

---

# **Pegasus Documentation**

***Release 1.8.1***

**Bo Li, Joshua Gould, Yiming Yang, Siranush Sarkizova, et al.**

**Aug 24, 2023**



**CONTENTS**

<b>1   Release Highlights in Current Stable</b>	<b>3</b>
<b>Bibliography</b>	<b>125</b>
<b>Index</b>	<b>127</b>



Pegasus is a tool for analyzing transcriptomes of millions of single cells. It is a command line tool, a python package and a base for Cloud-based analysis workflows.

[Read documentation](#)



## RELEASE HIGHLIGHTS IN CURRENT STABLE

### 1.1 1.8.1 August 23, 2023

- Bug fix in cell marker JSON files for `infer_cell_types` function.

### 1.2 1.8.0 July 21, 2023

#### New Feature and Improvement

- Update `human_immune` and `human_lung` marker sets.
- Add `mouse_liver` marker set.
- Add `split_one_cluster` function to subcluster one cluster into a specified number of subclusters.
- Update `neighbors` function to set `use_cache=False` by default, and adjust `K` to `min(K, int(sqrt(n_samples)))`. [PR 272]
- In `infer_doublets` function, argument `manual_correction` now accepts a float number threshold specified by users for cut-off. [PR 275]

#### Bug Fix

- Fix divide by zero issue in `integrative_nmf` function. [PR 258]
- Compatibility with Pandas v2.0. [PR 261]
- Allow `infer_doublets` to use any count matrix with key name specified by users. [PR 268 Thanks to Donghoon Lee]

#### 1.2.1 Installation

Pegasus works with Python 3.7, 3.8 and 3.9.

### Linux

#### Ubuntu/Debian

#### Prerequisites

On Ubuntu/Debian Linux, first install the following dependency by:

```
sudo apt install build-essential
```

Next, you can install Pegasus system-wide by PyPI (see [Ubuntu/Debian install via PyPI](#)), or within a Miniconda environment (see [Install via Conda](#)).

To use the Force-directed-layout (FLE) embedding feature, you'll need Java. You can either install [Oracle JDK](#), or install OpenJDK which is included in Ubuntu official repository:

```
sudo apt install default-jdk
```

#### Ubuntu/Debian install via PyPI

First, install Python 3, *pip* tool for Python 3 and *Cython* package:

```
sudo apt install python3 python3-pip
python3 -m pip install --upgrade pip
python3 -m pip install cython
```

Now install Pegasus with the required dependencies via *pip*:

```
python3 -m pip install pegasuspy
```

or install Pegasus with all dependencies:

```
python3 -m pip install pegasuspy[all]
```

Alternatively, you can install Pegasus with some of the additional optional dependencies as below:

- **torch**: This includes `harmony-pytorch` for data integration and `nmf-torch` for NMF and iNMF data integration, both of which uses PyTorch:

```
python3 -m pip install pegasuspy[torch]
```

- **louvain**: This includes `louvain` package, which provides Louvain clustering algorithm, besides the default Leiden algorithm in Pegasus:

```
python3 -m pip install pegasuspy[louvain]
```

---

**Note:** If installing from Python 3.9, to install `louvain`, you'll need to install the following packages system-wide first in order to locally compile it:

```
sudo apt install flex bison libtool
```

- **tsne**: This package is to calculate t-SNE plots using a fast algorithm FIt-SNE:



```
sudo apt install libfftw3-dev
python3 -m pip install pegasuspy[tsne]
```

- **forceatlas**: This includes forceatlas2-python package, a multi-thread Force Atlas 2 implementation for trajectory analysis:

```
python3 -m pip install pegasuspy[forceatlas]
```

- **scanorama**: This includes scanorama package, a widely-used method for batch correction:

```
python3 -m pip install pegasuspy[scanorama]
```

- **mkl**: This includes mkl package, which improve math routines for science and engineering applications. Notice that *mkl* not included in **pegasuspy[all]** above:

```
python3 -m pip install pegasuspy[mkl]
```

- **rpy2**: This includes rpy2 package, which is used by Pegasus wrapper on R functions, such as fgsea and DESeq2:

```
python3 -m pip install pegasuspy[rpy2]
```

- **scvi**: This includes scvi-tools package for data integration:

```
python3 -m pip install pegasuspy[scvi]
```

---

## Fedora

### Prerequisites

On Fedora Linux, first install the following dependency by:

```
sudo dnf install gcc gcc-c++
```

Next, you can install Pegasus system-wide by PyPI (see [Fedora install via PyPI](#)), or within a Miniconda environment (see [Install via Conda](#)).

To use the Force-directed-layout (FLE) embedding feature, you'll need Java. You can either install [Oracle JDK](#), or install OpenJDK which is included in Fedora official repository (e.g. `java-latest-openjdk`):

```
sudo dnf install java-latest-openjdk
```

or other OpenJDK version chosen from the searching result of command:

```
dnf search openjdk
```

## Fedora install via PyPI

We'll use Python 3.8 in this tutorial.

First, install Python 3 and *pip* tool for Python 3:

```
sudo dnf install python3.8
python3.8 -m ensurepip --user
python3.8 -m pip install --upgrade pip
```

Now install Pegasus with the required dependencies via *pip*:

```
python3.8 -m pip install pegasuspy
```

or install Pegasus with all dependencies:

```
python3.8 -m pip install pegasuspy[all]
```

Alternatively, you can install Pegasus with some of the additional optional dependencies as below:

- **torch**: This includes `harmony-pytorch` for data integration and `nmf-torch` for NMF and iNMF data integration, both of which uses PyTorch:

```
python3.8 -m pip install pegasuspy[torch]
```

- **louvain**: This includes `louvain` package, which provides Louvain clustering algorithm, besides the default Leiden algorithm in Pegasus:

```
python3.8 -m pip install pegasuspy[louvain]
```

---

**Note:** If installing from Python 3.9, to install `louvain`, you'll need to install the following packages system-wide first in order to locally compile it:

```
sudo dnf install flex bison libtool
```

---

- **tsne**: This package is to calculate t-SNE plots using a fast algorithm FIt-SNE:

```
sudo apt install libfftw3-dev
python3.8 -m pip install pegasuspy[tsne]
```

- **forceatlas**: This includes `forceatlas2-python` package, a multi-thread Force Atlas 2 implementation for trajectory analysis:

```
python3.8 -m pip install pegasuspy[forceatlas]
```

- **scanorama**: This includes `scanorama` package, a widely-used method for batch correction:

```
python3.8 -m pip install pegasuspy[scanorama]
```

- **mk1**: This includes `mk1` package, which improve math routines for science and engineering applications. Notice that `mk1` not included in **pegasuspy[all]** above:

```
python3.8 -m pip install pegasuspy[mk1]
```

- **rpy2**: This includes `rpy2` package, which is used by Pegasus wrapper on R functions, such as `fgsea` and `DESeq2`:

```
python3.8 -m pip install pegasuspy[rpy2]
```

- **scvi**: This includes `scvi-tools` package for data integration:

```
python3.8 -m pip install pegasuspy[scvi]
```

## macOS

### Prerequisites

First, install Homebrew by following the instruction on its website: <https://brew.sh/>. Then install the following dependencies:

```
brew install libomp
```

And install macOS command line tools:

```
xcode-select --install
```

Next, you can install Pegasus system-wide by PyPI (see [macOS installation via PyPI](#)), or within a Miniconda environment (see [Install via Conda](#)).

To use the Force-directed-layout (FLE) embedding feature, you'll need Java. You can either install [Oracle JDK](#), or install OpenJDK via Homebrew:

```
brew install java
```

### macOS install via PyPI

1. You need to install Python and *pip* tool first:

```
brew install python3
python3 -m pip install --upgrade pip
```

2. Now install Pegasus with required dependencies via *pip*:

```
python3 -m pip install pegasuspy
```

or install Pegasus with all dependencies:

```
python3 -m pip install pegasuspy[all]
```

Alternatively, you can install Pegasus with some of the additional optional dependencies as below:

- **torch**: This includes `harmony-pytorch` for data integration and `nmf-torch` for NMF and iNMF data integration, both of which uses PyTorch:

```
python3 -m pip install pegasuspy[torch]
```

- **louvain**: This includes `louvain` package, which provides Louvain clustering algorithm, besides the default Leiden algorithm in Pegasus:

```
python3 -m pip install pegasuspy[louvain]
```

- **tsne**: This package is to calculate t-SNE plots using a fast algorithm FIt-SNE:

```
sudo apt install libfftw3-dev  
python3 -m pip install pegasuspy[tsne]
```

- **forceatlas**: This includes forceatlas2-python package, a multi-thread Force Atlas 2 implementation for trajectory analysis:

```
python3 -m pip install pegasuspy[forceatlas]
```

- **scanorama**: This includes scanorama package, a widely-used method for batch correction:

```
python3 -m pip install pegasuspy[scanorama]
```

- **mkl**: This includes mkl packages, which improve math routines for science and engineering applications. Notice that *mkl* not included in **pegasuspy[all]** above:

```
python3 -m pip install pegasuspy[mkl]
```

- **rpy2**: This includes rpy2 package, which is used by Pegasus wrapper on R functions, such as fgsea and DESeq2:

```
python3 -m pip install pegasuspy[rpy2]
```

- **scvi**: This includes scvi-tools package for data integration:

```
python3 -m pip install pegasuspy[scvi]
```

---

## Install via Conda

Alternatively, you can install Pegasus via Conda, which is a separate virtual environment without touching your system-wide packages and settings.

You can install [Anaconda](#), or [Miniconda](#) (a minimal installer of conda). In this tutorial, we'll use Miniconda.

1. Download [Miniconda installer](#) for your OS. For example, if on 64-bit Linux, then use the following commands to install Miniconda:

```
export CONDA_PATH=/home/foo  
bash Miniconda3-latest-Linux-x86_64.sh -p $CONDA_PATH/miniconda3  
mv Miniconda3-latest-Linux-x86_64.sh $CONDA_PATH/miniconda3  
source ~/.bashrc
```

where `/home/foo` should be replaced by the directory to which you want to install Miniconda. Similarly for macOS.

2. Create a conda environment for pegasus. This tutorial uses **pegasus** as the environment name, but you are free to choose your own:

```
conda create -n pegasus -y python=3.8
```

Also notice that Python 3.8 is used in this tutorial. To choose a different version of Python, simply change the version number in the command above.

3. Enter pegasus environment by activating:

```
conda activate pegasus
```

4. Install Pegasus via conda:

```
conda install -c bioconda pegasuspy
```

5. (Optional) Use the following command to add support `nmf-torch`:

```
pip install nmf-torch
```

Enable Force Atlas 2 for trajectory analysis:

```
conda install -c bioconda forceatlas2-python
```

Enable support on scanorama:

```
conda install -c bioconda scanorama
```

Enable support on fgsea and deseq2 packages:

```
conda install -c bioconda rpy2 bioconductor-fgsea bioconductor-deseq2
```

Enable support on scvi-tools:

```
conda install -c conda-forge scvi-tools
```

---

## Install via Singularity

[Singularity](#) is a container engine similar to Docker. Its main difference from Docker is that Singularity can be used with unprivileged permissions.

---

**Note:** Please notice that Singularity Hub has been offline since April 26th, 2021 (see [blog post](#)). All existing containers held there are in archive, and we can no longer push new builds.

So if you fetch the container from Singularity Hub using the following command:

```
singularity pull shub://klarman-cell-observatory/pegasus
```

it will just give you a Singularity container of Pegasus v1.2.0 running on Ubuntu Linux 20.04 base with Python 3.8, in the name `pegasus_latest.sif` of about 2.4 GB.

---

On your local machine, first [install Singularity](#), then you can use our [Singularity spec file](#) to build a Singularity container by yourself:

```
singularity build pegasus.sif Singularity
```

where Singularity is the spec filename.

After that, you can interact with it by running the following command:

```
singularity run pegasus.sif
```

Please refer to [Singularity image interaction guide](#) for details.

---

### Development Version

To install Pegasus development version directly from its [GitHub repository](#), please do the following steps:

1. Install prerequisite libraries as mentioned in above sections.
2. Install Git. See [here](#) for how to install Git.
3. Use git to fetch repository source code, and install from it:

```
git clone https://github.com/lilab-bcb/pegasus.git
cd pegasus
pip install -e .[all]
```

where `-e` option of `pip` means to install in editing mode, so that your Pegasus installation will be automatically updated upon modifications in source code.

### 1.2.2 Use Pegasus as a command line tool

Pegasus can be used as a command line tool. Type:

```
pegasus -h
```

to see the help information:

```
Usage:
  pegasus <command> [<args>...]
  pegasus -h | --help
  pegasus -v | --version
```

pegasus has 9 sub-commands in 6 groups.

- Preprocessing:

#### **aggregate\_matrix**

Aggregate sample count matrices into a single count matrix. It also enables users to import metadata into the count matrix.

- Demultiplexing:

#### **demuxEM**

Demultiplex cells/nuclei based on DNA barcodes for cell-hashing and nuclei-hashing data.

- Analyzing:

#### **cluster**

Perform first-pass analysis using the count matrix generated from ‘aggregate\_matrix’. This sub-command could perform low quality cell filtration, batch correction, variable gene selection, dimension reduction, diffusion map calculation, graph-based clustering, visualization. The final results will be written into zarr-formatted file, or h5ad file, which Seurat could load.

**de\_analysis**

Detect markers for each cluster by performing differential expression analysis per cluster (within cluster vs. outside cluster). DE tests include Welch's t-test, Fisher's exact test, Mann-Whitney U test. It can also calculate AUROC values for each gene.

**find\_markers**

Find markers for each cluster by training classifiers using LightGBM.

**annotate\_cluster**

This subcommand is used to automatically annotate cell types for each cluster based on existing markers. Currently, it works for human/mouse immune/brain cells, etc.

- Plotting:

**plot**

Make static plots, which includes plotting tSNE, UMAP, and FLE embeddings by cluster labels and different groups.

- Web-based visualization:

**scp\_output**

Generate output files for single cell portal.

- MISC:

**check\_indexes**

Check CITE-Seq/hashing indexes to avoid index collision.

---

## Quick guide

Suppose you have `example.csv` ready with the following contents:

```
Sample,Source,Platform,Donor,Reference,Location
sample_1,bone_marrow,NextSeq,1,GRCh38,/my_dir/sample_1/raw_feature_bc_matrices.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,/my_dir/sample_2/raw_feature_bc_matrices.h5
sample_3,pbmc,NextSeq,1,GRCh38,/my_dir/sample_3/raw_gene_bc_matrices_h5.h5
sample_4,pbmc,NextSeq,2,GRCh38,/my_dir/sample_4/raw_gene_bc_matrices_h5.h5
```

You want to analyze all four samples but correct batch effects for bone marrow and pbmc samples separately. You can run the following commands:

```
pegasus aggregate_matrix --attributes Source,Platform,Donor example.csv example.aggr
pegasus cluster -p 20 --correct-batch-effect --batch-group-by Source --louvain --umap
↪ example.aggr.zarr.zip example
pegasus de_analysis -p 20 --labels louvain_labels example.zarr.zip example.de.xlsx
pegasus annotate_cluster example.zarr.zip example.anno.txt
pegasus plot compo --groupby louvain_labels --condition Donor example.zarr.zip example.
↪ composition.pdf
pegasus plot scatter --basis umap --attributes louvain_labels,Donor example.zarr.zip
↪ example.umap.pdf
```

The above analysis will give you UMAP embedding and Louvain cluster labels in `example.zarr.zip`, along with differential expression analysis result stored in `example.de.xlsx`, and putative cluster-specific cell type annotation stored in `example.anno.txt`. You can investigate donor-specific effects by looking at `example.composition.pdf`. `example.umap.pdf` plotted UMAP colored by `louvain_labels` and Donor info side-by-side.

---

## pegasus aggregate\_matrix

The first step for single cell analysis is to generate one count matrix from cellranger's channel-specific count matrices. `pegasus aggregate_matrix` allows aggregating arbitrary matrices with the help of a CSV file.

Type:

```
pegasus aggregate_matrix -h
```

to see the usage information:

Usage:

```
    pegasus aggregate_matrix <csv_file> <output_name> [--restriction <restriction>...
    ↪ --attributes <attributes> --default-reference <reference> --select-only-singlets --
    ↪ min-genes <number>]
    pegasus aggregate_matrix -h
```

- Arguments:

`csv_file` Input csv-formatted file containing information of each sc/snRNA-seq sample. This file must contain at least 2 columns - `Sample`, sample name; and `Location`, location of the sample count matrix in either 10x v2/v3, DGE, mtx, csv, tsv or loom format. Additionally, an optional `Reference` column can be used to select samples generated from a same reference (e.g. mm10). If the count matrix is in either DGE, mtx, csv, tsv, or loom format, the value in this column will be used as the reference since the count matrix file does not contain reference name information. Moreover, the `Reference` column can be used to aggregate count matrices generated from different genome versions or gene annotations together under a unified reference. For example, if we have one matrix generated from mm9 and the other one generated from mm10, we can write mm9\_10 for these two matrices in their `Reference` column. Pegasus will change their references to 'mm9\_10' and use the union of gene symbols from the two matrices as the gene symbols of the aggregated matrix. For HDF5 files (e.g. 10x v2/v3), the reference name contained in the file does not need to match the value in this column. In fact, we use this column to rename references in HDF5 files. For example, if we have two HDF files, one generated from mm9 and the other generated from mm10. We can set these two files' `Reference` column value to 'mm9\_10', which will rename their reference names into mm9\_10 and the aggregated matrix will contain all genes from either mm9 or mm10. This renaming feature does not work if one HDF5 file contain multiple references (e.g. mm10 and GRCh38). `csv_file` can optionally contain two columns - `nUMI` and `nGene`. These two columns define minimum number of UMIs and genes for cell selection for each sample. The values in these two columns overwrite the `--min-genes` and `--min-umis` arguments. See below for an example csv:

```
Sample,Source,Platform,Donor,Reference,Location
sample_1,bone_marrow,NextSeq,1,GRCh38,/my_dir/sample_1/raw_feature_bc_
↪ matrices.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,/my_dir/sample_2/raw_feature_bc_
↪ matrices.h5
sample_3,pbmc,NextSeq,1,GRCh38,/my_dir/sample_3/raw_gene_bc_matrices_h5.h5
sample_4,pbmc,NextSeq,2,GRCh38,/my_dir/sample_4/raw_gene_bc_matrices_h5.h5
```

**output\_name**

The output file name.

- Options:

**--restriction <restriction>...**

Select channels that satisfy all restrictions. Each restriction takes the format of `name:value,...,value` or `name:~value,...,value`, where `~` refers to not. You can specify multiple restrictions by setting this option multiple times.



**--attributes <attributes>**

Specify a comma-separated list of outputted attributes. These attributes should be column names in the csv file.

**--default-reference <reference>**

If sample count matrix is in either DGE, mtx, csv, tsv or loom format and there is no Reference column in the csv\_file, use <reference> as the reference.

**--select-only-singlets**

If we have demultiplexed data, turning on this option will make pegasus only include barcodes that are predicted as singlets.

**--remap-singlets <remap\_string>**

Remap singlet names using <remap\_string>, where <remap\_string> takes the format “new\_name\_i:old\_name\_1,old\_name\_2;new\_name\_ii:old\_name\_3;...”. For example, if we hashed 5 libraries from 3 samples sample1\_lib1, sample1\_lib2, sample2\_lib1, sample2\_lib2 and sample3, we can remap them to 3 samples using this string: “sample1:sample1\_lib1,sample1\_lib2;sample2:sample2\_lib1,sample2\_lib2”. In this way, the new singlet names will be in metadata field with key ‘assignment’, while the old names will be kept in metadata field with key ‘assignment.orig’.

**--subset-singlets <subset\_string>**

If select singlets, only select singlets in the <subset\_string>, which takes the format “name1,name2,...”. Note that if --remap-singlets is specified, subsetting happens after remapping. For example, we can only select singlets from sample 1 and 3 using “sample1,sample3”.

**--min-genes <number>**

Only keep barcodes with at least <ngene> expressed genes.

**--max-genes <number>**

Only keep cells with less than <number> of genes.

**--min-umis <number>**

Only keep cells with at least <number> of UMIs.

**--max-umis <number>**

Only keep cells with less than <number> of UMIs.

**--mito-prefix <prefix>**

Prefix for mitochondrial genes. If multiple prefixes are provided, separate them by comma (e.g. “MT-,mt-”).

**--percent-mito <percent>**

Only keep cells with mitochondrial percent less than <percent>%. Only when both mito\_prefix and percent\_mito set, the mitochondrial filter will be triggered.

**--no-append-sample-name**

Turn this option on if you do not want to append sample name in front of each sample’s barcode (concatenated using ‘-’).

**-h, --help**

Print out help information.

- Outputs:

**output\_name.zarr.zip**

A zipped Zarr file containing aggregated data.

- Examples:

```
pegasus aggregate_matrix --restriction Source:BM,CB --restriction Individual:1-8 --  
↪attributes Source,Platform Manton_count_matrix.csv aggr_data
```

---

## pegasus demuxEM

Demultiplex cell-hashing/nucleus-hashing data.

Type:

```
pegasus demuxEM -h
```

to see the usage information:

```
Usage:  
    pegasus demuxEM [options] <input_raw_gene_bc_matrices_h5> <input_hto_csv_file>  
↪<output_name>  
    pegasus demuxEM -h | --help  
    pegasus demuxEM -v | --version
```

- Arguments:

- input\_raw\_gene\_bc\_matrices\_h5**

- Input raw RNA expression matrix in 10x hdf5 format. It is important to feed raw (unfiltered) count matrix, as demuxEM uses it to estimate the background information.

- input\_hto\_csv\_file**

- Input HTO (antibody tag) count matrix in CSV format.

- output\_name**

- Output name. All outputs will use it as the prefix.

- Options:

- p <number>, --threads <number>**

- Number of threads. [default: 1]

- genome <genome>**

- Reference genome name. If not provided, we will infer it from the expression matrix file.

- alpha-on-samples <alpha>**

- The Dirichlet prior concentration parameter (alpha) on samples. An alpha value < 1.0 will make the prior sparse. [default: 0.0]

- min-num-genes <number>**

- We only demultiplex cells/nuclei with at least <number> of expressed genes. [default: 100]

- min-num-umis <number>**

- We only demultiplex cells/nuclei with at least <number> of UMIs. [default: 100]

- min-signal-hashtag <count>**

- Any cell/nucleus with less than <count> hashtags from the signal will be marked as unknown. [default: 10.0]

- random-state <seed>**

- The random seed used in the KMeans algorithm to separate empty ADT droplets from others. [default: 0]

**--generate-diagnostic-plots**

Generate a series of diagnostic plots, including the background/signal between HTO counts, estimated background probabilities, HTO distributions of cells and non-cells etc.

**--generate-gender-plot <genes>**

Generate violin plots using gender-specific genes (e.g. Xist). <gene> is a comma-separated list of gene names.

**-v, --version**

Show DemuxEM version.

**-h, --help**

Print out help information.

- Outputs:

**output\_name\_demux.zarr.zip**

RNA expression matrix with demultiplexed sample identities in Zarr format.

**output\_name.out.demuxEM.zarr.zip**

DemuxEM-calculated results in Zarr format, containing two datasets, one for HTO and one for RNA.

**output\_name.ambient\_hashtag.hist.pdf**

Optional output. A histogram plot depicting hashtag distributions of empty droplets and non-empty droplets.

**output\_name.background\_probabilities.bar.pdf**

Optional output. A bar plot visualizing the estimated hashtag background probability distribution.

**output\_name.real\_content.hist.pdf**

Optional output. A histogram plot depicting hashtag distributions of not-real-cells and real-cells as defined by total number of expressed genes in the RNA assay.

**output\_name.rna\_demux.hist.pdf**

Optional output. A histogram plot depicting RNA UMI distribution for singlets, doublets and unknown cells.

**output\_name.gene\_name.violin.pdf**

Optional outputs. Violin plots depicting gender-specific gene expression across samples. We can have multiple plots if a gene list is provided in ‘--generate-gender-plot’ option.

- Examples:

```
pegasus demuxEM -p 8 --generate-diagnostic-plots sample_raw_gene_bc_matrices.h5 ↵  
↵sample_hto.csv sample_output
```

## pegasus cluster

Once we collected the count matrix in 10x (`example_10x.h5`) or Zarr (`example.zarr.zip`) format, we can perform single cell analysis using `pegasus cluster`.

Type:

```
pegasus cluster -h
```

to see the usage information:

Usage:

```
pegasus cluster [options] <input_file> <output_name>
pegasus cluster -h
```

- Arguments:

- input\_file**

- Input file in either ‘zarr’, ‘h5ad’, ‘loom’, ‘10x’, ‘mtx’, ‘csv’, ‘tsv’ or ‘fcs’ format. If first-pass analysis has been performed, but you want to run some additional analysis, you could also pass a zarr-formatted file.

- output\_name**

- Output file name. All outputs will use it as the prefix.

- Options:

- p <number>, --threads <number>**

- Number of threads. [default: 1]

- processed**

- Input file is processed. Assume quality control, data normalization and log transformation, highly variable gene selection, batch correction/PCA and kNN graph building is done.

- channel <channel\_attr>**

- Use <channel\_attr> to create a ‘Channel’ column metadata field. All cells within a channel are assumed to come from a same batch.

- black-list <black\_list>**

- Cell barcode attributes in black list will be popped out. Format is “attr1,attr2,...,attrn”.

- select-singlets**

- Only select DemuxEM-predicted singlets for analysis.

- remap-singlets <remap\_string>**

- Remap singlet names using <remap\_string>, where <remap\_string> takes the format “new\_name\_i:old\_name\_1,old\_name\_2;new\_name\_ii:old\_name\_3;...”. For example, if we hashed 5 libraries from 3 samples `sample1_lib1`, `sample1_lib2`, `sample2_lib1`, `sample2_lib2` and `sample3`, we can remap them to 3 samples using this string: “`sample1:sample1_lib1,sample1_lib2;sample2:sample2_lib1,sample2_lib2`”. In this way, the new singlet names will be in metadata field with key ‘assignment’, while the old names will be kept in metadata field with key ‘assignment.orig’.

- subset-singlets <subset\_string>**

- If select singlets, only select singlets in the <subset\_string>, which takes the format “name1,name2,...”. Note that if `--remap-singlets` is specified, subsetting happens after remapping. For example, we can only select singlets from sample 1 and 3 using “`sample1,sample3`”.

**--genome <genome\_name>**

If sample count matrix is in either DGE, mtx, csv, tsv or loom format, use <genome\_name> as the genome reference name.

**--focus <keys>**

Focus analysis on Unimodal data with <keys>. <keys> is a comma-separated list of keys. If None, the self.\_selected will be the focused one.

**--append <key>**

Append Unimodal data <key> to any <keys> in --focus.

**--output-loom**

Output loom-formatted file.

**--output-h5ad**

Output h5ad-formatted file.

**--min-genes <number>**

Only keep cells with at least <number> of genes. [default: 500]

**--max-genes <number>**

Only keep cells with less than <number> of genes. [default: 6000]

**--min-umis <number>**

Only keep cells with at least <number> of UMIs.

**--max-umis <number>**

Only keep cells with less than <number> of UMIs.

**--mito-prefix <prefix>**

Prefix for mitochondrial genes. Can provide multiple prefixes for multiple organisms (e.g. "MT-" means to use "MT-", "GRCh38:MT-,mm10:mt-,MT-" means to use "MT-" for GRCh38, "mt-" for mm10 and "MT-" for all other organisms). [default: GRCh38:MT-,mm10:mt-,MT-]

**--percent-mito <ratio>**

Only keep cells with mitochondrial percent less than <percent>%. [default: 20.0]

**--gene-percent-cells <ratio>**

Only use genes that are expressed in at least <percent>% of cells to select variable genes. [default: 0.05]

**--output-filtration-results**

Output filtration results as a spreadsheet.

**--plot-filtration-results**

Plot filtration results as PDF files.

**--plot-filtration-figsize <figsize>**

Figure size for filtration plots. <figsize> is a comma-separated list of two numbers, the width and height of the figure (e.g. 6,4).

**--min-genes-before-filtration <number>**

If raw data matrix is input, empty barcodes will dominate pre-filtration statistics. To avoid this, for raw data matrix, only consider barcodes with at least <number> genes for pre-filtration condition. [default: 100]

**--counts-per-cell-after <number>**

Total counts per cell after normalization. [default: 1e5]

**--select-hvf-flavor <flavor>**

Highly variable feature selection method. <flavor> can be 'pegasus' or 'Seurat'. [default: pegasus]

**--select-hvf-ngenes <nfeatures>**

Select top <nfeatures> highly variable features. If <flavor> is 'Seurat' and <ngenes> is 'None', select HVGs with z-score cutoff at 0.5. [default: 2000]

**--no-select-hvf**

Do not select highly variable features.

**--plot-hvf**

Plot highly variable feature selection.

**--correct-batch-effect**

Correct for batch effects.

**--correction-method <method>**

Batch correction method, can be either 'L/S' for location/scale adjustment algorithm (Li and Wong. The analysis of Gene Expression Data 2003), 'harmony' for Harmony (Korsunsky et al. Nature Methods 2019), 'scanorama' for Scanorama (Hie et al. Nature Biotechnology 2019) or 'inmf' for integrative NMF (Yang and Michailidis Bioinformatics 2016, Welch et al. Cell 2019, Gao et al. Nature Biotechnology 2021) [default: harmony]

**--batch-group-by <expression>**

Batch correction assumes the differences in gene expression between channels are due to batch effects. However, in many cases, we know that channels can be partitioned into several groups and each group is biologically different from others. In this case, we will only perform batch correction for channels within each group. This option defines the groups. If <expression> is None, we assume all channels are from one group. Otherwise, groups are defined according to <expression>. <expression> takes the form of either 'attr', or 'attr1+attr2+...+attrn', or 'attr=value11,...,value1n\_1;value21,...,value2n\_2;...;valuem1,...,valuemn\_m'. In the first form, 'attr' should be an existing sample attribute, and groups are defined by 'attr'. In the second form, 'attr1',..., 'attrn' are n existing sample attributes and groups are defined by the Cartesian product of these n attributes. In the last form, there will be m + 1 groups. A cell belongs to group i (i > 0) if and only if its sample attribute 'attr' has a value among valuei1,...,valuein\_i. A cell belongs to group 0 if it does not belong to any other groups.

