anesthetic Documentation

Release 1.2.3

Will Handley

Nov 25, 2019

Contents

anesthetic nested sampling visualisation

Author Will Handley

Version 1.2.3

Homepage https://github.com/williamjameshandley/anesthetic

Documentation http://anesthetic.readthedocs.io/

`anesthetic` brings together tools for processing nested sampling chains, leveraging standard scientific python libraries.

You can see example usage and plots in the plot gallery, or in the corresponding Jupyter notebook.

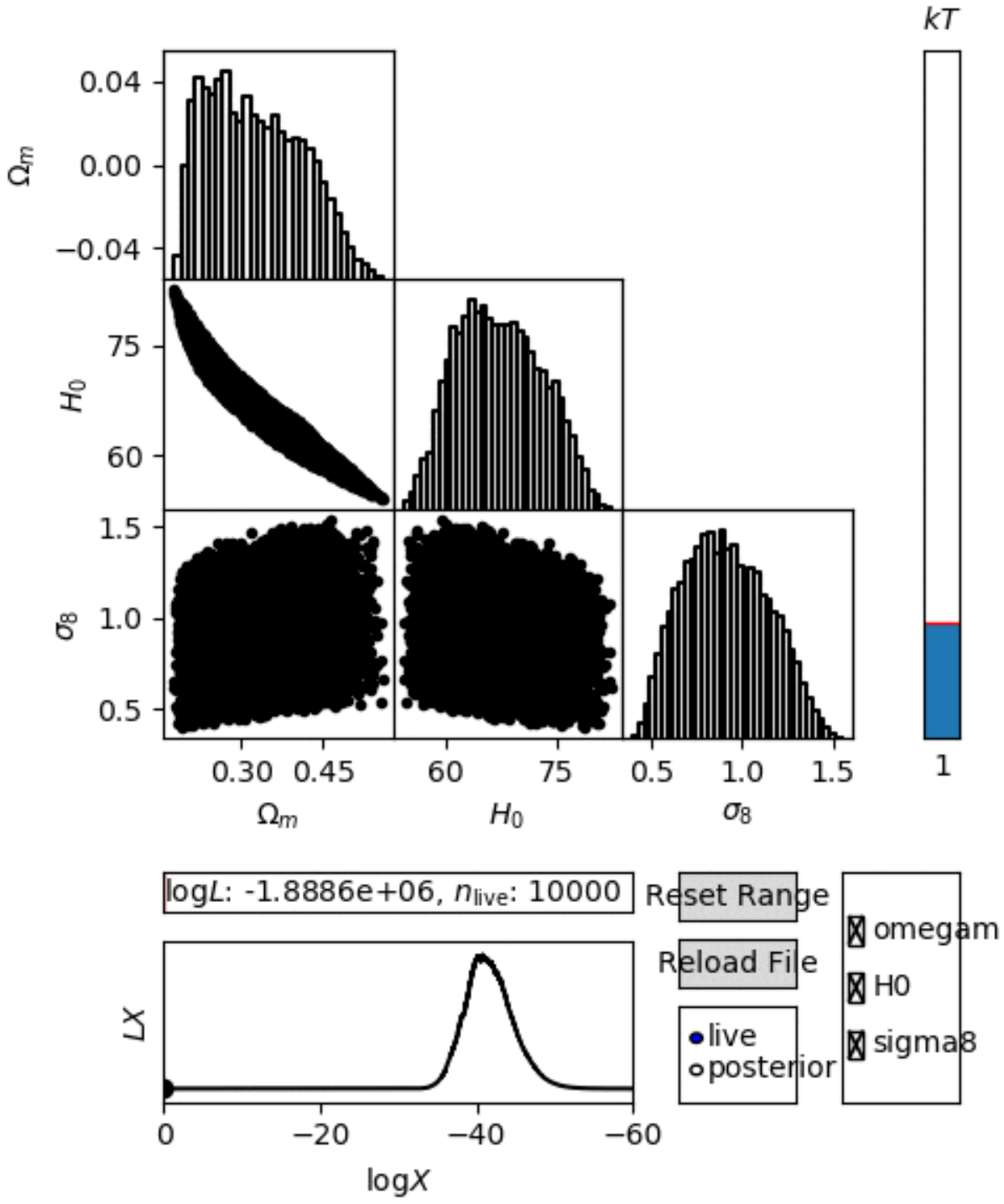
Current functionality includes:

- Computation of Bayesian evidences, Kullback-Liebler divergences and Bayesian model dimensionalities.
- Marginalised 1d and 2d plots.
- Dynamic replaying of nested sampling.

This tool was designed primarily for use with nested sampling outputs, although it can be used for normal MCMC chains.

For an interactive view of a nested sampling run, you can use the `anesthetic` script.

```
$ anesthetic <ns file root>
```


kT
0.04
0.00
−0.04
Ω_m
75
60
H_0
1.5
1.0
0.5
σ_8
0.30
0.45
60
75
0.5
1.0
1.5
Ω_m
H_0
σ_8
1
logL: -1.8886e+06, n_{live}: 10000
Reset Range
Reload File
live
posterior
omegam
H0
sigma8
LX
0
−20
−40
−60
logX

CHAPTER 1

Features

- Both samples and plotting axes are stored as a `pandas.DataFrame`, with parameter names as indices, which makes for easy access and modification.
- Sensible color scheme for plotting nearly flat distributions.
- For easy extension/modification, uses the standard python libraries: numpy, scipy, matplotlib and pandas.

CHAPTER 2

Installation

`anesthetic` can be installed via pip

```
pip install anesthetic
```

or via the setup.py

```
git clone https://github.com/williamjameshandley/anesthetic
cd anesthetic
python setup.py install --user
```

You can check that things are working by running the test suite:

```
export MPLBACKEND=Agg      # only necessary for OSX users
python -m pytest
flake8 anesthetic tests
pydocstyle --convention=numpy anesthetic
```

2.1 Dependencies

Basic requirements:

- Python 2.7 or 3.5+
- matplotlib
- numpy
- scipy
- pandas
- fastKDE

Documentation:

- sphinx
- numpydoc

Tests:

- pytest

CHAPTER 3

Documentation

Full Documentation is hosted at ReadTheDocs. To build your own local copy of the documentation you'll need to install sphinx. You can then run:

```
cd docs
make html
```

CHAPTER 4

Citation

If you use `anesthetic` to generate plots for a publication, please cite as:

```
Handley, (2019). anesthetic: nested sampling visualisation. Journal of Open
Source Software, 4(37), 1414, https://doi.org/10.21105/joss.01414
```

or using the BibTeX:

```
@article{anesthetic,
    doi = {10.21105/joss.01414},
    url = {http://dx.doi.org/10.21105/joss.01414},
    year  = {2019},
    month = {Jun},
    publisher = {The Open Journal},
    volume = {4},
    number = {37},
    pages = {1414},
    author = {Will Handley},
    title = {anesthetic: nested sampling visualisation},
    journal = {The Journal of Open Source Software}
}
```

CHAPTER 5

Contributing

There are many ways you can contribute via the GitHub repository.

- You can open an issue to report bugs or to propose new features.
- Pull requests are very welcome. Note that if you are going to propose major changes, be sure to open an issue for discussion first, to make sure that your PR will be accepted before you spend effort coding it.

CHAPTER 6

Questions/Comments

6.1 Another posterior plotting tool?

> This is my posterior plotter. There are many like it, but this one is mine.

There are several excellent tools for plotting marginalised posteriors:

- getdist
- corner
- pygtc
- dynesty
- MontePython

Why create another one? In general, any dedicated user of software will find that there is some functionality that in their use case is lacking, and the designs of previous codes make such extensions challenging. In my case this was:

1. For large numbers of samples, kernel density estimation is slow, or inaccurate (particularly for samples generated from nested sampling). There are kernel density estimators, such as fastKDE, which ameliorate many of these difficulties.
2. Existing tools can make it difficult to define new parameters. For example, the default cosmomc chain defines `omegabh2`, but not `omegab`. The transformation is easy, since `omegab = omegabh2/ (H0/100)**2`, but implementing this transformation in existing packages is not so trivial. `anesthetic` solves this issue by storing the samples as a pandas array, for which the relevant code for defining the above new parameter would be

```
from anesthetic import MCMCSamples
samples = MCMCSamples(root=file_root)                    # Load the samples
samples['omegab'] = samples.omegabh2/(samples.H0/100)**2 # Define omegab
samples.tex['omegab'] = '$\Omega_b$'                     # Label omegab
samples.plot_1d('omegab')                                # Simple 1D plot
```

3. Many KDE plotting tools have conventions that don't play well with uniformly distributed parameters, which presents a problem if you are trying to plot priors along with your posteriors. `anesthetic` has a sensible mechanism, by defining the contours by the amount of iso-probability mass they contain, but colouring the fill in relation to the probability density of the contour.

6.2 What's in a name?

There is an emerging convention for naming nested sampling packages with words that have nest in them (nestle and dynesty, nestorflow). Doing a UNIX grep:

```
grep nest /usr/share/dict/words
```

yields a lot of superlatives (e.g. greenest), but a few other cool names for future projects:

- amnesty
- defenestrate
- dishonestly
- inestimable
- minestrone
- rhinestone

I chose `anesthetic` because I liked the soft 'th', and in spite of the US spelling.

CHAPTER 7

anesthetic package

7.1 Subpackages

7.1.1 anesthetic.gui package

Submodules

anesthetic.gui.plot module

Main plotting tools.

class anesthetic.gui.plot.**Evolution**(*fig*, *gridspec*, *action*, *valmax*)

Bases: *anesthetic.gui.widgets.Slider*

Slider controlling the evolution stage of the live points.

Methods

`__call__`()	Return the current iteration as an integer.
`reset_range`([valmin, valmax])	Reset the range of the slider.
`set_text`(logL, n)	Set the text at end of slider.

set_text(*logL*, *n*)

Set the text at end of slider.

Parameters

logL: float

Current loglikelihood of evolution stage

n: int Current number of live points of evolution stage

class anesthetic.gui.plot.**Higson**(*fig*, *gridspec*)
Bases: *anesthetic.gui.widgets.Widget*

Higson plot as shown in https://arxiv.org/abs/1703.09701 .

Attributes

curve: matplotlib.lines.Line2D

points currently plotted as a curve.

point: matplotlib.lines.Line2D large indicator point currently plotted on the curve.

Methods

reset_range()	Reset the ranges of the higson plot.
update(logX, LX, i)	Update the line and the point in the higson plot.

reset_range()
Reset the ranges of the higson plot.

update(*logX*, *LX*, *i*)
Update the line and the point in the higson plot.

Parameters

logX: array-like

log-volume compression values to plot

LX: array-like Likelihood * volume compression

i: int Current location of higson point

class anesthetic.gui.plot.**RunPlotter**(*samples*, *params=None*)
Bases: object

Construct a control panel of information on a nested sampling run.

Parameters

samples: anesthetic.samples.NestedSamples The root string for the chains files to be used, or a set of nested samples.

Attributes

samples: anesthetic.samples.NestedSamples

Object for extracting nested sampling data from chains files.

fig: matplotlib.figure.Figure Reference to the underlying figure

triangle: anesthetic.gui.widgets.TrianglePlot Corner plot of live or posterior samples.

temperature: anesthetic.gui.plot.Temperature Slider selecting the posterior temperature.

evolution: anesthetic.gui.plot.Evolution Slider selecting the live iteration.

higson: anesthetic.gui.plot.Higson Higson plot of posterior weights.

reset: anesthetic.gui.widgets.Button Button that resets the parameter ranges.

reload: anesthetic.gui.widgets.Button Button that reloads the files.

type: anesthetic.gui.widgets.RadioButtons Radio buttons that selects whether to plot live or posteriors.

param_choice: anesthetic.gui.widgets.CheckButtons Checkbox that selects which parameters to plot.

