singlet Documentation

Release 0.5

Fabio Zanini

Feb 25, 2019

Contents

Single cell RNA-Seq analysis with quantitative phenotypes.

CHAPTER 1

Requirements

Python 3.4+ is required. Moreover, you will need:

- pyyaml
- numpy
- scipy
- pandas
- xarray
- scikit-learn
- matplotlib
- seaborn

1.1 Optional requirements

- umap (for UMAP dimensionality reduction)

Get those from pip or conda.

CHAPTER 2

Install

To get the latest **stable** version, use pip:

```
pip install singlet
```

To get the latest **development** version, clone the git repo and then call:

```
python3 setup.py install
```

CHAPTER 3

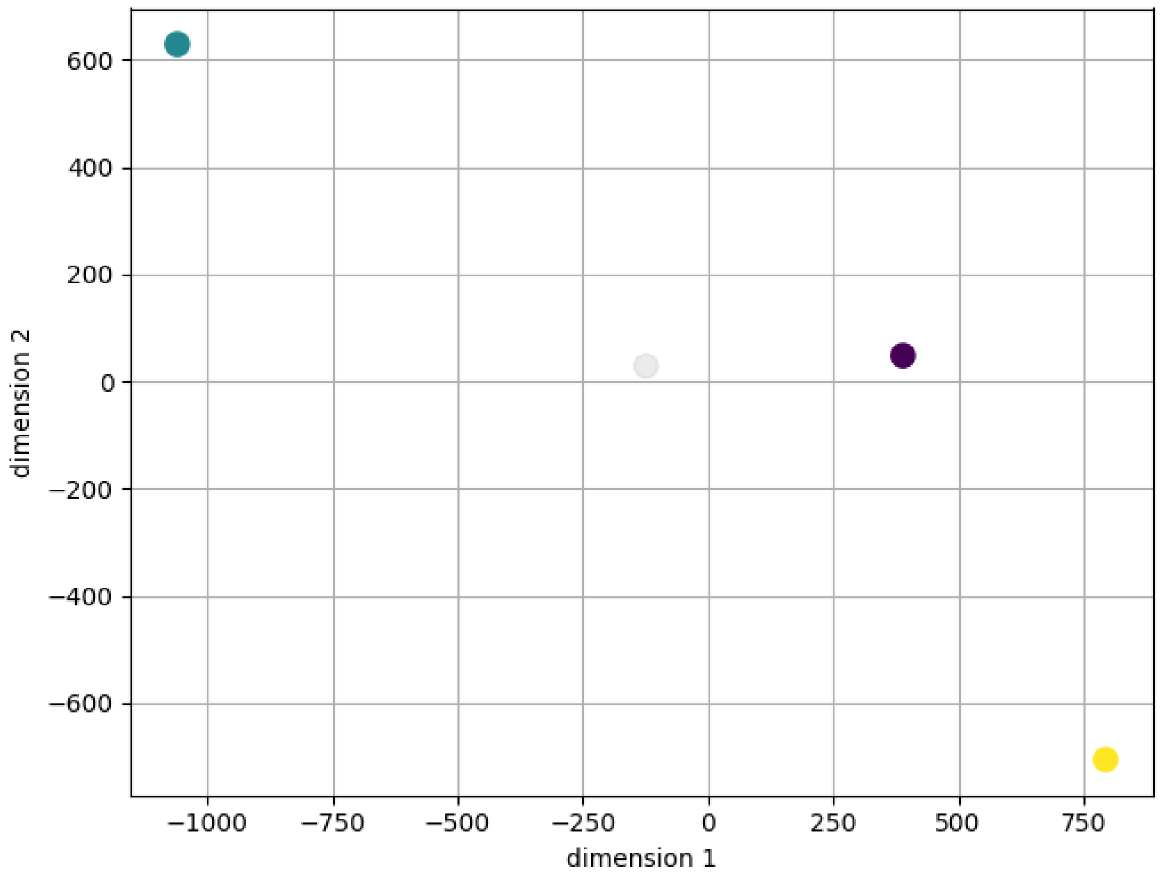
Usage example

You can have a look inside the *test* folder for examples. To start using the example dataset:

- Set the environment variable *SINGLET_CONFIG_FILENAME* to the location of the example YAML file
- Open a Python/IPython shell and type:

```python
from singlet.dataset import Dataset
ds = Dataset(
        samplesheet='example_sheet_tsv',
        counts_table='example_table_tsv')

ds.counts = ds.counts.iloc[:200]
vs = ds.dimensionality.tsne(
        n_dims=2,
        transform='log10',
        theta=0.5,
        perplexity=0.8)
ax = ds.plot.scatter_reduced_samples(
        vs,
        color_by='quantitative_phenotype_1_[A.U.]')
plt.show()
```

This will calculate a t-SNE embedding of the first 200 features and then show your samples in the reduced space. It should look like this:



Note: The figure looks different on OSX, but no worries, if you got there without errors chances are all is working correctly!

CHAPTER 4

Contents

4.1 Examples

4.1.1 Example: Feature Selection

It is typical in scRNA-Seq experiments to filter out features that are not expressed in any sample, or at low levels in very few samples. Moreover, of all remaining features, it is customary to select highly variable features for some applications such as dimensionality reduction.

```
from singlet.dataset import Dataset
ds = Dataset(
        samplesheet='example_sheet_tsv',
        counts_table='example_table_tsv')

ds.counts.normalize('counts_per_million', inplace=True)

# This selects only genes that are present at >= 5 counts per million in at least 2␣
↪samples
ds.feature_selection.expressed(
       n_samples=2,
       exp_min=5,
```

(continues on next page)

(continued from previous page)

```
        inplace=True)

# This selects highly variable features
ds.feature_selection.overdispersed_strata(
        inplace=True)
```

4.1.2 Example: Loom file

Loom files are becoming a common way of sharing single cell transcriptomic data. In a loom file, a counts table, a samplesheet, and a featuresheet are kept together inside a single file with an extension `.loom`. `singlet` supports reading from loom files via config files.

Your `singlet.yml` must contain a section such as:

```
datasets:
 ds1:
   path: xxx.loom
   format: loom
   axis_samples: columns
   index_samples: Cell
   index_features: Gene
```

Then you can load you `Dataset` easily:

```
from singlet.dataset import Dataset
ds = Dataset(dataset='ds1')
```

To export a `Dataset` to a loom file, you can use the method `to_dataset_file`:

```
from singlet.dataset import Dataset
ds = Dataset(dataset='ds1')
ds.to_dataset_file('xxx.loom')
```

4.1.3 Example: Classification of cell populations

A typical application of scRNA-Seq is classification of cell populations in a heterogeneous tissue. In this example, ~200 Peripheral Mononuclear Blood Cells (PBMCs) are classified using feature selection, dimensionality reduction, and unsupervised clustering.

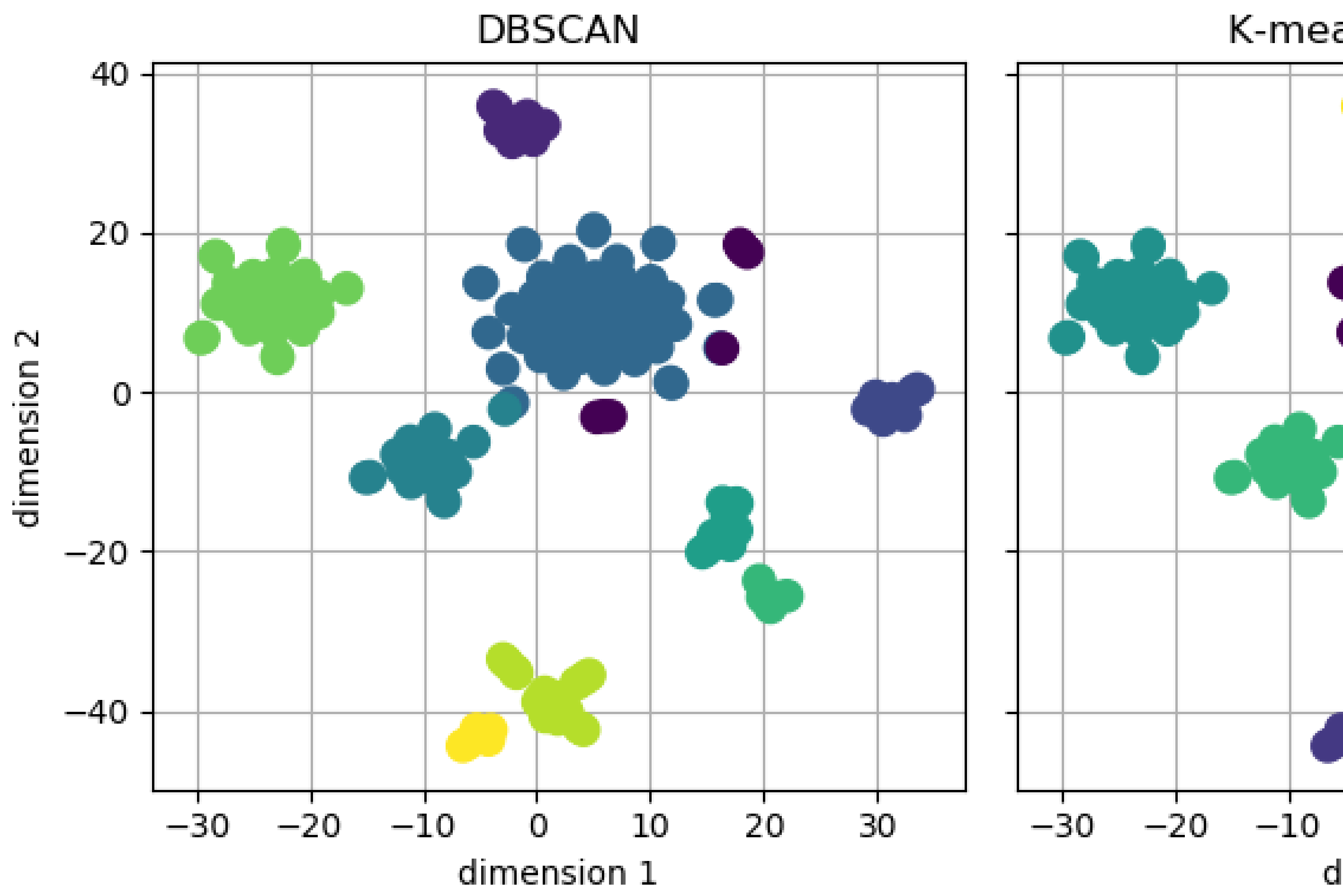
```
import matplotlib.pyplot as plt
from singlet.dataset import Dataset

ds = Dataset(counts_table='example_PBMC')

# Normalize
ds.counts.normalize(method='counts_per_million', inplace=True)
ds.counts.log(inplace=True)

# Select features
ds.feature_selection.expressed(n_samples=3, exp_min=1, inplace=True)
ds.feature_selection.overdispersed_strata(
        n_features_per_stratum=20,
        inplace=True)
```