**--harmony-nclusters <nclusters>**

Number of clusters used for Harmony batch correction.

**--inmf-lambda <lambda>**

Coefficient of regularization for iNMF. [default: 5.0]

**--random-state <seed>**

Random number generator seed. [default: 0]

**--temp-folder <temp\_folder>**

Joblib temporary folder for memmapping numpy arrays.

**--calc-signature-scores <sig\_list>**

Calculate signature scores for gene sets in <sig\_list>. <sig\_list> is a comma-separated list of strings. Each string should either be a <GMT\_file> or one of 'cell\_cycle\_human', 'cell\_cycle\_mouse', 'gender\_human', 'gender\_mouse', 'mitochondrial\_genes\_human', 'mitochondrial\_genes\_mouse', 'ribosomal\_genes\_human' and 'ribosomal\_genes\_mouse'.

**--pca-n <number>**

Number of principal components. [default: 50]

**--nmf**

Compute nonnegative matrix factorization (NMF) on highly variable features.

**--nmf-n <number>**

Number of NMF components. IF iNMF is used for batch correction, this parameter also sets

iNMF number of components. [default: 20]

**--knn-K <number>**

Number of nearest neighbors for building kNN graph. [default: 100]

**--knn-full-speed**

For the sake of reproducibility, we only run one thread for building kNN indices. Turn on this option will allow multiple threads to be used for index building. However, it will also reduce reproducibility due to the racing between multiple threads.

**--kBET**

Calculate kBET.

**--kBET-batch <batch>**

kBET batch keyword.

**--kBET-alpha <alpha>**

kBET rejection alpha. [default: 0.05]

**--kBET-K <K>**

kBET K. [default: 25]

**--diffmap**

Calculate diffusion maps.

**--diffmap-ndc <number>**

Number of diffusion components. [default: 100]

**--diffmap-solver <solver>**

Solver for eigen decomposition, either 'randomized' or 'eigsh'. [default: eigsh]

**--diffmap-maxt <max\_t>**

Maximum time stamp to search for the knee point. [default: 5000]

**--calculate-pseudotime <roots>**

Calculate diffusion-based pseudotimes based on <roots>. <roots> should be a comma-separated list of cell barcodes.

**--louvain**

Run louvain clustering algorithm.

**--louvain-resolution <resolution>**

Resolution parameter for the louvain clustering algorithm. [default: 1.3]

**--louvain-class-label <label>**

Louvain cluster label name in result. [default: louvain\_labels]

**--leiden**

Run leiden clustering algorithm.

**--leiden-resolution <resolution>**

Resolution parameter for the leiden clustering algorithm. [default: 1.3]

**--leiden-niter <niter>**

Number of iterations of running the Leiden algorithm. If <niter> is negative, run Leiden iteratively until no improvement. [default: -1]

**--leiden-class-label <label>**

Leiden cluster label name in result. [default: leiden\_labels]

**--spectral-louvain**

Run spectral-louvain clustering algorithm.

- spectral-louvain-basis <basis>**  
Basis used for KMeans clustering. Can be 'pca' or 'diffmap'. If 'diffmap' is not calculated, use 'pca' instead. [default: diffmap]
- spectral-louvain-nclusters <number>**  
Number of first level clusters for Kmeans. [default: 30]
- spectral-louvain-nclusters2 <number>**  
Number of second level clusters for Kmeans. [default: 50]
- spectral-louvain-ninit <number>**  
Number of Kmeans tries for first level clustering. Default is the same as scikit-learn Kmeans function. [default: 10]
- spectral-louvain-resolution <resolution>.**  
Resolution parameter for louvain. [default: 1.3]
- spectral-louvain-class-label <label>**  
Spectral-louvain label name in result. [default: spectral\_louvain\_labels]
- spectral-leiden**  
Run spectral-leiden clustering algorithm.
- spectral-leiden-basis <basis>**  
Basis used for KMeans clustering. Can be 'pca' or 'diffmap'. If 'diffmap' is not calculated, use 'pca' instead. [default: diffmap]
- spectral-leiden-nclusters <number>**  
Number of first level clusters for Kmeans. [default: 30]
- spectral-leiden-nclusters2 <number>**  
Number of second level clusters for Kmeans. [default: 50]
- spectral-leiden-ninit <number>**  
Number of Kmeans tries for first level clustering. Default is the same as scikit-learn Kmeans function. [default: 10]
- spectral-leiden-resolution <resolution>**  
Resolution parameter for leiden. [default: 1.3]
- spectral-leiden-class-label <label>**  
Spectral-leiden label name in result. [default: spectral\_leiden\_labels]
- tsne**  
Run Flt-SNE package to compute t-SNE embeddings for visualization.
- tsne-perplexity <perplexity>**  
t-SNE's perplexity parameter. [default: 30]
- tsne-initialization <choice>**  
<choice> can be either 'random' or 'pca'. 'random' refers to random initialization. 'pca' refers to PCA initialization as described in (CITE Kobak et al. 2019) [default: pca]
- umap**  
Run umap for visualization.
- umap-K <K>**  
K neighbors for umap. [default: 15]
- umap-min-dist <number>**  
Umap parameter. [default: 0.5]



- umap-spread <spread>**  
Umap parameter. [default: 1.0]
- fle**  
Run force-directed layout embedding.
- fle-K <K>**  
K neighbors for building graph for FLE. [default: 50]
- fle-target-change-per-node <change>**  
Target change per node to stop forceAtlas2. [default: 2.0]
- fle-target-steps <steps>**  
Maximum number of iterations before stopping the forceAtlas2 algorithm. [default: 5000]
- fle-memory <memory>**  
Memory size in GB for the Java FA2 component. [default: 8]
- net-down-sample-fraction <frac>**  
Down sampling fraction for net-related visualization. [default: 0.1]
- net-down-sample-K <K>**  
Use <K> neighbors to estimate local density for each data point for down sampling. [default: 25]
- net-down-sample-alpha <alpha>**  
Weighted down sample, proportional to  $\text{radius}^\alpha$ . [default: 1.0]
- net-regressor-L2-penalty <value>**  
L2 penalty parameter for the deep net regressor. [default: 0.1]
- net-umap**  
Run net umap for visualization.
- net-umap-polish-learning-rate <rate>**  
After running the deep regressor to predict new coordinate, what is the learning rate to use to polish the coordinates for UMAP. [default: 1.0]
- net-umap-polish-nepochs <nepochs>**  
Number of iterations for polishing UMAP run. [default: 40]
- net-umap-out-basis <basis>**  
Output basis for net-UMAP. [default: net\_umap]
- net-fle**  
Run net FLE.
- net-fle-polish-target-steps <steps>**  
After running the deep regressor to predict new coordinate, what is the number of force atlas 2 iterations. [default: 1500]
- net-fle-out-basis <basis>**  
Output basis for net-FLE. [default: net\_fle]
- infer-doublets**  
Infer doublets using the method described [here](#). Obs attribute 'doublet\_score' stores Scrublet-like doublet scores and attribute 'demux\_type' stores 'doublet/singlet' assignments.
- expected-doublet-rate <rate>**  
The expected doublet rate per sample. By default, calculate the expected rate based on number of cells from the 10x multiplet rate table.

**--dbl-cluster-attr <attr>**

<attr> refers to a cluster attribute containing cluster labels (e.g. 'louvain\_labels'). Doublet clusters will be marked based on <attr> with the following criteria: passing the Fisher's exact test and having  $\geq 50\%$  of cells identified as doublets. By default, the first computed cluster attribute in the list of leiden, louvain, spectral\_leiden and spectral\_louvain is used.

**--citeseq**

Input data contain both RNA and CITE-Seq modalities. This will set `--focus` to be the RNA modality and `--append` to be the CITE-Seq modality. In addition, 'ADT-' will be added in front of each antibody name to avoid name conflict with genes in the RNA modality.

**--citeseq-umap**

For high quality cells kept in the RNA modality, generate a UMAP based on their antibody expression.

**--citeseq-umap-exclude <list>**

<list> is a comma-separated list of antibodies to be excluded from the UMAP calculation (e.g. Mouse-IgG1, Mouse-IgG2a).

**-h, --help**

Print out help information.

- Outputs:

**output\_name.zarr.zip**

Output file in Zarr format. To load this file in python, use `import pegasus; data = pegasus.read_input('output_name.zarr.zip')`. The log-normalized expression matrix is stored in `data.X` as a CSR-format sparse matrix. The `obs` field contains cell related attributes, including clustering results. For example, `data.obs_names` records cell barcodes; `data.obs['Channel']` records the channel each cell comes from; `data.obs['n_genes']`, `data.obs['n_counts']`, and `data.obs['percent_mito']` record the number of expressed genes, total UMI count, and mitochondrial rate for each cell respectively; `data.obs['louvain_labels']` and `data.obs['approx_louvain_labels']` record each cell's cluster labels using different clustering algorithms; `data.obs['pseudo_time']` records the inferred pseudotime for each cell. The `var` field contains gene related attributes. For example, `data.var_names` records gene symbols, `data.var['gene_ids']` records Ensembl gene IDs, and `data.var['selected']` records selected variable genes. The `obsnm` field records embedding coordinates. For example, `data.obsm['X_pca']` records PCA coordinates, `data.obsm['X_tsne']` records tSNE coordinates, `data.obsm['X_umap']` records UMAP coordinates, `data.obsm['X_diffmap']` records diffusion map coordinates, and `data.obsm['X_fle']` records the force-directed layout coordinates from the diffusion components. The `uns` field stores other related information, such as reference genome (`data.uns['genome']`). This file can be loaded into R and converted into a Seurat object.

**output\_name.<group>.h5ad**

Optional output. Only exists if `--output-h5ad` is set. Results in h5ad format per focused <group>. This file can be loaded into R and converted into a Seurat object.

**output\_name.<group>.loom**

Optional output. Only exists if `--output-loom` is set. Results in loom format per focused <group>.

**output\_name.<group>.filt.xlsx**

Optional output. Only exists if `--output-filtration-results` is set. Filtration statistics per focused <group>. This file has two sheets — Cell filtration stats and Gene filtration stats. The first sheet records cell filtering results and it has 10 columns: Channel, channel name; kept, number of cells kept; median\_n\_genes, median number of expressed genes in kept cells; median\_n\_umis, median number of UMIs in kept cells; median\_percent\_mito, median mitochondrial rate as UMIs

between mitochondrial genes and all genes in kept cells; filt, number of cells filtered out; total, total number of cells before filtration, if the input contain all barcodes, this number is the cells left after ‘-min-genes-on-row’ filtration; median\_n\_genes\_before, median expressed genes per cell before filtration; median\_n\_umis\_before, median UMIs per cell before filtration; median\_percent\_mito\_before, median mitochondrial rate per cell before filtration. The channels are sorted in ascending order with respect to the number of kept cells per channel. The second sheet records genes that failed to pass the filtering. This sheet has 3 columns: gene, gene name; n\_cells, number of cells this gene is expressed; percent\_cells, the fraction of cells this gene is expressed. Genes are ranked in ascending order according to number of cells the gene is expressed. Note that only genes not expressed in any cell are removed from the data. Other filtered genes are marked as non-robust and not used for TPM-like normalization.

**output\_name.<group>.filt.gene.pdf**

Optional output. Only exists if ‘-plot-filtration-results’ is set. This file contains violin plots contrasting gene count distributions before and after filtration per channel per focused <group>.

**output\_name.<group>.filt.UMI.pdf**

Optional output. Only exists if ‘-plot-filtration-results’ is set. This file contains violin plots contrasting UMI count distributions before and after filtration per channel per focused <group>.

**output\_name.<group>.filt.mito.pdf**

Optional output. Only exists if ‘-plot-filtration-results’ is set. This file contains violin plots contrasting mitochondrial rate distributions before and after filtration per channel per focused <group>.

**output\_name.<group>.hvf.pdf**

Optional output. Only exists if ‘-plot-hvf’ is set. This file contains a scatter plot describing the highly variable gene selection procedure per focused <group>.

**output\_name.<group>.<channel>.dbl.png**

Optional output. Only exists if ‘-infer-doublers’ is set. Each figure consists of 4 panels showing diagnostic plots for doublet inference. If there is only one channel in <group>, file name becomes output\_name.<group>.dbl.png.

- Examples:

```
pegasus cluster -p 20 --correct-batch-effect --louvain --tsne example_10x.h5
↪ example_out
pegasus cluster -p 20 --leiden --umap --net-file example.zarr.zip example_out
```

## pegasus de\_analysis

Once we have the clusters, we can detect markers using `pegasus de_analysis`. We will calculate Mann-Whitney U test and AUROC values by default.

Type:

```
pegasus de_analysis -h
```

to see the usage information:

```
Usage:
    pegasus de_analysis [options] (--labels <attr>) <input_data_file> <output_
↪ spreadsheet>
    pegasus de_analysis -h
```

- Arguments:

- input\_data\_file**

- Single cell data with clustering calculated. DE results would be written back.

- output\_spreadsheet**

- Output spreadsheet with DE results.

- Options:

- labels <attr>**

- <attr> used as cluster labels. [default: louvain\_labels]

- p <threads>**

- Use <threads> threads. [default: 1]

- de-key <key>**

- Store DE results into AnnData varm with key = <key>. [default: de\_res]

- t**

- Calculate Welch's t-test.

- fisher**

- Calculate Fisher's exact test.

- temp-folder <temp\_folder>**

- Joblib temporary folder for memmapping numpy arrays.

- alpha <alpha>**

- Control false discovery rate at <alpha>. [default: 0.05]

- ndigits <ndigits>**

- Round non p-values and q-values to <ndigits> after decimal point in the excel. [default: 3]

- quiet**

- Do not show detailed intermediate outputs.

- h, --help**

- Print out help information.

- Outputs:

- input\_data\_file**

- DE results would be written back to the 'varm' field with name set by '--de-key <key>'.

- output\_spreadsheet**

- An excel spreadsheet containing DE results. Each cluster has two tabs in the spreadsheet. One is for up-regulated genes and the other is for down-regulated genes. If DE was performed on conditions within each cluster. Each cluster will have number of conditions tabs and each condition tab contains two spreadsheet: up for up-regulated genes and down for down-regulated genes.

- Examples:

```
pegasus de_analysis -p 26 --labels louvain_labels --t --fisher example.zarr.zip ↵  
↵ example_de.xlsx
```

## pegasus find\_markers

Once we have the DE results, we can optionally find cluster-specific markers with gradient boosting using `pegasus find_markers`.

Type:

```
pegasus find_markers -h
```

to see the usage information:

Usage:

```
pegasus find_markers [options] <input_data_file> <output_spreadsheet>
pegasus find_markers -h
```

- Arguments:

- input\_h5ad\_file**

- Single cell data after running the `de_analysis`.

- output\_spreadsheet**

- Output spreadsheet with LightGBM detected markers.

- Options:

- p <threads>**

- Use <threads> threads. [default: 1]

- labels <attr>**

- <attr> used as cluster labels. [default: `louvain_labels`]

- de-key <key>**

- Key for storing DE results in 'varm' field. [default: `de_res`]

- remove-ribo**

- Remove ribosomal genes with either RPL or RPS as prefixes.

- min-gain <gain>**

- Only report genes with a feature importance score (in gain) of at least <gain>. [default: 1.0]

- random-state <seed>**

- Random state for initializing LightGBM and KMeans. [default: 0]

- h, --help**

- Print out help information.

- Outputs:

- output\_spreadsheet**

- An excel spreadsheet containing detected markers. Each cluster has one tab in the spreadsheet and each tab has six columns, listing markers that are strongly up-regulated, weakly up-regulated, down-regulated and their associated LightGBM gains.

- Examples:

```
pegasus find_markers --labels louvain_labels --remove-ribo --min-gain 10.0 -p 10.0
↪ example.zarr.zip example.markers.xlsx
```

## pegasus annotate\_cluster

Once we have the DE results, we could optionally identify putative cell types for each cluster using `pegasus annotate_cluster`. This command has two forms: the first form generates putative annotations, and the second form write annotations into the Zarr object.

Type:

```
pegasus annotate_cluster -h
```

to see the usage information:

Usage:

```
pegasus annotate_cluster [--marker-file <file> --de-test <test> --de-alpha  
↪<alpha> --de-key <key> --minimum-report-score <score> --do-not-use-non-de-genes]  
↪<input_data_file> <output_file>  
pegasus annotate_cluster --annotation <annotation_string> <input_data_file>  
pegasus annotate_cluster -h
```

- Arguments:

- input\_data\_file**

- Single cell data with DE analysis done by `pegasus de_analysis`.

- output\_file**

- Output annotation file.

- Options:

- markers <str>**

- `<str>` is a comma-separated list. Each element in the list either refers to a JSON file containing legacy markers, or 'human\_immune'/'mouse\_immune'/'human\_brain'/'mouse\_brain'/'human\_lung' for predefined markers. [default: human\_immune]

- de-test <test>**

- DE test to use to infer cell types. [default: mwu]

- de-alpha <alpha>**

- False discovery rate to control family-wise error rate. [default: 0.05]

- de-key <key>**

- Keyword where the DE results store in 'varm' field. [default: de\_res]

- minimum-report-score <score>**

- Minimum cell type score to report a potential cell type. [default: 0.5]

- do-not-use-non-de-genes**

- Do not count non DE genes as down-regulated.

- annotation <annotation\_string>**

- Write cell type annotations in `<annotation_string>` into `<input_data_file>`. `<annotation_string>` has this format: 'anno\_name:clust\_name:anno\_1;anno\_2;...;anno\_n', where `anno_name` is the annotation attribute in the Zarr object, `clust_name` is the attribute with cluster ids, and `anno_i` is the annotation for cluster `i`.

- h, --help**

- Print out help information.

- Outputs:

**output\_file**

This is a text file. For each cluster, all its putative cell types are listed in descending order of the cell type score. For each putative cell type, all markers support this cell type are listed. If one putative cell type has cell subtypes, all subtypes will be listed under this cell type.

- Examples:

```
pegasus annotate_cluster example.zarr.zip example.anno.txt
pegasus annotate_cluster --markers human_immune,human_lung lung.zarr.zip lung.anno.
↪txt
pegasus annotate_cluster --annotation "anno:louvain_labels:T cells;B cells;NK cells;
↪Monocytes" example.zarr.zip
```

**pegasus plot**

We can make a variety of figures using `pegasus plot`.

Type:

```
pegasus plot -h
```

to see the usage information:

```
Usage:
    pegasus plot [options] [--restriction <restriction>...] [--palette <palette>...]
↪<plot_type> <input_file> <output_file>
    pegasus plot -h
```

- Arguments:

**plot\_type**

Plot type, either 'scatter' for scatter plots, 'compo' for composition plots, or 'wordcloud' for word cloud plots.

**input\_file**

Single cell data in Zarr or H5ad format.

**output\_file**

Output image file.

- Options:

**--dpi <dpi>**

DPI value for the figure. [default: 500]

**--basis <basis>**

Basis for 2D plotting, chosen from 'tsne', 'fitsne', 'umap', 'pca', 'fle', 'net\_tsne', 'net\_umap' or 'net\_fle'. [default: umap]

**--attributes <attrs>**

<attrs> is a comma-separated list of attributes to color the basis. This option is only used in 'scatter'.

**--restriction <restriction>...**

Set restriction if you only want to plot a subset of data. Multiple <restriction> strings are allowed. Each <restriction> takes the format of 'attr:value,value', or 'attr:~value,value..' which means excluding values. This option is used in 'composition' and 'scatter'.

**--alpha <alpha>**

Point transparent parameter. Can be a single value or a list of values separated by comma used for each attribute in <attrs>.

**--legend-loc <str>**

Legend location, can be either “right margin” or “on data”. If a list is provided, set ‘legend\_loc’ for each attribute in ‘attrs’ separately. [default: “right margin”]

**--palette <str>**

Used for setting colors for every categories in categorical attributes. Multiple <palette> strings are allowed. Each string takes the format of ‘attr:color1,color2,...,colorn’. ‘attr’ is the categorical attribute and ‘color1’ - ‘colorn’ are the colors for each category in ‘attr’ (e.g. ‘cluster\_labels:black,blue,red,...,yellow’). If there is only one categorical attribute in ‘attrs’, **palettes** can be set as a single string and the ‘attr’ keyword can be omitted (e.g. “blue,yellow,red”).

**--show-background**

Show points that are not selected as gray.

**--nrows <nrows>**

Number of rows in the figure. If not set, pegasus will figure it out automatically.

**--ncols <ncols>**

Number of columns in the figure. If not set, pegasus will figure it out automatically.

**--panel-size <sizes>**

Panel size in inches, w x h, separated by comma. Note that margins are not counted in the sizes. For composition, default is (6, 4). For scatter plots, default is (4, 4).

**--left <left>**

Figure’s left margin in fraction with respect to panel width.

**--bottom <bottom>**

Figure’s bottom margin in fraction with respect to panel height.

**--wspace <wspace>**

Horizontal space between panels in fraction with respect to panel width.

**--hspace <hspace>**

Vertical space between panels in fraction with respect to panel height.

**--groupby <attr>**

Use <attr> to categorize the cells for the composition plot, e.g. cell type.

**--condition <attr>**

Use <attr> to calculate frequency within each category defined by ‘--groupby’ for the composition plot, e.g. donor.

**--style <style>**

Composition plot styles. Can be either ‘frequency’ or ‘normalized’. [default: normalized]

**--factor <factor>**

Factor index (column index in data.uns[‘W’]) to be used to generate word cloud plot.

**--max-words <max\_words>**

Maximum number of genes to show in the image. [default: 20]

**-h, --help**

Print out help information.

Examples:



```

pegasus plot scatter --basis tsne --attributes louvain_labels,Donor example.h5ad scatter.
↪pdf
pegasus plot compo --groupby louvain_labels --condition Donor example.zarr.zip compo.pdf
pegasus plot wordcloud --factor 0 example.zarr.zip word_cloud_0.pdf

```

## pegasus scp\_output

If we want to visualize analysis results on [single cell portal](#) (SCP), we can generate required files for SCP using this subcommand.

Type:

```
pegasus scp_output -h
```

to see the usage information:

```

Usage:
    pegasus scp_output <input_data_file> <output_name>
    pegasus scp_output -h

```

- Arguments:
  - input\_data\_file**  
Analyzed single cell data in zarr format.
  - output\_name**  
Name prefix for all outputted files.
- Options:
  - dense**  
Output dense expression matrix instead.
  - round-to <ndigit>**  
Round expression to <ndigit> after the decimal point. [default: 2]
  - h, --help**  
Print out help information.
- Outputs:
  - output\_name.scp.metadata.txt, output\_name.scp.barcodes.tsv, output\_name.scp.genes.tsv, output\_name.scp.matrix.mtx, output\_name.scp.\*.coords.txt, output\_name.scp.expr.txt**  
Files that single cell portal needs.
- Examples:

```
pegasus scp_output example.zarr.zip example
```

## pegasus check\_indexes

If we run CITE-Seq or any kind of hashing, we need to make sure that the library indexes of CITE-Seq/hashing do not collide with 10x's RNA indexes. This command can help us to determine which 10x index sets we should use.

Type:

```
pegasus check_indexes -h
```

to see the usage information:

```
Usage:
  pegasus check_indexes [--num-mismatch <mismatch> --num-report <report>] <index_
↵file>
  pegasus check_indexes -h
```

- Arguments:

- index\_file**

- Index file containing CITE-Seq/hashing index sequences. One sequence per line.

- Options:

- num-mismatch <mismatch>**

- Number of mismatch allowed for each index sequence. [default: 1]

- num-report <report>**

- Number of valid 10x indexes to report. Default is to report all valid indexes. [default: 9999]

- h, --help**

- Print out help information.

- Outputs:

- Up to <report> number of valid 10x indexes will be printed out to standard output.

- Examples:

```
pegasus check_indexes --num-report 8 index_file.txt
```

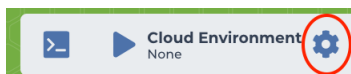
## 1.2.3 Use Pegasus on Terra Notebook

You need to first have a [Terra](#) account.

### 1. Start Notebook Runtime on Terra

The first time when you use Terra notebook, you need to create a Cloud Environment.

On the top-right panel of your workspace, click the following button within red circle:



Then click the CUSTOMIZE button shown in red rectangle:

### Cloud Environment

A cloud environment consists of application configuration, cloud compute and persistent disk(s).

#### Use default environment

[What's installed on this environment?](#)

- Default: (GATK 4.1.4.1, Python 3.7.10, R 4.0.5)
- Default compute size of **1 CPU(s)**, **3.75 GB memory**, and your existing **50 GB persistent disk**
- [Learn more about Persistent disks and where your disk is mounted](#)

Running cloud compute cost	Paused cloud compute cost	Persistent disk cost
<b>\$0.06</b> per hr	<b>&lt; \$0.01</b> per hr	<b>\$2.00</b> per month

#### Create custom environment

DELETE PERSISTENT DISK

Then you'll need to set the configuration of your cloud environment in the pop-out dialog (see image below):

Cloud Environment ×

A cloud environment consists of application configuration, cloud compute and persistent disk(s).

Running cloud compute cost  
\$0.06 per hr

Paused cloud compute cost  
< \$0.01 per hr

Persistent disk cost  
\$2.00 per month

Application configuration ⓘ  
Default: (GATK 4.2.0.0, Python 3.7.10, R 4.1.0) ▼  
What's installed on this environment? Updated: Jul 19, 2021  
Version: 2.0.1

Cloud compute profile

CPU1 ▼

Memory (GB) 3.75 ▼

☐ Enable GPUs BETA [Learn more about GPU cost and restrictions.](#)

Startup script  
URI

Compute type  
Standard VM ▼

Persistent disk size (GB)

Persistent disks store analysis data. [Learn more about persistent disks and where your disk is mounted.](#)

50 ▼

CREATE

## 1.1. Create from Terra official environment

Terra team maintains a list of cloud environments for users to quickly set up their own. In this way, you'll use the most recent stable Pegasus cloud environment.

In *Application configuration* field, select Pegasus from the drop-down menu:

**Application configuration** ⓘ

Default: (GATK 4.2.0.0, Python 3.7.10, R 4.1.0) ▼

Default: (GATK 4.2.0.0, Python 3.7.10, R 4.1.0) ✓

Legacy GATK (GATK 4.1.4.1, Python 3.7.8, R 4.0.2)

Legacy R / Bioconductor (R 4.0.5, Bioconductor 3.12, Python 3.8.5)

**COMMUNITY-MAINTAINED JUPYTER ENVIRONMENTS (VERIFIED PARTNERS)**

Pegasus (Pegasuspy 1.4.3, Python 3.7.10, harmony-pytorch 0.1.6, nmf-torch 0.1.1)

**COMMUNITY-MAINTAINED RSTUDIO ENVIRONMENTS (VERIFIED PARTNERS)**

RStudio (R 4.1.0, Bioconductor 3.13.0, Python 3.8.5)

**OTHER ENVIRONMENTS**

Custom Environment

In case you are interested in looking at which packages and tools are included in the Pegasus cloud environment, please click the [What's installed on this environment?](#) link right below the drop-down menu:

**Installed packages** ⏪ ⏩

Pegasus (Pegasuspy 1.4.3, Python 3.7.10, harmony-pytorch 0.1.6, nmf-torch 0.1.1) ▼

Updated: Jul 24, 2021  
Version: 1.4.3

**Language:** Python ▼

Package	Python	Version
adjustText	Tools	0.7.3
anndata		0.7.6
annoy		1.17.0
cirrocumulus		1.1.17.post1
conda		4.9.2
crcmod		1.7

After that set other fields in the pop-out dialog:

- In *Cloud compute profile* field, you can set the computing resources (CPUs and Memory size) you want to use.
- In *Compute type* input, choose **Standard VM**, as this is the cheapest type and is enough for using Pegasus.
- **(NEW)** In case you need GPUs, select *Enable GPUs*, then choose *GPU type* and type in number of *GPUs* to use. Notice that only version 1.4.3 or later supports this GPU mode.
- In *Persistent disk size (GB)* field, choose the size of the persistent disk for your cloud environment. This disk space by default remains after you delete your cloud environment (unless you choose to delete the persistent disk as well), but it costs even if you stop your cloud environment.

Now click the **CREATE** button to start the creation. After waiting for 1-2 minutes, your cloud environment will be ready to use, and it's started automatically.

## 1.2. Create from custom environment

Alternatively, you can create from a custom environment if you want to use an older version of Pegasus.

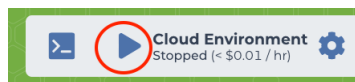
In the cloud environment setting page:

- In *Application configuration* field, choose **Custom Environment** (see the first red rectangle above).
- In *Container image* input, type `cumulusprod/pegasus-terra:<version>` (see the second red rectangle above), where `<version>` should be chosen from [this list](#). All the tags are for different versions of Pegasus.
- In *Cloud compute profile* field, set the computing resources (CPUs and Memory size) you want to use.
- In *Compute type* field, choose **Standard VM**, as this is the cheapest type and is enough for using Pegasus.
- In *Persistent disk size (GB)* field, choose the size of the persistent disk for your cloud environment.

When finishing the configuration, click **NEXT** button. You'll see a warning page, then click **CREATE** button to start the creation. After waiting for 1-2 minutes, your cloud environment will be ready to use, and it's started automatically.

## 1.3. Start an environment already created

After creation, this cloud environment is associated with your Terra workspace. You can start the same environment anytime in your workspace by clicking the following button within red circle on the top-right panel:



## 1. Create Your Terra Notebook

In the **NOTEBOOKS** tab of your workspace, you can either create a blank notebook, or upload your local notebook. After creation, click **EDIT** button to enter the edit mode, and Terra will automatically start your cloud environment.

When the start-up is done, you can type the following code in your notebook to check if Pegasus can be loaded and if it's the correct version you want to use:

```
import pegasus as pg
pg.__version__
```

## 3. Load Data into Cloud Environment

To use your data on Cloud (i.e. from the Google Bucket of your workspace), you should first copy it into your notebook's cloud environment by Google Cloud SDK:

```
!gsutil -m cp gs://link-to-count-matrix .
```

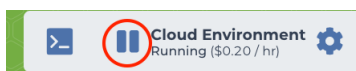
where `gs://link-to-count-matrix` is the Google Bucket URL to your count matrix data file, and `!` is the indicator of running terminal commands within Jupyter notebook.

