with [read_repertoires\(\)](#) or reopen one with [read_immundata\(\)](#).

Usage:

```
ImmunData$new(
  schema,
  annotations,
  repertoires = NULL,
  provenance = NULL,
  strata = NULL
)
```

Arguments:

schema A character vector or named list. A character vector names the features used to define a chain-agnostic receptor. A named list is created by [make_receptor_schema\(\)](#) and can also select receptor chains.

annotations A duckplyr table. It contains retained chain observations, receptor identifiers, and biological annotations.

repertoires A data frame or NULL. It contains one row per repertoire and its summary statistics and is usually created by [agg_repertoires\(\)](#).

provenance A list or NULL. It contains internal storage and snapshot history.

strata A data frame or NULL. It contains one row per stratum, its label, and the repertoire-level columns that define it.

[ImmunData\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
ImmunData$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

[read_repertoires\(\)](#), [read_immundata\(\)](#), [write_immundata\(\)](#), [agg_repertoires\(\)](#), [agg_strata\(\)](#), [filter_immundata\(\)](#), [mutate_immundata\(\)](#), [annotate\(\)](#)

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Load the small dataset included with immundata, then define one repertoire
```

```

# for each treatment-response group.
idata <- get_test_idata() |>
  agg_repertoires(schema = "Response")

idata$repertoires |>
  select(Response, n_barcodes, n_receptors) |>
  arrange(Response)
# Expected result:
#   Response n_barcodes n_receptors
#   FR           955         871
#   PR           947         867

# Under the current receptor definition, the full-response (FR) repertoire
# contains 955 chain observations grouped into 871 receptor units.

# Keep only the full-response repertoire. filter() returns a new object.
fr_only <- idata |>
  filter(Response == "FR")

tibble(
  original_repertoires = nrow(idata$repertoires),
  filtered_repertoires = nrow(fr_only$repertoires)
)
# Expected result:
#   original_repertoires filtered_repertoires
#               2             1
# The original object still contains both repertoires.

```

make_default_preprocessing

Process chain rows while reading repertoire files

Description

Use these functions to preprocess or postprocess rows of the input data before returning the final ImmunData object to the session. A couple of example use cases: keep productive receptor chains, remove technical columns, or make cell barcodes unique while importing repertoire files with [read_repertoires\(\)](#).

The defaults provide steps for common AIRR or 10x inputs. Use an individual step when your files need only one operation or when you are building a custom preprocess or postprocess list.

Preprocessing changes chain rows before receptors are defined. Barcode prefixing changes the cell identifier after receptor and manifest information are combined. The input files and input table are not changed: every step returns a new duckplyr table.

Usage

```
make_default_preprocessing(format = c("default", "airr", "10x"))
```

```

make_default_postprocessing()

make_exclude_columns(cols = imd_drop_cols("airr"))

make_productive_filter(col_name = c("productive"), truthy = TRUE)

make_barcode_prefix(prefix_col = "Prefix")

```

Arguments

format	A character string. One input format: "default", "airr", or "10x". The default is "default". This choice controls which technical columns are removed. It does not rename columns.
cols	A character vector. Columns to remove. The default is imd_drop_cols("airr"). Use character() to create a step that removes no columns.
col_name	A character string. Column containing the productive-chain indicator. The default is "productive".
truthy	A vector. Values that mean the chain is productive. Values are compared as text. The default is TRUE; use a character vector when the source uses several representations, for example c("TRUE", "true", "1").
prefix_col	A character vector. One or more candidate columns containing the text to place before each cell barcode. The first candidate present in the data is used. The default is "Prefix".

Value

make_default_preprocessing() and make_default_postprocessing() return named lists of processing functions. The other functions return one processing function. Each processing function accepts a duckplyr table as its first argument, accepts unused arguments through ..., and returns a new duckplyr table.

Choose processing steps

- make_default_preprocessing() returns two steps. The first removes common technical columns. The second keeps rows whose productive value indicates a productive chain. If the productive column is absent, the filtering step gives a warning and keeps all rows.
- make_default_postprocessing() returns one step that adds a sample-specific prefix to cell barcodes. If the prefix column is absent, the step gives a warning and leaves barcodes unchanged.
- make_exclude_columns() creates one step that removes the columns in cols. Column names that are not present are ignored.
- make_productive_filter() creates one step that keeps rows whose value in col_name matches any value in truthy.
- make_barcode_prefix() creates one step that joins a prefix, such as "Tumor_", to the start of each imd_barcode value.

read_repertoires() applies functions in list order. You can therefore add, remove, or reorder steps in a custom list.

Input formats

For `make_default_preprocessing()`, `format = "default"` removes the union of the standard AIRR and 10x technical columns. Use `format = "airr"` or `format = "10x"` to remove only the columns expected for that format. All three defaults recognize common text representations of a productive value, including "TRUE", "true", "yes", and "1".

See Also

`read_repertoires()`, `imd_drop_cols()`, `imd_rename_cols()`

Examples

```
library(immunodata)
library(dplyr)

# Three 10x chain rows from two samples. One chain is non-productive.
chains <- duckplyr::duckdb_tibble(
  imd_barcode = c("AAAC-1", "AAAG-1", "AATT-1"),
  cdr3_aa = c("CASSA", "CASSB", "CASSC"),
  productive = c("TRUE", "FALSE", "TRUE"),
  full_length = c(TRUE, TRUE, TRUE),
  Prefix = c("Tumor_", "Tumor_", "Blood_")
)

# read_repertoires() performs these calls for you. They are shown here to
# make the effect of each list clear.
prepared <- Reduce(
  function(data, step) step(data),
  make_default_preprocessing("10x"),
  init = chains
)

prepared <- Reduce(
  function(data, step) step(data),
  make_default_postprocessing(),
  init = prepared
)

prepared |>
  collect() |>
  select(imd_barcode, cdr3_aa, productive)
# Expected result:
#   imd_barcode  cdr3_aa productive
#   Tumor_AAAC-1 CASSA    TRUE
#   Blood_AATT-1 CASSC    TRUE

