{tidytof} Supplementary Information

Keyes et al.

2022-01-25

# Table of contents

[1. Table of contents 1](#_Toc94023088)

[2. Supplementary Tables 2](#_Toc94023089)

[2.1. Supplementary Table 1 - “Tidy” CyTOF data 2](#_Toc94023090)

[2.2. Supplementary Table 2 - Cell-level verbs 2](#_Toc94023091)

[2.3. Supplementary Table 3 - Cluster-level verbs 2](#_Toc94023092)

[2.4. Supplementary Table 4 - Sample-level verbs 2](#_Toc94023093)

[3. Supplementary Notes 3](#_Toc94023094)

[3.1. Supplementary Note 1 - Getting started with {tidytof} 3](#_Toc94023095)

[3.1.1. Prerequisites 3](#_Toc94023096)

[3.1.2. Workflow basics 4](#_Toc94023097)

[3.1.3. {tidytof} verb syntax 5](#_Toc94023098)

[3.1.4. Pipelines 6](#_Toc94023099)

[3.1.5. Other tips 7](#_Toc94023100)

# Supplementary Tables

## Supplementary Table 1 - “Tidy” CyTOF data

|  | Proteomic data | | | {tidytof} calculations | | | Metadata | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **cell\_id** | **protein\_1** | **protein\_2** | **protein\_3** | **tsne\_1** | **tsne\_2** | **cluster** | **sample\_type** | **patient** |
| Cell 1 | 1.02 | -0.10 | -0.08 | -6.84 | -7.50 | 1 | healthy | patient 1 |
| Cell 2 | 1.54 | 3.27 | 1.72 | 14.17 | -1.99 | 2 | healthy | patient 1 |
| Cell 3 | 0.43 | 0.36 | 0.42 | -0.31 | 10.30 | 4 | healthy | patient 1 |
| Cell 4 | 1.50 | -0.05 | -0.02 | -5.14 | 12.17 | 1 | healthy | patient 2 |
| Cell 5 | 0.03 | -0.19 | 0.56 | -4.12 | 7.26 | 4 | healthy | patient 2 |
| Cell 6 | 0.00 | 0.25 | 0.00 | -4.52 | -4.70 | 4 | healthy | patient 2 |
| Cell 7 | 0.58 | 1.24 | -0.10 | -17.37 | -2.64 | 3 | cancer | patient 3 |
| Cell 8 | 1.97 | 2.08 | 1.50 | 12.27 | -1.31 | 2 | cancer | patient 3 |
| Cell 9 | 0.20 | 1.04 | -0.11 | -19.30 | -2.48 | 3 | cancer | patient 3 |

**Supplementary Table 1 - Example of a {tidytof} data frame.** {tidytof} represents CyTOF data in a “tidy format” using an extended data frame called a “tof\_tbl”. In this format, data are represented such that each cell is given its own row and each measurement or piece of metadata is given its own column.

## Supplementary Table 2 - Cell-level verbs

## NULL

## Supplementary Table 3 - Cluster-level verbs

## NULL

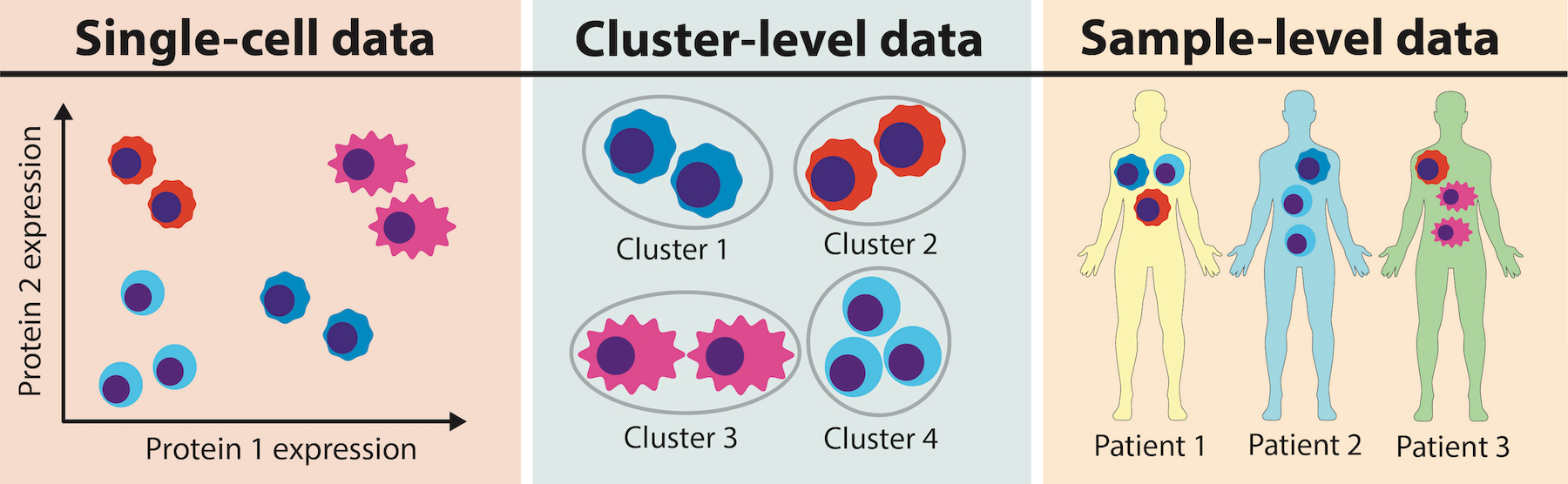
## Supplementary Table 4 - Sample-level verbs