After that, you can use Pegasus function to load it into memory.

Please refer to [tutorials](#) for how to use Pegasus on Terra notebook.

## 4. Stop Notebook Runtime

When you are done with the interactive analysis, to avoid being charged by Google Cloud while not using it, don't forget to stop your cloud environment by clicking the following button of the top-right panel of your workspace within red circle:



If you forget to stop manually, as far as you've closed all the webpages related to your cloud environment (e.g. Terra notebooks, Terra terminals, etc.), you'll still be safe. In this case, Terra will automatically stop the cloud environment for you after waiting for a few minutes.

### 1.2.4 Tutorials

- [Analysis Tutorial](#): A case study on single-cell RNA sequencing data analysis using Pegasus.
- [Plotting Tutorial](#): Show how to use Pegasus to generate different kinds of plots for analysis.
- [Batch Correction Tutorial](#): Introduction on batch correction / data integration methods available in Pegasus.
- [Spatial Analysis Tutorial](#): Introduction on spatial analysis using Pegasus.
- [Doublet Detection Tutorial](#): Introduction on doublet detection method in Pegasus.
- [Regress Out Tutorial](#): Introduction on regressing out via a case study on cell-cycle gene effects.
- [Pseudobulk Analysis Tutorial](#): Introduction on pseudobulk analysis using Pegasus.

## 1.2.5 API

*Pegasus* can also be used as a python package. Import *pegasus* by:

```
import pegasus as pg
```

### Read and Write

<code>read_input(input_file[, file_type, mode, ...])</code>	Load data into memory.
<code>write_output(data, output_file[, file_type, ...])</code>	Write data back to disk.
<code>aggregate_matrices(csv_file[, restrictions, ...])</code>	Aggregate channel-specific count matrices into one big count matrix.

### pegasus.read\_input

`pegasus.read_input(input_file, file_type=None, mode='r', genome=None, modality=None, black_list=None, select_data=None, select_genome=None, select_modality=None, transpose=False)`

Load data into memory. This function is used to load input data into memory. Inputs can be in ‘zarr’, ‘h5ad’, ‘loom’, ‘10x’, ‘mtx’, ‘csv’, ‘tsv’, ‘fcs’ (for flow/mass cytometry data) or ‘nanosttring’ (Nanosttring GeoMx spatial data) formats.

#### Parameters

- **input\_file** (*str*) – Input file name.
- **file\_type** (*str*, optional (default: None)) – File type, choosing from ‘zarr’, ‘h5ad’, ‘loom’, ‘10x’, ‘mtx’, ‘csv’, ‘tsv’, ‘fcs’ (for flow/mass cytometry data), ‘nanosttring’ or ‘visium’. If None, inferred from input\_file.
- **mode** (*str*, optional (default: ‘r’)) – File open mode, options are ‘r’ or ‘a’. If mode == ‘a’, file\_type must be zarr and ngene/select\_singlets cannot be set.
- **genome** (*str*, optional (default: None)) – For formats like loom, mtx, dge, csv and tsv, genome is used to provide genome name. In this case if genome is None, except mtx format, “unknown” is used as the genome name instead.
- **modality** (*str*, optional (default: None)) – Default modality, choosing from ‘rna’, ‘atac’, ‘tcr’, ‘bcr’, ‘crispr’, ‘hashing’, ‘citeseq’, ‘cyto’ (flow cytometry / mass cytometry) or ‘nanosttring’. If None, use ‘rna’ as default.
- **black\_list** (*Set[str]*, optional (default: None)) – Attributes in black list will be popped out.
- **select\_data** (*Set[str]*, optional (default: None)) – Only select data with keys in select\_data. Select\_data, select\_genome and select\_modality are mutually exclusive.
- **select\_genome** (*Set[str]*, optional (default: None)) – Only select data with genomes in select\_genome. Select\_data, select\_genome and select\_modality are mutually exclusive.
- **select\_modality** (*Set[str]*, optional (default: None)) – Only select data with modalities in select\_modality. Select\_data, select\_genome and select\_modality are mutually exclusive.
- **transpose** (*bool*, optional (default: False)) – Only applicable if input type is ‘csv’ or ‘tsv’. Need to turn it on if gene names are columns names.

#### Return type

A MultimodalData object.



## Examples

```
>>> data = io.read_input('example_10x.h5')
>>> data = io.read_input('example.h5ad')
>>> data = io.read_input('example_ADT.csv', genome = 'hashing-HTO', modality =
↳ 'hashing')
```

## pegasus.write\_output

`pegasus.write_output(data, output_file, file_type=None, is_sparse=True, precision=2)`

Write data back to disk. This function is used to write data back to disk.

### Parameters

- **data** (*MutimodalData*) – data to write back.
- **output\_file** (*str*) – output file name. Note that for mtx files, output\_file specifies a directory. For scp format, file\_type must be specified.
- **file\_type** (*str*, optional (default: None)) – File type can be ‘zarr’ (as folder), ‘zarr.zip’ (as a ZIP file), ‘h5ad’, ‘loom’, ‘mtx’, ‘scp’ or ‘h5’. If file\_type is None, it will be inferred based on output\_file.
- **is\_sparse** (*bool*, optional (default: True)) – Only used for writing out SCP-compatible files, if write expression as a sparse matrix.
- **precision** (*int*, optional (default: 2)) – Precision after decimal point for values in mtx and scp expression matrix.

### Return type

*None*

## Examples

```
>>> io.write_output(data, 'test.zarr')
```

## pegasus.aggregate\_matrices

`pegasus.aggregate_matrices(csv_file, restrictions=[], attributes=[], default_ref=None, append_sample_name=True, select_singlets=False, remap_string=None, subset_string=None, min_genes=None, max_genes=None, min_umis=None, max_umis=None, mito_prefix=None, percent_mito=None)`

Aggregate channel-specific count matrices into one big count matrix.

This function takes as input a `csv_file`, which contains at least 2 columns — Sample, sample name; Location, file that contains the count matrices (e.g. `filtered_gene_bc_matrices_h5.h5`), and merges matrices from the same genome together. If multi-modality exists, a third Modality column might be required. An aggregated Multi-modal Data will be returned.

If `csv_file` is a dictionary, similarly as above, it must contain key `Sample` for sample names. Besides, the dictionary can either take `Location` key for count matrix file paths for samples, or `Object` for the `MutimodalData` objects already loaded from those files. All the keys in the dictionary must keep values as lists of the same length, and `aggregate_matrices` function makes copies instead of editing the provided data objects.

The `csv_file` can optionally contain two columns - `nUMI` and `nGene`. These two columns define minimum number of UMIs and genes for cell selection for each sample. The values in these two columns overwrite the `min_genes` and `min_umis` arguments.

### Parameters

- **csv\_file** (*str*) – The CSV file containing information about each channel. Alternatively, a dictionary or `pd.DataFrame` can be passed.
- **restrictions** (*list[str]* or *str*, optional (default: [])) – A list of restrictions used to select channels, each restriction takes the format of `name:value,...,value` or `name:~value,...,value`, where `~` refers to not. If only one restriction is provided, it can be provided as a string instead of a list.
- **attributes** (*list[str]* or *str*, optional (default: [])) – A list of attributes need to be incorporated into the output count matrix. If only one attribute is provided, this attribute can be provided as a string instead of a list.
- **default\_ref** (*str*, optional (default: None)) – Default reference name to use. If there is no Reference column in the `csv_file`, a Reference column will be added with `default_ref` as its value. This argument can also be used for replacing genome names. For example, if `default_ref` is `'hg19:GRCh38,GRCh38'`, we will change any genome with name `'hg19'` to `'GRCh38'` and if no genome is provided, `'GRCh38'` is the default.
- **append\_sample\_name** (*bool*, optional (default: True)) – By default, append `sample_name` to each channel. Turn this option off if each channel has distinct barcodes.
- **select\_singlets** (*bool*, optional (default: False)) – If we have demultiplexed data, turning on this option will make pegasus only include barcodes that are predicted as singlets.
- **remap\_string** (*str*, optional, default None) – Remap singlet names using `<remap_string>`, where `<remap_string>` takes the format `"new_name_i:old_name_1,old_name_2;new_name_ii:old_name_3;..."`. For example, if we hashed 5 libraries from 3 samples `sample1_lib1`, `sample1_lib2`, `sample2_lib1`, `sample2_lib2` and `sample3`, we can remap them to 3 samples using this string: `"sample1:sample1_lib1,sample1_lib2;sample2:sample2_lib1,sample2_lib2"`. In this way, the new singlet names will be in metadata field with key `'assignment'`, while the old names will be kept in metadata field with key `'assignment.orig'`.
- **subset\_string** (*str*, optional, default None) – If select singlets, only select singlets in the `<subset_string>`, which takes the format `"name1,name2,..."`. Note that if `–remap-singlets` is specified, subsetting happens after remapping. For example, we can only select singlets from sample 1 and 3 using `"sample1,sample3"`.
- **min\_genes** (*int*, optional, default: None) – Only keep cells with at least `min_genes` genes.
- **max\_genes** (*int*, optional, default: None) – Only keep cells with less than `max_genes` genes.
- **min\_umis** (*int*, optional, default: None) – Only keep cells with at least `min_umis` UMIs.
- **max\_umis** (*int*, optional, default: None) – Only keep cells with less than `max_umis` UMIs.
- **mito\_prefix** (*str*, optional, default: None) – Prefix for mitochondrial genes.
- **percent\_mito** (*float*, optional, default: None) – Only keep cells with percent mitochondrial genes less than `percent_mito` % of total counts. Only when both `mito_prefix` and `percent_mito` set, the mitochondrial filter will be triggered.

### Returns

The aggregated count matrix as an `MultimodalData` object.

**Return type***MultimodalData* object.**Examples**

```
>>> data = aggregate_matrices('example.csv', restrictions=['Source:pbmc', 'Donor:1
↪'], attributes=['Source', 'Platform', 'Donor'])
>>> data = aggregate_matrices({'Sample': ['sample1', 'sample2'], 'Object': [data1,
↪data2]})
```

**Analysis Tools****Preprocess**

<code>qc_metrics(data[, select_singlets, ...])</code>	Generate Quality Control (QC) metrics regarding cell barcodes on the dataset.
<code>get_filter_stats(data[, min_genes_before_filt])</code>	Calculate filtration stats on cell barcodes.
<code>filter_data(data[, focus_list])</code>	Filter data based on qc_metrics calculated in pg.qc_metrics.
<code>identify_robust_genes(data[, percent_cells])</code>	Identify robust genes as candidates for HVG selection and remove genes that are not expressed in any cells.
<code>log_norm(data[, norm_count, base_matrix, ...])</code>	Normalize each cell by total counts, and then apply natural logarithm to the data.
<code>log1p(data[, base_matrix, target_matrix, select])</code>	Apply the Log(1+x) transformation on the given count matrix.
<code>normalize(data[, norm_count, base_matrix, ...])</code>	Normalize each cell by total count.
<code>arcsinh(data[, cofactor, jitter, ...])</code>	Conduct arcsinh transform on the current matrix.
<code>highly_variable_features(data[, batch, ...])</code>	Highly variable features (HVF) selection.
<code>select_features(data[, features, ...])</code>	Subset the features and store the resulting matrix in dense format in data.uns with '_tmp_fmat_' prefix, with the option of standardization and truncating based on max_value.
<code>pca(data[, n_components, features, ...])</code>	Perform Principle Component Analysis (PCA) to the data.
<code>nmf(data[, n_components, features, space, ...])</code>	Perform Nonnegative Matrix Factorization (NMF) to the data using Frobenius norm.
<code>regress_out(data, attrs[, rep])</code>	Regress out effects due to specific observational attributes.
<code>calculate_z_score(data[, n_bins])</code>	Calculate the standardized z scores of the count matrix.

## pegasus.qc\_metrics

`pegasus.qc_metrics(data, select_singlets=False, remap_string=None, subset_string=None, min_genes=None, max_genes=None, min_umis=None, max_umis=None, mito_prefix=None, percent_mito=None)`

Generate Quality Control (QC) metrics regarding cell barcodes on the dataset.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **select\_singlets** (`bool`, optional, default `False`) – If select only singlets.
- **remap\_string** (`str`, optional, default `None`) – Remap singlet names using `<remap_string>`, where `<remap_string>` takes the format “new\_name\_i:old\_name\_1,old\_name\_2;new\_name\_ii:old\_name\_3;...”. For example, if we hashed 5 libraries from 3 samples `sample1_lib1`, `sample1_lib2`, `sample2_lib1`, `sample2_lib2` and `sample3`, we can remap them to 3 samples using this string: “sample1:sample1\_lib1,sample1\_lib2;sample2:sample2\_lib1,sample2\_lib2”. In this way, the new singlet names will be in metadata field with key ‘assignment’, while the old names will be kept in metadata field with key ‘assignment.orig’.
- **subset\_string** (`str`, optional, default `None`) – If select singlets, only select singlets in the `<subset_string>`, which takes the format “name1,name2,...”. Note that if `-remap-singlets` is specified, subsetting happens after remapping. For example, we can only select singlets from sample 1 and 3 using “sample1,sample3”.
- **min\_genes** (`int`, optional, default: `None`) – Only keep cells with at least `min_genes` genes.
- **max\_genes** (`int`, optional, default: `None`) – Only keep cells with less than `max_genes` genes.
- **min\_umis** (`int`, optional, default: `None`) – Only keep cells with at least `min_umis` UMIs.
- **max\_umis** (`int`, optional, default: `None`) – Only keep cells with less than `max_umis` UMIs.
- **mito\_prefix** (`str`, optional, default: `None`) – Prefix for mitochondrial genes.
- **percent\_mito** (`float`, optional, default: `None`) – Only keep cells with percent mitochondrial genes less than `percent_mito` % of total counts.

### Return type

`None`

### Returns

- `None`
- Update `data.obs` –
  - `n_genes`: Total number of genes for each cell.
  - `n_counts`: Total number of counts for each cell.
  - `percent_mito`: Percent of mitochondrial genes for each cell.
  - `passed_qc`: Boolean type indicating if a cell passes the QC process based on the QC metrics.
  - `demux_type`: this column might be deleted if `select_singlets` is on.

## Examples

```
>>> pg.qc_metrics(data, min_genes=500, max_genes=6000, mito_prefix="MT-", percent_
↳ mito=10)
```

## pegasus.get\_filter\_stats

`pegasus.get_filter_stats(data, min_genes_before_filt=100)`

Calculate filtration stats on cell barcodes.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **min\_genes\_before\_filt** (int, optional, default 100) – If raw data matrix is input, empty barcodes will dominate pre-filtration statistics. To avoid this, for raw matrix, only consider barcodes with at least <number> genes for pre-filtration condition.

### Returns

**df\_cells** – Data frame of stats on cell filtration.

### Return type

`pandas.DataFrame`

## Examples

```
>>> df = pg.get_filter_stats(data)
```

## pegasus.filter\_data

`pegasus.filter_data(data, focus_list=None)`

Filter data based on qc\_metrics calculated in `pg.qc_metrics`.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **focus\_list** (`List[str]`, optional, default None) – `UnimodalData` objects with keys in `focus_list` were qc\_metrics marked. Filter them and make sure other modalities' barcodes are consistent with filtered barcodes. If `focus_list` is None and `self._selected`'s modality is "rna", `focus_list = [self._selected]`

### Return type

None

### Returns

- None
- Update data with cells after filtration.

## Examples

```
>>> pg.filter_data(data)
```

## pegasus.identify\_robust\_genes

`pegasus.identify_robust_genes(data, percent_cells=0.05)`

Identify robust genes as candidates for HVG selection and remove genes that are not expressed in any cells.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **percent\_cells** (float, optional, default: `0.05`) – Only assign genes to be robust that are expressed in at least `percent_cells` % of cells.

### Return type

None

### Returns

- None
- Update `data.var` –
  - `n_cells`: Total number of cells in which each gene is measured.
  - `percent_cells`: Percent of cells in which each gene is measured.
  - `robust`: Boolean type indicating if a gene is robust based on the QC metrics.
  - `highly_variable_features`: Boolean type indicating if a gene is a highly variable feature. By default, set all robust genes as highly variable features.

## Examples

```
>>> pg.identify_robust_genes(data, percent_cells = 0.05)
```

## pegasus.log\_norm

`pegasus.log_norm(data, norm_count=100000.0, base_matrix=None, target_matrix=None, select=True)`

Normalize each cell by total counts, and then apply natural logarithm to the data.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **norm\_count** (int, optional, default: `1e5`) – Total counts of one cell after normalization.
- **base\_matrix** (str, optional, default: `None`.) – The key name of the matrix to perform `log_norm`. If `None`, the current matrix.
- **target\_matrix** (str, optional, default: `None`.) – The key name of the matrix to store the `log_normed` results. If `None`, `base_matrix` + “.log\_norm”.

- **select** (bool, optional, default: None.) – Select the log\_normed matrix as the current matrix (can be accessed via data.X).

**Return type**

None

**Returns**

- None
- Add the log-normalized matrix to `data.matrices`. Add the normalization scale vector to `data.obs["scale"]` and `norm_count` parameter to `data.uns["norm_count"]`.
- Note that if the detected `base_matrix` == 'X', we'll change the name to *counts* instead.

**Examples**

```
>>> pg.log_norm(data)
```

**pegasus.log1p**

`pegasus.log1p(data, base_matrix=None, target_matrix=None, select=True)`

Apply the Log(1+x) transformation on the given count matrix.

**Parameters**

- **data** (pegasusio.MultimodalData) – Use current selected modality in data, which should contain one RNA expression matrix.
- **base\_matrix** (str, optional, default: None.) – The key name of the matrix to perform the transformation. If None, the current matrix.
- **target\_matrix** (str, optional, default: None.) – The key name of the matrix to store the transformed results. If None, `base_matrix + ".log1p"`.
- **select** (bool, optional, default: None.) – Select the transformed matrix as the current matrix (can be accessed via data.X).

**Return type**

None

**Returns**

- None
- Add the log1p matrix to `data.matrices`.
- Note that if the detected `base_matrix` == 'X', we'll change the name to *counts* instead.

## Examples

```
>>> pg.log1p(data)
```

## pegasus.normalize

`pegasus.normalize(data, norm_count=100000.0, base_matrix=None, target_matrix=None, select=True)`

Normalize each cell by total count.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data, which should contain one RNA expression matrix.
- **norm\_count** (int, optional, default: 1e5.) – Total counts of one cell after normalization.
- **base\_matrix** (str, optional, default: None.) – The key name of the matrix to perform normalize. If None, the current matrix.
- **target\_matrix** (str, optional, default: None.) – The key name of the matrix to store the normalize results. If None, `base_matrix + “.norm”`.
- **select** (bool, optional, default: None.) – Select the normalized matrix as the current matrix (can be accessed via `data.X`).

### Return type

None

### Returns

- None
- Add the normalized matrix to `data.matrices`. Add the normalization scale vector to `data.obs["scale"]` and `norm_count` parameter to `data.uns["norm_count"]`.
- Note that if the detected `base_matrix` == 'X', we'll change the name to `counts` instead.

## Examples

```
>>> pg.normalize(data)
```

## pegasus.arcsinh

`pegasus.arcsinh(data, cofactor=5.0, jitter=False, random_state=0, base_matrix=None, target_matrix=None, select=True)`

Conduct arcsinh transform on the current matrix.

If `jitter == True`, jittering by adding a randomized value in  $U([-0.5, 0.5])$ . This will also make the matrix dense. Mimic Cytobank.

### Parameters

- **cofactor** (float, optional, default: 5.0) – Cofactor used in cytobank,  $\text{arcsinh}(x / \text{cofactor})$ .
- **jitter** (bool, optional, default: False) – Add a 'arcsinh.jitter' matrix in dense format, jittering by adding a randomized value in  $U([-0.5, 0.5])$ .



- **random\_state** (int, optional, default: 0) – Random seed for generating jitters.
- **base\_matrix** (str, optional, default: None.) – The key name of the matrix to perform arcsinh. If None, the current matrix.
- **target\_matrix** (str, optional, default: None.) – The key name of the matrix to store the arcsinh results. If None, base\_matrix + “.arcsinh”.
- **select** (bool, optional, default: None.) – Select the arcsinh matrix as the current matrix (can be accessed via data.X).

**Return type**

None

**Returns**

- None
- Add the arcsinh matrix to `data.matrices`.
- Note that if the detected `base_matrix` == 'X', we'll change the name to `counts` instead.

**Examples**

```
>>> pg.arcsinh(data)
```

**pegasus.highly\_variable\_features**

```
pegasus.highly_variable_features(data, batch=None, flavor='pegasus', n_top=2000, span=0.02,
                                min_disp=0.5, max_disp=inf, min_mean=0.0125, max_mean=7,
                                n_jobs=-1)
```

Highly variable features (HVF) selection. The input data should be logarithmized.

**Parameters**

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **batch** (str, optional, default: None) – A key in data.obs specifying batch information. If `batch` is not set, do not consider batch effects in selecting highly variable features. Otherwise, if `data.obs[batch]` is not categorical, `data.obs[batch]` will be automatically converted into categorical before highly variable feature selection.
- **flavor** (str, optional, default: "pegasus") – The HVF selection method to use. Available choices are "pegasus" or "Seurat".
- **n\_top** (int, optional, default: 2000) – Number of genes to be selected as HVF. If None, no gene will be selected.
- **span** (float, optional, default: 0.02) – Only applicable when `flavor` is "pegasus". The smoothing factor used by *scikit-learn* *loess* model in pegasus HVF selection method.
- **min\_disp** (float, optional, default: 0.5) – Minimum normalized dispersion.
- **max\_disp** (float, optional, default: np.inf) – Maximum normalized dispersion. Set it to `np.inf` for infinity bound.
- **min\_mean** (float, optional, default: 0.0125) – Minimum mean.
- **max\_mean** (float, optional, default: 7) – Maximum mean.

- **n\_jobs** (int, optional, default: -1) – Number of threads to be used during calculation. If -1, all physical CPU cores will be used.

**Return type**

None

**Returns**

- None
- Update `data.var` –
  - `highly_variable_features`: replace with Boolean type array indicating the selected highly variable features.

**Examples**

```
>>> pg.highly_variable_features(data)
>>> pg.highly_variable_features(data, batch="Channel")
```

**pegasus.select\_features**

`pegasus.select_features(data, features='highly_variable_features', standardize=True, max_value=10.0)`

Subset the features and store the resulting matrix in dense format in `data.uns` with ‘\_tmp\_fmat\_’ prefix, with the option of standardization and truncating based on `max_value`. ‘\_tmp\_fmat\_’ will be removed before writing out the disk. :type data: Union[MultimodalData, UnimodalData] :param data: Annotated data matrix with rows for cells and columns for genes. :type data: pegasusio.MultimodalData :type features: str :param features: a keyword in `data.var`, which refers to a boolean array. If None, all features will be selected. :type features: str, optional, default: `highly_variable_features`. :type standardize: bool :param standardize: Whether to scale the data to unit variance and zero mean. :type standardize: bool, optional, default: True. :type max\_value: float :param max\_value: The threshold to truncate data after scaling. If None, do not truncate. :type max\_value: float, optional, default: 10.

**Return type**

str

**Returns**

- **keyword** (str) – The keyword in `data.uns` referring to the features selected.
- Update `data.uns` if needed –
  - `data.uns[keyword]`: A submatrix of the data containing features selected.

**Examples**

```
>>> pg.select_features(data)
```

**pegasus.pca**

```
pegasus.pca(data, n_components=50, features='highly_variable_features', standardize=True, max_value=10.0,
            n_jobs=-1, random_state=0)
```

Perform Principle Component Analysis (PCA) to the data.

The calculation uses *scikit-learn* implementation.

**Parameters**

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **n\_components** (int, optional, default: 50.) – Number of Principal Components to get.
- **features** (str, optional, default: "highly\_variable\_features".) – Keyword in data.var to specify features used for PCA. If None, all features will be selected.
- **standardize** (bool, optional, default: True.) – Whether to scale the data to unit variance and zero mean.
- **max\_value** (float, optional, default: 10.) – The threshold to truncate data after scaling. If None, do not truncate.
- **n\_jobs** (int, optional (default: -1)) – Number of threads to use. -1 refers to using all physical CPU cores.
- **random\_state** (int, optional, default: 0.) – Random seed to be set for reproducing result.

**Return type**

None

**Returns**

- None.
- Update `data.obsm` –
  - `data.obsm["X_pca"]`: PCA matrix of the data.
- Update `data.uns` –
  - `data.uns["PCs"]`: The principal components containing the loadings.
  - `data.uns["pca_variance"]`: Explained variance, i.e. the eigenvalues of the covariance matrix.
  - `data.uns["pca_variance_ratio"]`: Ratio of explained variance.
  - `data.uns["pca_features"]`: Record the features used to generate PCA components.

**Examples**

```
>>> pg.pca(data)
```

## pegasus.nmf

```
pegasus.nmf(data, n_components=20, features='highly_variable_features', space='log', init='nndsvdar',
            algo='halsvar', mode='batch', tol=0.0001, use_gpu=False, alpha_W=0.0, l1_ratio_W=0.0,
            alpha_H=0.0, l1_ratio_H=0.0, fp_precision='float', n_jobs=-1, random_state=0)
```

Perform Nonnegative Matrix Factorization (NMF) to the data using Frobenius norm. Steps include select features and L2 normalization and NMF and L2 normalization of resulting coordinates.

The calculation uses `nmf-torch` package.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **n\_components** (int, optional, default: 50.) – Number of Principal Components to get.
- **features** (str, optional, default: "highly\_variable\_features".) – Keyword in data. var to specify features used for nmf.
- **max\_value** (float, optional, default: None.) – The threshold to truncate data symmetrically after scaling. If None, do not truncate.
- **space** (str, optional, default: log.) – Choose from log and expression. log works on log-transformed expression space; expression works on the original expression space (normalized by total UMIs).
- **init** (str, optional, default: nndsvdar.) – Method to initialize NMF. Options are 'random', 'nndsvd', 'nndsvda' and 'nndsvdar'.
- **algo** (str, optional, default: halsvar) – Choose from mu (Multiplicative Update), hals (Hierarchical Alternative Least Square), halsvar (HALS variant, use HALS to mimic bpp and can get better convergence for sometimes) and bpp (alternative non-negative least squares with Block Principal Pivoting method).
- **mode** (str, optional, default: batch) – Learning mode. Choose from batch and online. Notice that online only works when beta=2.0. For other beta loss, it switches back to batch method.
- **tol** (float, optional, default: 1e-4) – The toleration used for convergence check.
- **use\_gpu** (bool, optional, default: False) – If True, use GPU if available. Otherwise, use CPU only.
- **alpha\_W** (float, optional, default: 0.0) – A numeric scale factor which multiplies the regularization terms related to W. If zero or negative, no regularization regarding W is considered.
- **l1\_ratio\_W** (float, optional, default: 0.0) – The ratio of L1 penalty on W, must be between 0 and 1. And thus the ratio of L2 penalty on W is (1 - l1\_ratio\_W).
- **alpha\_H** (float, optional, default: 0.0) – A numeric scale factor which multiplies the regularization terms related to H. If zero or negative, no regularization regarding H is considered.
- **l1\_ratio\_H** (float, optional, default: 0.0) – The ratio of L1 penalty on W, must be between 0 and 1. And thus the ratio of L2 penalty on H is (1 - l1\_ratio\_H).
- **fp\_precision** (str, optional, default: float) – The numeric precision on the results. Choose from float and double.
- **n\_jobs** (int, optional (default: -1)) – Number of threads to use. -1 refers to using all physical CPU cores.

- **random\_state** (int, optional, default: 0.) – Random seed to be set for reproducing result.

**Return type**

None

**Returns**

- None.
- Update `data.obsm` –
  - `data.obsm["X_nmf"]`: Scaled NMF coordinates of shape (n\_cells, n\_components). Each column has a unit variance.
  - `data.obsm["H"]`: The coordinate factor matrix of shape (n\_cells, n\_components).
- Update `data.uns` –
  - `data.uns["W"]`: The feature factor matrix of shape (n\_HVFs, n\_components).
  - `data.uns["nmf_err"]`: The NMF loss.
  - `data.uns["nmf_features"]`: Record the features used to perform NMF analysis.

**Examples**

```
>>> pg.nmf(data)
```

**pegasus.regress\_out**

`pegasus.regress_out(data, attrs, rep='pca')`

Regress out effects due to specific observational attributes.

**Parameters**

- **data** (MultimodalData or UnimodalData object) – Annotated data matrix with rows for cells and columns for genes.
- **attrs** (List[str]) – List of numeric cell attributes to be regressed out. They must exist in `data.obs` field.
- **rep** (str, optional, default: pca) – This is to specify which embedding to be used for regressing out. The key 'X\_'+rep must exist in `data.obsm` field. By default, use PCA embedding.

**Return type**

str

**Returns**

- **res\_key** (str) – The key to the resulting new embedding matrix in `data.obsm`. It's 'X\_'+rep+'\_regressed'.
- Update `data.obsm` –
  - `data.obsm[pca_key]`: The PCA matrix with effects of attributes specified regressed out.

## Examples

```
>>> pg.regress_out(data, attrs=['G1/S', 'G2/M'])
```

## pegasus.calculate\_z\_score

`pegasus.calculate_z_score(data, n_bins=50)`

Calculate the standardized z scores of the count matrix.

### Parameters

- **data** (MultimodalData, UnimodalData, or `anndata.AnnData` object.) – Single cell expression data.
- **n\_bins** (int, optional, default: 50) – Number of bins on expression levels for grouping genes.

### Returns

A 2D numpy array of shape (n\_cells, n\_features), which represents the standardized z-score expression matrix.

### Return type

numpy.array

## Examples

```
>>> pg.calculate_z_score(data)
>>> pg.calculate_z_score(data, n_bins=100)
```

## Batch Correction

<code>run_harmony(data[, batch, rep, n_comps, ...])</code>	Batch correction on PCs using Harmony.
<code>run_scanorama(data[, batch, n_components, ...])</code>	Batch correction using Scanorama.
<code>integrative_nmf(data[, batch, n_components, ...])</code>	Perform Integrative Nonnegative Matrix Factorization (iNMF) [Yang16] for data integration.
<code>run_scvi(data[, features, matkey, n_jobs, ...])</code>	Run scVI embedding.