Methods

points(label)	Get sample coordinates from nested sampling samples.
redraw(_)	Redraw the triangle plot upon parameter updating.
reload_file(_)	Reload the data from file.
reset_range(_)	Reset the parameter ranges.
update(_)	Update all the plots upon slider changes.

points(*label*)
Get sample coordinates from nested sampling samples.

Parameters

label: str label indicating the coordinate to extract.

Returns

array-like: sample 'label'-coordinates.

redraw(_)
Redraw the triangle plot upon parameter updating.

reload_file(_)
Reload the data from file.

reset_range(_)
Reset the parameter ranges.

update(_)
Update all the plots upon slider changes.

class anesthetic.gui.plot.**Temperature**(*fig*, *gridspec*, *action*)
Bases: *anesthetic.gui.widgets.Slider*

Logarithmic slider controlling temperature of the posterior points.

Methods

__call__()	Return the current temperature.
reset_range([valmin, valmax])	Reset the range of the slider.
set_text(kT)	Set the text at end of slider.

set_text(*kT*)
Set the text at end of slider.

Parameters

kT: float Current temperature of posterior points stage

anesthetic.gui.widgets module

Widget wrappers to matplotlib.

These extend the matplotlib widgets by plotting themselves onto an axis and storing a reference to both the widget object and the axis on which they are plotted.

class anesthetic.gui.widgets.**Button**(*fig*, *gridspec*, *action*, *text*)
Bases: *anesthetic.gui.widgets.Widget*

Push button that performs an action.

Parameters

action: func

What should be run upon clicking the button.

text: str Overlay text on the button.

Attributes

button: matplotlib.widgets.Button matplotlib Button.

class anesthetic.gui.widgets.**CheckButtons**(*fig*, *gridspec*, *labels*, *action*)
Bases: *anesthetic.gui.widgets.LabelsWidget*

Set of checkboxes.

Defaults to choosing the only the first label at start.

Parameters

action: func What should be done upon checking the box.

Attributes

buttons: matplotlib.widgets.CheckButtons matplotlib CheckButtons.

Methods

__call__()	Get current status of the check boxes.

class anesthetic.gui.widgets.**LabelsWidget**(*fig*, *gridspec*, *labels*)
Bases: *anesthetic.gui.widgets.Widget*

Widget with labels to choose from.

Parameters

labels: list(str) Set of labels to be tied to Widget.

Attributes

labels: list(str) Set of labels on Widget.

class anesthetic.gui.widgets.**RadioButtons**(*fig*, *gridspec*, *labels*, *action*)
Bases: *anesthetic.gui.widgets.LabelsWidget*

Set of radio selection choices.

Parameters

labels: list(str))

Attributes

buttons: matplotlib.widgets.RadioButtons matplotlib RadioButtons.

Methods

`__call__`()	Get current selection string.

class anesthetic.gui.widgets.**Slider**(*fig*, *gridspec*, *action*, *text*, *valmin*, *valmax*, *valinit*, *orientation*)

Bases: *anesthetic.gui.widgets.Widget*

Choose a parameter via a slider widget.

Parameters

action: func

What should be done upon altering the slider.

text: str Text at the start of the slider.

valmin, valmax, valinit: float Range and initial value for the slider.

orientation: str Orientation for slider ('horizontal' or 'vertical').

Attributes

slider: matplotlib.widgets.Slider matplotlib Slider.

Methods

`__call__`()	Return the float value in the slider.
reset_range([valmin, valmax])	Reset the range of the slider.
set_text(text)	Set the text at the end of the slider.

reset_range(*valmin=None*, *valmax=None*)

Reset the range of the slider.

Kwargs:

valmin, valmax: (float), optional: The new limits to the slider.

set_text(*text*)

Set the text at the end of the slider.

Parameters

text: str Text to be chosen

class anesthetic.gui.widgets.**TrianglePlot**(*fig*, *gridspec*)

Bases: *anesthetic.gui.widgets.Widget*

Triangle plot widget as exemplified by getdist and corner.py.

For other examples of these plots, see: https://getdist.readthedocs.io https://corner.readthedocs.io

Attributes

ax: pandas.DataFrame(matplotlib.axes.Axes) Mapping from pairs of parameters to axes for plotting.

Methods

draw(labels[, tex])	Draw a new triangular grid for list of parameters labels.
reset_range()	Reset the range of each grid.
update(f)	Update the points in the triangle plot using f function.

draw(*labels*, *tex={}*)
Draw a new triangular grid for list of parameters labels.

Parameters

labels: list(str) labels for the triangular grid.

reset_range()
Reset the range of each grid.

update(*f*)
Update the points in the triangle plot using f function.

Parameters

f: func: str -> array(float) this function should take in a parameter label i, and return an array-like object of the i-coordinate of the samples

class anesthetic.gui.widgets.**Widget**(*fig*, *gridspec*)
Bases: object

Base class for anesthetic gui widgets.

Stores a reference to the underlying figure, the gridspec that the widget is placed at and the axes of the widget.

Parameters

fig: matplotlib.figure.Figure

Figure for drawing widget on.

gridspec: matplotlib.gridspec.GridSpec Specification for where to draw in the figure. Technically could be any argument that can be passed to matplotlib.figure.Figure.add_subplot.

Attributes

fig: (matplotlib.figure.Figure

Figure for drawing widget on.

gridspec: matplotlib.gridspec.GridSpec Specification for where to draw in the figure. Technically could be any argument that can be passed to matplotlib.figure.Figure.add_subplot.

ax: matplotlib.axes.Axes Axes of widget.

Module contents

Graphical user interface for inspecting nested sampling runs.

7.1.2 anesthetic.read package

Submodules

anesthetic.read.base module

anesthetic.read.getdist module

anesthetic.read.multinest module

anesthetic.read.nested module

anesthetic.read.polychord module

Module contents

Module for reading samples from file.

7.2 Submodules

7.3 anesthetic.kde module

Kernel density estimation tools.

These act as a wrapper around fastKDE, but could be replaced in future by alternative kernel density estimators

anesthetic.kde.**fastkde_1d**(*d*, *xmin=None*, *xmax=None*)
Perform a one-dimensional kernel density estimation.

Wrapper round fastkde.fastKDE. Boundary corrections implemented by reflecting boundary conditions.

Parameters

d: numpy.array Data to perform kde on

xmin, xmax: float lower/upper prior bounds optional, default None

Returns

x: numpy.array x-coordinates of kernel density estimates

p: numpy.array kernel density estimates

anesthetic.kde.**fastkde_2d**(*d_x*, *d_y*, *xmin=None*, *xmax=None*, *ymin=None*, *ymax=None*)
Perform a two-dimensional kernel density estimation.

Wrapper round fastkde.fastKDE. Boundary corrections implemented by reflecting boundary conditions.

Parameters

d_x, d_y: numpy.array x/y coordinates of data to perform kde on

xmin, xmax, ymin, ymax: float lower/upper prior bounds in x/y coordinates optional, default None

Returns

x,y: numpy.array x/y-coordinates of kernel density estimates. One-dimensional array

p: numpy.array kernel density estimates. Two-dimensional array

7.4 anesthetic.plot module

Lower-level plotting tools.

Routines that may be of use to users wishing for more fine-grained control may wish to use.

- `make_1d_axes`
- `make_2d_axes`
- `get_legend_proxy`

to create a set of axes and legend proxies.

`anesthetic.plot.`**`basic_cmap`**(*color*)
Construct basic colormap a single color.

`anesthetic.plot.`**`fastkde_contour_plot_2d`**(*ax*, *data_x*, *data_y*, **args*, ***kwargs*)
Plot a 2d marginalised distribution as contours.

This functions as a wrapper around matplotlib.axes.Axes.contour, and matplotlib.axes.Axes.contourf with a kernel density estimation computation in between. All remaining keyword arguments are passed onwards to both functions.

Parameters

ax: matplotlib.axes.Axes axis object to plot on

data_x, data_y: numpy.array x and y coordinates of uniformly weighted samples to generate kernel density estimator.

levels: list amount of mass within each iso-probability contour. optional, default [0.68, 0.95]

xmin, xmax, ymin, ymax: float lower/upper prior bounds in x/y coordinates optional, default None

Returns

c: matplotlib.contour.QuadContourSet A set of contourlines or filled regions

`anesthetic.plot.`**`fastkde_plot_1d`**(*ax*, *data*, **args*, ***kwargs*)
Plot a 1d marginalised distribution.

This functions as a wrapper around matplotlib.axes.Axes.plot, with a kernel density estimation computation provided by the package fastkde in between. All remaining keyword arguments are passed onwards.

Parameters

ax: matplotlib.axes.Axes axis object to plot on

data: numpy.array Uniformly weighted samples to generate kernel density estimator.

xmin, xmax: float lower/upper prior bound optional, default None

Returns

lines: matplotlib.lines.Line2D A list of line objects representing the plotted data (same as matplotlib matplotlib.axes.Axes.plot command)

anesthetic.plot.**get_legend_proxy**(*fig*)

Extract a proxy for plotting onto a legend.

Example usage:

```
>>> fig, axes = modelA.plot_2d()
>>> modelB.plot_2d(axes)
>>> proxy = get_legend_proxy(fig)
>>> fig.legend(proxy, ['A', 'B']
```

Parameters

fig: matplotlib.figure.Figure Figure to extract colors from.

anesthetic.plot.**hist_plot_1d**(*ax*, *data*, **args*, ***kwargs*)

Plot a 1d histogram.

This functions is a wrapper around matplotlib.axes.Axes.hist. All remaining keyword arguments are passed onwards.

Parameters

ax: matplotlib.axes.Axes axis object to plot on

data: numpy.array Samples to generate histogram from

weights: numpy.array, optional Sample weights.

xmin, xmax: float lower/upper prior bound. optional, default data.min() and data.max() cannot be None (reverts to default in that case)

Returns

patches [list or list of lists] Silent list of individual patches used to create the histogram or list of such list if multiple input datasets.

Other Parameters

****kwargs** [~*matplotlib.axes.Axes.hist* properties]

anesthetic.plot.**hist_plot_2d**(*ax*, *data_x*, *data_y*, **args*, ***kwargs*)

Plot a 2d marginalised distribution as a histogram.

This functions as a wrapper around matplotlib.axes.Axes.hist2d

Parameters

ax: matplotlib.axes.Axes axis object to plot on

data_x, data_y: numpy.array x and y coordinates of uniformly weighted samples to generate kernel density estimator.

xmin, xmax, ymin, ymax: float lower/upper prior bounds in x/y coordinates optional, default None

levels: list Shade iso-probability contours containing these levels of probability mass. If None defaults to usual matplotlib.axes.Axes.hist2d colouring. optional, default None

Returns

c: matplotlib.collections.QuadMesh A set of colors

anesthetic.plot.**kde_contour_plot_2d**(*ax*, *data_x*, *data_y*, **args*, ***kwargs*)

Plot a 2d marginalised distribution as contours.

This functions as a wrapper around matplotlib.axes.Axes.tricontour, and matplotlib.axes.Axes.tricontourf with a kernel density estimation computation provided by scipy.stats.gaussian_kde in between. All remaining keyword arguments are passed onwards to both functions.