(continues on next page)

(continued from previous page)

```
# Reduce dimensionality
vs = ds.dimensionality.tsne(
        n_dims=2,
        theta=0.5,
        perplexity=0.8)

# Reset the counts with the reduced values
ds.counts = vs.T

# Cluster
ds.samplesheet['dbscan'] = ds.cluster.dbscan(eps=5, axis='samples')
ds.samplesheet['kmeans'] = ds.cluster.kmeans(n_clusters=7, axis='samples')

# Plot t-SNE
fig, axs = plt.subplots(
        nrows=1, ncols=2, sharex=True, sharey=True,
        figsize=(8, 4))
ds.plot.scatter_reduced_samples(vs, color_by='dbscan', ax=axs[0], zorder=10)
ds.plot.scatter_reduced_samples(vs, color_by='kmeans', ax=axs[1], zorder=10)

axs[0].set_title('DBSCAN')
axs[1].set_title('K-means, 7 clusters')

plt.tight_layout()
plt.show()
```

You should get figures similar to the following one:



4.1.4 Example: Principal Component Analysis

Principal Component Analysis (PCA) is a popular dimensionality reduction method. Because outlier samples can strongly affect the results of PCA, *singlet* also implements a robust PCA version via Principal Component Pursuit (cite).

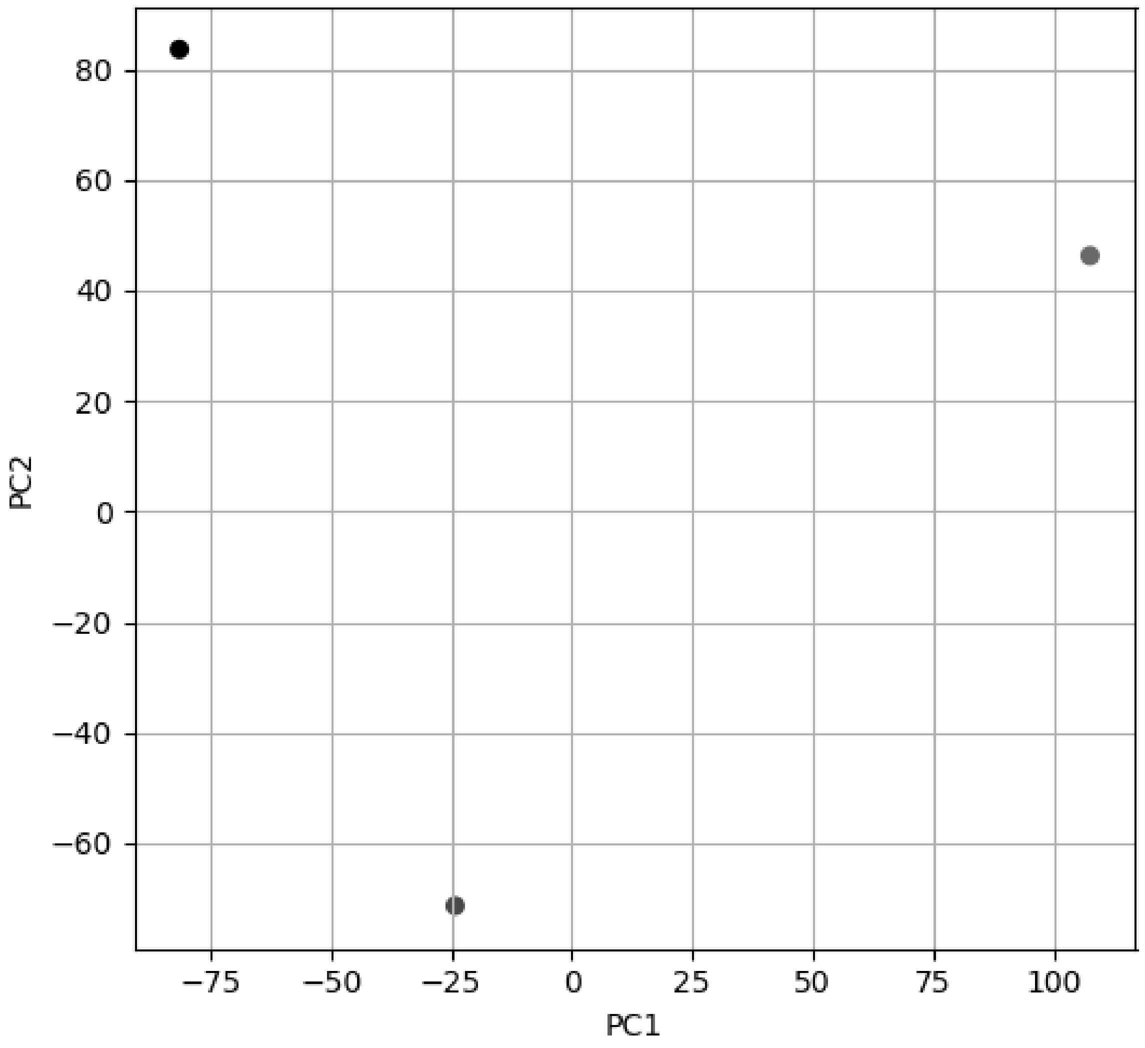
```
from singlet.dataset import Dataset
ds = Dataset(
        samplesheet='example_sheet_tsv',
        counts_table='example_table_tsv')

ds.counts.normalize('counts_per_million', inplace=True)
ds.counts = ds.counts.iloc[:200]

print('Calculate PCA')
vs = ds.dimensionality.pca(
        n_dims=2,
        transform='log10',
        robust=False)['vs']

print('Plot PCA')
```

(continues on next page)

(continued from previous page)

```
ax = ds.plot.scatter_reduced_samples(
        vs,
        color_by='ACTB')

plt.show()
```

You should get figures similar to the following ones:



4.1.5 Example: Quality controls

A typical task right off the bat in single-cell sequencing projects is to look at some statistics of the sequencing reads, for instance the number of reads for each cell (coverage), the fraction of mapped reads, and the abundance of spike-in controls and housekeeping genes.

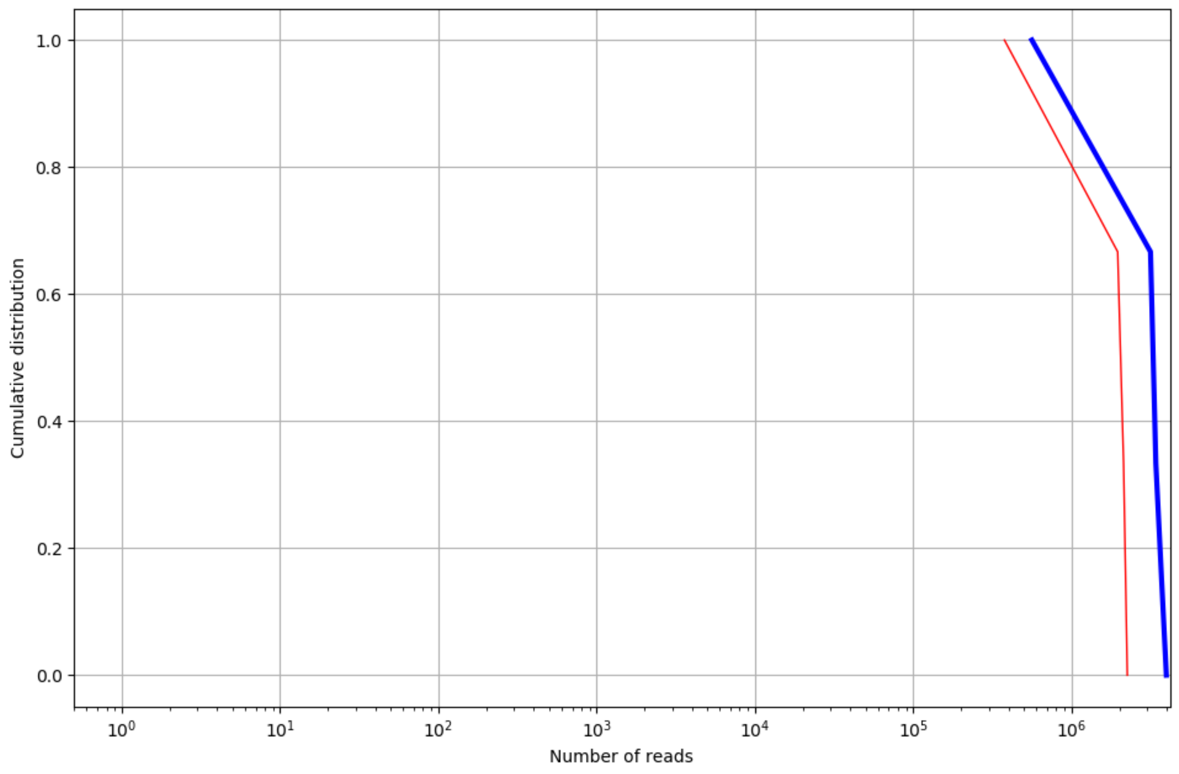
```
from singlet.dataset import Dataset
ds = Dataset(
```