## pegasus.run\_harmony

`pegasus.run_harmony(data, batch='Channel', rep='pca', n_comps=None, n_jobs=-1, n_clusters=None, random_state=0, use_gpu=False, max_iter_harmony=10)`

Batch correction on PCs using Harmony.

This is a wrapper of `harmony-pytorch` package, which is a Pytorch implementation of Harmony algorithm [Korsunsky19].

### Parameters

- **data** (MultimodalData.) – Annotated data matrix with rows for cells and columns for genes.

- **batch** (str, optional, default: "Channel".) – Which attribute in data.obs field represents batches, default is "Channel".
- **rep** (str, optional, default: "pca".) – Which representation to use as input of Harmony, default is PCA.
- **n\_comps** (int, optional (default: None)) – Number of components to be used in the *rep*. If `n_comps == None`, use all components; otherwise, use the minimum of `n_comps` and `rep`'s dimensions.
- **n\_jobs** (int, optional, default: -1.) – Number of threads to use in Harmony. -1 refers to using all physical CPU cores.
- **n\_clusters** (int, optional, default: None.) – Number of Harmony clusters. Default is None, which asks Harmony to estimate this number from the data.
- **random\_state** (int, optional, default: 0.) – Seed for random number generator
- **use\_gpu** (bool, optional, default: False.) – If True, use GPU if available. Otherwise, use CPU only.
- **max\_iter\_harmony** (int, optional, default: 10.) – Maximum iterations on running Harmony if not converged.

**Return type**

str

**Returns**

- **out\_rep** (str) – The keyword in data.obsm referring to the embedding calculated by Harmony algorithm.  
This keyword is `rep + '_harmony'`, where `rep` is the input parameter above.
- Update data.obsm –  
– data.obsm['X\_' + out\_rep]: The embedding calculated by Harmony algorithm.

**Examples**

```
>>> pg.run_harmony(data, rep = "pca", n_jobs = 10, random_state = 25)
```

**pegasus.run\_scanorama**

```
pegasus.run_scanorama(data, batch='Channel', n_components=50, features='highly_variable_features',
                      standardize=True, max_value=10.0, random_state=0)
```

Batch correction using Scanorama.

This is a wrapper of [Scanorama](#) package. See [Hie19] for details on the algorithm.

**Parameters**

- **data** (MultimodalData.) – Annotated data matrix with rows for cells and columns for genes.
- **batch** (str, optional, default: "Channel".) – Which attribute in data.obs field represents batches, default is "Channel".
- **n\_components** (int, optional default: 50.) – Number of integrated embedding components to keep. This sets Scanorama's `dimred` parameter.

- **features** (str, optional, default: "highly\_variable\_features".) – Keyword in data.var to specify features used for Scanorama.
- **standardize** (bool, optional, default: True.) – Whether to scale the data to unit variance and zero mean.
- **max\_value** (float, optional, default: 10.) – The threshold to truncate data after scaling. If None, do not truncate.
- **random\_state** (int, optional, default: 0.) – Seed for random number generator.

**Return type**

str

**Returns**

- **out\_rep** (str) – The keyword in data.obsm referring to the embedding calculated by Scanorama algorithm. out\_rep is always equal to “scanorama”
- Update data.obsm –
  - data.obsm['X\_scanorama']: The embedding calculated by Scanorama algorithm.

**Examples**

```
>>> pg.run_scanorama(data, random_state = 25)
```

**pegasus.integrative\_nmf**

```
pegasus.integrative_nmf(data, batch='Channel', n_components=20, features='highly_variable_features',  
                        space='log', algo='halsvar', mode='online', tol=0.0001, use_gpu=False, lam=5.0,  
                        fp_precision='float', n_jobs=-1, random_state=0, quantile_norm=True)
```

Perform Integrative Nonnegative Matrix Factorization (iNMF) [Yang16] for data integration.

The calculation uses `nmf-torch`.

This function assumes that cells in each batch are adjacent to each other. In addition, it will scale each batch with L2 norm separately. The resulting Hs will also be scaled with L2 norm. If `quantile_norm=True`, quantile normalization will be additionally performed.

See [Welch19] and [Gao21] for preprocessing and normalization details.

**Parameters**

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **batch** (str, optional, default: "Channel".) – Which attribute in data.obs field represents batches, default is “Channel”.
- **n\_components** (int, optional, default: 50.) – Number of Principal Components to get.
- **features** (str, optional, default: "highly\_variable\_features".) – Keyword in data.var to specify features used for integrative\_nmf.
- **space** (str, optional, default: log.) – Choose from log and expression. log works on log-transformed expression space; expression works on the original expression space (normalized by total UMIs).



- **algo** (str, optional, default: `halsvar`) – Choose from `mu` (Multiplicative Update), `halsvar` (HALS variant that mimic `bpp` but faster) and `bpp` (alternative non-negative least squares with Block Principal Pivoting method).
- **mode** (str, optional, default: `online`) – Learning mode. Choose from `batch` and `online`. Notice that `online` only works when `beta=2.0`. For other `beta` loss, it switches back to `batch` method.
- **tol** (float, optional, default: `1e-4`) – The toleration used for convergence check.
- **use\_gpu** (bool, optional, default: `False`) – If `True`, use GPU if available. Otherwise, use CPU only.
- **lam** (float, optional, default: `5.0`) – The coefficient for regularization terms. If `0`, then no regularization will be performed.
- **fp\_precision** (str, optional, default: `float`) – The numeric precision on the results. Choose from `float` and `double`.
- **n\_jobs** (int, optional (default: `-1`)) – Number of threads to use. `-1` refers to using all physical CPU cores.
- **random\_state** (int, optional, default: `0`.) – Random seed to be set for reproducing result.
- **quantile\_norm** (bool, optional, default: `True`.) – Perform quantile normalization as described in Gao et al. Nature Biotech 2021. Cluster refinement `K=20`; `min_cells=20`; `quantiles=50`.

#### Returns

**out\_rep** – The keyword in `data.obsm` referring to the embedding calculated by integrative NMF algorithm. `out_rep` is always equal to “`inmf`”

#### Return type

str

Update `data.obsm`:

- `data.obsm["X_inmf"]`: Scaled and possibly quantile normalized iNMF coordinates.
- `data.obsm["H"]`: The concatenation of coordinate factor matrices of shape (`n_cells`, `n_components`).

Update `data.uns`:

- `data.uns["W"]`: The feature factor matrix of shape (`n_HVFs`, `n_components`).
- `data.uns["V"]`: The batch specific feature factor matrices as one tensor of shape (`n_batches`, `n_components`, `n_HVFs`).
- `data.uns["inmf_err"]`: The iNMF loss.
- `data.uns["inmf_features"]`: Record the features used to perform iNMF analysis.

## Examples

```
>>> pg.integrative_nmf(data)
```

## pegasus.run\_scvi

```
pegasus.run_scvi(data, features='highly_variable_features', matkey='counts', n_jobs=-1, random_state=0,
                  max_epochs=None, batch=None, categorical_covariate_keys=None,
                  continuous_covariate_keys=None, use_gpu=None)
```

Run scVI embedding.

This is a wrapper of `scvtools` package.

### Parameters

- **data** (MultimodalData.) – Annotated data matrix with rows for cells and columns for genes.
- **features** (str, optional, default: "highly\_variable\_features") – Keyword in data.var, which refers to a boolean array. If None, all features will be selected.
- **matkey** (str, optional, default: "counts") – Matrix key for the raw count
- **n\_jobs** (int, optional, default: -1.) – Number of threads to use. -1 refers to using all physical CPU cores.
- **random\_state** (int, optional, default: 0.) – Seed for random number generator
- **max\_epochs** (int | None, optional, default: None.) – Maximum number of training epochs. Defaults to `np.min([round((20000 / n_cells) * 400), 400])`
- **batch** (str, optional, default: None.) – If only one categorical covariate, the obs key representing batches that should be corrected for, default is None.
- **categorical\_covariate\_keys** (List[str]) – If multiple categorical covariates, a list of obs keys listing categorical covariates that should be corrected for, default is None.
- **continuous\_covariate\_keys** (List[str]) – A list of obs keys listing continuous covariates that should be corrected for, default is None.
- **use\_gpu** (str | int | bool | None) – Use default GPU if available (if None or True), or index of GPU to use (if int), or name of GPU (if str, e.g., `cuda:0`), or use CPU (if False).

### Return type

str

### Returns

- **out\_rep** (str) – The keyword in data.obsm referring to the embedding calculated by integrative NMF algorithm. out\_rep is always equal to "scVI"
- Update data.obsm –
  - data.obsm['X\_scVI']: The embedding calculated by scVI.

## Examples

```
>>> pg.run_scvi(data, batch="Channel")
>>> pg.run_scvi(data, categorical_covariate_keys=["cell_source", "donor"],
↳ continuous_covariate_keys=["percent_mito", "percent_ribo"])
```

## Nearest Neighbors

<code>neighbors(data[, K, rep, n_comps, n_jobs, ...])</code>	Compute k nearest neighbors and affinity matrix, which will be used for diffmap and graph-based community detection algorithms.
<code>get_neighbors(data[, K, rep, n_comps, ...])</code>	Find K nearest neighbors for each data point and return the indices and distances arrays.
<code>calc_kBET(data, attr[, rep, K, alpha, ...])</code>	Calculate the kBET metric of the data regarding a specific sample attribute and embedding.
<code>calc_kSIM(data, attr[, rep, K, min_rate, ...])</code>	Calculate the kSIM metric of the data regarding a specific sample attribute and embedding.

## pegasus.neighbors

`pegasus.neighbors(data, K=100, rep='pca', n_comps=None, n_jobs=-1, random_state=0, full_speed=False, use_cache=False, dist='l2', method='hnsw', exact_k=False)`

Compute k nearest neighbors and affinity matrix, which will be used for diffmap and graph-based community detection algorithms.

The kNN calculation uses [hnswlib](#) introduced by [Malkov16].

K is determined by `min(K, sqrt(data.shape[0]))`.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **K** (`int`, optional, default: 100) – Number of neighbors, including the data point itself.
- **rep** (`str`, optional, default: "pca") – Embedding representation used to calculate kNN. If `None`, use `data.X`; otherwise, keyword 'X\_' + `rep` must exist in `data.obsm`.
- **n\_comps** (`int`, optional (default: `None`)) – Number of components to be used in the `rep`. If `n_comps == None`, use all components; otherwise, use the minimum of `n_comps` and `rep`'s dimensions.
- **n\_jobs** (`int`, optional, default: -1) – Number of threads to use. If -1, use all physical CPU cores.
- **random\_state** (`int`, optional, default: 0) – Random seed set for reproducing results.
- **full\_speed** (`bool`, optional, default: `False`) –
  - If `True`, use multiple threads in constructing `hnsw` index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **use\_cache** (`bool`, optional, default: `False`) –

- If `True` and found cached knn results, Pegasus will use cached results and do not recompute.
- Otherwise, compute kNN irrespective of caching status.
- **dist** (`str`, optional (default: "l2")) – Distance metric to use. By default, use squared L2 distance. Available options, "l2" or inner product "ip" or cosine similarity "cosine".
- **method** (`str`, optional (default: "hnsr")) – Choose from "hnsr" or "sklearn". "hnsr" uses HNSW algorithm for approximate nearest neighbor search and "sklearn" uses sklearn package for exact nearest neighbor search.
- **exact\_k** (`bool`, optional (default: `False`)) – If `True`, use exactly the `K` passed to the function; otherwise `K` is determined as  $\min(K, \sqrt{X.shape[0]})$ .

**Return type**

None

**Returns**

- None
- Update `data.obsm` –
  - `data.obsm[rep + "_knn_indices"]`: kNN index matrix. Row `i` is the index list of kNN of cell `i` (excluding itself), sorted from nearest to farthest.
  - `data.obsm[rep + "_knn_distances"]`: kNN distance matrix. Row `i` is the distance list of kNN of cell `i` (excluding itself), sorted from smallest to largest.
- Update `data.obsp` –
  - `data.obsp["W_" + rep]`: kNN graph of the data in terms of affinity matrix.

**Examples**

```
>>> pg.neighbors(data)
```

**pegasus.get\_neighbors**

```
pegasus.get_neighbors(data, K=100, rep='pca', n_comps=None, n_jobs=-1, random_state=0,  
                      full_speed=False, use_cache=False, dist='l2', method='hnsr', exact_k=False)
```

Find `K` nearest neighbors for each data point and return the indices and distances arrays.

`K` is determined by  $\min(K, \text{int}(\sqrt{\text{data.shape}[0]}))$  if `exact_k == False`.

**Parameters**

- **data** (*pegasusio.MultimodalData*) – An `AnnData` object.
- **K** (*int*, optional (default: 100)) – Number of neighbors, including the data point itself.
- **rep** (*str*, optional (default: 'pca')) – Representation used to calculate kNN. If `None` use `data.X`.
- **n\_comps** (*int*, optional (default: `None`)) – Number of components to be used in the `rep`. If `n_comps == None`, use all components; otherwise, use the minimum of `n_comps` and `rep`'s dimensions.
- **n\_jobs** (*int*, optional (default: -1)) – Number of threads to use. -1 refers to using all physical CPU cores.

- **random\_state** (*int*, optional (default: 0)) – Random seed for random number generator.
- **full\_speed** (*bool*, optional (default: False)) – If `full_speed`, use multiple threads in constructing hnsw index. However, the kNN results are not reproducible. If not `full_speed`, use only one thread to make sure results are reproducible.
- **use\_cache** (*bool*, optional (default: False)) – If `use_cache` and found cached knn results, will not recompute.
- **dist** (*str*, optional (default: 'l2')) – Distance metric to use. By default, use squared L2 distance. Available options, 'l2' or inner product 'ip' or cosine similarity 'cosine'.
- **method** (*str*, optional (default: 'hnsw')) – Choosing from 'hnsw' for approximate nearest neighbor search or 'sklearn' for exact nearest neighbor search.
- **exact\_k** (*bool*, optional (default: 'False')) – If True, use exactly the K passed to the function; otherwise K is determined as  $\min(K, \sqrt{X.shape[0]})$ .

**Return type**

Tuple[List[int], List[float], int]

**Returns**

- 
- *kNN indices array, distances array, and adjusted K.*

**Examples**

```
>>> indices, distances, K = tools.get_neighbors(data)
```

**pegasus.calc\_kBET**

```
pegasus.calc_kBET(data, attr, rep='pca', K=25, alpha=0.05, n_jobs=-1, random_state=0, temp_folder=None, use_cache=True)
```

Calculate the kBET metric of the data regarding a specific sample attribute and embedding.

The kBET metric is defined in [Büttner18], which measures if cells from different samples mix well in their local neighborhood.

**Parameters**

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **attr** (*str*) – The sample attribute to consider. Must exist in `data.obs`.
- **rep** (*str*, optional, default: "pca") – The embedding representation to be used. The key 'X\_' + `rep` must exist in `data.obsm`. By default, use PCA coordinates.
- **K** (*int*, optional, default: 25) – Number of nearest neighbors, using L2 metric.
- **alpha** (*float*, optional, default: 0.05) – Acceptance rate threshold. A cell is accepted if its kBET p-value is greater than or equal to `alpha`.
- **n\_jobs** (*int*, optional, default: -1) – Number of threads used. If -1, use all physical CPU cores.
- **random\_state** (*int*, optional, default: 0) – Random seed set for reproducing results.
- **temp\_folder** (*str*, optional, default: None) – Temporary folder for joblib execution.

- **use\_cache** (bool, optional, default: True) – If use cache results for kNN.

**Return type**

Tuple[float, float, float]

**Returns**

- **stat\_mean** (float) – Mean kBET chi-square statistic over all cells.
- **pvalue\_mean** (float) – Mean kBET p-value over all cells.
- **accept\_rate** (float) – kBET Acceptance rate of the sample.

**Examples**

```
>>> pg.calc_kBET(data, attr = 'Channel')
```

```
>>> pg.calc_kBET(data, attr = 'Channel', rep = 'umap')
```

**pegasus.calc\_kSIM**

`pegasus.calc_kSIM(data, attr, rep='pca', K=25, min_rate=0.9, n_jobs=-1, random_state=0, use_cache=True)`

Calculate the kSIM metric of the data regarding a specific sample attribute and embedding.

The kSIM metric is defined in [Li20], which measures if a sample attribute is not diffused too much in each cell's local neighborhood.

**Parameters**

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **attr** (str) – The sample attribute to consider. Must exist in `data.obs`.
- **rep** (str, optional, default: "pca") – The embedding representation to consider. The key 'X\_' + rep must exist in `data.obsm`.
- **K** (int, optional, default: 25) – The number of nearest neighbors to be considered.
- **min\_rate** (float, optional, default: 0.9) – Acceptance rate threshold. A cell is accepted if its kSIM rate is larger than or equal to `min_rate`.
- **n\_jobs** (int, optional, default: -1) – Number of threads used. If -1, use all physical CPU cores.
- **random\_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **use\_cache** (bool, optional, default: True) – If use cache results for kNN.

**Return type**

Tuple[float, float]

**Returns**

- **kSIM\_mean** (float) – Mean kSIM rate over all the cells.
- **kSIM\_accept\_rate** (float) – kSIM Acceptance rate of the sample.

## Examples

```
>>> pg.calc_kSIM(data, attr = 'cell_type')
```

```
>>> pg.calc_kSIM(data, attr = 'cell_type', rep = 'umap')
```

## Diffusion Map

<code>diffmap(data[, n_components, rep, solver, ...])</code>	Calculate Diffusion Map.
<code>calc_pseudotime(data, roots)</code>	Calculate Pseudotime based on Diffusion Map.
<code>infer_path(data, cluster, clust_id, path_name)</code>	Inference on path of a cluster.

## pegasus.diffmap

`pegasus.diffmap(data, n_components=100, rep='pca', solver='eigsh', max_t=5000, n_jobs=-1, random_state=0)`  
Calculate Diffusion Map.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **n\_components** (`int`, optional, default: 100) – Number of diffusion components to calculate.
- **rep** (`str`, optional, default: "pca") – Embedding Representation of data used for calculating the Diffusion Map. By default, use PCA coordinates.
- **solver** (`str`, optional, default: "eigsh") –

#### Solver for eigen decomposition:

- "eigsh": default setting. Use *scipy* `eigsh` as the solver to find eigenvalue and eigenvectors using the Implicitly Restarted Lanczos Method.
- "randomized": Use *scikit-learn* `randomized_svd` as the solver to calculate a truncated randomized SVD.
- **max\_t** (`float`, optional, default: 5000) – pegasus tries to determine the best t to sum up to between [1, max\_t].
- **n\_jobs** (`int`, optional (default: -1)) – Number of threads to use. -1 refers to using all physical CPU cores.
- **random\_state** (`int`, optional, default: 0) – Random seed set for reproducing results.

### Return type

None

### Returns

- None
- Update `data.obsm` –
  - `data.obsm["X_diffmap"]`: Diffusion Map matrix of the data.
- Update `data.uns` –
  - `data.uns["diffmap_evals"]`: Eigenvalues corresponding to Diffusion Map matrix.

## Examples

```
>>> pg.diffmap(data)
```

## pegasus.calc\_pseudotime

`pegasus.calc_pseudotime(data, roots)`

Calculate Pseudotime based on Diffusion Map.

### Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.
- **roots** (`List[str]`) – List of cell barcodes in the data.

### Return type

None

### Returns

- None
- Update `data.obs` –
  - `data.obs["pseudotime"]`: Pseudotime result.

## Examples

```
>>> pg.calc_pseudotime(adata, roots = list(adata.obs_names[0:100]))
```

## pegasus.infer\_path

`pegasus.infer_path(data, cluster, clust_id, path_name, k=10)`

Inference on path of a cluster.

### Parameters

- **data** (`anndata.AnnData`) – Annotated data matrix with rows for cells and columns for genes.
- **cluster** (`str`) – Cluster name. Must exist in `data.obs`.
- **clust\_id** – Cluster label. Must be a value of `data.obs[cluster]`.
- **path\_name** (`str`) – Key name of the resulting path information.
- **k** (`int`, optional, default: 10) – Number of nearest neighbors on Diffusion Map coordinates used in path reference.

### Returns

- None
- Update `data.obs` –
  - `data.obs[path_name]`: The inferred path information on Diffusion Map about a specific cluster.



## Examples

```
>>> pg.infer_path(adata, cluster = 'leiden_labels', clust_id = '1', path_name =
↳ 'leiden_1_path')
```

## Cluster Algorithms

<code>cluster(data[, algo, rep, resolution, ...])</code>	Cluster the data using the chosen algorithm.
<code>louvain(data[, rep, resolution, n_clust, ...])</code>	Cluster the cells using Louvain algorithm.
<code>leiden(data[, rep, resolution, n_clust, ...])</code>	Cluster the data using Leiden algorithm.
<code>split_one_cluster(data, clust_label, ...[, ...])</code>	Use Leiden algorithm to split 'clust_id' in 'clust_label' into 'n_components' sub-clusters and write the new clustering results to 'res_label'.
<code>spectral_louvain(data[, rep, resolution, ...])</code>	Cluster the data using Spectral Louvain algorithm.
<code>spectral_leiden(data[, rep, resolution, ...])</code>	Cluster the data using Spectral Leiden algorithm.

## pegasus.cluster

```
pegasus.cluster(data, algo='louvain', rep='pca', resolution=1.3, n_jobs=-1, random_state=0,
class_label=None, n_iter=-1, rep_kmeans='diffmap', n_clusters=30, n_clusters2=50,
n_init=10)
```

Cluster the data using the chosen algorithm.

Candidates are *louvain*, *leiden*, *spectral\_louvain* and *spectral\_leiden*. If data have < 1000 cells and there are clusters with sizes of 1, resolution is automatically reduced until no cluster of size 1 appears.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **algo** (str, optional, default: "louvain") – Which clustering algorithm to use. Choices are *louvain*, *leiden*, *spectral\_louvain*, *spectral\_leiden*
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X\_' + rep must exist in `data.obsm`. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters.
- **n\_jobs** (int, optional (default: -1)) – Number of threads to use for the KMeans step in 'spectral\_louvain' and 'spectral\_leiden'. -1 refers to using all physical CPU cores.
- **random\_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class\_label** (str, optional, default: None) – Key name for storing cluster labels in `data.obs`. If None, use 'algo\_labels'.
- **n\_iter** (int, optional, default: -1) – Number of iterations that Leiden algorithm runs. If -1, run the algorithm until reaching its optimal clustering.
- **rep\_kmeans** (str, optional, default: "diffmap") – The embedding representation on which the KMeans runs. Keyword must exist in `data.obsm`. By default, use Diffusion Map coordinates. If *diffmap* is not calculated, use PCA coordinates instead.
- **n\_clusters** (int, optional, default: 30) – The number of first level clusters.

- **n\_clusters2** (int, optional, default: 50) – The number of second level clusters.
- **n\_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.

#### Return type

None

#### Returns

- None
- Update `data.obs` –
  - `data.obs[class_label]`: Cluster labels of cells as categorical data.

### Examples

```
>>> pg.cluster(data, algo = 'leiden')
```

### pegasus.louvain

`pegasus.louvain(data, rep='pca', resolution=1.3, n_clust=None, random_state=0, class_label='louvain_labels')`

Cluster the cells using Louvain algorithm. [Blondel08]

#### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X\_' + **rep** must exist in `data.obsm` and nearest neighbors must be calculated so that affinity matrix 'W\_' + **rep** exists in `data.uns`. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters with smaller sizes.
- **n\_clust** (int, optional, default: None) – This option only takes effect if 'resolution = None'. Try to find an appropriate resolution by binary search such that the total number of clusters matches 'n\_clust'. The range of resolution to search is (0.01, 2.0].
- **random\_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class\_label** (str, optional, default: "louvain\_labels") – Key name for storing cluster labels in `data.obs`.

#### Return type

None

#### Returns

- None
- Update `data.obs` –
  - `data.obs[class_label]`: Cluster labels of cells as categorical data.

## Examples

```
>>> pg.louvain(data)
```

## pegasus.leiden

`pegasus.leiden(data, rep='pca', resolution=1.3, n_clust=None, n_iter=-1, random_state=0, class_label='leiden_labels')`

Cluster the data using Leiden algorithm. [Traag19]

### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X\_' + **rep** must exist in `data.obsm` and nearest neighbors must be calculated so that affinity matrix 'W\_' + **rep** exists in `data.uns`. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters.
- **n\_clust** (int, optional, default: None) – This option only takes effect if 'resolution = None'. Try to find an appropriate resolution by binary search such that the total number of clusters matches 'n\_clust'. The range of resolution to search is (0.01, 2.0].
- **n\_iter** (int, optional, default: -1) – Number of iterations that Leiden algorithm runs. If -1, run the algorithm until reaching its optimal clustering.
- **random\_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class\_label** (str, optional, default: "leiden\_labels") – Key name for storing cluster labels in `data.obs`.

### Return type

None

### Returns

- None
- Update `data.obs` –
  - `data.obs[class_label]`: Cluster labels of cells as categorical data.

## Examples

```
>>> pg.leiden(data)
```

## pegasus.split\_one\_cluster

`pegasus.split_one_cluster(data, clust_label, clust_id, n_clust, res_label, rep='pca', n_comps=None, random_state=0)`

Use Leiden algorithm to split 'clust\_id' in 'clust\_label' into 'n\_components' sub-clusters and write the new clustering results to 'res\_label'. The sub-cluster names are the concatenation of original cluster name and the subcluster id (e.g. 'T' -> 'T-1', 'T-2').

### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **clust\_label** (*str*) – Use existing clustering stored in `data.obs['clust_label']`.
- **clust\_id** (*str*) – Cluster ID in `data.obs['clust_label']`.
- **n\_clust** (*int*) – Split 'clust\_id' into 'n\_clust' subclusters.
- **res\_label** (*str*,) – Write new clustering in `data.obs['res_label']`. The sub-cluster names are the concatenation of original cluster name and the subcluster id (e.g. 'T' -> 'T-1', 'T-2').
- **rep** (*str*, optional, default: "pca") – The embedding representation used for Kmeans clustering. Keyword 'X\_' + rep must exist in `data.obsm`. By default, use PCA coordinates.
- **n\_comps** (*int*, optional (default: None)) – Number of components to be used in the rep. If `n_comps == None`, use all components; otherwise, use the minimum of `n_comps` and rep's dimensions.
- **n\_jobs** (*int*, optional (default: -1)) – Number of threads to use for the KMeans step in 'spectral\_louvain' and 'spectral\_leiden'. -1 refers to using all physical CPU cores.
- **random\_state** (*int*, optional, default: 0) – Random seed for reproducing results.

### Return type

None

### Returns

- None
- Update `data.obs` –
  - `data.obs[res_label]`: New cluster labels of cells as categorical data.

## Examples

```
>>> pg.split_one_cluster(data, 'leiden_labels', '15', 2, 'leiden_labels_split')
```

## pegasus.spectral\_louvain

```
pegasus.spectral_louvain(data, rep='pca', resolution=1.3, rep_kmeans='diffmap', n_clusters=30,
                        n_clusters2=50, n_init=10, n_jobs=-1, random_state=0,
                        class_label='spectral_louvain_labels')
```

Cluster the data using Spectral Louvain algorithm. [Li20]

### Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X\_' + rep must exist in data.obsm. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters with smaller sizes.
- **rep\_kmeans** (str, optional, default: "diffmap") – The embedding representation on which the KMeans runs. Keyword must exist in data.obsm. By default, use Diffusion Map coordinates. If diffmap is not calculated, use PCA coordinates instead.
- **n\_clusters** (int, optional, default: 30) – The number of first level clusters.
- **n\_clusters2** (int, optional, default: 50) – The number of second level clusters.
- **n\_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.
- **n\_jobs** (int, optional (default: -1)) – Number of threads to use for the KMeans step. -1 refers to using all physical CPU cores.
- **random\_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class\_label** (str, optional, default: "spectral\_louvain\_labels") – Key name for storing cluster labels in data.obs.

### Return type

None

### Returns

- None
- Update data.obs –
  - data.obs[class\_label]: Cluster labels for cells as categorical data.

### Examples

```
>>> pg.spectral_louvain(data)
```

## pegasus.spectral\_leiden

```
pegasus.spectral_leiden(data, rep='pca', resolution=1.3, rep_kmeans='diffmap', n_clusters=30,  
                        n_clusters2=50, n_init=10, n_jobs=-1, random_state=0,  
                        class_label='spectral_leiden_labels')
```

Cluster the data using Spectral Leiden algorithm. [Li20]

### Parameters

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (str, optional, default: "pca") – The embedding representation used for clustering. Keyword 'X\_' + rep must exist in data.obsm. By default, use PCA coordinates.
- **resolution** (int, optional, default: 1.3) – Resolution factor. Higher resolution tends to find more clusters.
- **rep\_kmeans** (str, optional, default: "diffmap") – The embedding representation on which the KMeans runs. Keyword must exist in data.obsm. By default, use Diffusion Map coordinates. If diffmap is not calculated, use PCA coordinates instead.
- **n\_clusters** (int, optional, default: 30) – The number of first level clusters.
- **n\_clusters2** (int, optional, default: 50) – The number of second level clusters.
- **n\_init** (int, optional, default: 10) – Number of kmeans tries for the first level clustering. Default is set to be the same as scikit-learn Kmeans function.
- **n\_jobs** (int, optional (default: -1)) – Number of threads to use for the KMeans step. -1 refers to using all physical CPU cores.
- **random\_state** (int, optional, default: 0) – Random seed for reproducing results.
- **class\_label** (str, optional, default: "spectral\_leiden\_labels") – Key name for storing cluster labels in data.obs.

### Return type

None

### Returns

- None
- Update data.obs –
  - data.obs[class\_label]: Cluster labels for cells as categorical data.