Parameters

ax: matplotlib.axes.Axes axis object to plot on.

data_x, data_y: numpy.array x and y coordinates of uniformly weighted samples to generate kernel density estimator.

weights: numpy.array, optional Sample weights.

ncompress: int, optional Degree of compression. optional, Default 1000

xmin, xmax, ymin, ymax: float lower/upper prior bounds in x/y coordinates. optional, default None

Returns

c: matplotlib.contour.QuadContourSet A set of contourlines or filled regions

anesthetic.plot.**kde_plot_1d**(*ax*, *data*, **args*, ***kwargs*)

Plot a 1d marginalised distribution.

This functions as a wrapper around matplotlib.axes.Axes.plot, with a kernel density estimation computation provided by scipy.stats.gaussian_kde in between. All remaining keyword arguments are passed onwards.

Parameters

ax: matplotlib.axes.Axes axis object to plot on.

data: numpy.array Samples to generate kernel density estimator.

weights: numpy.array, optional Sample weights.

ncompress: int, optional Degree of compression. Default 1000

xmin, xmax: float lower/upper prior bound. optional, default None

Returns

lines: matplotlib.lines.Line2D A list of line objects representing the plotted data (same as matplotlib matplotlib.axes.Axes.plot command)

anesthetic.plot.**make_1d_axes**(*params*, ***kwargs*)

Create a set of axes for plotting 1D marginalised posteriors.

Parameters

params: list(str)

names of parameters.

tex: dict(str:str), optional Dictionary mapping params to tex plot labels.

fig: matplotlib.figure.Figure, optional Figure to plot on. Default: matplotlib.pyplot.figure()

ncols: int Number of columns in the plot option, default ceil(sqrt(num_params))

subplot_spec: matplotlib.gridspec.GridSpec, optional gridspec to plot array as part of a subfigure Default: None

Returns

fig: matplotlib.figure.Figure New or original (if supplied) figure object

axes: pandas.Series(matplotlib.axes.Axes) Pandas array of axes objects

anesthetic.plot.**make_2d_axes**(*params*, ***kwargs*)
Create a set of axes for plotting 2D marginalised posteriors.

Parameters

params: lists of parameters

Can be either: * list(str) if the x and y axes are the same * [list(str),list(str)] if the x and y axes are different Strings indicate the names of the parameters

tex: dict(str:str), optional Dictionary mapping params to tex plot labels. Default: params

upper, lower, diagonal: logical, optional Whether to create 2D marginalised plots above or below the diagonal, or to create a 1D marginalised plot on the diagonal. Default: True

fig: matplotlib.figure.Figure, optional Figure to plot on. Default: matplotlib.pyplot.figure()

subplot_spec: matplotlib.gridspec.GridSpec, optional gridspec to plot array as part of a subfigure. Default: None

Returns

fig: matplotlib.figure.Figure New or original (if supplied) figure object

axes: pandas.DataFrame(matplotlib.axes.Axes) Pandas array of axes objects

anesthetic.plot.**make_diagonal**(*ax*)
Link x and y axes limits.

anesthetic.plot.**scatter_plot_2d**(*ax*, *data_x*, *data_y*, **args*, ***kwargs*)
Plot samples from a 2d marginalised distribution.

This functions as a wrapper around matplotlib.axes.Axes.plot, enforcing any prior bounds. All remaining keyword arguments are passed onwards.

Parameters

ax: matplotlib.axes.Axes axis object to plot on

data_x, data_y: numpy.array x and y coordinates of uniformly weighted samples to plot.

xmin, xmax, ymin, ymax: float lower/upper prior bounds in x/y coordinates optional, default None

Returns

lines: matplotlib.lines.Line2D A list of line objects representing the plotted data (same as matplotlib matplotlib.axes.Axes.plot command)

7.5 anesthetic.samples module

Main classes for the anesthetic module.

- `MCMCSamples`
- `NestedSamples`

class anesthetic.samples.**MCMCSamples**(**args*, ***kwargs*)

Bases: *anesthetic.weighted_pandas.WeightedDataFrame*

Storage and plotting tools for MCMC samples.

Extends the pandas.DataFrame by providing plotting methods and standardising sample storage.

Example plotting commands include

- `mcmc.plot_1d(['paramA', 'paramB'])`
- `mcmc.plot_2d(['paramA', 'paramB'])`
- `mcmc.plot_2d([['paramA', 'paramB'], ['paramC', 'paramD']])`

Parameters

root: str, optional root for reading chains from file. Overrides all other arguments.

data: numpy.array Coordinates of samples. shape = (nsamples, ndims).

columns: list(str) reference names of parameters

w: numpy.array weights of samples.

logL: numpy.array loglikelihoods of samples.

tex: dict mapping from columns to tex labels for plotting

limits: dict mapping from columns to prior limits

label: str Legend label

Attributes

T Transpose index and columns.

at Access a single value for a row/column label pair.

axes Return a list representing the axes of the DataFrame.

blocks Internal property, property synonym for as_blocks().

columns The column labels of the DataFrame.

dtypes Return the dtypes in the DataFrame.

empty Indicator whether DataFrame is empty.

ftypes Return the ftypes (indication of sparse/dense and dtype) in DataFrame.

iat Access a single value for a row/column pair by integer position.

iloc Purely integer-location based indexing for selection by position.

index The index (row labels) of the DataFrame.

is_copy Return the copy.

ix A primarily label-location based indexer, with integer position fallback.

loc Access a group of rows and columns by label(s) or a boolean array.

ndim Return an int representing the number of axes / array dimensions.

shape Return a tuple representing the dimensionality of the DataFrame.

size Return an int representing the number of elements in this object.

style Property returning a Styler object containing methods for building a styled HTML representation fo the DataFrame.

timetuple

values Return a Numpy representation of the DataFrame.

weight Sample weights.

Methods

abs()	Return a Series/DataFrame with absolute numeric value of each element.
add(other[, axis, level, fill_value])	Addition of dataframe and other, element-wise (binary operator *add*).
add_prefix(prefix)	Prefix labels with string *prefix*.
add_suffix(suffix)	Suffix labels with string *suffix*.
agg(func[, axis])	Aggregate using one or more operations over the specified axis.
aggregate(func[, axis])	Aggregate using one or more operations over the specified axis.
align(other[, join, axis, level, copy, ...])	Align two objects on their axes with the specified join method for each axis Index.
all([axis, bool_only, skipna, level])	Return whether all elements are True, potentially over an axis.
any([axis, bool_only, skipna, level])	Return whether any element is True, potentially over an axis.
append(other[, ignore_index, ...])	Append rows of *other* to the end of caller, returning a new object.
apply(func[, axis, broadcast, raw, reduce, ...])	Apply a function along an axis of the DataFrame.
applymap(func)	Apply a function to a Dataframe elementwise.
as_blocks([copy])	Convert the frame to a dict of dtype -> Constructor Types that each has a homogeneous dtype.
as_matrix([columns])	Convert the frame to its Numpy-array representation.
asfreq(freq[, method, how, normalize, ...])	Convert TimeSeries to specified frequency.
asof(where[, subset])	Return the last row(s) without any NaNs before *where*.
assign(**kwargs)	Assign new columns to a DataFrame.
astype(dtype[, copy, errors])	Cast a pandas object to a specified dtype `dtype`.
at_time(time[, asof, axis])	Select values at particular time of day (e.g.
between_time(start_time, end_time[, ...])	Select values between particular times of the day (e.g., 9:00-9:30 AM).
bfill([axis, inplace, limit, downcast])	Synonym for `DataFrame.fillna()` with `method='bfill'`.
bool()	Return the bool of a single element PandasObject.
boxplot([column, by, ax, fontsize, rot, ...])	Make a box plot from DataFrame columns.
clip([lower, upper, axis, inplace])	Trim values at input threshold(s).
clip_lower(threshold[, axis, inplace])	Trim values below a given threshold.
clip_upper(threshold[, axis, inplace])	Trim values above a given threshold.
combine(other, func[, fill_value, overwrite])	Perform column-wise combine with another DataFrame based on a passed function.

Continued on next page

Table 9 – continued from previous page

combine_first(other)	Update null elements with value in the same location in *other*.
compound([axis, skipna, level])	Return the compound percentage of the values for the requested axis.
compress([nsamples])	Reduce the number of samples by discarding low-weights.
convert_objects([convert_dates, ...])	Attempt to infer better dtype for object columns.
copy([deep])	Make a copy of this object's indices and data.
corr([method, min_periods])	Compute pairwise correlation of columns, excluding NA/null values.
corrwith(other[, axis, drop, method])	Compute pairwise correlation between rows or columns of DataFrame with rows or columns of Series or DataFrame.
count([axis, level, numeric_only])	Count non-NA cells for each column or row.
cov()	Weighted covariance of the sampled distribution.
cummax([axis, skipna])	Return cumulative maximum over a DataFrame or Series axis.
cummin([axis, skipna])	Return cumulative minimum over a DataFrame or Series axis.
cumprod([axis, skipna])	Return cumulative product over a DataFrame or Series axis.
cumsum([axis, skipna])	Return cumulative sum over a DataFrame or Series axis.
describe([percentiles, include, exclude])	Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
diff([periods, axis])	First discrete difference of element.
div(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
divide(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
dot(other)	Compute the matrix mutiplication between the DataFrame and other.
drop([labels, axis, index, columns, level, ...])	Drop specified labels from rows or columns.
drop_duplicates([subset, keep, inplace])	Return DataFrame with duplicate rows removed, optionally only considering certain columns.
droplevel(level[, axis])	Return DataFrame with requested index / column level(s) removed.
dropna([axis, how, thresh, subset, inplace])	Remove missing values.
duplicated([subset, keep])	Return boolean Series denoting duplicate rows, optionally only considering certain columns.
eq(other[, axis, level])	Equal to of dataframe and other, element-wise (binary operator *eq*).
equals(other)	Test whether two objects contain the same elements.
eval(expr[, inplace])	Evaluate a string describing operations on DataFrame columns.
ewm([com, span, halflife, alpha, ...])	Provides exponential weighted functions.
expanding([min_periods, center, axis])	Provides expanding transformations.
ffill([axis, inplace, limit, downcast])	Synonym for DataFrame.fillna() with method='ffill'.
fillna([value, method, axis, inplace, ...])	Fill NA/NaN values using the specified method.

Continued on next page

Table 9 – continued from previous page

filter([items, like, regex, axis])	Subset rows or columns of dataframe according to labels in the specified index.
first(offset)	Convenience method for subsetting initial periods of time series data based on a date offset.
first_valid_index()	Return index for first non-NA/null value.
floordiv(other[, axis, level, fill_value])	Integer division of dataframe and other, element-wise (binary operator *floordiv*).
from_csv(path[, header, sep, index_col, ...])	Read CSV file.
from_dict(data[, orient, dtype, columns])	Construct DataFrame from dict of array-like or dicts.
from_items(items[, columns, orient])	Construct a DataFrame from a list of tuples.
from_records(data[, index, exclude, ...])	Convert structured or record ndarray to DataFrame.
ge(other[, axis, level])	Greater than or equal to of dataframe and other, element-wise (binary operator *ge*).
get(key[, default])	Get item from object for given key (DataFrame column, Panel slice, etc.).
get_dtype_counts()	Return counts of unique dtypes in this object.
get_ftype_counts()	Return counts of unique ftypes in this object.
get_value(index, col[, takeable])	Quickly retrieve single value at passed column and index.
get_values()	Return an ndarray after converting sparse values to dense.
groupby([by, axis, level, as_index, sort, ...])	Group DataFrame or Series using a mapper or by a Series of columns.
gt(other[, axis, level])	Greater than of dataframe and other, element-wise (binary operator *gt*).
head([n])	Return the first *n* rows.
hist(*args, **kwargs)	Weighted histogram of the sampled distribution.
idxmax([axis, skipna])	Return index of first occurrence of maximum over requested axis.
idxmin([axis, skipna])	Return index of first occurrence of minimum over requested axis.
infer_objects()	Attempt to infer better dtypes for object columns.
info([verbose, buf, max_cols, memory_usage, ...])	Print a concise summary of a DataFrame.
insert(loc, column, value[, allow_duplicates])	Insert column into DataFrame at specified location.
interpolate([method, axis, limit, inplace, ...])	Interpolate values according to different methods.
isin(values)	Whether each element in the DataFrame is contained in values.
isna()	Detect missing values.
isnull()	Detect missing values.
items()	Iterator over (column name, Series) pairs.
iteritems()	Iterator over (column name, Series) pairs.
iterrows()	Iterate over DataFrame rows as (index, Series) pairs.
itertuples([index, name])	Iterate over DataFrame rows as namedtuples.
join(other[, on, how, lsuffix, rsuffix, sort])	Join columns of another DataFrame.
keys()	Get the 'info axis' (see Indexing for more)
kurt([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).