# The non-productive chain was removed, `full_length` was dropped, and the
# sample prefixes made the retained cell barcodes unique.
```

make_receptor_schema *Define which chain observations form the same receptor*

Description

Use `make_receptor_schema()` to define a biological receptor from sequence features and one or two receptor chains.

Use this function when reading single-cell data with one selected chain, when pairing chains such as TRA-TRB, or when accepting alternative light chains such as IGK or IGL. The unit being defined is the receptor. Creating a schema does not change any data or an existing [ImmunData](#) object.

Usage

```
make_receptor_schema(features, chains = NULL)
```

```
assert_receptor_schema(schema)
```

```
test_receptor_schema(schema)
```

Arguments

features	A non-empty character vector. Column names containing the chain fields that must match, such as <code>c("junction_aa", "v_call", "j_call")</code> . Use names as they appear after any input-column renaming.
chains	A character vector of length one or two, or NULL. Use one value, such as "TRB", to keep one chain; two values, such as <code>c("TRA", "TRB")</code> , to define a strict pair; or the "IGK IGL" syntax in the second value to accept either alternative. The default is NULL, which does not select loci.
schema	A non-empty character vector or receptor-schema list. An object to check. A schema created by <code>make_receptor_schema()</code> is accepted. A character vector supplies feature names for a chain-agnostic schema.

Value

`make_receptor_schema()` returns a list with character elements `features` and `chains`; `chains` is NULL when loci are not selected. `assert_receptor_schema()` returns TRUE for accepted input and otherwise stops with an error. `test_receptor_schema()` returns one logical value.

When observations are the same receptor

`features` names the fields that define chain identity. Common choices are the CDR3 amino acid sequence, V gene, and J gene. Two observations represent the same receptor only when the relevant chain loci and every selected feature match.

- With one chain, such as `chains = "TRB"`, only that locus is used. Two TRB observations are the same receptor when all their selected feature values match.

- With a strict pair, such as `chains = c("TRA", "TRB")`, chains are first paired within each cell barcode. Receptors from two cells are the same only when every selected TRA feature and every selected TRB feature match.
- With an alternative second chain, such as `chains = c("IGH", "IGK|IGL")`, each receptor must contain IGH and exactly one of IGK or IGL. Cells containing both IGK and IGL are excluded. The light-chain locus and all selected heavy- and light-chain features must match for two observations to be the same receptor.

A barcode determines which chains belong to the same cell; it does not by itself define receptor identity across cells. During single-cell import, `read_repertoires()` uses `umi_col` to choose one chain when a cell contains several observations from the same locus.

Use `chains = NULL` for chain-agnostic bulk or pre-filtered data. In that case, only the values in features define receptor identity.

Validate a schema

`assert_receptor_schema()` stops with an error if schema is not accepted. Use it inside another function when invalid input must stop the calculation. `test_receptor_schema()` returns one TRUE or FALSE value and is useful in conditional code.

Backend and storage

A receptor schema is a small R list containing features and chains. It stores no sequence data. `read_repertoires()` and `agg_receptors()` apply the schema to chain observations using `duckplyr`.

See Also

`read_repertoires()`, `agg_receptors()`, `imd_schema()`

Examples

```
# Single-chain TCR: compare TRB observations by CDR3, V gene, and J gene.
trb_schema <- make_receptor_schema(
  features = c("junction_aa", "v_call", "j_call"),
  chains = "TRB"
)
trb_schema
# Expected result:
# $features: "junction_aa" "v_call" "j_call"
# $chains:   "TRB"

# Paired alpha-beta TCR: all selected fields must match on both TRA and TRB.
ab_tcr_schema <- make_receptor_schema(
  features = c("junction_aa", "v_call", "j_call"),
  chains = c("TRA", "TRB")
)
ab_tcr_schema
# The result defines one receptor as a matched TRA-TRB pair from one cell.

# BCR: require IGH and accept either an IGK or IGL light chain.
```

```

bcr_schema <- make_receptor_schema(
  features = c("junction_aa", "v_call", "j_call"),
  chains = c("IGH", "IGK|IGL")
)
bcr_schema
# The result accepts IGH-IGK and IGH-IGL receptors, while keeping the two
# light-chain loci biologically distinct.

test_receptor_schema(bcr_schema)
# Expected result: TRUE

```

make_seq_options

Create options for comparing receptor sequences

Description

Create sequence comparison options for the `seq_options` argument of `filter_immundata()` or `mutate_immundata()`. Use these options to compare a sequence column with one or more reference sequences or patterns.

Usage

```

make_seq_options(
  query_col,
  patterns,
  method = c("exact", "lev", "hamm", "regex"),
  max_dist = NA,
  name_type = c("index", "pattern")
)

```

Arguments

<code>query_col</code>	Name of the sequence column to compare, such as "cdr3_aa".
<code>patterns</code>	One or more reference sequences or regular-expression patterns.
<code>method</code>	Comparison method: "exact", "regex", "lev" (Levenshtein distance), or "hamm" (Hamming distance). The default is "exact".
<code>max_dist</code>	Maximum distance accepted by <code>filter_immundata()</code> when <code>method = "lev"</code> or <code>method = "hamm"</code> . A value is required when filtering with either distance method. This argument has no effect on <code>mutate_immundata()</code> , which reports every calculated distance.
<code>name_type</code>	How result columns created by <code>mutate_immundata()</code> are named. "index", the default, creates short numbered names. "pattern" includes the reference pattern in each name. This argument does not change which receptors are kept by <code>filter_immundata()</code> .

Value

A named list for the `seq_options` argument of `filter_immundata()` or `mutate_immundata()`.

See Also

`filter_immundata()`, `mutate_immundata()`, `annotate_receptors()`

<code>mutate_immundata</code>	<i>Add or change annotation columns in ImmunData</i>
-------------------------------	--

Description

Use `mutate()` to add information to each row of an `ImmunData` object. For example, you can calculate CDR3 length, mark sequences of interest, or compare receptor sequences with reference sequences.

The function returns a new `ImmunData` object. The original object is not changed.

This function is a direct implementation of `dplyr::mutate`. Alternative function name is `mutate_immundata`.

Usage