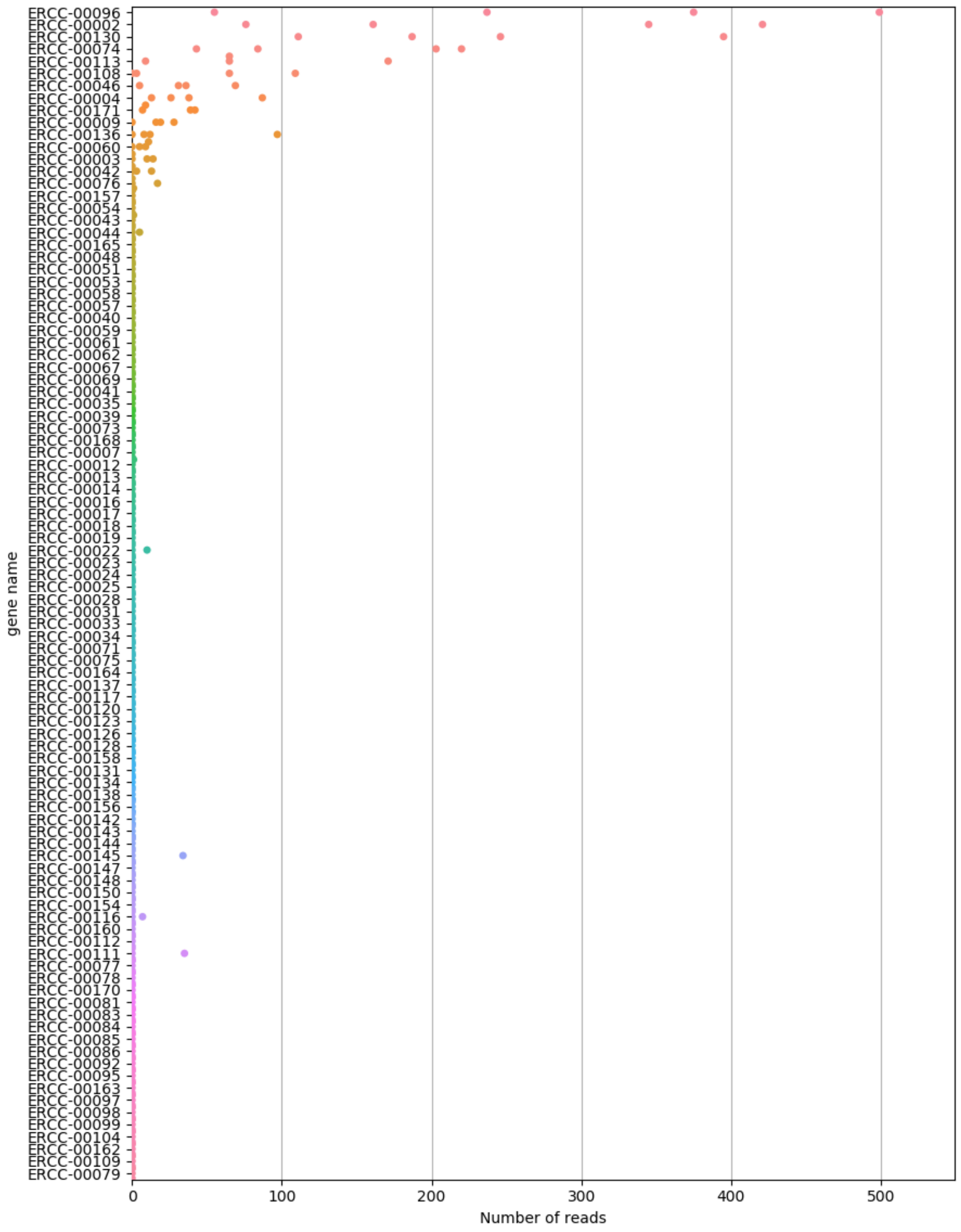
(continues on next page)

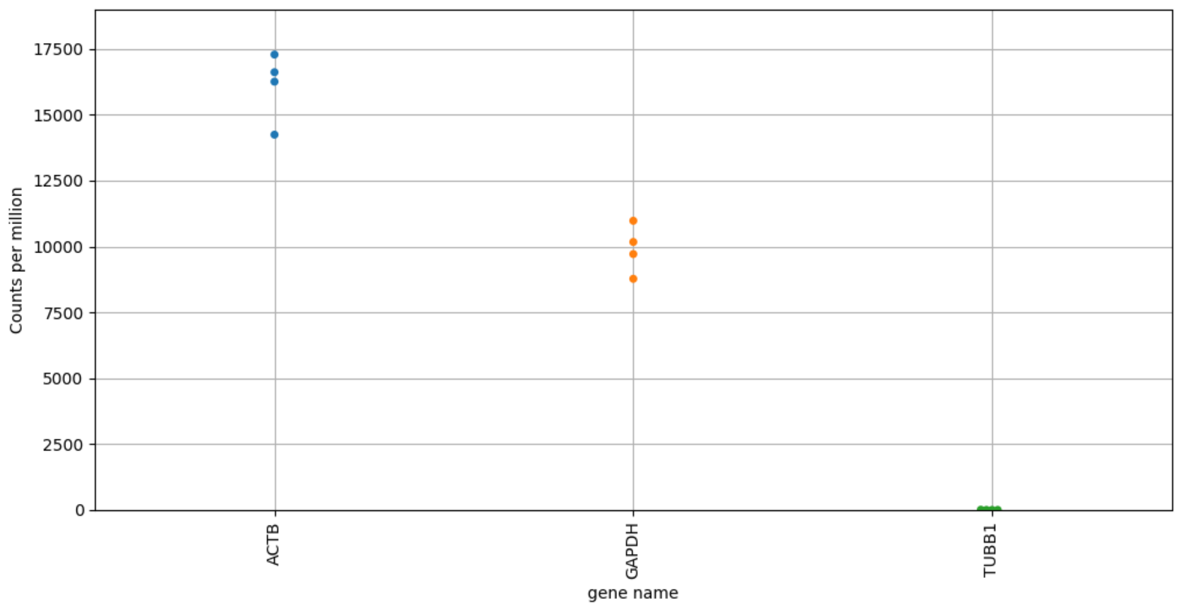
(continued from previous page)

```
        samplesheet='example_sheet_tsv',
        counts_table='example_table_tsv')

print('Plot coverage')
ax = ds.plot.plot_coverage(color='blue', lw=3)
ax = ds.plot.plot_coverage(
        features='other', color='red', linewidth=1,
        ax=ax)

print('Plot spike-in distributions')
ax = ds.plot.plot_distributions(
        kind='swarm',
        features='spikeins',
        orientation='horizontal',
        sort='descending')

print('Plot normalized distributions of housekeeping genes')
ds.counts.normalize('counts_per_million', inplace=True)
ax = ds.plot.plot_distributions(
        kind='swarm',
        features=['ACTB', 'TUBB1', 'GAPDH'],
        orientation='vertical',
        bottom='pseudocount',
        grid=True,
        sort='descending')

plt.show()
```

You should get figures similar to the following ones:




ERCC-00096
ERCC-00002
ERCC-00130
ERCC-00074
ERCC-00113
ERCC-00108
ERCC-00046
ERCC-00004
ERCC-00171
ERCC-00009
ERCC-00136
ERCC-00060
ERCC-00003
ERCC-00042
ERCC-00076
ERCC-00157
ERCC-00054
ERCC-00043
ERCC-00044
ERCC-00165
ERCC-00048
ERCC-00051
ERCC-00053
ERCC-00058
ERCC-00057
ERCC-00040
ERCC-00059
ERCC-00061
ERCC-00062
ERCC-00067
ERCC-00069
ERCC-00041
ERCC-00035
ERCC-00039
ERCC-00073
ERCC-00168
ERCC-00007
ERCC-00012
ERCC-00013
ERCC-00014
ERCC-00016
ERCC-00017
ERCC-00018
ERCC-00019
ERCC-00022
ERCC-00023
ERCC-00024
ERCC-00025
ERCC-00028
ERCC-00031
ERCC-00033
ERCC-00034
ERCC-00071
ERCC-00075
ERCC-00164
ERCC-00137
ERCC-00117
ERCC-00120
ERCC-00123
ERCC-00126
ERCC-00128
ERCC-00158
ERCC-00131
ERCC-00134
ERCC-00138
ERCC-00156
ERCC-00142
ERCC-00143
ERCC-00144
ERCC-00145
ERCC-00147
ERCC-00148
ERCC-00150
ERCC-00154
ERCC-00116
ERCC-00160
ERCC-00112
ERCC-00111
ERCC-00077
ERCC-00078
ERCC-00170
ERCC-00081
ERCC-00083
ERCC-00084
ERCC-00085
ERCC-00086
ERCC-00092
ERCC-00095
ERCC-00163
ERCC-00097
ERCC-00098
ERCC-00099
ERCC-00104
ERCC-00162
ERCC-00109
ERCC-00079
gene name
0
100
200
300
400
500
Number of reads