Continued on next page

Table 9 – continued from previous page

kurtosis([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).
last(offset)	Convenience method for subsetting final periods of time series data based on a date offset.
last_valid_index()	Return index for last non-NA/null value.
le(other[, axis, level])	Less than or equal to of dataframe and other, element-wise (binary operator *le*).
lookup(row_labels, col_labels)	Label-based "fancy indexing" function for DataFrame.
lt(other[, axis, level])	Less than of dataframe and other, element-wise (binary operator *lt*).
mad([axis, skipna, level])	Return the mean absolute deviation of the values for the requested axis.
mask(cond[, other, inplace, axis, level, ...])	Replace values where the condition is True.
max([axis, skipna, level, numeric_only])	Return the maximum of the values for the requested axis.
mean()	Weighted mean of the sampled distribution.
median()	Weighted median of the sampled distribution.
melt([id_vars, value_vars, var_name, ...])	Unpivots a DataFrame from wide format to long format, optionally leaving identifier variables set.
memory_usage([index, deep])	Return the memory usage of each column in bytes.
merge(right[, how, on, left_on, right_on, ...])	Merge DataFrame or named Series objects with a database-style join.
min([axis, skipna, level, numeric_only])	Return the minimum of the values for the requested axis.
mod(other[, axis, level, fill_value])	Modulo of dataframe and other, element-wise (binary operator *mod*).
mode([axis, numeric_only, dropna])	Get the mode(s) of each element along the selected axis.
mul(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *mul*).
multiply(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *mul*).
ne(other[, axis, level])	Not equal to of dataframe and other, element-wise (binary operator *ne*).
neff()	Effective number of samples.
nlargest(n, columns[, keep])	Return the first *n* rows ordered by *columns* in descending order.
notna()	Detect existing (non-missing) values.
notnull()	Detect existing (non-missing) values.
nsmallest(n, columns[, keep])	Return the first *n* rows ordered by *columns* in ascending order.
nunique([axis, dropna])	Count distinct observations over requested axis.
pct_change([periods, fill_method, limit, freq])	Percentage change between the current and a prior element.
pipe(func, *args, **kwargs)	Apply func(self, *args, **kwargs).
pivot([index, columns, values])	Return reshaped DataFrame organized by given index / column values.
pivot_table([values, index, columns, ...])	Create a spreadsheet-style pivot table as a DataFrame.

Continued on next page

Table 9 – continued from previous page

plot(ax, paramname_x[, paramname_y])	Interface for 2D and 1D plotting routines.
plot_1d(axes, *args, **kwargs)	Create an array of 1D plots.
plot_2d(axes, *args, **kwargs)	Create an array of 2D plots.
pop(item)	Return item and drop from frame.
pow(other[, axis, level, fill_value])	Exponential power of dataframe and other, element-wise (binary operator *pow*).
prod([axis, skipna, level, numeric_only, ...])	Return the product of the values for the requested axis.
product([axis, skipna, level, numeric_only, ...])	Return the product of the values for the requested axis.
quantile([q])	Weighted quantile of the sampled distribution.
query(expr[, inplace])	Query the columns of a DataFrame with a boolean expression.
radd(other[, axis, level, fill_value])	Addition of dataframe and other, element-wise (binary operator *radd*).
rank([axis, method, numeric_only, ...])	Compute numerical data ranks (1 through n) along axis.
rdiv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *rtruediv*).
reindex([labels, index, columns, axis, ...])	Conform DataFrame to new index with optional filling logic, placing NA/NaN in locations having no value in the previous index.
reindex_axis(labels[, axis, method, level, ...])	Conform input object to new index.
reindex_like(other[, method, copy, limit, ...])	Return an object with matching indices as other object.
rename([mapper, index, columns, axis, copy, ...])	Alter axes labels.
rename_axis([mapper, index, columns, axis, ...])	Set the name of the axis for the index or columns.
reorder_levels(order[, axis])	Rearrange index levels using input order.
replace([to_replace, value, inplace, limit, ...])	Replace values given in *to_replace* with *value*.
resample(rule[, how, axis, fill_method, ...])	Resample time-series data.
reset_index([level, drop, inplace, ...])	Reset the index, or a level of it.
rfloordiv(other[, axis, level, fill_value])	Integer division of dataframe and other, element-wise (binary operator *rfloordiv*).
rmod(other[, axis, level, fill_value])	Modulo of dataframe and other, element-wise (binary operator *rmod*).
rmul(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *rmul*).
rolling(window[, min_periods, center, ...])	Provides rolling window calculations.
round([decimals])	Round a DataFrame to a variable number of decimal places.
rpow(other[, axis, level, fill_value])	Exponential power of dataframe and other, element-wise (binary operator *rpow*).
rsub(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *rsub*).
rtruediv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *rtruediv*).
sample([n, frac, replace, weights, ...])	Return a random sample of items from an axis of object.
select(crit[, axis])	Return data corresponding to axis labels matching criteria.

Continued on next page

Table 9 – continued from previous page

select_dtypes([include, exclude])	Return a subset of the DataFrame's columns based on the column dtypes.
sem([axis, skipna, level, ddof, numeric_only])	Return unbiased standard error of the mean over requested axis.
set_axis(labels[, axis, inplace])	Assign desired index to given axis.
set_index(keys[, drop, append, inplace, ...])	Set the DataFrame index using existing columns.
set_value(index, col, value[, takeable])	Put single value at passed column and index.
shift([periods, freq, axis, fill_value])	Shift index by desired number of periods with an optional time *freq*.
skew([axis, skipna, level, numeric_only])	Return unbiased skew over requested axis Normalized by N-1.
slice_shift([periods, axis])	Equivalent to *shift* without copying data.
sort_index([axis, level, ascending, ...])	Sort object by labels (along an axis)
sort_values(by[, axis, ascending, inplace, ...])	Sort by the values along either axis
squeeze([axis])	Squeeze 1 dimensional axis objects into scalars.
stack([level, dropna])	Stack the prescribed level(s) from columns to index.
std()	Weighted standard deviation of the sampled distribution.
sub(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *sub*).
subtract(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *sub*).
sum([axis, skipna, level, numeric_only, ...])	Return the sum of the values for the requested axis.
swapaxes(axis1, axis2[, copy])	Interchange axes and swap values axes appropriately.
swaplevel([i, j, axis])	Swap levels i and j in a MultiIndex on a particular axis.
tail([n])	Return the last *n* rows.
take(indices[, axis, convert, is_copy])	Return the elements in the given *positional* indices along an axis.
to_clipboard([excel, sep])	Copy object to the system clipboard.
to_csv([path_or_buf, sep, na_rep, ...])	Write object to a comma-separated values (csv) file.
to_dense()	Return dense representation of NDFrame (as opposed to sparse).
to_dict([orient, into])	Convert the DataFrame to a dictionary.
to_excel(excel_writer[, sheet_name, na_rep, ...])	Write object to an Excel sheet.
to_feather(fname)	Write out the binary feather-format for DataFrames.
to_gbq(destination_table[, project_id, ...])	Write a DataFrame to a Google BigQuery table.
to_hdf(path_or_buf, key, **kwargs)	Write the contained data to an HDF5 file using HDFStore.
to_html([buf, columns, col_space, header, ...])	Render a DataFrame as an HTML table.
to_json([path_or_buf, orient, date_format, ...])	Convert the object to a JSON string.
to_latex([buf, columns, col_space, header, ...])	Render an object to a LaTeX tabular environment table.
to_msgpack([path_or_buf, encoding])	Serialize object to input file path using msgpack format.
to_numpy([dtype, copy])	Convert the DataFrame to a NumPy array.
to_panel()	Transform long (stacked) format (DataFrame) into wide (3D, Panel) format.
to_parquet(fname[, engine, compression, ...])	Write a DataFrame to the binary parquet format.

Continued on next page

Table 9 – continued from previous page

to_period([freq, axis, copy])	Convert DataFrame from DatetimeIndex to PeriodIndex with desired frequency (inferred from index if not passed).
to_pickle(path[, compression, protocol])	Pickle (serialize) object to file.
to_records([index, convert_datetime64, ...])	Convert DataFrame to a NumPy record array.
to_sparse([fill_value, kind])	Convert to SparseDataFrame.
to_sql(name, con[, schema, if_exists, ...])	Write records stored in a DataFrame to a SQL database.
to_stata(fname[, convert_dates, ...])	Export DataFrame object to Stata dta format.
to_string([buf, columns, col_space, header, ...])	Render a DataFrame to a console-friendly tabular output.
to_timestamp([freq, how, axis, copy])	Cast to DatetimeIndex of timestamps, at *beginning* of period.
to_xarray()	Return an xarray object from the pandas object.
transform(func[, axis])	Call func on self producing a DataFrame with transformed values and that has the same axis length as self.
transpose(*args, **kwargs)	Transpose index and columns.
truediv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
truncate([before, after, axis, copy])	Truncate a Series or DataFrame before and after some index value.
tshift([periods, freq, axis])	Shift the time index, using the index's frequency if available.
tz_convert(tz[, axis, level, copy])	Convert tz-aware axis to target time zone.
tz_localize(tz[, axis, level, copy, ...])	Localize tz-naive index of a Series or DataFrame to target time zone.
unstack([level, fill_value])	Pivot a level of the (necessarily hierarchical) index labels, returning a DataFrame having a new level of column labels whose inner-most level consists of the pivoted index labels.
update(other[, join, overwrite, ...])	Modify in place using non-NA values from another DataFrame.
var()	Weighted variance of the sampled distribution.
where(cond[, other, inplace, axis, level, ...])	Replace values where the condition is False.
xs(key[, axis, level, drop_level])	Return cross-section from the Series/DataFrame.

plot (*ax*, *paramname_x*, *paramname_y=None*, **args*, ***kwargs*)

Interface for 2D and 1D plotting routines.

Produces a single 1D or 2D plot on an axis.

Parameters

ax: matplotlib.axes.Axes Axes to plot on

paramname_x: str Choice of parameter to plot on x-coordinate from self.columns.

paramname_y: str Choice of parameter to plot on y-coordinate from self.columns. optional, if not provided, or the same as paramname_x, then 1D plot produced.

plot_type: str Must be in {'kde', 'scatter', 'hist', 'fastkde'} for 2D plots and in {'kde', 'hist', 'fastkde', 'astropyhist'} for 1D plots. optional, (Default: 'kde')

ncompress: int Number of samples to use in plotting routines. optional, Default dynamically chosen

Returns

fig: matplotlib.figure.Figure New or original (if supplied) figure object

axes: pandas.DataFrame or pandas.Series of matplotlib.axes.Axes Pandas array of axes objects

plot_1d(*axes*, **args*, ***kwargs*)
Create an array of 1D plots.