```
mutate_immundata(idata, ..., .by = NULL, seq_options = NULL)
```

```
## S3 method for class 'ImmunData'
```

```
mutate(.data, ..., .by = NULL, seq_options = NULL)
```

Arguments

<code>idata, .data</code>	An <code>ImmunData</code> object.
<code>...</code>	One or more named calculations in the form <code>new_column = calculation</code> . Refer to existing columns directly by name. You can add new annotation columns or change columns that are not protected.
<code>.by</code>	Optional columns used to form temporary groups for this operation. For example, <code>.by = Response</code> calculates separately for each response, and <code>.by = c(Response, imd_group_id)</code> uses each response and receptor-cluster combination. The grouping applies only to this <code>mutate()</code> call.
<code>seq_options</code>	Options for comparing sequences with reference sequences or patterns. Create these options with <code>make_seq_options()</code> . If <code>NULL</code> , the default, no sequence comparisons are performed.

Details

You can use `mutate()` in three ways:

- Supply named calculations in `...` to create annotation columns from existing data. For example, `cmv_specific = cdr3_aa %in% cmv_cdr3s` adds a column containing `TRUE` or `FALSE` for each row.

- Supply `.by` to perform calculations separately for temporary groups. The number of rows does not change. A group statistic is repeated for all rows in that group.
- Supply `seq_options`, created with `make_seq_options()`, to compare a sequence column with one or more reference sequences or patterns. One result column is added for each reference.

Named calculations in `...` are performed before sequence comparisons.

Most grouped calculations are translated directly to DuckDB. Some group statistics, such as `n_distinct()`, are not available as DuckDB window calculations when a large dataset must stay on disk. In that case, `mutate()` automatically calculates one summary row per group and joins the values back to the annotation rows. This remains lazy and does not load the full dataset into R memory.

The automatic fallback works when every calculation in the call produces one value per group. If a call combines a row-level calculation with a group statistic that needs the fallback, use two `mutate()` calls. Also use a second call when a later calculation refers to a group statistic created by the fallback. See the examples below.

Columns used to identify receptors or repertoires, and identifiers managed by `ImmunData`, are protected. This prevents accidental changes that would make the object inconsistent. You can add new columns and change other annotation columns.

Sequence comparison methods are:

- `"exact"`: TRUE when the sequence is identical to the reference.
- `"regex"`: TRUE when the sequence matches a regular-expression pattern. This is an advanced option for matching text patterns.
- `"lev"`: the number of substitutions, insertions, or deletions needed to change one sequence into the other.
- `"hamm"`: the number of different positions between sequences of the same length. Sequences with different lengths receive NA.

For the distance methods, 0 means an exact match and smaller values mean more similar sequences. With `name_type = "index"`, the result columns have short names such as `imd_sim_exact_1` or `imd_sim_lev_1`. With `name_type = "pattern"`, each column name includes its reference pattern.

`max_dist` is used by `filter_immundata()` but has no effect here because `mutate()` reports every calculated distance.

Existing repertoire and strata summaries are carried to the new object without modification.

Value

A new `ImmunData` object containing the added or changed annotation columns. Existing repertoire and strata summaries are preserved.

See Also

`dplyr::mutate()`, `make_seq_options()`, `filter_immundata()`, `annotate_receptors()`, `agg_repertoires()`, `ImmunData`

Examples

```

library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Load data included with immundata
idata <- get_test_idata()

# Add the length of each CDR3 amino acid sequence
idata_with_length <- idata |>
  mutate(cdr3_length = dd$length(cdr3_aa))

idata_with_length |>
  collect() |>
  select(cdr3_aa, cdr3_length) |>
  slice_head(n = 3)
# Expected result:
#   cdr3_aa      cdr3_length
# ASFPVLSPYNEQF      13
# ASRAGAGTGELF       12
# ASSPGQGLDTQY       12

# Compare CDR3 sequences with one reference sequence
reference_cdr3 <- "ASFPVLSPYNEQF"

idata_with_matches <- idata |>
  mutate(
    seq_options = make_seq_options(
      query_col = "cdr3_aa",
      patterns = reference_cdr3,
      method = "exact"
    )
  )

idata_with_matches |>
  collect() |>
  count(imd_sim_exact_1)
# Expected result:
#   imd_sim_exact_1    n
# FALSE           1901
# TRUE              1

# Calculate Levenshtein distance from the reference sequence
idata_with_distance <- idata |>
  mutate(
    seq_options = make_seq_options(
      query_col = "cdr3_aa",
      patterns = reference_cdr3,
      method = "lev"
    )
  )

```

```

idata_with_distance |>
  collect() |>
  select(cdr3_aa, imd_sim_lev_1) |>
  arrange(imd_sim_lev_1, cdr3_aa) |>
  slice_head(n = 3)
# Expected result:
#   cdr3_aa      imd_sim_lev_1
#   ASFPVLSPYNEQF           0
#   ASSPDSPSYNEQF           4
#   ASSPGLAAYNEQF           4

# Mark selected sequences
cmv_cdr3s <- c(
  "ASFPVLSPYNEQF",
  "ASRAGAGTGELF"
)

marked_sequences <- idata |>
  mutate(
    cmv_specific = cdr3_aa %in% cmv_cdr3s
  )

marked_sequences |>
  collect() |>
  count(cmv_specific)
# Expected result:
#   cmv_specific    n
#   FALSE         1900
#   TRUE           2

# Mark selected receptor identities
cmv_hits <- tibble(
  imd_receptor_id = c(1L, 105L),
  cmv_specific = TRUE
)

marked_receptors <- idata |>
  annotate_receptors(cmv_hits) |>
  mutate(
    cmv_specific = coalesce(cmv_specific, FALSE)
  )

marked_receptors |>
  collect() |>
  count(cmv_specific)
# Expected result:
#   cmv_specific    n
#   FALSE         1898
#   TRUE           4

# Add response-level statistics to every annotation row
# `.by` means: calculate separately for each response.

```

```

response_stats <- idata |>
  mutate(
    response_n_rows = n(),
    response_n_receptors = n_distinct(imd_receptor_id),
    .by = Response
  )

response_stats |>
  collect() |>
  distinct(Response, response_n_rows, response_n_receptors) |>
  arrange(Response)
# Expected result:
#   Response response_n_rows response_n_receptors
#   FR           955           871
#   PR           947           867

# A grouped calculation can also produce a different value for every row.
response_centered <- idata |>
  mutate(
    centered_counts = counts - mean(counts, na.rm = TRUE),
    .by = Response
  )