## Examples

```
>>> pg.spectral_leiden(data)
```

## Visualization Algorithms

<code>tsne(data[, rep, rep_ncomps, n_jobs, ...])</code>	Calculate t-SNE embedding of cells using the FIt-SNE package.
<code>umap(data[, rep, rep_ncomps, n_components, ...])</code>	Calculate UMAP embedding of cells.
<code>fle(data[, file_name, n_jobs, rep, ...])</code>	Construct the Force-directed (FLE) graph.
<code>net_umap(data[, rep, n_jobs, n_components, ...])</code>	Calculate Net-UMAP embedding of cells.
<code>net_fle(data[, file_name, n_jobs, rep, K, ...])</code>	Construct Net-Force-directed (FLE) graph.

### pegasus.tsne

```
pegasus.tsne(data, rep='pca', rep_ncomps=None, n_jobs=-1, n_components=2, perplexity=30,
              early_exaggeration=12, learning_rate='auto', initialization='pca', random_state=0,
              out_basis='tsne')
```

Calculate t-SNE embedding of cells using the FIt-SNE package.

This function uses `fitsne` package. See [Linderman19] for details on FIt-SNE algorithm.

#### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (`str`, optional, default: "pca") – Representation of data used for the calculation. By default, use PCA coordinates. If `None`, use the count matrix `data.X`.
- **rep\_ncomps** (`int`, optional (default: `None`)) – Number of components to be used in `rep`. If `rep_ncomps == None`, use all components; otherwise, use the minimum of `rep_ncomps` and `rep`'s dimensions.
- **n\_jobs** (`int`, optional, default: -1) – Number of threads to use. If -1, use all physical CPU cores.
- **n\_components** (`int`, optional, default: 2) – Dimension of calculated FI-tSNE coordinates. By default, generate 2-dimensional data for 2D visualization.
- **perplexity** (`float`, optional, default: 30) – The perplexity is related to the number of nearest neighbors used in other manifold learning algorithms. Larger datasets usually require a larger perplexity.
- **early\_exaggeration** (`int`, optional, default: 12) – Controls how tight natural clusters in the original space are in the embedded space, and how much space will be between them.
- **learning\_rate** (`float`, optional, default: `auto`) – By default, the learning rate is determined automatically as `max(data.shape[0] / early_exaggeration, 200)`. See [Belkina19] and [Kobak19] for details.
- **initialization** (`str`, optional, default: `pca`) – Initialization can be either `pca` or `random` or `np.ndarray`. By default, we use `pca` initialization according to [Kobak19].
- **random\_state** (`int`, optional, default: 0) – Random seed set for reproducing results.
- **out\_basis** (`str`, optional, default: "fitsne") – Key name for calculated FI-tSNE coordinates to store.

#### Return type

`None`

#### Returns

- None
- Update `data.obsm` –
  - `data.obsm['X_' + out_basis]`: FI-tSNE coordinates of the data.

## Examples

```
>>> pg.tsne(data)
```

## pegasus.umap

```
pegasus.umap(data, rep='pca', rep_ncomps=None, n_components=2, n_neighbors=15, min_dist=0.5, spread=1.0,  
             densmap=False, dens_lambda=2.0, dens_frac=0.3, dens_var_shift=0.1, n_jobs=-1,  
             full_speed=False, use_cache=True, random_state=0, out_basis='umap')
```

Calculate UMAP embedding of cells.

This function uses [umap-learn](#) package. See [\[McInnes18\]](#) for details on UMAP.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (`str`, optional, default: "pca") – Representation of data used for the calculation. By default, use PCA coordinates. If `None`, use the count matrix `data.X`.
- **rep\_ncomps** (`int`, optional (default: `None`)) – Number of components to be used in `rep`. If `rep_ncomps == None`, use all components; otherwise, use the minimum of `rep_ncomps` and `rep`'s dimensions.
- **n\_components** (`int`, optional, default: 2) – Dimension of calculated UMAP coordinates. By default, generate 2-dimensional data for 2D visualization.
- **n\_neighbors** (`int`, optional, default: 15) – Number of nearest neighbors considered during the computation.
- **min\_dist** (`float`, optional, default: 0.5) – The effective minimum distance between embedded data points.
- **spread** (`float`, optional, default: 1.0) – The effective scale of embedded data points.
- **densmap** (`bool`, optional, default: `False`) – Whether the density-augmented objective of densMAP should be used for optimization, which will generate an embedding where local densities are encouraged to be correlated with those in the original space.
- **dens\_lambda** (`float`, optional, default: 2.0) – Controls the regularization weight of the density correlation term in densMAP. Only works when `densmap` is `True`. Larger values prioritize density preservation over the UMAP objective, while values closer to 0 for the opposite direction. Notice that setting this parameter to 0 is equivalent to running the original UMAP algorithm.
- **dens\_frac** (`float`, optional, default: 0.3) – Controls the fraction of epochs (between 0 and 1) where the density-augmented objective is used in densMAP. Only works when `densmap` is `True`. The first  $(1 - \text{dens\_frac})$  fraction of epochs optimize the original UMAP objective before introducing the density correlation term.



- **dens\_var\_shift** (float, optional, default: 0.1) – A small constant added to the variance of local radii in the embedding when calculating the density correlation objective to prevent numerical instability from dividing by a small number. Only works when *densmap* is True.
- **n\_jobs** (int, optional, default: -1) – Number of threads to use for computing kNN graphs. If -1, use all physical CPU cores.
- **full\_speed** (bool, optional, default: False) –
  - If True, use multiple threads in constructing *hns*w index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **use\_cache** (bool, optional, default: True) – If use\_cache and found cached knn results, will not recompute.
- **random\_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **out\_basis** (str, optional, default: "umap") – Key name for calculated UMAP coordinates to store.

**Return type**

None

**Returns**

- None
- Update `data.obsm` –
  - `data.obsm['X_' + out_basis]`: UMAP coordinates of the data.

**Examples**

```
>>> pg.umap(data)
```

**pegasus.fle**

```
pegasus.fle(data, file_name=None, n_jobs=-1, rep='diffmap', rep_ncomps=None, K=50, full_speed=False,
            target_change_per_node=2.0, target_steps=5000, is3d=False, memory=8, random_state=0,
            out_basis='fle')
```

Construct the Force-directed (FLE) graph.

This implementation uses [forceatlas2-python](#) package, which is a Python wrapper of [ForceAtlas2](#).

See [\[Jacomy14\]](#) for details on FLE.

**Parameters**

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **file\_name** (str, optional, default: None) – Temporary file to store the coordinates as the input to *forceatlas2*. If None, use `tempfile.mkstemp` to generate file name.
- **n\_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all physical CPU cores.
- **rep** (str, optional, default: "diffmap") – Representation of data used for the calculation. By default, use Diffusion Map coordinates. If None, use the count matrix `data.X`.

- **rep\_ncomps** (int, optional (default: None)) – Number of components to be used in *rep*. If `rep_ncomps == None`, use all components; otherwise, use the minimum of `rep_ncomps` and *rep*'s dimensions.
- **K** (int, optional, default: 50) – Number of nearest neighbors to be considered during the computation.
- **full\_speed** (bool, optional, default: False) –
  - If True, use multiple threads in constructing `hnsw` index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **target\_change\_per\_node** (float, optional, default: 2.0) – Target change per node to stop ForceAtlas2.
- **target\_steps** (int, optional, default: 5000) – Maximum number of iterations before stopping the ForceAtlas2 algorithm.
- **is3d** (bool, optional, default: False) – If True, calculate 3D force-directed layout.
- **memory** (int, optional, default: 8) – Memory size in GB for the Java FA2 component. By default, use 8GB memory.
- **random\_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **out\_basis** (str, optional, default: "fle") – Key name for calculated FLE coordinates to store.

**Return type**

None

**Returns**

- None
- Update `data.obsm` –
  - `data.obsm['X_' + out_basis]`: FLE coordinates of the data.

**Examples**

```
>>> pg.fle(data)
```

**pegasus.net\_umap**

```
pegasus.net_umap(data, rep='pca', n_jobs=-1, n_components=2, n_neighbors=15, min_dist=0.5, spread=1.0,
                  densmap=False, dens_lambda=2.0, dens_frac=0.3, dens_var_shift=0.1, random_state=0,
                  select_frac=0.1, select_K=25, select_alpha=1.0, full_speed=False, use_cache=True,
                  net_alpha=0.1, polish_learning_rate=10.0, polish_n_epochs=30, out_basis='net_umap')
```

Calculate Net-UMAP embedding of cells.

Net-UMAP is an approximated UMAP embedding using Deep Learning model to improve the speed.

In specific, the deep model used is [MLPRegressor](#), the *scikit-learn* implementation of Multi-layer Perceptron regressor.

See [\[Li20\]](#) for details.

**Parameters**

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **rep** (`str`, optional, default: "pca") – Representation of data used for the calculation. By default, use PCA coordinates. If `None`, use the count matrix `data.X`.
- **n\_jobs** (`int`, optional, default: -1) – Number of threads to use. If -1, use all physical CPU cores.
- **n\_components** (`int`, optional, default: 2) – Dimension of calculated UMAP coordinates. By default, generate 2-dimensional data for 2D visualization.
- **n\_neighbors** (`int`, optional, default: 15) – Number of nearest neighbors considered during the computation.
- **min\_dist** (`float`, optional, default: 0.5) – The effective minimum distance between embedded data points.
- **spread** (`float`, optional, default: 1.0) – The effective scale of embedded data points.
- **densmap** (`bool`, optional, default: False) – Whether the density-augmented objective of densMAP should be used for optimization, which will generate an embedding where local densities are encouraged to be correlated with those in the original space.
- **dens\_lambda** (`float`, optional, default: 2.0) – Controls the regularization weight of the density correlation term in densMAP. Only works when *densmap* is `True`. Larger values prioritize density preservation over the UMAP objective, while values closer to 0 for the opposite direction. Notice that setting this parameter to 0 is equivalent to running the original UMAP algorithm.
- **dens\_frac** (`float`, optional, default: 0.3) – Controls the fraction of epochs (between 0 and 1) where the density-augmented objective is used in densMAP. Only works when *densmap* is `True`. The first  $(1 - \text{dens\_frac})$  fraction of epochs optimize the original UMAP objective before introducing the density correlation term.
- **dens\_var\_shift** (`float`, optional, default, 0.1) – A small constant added to the variance of local radii in the embedding when calculating the density correlation objective to prevent numerical instability from dividing by a small number. Only works when *densmap* is `True`.
- **random\_state** (`int`, optional, default: 0) – Random seed set for reproducing results.
- **select\_frac** (`float`, optional, default: 0.1) – Down sampling fraction on the cells.
- **select\_K** (`int`, optional, default: 25) – Number of neighbors to be used to estimate local density for each data point for down sampling.
- **select\_alpha** (`float`, optional, default: 1.0) – Weight the down sample to be proportional to  $\text{radius}^{**} \text{select\_alpha}$ .
- **full\_speed** (`bool`, optional, default: False) –
  - If `True`, use multiple threads in constructing `hns` index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **use\_cache** (`bool`, optional, default: True) – If `use_cache` and found cached knn results, will not recompute.
- **net\_alpha** (`float`, optional, default: 0.1) – L2 penalty (regularization term) parameter of the deep regressor.

- **polish\_learning\_frac** (float, optional, default: 10.0) – After running the deep regressor to predict new coordinates, use `polish_learning_frac * n_obs` as the learning rate to polish the coordinates.
- **polish\_n\_iter** (int, optional, default: 30) – Number of iterations for polishing UMAP run.
- **out\_basis** (str, optional, default: "net\_umap") – Key name for calculated UMAP coordinates to store.

**Return type**

None

**Returns**

- None
- Update `data.obsm` –
  - `data.obsm['X_' + out_basis]`: Net UMAP coordinates of the data.
- Update `data.obs` –
  - `data.obs['ds_selected']`: Boolean array to indicate which cells are selected during the down sampling phase.

**Examples**

```
>>> pg.net_umap(data)
```

**pegasus.net\_fle**

```
pegasus.net_fle(data, file_name=None, n_jobs=-1, rep='diffmap', K=50, full_speed=False, use_cache=True,
target_change_per_node=2.0, target_steps=5000, is3d=False, memory=8, random_state=0,
select_frac=0.1, select_K=25, select_alpha=1.0, net_alpha=0.1, polish_target_steps=1500,
out_basis='net_fle')
```

Construct Net-Force-directed (FLE) graph.

Net-FLE is an approximated FLE graph using Deep Learning model to improve the speed.

In specific, the deep model used is [MLPRegressor](#), the *scikit-learn* implementation of Multi-layer Perceptron regressor.

See [\[Li20\]](#) for details.

**Parameters**

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **file\_name** (str, optional, default: None) – Temporary file to store the coordinates as the input to `forceatlas2`. If None, use `tempfile.mkstemp` to generate file name.
- **n\_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all physical CPU cores.
- **rep** (str, optional, default: "diffmap") – Representation of data used for the calculation. By default, use Diffusion Map coordinates. If None, use the count matrix `data.X`.

- **K** (int, optional, default: 50) – Number of nearest neighbors to be considered during the computation.
- **full\_speed** (bool, optional, default: False) –
  - If True, use multiple threads in constructing `hns` index. However, the kNN results are not reproducible.
  - Otherwise, use only one thread to make sure results are reproducible.
- **use\_cache** (bool, optional, default: True) – If `use_cache` and found cached knn results, will not recompute.
- **target\_change\_per\_node** (float, optional, default: 2.0) – Target change per node to stop ForceAtlas2.
- **target\_steps** (int, optional, default: 5000) – Maximum number of iterations before stopping the ForceAtlas2 algorithm.
- **is3d** (bool, optional, default: False) – If True, calculate 3D force-directed layout.
- **memory** (int, optional, default: 8) – Memory size in GB for the Java FA2 component. By default, use 8GB memory.
- **random\_state** (int, optional, default: 0) – Random seed set for reproducing results.
- **select\_frac** (float, optional, default: 0.1) – Down sampling fraction on the cells.
- **select\_K** (int, optional, default: 25) – Number of neighbors to be used to estimate local density for each data point for down sampling.
- **select\_alpha** (float, optional, default: 1.0) – Weight the down sample to be proportional to `radius ** select_alpha`.
- **net\_alpha** (float, optional, default: 0.1) – L2 penalty (regularization term) parameter of the deep regressor.
- **polish\_target\_steps** (int, optional, default: 1500) – After running the deep regressor to predict new coordinate, Number of ForceAtlas2 iterations.
- **out\_basis** (str, optional, default: "net\_fle") – Key name for calculated FLE coordinates to store.

**Return type**

None

**Returns**

- None
- Update `data.obsm` –
  - `data.obsm['X_' + out_basis]`: Net FLE coordinates of the data.
- Update `data.obs` –
  - `data.obs['ds_selected']`: Boolean array to indicate which cells are selected during the down sampling phase.

## Examples

```
>>> pg.net_fle(data)
```

## Doublet Detection

<code><i>infer_doublets</i>(data[, channel_attr, ...])</code>	Infer doublets by first calculating Scrublet-like [Wolock18] doublet scores and then smartly determining an appropriate doublet score cutoff [Li20-2] .
<code><i>mark_doublets</i>(data[, demux_attr, dbl_clusts])</code>	Convert doublet prediction into doublet annotations that Pegasus can recognize.

## pegasus.infer\_doublets

```
pegasus.infer_doublets(data, channel_attr=None, clust_attr=None, raw_mat_key=None, min_cell=100,  
                        expected_doublet_rate=None, sim_doublet_ratio=2.0, n_prin_comps=30, k=None,  
                        n_jobs=-1, alpha=0.05, random_state=0, plot_hist='sample',  
                        manual_correction=None)
```

Infer doublets by first calculating Scrublet-like [Wolock18] doublet scores and then smartly determining an appropriate doublet score cutoff [Li20-2] .

This function should be called after clustering if `clust_attr` is not `None`. In this case, we will test if each cluster is significantly enriched for doublets using Fisher’s exact test.

### Parameters

- **data** (`pegasusio.MultimodalData`) – Annotated data matrix with rows for cells and columns for genes.
- **channel\_attr** (`str`, optional, default: `None`) – Attribute indicating sample channels. If set, calculate scrublet-like doublet scores per channel.
- **clust\_attr** (`str`, optional, default: `None`) – Attribute indicating cluster labels. If set, estimate proportion of doublets in each cluster and statistical significance.
- **raw\_mat\_key** (`str`, optional, default: `None`) – The key for raw count matrix. By default, Pegasus will first try “counts” and then try “raw.X”
- **min\_cell** (`int`, optional, default: 100) – Minimum number of cells per sample to calculate doublet scores. For samples having less than ‘min\_cell’ cells, doublet score calculation will be skipped.
- **expected\_doublet\_rate** (`float`, optional, default: `None`) – The expected doublet rate for the experiment. By default, calculate the expected rate based on number of cells from the 10x multiplet rate table
- **sim\_doublet\_ratio** (`float`, optional, default: 2.0) – The ratio between synthetic doublets and observed cells.
- **n\_prin\_comps** (`int`, optional, default: 30) – Number of principal components.
- **k** (`int`, optional, default: `None`) – Number of observed cell neighbors. If `None`,  $k = \text{round}(0.5 * \sqrt{\text{number of observed cells}})$ . Total neighbors  $k_{\text{adj}} = \text{round}(k * (1.0 + \text{sim\_doublet\_ratio}))$ .

- **n\_jobs** (int, optional, default: -1) – Number of threads to use. If -1, use all physical CPU cores.
- **alpha** (float, optional, default: 0.05) – FDR significant level for cluster-level fisher exact test.
- **random\_state** (int, optional, default: 0) – Random seed for reproducing results.
- **plot\_hist** (str, optional, default: sample) – If not None, plot diagnostic histograms using plot\_hist as the prefix. If channel\_attr is None, plot\_hist.dbl.png is generated; Otherwise, plot\_hist.channel\_name.dbl.png files are generated. Each figure consists of 4 panels showing histograms of doublet scores for observed cells (panel 1, density in log scale), simulated doublets (panel 2, density in log scale), KDE plot (panel 3) and signed curvature plot (panel 4) of log doublet scores for simulated doublets. Each plot contains two dashed lines. The red dashed line represents the theoretical cutoff (calculated based on number of cells and 10x doublet table) and the black dashed line represents the cutoff inferred from the data.
- **manual\_correction** (str, optional, default: None) – Use human guide to correct doublet threshold for certain channels. This is string representing a comma-separated list. Each item in the list represents one sample and the sample name and correction guide are separated using ':'. The correction guides supported are 'peak', 'expected' and threshold. 'peak' means cutting at the center of the peak; 'expected' means cutting at the expected doublet rate; threshold is the user-specified doublet threshold; if the guide is neither 'peak' nor 'expected', pegasus will try to convert the string into float and use it as doublet threshold. If only one sample is available, no need to specify sample name.

**Return type**

None

**Returns**

- None
- Update data.obs –
  - data.obs['pred\_dbl']: Predicted singlet/doublet types.
  - data.uns['pred\_dbl\_cluster']: Only generated if 'clust\_attr' is not None. This is a dataframe with two columns, 'Cluster' and 'Qval'. Only clusters with significantly more doublets than expected will be recorded here.

**Examples**

```
>>> pg.infer_doublets(data, channel_attr = 'Channel', clust_attr = 'Annotation')
```

**pegasus.mark\_doublets**

pegasus.mark\_doublets(data, demux\_attr='demux\_type', dbl\_clusts=None)

Convert doublet prediction into doublet annotations that Pegasus can recognize. In addition, clusters in dbl\_clusts will be marked as doublets.

Must run infer\_doublets first.

**Parameters**

- **data** (pegasusio.MultimodalData) – Annotated data matrix with rows for cells and columns for genes.

- **demux\_attr** (str, optional, default: `demux_type`) – Attribute indicating singlets/doublets that Pegasus can recognize. Currently this is ‘`demux_type`’, which is also used for hashing.
- **dbl\_clusts** (str, optional, default: `None`) – Indicate which clusters should be marked as all doublets. It takes the format of ‘`clust:value1,value2,...`’, where ‘`clust`’ refers to the cluster attribute.

**Return type**

`None`

**Returns**

- `None`
- Update `data.obs` –
  - `data.obs[demux_attr]`: Singlet/doublet annotation.

**Examples**

```
>>> pg.mark_doublets(data, dbl_clusts='Annotation:B/T doublets')
```

**Gene Module Score**

---

<code>calc_signature_score(data, signatures[, ...])</code>	Calculate signature / gene module score.
--	--

---

**pegasus.calc\_signature\_score**

`pegasus.calc_signature_score(data, signatures, n_bins=50, show_omitted_genes=False, skip_threshold=1, random_state=0)`

Calculate signature / gene module score. [Li20-1]

This is an improved version of implementation in [Jerby-Arnon18].

**Parameters**

- **data** (`MultimodalData`, `UnimodalData`, or `anndata.AnnData` object.) – Single cell expression data.
- **signatures** (`Dict[str, List[str]]` or `str`) – This argument accepts either a dictionary or a string. If `signatures` is a dictionary, it can contain multiple signature score calculation requests. Each key in the dictionary represents a separate signature score calculation and its corresponding value contains a list of gene symbols. Each score will be stored in `data.obs` field with key as the keyword. If `signatures` is a string, it should refer to a Gene Matrix Transposed (GMT)-formatted file. Pegasus will load signatures from the GMT file.

Pegasus also provide 5 default signature panels for each of human and mouse. They are `cell_cycle_human`, `gender_human`, `mitochondrial_genes_human`, `ribosomal_genes_human` and `apoptosis_human` for human; `cell_cycle_mouse`, `gender_mouse`, `mitochondrial_genes_mouse`, `ribosomal_genes_mouse` and `apoptosis_mouse` for mouse.

- `cell_cycle_human` contains two cell-cycle signatures, G1/S and G2/M, obtained from Tirosh et al. 2016. We also updated gene symbols according to Seurat’s `cc.genes.updated.2019` vector. We additionally calculate signature scores cycling



and `cycle_diff`, which are  $\max\{G2/M, G1/S\}$  and  $G2/M - G1/S$  respectively. We provide predicted cell cycle phases in `data.obs['predicted_phase']` in case it is useful. `predicted_phase` is predicted based on `G1/S` and `G2/M` scores. First, we identify `G0` cells. We apply KMeans algorithm to obtain 2 clusters based on the cycling signature. `G0` cells are from the cluster with smallest mean value. For each cell from the other cluster, if  $G1/S > G2/M$ , it is a `G1/S` cell, otherwise it is a `G2/M` cell.

- `gender_human` contains two gender-specific signatures, `female_score` and `male_score`. Genes were selected based on DE analysis between genders based on 8 channels of bone marrow data from HCA Census of Immune Cells and the brain nuclei data from Gaublot and Li et al, 2019, Nature Communications. After calculation, three signature scores will be calculated: `female_score`, `male_score` and `gender_score`. `female_score` and `male_score` are calculated based on female and male signatures respectively and a larger score represent a higher likelihood of that gender. `gender_score` is calculated as `male_score - female_score`. A large positive score likely represents male and a large negative score likely represents female. Pegasus also provides predicted gender for each cell based on `gender_score`, which is stored in `data.obs['predicted_gender']`. To predict genders, we apply the KMeans algorithm to the `gender_score` and ask for 3 clusters. The clusters with a minimum and maximum cluster centers are predicted as `female` and `male` respectively and the cluster in the middle is predicted as `uncertain`. Note that this approach is conservative and it is likely that users can predict genders based on `gender_score` for cells in the `uncertain` cluster with a reasonable accuracy.
- `mitochondrial_genes_human` contains two signatures, `mito_genes` and `mito_ribo`. `mito_genes` contains 13 mitochondrial genes from chrM and `mito_ribo` contains mitochondrial ribosomal genes that are not from chrM. Note that `mito_genes` correlates well with percent of mitochondrial UMIs and `mito_ribo` does not.
- `ribosomal_genes_human` contains one signature, `ribo_genes`, which includes ribosomal genes from both large and small units.
- `apoptosis_human` contains one signature, `apoptosis`, which includes apoptosis-related genes from the KEGG pathway.
- `cell_cycle_mouse`, `gender_mouse`, `mitochondrial_genes_mouse`, `ribosomal_genes_mouse` and `apoptosis_mouse` are the corresponding signatures for mouse. Gene symbols are directly translated from human genes.

In addition, Pegasus provides the following 4 curated signature panels:

- `emt_human`, the Epithelial-Mesenchymal Transition signature from Gibbons and Creighton Dev. Dyn. 2018.
  - `human_lung`, human lung cell type markers.
  - `mouse_brain`, mouse brain cell type markers.
  - `mouse_liver`, mouse liver cell type markers.
- **`n_bins`** (int, optional, default: 50) – Number of bins on expression levels for grouping genes.
  - **`show_omitted_genes`** (bool, optional, default False) – Signature genes that are not expressed in the data will be omitted. By default, pegasus does not report which genes are omitted. If this option is turned on, report omitted genes.
  - **`skip_threshold`** (int, optional, default 1) – Skip signature calculation of number of kept genes is less than `skip_threshold`.

- **random\_state** (int, optional, default: 0) – Random state used by KMeans if signature == gender\_human or gender\_mouse.

#### Return type

None

#### Returns

- None.
- Update data.obs –
  - data.obs["key"]: signature / gene module score for signature “key”
- Update data.var –
  - data.var["mean"]: Mean expression of each gene across all cells. Only updated if “mean” does not exist in data.var.
  - data.var["bins"]: Bin category for each gene. Only updated if data.uns[“sig\_n\_bins”] is updated.
- Update data.obsm –
  - data.obsm["sig\_background"]: Expected signature score for each bin category. Only updated if data.uns[“sig\_n\_bins”] is updated.
- Update data.uns –
  - data.uns["sig\_n\_bins"]: Number of bins to partition genes into. Only updated if “sig\_n\_bins” does not exist or the recorded number of bins does not match n\_bins.

#### Examples

```
>>> pg.calc_signature_score(data, {"T_cell_sig": ["CD3D", "CD3E", "CD3G", "TRAC"]})
>>> pg.calc_signature_score(data, "cell_cycle_human")
```

#### Label Transfer

<code>train_scarches_scanvi(data, dir_path, label)</code>	Run scArches training.
<code>predict_scarches_scanvi(data, dir_path, label)</code>	Run scArches training.

#### pegasus.train\_scarches\_scanvi

```
pegasus.train_scarches_scanvi(data, dir_path, label, unlabeled_category='Unknown',
                               features='highly_variable_features', matkey='counts', n_jobs=-1,
                               random_state=0, max_epochs=None, batch=None,
                               categorical_covariate_keys=None, continuous_covariate_keys=None,
                               semisupervised_max_epochs=None, n_samples_per_label=None,
                               use_gpu=None, arches_params={'dropout_rate': 0.2, 'encode_covariates':
                               True, 'n_layers': 2, 'use_batch_norm': 'none', 'use_layer_norm': 'both'})
```

Run scArches training.

This is a wrapper of `scvtools` package.

#### Parameters

- **data** (MultimodalData.) – Annotated data matrix with rows for cells and columns for genes.
- **dir\_path** (str.) – Save the model to this directory.
- **label** (str.) – The obs key representing labels.
- **unlabeled\_category** (str, default: "Unknown") – Value used for unlabeled cells in label.
- **features** (str, optional, default: "highly\_variable\_features") – Keyword in data.var, which refers to a boolean array. If None, all features will be selected.
- **matkey** (str, optional, default: "counts") – Matrix key for the raw count
- **n\_jobs** (int, optional, default: -1.) – Number of threads to use. -1 refers to using all physical CPU cores.
- **random\_state** (int, optional, default: 0.) – Seed for random number generator
- **max\_epochs** (int | None, optional, default: None.) – Maximum number of unsupervised training epochs. Defaults to  $\text{np.min}([\text{round}((20000 / \text{n\_cells}) * 400), 400])$
- **batch** (str, optional, default: None.) – If only one categorical covariate, the obs key representing batches that should be corrected for, default is None.
- **categorical\_covariate\_keys** (List[str]) – If multiple categorical covariates, a list of obs keys listing categorical covariates that should be corrected for, default is None.
- **continuous\_covariate\_keys** (List[str]) – A list of obs keys listing continuous covariates that should be corrected for, default is None.
- **semisupervised\_max\_epochs** (int | None, optional, default: None.) – Maximum number of semisupervised training epochs. Defaults to  $\text{np.min}([\text{round}(\text{np.sqrt}(\text{max\_epochs})), 20])$
- **n\_samples\_per\_label** (int, optional, default: None.) – Number of subsamples for each label class to sample per epoch. By default, there is no label subsampling.
- **use\_gpu** (str | int | bool | None) – Use default GPU if available (if None or True), or index of GPU to use (if int), or name of GPU (if str, e.g., 'cuda:0'), or use CPU (if False).
- **arches\_params** (dict.) – Hyperparameters for VAE. See <https://docs.scvi-tools.org/en/stable/api/reference/scvi.module.VAE.html#scvi.module.VAE> for more details

### Returns

- `data.obsm['X_scVI']`: The embedding calculated by scVI.
- `data.obsm['X_scanVI']`: The embedding calculated by scanVI.

### Return type

Update data.obsm

## Examples

```
>>> pg.train_scarches_scanvi(data, dir_path="scanvi_model/", label="celltype",  
↪matkey="counts", batch="tech", n_samples_per_label=100)
```

## pegasus.predict\_scarches\_scanvi

```
pegasus.predict_scarches_scanvi(data, dir_path, label, predictions='predictions', matkey='counts',  
                                n_jobs=-1, random_state=0, max_epochs=None, batch=None,  
                                categorical_covariate_keys=None, continuous_covariate_keys=None,  
                                use_gpu=None)
```

Run scArches training.

This is a wrapper of `scvitools` package.

### Parameters

- **data** (MultimodalData.) – Annotated data matrix with rows for cells and columns for genes.
- **dir\_path** (str.) – Save the model to this directory.
- **label** (str.) – The obs key representing labels.
- **predictions** (str, , optional, default: "predictions") – The obs key to store predicted labels.
- **matkey** (str, optional, default: "counts") – Matrix key for the raw count
- **n\_jobs** (int, optional, default: -1.) – Number of threads to use. -1 refers to using all physical CPU cores.
- **random\_state** (int, optional, default: 0.) – Seed for random number generator
- **max\_epochs** (int | None, optional, default: None.) – Maximum number of training epochs. Defaults to `np.min([round((20000 / n_cells) * 100), 100])`
- **batch** (str, optional, default: None.) – If only one categorical covariate, the obs key representing batches that should be corrected for, default is None.
- **categorical\_covariate\_keys** (List[str]) – If multiple categorical covariates, a list of obs keys listing categorical covariates that should be corrected for, default is None.
- **continuous\_covariate\_keys** (List[str]) – A list of obs keys listing continuous covariates that should be corrected for, default is None.
- **use\_gpu** (str | int | bool | None) – Use default GPU if available (if None or True), or index of GPU to use (if int), or name of GPU (if str, e.g., 'cuda:0'), or use CPU (if False).