Parameters

axes: plotting axes

Can be:

- list(str) or str
- pandas.Series(matplotlib.axes.Axes)

If a pandas.Series is provided as an existing set of axes, then this is used for creating the plot. Otherwise a new set of axes are created using the list or lists of strings.

Returns

fig: matplotlib.figure.Figure New or original (if supplied) figure object

axes: pandas.Series of matplotlib.axes.Axes Pandas array of axes objects

plot_2d(*axes*, **args*, ***kwargs*)
Create an array of 2D plots.

To avoid intefering with y-axis sharing, one-dimensional plots are created on a separate axis, which is monkey-patched onto the argument ax as the attribute ax.twin.

Parameters

axes: plotting axes

Can be:

- list(str) if the x and y axes are the same
- [list(str),list(str)] if the x and y axes are different
- pandas.DataFrame(matplotlib.axes.Axes)

If a pandas.DataFrame is provided as an existing set of axes, then this is used for creating the plot. Otherwise a new set of axes are created using the list or lists of strings.

types: dict, optional What type (or types) of plots to produce. Takes the keys 'diagonal' for the 1D plots and 'lower' and 'upper' for the 2D plots. The options for 'diagonal are:

- 'kde'
- 'hist'
- 'astropyhist'

The options for 'lower' and 'upper' are:

- 'kde'
- 'scatter'
- 'hist'

- 'fastkde'

Default: {'diagonal': 'kde', 'lower': 'kde', 'upper':'scatter'}

diagonal_kwargs, lower_kwargs, upper_kwargs: dict, optional kwargs for the diagonal (1D)/lower or upper (2D) plots. This is useful when there is a conflict of kwargs for different types of plots. Note that any kwargs directly passed to plot_2d will overwrite any kwarg with the same key passed to <sub>_kwargs. Default: {}

Returns

fig: matplotlib.figure.Figure New or original (if supplied) figure object

axes: pandas.DataFrame of matplotlib.axes.Axes Pandas array of axes objects

class anesthetic.samples.**NestedSamples**(**args*, ***kwargs*)

Bases: *anesthetic.samples.MCMCSamples*

Storage and plotting tools for Nested Sampling samples.

We extend the MCMCSamples class with the additional methods:

- `self.ns_output()`
- `self.live_points(logL)`
- `self.posterior_points(beta)`

Parameters

root: str, optional root for reading chains from file. Overrides all other arguments.

data: numpy.array Coordinates of samples. shape = (nsamples, ndims).

columns: list(str) reference names of parameters

logL: numpy.array loglikelihoods of samples.

logL_birth: numpy.array or int birth loglikelihoods, or number of live points.

tex: dict mapping from columns to tex labels for plotting

limits: dict mapping from columns to prior limits

label: str Legend label

beta: float thermodynamic temperature

Attributes

T Transpose index and columns.

at Access a single value for a row/column label pair.

axes Return a list representing the axes of the DataFrame.

beta Thermodynamic inverse temperature.

blocks Internal property, property synonym for as_blocks().

columns The column labels of the DataFrame.

dtypes Return the dtypes in the DataFrame.

empty Indicator whether DataFrame is empty.

ftypes Return the ftypes (indication of sparse/dense and dtype) in DataFrame.

iat Access a single value for a row/column pair by integer position.

iloc Purely integer-location based indexing for selection by position.

index The index (row labels) of the DataFrame.

is_copy Return the copy.

ix A primarily label-location based indexer, with integer position fallback.

loc Access a group of rows and columns by label(s) or a boolean array.

ndim Return an int representing the number of axes / array dimensions.

shape Return a tuple representing the dimensionality of the DataFrame.

size Return an int representing the number of elements in this object.

style Property returning a Styler object containing methods for building a styled HTML representation fo the DataFrame.

timetuple

values Return a Numpy representation of the DataFrame.

weight Sample weights.

Methods

D([nsamples])	Kullback-Leibler divergence.
abs()	Return a Series/DataFrame with absolute numeric value of each element.
add(other[, axis, level, fill_value])	Addition of dataframe and other, element-wise (binary operator *add*).
add_prefix(prefix)	Prefix labels with string *prefix*.
add_suffix(suffix)	Suffix labels with string *suffix*.
agg(func[, axis])	Aggregate using one or more operations over the specified axis.
aggregate(func[, axis])	Aggregate using one or more operations over the specified axis.
align(other[, join, axis, level, copy, ...])	Align two objects on their axes with the specified join method for each axis Index.
all([axis, bool_only, skipna, level])	Return whether all elements are True, potentially over an axis.
any([axis, bool_only, skipna, level])	Return whether any element is True, potentially over an axis.
append(other[, ignore_index, ...])	Append rows of *other* to the end of caller, returning a new object.
apply(func[, axis, broadcast, raw, reduce, ...])	Apply a function along an axis of the DataFrame.
applymap(func)	Apply a function to a Dataframe elementwise.
as_blocks([copy])	Convert the frame to a dict of dtype -> Constructor Types that each has a homogeneous dtype.
as_matrix([columns])	Convert the frame to its Numpy-array representation.
asfreq(freq[, method, how, normalize, ...])	Convert TimeSeries to specified frequency.
asof(where[, subset])	Return the last row(s) without any NaNs before *where*.
assign(**kwargs)	Assign new columns to a DataFrame.
astype(dtype[, copy, errors])	Cast a pandas object to a specified dtype dtype.

Continued on next page

Table 10 – continued from previous page

at_time(time[, asof, axis])	Select values at particular time of day (e.g.
between_time(start_time, end_time[, ...])	Select values between particular times of the day (e.g., 9:00-9:30 AM).
bfill([axis, inplace, limit, downcast])	Synonym for DataFrame.fillna() with method='bfill'.
bool()	Return the bool of a single element PandasObject.
boxplot([column, by, ax, fontsize, rot, ...])	Make a box plot from DataFrame columns.
clip([lower, upper, axis, inplace])	Trim values at input threshold(s).
clip_lower(threshold[, axis, inplace])	Trim values below a given threshold.
clip_upper(threshold[, axis, inplace])	Trim values above a given threshold.
combine(other, func[, fill_value, overwrite])	Perform column-wise combine with another DataFrame based on a passed function.
combine_first(other)	Update null elements with value in the same location in *other*.
compound([axis, skipna, level])	Return the compound percentage of the values for the requested axis.
compress([nsamples])	Reduce the number of samples by discarding low-weights.
convert_objects([convert_dates, ...])	Attempt to infer better dtype for object columns.
copy([deep])	Make a copy of this object's indices and data.
corr([method, min_periods])	Compute pairwise correlation of columns, excluding NA/null values.
corrwith(other[, axis, drop, method])	Compute pairwise correlation between rows or columns of DataFrame with rows or columns of Series or DataFrame.
count([axis, level, numeric_only])	Count non-NA cells for each column or row.
cov()	Weighted covariance of the sampled distribution.
cummax([axis, skipna])	Return cumulative maximum over a DataFrame or Series axis.
cummin([axis, skipna])	Return cumulative minimum over a DataFrame or Series axis.
cumprod([axis, skipna])	Return cumulative product over a DataFrame or Series axis.
cumsum([axis, skipna])	Return cumulative sum over a DataFrame or Series axis.
d([nsamples])	Bayesian model dimensionality.
describe([percentiles, include, exclude])	Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
diff([periods, axis])	First discrete difference of element.
div(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
divide(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
dlogX([nsamples])	Compute volume of shell of loglikelihood.
dot(other)	Compute the matrix mutiplication between the DataFrame and other.
drop([labels, axis, index, columns, level, ...])	Drop specified labels from rows or columns.
drop_duplicates([subset, keep, inplace])	Return DataFrame with duplicate rows removed, optionally only considering certain columns.

Continued on next page

Table 10 – continued from previous page

droplevel(level[, axis])	Return DataFrame with requested index / column level(s) removed.
dropna([axis, how, thresh, subset, inplace])	Remove missing values.
duplicated([subset, keep])	Return boolean Series denoting duplicate rows, optionally only considering certain columns.
eq(other[, axis, level])	Equal to of dataframe and other, element-wise (binary operator *eq*).
equals(other)	Test whether two objects contain the same elements.
eval(expr[, inplace])	Evaluate a string describing operations on DataFrame columns.
ewm([com, span, halflife, alpha, ...])	Provides exponential weighted functions.
expanding([min_periods, center, axis])	Provides expanding transformations.
ffill([axis, inplace, limit, downcast])	Synonym for `DataFrame.fillna()` with `method='ffill'`.
fillna([value, method, axis, inplace, ...])	Fill NA/NaN values using the specified method.
filter([items, like, regex, axis])	Subset rows or columns of dataframe according to labels in the specified index.
first(offset)	Convenience method for subsetting initial periods of time series data based on a date offset.
first_valid_index()	Return index for first non-NA/null value.
floordiv(other[, axis, level, fill_value])	Integer division of dataframe and other, element-wise (binary operator *floordiv*).
from_csv(path[, header, sep, index_col, ...])	Read CSV file.
from_dict(data[, orient, dtype, columns])	Construct DataFrame from dict of array-like or dicts.
from_items(items[, columns, orient])	Construct a DataFrame from a list of tuples.
from_records(data[, index, exclude, ...])	Convert structured or record ndarray to DataFrame.
ge(other[, axis, level])	Greater than or equal to of dataframe and other, element-wise (binary operator *ge*).
get(key[, default])	Get item from object for given key (DataFrame column, Panel slice, etc.).
get_dtype_counts()	Return counts of unique dtypes in this object.
get_ftype_counts()	Return counts of unique ftypes in this object.
get_value(index, col[, takeable])	Quickly retrieve single value at passed column and index.
get_values()	Return an ndarray after converting sparse values to dense.
groupby([by, axis, level, as_index, sort, ...])	Group DataFrame or Series using a mapper or by a Series of columns.
gt(other[, axis, level])	Greater than of dataframe and other, element-wise (binary operator *gt*).
gui([params])	Construct a graphical user interface for viewing samples.
head([n])	Return the first *n* rows.
hist(*args, **kwargs)	Weighted histogram of the sampled distribution.
idxmax([axis, skipna])	Return index of first occurrence of maximum over requested axis.
idxmin([axis, skipna])	Return index of first occurrence of minimum over requested axis.
infer_objects()	Attempt to infer better dtypes for object columns.
info([verbose, buf, max_cols, memory_usage, ...])	Print a concise summary of a DataFrame.