# Do not combine that row-level calculation with a statistic that needs the
# automatic summary fallback in the same call:
# idata |>
#   mutate(
#     centered_counts = counts - mean(counts, na.rm = TRUE),
#     response_n_receptors = n_distinct(imd_receptor_id),
#     .by = Response
#   )

# Use two mutate calls instead. The work remains lazy in DuckDB.
response_details <- idata |>
  mutate(
    centered_counts = counts - mean(counts, na.rm = TRUE),
    .by = Response
  ) |>
  mutate(
    response_n_receptors = n_distinct(imd_receptor_id),
    .by = Response
  )

# Also use a second call when a new calculation uses a statistic created by
# the fallback.
response_details <- idata |>
  mutate(
    response_n_receptors = n_distinct(imd_receptor_id),
    .by = Response
  ) |>
  mutate(
    twice_response_n_receptors = response_n_receptors * 2
  )

```

names<- .ImmunData	<i>Prevent Renaming ImmunData via names</i>
--------------------	---

Description

Disallows replacing names on ImmunData.

Usage

```
## S3 replacement method for class 'ImmunData'
names(x) <- value
```

Arguments

x	ImmunData object.
value	Not used.

print.ImmunData	<i>Display the contents and biological definitions of ImmunData</i>
-----------------	---

Description

Use print() to inspect the receptor table, chain annotations, and biological schemas stored in an [ImmunData](#) object.

Use this method for a quick overview after reading, filtering, or aggregating repertoire data. It displays the units available in the object: receptors, chain rows, repertoires, and strata. It also shows the feature and chain definitions used to construct receptors.

Printing is read-only. It does not collect the complete dataset into R and does not change the original object. The object is returned invisibly so it can still be assigned or used in a pipeline.

Usage

```
## S3 method for class 'ImmunData'
print(x, ...)
```

Arguments

x	An ImmunData object to display.
...	Additional arguments. Currently not used.

Details

A section is shown only when that information is available. An object without repertoire definitions, for example, has no repertoire schema or repertoire summary section. Duckplyr prints a preview of large tables rather than every row.

Value

x, invisibly. The displayed output is a human-readable overview; no data are modified.

See Also

[ImmunData](#), [dplyr::collect\(\)](#), [dplyr::count\(\)](#)

Examples

```
library(immundata)

options(immundata.verbose = FALSE)
idata <- get_test_idata()

print(idata)
# Expected output contains these sections:
# ImmunData
# Receptors
# Annotations
# Receptor schema
# Repertoire schema
# List of repertoires

# `Receptors` previews distinct biological receptor definitions.
# `Annotations` previews the retained chain rows and sample information.
# The schema sections explain how receptors and repertoires were defined.
```

read_immundata

Load an ImmunData object from disk

Description

Continue an analysis later by reopening an [ImmunData](#) dataset saved on disk. Use `read_immundata()` after restarting R, in another script, or when another person gives you a dataset created by [write_immundata\(\)](#) or [read_repertoires\(\)](#). It is that simple, just don't forget to save the ImmunData object first!

The unit restored retains all information: chain rows, cell and receptor identifiers, repertoire and stratum definitions, and provenance. The function does not change these biological units or the saved files. It returns a new [ImmunData](#) object.

Usage

```
read_immundata(
  path,
  tag = NULL,
  version = NULL,
  prudence = "stingy",
  verbose = getOption("immundata.verbose", TRUE)
)
```

Arguments

path	A character string. Path to a saved dataset directory. The directory must contain <code>annotations.parquet</code> and <code>metadata.json</code> . When tag is supplied, use the project home directory that contains the snapshots directory. Read more about snapshots on the website.
tag	A character string or NULL. Snapshot tag to read from <code>path/snapshots/<tag>/vNNN</code> . If NULL, the default, path itself is read.
version	A non-negative integer or NULL. Snapshot version within tag. For example, 1 reads <code>v001</code> . If NULL, the default, the latest available version for the tag is read. version can only be used with tag.
prudence	A character string. Memory protection used while reading the Parquet data. This controls whether duckplyr may convert an intermediate result from DuckDB-managed memory to an R data frame: "stingy", the default here, never permits conversion; "thrifty" permits up to 1 million table cells (rows multiplied by columns); and "lavish" permits conversion regardless of size. Here, "table cells" does not mean biological cells. Passed to <code>duckplyr::read_parquet_duckdb()</code> .
verbose	A logical value. Whether to print progress and summary messages. Defaults to <code>getOption("immundata.verbose", TRUE)</code> .

Details

Read either a dataset directory directly or a versioned snapshot within its project home.

Value

A new, disk-backed `ImmunData` object representing the selected saved state. Its provenance records the directory that was read.

Choose the saved state

To reopen a dataset saved directly in a folder, supply that folder as path and leave tag and version as NULL.

To reopen a managed snapshot, supply the project home as path and its tag. By default, the latest version for that tag is read. Supply version when you need an exact earlier state.

Backend and serialized data

annotations.parquet stores the retained chain-level annotation table. It is reopened as a lazy duckplyr table, so the complete table does not need to be loaded into R memory. metadata.json stores the format and package versions, receptor, repertoire, and stratum schemas, the repertoire table, the snapshot identifier, lineage events, and provenance paths.

Receptor and stratum views are reconstructed from this serialized state; they are not stored as separate files. Please also mind, that the saved files is an ImmunData-specific serialization, not an RDS file.

See Also

[write_immundata\(\)](#) for saving an analysis, [read_repertoires\(\)](#) for importing AIRR-seq files, [ImmunData](#)

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Create a project home and save a filtered biological state as a snapshot
idata <- get_test_idata()
project_dir <- tempfile("immundata-project-")

project_idata <- write_immundata(
  idata,
  output_folder = project_dir,
  rehome = TRUE
)

fr_response <- project_idata |>
  filter(Response == "FR")

write_immundata(fr_response, tag = "fr-response")

# Read the exact first version of this snapshot
continued_fr <- read_immundata(
  project_dir,
  tag = "fr-response",
  version = 1
)

continued_fr |>
  collect() |>
  summarise(
    n_chains = n(),
    n_receptors = n_distinct(imd_receptor_id)
  )

# Expected result: the snapshot contains the 955 chain rows and 871
# receptors from the FR response group.
```

```
#   n_chains n_receptors
#       955       871

list.files(file.path(project_dir, "snapshots", "fr-response"))
# Expected result: "v001"

unlink(project_dir, recursive = TRUE)
```

read_manifest

Load and Validate a Manifest for Immune Repertoire Files

Description

This function loads a manifest from either a file path or a data frame, validates the presence of a column with repertoire file paths, and converts all file paths to absolute paths. It is used to support flexible pipelines for loading bulk or single-cell immune repertoire data across samples.