### Returns

- `data.obsm['X_scanVI']`: The embedding calculated by scanVI.
- `data.obsm[predictions]`: The predicted labels by scanVI.

### Return type

Update `data.obsm`

## Examples

```
>>> pg.predict_scanvi(data, dir_path="scanvi_model/", label="celltype",
↳matkey="counts", batch="tech")
```

## Differential Expression and Gene Set Enrichment Analysis

<code>de_analysis(data, cluster[, condition, ...])</code>	Perform Differential Expression (DE) Analysis on data.
<code>markers(data[, head, de_key, alpha])</code>	Extract DE results into a human readable structure.
<code>write_results_to_excel(results, output_file)</code>	Write DE analysis results into Excel workbook.
<code>fgsea(data, log2fc_key, pathways[, de_key, ...])</code>	Perform Gene Set Enrichment Analysis using fGSEA.

### pegasus.de\_analysis

```
pegasus.de_analysis(data, cluster, condition=None, subset=None, de_key='de_res', n_jobs=-1, t=False,
fisher=False, temp_folder=None, verbose=True)
```

Perform Differential Expression (DE) Analysis on data.

The analysis considers one cluster at one time, comparing gene expression levels on cells within the cluster with all the others using a number of statistical tools, and determining up-regulated genes and down-regulated genes of the cluster.

Mann-Whitney U test and AUROC are calculated by default. Welch's T test and Fisher's Exact test are optionally.

The scalability performance on calculating all the test statistics is improved by the inspiration from [Presto](#).

#### Parameters

- **data** (`MultimodalData`, `UnimodalData`, or `anndata.AnnData`) – Data matrix with rows for cells and columns for genes.
- **cluster** (`str`) – Cluster labels used in DE analysis. Must exist in `data.obs`.
- **condition** (`str`, optional, default: `None`) – Sample attribute used as condition in DE analysis. If `None`, no condition is considered; otherwise, must exist in `data.obs`. If `condition` is used, the DE analysis will be performed on cells of each level of `data.obs[condition]` respectively, and collect the results after finishing.
- **subset** (`List[str]`, optional, default: `None`) – Perform DE analysis on only a subset of cluster IDs. Cluster ID subset is specified as a list of strings, such as `[clust_1, clust_3, clust_5]`, where all IDs must exist in `data.obs[cluster]`.
- **de\_key** (`str`, optional, default: `"de_res"`) – Key name of DE analysis results stored.
- **n\_jobs** (`int`, optional, default: `-1`) – Number of threads to use. If `-1`, use all available threads.
- **t** (`bool`, optional, default: `False`) – If `True`, calculate Welch's t test.
- **fisher** (`bool`, optional, default: `False`) – If `True`, calculate Fisher's exact test.
- **temp\_folder** (`str`, optional, default: `None`) – Joblib temporary folder for memmapping numpy arrays.
- **verbose** (`bool`, optional, default: `True`) – If `True`, show detailed intermediate output.

#### Return type

`None`

**Returns**

- None
- Update `data.varm` – `data.varm[de_key]`: DE analysis result.

**Examples**

```
>>> pg.de_analysis(data, cluster='spectral_leiden_labels')
>>> pg.de_analysis(data, cluster='louvain_labels', condition='anno')
```

**pegasus.markers**

`pegasus.markers(data, head=None, de_key='de_res', alpha=0.05)`

Extract DE results into a human readable structure.

This function extracts information from `data.varm[de_key]`, and return as a human readable dictionary of pandas DataFrame objects.

**Parameters**

- **data** (MultimodalData, UnimodalData, or `anndata.AnnData`) – Data matrix with rows for cells and columns for genes.
- **head** (int, optional, default: None) – List only top head genes for each cluster. If None, show any DE genes.
- **de\_key** (str, optional, default, `de_res`) – Keyword of DE result stored in `data.varm`.
- **alpha** (float, optional, default: 0.05) – q-value threshold for getting significant DE genes. Only those with q-value of MWU test no less than `alpha` are significant, and thus considered as DE genes.

**Returns**

**results** – A Python dictionary containing markers. If DE is performed between clusters, the structure is `dict[cluster_id]['up' or 'down'][dataframe]`. If DE is performed between conditions within each cluster, the structure is `dict[cluster_id][condition_id]['up' or 'down'][dataframe]`. ‘up’ refers to up-regulated genes, which should have ‘auroc’ > 0.5. ‘down’ refers to down-regulated genes, which should have ‘auroc’ < 0.5.

**Return type**

`Dict[str, Dict[str, pd.DataFrame]]`

**Examples**

```
>>> marker_dict = pg.markers(data)
```

## pegasus.write\_results\_to\_excel

`pegasus.write_results_to_excel(results, output_file, ndigits=3)`

Write DE analysis results into Excel workbook.

### Parameters

- **results** (Dict[str, Dict[str, pd.DataFrame]], or Dict[str, Dict[str, Dict[str, pd.DataFrame]]]) – DE marker dictionary generated by `pg.markers`.
- **output\_file** (str) – File name to which the marker dictionary is written.
- **ndigits** (int, optional, default: 3) – Round non p-values and q-values to ndigits after decimal point in the excel.

### Return type

None

### Returns

- None
- Marker information is written to file with name `output_file`.
- *In the generated Excel workbook, –*
  - If condition is None in `pg.de_analysis`: Each tab stores DE result of up/down-regulated genes of cells within one cluster, and its name follows the pattern: “**cluster\_id|up**” or “**cluster\_id|dn**”.
  - If condition is not None in `pg.de_analysis`: Each tab stores DE result of up/down-regulated genes of cells within one cluster under one condition level. The tab’s name follows the pattern: “**cluster\_id|cond\_level|up**” or “**cluster\_id|cond\_level|dn**”.
  - Notice that the tab name in Excel only allows at most 31 characters. Therefore, some of the resulting tab names may be truncated if their names are longer than this threshold.

## Examples

```
>>> pg.write_results_to_excel(marker_dict, "result.de.xlsx")
```

## pegasus.fgsea

`pegasus.fgsea(data, log2fc_key, pathways, de_key='de_res', minSize=15, maxSize=500, nproc=0, seed=0, fgsea_key='fgsea_out')`

Perform Gene Set Enrichment Analysis using fGSEA. This function calls R package fGSEA, requiring fGSEA in R installed.

### Parameters

- **data** (Union[MultimodalData, UnimodalData]) – Single-cell or pseudo-bulk data.
- **log2fc\_key** (str) – Key in pre-computed DE results representing log2 fold change.
- **pathways** (str) – Either a string or a path to the gene set file in GMT format. If string, choosing from “hallmark” and “canonical\_pathways” (MSigDB H and C2/CP).
- **de\_key** (str, optional, default: “de\_res”) – Key name of DE analysis results stored. `data.varm[de_key]` should contain a record array of DE results.

- **minSize** (int, optional, default: 15) – Minimal size of a gene set to consider.
- **maxSize** (int, optional, default: 500) – Maximal size of a gene set to consider.
- **nproc** (int, optional, default: 0) – Numbr of processes for parallel computation. If nproc > 0, set BPPARAM.
- **seed** (int, optional, default: 0) – Random seed to make sure fGSEA results are reproducible.
- **fgsea\_key** (str, optional, default: "fgsea\_out") – Key to use to store fGSEA results as a data frame.

**Return type**

None

**Returns**

- None
- Update data.uns – data.uns[fgsea\_key]: fGSEA outputs sorted by padj.

**Examples**

```
>>> pg.fgsea(data, '3:log2FC', hallmark', fgsea_key='fgsea_res')
```

**Annotate clusters**

<code>infer_cell_types</code> (data, markers[, de_test, ...])	Infer putative cell types for each cluster using legacy markers.
<code>annotate</code> (data, name, based_on, anno_dict)	Add annotation to the data object as a categorical variable.

**pegasus.infer\_cell\_types**

`pegasus.infer_cell_types`(data, markers, de\_test='mwu', de\_alpha=0.05, de\_key='de\_res', threshold=0.5, ignore\_nonde=False, output\_file=None)

Infer putative cell types for each cluster using legacy markers.

**Parameters**

- **data** (MultimodalData, UnimodalData, or anndata.AnnData.) – Data structure of count matrix and DE analysis results.
- **markers** (str or Dict) –
  - If str, it is a string representing a comma-separated list; each element in the list
    - \* either refers to a JSON file containing legacy markers, or predefined markers
    - \* 'human\_immune' for human immune cells;
    - \* 'mouse\_immune' for mouse immune cells;
    - \* 'human\_brain' for human brain cells;
    - \* 'mouse\_brain' for mouse brain cells;



- \* 'human\_lung' for human lung cells;
  - \* 'mouse\_lung' for mouse lung cells;
  - \* 'mouse\_liver' for mouse liver cells.
- If Dict, it refers to a Python dictionary describing the markers.
- **de\_test** (str, optional, default: "mwu") – pegasus determines cell types using DE test results. This argument indicates which DE test result to use, can be either 't', 'fisher' or 'mwu'. By default, it uses 'mwu'.
  - **de\_alpha** (float, optional, default: 0.05) – False discovery rate for controlling family-wide error.
  - **de\_key** (str, optional, default: "de\_res") – The keyword in data.varm that stores DE analysis results.
  - **threshold** (float, optional, default: 0.5) – Only report putative cell types with a score larger than or equal to threshold.
  - **ignore\_nonde** (bool, optional, default: False) – Do not consider non DE genes as weak negative markers.
  - **output\_file** (str, optional, default: None) – File name of output cluster annotation. If None, do not write to any file.

**Returns**

Python dictionary with cluster ID's being keys, and their corresponding cell type lists sorted by scores being values.

**Return type**

Dict[str, List["CellType"]]

**Examples**

```
>>> cell_type_dict = pg.infer_cell_types(adata, markers = 'human_immune,human_brain
↪')
```

**pegasus.annotate**

`pegasus.annotate(data, name, based_on, anno_dict)`

Add annotation to the data object as a categorical variable.

**Parameters**

- **data** (MultimodalData, UnimodalData, or anndata.AnnData) – Gene-count matrix with DE analysis information.
- **name** (str) – Name of the new annotation in data.obs.
- **based\_on** (str) – Name of the attribute the cluster ids coming from.
- **anno\_dict** (Dict[str, str] or List[str]) – Dictionary mapping from cluster id to cell type. If it is a List, map cell types to cluster ids one to one in correspondence.

**Return type**

None

**Returns**

- 
- *None*

## Examples

```
>>> pg.annotate(data, 'anno', 'spectral_louvain_labels', {'1': 'T cell', '2': 'B_
↪ cell'})
>>> pg.annotate(data, 'anno', 'louvain_labels', ['T cell', 'B cell'])
```

## Plotting

<code>scatter</code> (data[, attrs, basis, components, ...])	Generate scatter plots for different attributes
<code>scatter_groups</code> (data, attr, groupby[, basis, ...])	Generate scatter plots of attribute 'attr' for each category in attribute 'group'.
<code>spatial</code> (data[, attrs, basis, resolution, ...])	Scatter plot on spatial coordinates.
<code>compo_plot</code> (data, groupby, condition[, ...])	Generate a composition plot, which shows the percentage of cells from each condition for every cluster.
<code>violin</code> (data, attrs, groupby[, hue, matkey, ...])	Generate a stacked violin plot.
<code>heatmap</code> (data, attrs, groupby[, matkey, ...])	Generate a heatmap.
<code>dotplot</code> (data, genes, groupby[, ...])	Generate a dot plot.
<code>dendrogram</code> (data, groupby[, rep, genes, ...])	Generate a dendrogram on hierarchical clustering result.
<code>hvfplot</code> (data[, top_n, panel_size, ...])	Generate highly variable feature plot.
<code>qcviolin</code> (data, plot_type[, ...])	Plot quality control statistics (before filtration vs.
<code>volcano</code> (data, cluster_id[, de_key, de_test, ...])	Generate Volcano plots (-log10 p value vs.
<code>rank_plot</code> (data[, panel_size, return_fig, dpi])	Generate a barcode rank plot, which shows the total UMIs against barcode rank (in descending order with respect to total UMIs)
<code>ridgeplot</code> (data, features[, matrix_key, ...])	Generate ridge plots, up to 8 features can be shown in one figure.
<code>wordcloud</code> (data, factor[, max_words, ...])	Generate one word cloud image for factor (starts from 0) in data.uns['W'].
<code>plot_gsea</code> (data, gsea_keyword[, alpha, ...])	Generate GSEA barplots
<code>elbowplot</code> (data[, rep, pval, panel_size, ...])	Generate Elbowplot and suggest n_comps to select based on random matrix theory (see <code>utils.largest_variance_from_random_matrix</code> ).

## pegasus.scatter

`pegasus.scatter`(data, attrs=None, basis='umap', components=(1, 2), matkey=None, restrictions=None, show\_background=False, fix\_corners=True, alpha=1.0, legend\_loc='right margin', legend\_fontsize=10, legend\_ncol=None, palettes=None, cmaps='YlOrRd', vmin=None, vmax=None, nrows=None, ncols=None, panel\_size=(4, 4), left=0.2, bottom=0.15, wspace=0.4, hspace=0.15, marker\_size=None, scale\_factor=None, return\_fig=False, dpi=300.0, show\_neg\_for\_sig=False, \*\*kwargs)

Generate scatter plots for different attributes

### Parameters

- **data** (`pegasusio.MultimodalData`) – Use current selected modality in data.

- **attrs** (str or List[str], default: None) – Color scatter plots by attrs. Each attribute in attrs can be one key in data.obs, data.var\_names (e.g. one gene) or data.obsm (attribute has the format of ‘obsm\_key@component’, like ‘X\_pca@0’). If one attribute is categorical, a palette will be used to color each category separately. Otherwise, a color map will be used. If no attributes are provided, plot the basis for all data.
- **basis** (str or List[str], optional, default: umap) – Basis to be used to generate scatter plots. Can be either ‘pca’, ‘diffmap’, ‘umap’, ‘tsne’, ‘fitsne’, ‘fle’, ‘net\_tsne’, ‘net\_fitsne’, ‘net\_umap’ or ‘net\_fle’. If *basis* is a list, each of element in *attrs* will be plotted for each basis in *basis*.
- **components** (Tuple[int, int] or List[Tuple[int, int]], optional, default: (1, 2)) – Components in basis to be used. Default to the first two components. If *components* is a list, for each element in *attrs* and each *basis*, all components enumeration will be plotted.
- **matkey** (str, optional, default: None) – If matkey is set, select matrix with matkey as keyword in the current modality. Only works for MultimodalData or UnimodalData objects.
- **restrictions** (str or List[str], optional, default: None) – A list of restrictions to subset data for plotting. There are two types of restrictions: global restriction and attribute-specific restriction. Global restriction applies to all attributes in *attrs* and takes the format of ‘key:value,value...’, or ‘key:~value,value...’. This restriction selects cells with the data.obs[key] values belong to ‘value,value...’ (or not belong to if ‘~’ shows). Attribute-specific restriction takes the format of ‘attr:key:value,value...’, or ‘attr:key:~value,value...’. It only applies to one attribute ‘attr’. If ‘attr’ and ‘key’ are the same, one can use ‘.’ to replace ‘key’ (e.g. cluster\_labels:.:value1,value2).
- **show\_background** (bool, optional, default: False) – Only applicable if *restrictions* is set. By default, only data points selected are shown. If show\_background is True, data points that are not selected will also be shown.
- **fix\_corners** (bool, optional, default: True) – If True, fix the corners of the plots as defined using all data points.
- **alpha** (float or List[float], optional, default: 1.0) – Alpha value for blending, from 0.0 (transparent) to 1.0 (opaque). If this is a list, the length must match attrs, which means we set a separate alpha value for each attribute.
- **legend\_loc** (str or List[str], optional, default: right margin) – Legend location. Can be either “right margin” or “on data”. If a list is provided, set ‘legend\_loc’ for each attribute in ‘attrs’ separately.
- **legend\_fontsize** (int or List[int], optional, default: 10) – Legend fontsize. If a list is provided, set ‘legend\_fontsize’ for each attribute in ‘attrs’ separately.
- **legend\_ncol** (str, optional, default: None) – Only applicable if legend\_loc == “right margin”. Set number of columns used to show legends.
- **palettes** (str or List[str], optional, default: None) – Used for setting colors for every categories in categorical attributes. Each string in *palettes* takes the format of ‘attr:color1,color2,...,colorn’. ‘attr’ is the categorical attribute and ‘color1’ - ‘colorn’ are the colors for each category in ‘attr’ (e.g. ‘cluster\_labels:black,blue,red,...,yellow’). If there is only one categorical attribute in ‘attrs’, *palettes* can be set as a single string and the ‘attr’ keyword can be omitted (e.g. “blue,yellow,red”).
- **cmaps** (str or List[str], optional, default: YlOrRd) – Used for setting colormap for numeric attributes. Each string in *cmaps* takes the format of ‘colormap’ or ‘attr:colormap’.

'colormap' sets the default colormap for all numeric attributes. 'attr:colormap' overwrites attribute 'attr's colormap as 'colormap'.

- **vmin** (float, optional, default: None) – Minimum value to show on a numeric scatter plot (feature plot).
- **vmax** (float, optional, default: None) – Maximum value to show on a numeric scatter plot (feature plot).
- **nrows** (int, optional, default: None) – Number of rows in the figure. If not set, pegasus will figure it out automatically.
- **ncols** (int, optional, default: None) – Number of columns in the figure. If not set, pegasus will figure it out automatically.
- **panel\_size** (tuple, optional (default: (6, 4))) – The panel size (width, height) in inches.
- **left** (float, optional (default: 0.2)) – This parameter sets the figure's left margin as a fraction of panel's width (left \* panel\_size[0]).
- **bottom** (float, optional (default: 0.15)) – This parameter sets the figure's bottom margin as a fraction of panel's height (bottom \* panel\_size[1]).
- **wspace** (float, optional (default: 0.4)) – This parameter sets the width between panels and also the figure's right margin as a fraction of panel's width (wspace \* panel\_size[0]).
- **hspace** (float, optional (default: 0.15)) – This parameter sets the height between panels and also the figure's top margin as a fraction of panel's height (hspace \* panel\_size[1]).
- **marker\_size** (float, optional (default: None)) – Manually set the marker size in the plot. If None, automatically adjust the marker size to the plot size.
- **scale\_factor** (float, optional (default: None)) – Manually set the scale factor in the plot if it's not None. This is used by generating the spatial plots for 10x Visium data.
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution of the figure in dots-per-inch.
- **show\_neg\_for\_sig** (bool, optional, default: False) – For signature scores (i.e. attribute type registered as 'signature'), if we should show negative scores or show them as zeros. Default is False (i.e. show them as zeros).

**Return type**

Optional[Figure]

**Returns**

- 
- *Figure* object – A matplotlib.figure.Figure object containing the dot plot if return\_fig == True

## Examples

```
>>> pg.scatter(data, attrs=['louvain_labels', 'Channel'], basis='fitsne')
>>> pg.scatter(data, attrs=['CD14', 'TRAC'], basis='umap')
```

## pegasus.scatter\_groups

`pegasus.scatter_groups`(*data*, *attr*, *groupby*, *basis*='umap', *matkey*=None, *restrictions*=None, *show\_full*=True, *categories*=None, *alpha*=1.0, *legend\_loc*='right margin', *legend\_ncol*=None, *palette*=None, *cmap*='YlOrRd', *vmin*=None, *vmax*=None, *nrows*=None, *ncols*=None, *panel\_size*=(4, 4), *left*=0.2, *bottom*=0.15, *wspace*=0.4, *hspace*=0.15, *return\_fig*=False, *dpi*=300.0, *show\_neg\_for\_sig*=False, *\*\*kwargs*)

Generate scatter plots of attribute ‘attr’ for each category in attribute ‘group’. Optionally show scatter plot containing data points from all categories in ‘group’.

### Parameters

- **data** (pegasusio.MultimodalData) – Use current selected modality in data.
- **attr** (str) – Color scatter plots by attribute ‘attr’. This attribute should be one key in data.obs, data.var\_names (e.g. one gene) or data.obsm (attribute has the format of ‘obsm\_key@component’, like ‘X\_pca@0’). If it is categorical, a palette will be used to color each category separately. Otherwise, a color map will be used.
- **groupby** (str) – Generate separate scatter plots of ‘attr’ for data points in each category in ‘groupby’, which should be a key in data.obs representing one categorical variable.
- **basis** (str, optional, default: umap) – Basis to be used to generate scatter plots. Can be either ‘umap’, ‘tsne’, ‘fitsne’, ‘fle’, ‘net\_tsne’, ‘net\_fitsne’, ‘net\_umap’ or ‘net\_fle’.
- **matkey** (str, optional, default: None) – If matkey is set, select matrix with matkey as keyword in the current modality. Only works for MultimodalData or UnimodalData objects.
- **restrictions** (str or List[str], optional, default: None) – A list of restrictions to subset data for plotting. Each restriction takes the format of ‘key:value,value...’, or ‘key:~value,value...’. This restriction selects cells with the data.obs[key] values belong to ‘value,value...’ (or not belong to if ‘~’ shows).
- **show\_full** (bool, optional, default: True) – Show the scatter plot with all categories in ‘groupby’ as the first plot.
- **categories** (List[str], optional, default: None) – Redefine group structure based on attribute ‘groupby’. If ‘categories’ is not None, each string in the list takes the format of ‘category\_name:value,value...’, or ‘category\_name:~value,value...’, where ‘category\_name’ refers to new category name, ‘value’ refers to one of the category in ‘groupby’ and ‘~’ refers to exclude values.
- **alpha** (float, optional, default: 1.0) – Alpha value for blending, from 0.0 (transparent) to 1.0 (opaque).
- **legend\_loc** (str, optional, default: right margin) – Legend location. Can be either “right margin” or “on data”.
- **legend\_ncol** (str, optional, default: None) – Only applicable if legend\_loc == “right margin”. Set number of columns used to show legends.

- **palette** (str, optional, default: None) – Used for setting colors for one categorical attribute (e.g. “black,blue,red,...,yellow”).
- **cmap** (str, optional, default: YlOrRd) – Used for setting colormap for one numeric attribute.
- **vmin** (float, optional, default: None) – Minimum value to show on a numeric scatter plot (feature plot).
- **vmax** (float, optional, default: None) – Maximum value to show on a numeric scatter plot (feature plot).
- **nrows** (int, optional, default: None) – Number of rows in the figure. If not set, pegasus will figure it out automatically.
- **ncols** (int, optional, default: None) – Number of columns in the figure. If not set, pegasus will figure it out automatically.
- **panel\_size** (tuple, optional (default: (6, 4))) – The panel size (width, height) in inches.
- **left** (float, optional (default: 0.2)) – This parameter sets the figure’s left margin as a fraction of panel’s width (left \* panel\_size[0]).
- **bottom** (float, optional (default: 0.15)) – This parameter sets the figure’s bottom margin as a fraction of panel’s height (bottom \* panel\_size[1]).
- **wspace** (float, optional (default: 0.4)) – This parameter sets the width between panels and also the figure’s right margin as a fraction of panel’s width (wspace \* panel\_size[0]).
- **hspace** (float, optional (default: 0.15)) – This parameter sets the height between panels and also the figure’s top margin as a fraction of panel’s height (hspace \* panel\_size[1]).
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution of the figure in dots-per-inch.
- **show\_neg\_for\_sig** (bool, optional, default: False) – For signature scores (i.e. attribute type registered as ‘signature’), if we should show negative scores or show them as zeros. Default is False (i.e. show them as zeros).

**Return type**

Optional[Figure]

**Returns**

- 
- *Figure* object – A matplotlib.figure.Figure object containing the dot plot if return\_fig == True

**Examples**

```
>>> pg.scatter_groups(data, attr='louvain_labels', groupby='Individual', basis='tsne',
↳ rows = 2, ncols = 4, alpha = 0.5)
>>> pg.scatter_groups(data, attr='anno', groupby='Channel', basis='umap',
↳ categories=['new_cat1:channel1,channel2', 'new_cat2:channel3'])
```

## pegasus.spatial

`pegasus.spatial`(*data*, *attrs=None*, *basis='spatial'*, *resolution='hires'*, *cmaps='viridis'*, *vmin=None*, *vmax=None*, *alpha=1.0*, *alpha\_img=1.0*, *dpi=300.0*, *return\_fig=False*, *\*\*kwargs*)

Scatter plot on spatial coordinates. This function is inspired by SCANPY's `pl.spatial` function.

### Parameters

- **data** (`pegasusio.MultimodalData` or `pegasusio.UnimodalData` or `anndata.AnnData`) – Use current selected modality in data.
- **attr** (str, optional, default: `None`) – Color scatter plots by attribute 'attr'. This attribute should be one key in `data.obs`, `data.var_names` (e.g. one gene) or `data.obsm` (attribute has the format of '`obsm_key@component`', like '`X_pca@0`'). If it is categorical, a palette will be used to color each category separately. Otherwise, a color map will be used. If `None`, just plot data points of the same color.
- **basis** (str, optional, default: `spatial`) – Basis to be used to generate spatial plots. Must be the 2D array showing the spatial coordinates of data points.
- **resolution** (str, optional, default: `hires`) – Use the spatial image whose value is specified in `data.img['image_id']` to show in background. For 10X Visium data, user can either specify `hires` or `lowres` to use High or Low resolution spatial images, respectively.
- **cmaps** (str or `List[str]`, optional, default: `viridis`) – The colormap(s) for plotting numeric attributes. The default `viridis` colormap theme follows the spatial plot function in SCANPY (`scanpy.pl.spatial`).
- **vmin** (float, optional, default: `None`) – Minimum value to show on a numeric scatter plot (feature plot).
- **vmax** (float, optional, default: `None`) – Maximum value to show on a numeric scatter plot (feature plot).
- **alpha** (float or `List[float]`, optional, default: `1.0`) – Alpha value for blending the attribute layers, from 0.0 (transparent) to 1.0 (opaque). If this is a list, the length must match `attrs`, which means we set a separate alpha value for each attribute.
- **alpha\_img** (float, optional, default: `1.0`) – Alpha value for blending the background spatial image, from 0.0 (transparent) to 1.0 (opaque).
- **dpi** (float, optional, default: `300.0`) – The resolution of the figure in dots-per-inch.
- **return\_fig** (bool, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.

### Returns

A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

### Return type

Figure object

## Examples

```
>>> pg.spatial(data, attrs=['louvain_labels', 'Channel'])
>>> pg.spatial(data, attrs=['CD14', 'TRAC'], resolution='lowres')
```

## pegasus.compo\_plot

```
pegasus.compo_plot(data, groupby, condition, style='frequency', restrictions=None, switch_axes=False,
                    groupby_label=None, sort_function='natsorted', panel_size=(6, 4), palette=None,
                    color_unused=False, left=0.15, bottom=0.15, wspace=0.3, hspace=0.15, return_fig=False,
                    dpi=300.0, **kwargs)
```

Generate a composition plot, which shows the percentage of cells from each condition for every cluster.

This function is used to generate composition plots, which are bar plots showing the cell compositions (from different conditions) for each cluster. This type of plots is useful to fast assess library quality and batch effects.

### Parameters

- **data** (AnnData or UnimodalData or MultimodalData object) – Single cell expression data.
- **groupby** (str) – A categorical variable in data.obs that is used to categorize the cells, e.g. cell type.
- **condition** (str) – A categorical variable in data.obs that is used to calculate frequency within each category defined by groupby, e.g. donor.
- **style** (str, optional (default: frequency)) – Composition plot style. Can be either frequency, or normalized. Within each cluster, the frequency style show the percentage of cells from each condition within each category in groupby (stacked), the normalized style shows for each category in groupby the percentage of cells that are also in each condition over all cells that are in the same condition (not stacked).
- **restrictions** (str or List[str], optional, default: None) – A list of restrictions to subset data for plotting. Each restriction takes the format of 'key:value,value...', or 'key:~value,value...'. This restriction selects cells with the data.obs[key] values belong to 'value,value...' (or not belong to if '~' shows).
- **switch\_axes** (bool, optional, default: False) – By default, X axis is for groupby, and Y axis for frequencies with respect to condition. If this parameter is True, switch the axes.
- **groupby\_label** (str, optional (default None)) – Label for the axis displaying groupby categories. If None, use groupby.
- **sort\_function** (Union[Callable[List[str], List[str]], str], optional, default: natsorted) – Function used for sorting both groupby and condition labels. If natsorted, apply natsorted function to sort by natural order. If None, don't sort. Otherwise, a callable function will be applied to the labels for sorting.
- **panel\_size** (tuple, optional (default: (6, 4))) – The plot size (width, height) in inches.
- **palette** (List[str], optional (default: None)) – Used for setting colors for categories in condition. Within the list, each string is the color for one category.
- **left** (float, optional (default: 0.15)) – This parameter sets the figure's left margin as a fraction of panel's width (left \* panel\_size[0]).



- **bottom** (float, optional (default: 0.15)) – This parameter sets the figure’s bottom margin as a fraction of panel’s height (`bottom * panel_size[1]`).
- **wspace** (float, optional (default: 0.3)) – This parameter sets the width between panels and also the figure’s right margin as a fraction of panel’s width (`wspace * panel_size[0]`).
- **hspace** (float, optional (default: 0.15)) – This parameter sets the height between panels and also the figure’s top margin as a fraction of panel’s height (`hspace * panel_size[1]`).
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

**Return type**

Optional[Figure]

**Returns**

- 
- Figure object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

**Examples**

```
>>> fig = pg.compo_plot(data, 'louvain_labels', 'Donor', style = 'normalized')
```

**pegasus.violin**

`pegasus.violin`(data, attrs, groupby, hue=None, matkey=None, stripplot=False, stripsize=1, inner=None, scale='width', panel\_size=(8, 0.5), palette=None, left=0.15, bottom=0.15, wspace=0.1, ylabel=None, return\_fig=False, dpi=300.0, \*\*kwargs)

Generate a stacked violin plot.