Continued on next page

Table 10 – continued from previous page

insert(loc, column, value[, allow_duplicates])	Insert column into DataFrame at specified location.
interpolate([method, axis, limit, inplace, ...])	Interpolate values according to different methods.
isin(values)	Whether each element in the DataFrame is contained in values.
isna()	Detect missing values.
isnull()	Detect missing values.
items()	Iterator over (column name, Series) pairs.
iteritems()	Iterator over (column name, Series) pairs.
iterrows()	Iterate over DataFrame rows as (index, Series) pairs.
itertuples([index, name])	Iterate over DataFrame rows as namedtuples.
join(other[, on, how, lsuffix, rsuffix, sort])	Join columns of another DataFrame.
keys()	Get the 'info axis' (see Indexing for more)
kurt([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).
kurtosis([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).
last(offset)	Convenience method for subsetting final periods of time series data based on a date offset.
last_valid_index()	Return index for last non-NA/null value.
le(other[, axis, level])	Less than or equal to of dataframe and other, element-wise (binary operator *le*).
live_points(logL)	Get the live points within logL.
logZ([nsamples])	Log-Evidence.
lookup(row_labels, col_labels)	Label-based "fancy indexing" function for DataFrame.
lt(other[, axis, level])	Less than of dataframe and other, element-wise (binary operator *lt*).
mad([axis, skipna, level])	Return the mean absolute deviation of the values for the requested axis.
mask(cond[, other, inplace, axis, level, ...])	Replace values where the condition is True.
max([axis, skipna, level, numeric_only])	Return the maximum of the values for the requested axis.
mean()	Weighted mean of the sampled distribution.
median()	Weighted median of the sampled distribution.
melt([id_vars, value_vars, var_name, ...])	Unpivots a DataFrame from wide format to long format, optionally leaving identifier variables set.
memory_usage([index, deep])	Return the memory usage of each column in bytes.
merge(right[, how, on, left_on, right_on, ...])	Merge DataFrame or named Series objects with a database-style join.
min([axis, skipna, level, numeric_only])	Return the minimum of the values for the requested axis.
mod(other[, axis, level, fill_value])	Modulo of dataframe and other, element-wise (binary operator *mod*).
mode([axis, numeric_only, dropna])	Get the mode(s) of each element along the selected axis.
mul(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *mul*).
multiply(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *mul*).

Continued on next page

Table 10 – continued from previous page

ne(other[, axis, level])	Not equal to of dataframe and other, element-wise (binary operator *ne*).
neff()	Effective number of samples.
nlargest(n, columns[, keep])	Return the first *n* rows ordered by *columns* in descending order.
notna()	Detect existing (non-missing) values.
notnull()	Detect existing (non-missing) values.
ns_output([nsamples])	Compute Bayesian global quantities.
nsmallest(n, columns[, keep])	Return the first *n* rows ordered by *columns* in ascending order.
nunique([axis, dropna])	Count distinct observations over requested axis.
pct_change([periods, fill_method, limit, freq])	Percentage change between the current and a prior element.
pipe(func, *args, **kwargs)	Apply func(self, *args, **kwargs).
pivot([index, columns, values])	Return reshaped DataFrame organized by given index / column values.
pivot_table([values, index, columns, ...])	Create a spreadsheet-style pivot table as a DataFrame.
plot(ax, paramname_x[, paramname_y])	Interface for 2D and 1D plotting routines.
plot_1d(axes, *args, **kwargs)	Create an array of 1D plots.
plot_2d(axes, *args, **kwargs)	Create an array of 2D plots.
pop(item)	Return item and drop from frame.
posterior_points(beta)	Get the posterior points at temperature beta.
pow(other[, axis, level, fill_value])	Exponential power of dataframe and other, element-wise (binary operator *pow*).
prod([axis, skipna, level, numeric_only, ...])	Return the product of the values for the requested axis.
product([axis, skipna, level, numeric_only, ...])	Return the product of the values for the requested axis.
quantile([q])	Weighted quantile of the sampled distribution.
query(expr[, inplace])	Query the columns of a DataFrame with a boolean expression.
radd(other[, axis, level, fill_value])	Addition of dataframe and other, element-wise (binary operator *radd*).
rank([axis, method, numeric_only, ...])	Compute numerical data ranks (1 through n) along axis.
rdiv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *rtruediv*).
reindex([labels, index, columns, axis, ...])	Conform DataFrame to new index with optional filling logic, placing NA/NaN in locations having no value in the previous index.
reindex_axis(labels[, axis, method, level, ...])	Conform input object to new index.
reindex_like(other[, method, copy, limit, ...])	Return an object with matching indices as other object.
rename([mapper, index, columns, axis, copy, ...])	Alter axes labels.
rename_axis([mapper, index, columns, axis, ...])	Set the name of the axis for the index or columns.
reorder_levels(order[, axis])	Rearrange index levels using input order.
replace([to_replace, value, inplace, limit, ...])	Replace values given in *to_replace* with *value*.
resample(rule[, how, axis, fill_method, ...])	Resample time-series data.
reset_index([level, drop, inplace, ...])	Reset the index, or a level of it.

Continued on next page

Table 10 – continued from previous page

rfloordiv(other[, axis, level, fill_value])	Integer division of dataframe and other, element-wise (binary operator *rfloordiv*).
rmod(other[, axis, level, fill_value])	Modulo of dataframe and other, element-wise (binary operator *rmod*).
rmul(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *rmul*).
rolling(window[, min_periods, center, ...])	Provides rolling window calculations.
round([decimals])	Round a DataFrame to a variable number of decimal places.
rpow(other[, axis, level, fill_value])	Exponential power of dataframe and other, element-wise (binary operator *rpow*).
rsub(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *rsub*).
rtruediv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *rtruediv*).
sample([n, frac, replace, weights, ...])	Return a random sample of items from an axis of object.
select(crit[, axis])	Return data corresponding to axis labels matching criteria.
select_dtypes([include, exclude])	Return a subset of the DataFrame's columns based on the column dtypes.
sem([axis, skipna, level, ddof, numeric_only])	Return unbiased standard error of the mean over requested axis.
set_axis(labels[, axis, inplace])	Assign desired index to given axis.
set_beta(beta[, inplace])	Change the inverse temperature.
set_index(keys[, drop, append, inplace, ...])	Set the DataFrame index using existing columns.
set_value(index, col, value[, takeable])	Put single value at passed column and index.
shift([periods, freq, axis, fill_value])	Shift index by desired number of periods with an optional time *freq*.
skew([axis, skipna, level, numeric_only])	Return unbiased skew over requested axis Normalized by N-1.
slice_shift([periods, axis])	Equivalent to *shift* without copying data.
sort_index([axis, level, ascending, ...])	Sort object by labels (along an axis)
sort_values(by[, axis, ascending, inplace, ...])	Sort by the values along either axis
squeeze([axis])	Squeeze 1 dimensional axis objects into scalars.
stack([level, dropna])	Stack the prescribed level(s) from columns to index.
std()	Weighted standard deviation of the sampled distribution.
sub(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *sub*).
subtract(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *sub*).
sum([axis, skipna, level, numeric_only, ...])	Return the sum of the values for the requested axis.
swapaxes(axis1, axis2[, copy])	Interchange axes and swap values axes appropriately.
swaplevel([i, j, axis])	Swap levels i and j in a MultiIndex on a particular axis.
tail([n])	Return the last *n* rows.
take(indices[, axis, convert, is_copy])	Return the elements in the given *positional* indices along an axis.
to_clipboard([excel, sep])	Copy object to the system clipboard.
to_csv([path_or_buf, sep, na_rep, ...])	Write object to a comma-separated values (csv) file.

Continued on next page

Table 10 – continued from previous page

to_dense()	Return dense representation of NDFrame (as opposed to sparse).
to_dict([orient, into])	Convert the DataFrame to a dictionary.
to_excel(excel_writer[, sheet_name, na_rep, ...])	Write object to an Excel sheet.
to_feather(fname)	Write out the binary feather-format for DataFrames.
to_gbq(destination_table[, project_id, ...])	Write a DataFrame to a Google BigQuery table.
to_hdf(path_or_buf, key, **kwargs)	Write the contained data to an HDF5 file using HDFStore.
to_html([buf, columns, col_space, header, ...])	Render a DataFrame as an HTML table.
to_json([path_or_buf, orient, date_format, ...])	Convert the object to a JSON string.
to_latex([buf, columns, col_space, header, ...])	Render an object to a LaTeX tabular environment table.
to_msgpack([path_or_buf, encoding])	Serialize object to input file path using msgpack format.
to_numpy([dtype, copy])	Convert the DataFrame to a NumPy array.
to_panel()	Transform long (stacked) format (DataFrame) into wide (3D, Panel) format.
to_parquet(fname[, engine, compression, ...])	Write a DataFrame to the binary parquet format.
to_period([freq, axis, copy])	Convert DataFrame from DatetimeIndex to PeriodIndex with desired frequency (inferred from index if not passed).
to_pickle(path[, compression, protocol])	Pickle (serialize) object to file.
to_records([index, convert_datetime64, ...])	Convert DataFrame to a NumPy record array.
to_sparse([fill_value, kind])	Convert to SparseDataFrame.
to_sql(name, con[, schema, if_exists, ...])	Write records stored in a DataFrame to a SQL database.
to_stata(fname[, convert_dates, ...])	Export DataFrame object to Stata dta format.
to_string([buf, columns, col_space, header, ...])	Render a DataFrame to a console-friendly tabular output.
to_timestamp([freq, how, axis, copy])	Cast to DatetimeIndex of timestamps, at *beginning* of period.
to_xarray()	Return an xarray object from the pandas object.
transform(func[, axis])	Call `func` on self producing a DataFrame with transformed values and that has the same axis length as self.
transpose(*args, **kwargs)	Transpose index and columns.
truediv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
truncate([before, after, axis, copy])	Truncate a Series or DataFrame before and after some index value.
tshift([periods, freq, axis])	Shift the time index, using the index's frequency if available.
tz_convert(tz[, axis, level, copy])	Convert tz-aware axis to target time zone.
tz_localize(tz[, axis, level, copy, ...])	Localize tz-naive index of a Series or DataFrame to target time zone.
unstack([level, fill_value])	Pivot a level of the (necessarily hierarchical) index labels, returning a DataFrame having a new level of column labels whose inner-most level consists of the pivoted index labels.

Continued on next page

Table 10 – continued from previous page

update(other[, join, overwrite, ...])	Modify in place using non-NA values from another DataFrame.
var()	Weighted variance of the sampled distribution.
where(cond[, other, inplace, axis, level, ...])	Replace values where the condition is False.
xs(key[, axis, level, drop_level])	Return cross-section from the Series/DataFrame.

D(*nsamples=None*)
Kullback-Leibler divergence.

- If nsamples is not supplied, return mean KL divergence
- If nsamples is integer, return nsamples from the distribution
- If nsamples is array, use nsamples as volumes of evidence shells

beta
Thermodynamic inverse temperature.

d(*nsamples=None*)
Bayesian model dimensionality.

- If nsamples is not supplied, return mean BMD
- If nsamples is integer, return nsamples from the distribution
- If nsamples is array, use nsamples as volumes of evidence shells

dlogX(*nsamples=None*)
Compute volume of shell of loglikelihood.

Parameters

nsamples: int, optional Number of samples to generate. optional. If None, then compute the statistical average. If integer, generate samples from the distribution. (Default: None)

gui(*params=None*)
Construct a graphical user interface for viewing samples.

live_points(*logL*)
Get the live points within logL.

logZ(*nsamples=None*)
Log-Evidence.

- If nsamples is not supplied, return mean log evidence
- If nsamples is integer, return nsamples from the distribution
- If nsamples is array, use nsamples as volumes of evidence shells

ns_output(*nsamples=200*)
Compute Bayesian global quantities.

Using nested sampling we can compute the evidence (logZ), Kullback-Leibler divergence (D) and Bayesian model dimensionality (d). More precisely, we can infer these quantities via their probability distribution.

Parameters

nsamples: int, optional number of samples to generate (Default: 100)

Returns

pandas.DataFrame Samples from the P(logZ, D, d) distribution

posterior_points(*beta*)
Get the posterior points at temperature beta.

set_beta(*beta*, *inplace=False*)
Change the inverse temperature.