If the input is a file path, the function reads it with `readr::read_delim`. If the input is a data frame, it checks whether file paths are absolute; relative paths are only allowed when the manifest is loaded from a file.

It warns the user if many of the files listed in the manifest are missing, and stops execution if none of the files exist.

The column with file paths is normalized into the internal filename schema.

Usage

```
read_manifest(
  manifest,
  file_col = "file",
  delim = NULL,
  ...,
  verbose = getOption("immundata.verbose", TRUE)
)
```

Arguments

manifest	A manifest table. Can be either: • a data frame with per-file annotations, • or a path to a CSV/TSV/TXT manifest file.
file_col	A string specifying the name of the column in the manifest that contains paths to repertoire files. Defaults to "file".
delim	Delimiter used to read the manifest file. If NULL, it is inferred from the extension: comma for .csv, tab for .tsv and .txt.
...	Additional arguments passed to <code>readr::read_delim()</code> when reading a manifest from a file.
verbose	Logical(1). Whether to print informative messages. Defaults to <code>getOption("immundata.verbose", TRUE)</code> .

Value

A validated and updated manifest data frame with absolute file paths and an additional internal column named `imd_filename`.

<code>read_repertoires</code>	<i>Read immune repertoire files into ImmunData</i>
-------------------------------	--

Description

`read_repertoires()` is the main function for importing AIRR-seq data. It reads one or more repertoire files, defines biological receptors, adds sample information from an optional manifest, and returns an [ImmunData](#) object.

The function saves the processed data in `output_folder`. This lets you work with large datasets without loading everything into memory and reopen the result later with [read_immundata\(\)](#).

Usage

```
read_repertoires(
  path,
  schema,
  manifest = NULL,
  barcode_col = NULL,
  count_col = NULL,
  locus_col = NULL,
  umi_col = NULL,
  preprocess = make_default_preprocessing(),
  postprocess = make_default_postprocessing(),
  rename_columns = imd_rename_cols("10x"),
  enforce_schema = TRUE,
  manifest_file_col = "file",
  output_folder = NULL,
  repertoire_schema = "<auto>",
  verbose = getOption("immundata.verbose", TRUE),
  prematerialize = TRUE,
  prematerialize_folder = NULL
)
```

Arguments

<code>path</code>	One or more repertoire file paths, or a glob pattern such as <code>"/path/to/data/*.tsv.gz"</code> . Supported formats are Parquet, CSV, TSV, and gzipped CSV or TSV. All input files must have the same file type. Use <code>"<manifest>"</code> to take file paths from manifest instead. In that case, manifest is required.
<code>schema</code>	Definition of receptor identity. Supply either:

- A character vector naming the features that must match, such as `c("v_call", "j_call", "junction_aa")`.
- An object created by `make_receptor_schema()` to select one locus or pair two loci from the same cell.

Use column names as they appear *after* `rename_columns` is applied. For example, if the input columns are `CDR3_aa` and `V.name`, use `rename_columns = c(cdr3_aa = "CDR3_aa", v_call = "V.name")` together with `schema = c("cdr3_aa", "v_call")`.

manifest	An optional data frame with one row per repertoire file and columns containing sample, donor, tissue, treatment, or other information. Use <code>read_manifest()</code> to read and validate a manifest file. Manifest paths must be unique. When <code>path = "<manifest>"</code> , the column named by <code>manifest_file_col</code> supplies the repertoire file paths. The default is <code>NULL</code> .
barcode_col	Name of the column containing cell barcodes. Supplying it selects single-cell processing, requires <code>umi_col</code> , and prevents use of <code>count_col</code> . Use the column name after renaming. The default is <code>NULL</code> .
count_col	Name of the column containing non-negative abundance values for bulk repertoire data. It cannot be used with <code>barcode_col</code> . Use the column name after renaming. The default is <code>NULL</code> .
locus_col	Name of the column containing receptor loci such as <code>"TRA"</code> , <code>"TRB"</code> , <code>"IGH"</code> , <code>"IGK"</code> , or <code>"IGL"</code> . It is required when <code>schema</code> selects or pairs chains. Use the column name after renaming. The default is <code>NULL</code> .
umi_col	Name of the column containing per-chain UMI or read counts. It is required whenever <code>barcode_col</code> is supplied and is used to choose one chain when a cell contains several chains from the same locus. Use the column name after renaming. The default is <code>NULL</code> .
preprocess	A named list of functions applied in order before receptors are defined. Each function must accept a duckplyr table as its first argument and return a duckplyr table. By default, <code>make_default_preprocessing()</code> removes selected technical columns and keeps productive sequences when a productive column is available. Use <code>NULL</code> or <code>list()</code> to disable preprocessing.
postprocess	A named list of functions applied in order after receptors are defined and manifest information is added. Each function must accept and return a duckplyr table. By default, <code>make_default_postprocessing()</code> prefixes cell barcodes when the manifest contains a <code>Prefix</code> column. Use <code>NULL</code> or <code>list()</code> to disable postprocessing.
rename_columns	An optional named character vector in the form <code>c(new_name = "old_name")</code> . Renaming occurs before preprocessing and receptor definition. The default, <code>imd_rename_cols("10x")</code> , standardizes common 10x names such as <code>v_gene</code> to <code>v_call</code> and <code>chain</code> to <code>locus</code> when those source columns are present. Use <code>NULL</code> to preserve all input names.
enforce_schema	Whether multiple input files must have the same columns and column types. The default is <code>TRUE</code> . If <code>FALSE</code> , columns are combined by name and missing values are added where necessary. This is slower and can require more memory.