**Parameters**

- **data** (AnnData or MultimodalData or UnimodalData object) – Single-cell expression data.
- **attrs** (str or List[str]) – Cell attributes or features to plot. Cell attributes must exist in `data.obs` and must be numeric. Features must exist in `data.var`.
- **groupby** (str) – A categorical variable in `data.obs` that is used to categorize the cells, e.g. Clusters.
- **hue** (str, optional, default: None) – ‘hue’ should be a categorical variable in `data.obs` that has only two levels. Set ‘hue’ will show us split violin plots.
- **matkey** (str, optional, default: None) – If `matkey` is set, select matrix with `matkey` as keyword in the current modality. Only works for MultimodalData or UnimodalData objects.
- **stripplot** (bool, optional, default: False) – Attach a stripplot to the violinplot or not. This option will be automatically turn off if ‘hue’ is set.
- **inner** (str, optional, default: None) –

**Representation of the datapoints in the violin interior:**

- If `box`, draw a miniature boxplot.
- If `quartiles`, draw the quartiles of the distribution.
- If `point` or `stick`, show each underlying datapoint.
- If `None`, will draw unadorned violins.
- **scale** (`str`, optional, default: `width`) –  
    **The method used to scale the width of each violin:**
  - If `width`, each violin will have the same width.
  - If `area`, each violin will have the same area.
  - If `count`, the width of the violins will be scaled by the number of observations in that bin.
- **panel\_size** (`Tuple[float, float]`, optional, default: `(8, 0.5)`) – The size (width, height) in inches of each violin panel.
- **palette** (`List[str]`, optional (default: `None`)) – Used for setting colors for categories in `groupby`. Within the list, each string is the color for one category.
- **left** (`float`, optional, default: `0.15`) – This parameter sets the figure's left margin as a fraction of panel's width (`left * panel_size[0]`).
- **bottom** (`float`, optional, default: `0.15`) – This parameter sets the figure's bottom margin as a fraction of panel's height (`bottom * panel_size[1]`).
- **wspace** (`float`, optional, default: `0.1`) – This parameter sets the width between panels and also the figure's right margin as a fraction of panel's width (`wspace * panel_size[0]`).
- **ylabel** (`str`, optional, default: `None`) – Y-axis label. No label to show if `None`.
- **return\_fig** (`bool`, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (`float`, optional, default: `300.0`) – The resolution in dots per inch.
- **kwargs** – Are passed to `seaborn.violinplot`.

**Returns**

A `matplotlib.figure.Figure` object containing the dot plot if `show == False`

**Return type**

`Figure` object

**Examples**

```
>>> pg.violin(data, attrs=['CD14', 'TRAC', 'CD34'], groupby='louvain_labels')
```

**pegasus.heatmap**

`pegasus.heatmap(data, attrs, groupby, matkey=None, on_average=True, switch_axes=False, attrs_cluster=False, attrs_dendrogram=True, groupby_cluster=True, groupby_dendrogram=True, attrs_labelsize=10.0, groupby_labelsize=10.0, cbar_labelsize=10.0, panel_size=(10, 10), return_fig=False, dpi=300.0, **kwargs)`

Generate a heatmap.

**Parameters**

- **data** (AnnData or MultimodalData or UnimodalData object) – Single-cell expression data.
- **attrs** (str or List[str]) – Cell attributes or features to plot. Cell attributes must exist in `data.obs` and must be numeric. Features must exist in `data.var`. By default, `attrs` are plotted as columns.
- **groupby** (str) – A categorical variable in `data.obs` that is used to categorize the cells, e.g. Clusters. By default, `data.obs[‘groupby’]` is plotted as rows.
- **matkey** (str, optional, default: None) – If `matkey` is set, select matrix with `matkey` as keyword in the current modality. Only works for MultimodalData or UnimodalData objects.
- **on\_average** (bool, optional, default: True) – If True, plot cluster average gene expression (i.e. show a Matrixplot); otherwise, plot a general heatmap.
- **switch\_axes** (bool, optional, default: False) – By default, X axis is for attributes, and Y axis for clusters. If this parameter is True, switch the axes. Moreover, with `on_average` being False, if `switch_axes` is False, `row_cluster` is enforced to be False; if `switch_axes` is True, `col_cluster` is enforced to be False.
- **attrs\_cluster** (bool, optional, default: False) – Cluster attributes and generate a attribute-wise dendrogram.
- **attrs\_dendrogram** (bool, optional, default: True) – Only matters if `attrs_cluster` is True. Show the dendrogram if this option is True.
- **groupby\_cluster** (bool, optional, default: True) – Cluster `data.obs[‘groupby’]` and generate a cluster-wise dendrogram.
- **groupby\_dendrogram** (bool, optional, default: True) – Only matters if `groupby_cluster` is True. Show the dendrogram if this option is True.
- **attrs\_labelsize** (float, optional, default: 10.0) – Fontsize for labels of `attrs`.
- **groupby\_labelsize** (float, optional, default: 10.0) – Fontsize for labels of `data.obs[‘groupby’]`.
- **cbar\_labelsize** (float, optional, default: 10.0) – Fontsize of the color bar.
- **panel\_size** (Tuple[float, float], optional, default: (10, 10)) – Overall size of the heatmap in (width, height) form.
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.
- **kwargs** – Are passed to `seaborn.heatmap`.
- **documentation** (.. `_colormap`) –

**Return type**

Optional[Figure]

**Returns**

- 
- Figure object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

**Examples**

```
>>> pg.heatmap(data, attrs=['CD14', 'TRAC', 'CD34'], groupby='louvain_labels')
```

**pegasus.dotplot**

```
pegasus.dotplot(data, genes, groupby, reduce_function=<function mean>, fraction_min=0, fraction_max=None,
                dot_min=0, dot_max=20, switch_axes=False, cmap='Reds', sort_function='natsorted',
                grid=True, return_fig=False, dpi=300.0, **kwds)
```

Generate a dot plot.

**Parameters**

- **data** (AnnData or UnimodalData or MultimodalData object) – Single cell expression data.
- **genes** (str or List[str]) – Features to plot.
- **groupby** (str) – A categorical variable in `data.obs` that is used to categorize the cells, e.g. Clusters.
- **reduce\_function** (Callable[[np.ndarray], float], optional, default: `np.mean`) – Function to calculate statistic on expression data. Default is mean.
- **fraction\_min** (float, optional, default: 0.) – Minimum fraction of expressing cells to consider.
- **fraction\_max** (float, optional, default: None.) – Maximum fraction of expressing cells to consider. If None, use the maximum value from data.
- **dot\_min** (int, optional, default: 0.) – Minimum size in pixels for dots.
- **dot\_max** (int, optional, default: 20.) – Maximum size in pixels for dots.
- **switch\_axes** (bool, optional, default: False.) – If True, switch X and Y axes.
- **cmap** (str or List[str] or Tuple[str], optional, default: `Reds`) – Color map.
- **sort\_function** (Union[Callable[List[str], List[str]], str], optional, default: `natsorted`) – Function used for sorting groupby labels. If `natsorted`, apply `natsorted` function to sort by natural order. If None, don't sort. Otherwise, a callable function will be applied to the labels for sorting.
- **grid** (bool, optional, default: True) – If True, plot grids.
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.
- **\*\*kwds** – Are passed to `matplotlib.pyplot.scatter`.

**Return type**

Optional[Figure]

**Returns**

- 
- Figure object – A matplotlib.figure.Figure object containing the dot plot if `return_fig == True`

**Examples**

```
>>> pg.dotplot(data, genes = ['CD14', 'TRAC', 'CD34'], groupby = 'louvain_labels')
```

**pegasus.dendrogram**

```
pegasus.dendrogram(data, groupby, rep='pca', genes=None, correlation_method='pearson', n_clusters=None,
                    affinity='euclidean', linkage='complete', compute_full_tree='auto', distance_threshold=0,
                    panel_size=(6, 6), orientation='top', color_threshold=None, return_fig=False, dpi=300.0,
                    **kwargs)
```

Generate a dendrogram on hierarchical clustering result.

The metrics used here are consistent with SCANPY's [dendrogram](#) implementation.

*scikit-learn* [Agglomerative Clustering](#) implementation is used for hierarchical clustering.

**Parameters**

- **data** (MultimodalData, UnimodalData, or AnnData object) – Single cell expression data.
- **genes** (List[str], optional, default: None) – List of genes to use. Gene names must exist in `data.var`. If set, use the counts in `data.X` for plotting; if set as None, use the embedding specified in `rep` for plotting.
- **rep** (str, optional, default: pca) – Cell embedding to use. It only works when `genes` is ``None`, and its key "`X_`" + `rep` must exist in `data.obsm`. By default, use PCA coordinates.
- **groupby** (str) – Categorical cell attribute to plot, which must exist in `data.obs`.
- **correlation\_method** (str, optional, default: pearson) – Method of correlation between categories specified in `data.obs`. Available options are: `pearson`, `kendall`, `spearman`. See [pandas corr documentation](#) for details.
- **n\_clusters** (int, optional, default: None) – The number of clusters to find, used by hierarchical clustering. It must be None if `distance_threshold` is not None.
- **affinity** (str, optional, default: correlation) –

**Metric used to compute the linkage, used by hierarchical clustering. Valid values for metric are:**

- From *scikit-learn*: `cityblock`, `cosine`, `euclidean`, `l1`, `l2`, `manhattan`.
- From `scipy.spatial.distance`: `braycurtis`, `canberra`, `chebyshev`, `correlation`, `dice`, `hamming`, `jaccard`, `kulsinski`, `mahalanobis`, `minkowski`, `rogerstanimoto`, `russellrao`, `seuclidean`, `sokalmichener`, `sokalsneath`, `squeuclidean`, `yule`.

Default is the correlation distance. See [scikit-learn distance documentation](#) for details.

- **linkage** (str, optional, default: complete) –

Which linkage criterion to use, used by hierarchical clustering. Below are available options:

- ward minimizes the variance of the clusters being merged.
- average uses the average of the distances of each observation of the two sets.
- complete uses the maximum distances between all observations of the two sets. (Default)
- single uses the minimum of the distances between all observations of the two sets.

See [scikit-learn documentation](#) for details.

- **compute\_full\_tree** (str or bool, optional, default: auto) – Stop early the construction of the tree at `n_clusters`, used by hierarchical clustering. It must be `True` if `distance_threshold` is not `None`. By default, this option is `auto`, which is `True` if and only if `distance_threshold` is not `None`, or `n_clusters` is less than `min(100, 0.02 * n_groups)`, where `n_groups` is the number of categories in `data.obs[groupby]`.
- **distance\_threshold** (float, optional, default: 0) – The linkage distance threshold above which, clusters will not be merged. If not `None`, `n_clusters` must be `None` and `compute_full_tree` must be `True`.
- **panel\_size** (Tuple[float, float], optional, default: (6, 6)) – The size (width, height) in inches of figure.
- **orientation** (str, optional, default: top) – The direction to plot the dendrogram. Available options are: `top`, `bottom`, `left`, `right`. See [scipy dendrogram documentation](#) for explanation.
- **color\_threshold** (float, optional, default: None) – Threshold for coloring clusters. See [scipy dendrogram documentation](#) for explanation.
- **return\_fig** (bool, optional, default: False) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.
- **\*\*kwargs** – Are passed to `scipy.cluster.hierarchy.dendrogram`.

#### Return type

Optional[Figure]

#### Returns

- 
- `Figure` object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

## Examples

```
>>> pg.dendrogram(data, genes=data.var_names, groupby='louvain_labels')
>>> pg.dendrogram(data, rep='pca', groupby='louvain_labels')
```

## pegasus.hvplot

`pegasus.hvplot(data, top_n=20, panel_size=(6, 4), return_fig=False, dpi=300.0)`

Generate highly variable feature plot. Only works for HVGs returned by `highly_variable_features` method with `flavor=='pegasus'`.

### Parameters

- **data** (MultimodalData, UnimodalData, or `anndata.AnnData` object.) – Single cell expression data.
- **top\_n** (int, optional, default: 20) – Number of top highly variable features to show names.
- **panel\_size** (Tuple[float, float], optional, default: (6, 4)) – The size (width, height) in inches of figure.
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

### Return type

Optional[Figure]

### Returns

- Figure object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

## Examples

```
>>> pg.hvplot(data)
>>> pg.hvplot(data, top_n=10, dpi=150)
```

## pegasus.qcviolin

`pegasus.qcviolin(data, plot_type, min_genes_before_filt=100, n_violin_per_panel=8, panel_size=(6, 4), left=0.2, bottom=0.15, wspace=0.3, hspace=0.35, return_fig=False, dpi=300.0)`

Plot quality control statistics (before filtration vs. after filtration) as violin plots. Require statistics such as “n\_genes”, “n\_counts” and “percent\_mito” precomputed.

### Parameters

- **data** (MultimodalData, UnimodalData, or `anndata.AnnData` object.) – Single cell expression data.
- **plot\_type** (str) – Choose from `gene`, `count` and `mito`, which shows number of expressed genes, number of UMIs and percentage of mitochondrial rate.

- **min\_genes\_before\_filt** (int, optional, default: 100) – If data loaded are raw data (i.e. `min(n_genes) == 0`), filter out cell barcodes with less than `min_genes_before_filt` for better visual effects.
- **n\_violin\_per\_panel** (int, optional, default: 8) – Number of violin plots (samples) shown in one panel.
- **panel\_size** (tuple, optional (default: (6, 4))) – The panel size (width, height) in inches.
- **left** (float, optional (default: 0.2)) – This parameter sets the figure's left margin as a fraction of panel's width (`left * panel_size[0]`).
- **bottom** (float, optional (default: 0.15)) – This parameter sets the figure's bottom margin as a fraction of panel's height (`bottom * panel_size[1]`).
- **wspace** (float, optional (default: 0.4)) – This parameter sets the width between panels and also the figure's right margin as a fraction of panel's width (`wspace * panel_size[0]`).
- **hspace** (float, optional (default: 0.15)) – This parameter sets the height between panels and also the figure's top margin as a fraction of panel's height (`hspace * panel_size[1]`).
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

**Return type**

Optional[Figure]

**Returns**

- 
- Figure object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

**Examples**

```
>>> pg.qcviolin(data, "mito", dpi = 500)
```

**pegasus.volcano**

```
pegasus.volcano(data, cluster_id, de_key='de_res', de_test='mwu', qval_threshold=0.05, log2fc_threshold=1.0, top_n=20, panel_size=(6, 4), return_fig=False, dpi=300.0)
```

Generate Volcano plots (-log10 p value vs. log2 fold change) for visualizing DE results.

**Parameters**

- **data** (MultimodalData, UnimodalData, or `anndata.AnnData` object.) – Single cell expression data.
- **cluster\_id** (str) –

**Cluster ID for the cluster we want to show DE results. There are two cases:**

- If condition is None in `pg.de_analysis`: Just specify one cluster label in the cluster attribute used in `pg.de_analysis`.



- If `condition` is not `None` in `pg.de_analysis`: Specify cluster ID in this format: “**cluster\_label:cond\_level**”, where **cluster\_label** is the cluster label, and **cond\_level** is the condition ID. And this shows result of cells within the cluster under the specific condition.
- **de\_key** (str, optional, default: `de_res`) – The varm keyword for DE results. `data.varm[de_key]` should store the full DE result table.
- **de\_test** (str, optional, default: `mwu`) – Which DE test results to show. Use MWU test result by default.
- **qval\_threshold** (float, optional, default: 0.05.) – Selected FDR rate. A horizontal line indicating this rate will be shown in the figure.
- **log2fc\_threshold** (float, optional, default: 1.0) – Log2 fold change threshold to highlight biologically interesting genes. Two vertical lines representing negative and positive log2 fold change will be shown.
- **top\_n** (int, optional, default: 20) – Number of top DE genes to show names. Genes are ranked by Log2 fold change.
- **panel\_size** (Tuple[float, float], optional, default: (6, 4)) – The size (width, height) in inches of figure.
- **return\_fig** (bool, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

**Return type**

Optional[Figure]

**Returns**

- 
- Figure object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

**Examples**

```
>>> pg.volcano(data, cluster_id = '1', dpi=200)
```

**pegasus.rank\_plot**

`pegasus.rank_plot(data, panel_size=(6, 4), return_fig=False, dpi=300.0, **kwargs)`

Generate a barcode rank plot, which shows the total UMIs against barcode rank (in descending order with respect to total UMIs)

**Parameters**

- **data** (*AnnData* or *UnimodalData* or *MultimodalData* object) – The main data object.
- **panel\_size** (tuple, optional (default: (6, 4))) – The plot size (width, height) in inches.
- **return\_fig** (bool, optional, default: `False`) – Return a `Figure` object if `True`; return `None` otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

**Return type**

Optional[Figure]

**Returns**

- 
- *Figure* object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

**Examples**

```
>>> fig = pg.rank_plot(data, dpi = 500)
```

**pegasus.ridgeplot**

`pegasus.ridgeplot`(*data*, *features*, *matrix\_key*=None, *donor\_attr*=None, *qc\_attr*=None, *overlap*=0.5, *left\_adjust*=0.35, *panel\_size*=(6, 4), *return\_fig*=False, *dpi*=300.0, *\*\*kwargs*)

Generate ridge plots, up to 8 features can be shown in one figure.

**Parameters**

- **data** (*UnimodalData* or *MultimodalData* object) – Data matrix.
- **features** (*str* or *List[str]*) – One or more features to display.
- **matrix\_key** (*str*, optional, default None) – Which matrix to search features for. If None, use the current matrix.
- **donor\_attr** (*str*, optional, default None) – If not None, *features* must contain only one feature, plot this feature by donor indicated as *donor\_attr*.
- **qc\_attr** (*str*, optional, default None) – If not None, only `data.obs[qc_attr] == True` are used.
- **overlap** (*float*, default 0.5) – Overlap between adjacent ridge plots (top and bottom).
- **left\_adjust** (*float*, default 0.35) – Left margin for displaying labels.
- **panel\_size** (*tuple*, optional (default: (6, 4))) – The plot size (width, height) in inches.
- **return\_fig** (*bool*, optional, default: False) – Return a *Figure* object if True; return None otherwise.
- **dpi** (*float*, optional, default: 300.0) – The resolution in dots per inch.

**Return type**

Optional[Figure]

**Returns**

- 
- *Figure* object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

## Examples

```
>>> fig = pg.ridgeplot(data, features = ['CD8', 'CD4', 'CD3'], dpi = 500)
>>> fig = pg.ridgeplot(data, features = 'CD3', donor_attr = 'assignment', dpi = 500)
```

## pegasus.wordcloud

`pegasus.wordcloud(data, factor, max_words=20, random_state=0, colormap='hsv', width=800, height=400, panel_size=(6, 4), return_fig=False, dpi=300.0, **kwargs)`

Generate one word cloud image for factor (starts from 0) in `data.uns['W']`.

### Parameters

- **data** (AnnData or UnimodalData or MultimodalData object) – The main data object.
- **factor** (int) – Which factor to plot. factor starts from 0.
- **max\_words** (int, optional, default: 20) – Maximum number of genes to show in the image.
- **random\_state** (int, optional, default: 0) – Random seed passing to WordCloud function.
- **colormap** (str, optional, default: hsv) – Color map for plotting words.
- **width** (int, optional, default: 800) – Canvas width.
- **height** (int, optional, default: 400) – Canvas height.
- **panel\_size** (tuple, optional, default: (6, 4)) – The plot size (width, height) in inches.
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

### Return type

Optional[Figure]

### Returns

- *Figure* object – A `matplotlib.figure.Figure` object containing the dot plot if `return_fig == True`

## Examples

```
>>> fig = pg.wordcloud(data, factor=0)
```

## pegasus.plot\_gsea

`pegasus.plot_gsea(data, gsea_keyword, alpha=0.1, top_n=20, panel_size=(6, 4), return_fig=False, dpi=300.0, **kwargs)`

Generate GSEA barplots

### Parameters

- **data** (UnimodalData or MultimodalData object) – The main data object.
- **gsea\_keyword** (str) – Keyword in data.uns that stores the fGSEA results in pandas data frame.
- **alpha** (float, optional, default: 0.1) – False discovery rate threshold.
- **top\_n** (int, optional, default: 20) – Only show top\_n up/down regulated pathways.
- **panel\_size** (tuple, optional (default: (6, 4))) – The plot size (width, height) in inches.
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

### Return type

Optional[Figure]

### Returns

- *Figure* object – A matplotlib.figure.Figure object containing the dot plot if `return_fig == True`. Each figure contains two panels. The top panel shows up-regulated pathways (color: red) and the bottom panel shows down-regulated pathways (color: green).

## Examples

```
>>> fig = pg.plot_gsea(data, 'fgsea_out', dpi = 500)
```

## pegasus.elbowplot

`pegasus.elbowplot(data, rep='pca', pval='0.05', panel_size=(6, 4), return_fig=False, dpi=300.0, **kwargs)`

Generate Elbowplot and suggest n\_comps to select based on random matrix theory (see `utils.largest_variance_from_random_matrix`).

### Parameters

- **data** (UnimodalData or MultimodalData object.) – The main data object.
- **rep** (str, optional, default: pca) – Representation to consider, either “pca” or “tsvd”.
- **pval** (str, optional (default: “0.05”).) – P value cutoff on the null distribution (random matrix), choosing from “0.01” and “0.05”.
- **top\_n** (int, optional, default: 20) – Only show top\_n up/down regulated pathways.
- **panel\_size** (tuple, optional (default: (6, 4))) – The plot size (width, height) in inches.

- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.

**Return type**

Optional[Figure]

**Returns**

- 
- *Figure* object – A matplotlib.figure.Figure object containing the dot plot if return\_fig == True.
- Update data.uns –
  - {rep}\_ncomps: Recommended components to pick.

**Examples**

```
>>> fig = pg.elbowplot(data, dpi = 500)
```

**Pseudo-bulk analysis**

<code>pseudobulk</code> (data, sample[, attrs, mat_key, ...])	Generate Pseudo-bulk count matrices.
<code>deseq2</code> (pseudobulk, design, contrast[, ...])	Perform Differential Expression (DE) Analysis using DESeq2 on pseudobulk data.
<code>pseudo.markers</code> (pseudobulk[, head, de_key, alpha])	Extract pseudobulk DE results into a human readable structure.
<code>pseudo.write_results_to_excel</code> (results, ...)	Write pseudo-bulk DE analysis results into a Excel work-book.
<code>pseudo.volcano</code> (pseudobulk[, de_key, ...])	Generate Volcano plots (-log10 p value vs.

**pegasus.pseudobulk**

`pegasus.pseudobulk`(data, sample, attrs=None, mat\_key='counts', cluster=None)

Generate Pseudo-bulk count matrices.

**Parameters**

- **data** (MultimodalData or UnimodalData object) – Annotated data matrix with rows for cells and columns for genes.
- **sample** (str) – Specify the cell attribute used for aggregating pseudo-bulk data. Key must exist in data.obs.
- **attrs** (str or List[str], optional, default: None) – Specify additional cell attributes to remain in the pseudo bulk data. If set, all attributes' keys must exist in data.obs. Notice that for a categorical attribute, each pseudo-bulk's value is the one of highest frequency among its cells, and for a numeric attribute, each pseudo-bulk's value is the mean among its cells.
- **mat\_key** (str, optional, default: counts) – Specify the single-cell count matrix used for aggregating pseudo-bulk counts: If specified, use the count matrix with key mat\_key from matrices of data; otherwise, default is counts.

- **cluster** (str, optional, default: None) – If set, additionally generate pseudo-bulk matrices per cluster specified in `data.obs[cluster]`.

**Return type**

UnimodalData

**Returns**

- A UnimodalData object `udata` containing pseudo-bulk information –
  - It has the following count matrices:
    - \* `X`: The pseudo-bulk count matrix over all cells.
    - \* If `cluster` is set, a number of pseudo-bulk count matrices of cells belonging to the clusters, respectively.
  - `udata.obs`: It contains pseudo-bulk attributes aggregated from the corresponding single-cell attributes.
  - `udata.var`: Gene names and Ensembl IDs are maintained.
- Update data –
  - Add the returned UnimodalData object above to `data` with key `<sample>-pseudobulk`, where `<sample>` is replaced by the actual value of `sample` argument.

**Examples**

```
>>> pg.pseudobulk(data, sample="Channel")
```

**pegasus.deseq2**

`pegasus.deseq2(pseudobulk, design, contrast, de_key='deseq2', replaceOutliers=True)`

Perform Differential Expression (DE) Analysis using DESeq2 on pseudobulk data. This function calls R package DESeq2, requiring DESeq2 in R installed.

DE analysis will be performed on all pseudo-bulk matrices in `pseudobulk`.

**Parameters**

- **pseudobulk** (UnimodalData) – Pseudobulk data with rows for samples and columns for genes. If `pseudobulk` contains multiple matrices, DESeq2 will apply to all matrices.
- **design** (str) – Design formula that will be passed to DESeq2
- **contrast** (Tuple[str, str, str]) – A tuple of three elements passing to DESeq2: a factor in design formula, a level in the factor as numerator of fold change, and a level as denominator of fold change.
- **de\_key** (str, optional, default: "deseq2") – Key name of DE analysis results stored. For `cluster.X`, stored key will be `cluster.de_key`
- **replaceOutliers** (bool, optional, default: True) – If execute DESeq2's `replaceOutliers` step. If set to False, we will set `minReplicatesForReplace=Inf` in DESeq function and set `cooksCutoff=False` in `results` function.

**Return type**

None

**Returns**

- None
- Update `pseudobulk.varm` – `pseudobulk.varm[de_key]`: DE analysis result for pseudo-bulk count matrix. `pseudobulk.varm[cluster.de_key]`: DE results for cluster-specific pseudo-bulk count matrices.

**Examples**

```
>>> pg.deseq2(pseudobulk, '~gender', ('gender', 'female', 'male'))
```

**pegasus.pseudo.markers**

`pegasus.pseudo.markers(pseudobulk, head=None, de_key='deseq2', alpha=0.05)`

Extract pseudobulk DE results into a human readable structure.

This function extracts information from `data.varm[de_key]`, and return as a human readable dictionary of pandas DataFrame objects.

**Parameters**

- **pseudobulk** (UnimodalData) – Pseudobulk data matrix with rows for cells and columns for genes.
- **head** (int, optional, default: None) – List only top head genes. If None, show any DE genes.
- **de\_key** (str, optional, default, `deseq2`) – Keyword of DE result stored in `data.varm`.
- **alpha** (float, optional, default: 0.05) – q-value threshold for getting significant DE genes. Only those with q-value of MWU test no less than `alpha` are significant, and thus considered as DE genes.

**Returns**

**results** – A Python dictionary containing DE results. This dictionary contains two keywords: 'up' and 'down'. 'up' refers to up-regulated genes, which should have 'log2FoldChange' > 0.5. 'down' refers to down-regulated genes, which should have 'log2FoldChange' < 0.5.

**Return type**

Dict[str, pd.DataFrame]

**Examples**

```
>>> markers = pg.pseudo.markers(pseudobulk)
```

### pegasus.pseudo.write\_results\_to\_excel

`pegasus.pseudo.write_results_to_excel(results, output_file, ndigits=3)`

Write pseudo-bulk DE analysis results into a Excel workbook.

#### Parameters

- **results** (Dict[str, pd.DataFrame]) – DE marker dictionary generated by `pg.pseudo.markers`.
- **output\_file** (str) – File name to which the marker dictionary is written.
- **ndigits** (int, optional, default: 3) – Round non p-values and q-values to `ndigits` after decimal point in the excel.

#### Return type

None

#### Returns

- None
- Marker information is written to file with name `output_file`. In the generated Excel workbook, each tab stores DE result of up (**up**) / down (**down**)-regulated genes.

### Examples

```
>>> pg.pseudo.write_results_to_excel(marker_dict, "result.de.xlsx")
```

### pegasus.pseudo.volcano

`pegasus.pseudo.volcano(pseudobulk, de_key='deseq2', qval_threshold=0.05, log2fc_threshold=1.0, top_n=20, panel_size=(6, 4), return_fig=False, dpi=300.0)`

Generate Volcano plots (-log10 p value vs. log2 fold change) for visualizing DE results.

#### Parameters

- **pseudobulk** (UnimodalData object.) – Pseudobulk data matrix.
- **de\_key** (str, optional, default: `deseq2`) – The varm keyword for DE results. `data.varm[de_key]` should store the full DE result table.
- **qval\_threshold** (float, optional, default: 0.05.) – Selected FDR rate. A horizontal line indicating this rate will be shown in the figure.
- **log2fc\_threshold** (float, optional, default: 1.0) – Log2 fold change threshold to highlight biologically interesting genes. Two vertical lines representing negative and positive log2 fold change will be shown.
- **top\_n** (int, optional, default: 20) – Number of top DE genes to show names. Genes are ranked by Log2 fold change.
- **panel\_size** (Tuple[float, float], optional, default: (6, 4)) – The size (width, height) in inches of figure.
- **return\_fig** (bool, optional, default: False) – Return a Figure object if True; return None otherwise.
- **dpi** (float, optional, default: 300.0) – The resolution in dots per inch.



**Return type**

Optional[Figure]

**Returns**

- 
- Figure object – A matplotlib.figure.Figure object containing the dot plot if `return_fig == True`

**Examples**

```
>>> pg.pseudo.volcano(pseudobulk, dpi=200)
```

**Demultiplexing**

<code>estimate_background_probs</code> (hashing_data[, ...])	For cell-hashing data, estimate antibody background probability using KMeans algorithm.
<code>demultiplex</code> (rna_data, hashing_data[, ...])	Demultiplexing cell/nucleus-hashing data, using the estimated antibody background probability calculated in <code>demuxEM.estimate_background_probs</code> .
<code>attach_demux_results</code> (input_rna_file, rna_data)	Write demultiplexing results into raw gene expression matrix.

**pegasus.estimate\_background\_probs**

`pegasus.estimate_background_probs`(*hashing\_data*, *random\_state=0*)

For cell-hashing data, estimate antibody background probability using KMeans algorithm.