## NULL

# Supplementary Notes

## Supplementary Note 1 - Getting started with {tidytof}

Analyzing single-cell data can be surprisingly complicated. One one hand, this is partially because single-cell data analysis is an incredibly active area of research, with new methods being published on a weekly - or even daily! - basis. Accordingly, when new tools are published, they often require researchers to learn unique, method-specific application programming interfaces (APIs) with distinct requirements for input data formatting, function syntax, and output data structure. On the other hand, analyzing single-cell data can be challenging because it often involves simultaneously asking questions at multiple levels of biological scope - the single-cell level, the cell subpopulation (i.e. cluster) level, and the whole-sample or whole-patient level - each of which has distinct data processing needs.



**Figure 1 -** Researchers are often interested in biological questions that manifest at multiple levels of single-cell data processing. At the single-cell level, you may be interested in individual cells’ marker expression profiles. At the cluster level, you may be interested in how cells organize into subpopulations with shared phenotypic characteristics of varying size and similarity. And at the whole-sample level, you may be interested in how individual cells’ and clusters’ characteristics manifest in disease, treatment response, or a variety of other clinical or experimental variables.

To address both of these challenges for [mass cytometry (CyTOF)](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4860251/), {tidytof} implements a concise, integrated “grammar” of single-cell data analysis capable of answering a variety of biological questions. Available as an open-source R package, {tidytof} provides an easy-to-use pipeline for analyzing CyTOF data by automating many common data-processing tasks under a common [“tidy data”](https://r4ds.had.co.nz/tidy-data.html) interface. This vignette introduces you to the tidytof’s high-level API and shows quick examples of how they can be applied to CyTOF datasets.

### Prerequisites

{tidytof} makes heavy use of two concepts that may be unfamiliar to R beginners. The first is the {magrittr} pipe (%>%), which you can read about [here](https://r4ds.had.co.nz/pipes.html). The second is “grouping” data in a data.frame or tibble using dplyr::group\_by, which you can read about [here](https://dplyr.tidyverse.org/articles/grouping.html). Most {tidytof} users will also benefit from a relatively in-depth understanding of the dplyr package, which has a wonderful introductory vignette here:

vignette("dplyr")

Everything else should be self-explanatory for both beginner and advanced R users, though if you have *zero* background in running R code, you should read [this chapter](https://r4ds.had.co.nz/workflow-basics.html) of [R for Data Science](https://r4ds.had.co.nz/index.html) by Hadley Wickham.

### Workflow basics

Broadly speaking, {tidytof}’s functionality is organized to support the 3 levels of analysis inherent to single-cell data described above:

1. Reading, writing, preprocessing, and visualizing data at the level of **individual cells**
2. Identifying and describing cell **subpopulations** or **clusters**
3. Building models (for inference or prediction) at the level of **patients** or **samples**

{tidytof} provides functions (or “verbs”) that operate at each of these levels of analysis:

* Cell-level data:
  + tof\_read\_data() reads single-cell data from FCS or CSV files on disk into a tidy data frame called a tof\_tbl. tof\_tbls represent each cell as a row and each protein measurement (or other piece of information associated with a given cell) as a column.
  + tof\_preprocess() transforms protein expression values using a user-provided function (i.e. log-transformation, centering, scaling)
  + tof\_downsample() reduces the number of cells in a tof\_tibble via subsampling.
  + tof\_reduce\_dimensions() performs dimensionality reduction (across columns)
  + tof\_write\_data writes single-cell data in a tof\_tibble back to disk in the form of an FCS or CSV file.
* Cluster-level data:
  + tof\_cluster() clusters cells using one of several algorithms commonly applied to CyTOF data
  + tof\_metacluster() agglomerates clusters into a smaller number of metaclusters
  + tof\_daa() performs differential abundance analysis (DAA) for clusters or metaclusters across experimental groups
  + tof\_dea() performs differential expression analysis (DEA) for clusters’ or metaclusters’ marker expression levels across experimental groups
  + tof\_extract\_features() computes summary statistics (such as mean marker expression) for each cluster. Also (optionally) pivots these summary statistics into a sample-level tidy data frame in which each row represents a sample and each column represents a cluster-level summary statistic.
* Sample-level data:
  + tof\_split\_data() splits sample-level data into a training and test set for predictive modeling
  + tof\_create\_grid() creates an elastic net hyperparameter search grid for model tuning
  + tof\_train\_model() trains a sample-level elastic net model and saves it as a tof\_model object
  + tof\_predict() Applies a trained tof\_model to new data to predict sample-level outcomes
  + tof\_assess\_model() calculates performance metrics for a trained tof\_model

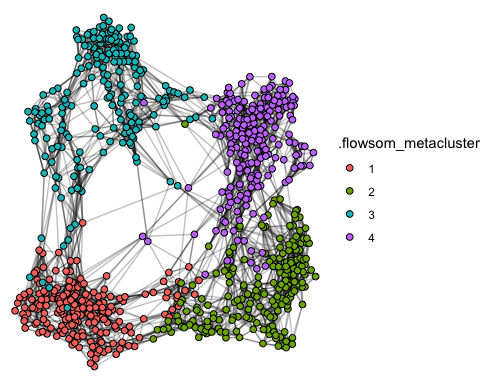
### {tidytof} verb syntax

With very few exceptions, {tidytof} functions follow a specific, shared syntax that involves 3 types of arguments that always occur in the same order. These argument types are as follows:

1. For almost all {tidytof} functions, the first argument is a data frame (or tibble). This enables the use of the pipe (%>%) for multi-step calculations, which means that your first argument for most functions will be implicit (passed from the previous function using the pipe). This also means that most {tidytof} functions are so-called [“single-table verbs,”](https://cran.r-project.org/web/packages/dplyr/vignettes/two-table.html) with the exception of tof\_cluster\_ddpr, which is a “two-table verb” (for details about how to use tof\_cluster\_ddpr, see the “clustering-and-metaclustering” vignette).
2. The second group of arguments are called *column specifications*, and they end in the suffix \_col or \_cols. Column specifications are unquoted column names that tell a {tidytof} verb which columns to compute over for a particular operation. For example, the cluster\_cols argument in tof\_cluster allows the user to specify which column in the input data frames should be used to perform the clustering. Regardless of which verb requires them, column specifications support [tidyselect helpers](https://tidyselect.r-lib.org/reference/language.html).
3. Finally, the third group of arguments for each {tidytof} verb are called *method specifications*, and they’re comprised of every argument that isn’t an input data frame or a column specification. Whereas column specifications represent which columns should be used to perform an operation, method specifications represent the details of how that operation should be performed. For example, the tof\_cluster\_phenograph() function requires the method specification num\_neighbors, which specifies how many nearest neighbors should be used to construct the PhenoGraph algorithm’s k-nearest-neighbor graph. In most cases, {tidytof} sets reasonable defaults for each verb’s particular method specifications, but your workflows are can also be customized by experimenting with non-default values.

The following code demonstrates how {tidytof} verb syntax looks in practice, with column and method specifications explicitly pointed out:

set.seed(777L)  
  
ddpr\_data %>%   
 tof\_preprocess() %>%  
 tof\_cluster(  
 cluster\_cols = starts\_with("cd"), # column specification  
 method = "flowsom", # method specification,   
 num\_metaclusters = 5 # method specification  
 ) %>%   
 tof\_downsample(  
 group\_cols = .flowsom\_metacluster, # column specification  
 method = "constant", # method specification  
 num\_cells = 200 # method specification  
 ) %>%  
 tof\_plot\_cells\_layout(  
 knn\_cols = starts\_with("cd"), # column specification  
 color\_col = .flowsom\_metacluster, # column specification  
 num\_neighbors = 7, # method specification  
 node\_size = 2 # method specification  
 )