manifest_file_col	Name of the manifest column containing repertoire file paths when path = "<manifest>". The default is "file". Use the same name passed as file_col to read_manifest() when it is not "file".
output_folder	Directory in which to write annotations.parquet and metadata.json. These files are the persistent backing storage for the returned object. If NULL, a folder beginning with immundata- is created beside the first input file. Supplying an existing folder replaces its annotations.parquet and metadata.json. The default is NULL.
repertoire_schema	Definition of repertoires. Supply one of: • A character vector naming columns that define one repertoire, such as c("donor", "timepoint"). • "<auto>", the default. This creates one repertoire per input file, or one per manifest row when path = "<manifest>". • "<manifest>", which uses all manifest columns when a manifest is available, or the input filename otherwise. • NULL to leave repertoires undefined.
verbose	Whether to print progress and summary messages. Defaults to <code>getOption("immundata.verbose", TRUE)</code> .
prematerialize	Whether CSV, TSV, and compressed text inputs should be combined into a temporary Parquet file before receptor processing. This avoids repeatedly scanning text input during downstream lazy queries. Existing Parquet input is used directly. The default is TRUE.
prematerialize_folder	Directory in which to create the temporary combined Parquet file. If NULL, the default, tempdir() is used. The directory is created when necessary. The temporary file is deleted when <code>read_repertoires()</code> exits, including after an error.

Details

The required arguments depend on how receptor observations are represented in the input files.

Value

A disk-backed [ImmunData](#) object containing the retained chain rows, receptor definitions, manifest annotations, and ingestion provenance. If repertoire_schema is not NULL, it also contains repertoire definitions and summary statistics calculated by [agg_repertoires\(\)](#).

Choose arguments for your data

- **Uncounted repertoire table:** Supply schema. Leave barcode_col and count_col as NULL. Each retained row represents one observed chain.
- **Bulk repertoire with abundance:** Supply schema and count_col. The abundance values are preserved for later repertoire statistics.
- **Single-cell, one selected chain:** Use [make_receptor_schema\(\)](#) with one chain and supply barcode_col, locus_col, and umi_col.

- **Single-cell, paired chains:** Use `make_receptor_schema()` with two chains and supply `barcode_col`, `locus_col`, and `umi_col`. Only cells containing both requested chains are retained.
- **Single-cell, relaxed paired chains:** Use a schema such as `chains = c("IGH", "IGL|IGK")` with `barcode_col`, `locus_col`, and `umi_col`. This accepts either an IGH-IGL or IGH-IGK receptor.

In single-cell data, the chain with the highest `umi_col` value is retained when a cell contains several chains from the same locus.

What happens by default

Unless you override the relevant arguments, `read_repertoires()`:

- temporarily combines text input into Parquet before processing;
- standardizes common 10x column names;
- removes selected technical columns;
- keeps productive sequences when productivity information is present;
- prefixes barcodes when a manifest `Prefix` column is present;
- creates repertoires automatically; and
- writes the completed dataset to disk.

Set `rename_columns`, `preprocess`, `postprocess`, or `repertoire_schema` to `NULL` to disable the corresponding behavior.

Processing order

The function:

1. finds and reads the input files as one duckplyr table;
2. temporarily combines non-Parquet input into one Parquet file when `prematerialize = TRUE`;
3. renames columns;
4. applies preprocessing;
5. defines receptors using schema;
6. adds manifest information;
7. applies postprocessing;
8. defines repertoires when requested; and
9. writes and reopens the completed [ImmunData](#) dataset.

Manifests and repertoires

A manifest *annotates* each input file with biological information. The `repertoire_schema` argument chooses which annotation columns *define a repertoire* and therefore determine receptor counts and proportions.

With `path = "<manifest>"` and the default `repertoire_schema = "<auto>"`, all manifest columns are used and each manifest row becomes one repertoire. With an explicit file path or vector of paths, `"<auto>"` creates one repertoire per input file.

Output storage

The output folder is not a temporary cache. The returned object reads its receptor annotations from `annotations.parquet`, while `metadata.json` stores its schemas, repertoire summaries, and provenance. Keep this folder for as long as you need the object, or reopen it later with `read_immundata()`.

Important: Reusing the same `output_folder` replaces the existing `annotations.parquet` and `metadata.json` without creating a new version.

See Also

`read_manifest()`, `make_receptor_schema()`, `agg_receptors()`, `agg_repertoires()`, `make_default_preprocessing()`, `make_default_postprocessing()`, `read_immundata()`, `write_immundata()`, `ImmunData`

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Read one bulk AIRR file and preserve its abundance column
bulk_file <- system.file(
  "extdata/tsv",
  "sample_0_1k.tsv",
  package = "immundata"
)

bulk_idata <- read_repertoires(
  path = bulk_file,
  schema = c("cdr3_aa", "v_call"),
  count_col = "counts",
  output_folder = tempfile("immundata-bulk-")
)

tibble(
  n_records = bulk_idata |> count() |> pull(n),
  n_receptors = bulk_idata$receptors |> count() |> collect() |> pull(n),
  n_repertoires = nrow(bulk_idata$repertoires)
)

# Expected result:
#   n_records n_receptors n_repertoires
#         955         871             1

# Read multiple files and their sample information from a manifest
manifest_path <- system.file(
  "extdata/tsv",
  "manifest.csv",
  package = "immundata"
)
manifest <- read_manifest(manifest_path)

manifest_idata <- read_repertoires(
```

```

path = "<manifest>",
manifest = manifest,
schema = c("cdr3_aa", "v_call"),
count_col = "counts",
output_folder = tempfile("immundata-manifest-")
)

manifest_idata$repertoires |>
  select(Therapy, Response, n_barcodes, n_receptors) |>
  arrange(Response)
# Expected result:
#   Therapy Response n_barcodes n_receptors
#   ICI      FR      4725        871
#   CAR-T    PR      4758        867