4.1.6 Example: Split and compare

Singlet allows you to split a dataset based on metadata in a single line. Moreover, it is easy to perform statistical comparisons between two datasets, comparing feature expression and/or phenotypes with any statistical test you like.

```
from singlet.dataset import Dataset
ds = Dataset(
        samplesheet='example_sheet_tsv',
        counts_table='example_table_tsv')

ds.counts.normalize('counts_per_million', inplace=True)

# Split dataset based on metadata
dataset_dict = ds.split('experiment')

# Statistical comparison of features between datasets
dataset_dict['test_pipeline'].compare(
       dataset_dict['exp1'],
       method='mann-whitney')
```

Note: Mann-Whitney's U test and two sample Kolmogorov-Smirnov's test are built-ins, but you can just set *method* to any function you want that calculates the P-values.

4.1.7 Example: t-SNE

t-SNE *[tsne]* is a commonly used algorithm to reduce dimensionality in single cell data.

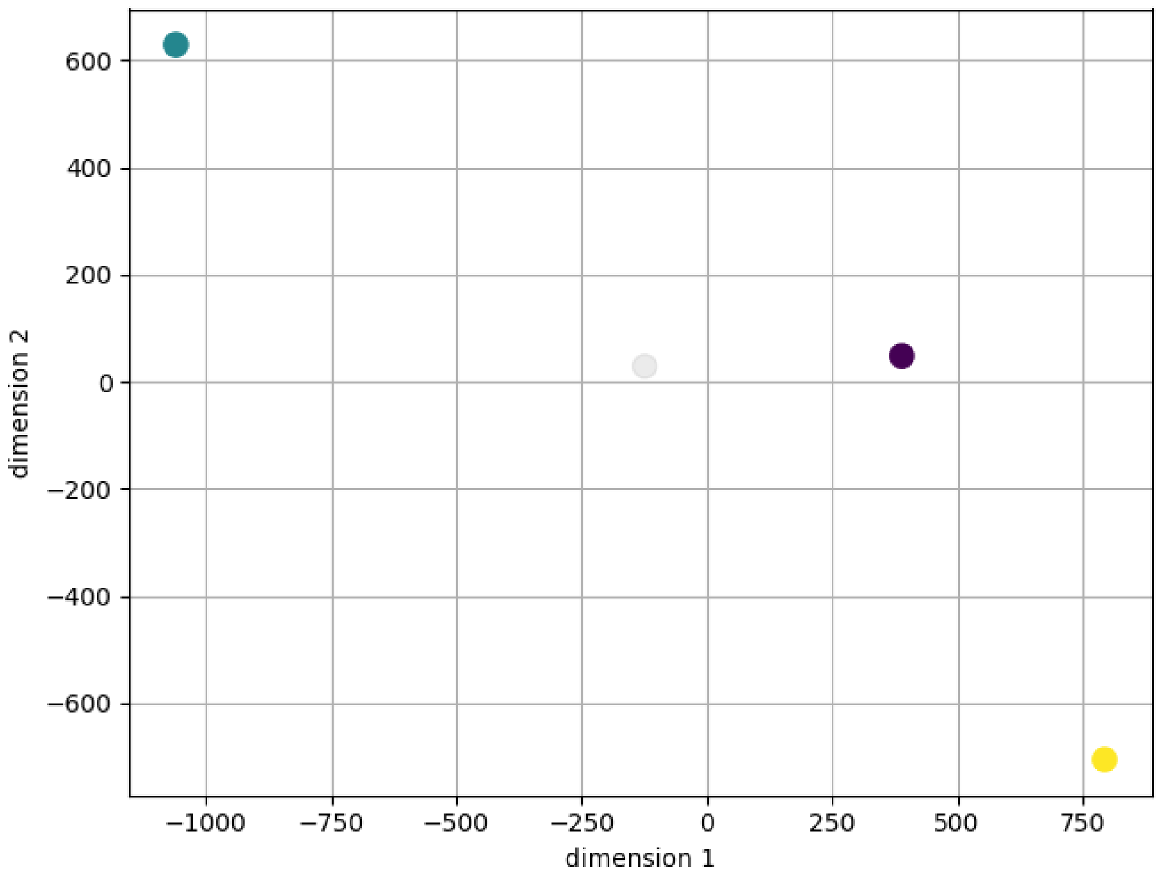
```
from singlet.dataset import Dataset
ds = Dataset(
        samplesheet='example_sheet_tsv',
```

(continues on next page)

(continued from previous page)

```
        counts_table='example_table_tsv')

ds.counts.normalize('counts_per_million', inplace=True)
ds.counts = ds.counts.iloc[:200]

print('Calculate t-SNE')
vs = ds.dimensionality.tsne(
        n_dims=2,
        transform='log10',
        theta=0.5,
        perplexity=0.8)

print('Plot t-SNE')
ax = ds.plot.scatter_reduced_samples(
        vs,
        color_by='quantitative_phenotype_1_[A.U.]')

plt.show()
```

You should get figures similar to the following ones:



4.2 Configuration

Singlet is designed to work on separate projects at once. To keep projects tidy and independent, there are two layers of configuration.

4.2.1 The `SINGLET_CONFIG_FILENAME` environment variable

If you set the environment variable `SINGLET_CONFIG_FILENAME` to point at a YAML file, singlet will use it to configure your current session. To use separate sessions in parallel, just prepend your scripts with:

```
import os
os.environ['SINGLET_CONFIG_FILENAME'] = '<full path to config file>'
```

and each will work totally independently.

4.2.2 The configuration file

Singlet loads a configuration file in YAML format when you import the `singlet` module. If you have not specified the location of this file with the `SINGLET_CONFIG_FILENAME` environment variable, it defaults to:

```
<your home folder>/.singlet/config.yml
```

so the software will look there. An example configuration file is online. If you are not familiar with YAML syntax, it is a bit like Python dictionaries without brackets. . . or like JSON.

Before going into the specifics, here's a schematic example of the configuration file:

```
io:
  samplesheets:
    ss1:
      path: xxx.csv
      index: samplename

  featuresheets:
    fs1:
      path: yyy.csv
      index: EnsemblGeneID

  count_tables:
    ct1:
      path: zzz.csv
      normalized: no
      spikeins:
        - ERCC-00002
        - ERCC-00003
      other:
        - __alignment_not_unique
        - __not_aligned

  datasets:
    ds1:
      path: xxx.loom
      format: loom
      axis_samples: columns
```

(continues on next page)

(continued from previous page)

```
      index_samples: Cell
      index_features: Gene
```

Now for the full specification, the root key value pairs are:

- `io`: for input/output specifications. At the moment this key is the only master key and is required.

There are no root lists.

io

The `io` section has the following key value pairs:

- `samplesheets`: samplesheet files or Google Documents (for sample metadata).
- `featuresheets`: featuresheet files (for feature/gene annotations or metadata).
- `count_tables`: count table files.
- `datasets`: integrated datasets (a single file contains all three properties above, e.g. loom files).

samplesheets

The `samplesheets` section contains an arbitrary number of key value pairs and no lists. Each entry describes a samplesheet a

- the key determines the id of the samplesheet: this id is used in the contstructor of `Dataset`.
- the value is a series of key value pairs, no lists.

Singlet can source samplesheets either from a local file or from an online Google Sheet. If you want to use a local file, use the fol

- `path`: a filename on disk containing the samplesheet, usually in CSV/TSV format.
- `format`: a file format of the samplesheet (optional). If missing, it is inferred from the `path` filename.

If you prefer to source an online Google Sheet, use the following key value pairs:

- `url`: the URL of the spreadsheet, e.g. 'https://docs.google.com/spreadsheets/d/15OKOC48WZYFUQvYl9E7qEsR6AjqE4_BW7qcCsjJAD6w' for the example sheet.
- `client_id_filename`: a local filename (initially empty) where your login information for OAUTH2 is stored. This is a JSON file so this variable typically ends with `.json`
- `client_secret_filename`: a local filename (initially empry) where your secret information for OAUTH2 is stored. This is a JSON file so this variable typically ends with `.json`
- `sheet`: the name of the sheet with the data within the spreadsheet.

Whichever way you are using to source the data, the following key value pairs are available:

- `description`: a description of the sample sheet (optional).
- `cells`: one of `rows` or `columns`. If each row in the samplesheet is a sample, use `rows`, else use `columns`. Notice that singlet samplesheets have samples as **rows**.
- `index`: the name of the column/row of the samplesheet containing the sample names. This defaults to `name` (optional).

count_tables

The `count_tables` section contains an arbitrary number of key value pairs and no lists. Each entry describes a counts table a

- the key determines the id of the counts table: this id is used in the contstructor of `Dataset`.
- the value is a series of key value pairs, no lists.