**Parameters**

- **hashing\_data** (UnimodalData) – Annotated data matrix for antibody.
- **random\_state** (int, optional, default: 0) – Random seed set for reproducing results.

**Return type**

None

**Returns**

- None
- **Update ``hashing\_data``** (*Filtered cell barcodes with 0 counts*)
- Update `hashing_data.uns` –
  - `hashing_data.uns["background_probs"]`: estimated antibody background probability.

### Example

```
>>> estimate_background_probs(hashing_data)
```

### pegasus.demultiplex

```
pegasus.demultiplex(rna_data, hashing_data, min_signal=10.0, alpha=0.0, alpha_noise=1.0, tol=1e-06,
                    n_threads=1)
```

Demultiplexing cell/nucleus-hashing data, using the estimated antibody background probability calculated in `demuxEM.estimate_background_probs`.

#### Parameters

- **rna\_data** (UnimodalData) – Data matrix for gene expression matrix.
- **hashing\_data** (UnimodalData) – Data matrix for HTO count matrix.
- **min\_signal** (float, optional, default: 10.0) – Any cell/nucleus with less than `min_signal` hashtags from the signal will be marked as `unknown`.
- **alpha** (float, optional, default: 0.0) – The Dirichlet prior concentration parameter (`alpha`) on samples. An `alpha` value  $< 1.0$  will make the prior sparse.
- **alpha\_noise** (float, optional, default: 1.0) – The Dirichlet prior concentration parameter on the background noise.
- **tol** (float, optional, default:  $1e-6$ ) – Threshold used for the EM convergence.
- **n\_threads** (int, optional, default: 1) – Number of threads to use. Must be a positive integer.

#### Returns

- None
- Update `data.obs` –
  - `data.obs["demux_type"]`: Demultiplexed types of the cells. Either `singlet`, `doublet`, or `unknown`.
  - `data.obs["assignment"]`: Assigned samples of origin for each cell barcode.
  - `data.obs["assignment.dedup"]`: Only exist if one sample name can correspond to multiple feature barcodes. In this case, each feature barcode is assigned a unique sample name.

### Examples

```
>>> demultiplex(rna_data, hashing_data)
```

## pegasus.attach\_demux\_results

`pegasus.attach_demux_results(input_rna_file, rna_data)`

Write demultiplexing results into raw gene expression matrix.

### Parameters

- **input\_rna\_file** (str) – Input file for the raw gene count matrix.
- **rna\_data** (UnimodalData) – Processed gene count matrix containing demultiplexing results

### Return type

MultimodalData

### Returns

- MultimodalData
- *A multimodal data object.*

## Examples

```
>>> data = attach_demux_results('raw_data.h5', rna_data)
```

## Miscellaneous

<code>search_genes(data, gene_list[, rec_key, measure])</code>	Extract and display gene expressions for each cluster.
<code>search_de_genes(data, gene_list[, rec_key, ...])</code>	Extract and display differential expression analysis results of markers for each cluster.
<code>find_outlier_clusters(data, cluster_attr, ...)</code>	Using MWU test to detect if any cluster is an outlier regarding one of the qc attributes: n_genes, n_counts, percent_mito.
<code>find_markers(data, label_attr[, de_key, ...])</code>	Find markers using gradient boosting method.

## pegasus.search\_genes

`pegasus.search_genes(data, gene_list, rec_key='de_res', measure='percentage')`

Extract and display gene expressions for each cluster.

This function helps to see marker expressions in clusters via the interactive python environment.

### Parameters

- **data** (MultimodalData or UnimodalData object) – Annotated data matrix containing the expression matrix and differential expression results.
- **gene\_list** (List[str]) – A list of gene symbols.
- **rec\_key** (str, optional, default: "de\_res") – Keyword of DE analysis result stored in data.varm.
- **measure** (str, optional, default: "percentage") –  
Can be either "percentage" or "mean\_logExpr":

- percentage shows the percentage of cells expressed the genes;
- mean\_logExpr shows the mean log expression.

**Returns**

A data frame containing marker expressions in each cluster.

**Return type**

pandas.DataFrame

**Examples**

```
>>> results = pg.search_genes(adata, ['CD3E', 'CD4', 'CD8'])
```

**pegasus.search\_de\_genes**

`pegasus.search_de_genes(data, gene_list, rec_key='de_res', de_test='fisher', de_alpha=0.05, thre=1.5)`

Extract and display differential expression analysis results of markers for each cluster.

**This function helps to see if markers are up or down regulated in each cluster via the interactive python environment:**

- ++ indicates up-regulated and fold change  $\geq$  threshold;
- + indicates up-regulated but fold change  $<$  threshold;
- -- indicates down-regulated and fold change  $\leq 1 / \text{threshold}$ ;
- - indicates down-regulated but fold change  $> 1 / \text{threshold}$ ;
- ? indicates not differentially expressed.

**Parameters**

- **data** (MultimodalData or UnimodalData object) – Annotated data matrix containing the expression matrix and differential expression results.
- **gene\_list** (List[str]) – A list of gene symbols.
- **rec\_key** (str, optional, default: "de\_res") – Keyword of DE analysis result stored in data.varm.
- **de\_test** (str, optional, default: "fisher") – Differential expression test to look at, could be either t, fisher or mwu.
- **de\_alpha** (float, optional, default: 0.05) – False discovery rate.
- **thre** (float, optional, default: 1.5) – Fold change threshold to determine if the marker is a strong DE (++ or --) or weak DE (+ or -).

**Returns**

A data frame containing marker differential expression results for each cluster.

**Return type**

pandas.DataFrame

## Examples

```
>>> df = pegasus.misc.search_de_genes(adata, ['CD3E', 'CD4', 'CD8'], thre = 2.0)
```

### pegasus.find\_outlier\_clusters

`pegasus.find_outlier_clusters(data, cluster_attr, qc_attr)`

Using MWU test to detect if any cluster is an outlier regarding one of the qc attributes: `n_genes`, `n_counts`, `percent_mito`.

#### Parameters

- **data** (MultimodalData or UnimodalData object) – Annotated data matrix with rows for cells and columns for genes.
- **cluster\_attr** (str) – Attribute in `data.obs` representing cluster assignment, e.g. `'louvain_labels'`
- **qc\_attr** (str) – One of the QC attribute, choosing from `'n_genes'`, `'n_counts'` and `'percent_mito'`.

#### Return type

DataFrame

### pegasus.find\_markers

`pegasus.find_markers(data, label_attr, de_key='de_res', n_jobs=-1, min_gain=1.0, random_state=0, remove_ribo=False)`

Find markers using gradient boosting method.

#### Parameters

- **data** (anndata.AnnData) – Annotated data matrix with rows for cells and columns for genes.
- **label\_attr** (str) – Cluster labels used for finding markers. Must exist in `data.obs`.
- **de\_key** (str, optional, default: `"de_res"`) – Keyword of DE analysis result stored in `data.varm`.
- **n\_jobs** (int, optional, default: `-1`) – Number of threads to used. `-1` refers to using all physical CPU cores.
- **min\_gain** (float, optional, default: `1.0`) – Only report genes with a feature importance score (in gain) of at least `min_gain`.
- **random\_state** (int, optional, default: `0`) – Random seed set for reproducing results.
- **remove\_ribo** (bool, optional, default: `False`) – If `True`, remove ribosomal genes with either RPL or RPS as prefixes.

#### Returns

**markers** – A Python dictionary containing marker information in structure `dict[cluster_id]['up' or 'down'][dataframe]`.

#### Return type

Dict[str, Dict[str, List[str]]]

## Examples

```
>>> marker_dict = pg.find_markers(adata, label_attr = 'leiden_labels')
```

## 1.2.6 Release Notes

---

**Note:** Also see the release notes of [PegasusIO](#).

---

### Version 1.8

#### 1.8.1 August 23, 2023

- Bug fix in cell marker JSON files for `infer_cell_types` function.

#### 1.8.0 July 21, 2023

##### New Feature and Improvement

- Update `human_immune` and `human_lung` marker sets.
- Add `mouse_liver` marker set.
- Add `split_one_cluster` function to subcluster one cluster into a specified number of subclusters.
- Update `neighbors` function to set `use_cache=False` by default, and adjust `K` to `min(K, int(sqrt(n_samples)))`. [PR 272]
- In `infer_doublets` function, argument `manual_correction` now accepts a float number threshold specified by users for cut-off. [PR 275]

##### Bug Fix

- Fix divide by zero issue in `integrative_nmf` function. [PR 258]
- Compatibility with Pandas v2.0. [PR 261]
- Allow `infer_doublets` to use any count matrix with key name specified by users. [PR 268 Thanks to [Donghoon Lee](#)]

### Version 1.7

#### 1.7.1 July 29, 2022

- Make command-line tool and API work with PegasusIO v0.7.0.
- Rename function `arcsinh_transform` to `arcsinh`, and allow set a user-specified count matrix for the transformation.
- Add functions `log1p` and `normalize` for preprocessing the count matrix.

## 1.7.0 July 5, 2022

### New Features

- Add `pegasus.elbowplot` function to generate elbowplot, with an automated suggestion on number of PCs to be selected based on random matrix theory ([Johnstone 2001] and [Shekhar 2022]).
- Add `arcsinh_transform` function for arcsinh transformation on the count matrix.

### Improvement

- Function `nearest_neighbors` has additional argument `n_comps` to allow use part of the components from the source embedding for calculating the nearest neighbor graph.
- Add `n_comps` argument for `run_harmony`, `tsne`, `umap`, and `fle` (argument name is `rep_ncomps`) functions to allow select part of the components from the source embedding.
- Function `scatter` can plot multiple components and bases.

## Version 1.6

### 1.6.0 April 16, 2022

#### New Features

- Add support for `scVI-tools`:
  - Function `pegasus.run_scvi`, which is a wrapper of `scVI` for data integration.
  - Add a dedicated section for `scVI` method in Pegasus [batch correction tutorial](#).
  - Function `pegasus.train_scarches_scanvi` and `pegasus.predict_scarches_scanvi` to wrap `scArches` for transfer learning on single-cell data labels.

## Version 1.5

### 1.5.0 March 9, 2022

#### New Features

- Spatial data analysis:
  - Enable `pegasus.read_input` function to load 10x Visium data: set `file_type="visium"` option.
  - Add `pegasus.spatial` function to generate spatial plot for 10x Visium data.
  - Add *Spatial Analysis Tutorial* in [Tutorials](#).
- Pseudobulk analysis: see [summary](#)
  - Add `pegasus.pseudobulk` function to generate pseudobulk matrix.
  - Add `pegasus.deseq2` function to perform pseudobulk differential expression (DE) analysis, which is a Python wrapper of `DESeq2`.
    - \* Requires `rpyp2` and the original `DESeq2` R package installed.
  - Add `pegasus.pseudo.markers`, `pegasus.pseudo.write_results_to_excel` and `pegasus.pseudo.volcano` functions for processing pseudobulk DE results.
  - Add *Pseudobulk Analysis Tutorial* in [Tutorials](#).

- Add `pegasus.fgsea` function to perform Gene Set Enrichment Analysis (GSEA) on DE results and plotting, which is a Python wrapper of `fgsea`.
  - Requires `ipy2` and the original `fgsea` R package installed.

### API Changes

- Function `correct_batch`, which implements the L/S adjustment batch correction method, is obsolete. We recommend using `run_harmony` instead, which is also the default of `--correct-batch-effect` option in `pegasus cluster` command.
- `pegasus.highly_variable_features` allows specify custom attribute key for batches (`batch` option), and thus remove `consider_batch` option. To select HVGs without considering batch effects, simply use the default, or equivalently use `batch=None` option.
- Add `dist` option to `pegasus.neighbors` function to allow use distance other than L2. (Contribution by [hoondy](#) in [PR 233](#))
  - Available options: `l2` for L2 (default), `ip` for inner product, and `cosine` for cosine similarity.
- The kNN graph returned by `pegasus.neighbors` function is now stored in `obsm` field of the data object, no longer in `uns` field. Moreover, the kNN affinity matrix is stored in `obsp` field.

### Improvements

- Adjust `pegasus.write_output` function to work with [Zarr](#) v2.11.0+.

## Version 1.4

### 1.4.5 January 24, 2022

- Make several dependencies optional to meet with different use cases.
- Adjust to `umap-learn` v0.5.2 interface.

### 1.4.4 October 22, 2021

- Use `PegasusIO` v0.4.0+ for data manipulation.
- Add `calculate_z_score` function to calculate standardized z-scored count matrix.

### 1.4.3 July 25, 2021

- Allow `run_harmony` function to use GPU for computation.

### 1.4.2 July 19, 2021

- Bug fix for `--output-h5ad` and `--citeseq` options in `pegasus cluster` command.



### 1.4.1 July 17, 2021

- Add NMF-related options to `pegasus cluster` command.
- Add word cloud graph plotting feature to `pegasus plot` command.
- `pegasus aggregate_matrix` command now allow sample-specific filtration with parameters set in the input CSV-format sample sheet.
- Update doublet detection method: `infer_doublets` and `mark_doublets` functions.
- Bug fix.

### 1.4.0 June 24, 2021

- Add `nmf` and `integrative_nmf` functions to compute NMF and iNMF using `nmf-torch` package; `integrative_nmf` supports quantile normalization proposed in the *LIGER* papers ([Welch19], [Gao21]).
- Change the parameter defaults of function `qc_metrics`: Now all defaults are `None`, meaning not performing any filtration on cell barcodes.
- In **Annotate Clusters** API functions:
  - Improve human immune cell markers and auto cell type assignment for human immune cells. (`infer_cell_types` function)
  - Update mouse brain cell markers (`infer_cell_types` function)
  - `annotate` function now adds annotation as a categorical variable and sort categories in natural order.
- Add `find_outlier_clusters` function to detect if any cluster is an outlier regarding one of the qc attributes (`n_genes`, `n_counts`, `percent_mito`) using MWU test.
- In **Plotting** API functions:
  - `scatter` function now plots all cells if `attrs == None`; Add `fix_corners` option to fix the four corners when only a subset of cells is shown.
  - Fix a bug in `heatmap` plotting function.
- Fix bugs in functions `spectral_leiden` and `spectral_louvain`.
- Improvements:
  - Support *umap-learn* v0.5+. (`umap` and `net_umap` functions)
  - Update doublet detection algorithm. (`infer_doublets` function)
  - Improve message reporting execution time spent each step. (Pegasus command line tool)

## Version 1.3

### 1.3.0 February 2, 2021

- Make PCA more reproducible. No need to keep options for robust PCA calculation:
  - In `pca` function, remove argument `robust`.
  - In `infer_doublets` function, remove argument `robust`.
  - In `pegasus cluster` command, remove option `--pca-robust`.
- Add control on number of parallel threads for OpenMP/BLAS.

- Now `n_jobs = -1` refers to use all physical CPU cores instead of logical CPU cores.
- Remove function `reduce_diffmap_to_3d`. In **pegasus cluster** command, remove option `--diffmap-to-3d`.
- Enhance `compo_plot` and `dotplot` functions' usability.
- Bug fix.

## Version 1.2

### 1.2.0 December 25, 2020

- tSNE support:
  - `tsne` function in API: Use [FIt-SNE](#) for tSNE embedding calculation. No longer support MulticoreTSNE.
  - Determine `learning_rate` argument in `tsne` more dynamically. ([\[Belkina19\]](#), [\[Kobak19\]](#))
  - By default, use PCA embedding for initialization in `tsne`. ([\[Kobak19\]](#))
  - Remove `net_tsne` and `fitsne` functions from API.
  - Remove `--net-tsne` and `--fitsne` options from **pegasus cluster** command.
- Add multimodal support on RNA and CITE-Seq data back: `--citeseq`, `--citeseq-umap`, and `--citeseq-umap-exclude` in **pegasus cluster** command.
- Doublet detection:
  - Add automated doublet cutoff inference to `infer_doublets` function in API. ([\[Li20-2\]](#))
  - Expose doublet detection to command-line tool: `--infer-doublets`, `--expected-doublet-rate`, and `--dbl-cluster-attr` in **pegasus cluster** command.
  - Add doublet detection tutorial.
- Allow multiple marker files used in cell type annotation: `annotate` function in API; `--markers` option in **pegasus annotate\_cluster** command.
- Rename `pc_regress_out` function in API to `regress_out`.
- Update the regress out tutorial.
- Bug fix.

## Version 1.1

### 1.1.0 December 7, 2020

- Improve doublet detection in Scrublet-like way using automatic threshold selection strategy: `infer_doublets`, and `mark_doublets`. Remove *Scrublet* from dependency, and remove `run_scrublet` function.
- Enhance performance of log-normalization (`log_norm`) and signature score calculation (`calc_signature_score`).
- In **pegasus cluster** command, add `--genome` option to specify reference genome name for input data of `dge`, `csv`, or `loom` format.
- Update [Regress out tutorial](#).
- Add `ridgeplot`.
- Improve plotting functions: `heatmap`, and `dendrogram`.

- Bug fix.

## Version 1.0

### 1.0.0 September 22, 2020

- New features:
  - Use `zarr` file format to handle data, which has a better I/O performance in general.
  - Multi-modality support:
    - \* Data are manipulated in Multi-modal structure in memory.
    - \* Support focus analysis on Unimodal data, and appending other Unimodal data to it. (`--focus` and `--append` options in `cluster` command)
  - Calculate signature / gene module scores. (`calc_signature_score`)
  - `Doublet detection` based on `Scrublet`: `run_scrublet`, `infer_doublets`, and `mark_doublets`.
  - Principal-Component-level regress out. (`pc_regress_out`)
  - Batch correction using `Scanorama`. (`run_scanorama`)
  - Allow DE analysis with sample attribute as condition. (Set `condition` argument in `de_analysis`)
  - Use static plots to show results (see `Plotting`):
    - \* Provide static plots: composition plot, embedding plot (e.g. tSNE, UMAP, FLE, etc.), dot plot, feature plot, and volcano plot;
    - \* Add more gene-specific plots: dendrogram, heatmap, violin plot, quality-control violin, HVF plot.
- Deprecations:
  - No longer support `h5sc` file format, which was the output format of `aggregate_matrix` command in Pegasus version `0.x`.
  - Remove `net_fitsne` function.
- API changes:
  - In cell quality-control, default percent of mitochondrial genes is changed from **10.0** to **20.0**. (`percent_mito` argument in `qc_metrics`; `--percent-mito` option in `cluster` command)
  - Move gene quality-control out of `filter_data` function to be a separate step. (`identify_robust_genes`)
  - DE analysis now uses MWU test by default, not t test. (`de_analysis`)
  - `infer_cell_types` uses MWU test as the default `de_test`.
- Performance improvement:
  - Speed up MWU test in DE analysis, which is inspired by `Presto`.
  - Integrate Fisher's exact test via Cython in DE analysis to improve speed.
- Other highlights:
  - Make I/O and count matrices aggregation a dedicated package `PegasusIO`.
  - Tutorials:
    - \* Update `Analysis tutorial`;

- \* Add 3 more tutorials: one on [plotting library](#), one on [batch correction and data integration](#), and one on [regress out](#).

### Version 0.x

#### 0.17.2 June 26, 2020

- Make Pegasus compatible with *umap-learn* v0.4+.
- Use *louvain* 0.7+ for Louvain clustering.
- Update tutorial.

#### 0.17.1 April 6, 2020

- Improve pegasus command-line tool log.
- Add human lung markers.
- Improve log-normalization speed.
- Provide robust version of PCA calculation as an option.
- Add signature score calculation API.
- Fix bugs.

#### 0.17.0 March 10, 2020

- Support *anndata* 0.7 and *pandas* 1.0.
- Better loom format output writing function.
- Bug fix on mtx format output writing function.
- Update human immune cell markers.
- Improve pegasus `scp_output` command.

#### 0.16.11 February 28, 2020

- Add `--remap-singlets` and `--subset-singlets` options to ‘cluster’ command.
- Allow reading loom file with user-specified batch key and black list.

#### 0.16.9 February 17, 2020

Allow reading h5ad file with user-specified batch key.

### 0.16.8 January 30, 2020

Allow input annotated loom file.

### 0.16.7 January 28, 2020

Allow input `mtx` files of more filename formats.

### 0.16.5 January 23, 2020

Add Harmony algorithm for data integration.

### 0.16.3 December 17, 2019

Add support for loading `mtx` files generated from BUStools.

### 0.16.2 December 8, 2019

Fix bug in ‘subcluster’ command.

### 0.16.1 December 4, 2019

Fix one bug in clustering pipeline.

### 0.16.0 December 3, 2019

- Change options in ‘aggregate\_matrix’ command: remove ‘–google-cloud’, add ‘–default-reference’.
- Fix bug in ‘–annotation’ option of ‘annotate\_cluster’ command.
- Fix bug in ‘net\_fle’ function with 3-dimension coordinates.
- Use **fisher** package version 0.1.9 or above, as modifications in our forked **fisher-modified** package has been merged into it.

### 0.15.0 October 2, 2019

Rename package to *PegasusPy*, with module name *pegasus*.

### **0.14.0 September 17, 2019**

Provide Python API for interactive analysis.

### **0.10.0 January 31, 2019**

Added ‘find\_markers’ command to find markers using LightGBM.

Improved file loading speed and enabled the parsing of channels from barcode strings for cellranger aggregated h5 files.

### **0.9.0 January 17, 2019**

In ‘cluster’ command, changed ‘–output-seurat-compatible’ to ‘–make-output-seurat-compatible’. Do not generate output\_name.seurat.h5ad. Instead, output\_name.h5ad should be able to convert to a Seurat object directly. In the seurat object, raw.data slot refers to the filtered count data, data slot refers to the log-normalized expression data, and scale.data refers to the variable-gene-selected, scaled data.

In ‘cluster’ command, added ‘–min-umis’ and ‘–max-umis’ options to filter cells based on UMI counts.

In ‘cluster’ command, ‘–output-filtration-results’ option does not require a spreadsheet name anymore. In addition, added more statistics such as median number of genes per cell in the spreadsheet.

In ‘cluster’ command, added ‘–plot-filtration-results’ and ‘–plot-filtration-figsize’ to support plotting filtration results. Improved documentation on ‘cluster command’ outputs.

Added ‘parquet’ command to transfer h5ad file into a parquet file for web-based interactive visualization.

### **0.8.0 November 26, 2018**

Added support for checking index collision for CITE-Seq/hashing experiments.

### **0.7.0 October 26, 2018**

Added support for CITE-Seq analysis.

### **0.6.0 October 23, 2018**

Renamed scrttools to scCloud.

Added demuxEM module for cell/nuclei-hashing.

### **0.5.0 August 21, 2018**

Fixed a problem related AnnData.

Added support for BigQuery.

### 0.4.0 August 2, 2018

Added mouse brain markers.

Allow aggregate matrix to take 'Sample' as attribute.

### 0.3.0 June 26, 2018

scrttools supports fast preprocessing, batch-correction, dimension reduction, graph-based clustering, diffusion maps, force-directed layouts, and differential expression analysis, annotate clusters, and plottings.

## 1.2.7 References

## 1.2.8 Contributors

Besides the Pegasus team, we would like to sincerely give our thanks to the following contributors to this project:

Name	Commits	Date	Note
Tariq Daouda	774c2cc	2019/10/17	Decorator for time logging and garbage collection.

## 1.2.9 Contact us

If you have any questions related to Pegasus, please feel free to contact us via [Cumulus Support Google Group](#).





## BIBLIOGRAPHY

- [Belkina19] A. C. Belkina, C. O. Ciccolella, R. Anno, R. Halpert, J. Spidlen, and J. E. Snyder-Cappione, “Automated optimized parameters for T-distributed stochastic neighbor embedding improve visualization and analysis of large datasets”, In *Nature Communications*, 2019.
- [Blondel08] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks”, In *Journal of Statistical Mechanics: Theory and Experiment*, 2008.
- [Büttner18] M. Büttner, et al., “A test metric for assessing single-cell RNA-seq batch correction”, In *Nature Methods*, 2018.
- [Gao21] Ch. Gao, et al., “Iterative single-cell multi-omic integration using online learning”, In *Nature Biotechnology*, 2021.
- [Hie19] B. Hie, B. Bryson, and B. Berger, “Efficient integration of heterogeneous single-cell transcriptomes using Scanorama”, In *Nature Biotechnology*, 2019.
- [Jacomy14] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, “ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software”, In *PLoS ONE*, 2014.
- [Kobak19] D. Kobak, and P. Berens, “Kobak, D., & Berens, P. (2019). The art of using t-SNE for single-cell transcriptomics”, In *Nature Communications*, 2019.
- [Korsunsky19] I. Korsunsky, et al., “Fast, sensitive and accurate integration of single-cell data with Harmony”, In *Nature Methods*, 2019.
- [Li20] B. Li, J. Gould, Y. Yang, et al., “Cumulus provides cloud-based data analysis for large-scale single-cell and single-nucleus RNA-seq”, In *Nature Methods*, 2020.
- [Li20-1] B. Li, “A streamlined method for signature score calculation”, In *Pegasus repository*, 2020.
- [Li20-2] B. Li, “Doublet detection in Pegasus”, In *Pegasus repository*, 2020.
- [Linderman19] G.C. Linderman, et al., “Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data”, In *Nature Methods*, 2019.
- [Jerby-Arnon18] L. Jerby-Arnon, et al., “A cancer cell program promotes T cell exclusion and resistance to checkpoint blockade”, In *Cell*, 2018.
- [Malkov16] Yu. A. Malkov, and D.A. Yashunin, “Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs”, In *arXiv*, 2016.
- [McInnes18] L. McInnes, J. Healy, and J. Melville, “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”, Preprint at *arXiv*, 2018.
- [Traag19] V. A. Traag, L. Waltman, and N. J. van Eck, “From Louvain to Leiden: guaranteeing well-connected communities”, In *Scientific Reports*, 2019.

- [Welch19] J. D. Welch, et al., “Single-Cell Multi-omic Integration Compares and Contrasts Features of Brain Cell Identity” In [Cell](#), 2019.
- [Wolock18] S.L. Wolock, R. Lopez, and A.M. Klein, “Scrublet: computational identification of cell doublets in single-cell transcriptomic Data”, In [Cell Systems](#), 2018.
- [Yang16] Z. Yang, and G. Michailidis, “A non-negative matrix factorization method for detecting modules in heterogeneous omics multi-modal data”, In [Bioinformatics](#), 2016.

## A

aggregate\_matrices() (in module pegasus), 37  
 annotate() (in module pegasus), 85  
 arcsinh() (in module pegasus), 44  
 attach\_demux\_results() (in module pegasus), 111

## C

calc\_kBET() (in module pegasus), 57  
 calc\_kSIM() (in module pegasus), 58  
 calc\_pseudotime() (in module pegasus), 60  
 calc\_signature\_score() (in module pegasus), 76  
 calculate\_z\_score() (in module pegasus), 50  
 cluster() (in module pegasus), 61  
 compo\_plot() (in module pegasus), 92

## D

de\_analysis() (in module pegasus), 81  
 demultiplex() (in module pegasus), 110  
 dendrogram() (in module pegasus), 97  
 deseq2() (in module pegasus), 106  
 diffmap() (in module pegasus), 59  
 dotplot() (in module pegasus), 96

## E

elbowplot() (in module pegasus), 104  
 estimate\_background\_probs() (in module pegasus), 109

## F

fgsea() (in module pegasus), 83  
 filter\_data() (in module pegasus), 41  
 find\_markers() (in module pegasus), 113  
 find\_outlier\_clusters() (in module pegasus), 113  
 fle() (in module pegasus), 69

## G

get\_filter\_stats() (in module pegasus), 41  
 get\_neighbors() (in module pegasus), 56

## H

heatmap() (in module pegasus), 95

highly\_variable\_features() (in module pegasus), 45  
 hvfplot() (in module pegasus), 99

## I

identify\_robust\_genes() (in module pegasus), 42  
 infer\_cell\_types() (in module pegasus), 84  
 infer\_doublets() (in module pegasus), 74  
 infer\_path() (in module pegasus), 60  
 integrative\_nmf() (in module pegasus), 52

## L

leiden() (in module pegasus), 63  
 loglp() (in module pegasus), 43  
 log\_norm() (in module pegasus), 42  
 louvain() (in module pegasus), 62

## M

mark\_doublets() (in module pegasus), 75  
 markers() (in module pegasus), 82  
 markers() (in module pegasus.pseudo), 107

## N

neighbors() (in module pegasus), 55  
 net\_fle() (in module pegasus), 72  
 net\_umap() (in module pegasus), 70  
 nmf() (in module pegasus), 48  
 normalize() (in module pegasus), 44

## P

pca() (in module pegasus), 47  
 plot\_gsea() (in module pegasus), 104  
 predict\_scarches\_scanvi() (in module pegasus), 80  
 pseudobulk() (in module pegasus), 105

## Q

qc\_metrics() (in module pegasus), 40  
 qcviolin() (in module pegasus), 99

## R

rank\_plot() (in module pegasus), 101

`read_input()` (*in module pegasus*), 36  
`regress_out()` (*in module pegasus*), 49  
`ridgeplot()` (*in module pegasus*), 102  
`run_harmony()` (*in module pegasus*), 50  
`run_scanorama()` (*in module pegasus*), 51  
`run_scvi()` (*in module pegasus*), 54

## S

`scatter()` (*in module pegasus*), 86  
`scatter_groups()` (*in module pegasus*), 89  
`search_de_genes()` (*in module pegasus*), 112  
`search_genes()` (*in module pegasus*), 111  
`select_features()` (*in module pegasus*), 46  
`spatial()` (*in module pegasus*), 91  
`spectral_leiden()` (*in module pegasus*), 66  
`spectral_louvain()` (*in module pegasus*), 65  
`split_one_cluster()` (*in module pegasus*), 64

## T

`train_scarches_scanvi()` (*in module pegasus*), 78  
`tsne()` (*in module pegasus*), 67

## U

`umap()` (*in module pegasus*), 68

## V

`violin()` (*in module pegasus*), 93  
`volcano()` (*in module pegasus*), 100  
`volcano()` (*in module pegasus.pseudo*), 108

## W

`wordcloud()` (*in module pegasus*), 103  
`write_output()` (*in module pegasus*), 37  
`write_results_to_excel()` (*in module pegasus*), 83  
`write_results_to_excel()` (*in module pegasus.pseudo*), 108