Parameters

beta: float Temperature to set

inplace: bool, optional Indicates whether to modify the existing array, or return a copy with the temperature changed. Default: False

anesthetic.samples.**merge_nested_samples**(*runs*)
Merge two or more nested sampling runs.

Parameters

runs: list(NestedSamples) list or array-like of nested sampling runs.

Returns

samples: NestedSamples Merged run.

7.6 anesthetic.utils module

Data-processing utility functions.

anesthetic.utils.**channel_capacity**(*w*)
Channel capacity (effective sample size).

$$H = \sum_i p_i \log p_i$$
$$p_i = \frac{w_i}{\sum_j w_j}$$
$$N = e^{-H}$$

anesthetic.utils.**check_bounds**(*d*, *xmin=None*, *xmax=None*)
Check if we need to apply strict bounds.

anesthetic.utils.**compress_weights**(*w*, *u=None*, *nsamples=None*)
Compresses weights to their approximate channel capacity.

anesthetic.utils.**compute_nlive**(*death*, *birth*)
Compute number of live points from birth and death contours.

anesthetic.utils.**histogram**(*a*, ***kwargs*)
Produce a histogram for path-based plotting.

This is a cheap histogram. Necessary if one wants to update the histogram dynamically, and redrawing and filling is very expensive.

This has the same arguments and keywords as numpy.histogram, but is normalised to 1.

anesthetic.utils.**iso_probability_contours**(*pdf, contours=[0.68, 0.95]*)
Compute the iso-probability contour values.

anesthetic.utils.**iso_probability_contours_from_samples**(*pdf*, *contours=[0.68, 0.95]*, *weights=None*)
Compute the iso-probability contour values.

anesthetic.utils.**mirror_1d**(*d*, *xmin=None*, *xmax=None*)
If necessary apply reflecting boundary conditions.

anesthetic.utils.**mirror_2d**(*d_x_*, *d_y_*, *xmin=None*, *xmax=None*, *ymin=None*, *ymax=None*)
If necessary apply reflecting boundary conditions.

anesthetic.utils.**nest_level**(*lst*)
Calculate the nesting level of a list.

anesthetic.utils.**quantile**(*a*, *q*, *w=None*)
Compute the weighted quantile for a one dimensional array.

anesthetic.utils.**sample_compression_1d**(*x*, *w=None*, *n=1000*)
Histogram a 1D set of weighted samples via subsampling.

This compresses the number of samples, combining weights.

Parameters

x: array-like x coordinate of samples for compressing

w: pandas.Series, optional weights of samples

n: int, optional number of samples returned. Default 1000

Returns

x, w, array-like Compressed samples and weights

anesthetic.utils.**triangular_sample_compression_2d**(*x*, *y*, *w=None*, *n=1000*)
Histogram a 2D set of weighted samples via triangulation.

This defines bins via a triangulation of the subsamples, sums weights within triangles, and computes weighted centroids of triangles.

Parameters

x, y: array-like x and y coordinates of samples for compressing

w: pandas.Series, optional weights of samples

n: int, optional number of samples returned. Default 1000

Returns

x, y, w, array-like Compressed samples and weights

anesthetic.utils.**unique**(*a*)
Find unique elements, retaining order.

7.7 anesthetic.weighted_pandas module

Pandas DataFrame and Series with weighted samples.

class anesthetic.weighted_pandas.**WeightedDataFrame**(**args*, ***kwargs*)
Bases: anesthetic.weighted_pandas._WeightedObject, pandas.core.frame.DataFrame

Weighted version of pandas.DataFrame.

Attributes

T Transpose index and columns.

at Access a single value for a row/column label pair.

axes Return a list representing the axes of the DataFrame.

blocks Internal property, property synonym for as_blocks().

columns The column labels of the DataFrame.

dtypes Return the dtypes in the DataFrame.

empty Indicator whether DataFrame is empty.

ftypes Return the ftypes (indication of sparse/dense and dtype) in DataFrame.

iat Access a single value for a row/column pair by integer position.

iloc Purely integer-location based indexing for selection by position.

index The index (row labels) of the DataFrame.

is_copy Return the copy.

ix A primarily label-location based indexer, with integer position fallback.

loc Access a group of rows and columns by label(s) or a boolean array.

ndim Return an int representing the number of axes / array dimensions.

shape Return a tuple representing the dimensionality of the DataFrame.

size Return an int representing the number of elements in this object.

style Property returning a Styler object containing methods for building a styled HTML representation fo the DataFrame.

timetuple

values Return a Numpy representation of the DataFrame.

weight Sample weights.

Methods

abs()	Return a Series/DataFrame with absolute numeric value of each element.
add(other[, axis, level, fill_value])	Addition of dataframe and other, element-wise (binary operator *add*).
add_prefix(prefix)	Prefix labels with string *prefix*.
add_suffix(suffix)	Suffix labels with string *suffix*.
agg(func[, axis])	Aggregate using one or more operations over the specified axis.
aggregate(func[, axis])	Aggregate using one or more operations over the specified axis.
align(other[, join, axis, level, copy, ...])	Align two objects on their axes with the specified join method for each axis Index.
all([axis, bool_only, skipna, level])	Return whether all elements are True, potentially over an axis.

Continued on next page

Table 11 – continued from previous page

any([axis, bool_only, skipna, level])	Return whether any element is True, potentially over an axis.
append(other[, ignore_index, ...])	Append rows of *other* to the end of caller, returning a new object.
apply(func[, axis, broadcast, raw, reduce, ...])	Apply a function along an axis of the DataFrame.
applymap(func)	Apply a function to a Dataframe elementwise.
as_blocks([copy])	Convert the frame to a dict of dtype -> Constructor Types that each has a homogeneous dtype.
as_matrix([columns])	Convert the frame to its Numpy-array representation.
asfreq(freq[, method, how, normalize, ...])	Convert TimeSeries to specified frequency.
asof(where[, subset])	Return the last row(s) without any NaNs before *where*.
assign(**kwargs)	Assign new columns to a DataFrame.
astype(dtype[, copy, errors])	Cast a pandas object to a specified dtype `dtype`.
at_time(time[, asof, axis])	Select values at particular time of day (e.g.
between_time(start_time, end_time[, ...])	Select values between particular times of the day (e.g., 9:00-9:30 AM).
bfill([axis, inplace, limit, downcast])	Synonym for `DataFrame.fillna()` with `method='bfill'`.
bool()	Return the bool of a single element PandasObject.
boxplot([column, by, ax, fontsize, rot, ...])	Make a box plot from DataFrame columns.
clip([lower, upper, axis, inplace])	Trim values at input threshold(s).
clip_lower(threshold[, axis, inplace])	Trim values below a given threshold.
clip_upper(threshold[, axis, inplace])	Trim values above a given threshold.
combine(other, func[, fill_value, overwrite])	Perform column-wise combine with another DataFrame based on a passed function.
combine_first(other)	Update null elements with value in the same location in *other*.
compound([axis, skipna, level])	Return the compound percentage of the values for the requested axis.
compress([nsamples])	Reduce the number of samples by discarding low-weights.
convert_objects([convert_dates, ...])	Attempt to infer better dtype for object columns.
copy([deep])	Make a copy of this object's indices and data.
corr([method, min_periods])	Compute pairwise correlation of columns, excluding NA/null values.
corrwith(other[, axis, drop, method])	Compute pairwise correlation between rows or columns of DataFrame with rows or columns of Series or DataFrame.
count([axis, level, numeric_only])	Count non-NA cells for each column or row.
cov()	Weighted covariance of the sampled distribution.
cummax([axis, skipna])	Return cumulative maximum over a DataFrame or Series axis.
cummin([axis, skipna])	Return cumulative minimum over a DataFrame or Series axis.
cumprod([axis, skipna])	Return cumulative product over a DataFrame or Series axis.
cumsum([axis, skipna])	Return cumulative sum over a DataFrame or Series axis.

Continued on next page

Table 11 – continued from previous page

describe([percentiles, include, exclude])	Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
diff([periods, axis])	First discrete difference of element.
div(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
divide(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
dot(other)	Compute the matrix mutiplication between the DataFrame and other.
drop([labels, axis, index, columns, level, ...])	Drop specified labels from rows or columns.
drop_duplicates([subset, keep, inplace])	Return DataFrame with duplicate rows removed, optionally only considering certain columns.
droplevel(level[, axis])	Return DataFrame with requested index / column level(s) removed.
dropna([axis, how, thresh, subset, inplace])	Remove missing values.
duplicated([subset, keep])	Return boolean Series denoting duplicate rows, optionally only considering certain columns.
eq(other[, axis, level])	Equal to of dataframe and other, element-wise (binary operator *eq*).
equals(other)	Test whether two objects contain the same elements.
eval(expr[, inplace])	Evaluate a string describing operations on DataFrame columns.
ewm([com, span, halflife, alpha, ...])	Provides exponential weighted functions.
expanding([min_periods, center, axis])	Provides expanding transformations.
ffill([axis, inplace, limit, downcast])	Synonym for DataFrame.fillna() with method='ffill'.
fillna([value, method, axis, inplace, ...])	Fill NA/NaN values using the specified method.
filter([items, like, regex, axis])	Subset rows or columns of dataframe according to labels in the specified index.
first(offset)	Convenience method for subsetting initial periods of time series data based on a date offset.
first_valid_index()	Return index for first non-NA/null value.
floordiv(other[, axis, level, fill_value])	Integer division of dataframe and other, element-wise (binary operator *floordiv*).
from_csv(path[, header, sep, index_col, ...])	Read CSV file.
from_dict(data[, orient, dtype, columns])	Construct DataFrame from dict of array-like or dicts.
from_items(items[, columns, orient])	Construct a DataFrame from a list of tuples.
from_records(data[, index, exclude, ...])	Convert structured or record ndarray to DataFrame.
ge(other[, axis, level])	Greater than or equal to of dataframe and other, element-wise (binary operator *ge*).
get(key[, default])	Get item from object for given key (DataFrame column, Panel slice, etc.).
get_dtype_counts()	Return counts of unique dtypes in this object.
get_ftype_counts()	Return counts of unique ftypes in this object.
get_value(index, col[, takeable])	Quickly retrieve single value at passed column and index.
get_values()	Return an ndarray after converting sparse values to dense.
groupby([by, axis, level, as_index, sort, ...])	Group DataFrame or Series using a mapper or by a Series of columns.