The following key value pairs are available:

- `description`: a description of the counts table (optional).
- `path`: a filename on disk containing the counts table, usually in CSV/TSV format.
- `format`: a file format of the counts table (optional). If missing, it is inferred from the `path` filename.
- `cells`: one of `rows` or `columns`. If each row in the counts table is a sample, use `rows`, else use `columns`.
- `normalized`: either `yes` or `no`. If data is not normalized, you can normalize it with singlet by using the `CountsTable.normalize` method.
- `sparse`: either `yes` or `no` (default). If `yes`, the count table will be loaded by default as `CountsTableSparse`, else as `CountsTable` (dense).
- `spikeins`: a YAML list of features that appear in the counts table and represent spike-in controls as opposed to real features. Spikeins can be excluded from the counts table using `CountsTable.exclude_features`.
- `other`: a YAML list of features that are neither biological features nor spike-in controls. This list typically includes ambiguous alignments, multiple-aligned reads, reads outside features, etc. Other features can be excluded from the counts table using `CountsTable.exclude_features`.

The first column/row of the counts table must be the list of samples.

featuresheets

The `featuresheets` section contains an arbitrary number of key value pairs and no lists. Each entry describes a featureshee

- the key is the id of the featuresheet: this id is used in the constructor of `Dataset`.
- the value is a series of key value pairs, no lists.

The following key value pairs are available:

- `description`: a description of the featuresheet (optional).
- `path`: a filename on disk containing the featuresheet, usually in CSV/TSV format.
- `format`: a file format of the featuresheet (optional). If missing, it is inferred from the `path` filename.
- `features`: one of `rows` or `columns`. If each feature in the featuresheet is a feature, use `rows`, otherwise use `columns`.
- `index`: the name of the column/row of the featuresheet containing the feature names. This defaults to `name` (optional).

datasets

The `datasets` section contains an arbitrary number of key value pairs and no lists. Each entry describes an integrated datase

- the key is the id of the dataset: this id is used in the constructor of `Dataset`.
- the value is a series of key value pairs, no lists.

The following key value pairs are available:

- `description`: a description of the dataset (optional).
- `path`: a filename on disk containing the integrated dataset, e.g. in LOOM format.
- `format`: a file format of the dataset (optional). If missing, it is inferred from the `path` filename.
- `axis_samples`: one of `rows` or `columns`. If every sample is a column in the count matrix, use `columns`, else use `rows`.
- `index_samples`: the name of the column/row of the dataset containing the sample names.
- `index_features`: the name of the column/row of the dataset containing the feature names.

4.3 API

Singlet analysis is centered around the *Dataset* class, which describes a set of samples (usually single cells). Each *Dataset* has three main properties:

- a *CountsTable* with the counts of genomic features, typically transcripts
- a *SampleSheet* with the sample metdata and phenotypic information.
- a *FeatureSheet* with the feature metdata, for instance alternative names and Gene Ontology terms.

At least one of the three properties must be present. In fact, you are perfectly free to set only the feature counts or even, although may be not so useful, only the sample metadata. Moreover, a *Dataset* has a number of "action properties" that perform operations on the data:

- *Dataset.correlations*: correlate feature expressions and phenotypes
- *Dataset.feature_selection*: select features based on expression patterns
- *Dataset.dimensionality*: reduce dimensionality of the data including phenotypes
- *Dataset.cluster*: cluster samples, features, and phenotypes
- *Dataset.fit*: fit (regress on) feature expression and metadata
- *Dataset.plot*: plot the results of various analyses

Supporting modules are useful for particular purposes or internal use only:

- *config*
- *utils*
- *io*

4.3.1 API reference

singlet.dataset

class `singlet.dataset.`**`Dataset`**(*counts_table=None*, *samplesheet=None*, *featuresheet=None*, *dataset=None*, *plugins=None*)

Bases: `object`

Collection of cells, with feature counts and metadata

`average`(*axis*, *column*)

Average samples or features based on metadata

Parameters

- **axis** (*`string`*) – Must be 'samples' or 'features'.
- **column** (*`string`*) – Must be a column of the samplesheet (for axis='samples') or of the featuresheet (for axis='features'). Samples or features with a common value in this column are averaged over.

Returns A Dataset with the averaged counts.

Note: if you average over samples, you get an empty samplesheet. Simlarly, if you average over features, you get an empty featuresheet.

`bootstrap`(*groupby=None*)

Resample with replacement, aka bootstrap dataset

Parameters

- **groupby** (*`str or list of str or None`*) – If None, bootstrap random
- **disregarding sample metadata. If a string or a list of** (*`samples`*) –
- **boostrap over groups of samples with consistent** (*`strings,`*) –
- **for that/those columns.** (*`entries`*) –

Returns A Dataset with the resampled samples.

`compare`(*other*, *features='mapped'*, *phenotypes=()*, *method='kolmogorov-smirnov'*)

Statistically compare with another Dataset.

Parameters

- **other** (*`Dataset`*) – The Dataset to compare with.
- **features** (*`list, string, or None`*) – Features to compare. The string 'total' means all features including spikeins and other, 'mapped' means all features excluding spikeins and other, 'spikeins' means only spikeins, and 'other' means only 'other' features. If empty list or None, do not compare features (useful for phenotypic comparison).
- **phenotypes** (*`list of strings`*) – Phenotypes to compare.
- **method** (*`string or function`*) – Statistical test to use for the comparison. If a string it must be one of 'kolmogorov-smirnov' or 'mann-whitney'. If a function, it must accept two arrays as arguments (one for each dataset, running over the samples) and return a P-value for the comparison.

Returns

A pandas.DataFrame containing the P-values of the comparisons for all features and phenotypes.

copy()
Copy of the Dataset

counts
Matrix of gene expression counts.

Rows are features, columns are samples.

Notice: If you reset this matrix with features that are not in the featuresheet or samples that are not in the samplesheet, those tables will be reset to empty.

featuremetadatanames
pandas.Index of feature metadata column names

featurenames
pandas.Index of feature names

featuresheet
Matrix of feature metadata.

Rows are features, columns are metadata (e.g. Gene Ontologies).

n_features
Number of features

n_samples
Number of samples

query_features_by_counts(*expression*, *inplace=False*, *local_dict=None*)
Select features based on their expression.

Parameters

- **expression** (*string*) – An expression compatible with pandas.DataFrame.query.
- **inplace** (*bool*) – Whether to change the Dataset in place or return a new one.
- **local_dict** (*dict*) – A dictionary of local variables, useful if you are using @var assignments in your expression. By far the most common usage of this argument is to set local_dict=locals().

Returns If *inplace* is True, None. Else, a Dataset.

query_features_by_metadata(*expression*, *inplace=False*, *local_dict=None*)
Select features based on metadata.

Parameters

- **expression** (*string*) – An expression compatible with pandas.DataFrame.query.
- **inplace** (*bool*) – Whether to change the Dataset in place or return a new one.
- **local_dict** (*dict*) – A dictionary of local variables, useful if you are using @var assignments in your expression. By far the most common usage of this argument is to set local_dict=locals().

Returns If *inplace* is True, None. Else, a Dataset.

query_features_by_name(*featurenames*, *inplace=False*, *ignore_missing=False*)
Select features by name.

Parameters

- **featurenames** – names of the features to keep.
- **inplace** (*bool*) – Whether to change the Dataset in place or return a new one.

- **ignore_missing** (*bool*) – Whether to silently skip missing features.

query_samples_by_counts(*expression*, *inplace=False*, *local_dict=None*)
Select samples based on gene expression.

Parameters

- **expression** (*string*) – An expression compatible with pandas.DataFrame.query.
- **inplace** (*bool*) – Whether to change the Dataset in place or return a new one.
- **local_dict** (*dict*) – A dictionary of local variables, useful if you are using @var assignments in your expression. By far the most common usage of this argument is to set local_dict=locals().

Returns If *inplace* is True, None. Else, a Dataset.

query_samples_by_metadata(*expression*, *inplace=False*, *local_dict=None*)
Select samples based on metadata.

Parameters

- **expression** (*string*) – An expression compatible with pandas.DataFrame.query.
- **inplace** (*bool*) – Whether to change the Dataset in place or return a new one.
- **local_dict** (*dict*) – A dictionary of local variables, useful if you are using @var assignments in your expression. By far the most common usage of this argument is to set local_dict=locals().

Returns If *inplace* is True, None. Else, a Dataset.

query_samples_by_name(*samplenames*, *inplace=False*, *ignore_missing=False*)
Select samples by name.

Parameters

- **samplenames** – names of the samples to keep.
- **inplace** (*bool*) – Whether to change the Dataset in place or return a new one.
- **ignore_missing** (*bool*) – Whether to silently skip missing samples.

reindex(*axis*, *column*, *drop=False*, *inplace=False*)
Reindex samples or features from a metadata column

Parameters

- **axis** (*string*) – Must be 'samples' or 'features'.
- **column** (*string*) – Must be a column of the samplesheet (for axis='samples') or of the featuresheet (for axis='features') with unique names of samples or features.
- **drop** (*bool*) – Whether to drop the column from the metadata table.
- **inplace** (*bool*) – Whether to change the Dataset in place or return a new one.

rename(*axis*, *column*, *inplace=False*)
Rename samples or features

Parameters

- **axis** (*string*) – Must be 'samples' or 'features'.
- **column** (*string*) – Must be a column of the samplesheet (for axis='samples') or of the featuresheet (for axis='features') with unique names of samples or features.
- **inplace** (*bool*) – Whether to change the Dataset in place or return a new one.