# Read paired TRA-TRB receptors from a small single-cell table
paired_input <- tibble(
  cell_id = c("cell1", "cell1", "cell2", "cell2", "cell3"),
  locus = c("TRA", "TRB", "TRA", "TRB", "TRA"),
  v_call = c("TRAV1", "TRBV1", "TRAV1", "TRBV1", "TRAV2"),
  j_call = c("TRAJ1", "TRBJ1", "TRAJ1", "TRBJ1", "TRAJ2"),
  junction_aa = c("CAVA", "CASSB", "CAVA", "CASSB", "CAVC"),
  umi_count = c(10L, 8L, 12L, 9L, 7L)
)
paired_file <- tempfile(fileext = ".tsv")
readr::write_tsv(paired_input, paired_file)

paired_idata <- read_repertoires(
  path = paired_file,
  schema = make_receptor_schema(
    features = c("v_call", "j_call", "junction_aa"),
    chains = c("TRA", "TRB")
  ),
  barcode_col = "cell_id",
  locus_col = "locus",
  umi_col = "umi_count",
  repertoire_schema = NULL,
  output_folder = tempfile("immundata-paired-")
)

tibble(
  n_chains = paired_idata |> count() |> pull(n),
  n_cells = paired_idata |> collect() |> distinct(imd_barcode) |> nrow(),
  n_receptors = paired_idata$receptors |> count() |> collect() |> pull(n)
)
# Expected result:
#   n_chains n_cells n_receptors
#         4         2           1

```

Description

Use `rename_strata()` to replace automatic stratum labels with names that are clear in figures and result tables, such as "Control", "Treated", or "Tumour tissue".

Use this function after `agg_strata()` when labels such as "Strata1" do not describe the biological groups. The unit being changed is the stratum label. Stratum membership and the repertoires, receptors, cells, and chains remain unchanged.

The function returns a new `ImmunData` object. The original object is not changed.

Usage

```
rename_strata(
  idata,
  names,
  unnamed = c("error", "auto", "keep"),
  auto_prefix = "Strata"
)
```

Arguments

<code>idata</code>	An <code>ImmunData</code> object with strata already created by <code>agg_strata()</code> .
<code>names</code>	A named character vector or a data frame. New labels matched to <code>imd_strata_id</code> . Supply either: - a named character vector, such as <code>c("1" = "Control", "2" = "Treated")</code>; or - a data frame with columns <code>imd_strata_id</code> and <code>strata_name</code>. Every new label must be non-empty and unique.
<code>unnamed</code>	A character string. What to do when names does not include every stratum. The default, "error", asks for a complete mapping. Use "auto" to generate labels for missing strata or "keep" to preserve their current labels.
<code>auto_prefix</code>	A non-empty character string. Prefix used to generate labels when <code>unnamed = "auto"</code> . The default is "Strata".

Details

The names of a named character vector are the stratum IDs, not the current labels. Inspect `idata$strata` to find the ID for each biological group.

The mapping cannot contain unknown or repeated IDs, and the resulting labels must be unique across strata.

Value

A new `ImmunData` object with the requested labels in its `$strata` and `$repertoires` tables. All biological group assignments and repertoire summaries are preserved.

Storage details

The readable strata_name is stored in the repertoire and strata tables. The underlying chain annotations keep only imd_strata_id, so renaming a stratum does not rewrite or regroup chain-level data.

See Also

`agg_strata()`, `agg_repertoires()`

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Create treatment strata for the sample repertoires in the test data
treatment_groups <- get_test_idata() |>
  agg_repertoires(c("Response", "Therapy")) |>
  agg_strata(schema = "Therapy")

# Replace automatic labels with names suitable for a figure
labeled_groups <- treatment_groups |>
  rename_strata(
    names = c("1" = "CAR-T arm", "2" = "ICI arm")
  )

labeled_groups$strata |>
  select(Therapy, imd_strata_id, strata_name) |>
  arrange(imd_strata_id)
# Expected result:
#   Therapy imd_strata_id strata_name
#   CAR-T           1 CAR-T arm
#   ICI             2 ICI arm

# Only the labels changed. Each sample repertoire remains in the same
# treatment stratum.
```

write_immundata

Save an ImmunData object to disk

Description

Save ImmunData to disk so you can close R and continue the work later (I cannot believe it, but it works, I tried it). Use write_immundata() after importing or transforming repertoire data, or when you want a named snapshot before the next analysis step.

The unit saved is the complete [ImmunData](#) object. This includes retained chain rows, cell and receptor identifiers, repertoire and stratum definitions, and provenance. Saving does not add, remove, or change any biological unit.

Usage

```

write_immundata(
  idata,
  output_folder = NULL,
  tag = NULL,
  rehome = FALSE,
  compression = "zstd",
  compression_level = 9,
  verbose = getOption("immundata.verbose", TRUE)
)

```

Arguments

idata	An ImmunData object you want to save.
output_folder	A character string or NULL. Directory in which to write annotations.parquet and metadata.json. If NULL, the default, a managed snapshot is created at home_path/snapshots/<tag>/vNNN. The home path comes from the object's provenance.
tag	A character string or NULL. Snapshot tag. With output_folder = NULL, it names the managed snapshot series; if tag is also NULL, "default" is used. With an explicit output_folder, a supplied tag is recorded in the lineage but does not change the output path.
rehome	A logical value. Whether an explicit output_folder becomes the home for future managed snapshots. The default is FALSE, which preserves an existing home. If the object has no home yet, its first explicit output folder becomes the home with either value. TRUE requires an explicit output_folder.
compression	A character string or NULL. Parquet compression codec passed to DuckDB. The default is "zstd". Use NULL to let DuckDB choose.
compression_level	A number or NULL. Compression level for codecs that support it. The default is 9. Use NULL to let DuckDB choose.
verbose	A logical value. Whether to print progress and summary messages. Defaults to getOption("immundata.verbose", TRUE).

Details

Save to an explicit folder for a direct saved state, or use the object's home to create a versioned managed snapshot.

Value

Invisibly returns a newly reopened, disk-backed [ImmunData](#) object with provenance for the new save. The input idata remains unchanged.

Choose how to save

Supply `output_folder` to save a standalone state in a specific directory. This is useful when sharing a dataset or choosing its first project home. If the directory already contains an `ImmunData` dataset, its `annotations.parquet` and `metadata.json` are replaced.

Leave `output_folder = NULL` to create a managed snapshot. The function uses the object's home path and writes the next version under `snapshots/<tag>/vNNN`, for example `snapshots/baseline/v001`. Later writes with the same tag create `v002`, `v003`, and so on; earlier versions remain available. Use [read_immundata\(\)](#) with tag and version to reopen one.