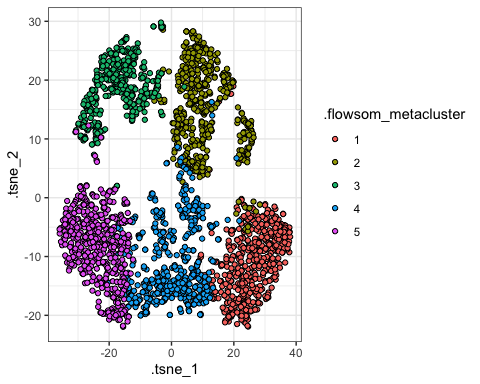


### Pipelines

{tidytof} verbs can be used on their own or in combination with one another using the pipe (%>%) operator. For example, here is a multistep “pipeline” that takes a built-in {tidytof} dataset and performs the following analytical steps:

1. Arcsinh-transform each column of protein measurements (the default behavior of the tof\_preprocess verb
2. Cluster our cells based on the surface markers in our panel
3. Downsample the dataset such that 100 random cells are picked from each cluster
4. Perform dimensionality reduction on the downsampled dataset using tSNE
5. Visualize the clusters using a low-dimensional tSNE embedding

ddpr\_data %>%   
 # step 1  
 tof\_preprocess() %>%   
 # step 2  
 tof\_cluster(  
 cluster\_cols = starts\_with("cd"),   
 method = "flowsom",   
 seed = 2020L  
 ) %>%   
 # step 3  
 tof\_downsample(  
 group\_cols = .flowsom\_metacluster,   
 method = "constant",   
 num\_cells = 500  
 ) %>%   
 # step 4  
 tof\_reduce\_dimensions(method = "tsne") %>%   
 # step 5  
 tof\_plot\_cells\_dr(  
 dr\_cols = contains("tsne"),  
 color\_col = .flowsom\_metacluster   
 )



### Other tips

{tidytof} was designed by a multidisciplinary team of wet-lab biologists, bioinformaticians, and physician-scientists who analyze CyTOF and other kinds of single-cell data to solve a variety of problems. As a result, {tidytof}’s high-level API was designed with great care to mirror that of the {tidyverse} itself - that is, to be [human-centered, consistent, composable, and inclusive](https://design.tidyverse.org/unifying-principles.html) for a wide userbase.

Practically speaking, this means a few things about using {tidytof}.

First, it means that {tidytof} was designed with a few quality-of-life features in mind. For example, you may notice that most {tidytof} functions begin with the prefix tof\_. This is intentional, as it will allow you to use your development environment’s code-completing software to search for {tidytof} functions easily (even if you can’t remember a specific function name). For this reason, we recommend using {tidytof} within the RStudio development environment; however, many code editors have predictive text functionality that serves a similar function. In general, {tidytof} verbs are organized in such a way that your IDE’s code-completion tools should also allow you to search for (and compare) related functions with relative ease. (For instance, the tof\_cluster\_ prefix is used for all clustering functions, and the tof\_downsample\_ prefix is used for all downsampling functions).

Second, it means that {tidytof} functions *should* be relatively intuitive to use due to their shared logic - in other words, if you understand how to use one {tidytof} function, you should understand how to use most of the others. An example of shared logic across {tidytof} functions is the argument group\_cols, which shows up in multiple verbs (tof\_downsample, tof\_cluster, tof\_daa, tof\_dea, tof\_extract\_features, and tof\_write\_data). In each case, group\_cols works the same way: it accepts an unquoted vector of column names (specified manually or using [tidyselection](https://r4ds.had.co.nz/transform.html#select)) that should be used to group cells before an operation is performed. This idea generalizes throughout {tidytof}: if you see an argument in one place, it will behave identically (or at least very similarly) wherever else you encounter it.

Finally, it means that {tidytof} is optimized first for ease-of-use, then for performance. Because humans and computers interact with data differently, there is always a trade-off between choosing a data representation that is intuitive to a human user vs. choosing a data representation optimized for computational speed and memory efficiency. When these design choices conflict with one another, our team tends to err on the side of choosing a representation that is easy-to-understand for users even at the expense of small performance costs. Ultimately, this means that {tidytof} may not be the optimal tool for every CyTOF analysis, though hopefully its general framework will provide most users with some useful functionality.

knitr::knit\_exit()