DEPRECATED: use *reindex* instead.

`samplemetadatanames`
pandas.Index of sample metadata column names

`samplenames`
pandas.Index of sample names

`samplesheet`
Matrix of sample metadata.

Rows are samples, columns are metadata (e.g. phenotypes).

`split` (*phenotypes*, *copy=True*)
Split Dataset based on one or more categorical phenotypes

Parameters **`phenotypes`** (*`string or list of strings`*) – one or more phenotypes to use for the split. Unique values of combinations of these determine the split Datasets.

Returns

the keys are either unique values of the phenotype chosen or, if more than one, tuples of unique combinations.

Return type dict of Datasets

`to_dataset_file` (*filename*, *fmt=None*, ***kwargs*)
Store dataset into an integrated dataset file

Parameters

- **`filename`** (*`str`*) – path of the file to write to.
- **`fmt`** (*`str or None`*) – file format. If None, infer from the file
- **`extension.`** –
- **`**kwargs`** (*`keyword arguments`*) – depend on the format.

The additional keyword argument for the supported formats are: - loom:

- axis_samples: *rows* or *columns* (default)

singlet.counts_table.counts_table

class `singlet.counts_table.counts_table.`**`CountsTable`**(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*)

Bases: `pandas.core.frame.DataFrame`

Table of gene expression counts

- Rows are features, e.g. genes.
- Columns are samples.

`bin` (*bins=5*, *result='index'*, *inplace=False*)
Bin feature counts.

Parameters

- **`bins`** (*`int, array, or list of arrays`*) – If an int, number equal-width bins between pseudocounts and the max of the counts matrix. If an array of indices of the same length as the number of features, use a different number of equal-width bins for each feature. If an array of any other length, use these bin edges (including rightmost edge) for all features. If a list of arrays, it has to be as long as the number of features, and every array in the list determines the bin edges (including rightmost edge) for that feature, in order.

- **result** (*string*) – Has to be one of 'index' (default), 'left', 'center', 'right'. 'index' assign to the feature the index (starting at 0) of that bin, 'left' assign the left bin edge, 'center' the bin center, 'right' the right edge. If result is 'index', out-of-bounds values will be assigned the value -1, which means Not A Number in ths context.
- **inplace** (*bool*) – Whether to perform the operation in place.

Returns If inplace is False, a CountsTable with the binned counts.

center(*axis='samples'*, *inplace=False*)
Center the counts table (subtract mean).

Parameters

- **axis** (*string*) – The axis to average over, has to be 'samples' or 'features'.
- **inplace** (*bool*) – Whether to do the operation in place or return a new CountsTable

Returns If inplace is False, a transformed CountsTable.

dataset = None

exclude_features(*spikeins=True*, *other=True*, *inplace=False*, *errors='raise'*)
Get a slice that excludes secondary features.

Parameters

- **spikeins** (*bool*) – Whether to exclude spike-ins
- **other** (*bool*) – Whether to exclude other features, e.g. unmapped reads
- **inplace** (*bool*) – Whether to drop those features in place.
- **errors** (*string*) – Whether to raise an exception if the features to be excluded are already not present. Must be 'ignore' or 'raise'.

Returns a slice of self without those features.

Return type *CountsTable*

classmethod from_datasetname(*datasetname*)
Instantiate a CountsTable from its name in the config file.

Parameters **datasetname** (*string*) – name of the dataset in the config file.

Returns the counts table.

Return type *CountsTable*

classmethod from_tablename(*tablename*)
Instantiate a CountsTable from its name in the config file.

Parameters **tablename** (*string*) – name of the counts table in the config file.

Returns the counts table.

Return type *CountsTable*

get_other_features()
Get other features

Returns a slice of self with only other features (e.g. unmapped).

Return type *CountsTable*

get_spikeins()
Get spike-in features

Returns a slice of self with only spike-ins.

Return type *CountsTable*

get_statistics(*metrics=('mean', 'cv')*)
Get statistics of the counts.

Parameters **metrics** (*sequence of strings*) – any of 'mean', 'var', 'std', 'cv', 'fano', 'min', 'max'.

Returns pandas.DataFrame with features as rows and metrics as columns.

log(*base=10*, *inplace=False*)
Take the pseudocounted log of the counts.

Parameters

- **base** (*float*) – Base of the log transform
- **inplace** (*bool*) – Whether to do the operation in place or return a new CountsTable

Returns If inplace is False, a transformed CountsTable.

normalize(*method='counts_per_million'*, *include_spikeins=False*, *inplace=False*, ***kwargs*)
Normalize counts and return new CountsTable.

Parameters

- **method** (*string or function*) – The method to use for normalization.
- **of 'counts_per_million', 'counts_per_thousand_spikeins',** (*One*) –
- **If this argument is a function, its** (*'counts_per_thousand_features'.*) –
- **depends on the inplace argument. If inplace=False, it** (*signature*) –
- **take the CountsTable as input and return the normalized one as** (*must*) –
- **If inplace=True, it must take the CountsTableXR as input** (*output.*) –
- **modify it in place. Notice that if inplace=True and you do** (*and*) –
- **operations you might lose the _metadata properties. You** (*non-inplace*) –
- **end your function by self[** (*can*) –] = <normalized counts>.
- **include_spikeins** (*bool*) – Whether to include spike-ins in the
- **and result.** (*normalization*) –
- **inplace** (*bool*) – Whether to modify the CountsTableXR in place or
- **a new one.** (*return*) –

Returns If *inplace* is False, a new, normalized CountsTable.

pseudocount = 0.1

standard_scale(*axis='samples'*, *inplace=False*, *add_to_den=0*)
Subtract minimum and divide by (maximum - minimum).

Parameters

- **axis** (*string*) – The axis to average over, has to be 'samples' or 'features'.
- **inplace** (*bool*) – Whether to do the operation in place or return a new CountsTable
- **add_to_den** (*float*) – Whether to add a (small) value to the denominator to avoid NaNs. 1e-5 or so should be fine.

Returns If inplace is False, a transformed CountsTable.

unlog(*base=10*, *inplace=False*)
Reverse the pseudocounted log of the counts.

Parameters

- **base** (*float*) – Base of the log transform
- **inplace** (*bool*) – Whether to do the operation in place or return a new CountsTable

Returns If inplace is False, a transformed CountsTable.

z_score(*axis='samples'*, *inplace=False*, *add_to_den=0*)
Calculate the z scores of the counts table.

In other words, subtract the mean and divide by the standard deviation.

Parameters

- **axis** (*string*) – The axis to average over, has to be 'samples' or 'features'.
- **inplace** (*bool*) – Whether to do the operation in place or return a new CountsTable
- **add_to_den** (*float*) – Whether to add a (small) value to the denominator to avoid NaNs. 1e-5 or so should be fine.

Returns If inplace is False, a transformed CountsTable.

singlet.samplesheet

class singlet.samplesheet.**SampleSheet**(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*)
Bases: `pandas.core.frame.DataFrame`

classmethod from_datasetname(*datasetname*)

classmethod from_sheetname(*sheetname*)

singlet.featuresheet

class singlet.featuresheet.**FeatureSheet**(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*)
Bases: `pandas.core.frame.DataFrame`

classmethod from_datasetname(*datasetname*)

classmethod from_sheetname(*sheetname*)

singlet.dataset.cluster

class `singlet.dataset.cluster.`**`Cluster`**(*dataset*)

Bases: `singlet.dataset.plugins.Plugin`

Cluster samples, features, and phenotypes

`affinity_propagation`(*axis*, *phenotypes=()*, *metric='correlation'*, *log_features=False*)

Affinity/label/message propagation.

Parameters

- **`axis`** (*`string`*) – It must be 'samples' or 'features'. The Dataset.counts matrix is used and either samples or features are clustered.
- **`phenotypes`** (*`iterable of strings`*) – Phenotypes to add to the features for joint clustering.
- **`metric`** (*`string or matrix`*) – Metric to calculate the distance matrix. If it is a matrix already, use it as distance (squared). Else it should be a string accepted by scipy.spatial.distance.pdist.
- **`log_features`** (*`bool`*) – Whether to add pseudocounts and take a log of the feature counts before calculating distances.

Returns dict with the linkage, distance matrix, and ordering.

`dbscan`(*axis*, *phenotypes=()*, ***kwargs*)

Density-Based Spatial Clustering of Applications with Noise.

Parameters

- **`axis`** (*`string`*) – It must be 'samples' or 'features'. The Dataset.counts matrix is used and either samples or features are clustered.
- **`phenotypes`** (*`iterable of strings`*) – Phenotypes to add to the features for joint clustering.
- **`log_features`** (*`bool`*) – Whether to add pseudocounts and take a log of the feature counts before calculating distances.
- **`**kwargs`** – arguments passed to sklearn.cluster.DBSCAN.

Returns pd.Series with the labels of the clusters.

`hierarchical`(*axis*, *phenotypes=()*, *metric='correlation'*, *method='average'*, *log_features=False*, *optimal_ordering=False*)

Hierarchical clustering.

Parameters

- **`axis`** (*`string`*) – It must be 'samples' or 'features'. The Dataset.counts matrix is used and either samples or features are clustered.
- **`phenotypes`** (*`iterable of strings`*) – Phenotypes to add to the features for joint clustering.
- **`metric`** (*`string or matrix`*) – Metric to calculate the distance matrix. If it is a matrix already, use it as distance (squared). Else it should be a string accepted by scipy.spatial.distance.pdist.
- **`method`** (*`string`*) – Clustering method. Must be a string accepted by scipy.cluster.hierarchy.linkage.