Every save receives a new snapshot identifier and appends a provenance event. The returned object records the new saved directory as its current path.

Backend and serialization

The retained chain-level annotation table is materialized as compressed `annotations.parquet`. Materialization executes any pending lazy dplyr calculations. `metadata.json` serializes format and package versions, receptor, repertoire, and stratum schemas, the small repertoire table, the snapshot identifier, lineage events, and provenance paths.

Receptor and stratum views are not written as separate files; they can be reconstructed from the annotation table and metadata. This Parquet and JSON pair is an `ImmunData`-specific serialization, not an RDS file.

See Also

[read_immundata\(\)](#) for continuing a saved analysis, [read_repertoires\(\)](#) for importing AIRR-seq files, [ImmunData](#)

Examples

```
library(immundata)
library(dplyr)

options(immundata.verbose = FALSE)

# Save a small immune-repertoire analysis
idata <- get_test_idata()
save_dir <- tempfile("saved-immundata-")

saved_idata <- write_immundata(idata, save_dir)

list.files(save_dir)
# Expected result: the analysis is serialized as two files.
# [1] "annotations.parquet" "metadata.json"

# Continue the analysis from the saved files
continued_idata <- read_immundata(save_dir)

continued_idata |>
  collect() |>
  summarise(
```

```
        n_chains = n(),
        n_receptors = n_distinct(imd_receptor_id)
    )
# Expected result: all 1,902 chain rows and 1,668 receptors are restored.
#   n_chains n_receptors
#       1902       1668

unlink(save_dir, recursive = TRUE)
```

Index

- * **Annotation**
 - annotate_immundata, 10
 - annotate_seurat, 15
- * **aggregation**
 - agg_receptors, 2
 - agg_repertoires, 4
 - agg_strata, 7
 - rename_strata, 55
- * **annotation**
 - annotate_immundata, 10
- * **core_immundata**
 - ImmunData, 29
- * **filtering**
 - downsample_immundata, 20
 - filter_immundata, 22
- * **ingestion**
 - from_immunarch, 26
 - imd_rename_cols, 27
 - read_immundata, 45
 - read_manifest, 48
 - read_repertoires, 49
 - write_immundata, 56
- * **mutation**
 - mutate_immundata, 39
- * **operations**
 - collect.ImmunData, 16
 - compute.ImmunData, 17
 - count.ImmunData, 17
 - dimnames.ImmunData, 19
 - dimnames<- .ImmunData, 19
 - names<- .ImmunData, 44
 - print.ImmunData, 44
- * **processing**
 - make_default_preprocessing, 33
- * **schema**
 - imd_schema, 28
- * **utils**
 - make_receptor_schema, 36
 - make_seq_options, 38
- agg_receptors, 2
- agg_receptors(), 37, 53
- agg_repertoires, 4
- agg_repertoires(), 4, 7, 8, 13, 20, 21, 24, 30, 32, 40, 51, 53, 56
- agg_strata, 7
- agg_strata(), 6, 30, 32, 55, 56
- anndataR::AbstractAnnData, 9
- annotate(annotate_immundata), 10
- annotate(), 30, 32
- annotate_anndata, 9
- annotate_barcodes(annotate_immundata), 10
- annotate_barcodes(), 12
- annotate_chains(annotate_immundata), 10
- annotate_chains(), 12
- annotate_immundata, 10
- annotate_receptors(annotate_immundata), 10
- annotate_receptors(), 12, 39, 40
- annotate_seurat, 15
- assert_receptor_schema(make_receptor_schema), 36
- collect.ImmunData, 16
- compute.ImmunData, 17
- count.ImmunData, 17
- dimnames.ImmunData, 19
- dimnames<- .ImmunData, 19
- downsample_immundata, 20
- dplyr::collect(), 16, 18, 30, 45
- dplyr::compute(), 17
- dplyr::count(), 18, 45
- dplyr::filter, 22
- dplyr::filter(), 23, 24
- dplyr::left_join(), 13
- dplyr::mutate, 39
- dplyr::mutate(), 40
- dplyr::pull(), 18

duckplyr::read_parquet_duckdb(), 46
 filter.ImmunData (filter_immundata), 22
 filter_barcodes (filter_immundata), 22
 filter_immundata, 22
 filter_immundata(), 13, 21, 30, 32, 38–40
 filter_receptors (filter_immundata), 22
 from_immunarch, 26

 imd_drop_cols (imd_rename_cols), 27
 imd_drop_cols(), 35
 imd_rename_cols, 27
 imd_rename_cols(), 28, 35
 imd_schema, 28
 imd_schema(), 27, 37
 ImmunData, 2, 4–11, 13, 15, 17, 18, 20, 22–24, 26–28, 29, 36, 39, 40, 44–47, 49, 51–53, 55–58

 make_barcode_prefix
 (make_default_preprocessing), 33
 make_default_postprocessing
 (make_default_preprocessing), 33
 make_default_postprocessing(), 50, 53
 make_default_preprocessing, 33
 make_default_preprocessing(), 27, 50, 53
 make_exclude_columns
 (make_default_preprocessing), 33
 make_productive_filter
 (make_default_preprocessing), 33
 make_receptor_schema, 36
 make_receptor_schema(), 3, 4, 28, 32, 50–53
 make_seq_options, 38
 make_seq_options(), 23, 24, 39, 40
 mutate.ImmunData (mutate_immundata), 39
 mutate_immundata, 39
 mutate_immundata(), 13, 24, 30, 32, 38, 39

 names<- .ImmunData, 44

 print.ImmunData, 44

 read_immundata, 45
 read_immundata(), 5, 6, 26, 30, 32, 49, 53, 58
 read_manifest, 48
 read_manifest(), 50, 51, 53
 read_repertoires, 49
 read_repertoires(), 2, 4–6, 20, 26, 27, 30, 32, 33, 35, 37, 45, 47, 58
 rename_strata, 54
 rename_strata(), 8

 SeuratObject::AddMetaData, 15

 tempdir(), 51
 test_receptor_schema
 (make_receptor_schema), 36

 write_immundata, 56
 write_immundata(), 6, 21, 30, 32, 45, 47, 53