- **`log_features`** (*`bool`*) – Whether to add pseudocounts and take a log of the feature counts before calculating distances.
- **`optimal_ordering`** (*`bool`*) – Whether to resort the linkage so that nearest neighbours have shortest distance. This may take longer than the clustering itself.

Returns dict with the linkage, distance matrix, and ordering.

`kmeans`(*n_clusters*, *axis*, *phenotypes=()*, *random_state=0*)
K-Means clustering.

Parameters

- **`n_clusters`** (*`int`*) – The number of clusters you want.
- **`axis`** (*`string`*) – It must be 'samples' or 'features'. The Dataset.counts matrix is used and either samples or features are clustered.
- **`phenotypes`** (*`iterable of strings`*) – Phenotypes to add to the features for joint clustering.
- **`log_features`** (*`bool`*) – Whether to add pseudocounts and take a log of the feature counts before calculating distances.
- **`random_state`** (*`int`*) – Set to the same int for deterministic results.

Returns pd.Series with the labels of the clusters.

singlet.dataset.correlations

class `singlet.dataset.correlations.`**`Correlation`**(*dataset*)
Bases: `singlet.dataset.plugins.Plugin`

Correlate gene expression and phenotype in single cells

`correlate_features_features`(*features='all'*, *features2=None*, *method='spearman'*)
Correlate feature expression with one or more phenotypes.

Parameters

- **`features`** (*`list or string`*) – list of features to correlate. Use a string for a single feature. The special string 'all' (default) uses all features.
- **`features2`** (*`list or string`*) – list of features to correlate with. Use a string for a single feature. The special string 'all' uses all features. None (default) takes the same list as features, returning a square matrix.
- **`method`** (*`string`*) – type of correlation. Must be one of 'pearson' or 'spearman'.

Returns

pandas.DataFrame with the correlation coefficients. If either features or features2 is a single string, the function returns a pandas.Series. If both are a string, it returns a single correlation coefficient.

`correlate_features_phenotypes`(*phenotypes*, *features='all'*, *method='spearman'*, *fillna=None*)
Correlate feature expression with one or more phenotypes.

Parameters

- **`phenotypes`** (*`list of string`*) – list of phenotypes, i.e. columns of the samplesheet. Use a string for a single phenotype.

- **features** (*list or string*) – list of features to correlate. Use a string for a single feature. The special string 'all' (default) uses all features.
- **method** (*string*) – type of correlation. Must be one of 'pearson' or 'spearman'.
- **fillna** (*dict, int, or None*) – a dictionary with phenotypes as keys and numbers to fill for NaNs as values. None will do nothing.

Returns

pandas.DataFrame with the correlation coefficients. If either phenotypes or features is a single string, the function returns a pandas.Series. If both are a string, it returns a single correlation coefficient.

correlate_phenotypes_phenotypes(*phenotypes*, *phenotypes2=None*, *method='spearman'*, *fillna=None*, *fillna2=None*)

Correlate feature expression with one or more phenotypes.

Parameters

- **phenotypes** (*list of string*) – list of phenotypes, i.e. columns of the samplesheet. Use a string for a single phenotype.
- **phenotypes2** (*list of string*) – list of phenotypes, i.e. columns of the samplesheet. Use a string for a single phenotype. None (default) uses the same as phenotypes.
- **method** (*string*) – type of correlation. Must be one of 'pearson' or 'spearman'.
- **fillna** (*dict, int, or None*) – a dictionary with phenotypes as keys and numbers to fill for NaNs as values. None will do nothing, potentially yielding NaN as correlation coefficients.
- **fillna2** (*dict, int, or None*) – as fillna, but for phenotypes2.

Returns

pandas.DataFrame with the correlation coefficients. If either phenotypes or features is a single string, the function returns a pandas.Series. If both are a string, it returns a single correlation coefficient.

correlate_samples(*samples='all'*, *samples2=None*, *phenotypes=None*, *method='spearman'*)

Correlate feature expression with one or more phenotypes.

Parameters

- **samples** (*list or string*) – list of samples to correlate. Use a string for a single sample. The special string 'all' (default) uses all samples.
- **samples2** (*list or string*) – list of samples to correlate with. Use a string for a single sample. The special string 'all' uses all samples. None (default) takes the same list as samples, returning a square matrix.
- **method** (*string*) – type of correlation. Must be one of 'pearson' or 'spearman'.
- **phenotypes** (*list*) – phenotypes to include as additional features in the correlation calculation. None (default) means only feature counts are used.

Returns

pandas.DataFrame with the correlation coefficients. If either samples or samples2 is a single string, the function returns a pandas.Series. If both are a string, it returns a single correlation coefficient.

singlet.dataset.dimensionality

class `singlet.dataset.dimensionality.`**`DimensionalityReduction`**(*dataset*)

Bases: `singlet.dataset.plugins.Plugin`

Reduce dimensionality of gene expression and phenotype

`pca`(*n_dims=2*, *transform='log10'*, *robust=True*, *random_state=None*)

Principal component analysis

Parameters

- **`n_dims`** (*`int`*) – Number of dimensions (2+).
- **`transform`** (*`string or None`*) – Whether to preprocess the data.
- **`robust`** (*`bool`*) – Whether to use Principal Component Pursuit to exclude outliers.

Returns

dict of the left eigenvectors (vs), right eigenvectors (us) of the singular value decomposition, eigenvalues (lambdas), the transform, and the whiten function (for plotting).

`tsne`(*n_dims=2*, *perplexity=30*, *theta=0.5*, *rand_seed=0*, ***kwargs*)

t-SNE algorithm.

Parameters

- **`n_dims`** (*`int`*) – Number of dimensions to use.
- **`perplexity`** (*`float`*) – Perplexity of the algorithm.
- **`theta`** (*`float`*) – A number between 0 and 1. Higher is faster but less accurate (via the Barnes-Hut approximation).
- **`rand_seed`** (*`int`*) – Random seed. -1 randomizes each run.
- **`**kwargs`** – Named arguments passed to the t-SNE algorithm.

Returns:

`umap`(*n_dims=2*, *rand_seed=0*, ***kwargs*)

Uniform Manifold Approximation and Projection.

Parameters

- **`n_dims`** (*`int`*) – Number of dimensions to use.
- **`rand_seed`** (*`int`*) – Random seed. -1 randomizes each run.
- **`**kwargs`** – Named arguments passed to umap.UMAP.

Returns:

singlet.dataset.feature_selection

class `singlet.dataset.feature_selection.`**`FeatureSelection`**(*dataset*)

Bases: `singlet.dataset.plugins.Plugin`

Plot gene expression and phenotype in single cells

`expressed`(*n_samples*, *exp_min*, *inplace=False*)

Select features that are expressed in at least some samples.

Parameters

- **n_samples** (*int*) – Minimum number of samples the features should be expressed in.
- **exp_min** (*float*) – Minimum level of expression of the features.
- **inplace** (*bool*) – Whether to change the feature list in place.

Returns pd.Index of selected features if not inplace, else None.

gate_features_from_statistics(*features='mapped'*, *x='mean'*, *y='cv'*, ***kwargs*)

Select features for downstream analysis with a gate.

Usage: Click with the left mouse button to set the vertices of a polygon. Double left-click closes the shape. Right click resets the plot.

Parameters

- **features** (*list or string*) – List of features to plot. The string 'mapped' means everything excluding spikeins and other, 'all' means everything including spikeins and other.
- **x** (*string*) – Statistics to plot on the x axis.
- **y** (*string*) – Statistics to plot on the y axis.
- ****kwargs** – named arguments passed to the plot function.

Returns pd.Index of features within the gate.

overdispersed_strata(*bins=10*, *n_features_per_stratum=50*, *inplace=False*)

Select overdispersed features in strata of increasing expression.

Parameters

- **bins** (*int or list*) – Bin edges determining the strata. If this is a number, split the expression in this many equally spaced bins between minimal and maximal expression.
- **n_features_per_stratum** (*int*) – Number of features per stratum to select.

Returns pd.Index of selected features if not inplace, else None.

Notice that the number of selected features may be smaller than expected if some strata have no dispersion (e.g. only dropouts). Because of this, it is recommended you restrict the counts to expressed features before using this function.

sam(*k=None*, *distance='correlation'*, **args*, ***kwargs*)

Calculate feature weights via self-assembling manifolds

Parameters

- **k** (*int or None*) – The number of nearest neighbors for each sample
- **distance** (*str*) – The distance matrix
- ****kwargs** (**args,*) – Arguments to SAM.run

Returns SAM instance containing SAM.output_vars['gene_weights']

See also: https://github.com/atarashansky/self-assembling-manifold

unique(*inplace=False*)

Select features with unique ids

Parameters **inplace** (*bool*) – Whether to change the feature list in place.

Returns pd.Index of selected features if not inplace, else None.

singlet.dataset.fit

class `singlet.dataset.fit.`**`Fit`**(*dataset*)

Bases: `singlet.dataset.plugins.Plugin`

Fit gene expression and phenotype in single cells

`fit_single`(*xs*, *ys*, *model*, *method='least-squares'*, *handle_nans='ignore'*, ***kwargs*)

Fit feature expression or phenotypes against other.

Parameters

- **xs** (*list or string*) – Features and/or phenotypes to use as abscissa (independent variable). The string 'total' means all features including spikeins and other, 'mapped' means all features excluding spikeins and other, 'spikeins' means only spikeins, and 'other' means only 'other' features.
- **ys** (*list or string*) – Features and/or phenotypes to use as ordinate (dependent variable). The string 'total' means all features including spikeins and other, 'mapped' means all features excluding spikeins and other, 'spikeins' means only spikeins, and 'other' means only 'other' features.
- **model** (*string or function*) – The model to use for fitting. If a string, it must be one of 'linear', 'threshold-linear', 'logistic'. If a function, it must accept an array as first argument (the x) and the parameters as additional arguments (like scipy.optimize.curve_fit).
- **method** (*string or function*) – The minimization algorithm. For now, only 'least-squares' is accepted. In this case, the goodness of fit is the sum of the squared residues.
- **handle_nans** (*string*) – How to deal with Not a Numbers, typically in the phenotypes. Must be either 'ignore' (default), in which case only the non-NaN samples will be used for fitting, or 'raise', in which case NaNs will stop the fit.
- ****kwargs** – Passed to the fit function. For nonlinear least-squares, this is scipy.optimize.curve_fit. Linear least-squares is analytical so it ignores **kwargs.

Returns A 3-dimensional xarray with the xs, ys as first two axes. The third axis, called 'results', contains the parameters and an assessment of the fit quality. If method is least-squres, it is the sum of squared residuals.

NOTE: This function fits every combination of x and y independently, interactions are not considered.

singlet.dataset.plot

class `singlet.dataset.plot.`**`Plot`**(*dataset*)

Bases: `singlet.dataset.plugins.Plugin`

Plot gene expression and phenotype in single cells

`clustermap`(*cluster_samples=False*, *cluster_features=False*, *phenotypes_cluster_samples=()*, *phenotypes_cluster_features=()*, *annotate_samples=False*, *annotate_features=False*, *labels_samples=True*, *labels_features=True*, *orientation='horizontal'*, *colorbars=False*, ***kwargs*)

Samples versus features / phenotypes.

Parameters

- **cluster_samples** (*bool or linkage*) – Whether to cluster samples and show the dendrogram. Can be either, False, True, or a linkage from scipy.cluster.hierarchy.linkage.
- **cluster_features** (*bool or linkage*) – Whether to cluster features and show the dendrogram. Can be either, False, True, or a linkage from scipy.cluster.hierarchy.linkage.
- **phenotypes_cluster_samples** (*iterable of strings*) – Phenotypes to add to the features for joint clustering of the samples. If the clustering has been precomputed including phenotypes and the linkage matrix is explicitely set as cluster_samples, the *same* phenotypes must be specified here, in the same order.
- **phenotypes_cluster_features** (*iterable of strings*) – Phenotypes to add to the features for joint clustering of the features and phenotypes. If the clustering has been precomputed including phenotypes and the linkage matrix is explicitely set as cluster_features, the *same* phenotypes must be specified here, in the same order.
- **annotate_samples** (*dict, or False*) – Whether and how to annotate the samples with separate colorbars. The dictionary must have phenotypes or features as keys. For qualitative phenotypes, the values can be palette names or palettes (with at least as many colors as there are categories). For quantitative phenotypes and features, they can be colormap names or colormaps.
- **annotate_features** (*dict, or False*) – Whether and how to annotate the featues with separate colorbars. The dictionary must have features metadata as keys. For qualitative annotations, the values can be palette names or palettes (with at least as many colors as there are categories). For quantitative annotatoins, the values can be colormap names or colormaps. Keys must be columns of the Dataset.featuresheet, except for the key 'mean expression' which is interpreted to mean the average of the counts for that feature.
- **labels_samples** (*bool*) – Whether to show the sample labels. If you have hundreds or more samples, you may want to turn this off to make the plot tidier.
- **labels_features** (*bool*) – Whether to show the feature labels. If you have hundreds or more features, you may want to turn this off to make the plot tidier.
- **orientation** (*string*) – Whether the samples are on the abscissa ('horizontal') or on the ordinate ('vertical').
- **tight_layout** (*bool or dict*) – Whether to call matplotlib.pyplot.tight_layout at the end of the plotting. If it is a dict, pass it unpacked to that function.
- **colorbars** (*bool*) – Whether to add colorbars. One colorbar refers to the heatmap. Moreover, if annotations for samples or features are shown, a colorbar for each of them will be shown as well.
- ****kwargs** – named arguments passed to seaborn.clustermap.

Returns A seaborn ClusterGrid instance.

plot_coverage (*features='total', kind='cumulative', ax=None, tight_layout=True, legend=False, **kwargs*)

Plot number of reads for each sample

Parameters

- **features** (*list or string*) – Features to sum over. The string 'total' means all features including spikeins and other, 'mapped' means all features excluding spikeins and other, 'spikeins' means only spikeins, and 'other' means only 'other' features.
- **kind** (*string*) – Kind of plot (default: cumulative distribution).

- **ax** (*matplotlib.axes.Axes*) – The axes to plot into. If None (default), a new figure with one axes is created. ax must not strictly be a matplotlib class, but it must have common methods such as 'plot' and 'set'.
- **tight_layout** (*bool or dict*) – Whether to call matplotlib.pyplot.tight_layout at the end of the plotting. If it is a dict, pass it unpacked to that function.
- **legend** (*bool or dict*) – If True, call ax.legend(). If a dict, pass as **kwargs to ax.legend.
- ****kwargs** – named arguments passed to the plot function.

Returns matplotlib.axes.Axes with the axes containg the plot.

plot_distributions (*features*, *kind='violin'*, *ax=None*, *tight_layout=True*, *legend=False*, *orientation='vertical'*, *sort=False*, *bottom=0*, *grid=None*, ***kwargs*)

Plot distribution of spike-in controls

Parameters

- **features** (*list or string*) – List of features to plot. If it is the string 'spikeins', plot all spikeins, if the string 'other', plot other features.
- **kind** (*string*) – Kind of plot, one of 'violin' (default), 'box', 'swarm'.
- **ax** (*matplotlib.axes.Axes*) – Axes to plot into. If None (default), create a new figure and axes.
- **tight_layout** (*bool or dict*) – Whether to call matplotlib.pyplot.tight_layout at the end of the plotting. If it is a dict, pass it unpacked to that function.
- **legend** (*bool or dict*) – If True, call ax.legend(). If a dict, pass as **kwargs to ax.legend. Notice that legend has a special meaning in these kinds of seaborn plots.
- **orientation** (*string*) – 'horizontal' or 'vertical'.
- **sort** (*bool or string*) – True or 'ascending' sorts the features by median, 'descending' uses the reverse order.
- **bottom** (*float or string*) – The value of zero-count features. If you are using a log axis, you may want to set this to 0.1 or any other small positive number. If a string, it must be 'pseudocount', then the CountsTable.pseudocount will be used.
- **grid** (*bool or None*) – Whether to add a grid to the plot. None defaults to your existing settings.
- ****kwargs** – named arguments passed to the plot function.

Returns The axes with the plot.

Return type matplotlib.axes.Axes

scatter_reduced_samples (*vectors_reduced*, *color_by=None*, *color_log=None*, *cmap='viridis'*, *ax=None*, *tight_layout=True*, ***kwargs*)

Scatter samples after dimensionality reduction.

Parameters

- **vectors_reduced** (*pandas.Dataframe*) – matrix of coordinates of the samples after dimensionality reduction. Rows are samples, columns (typically 2 or 3) are the component in the low-dimensional embedding.
- **color_by** (*string or None*) – color sample dots by phenotype or expression of a certain feature.

- **color_log** (*bool or None*) – use log of phenotype/expression in the colormap. Default None only logs expression, but not phenotypes.
- **cmap** (*string or matplotlib colormap*) – color map to use for the sample dots.
- **ax** (*matplotlib.axes.Axes*) – The axes to plot into. If None (default), a new figure with one axes is created. ax must not strictly be a matplotlib class, but it must have common methods such as 'plot' and 'set'.
- **tight_layout** (*bool or dict*) – Whether to call matplotlib.pyplot.tight_layout at the end of the plotting. If it is a dict, pass it unpacked to that function.
- ****kwargs** – named arguments passed to the plot function.

Returns matplotlib.axes.Axes with the axes containing the plot.

scatter_statistics (*features='mapped', x='mean', y='cv', ax=None, tight_layout=True, legend=False, grid=None, **kwargs*)
Scatter plot statistics of features.

Parameters

- **features** (*list or string*) – List of features to plot. The string 'mapped' means everything excluding spikeins and other, 'all' means everything including spikeins and other.
- **x** (*string*) – Statistics to plot on the x axis.
- **y** (*string*) – Statistics to plot on the y axis.
- **ax** (*matplotlib.axes.Axes*) – The axes to plot into. If None (default), a new figure with one axes is created. ax must not strictly be a matplotlib class, but it must have common methods such as 'plot' and 'set'.
- **tight_layout** (*bool or dict*) – Whether to call matplotlib.pyplot.tight_layout at the end of the plotting. If it is a dict, pass it unpacked to that function.
- **legend** (*bool or dict*) – If True, call ax.legend(). If a dict, pass as **kwargs to ax.legend.
- **grid** (*bool or None*) – Whether to add a grid to the plot. None defaults to your existing settings.
- ****kwargs** – named arguments passed to the plot function.

Returns matplotlib.axes.Axes with the axes contaiing the plot.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Bibliography

[tsne] L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008.

Python Module Index

Index