Continued on next page

Table 11 – continued from previous page

gt(other[, axis, level])	Greater than of dataframe and other, element-wise (binary operator *gt*).
head([n])	Return the first *n* rows.
hist(*args, **kwargs)	Weighted histogram of the sampled distribution.
idxmax([axis, skipna])	Return index of first occurrence of maximum over requested axis.
idxmin([axis, skipna])	Return index of first occurrence of minimum over requested axis.
infer_objects()	Attempt to infer better dtypes for object columns.
info([verbose, buf, max_cols, memory_usage, ...])	Print a concise summary of a DataFrame.
insert(loc, column, value[, allow_duplicates])	Insert column into DataFrame at specified location.
interpolate([method, axis, limit, inplace, ...])	Interpolate values according to different methods.
isin(values)	Whether each element in the DataFrame is contained in values.
isna()	Detect missing values.
isnull()	Detect missing values.
items()	Iterator over (column name, Series) pairs.
iteritems()	Iterator over (column name, Series) pairs.
iterrows()	Iterate over DataFrame rows as (index, Series) pairs.
itertuples([index, name])	Iterate over DataFrame rows as namedtuples.
join(other[, on, how, lsuffix, rsuffix, sort])	Join columns of another DataFrame.
keys()	Get the 'info axis' (see Indexing for more)
kurt([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).
kurtosis([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).
last(offset)	Convenience method for subsetting final periods of time series data based on a date offset.
last_valid_index()	Return index for last non-NA/null value.
le(other[, axis, level])	Less than or equal to of dataframe and other, element-wise (binary operator *le*).
lookup(row_labels, col_labels)	Label-based "fancy indexing" function for DataFrame.
lt(other[, axis, level])	Less than of dataframe and other, element-wise (binary operator *lt*).
mad([axis, skipna, level])	Return the mean absolute deviation of the values for the requested axis.
mask(cond[, other, inplace, axis, level, ...])	Replace values where the condition is True.
max([axis, skipna, level, numeric_only])	Return the maximum of the values for the requested axis.
mean()	Weighted mean of the sampled distribution.
median()	Weighted median of the sampled distribution.
melt([id_vars, value_vars, var_name, ...])	Unpivots a DataFrame from wide format to long format, optionally leaving identifier variables set.
memory_usage([index, deep])	Return the memory usage of each column in bytes.
merge(right[, how, on, left_on, right_on, ...])	Merge DataFrame or named Series objects with a database-style join.

Continued on next page

Table 11 – continued from previous page

min([axis, skipna, level, numeric_only])	Return the minimum of the values for the requested axis.
mod(other[, axis, level, fill_value])	Modulo of dataframe and other, element-wise (binary operator *mod*).
mode([axis, numeric_only, dropna])	Get the mode(s) of each element along the selected axis.
mul(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *mul*).
multiply(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *mul*).
ne(other[, axis, level])	Not equal to of dataframe and other, element-wise (binary operator *ne*).
neff()	Effective number of samples.
nlargest(n, columns[, keep])	Return the first *n* rows ordered by *columns* in descending order.
notna()	Detect existing (non-missing) values.
notnull()	Detect existing (non-missing) values.
nsmallest(n, columns[, keep])	Return the first *n* rows ordered by *columns* in ascending order.
nunique([axis, dropna])	Count distinct observations over requested axis.
pct_change([periods, fill_method, limit, freq])	Percentage change between the current and a prior element.
pipe(func, *args, **kwargs)	Apply func(self, *args, **kwargs).
pivot([index, columns, values])	Return reshaped DataFrame organized by given index / column values.
pivot_table([values, index, columns, ...])	Create a spreadsheet-style pivot table as a DataFrame.
plot	alias of pandas.plotting._core.FramePlotMethods
pop(item)	Return item and drop from frame.
pow(other[, axis, level, fill_value])	Exponential power of dataframe and other, element-wise (binary operator *pow*).
prod([axis, skipna, level, numeric_only, ...])	Return the product of the values for the requested axis.
product([axis, skipna, level, numeric_only, ...])	Return the product of the values for the requested axis.
quantile([q])	Weighted quantile of the sampled distribution.
query(expr[, inplace])	Query the columns of a DataFrame with a boolean expression.
radd(other[, axis, level, fill_value])	Addition of dataframe and other, element-wise (binary operator *radd*).
rank([axis, method, numeric_only, ...])	Compute numerical data ranks (1 through n) along axis.
rdiv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *rtruediv*).
reindex([labels, index, columns, axis, ...])	Conform DataFrame to new index with optional filling logic, placing NA/NaN in locations having no value in the previous index.
reindex_axis(labels[, axis, method, level, ...])	Conform input object to new index.
reindex_like(other[, method, copy, limit, ...])	Return an object with matching indices as other object.

Continued on next page

Table 11 – continued from previous page

rename([mapper, index, columns, axis, copy, ...])	Alter axes labels.
rename_axis([mapper, index, columns, axis, ...])	Set the name of the axis for the index or columns.
reorder_levels(order[, axis])	Rearrange index levels using input order.
replace([to_replace, value, inplace, limit, ...])	Replace values given in *to_replace* with *value*.
resample(rule[, how, axis, fill_method, ...])	Resample time-series data.
reset_index([level, drop, inplace, ...])	Reset the index, or a level of it.
rfloordiv(other[, axis, level, fill_value])	Integer division of dataframe and other, element-wise (binary operator *rfloordiv*).
rmod(other[, axis, level, fill_value])	Modulo of dataframe and other, element-wise (binary operator *rmod*).
rmul(other[, axis, level, fill_value])	Multiplication of dataframe and other, element-wise (binary operator *rmul*).
rolling(window[, min_periods, center, ...])	Provides rolling window calculations.
round([decimals])	Round a DataFrame to a variable number of decimal places.
rpow(other[, axis, level, fill_value])	Exponential power of dataframe and other, element-wise (binary operator *rpow*).
rsub(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *rsub*).
rtruediv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *rtruediv*).
sample([n, frac, replace, weights, ...])	Return a random sample of items from an axis of object.
select(crit[, axis])	Return data corresponding to axis labels matching criteria.
select_dtypes([include, exclude])	Return a subset of the DataFrame's columns based on the column dtypes.
sem([axis, skipna, level, ddof, numeric_only])	Return unbiased standard error of the mean over requested axis.
set_axis(labels[, axis, inplace])	Assign desired index to given axis.
set_index(keys[, drop, append, inplace, ...])	Set the DataFrame index using existing columns.
set_value(index, col, value[, takeable])	Put single value at passed column and index.
shift([periods, freq, axis, fill_value])	Shift index by desired number of periods with an optional time *freq*.
skew([axis, skipna, level, numeric_only])	Return unbiased skew over requested axis Normalized by N-1.
slice_shift([periods, axis])	Equivalent to *shift* without copying data.
sort_index([axis, level, ascending, ...])	Sort object by labels (along an axis)
sort_values(by[, axis, ascending, inplace, ...])	Sort by the values along either axis
squeeze([axis])	Squeeze 1 dimensional axis objects into scalars.
stack([level, dropna])	Stack the prescribed level(s) from columns to index.
std()	Weighted standard deviation of the sampled distribution.
sub(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *sub*).
subtract(other[, axis, level, fill_value])	Subtraction of dataframe and other, element-wise (binary operator *sub*).
sum([axis, skipna, level, numeric_only, ...])	Return the sum of the values for the requested axis.
swapaxes(axis1, axis2[, copy])	Interchange axes and swap values axes appropriately.

Continued on next page

Table 11 – continued from previous page

swaplevel([i, j, axis])	Swap levels i and j in a MultiIndex on a particular axis.
tail([n])	Return the last *n* rows.
take(indices[, axis, convert, is_copy])	Return the elements in the given *positional* indices along an axis.
to_clipboard([excel, sep])	Copy object to the system clipboard.
to_csv([path_or_buf, sep, na_rep, ...])	Write object to a comma-separated values (csv) file.
to_dense()	Return dense representation of NDFrame (as opposed to sparse).
to_dict([orient, into])	Convert the DataFrame to a dictionary.
to_excel(excel_writer[, sheet_name, na_rep, ...])	Write object to an Excel sheet.
to_feather(fname)	Write out the binary feather-format for DataFrames.
to_gbq(destination_table[, project_id, ...])	Write a DataFrame to a Google BigQuery table.
to_hdf(path_or_buf, key, **kwargs)	Write the contained data to an HDF5 file using HDFStore.
to_html([buf, columns, col_space, header, ...])	Render a DataFrame as an HTML table.
to_json([path_or_buf, orient, date_format, ...])	Convert the object to a JSON string.
to_latex([buf, columns, col_space, header, ...])	Render an object to a LaTeX tabular environment table.
to_msgpack([path_or_buf, encoding])	Serialize object to input file path using msgpack format.
to_numpy([dtype, copy])	Convert the DataFrame to a NumPy array.
to_panel()	Transform long (stacked) format (DataFrame) into wide (3D, Panel) format.
to_parquet(fname[, engine, compression, ...])	Write a DataFrame to the binary parquet format.
to_period([freq, axis, copy])	Convert DataFrame from DatetimeIndex to PeriodIndex with desired frequency (inferred from index if not passed).
to_pickle(path[, compression, protocol])	Pickle (serialize) object to file.
to_records([index, convert_datetime64, ...])	Convert DataFrame to a NumPy record array.
to_sparse([fill_value, kind])	Convert to SparseDataFrame.
to_sql(name, con[, schema, if_exists, ...])	Write records stored in a DataFrame to a SQL database.
to_stata(fname[, convert_dates, ...])	Export DataFrame object to Stata dta format.
to_string([buf, columns, col_space, header, ...])	Render a DataFrame to a console-friendly tabular output.
to_timestamp([freq, how, axis, copy])	Cast to DatetimeIndex of timestamps, at *beginning* of period.
to_xarray()	Return an xarray object from the pandas object.
transform(func[, axis])	Call func on self producing a DataFrame with transformed values and that has the same axis length as self.
transpose(*args, **kwargs)	Transpose index and columns.
truediv(other[, axis, level, fill_value])	Floating division of dataframe and other, element-wise (binary operator *truediv*).
truncate([before, after, axis, copy])	Truncate a Series or DataFrame before and after some index value.
tshift([periods, freq, axis])	Shift the time index, using the index's frequency if available.
tz_convert(tz[, axis, level, copy])	Convert tz-aware axis to target time zone.

Continued on next page

Table 11 – continued from previous page

tz_localize(tz[, axis, level, copy, ...])	Localize tz-naive index of a Series or DataFrame to target time zone.
unstack([level, fill_value])	Pivot a level of the (necessarily hierarchical) index labels, returning a DataFrame having a new level of column labels whose inner-most level consists of the pivoted index labels.
update(other[, join, overwrite, ...])	Modify in place using non-NA values from another DataFrame.
var()	Weighted variance of the sampled distribution.
where(cond[, other, inplace, axis, level, ...])	Replace values where the condition is False.
xs(key[, axis, level, drop_level])	Return cross-section from the Series/DataFrame.

compress(*nsamples=None*)
Reduce the number of samples by discarding low-weights.

Parameters

neff: int, optional effective number of samples after compression. If not supplied, then reduce to the channel capacity (theoretical optimum compression). If <=0, then compress so that all weights are unity.

cov()
Weighted covariance of the sampled distribution.

hist(**args*, ***kwargs*)
Weighted histogram of the sampled distribution.

mean()
Weighted mean of the sampled distribution.

quantile(*q=0.5*)
Weighted quantile of the sampled distribution.

var()
Weighted variance of the sampled distribution.

class anesthetic.weighted_pandas.**WeightedSeries**(**args*, ***kwargs*)
Bases: anesthetic.weighted_pandas._WeightedObject, pandas.core.series.Series

Weighted version of pandas.Series.

Attributes

T Return the transpose, which is by definition self.

array The ExtensionArray of the data backing this Series or Index.

asobject Return object Series which contains boxed values.

at Access a single value for a row/column label pair.

axes Return a list of the row axis labels.

base Return the base object if the memory of the underlying data is shared.

blocks Internal property, property synonym for as_blocks().

data Return the data pointer of the underlying data.

dtype Return the dtype object of the underlying data.

dtypes Return the dtype object of the underlying data.

empty

flags Return the ndarray.flags for the underlying data.

ftype Return if the data is sparse|dense.

ftypes Return if the data is sparse|dense.

hasnans Return if I have any nans; enables various perf speedups.

iat Access a single value for a row/column pair by integer position.

iloc Purely integer-location based indexing for selection by position.

imag Return imag value of vector.

index The index (axis labels) of the Series.

is_copy Return the copy.

is_monotonic Return boolean if values in the object are monotonic_increasing.

is_monotonic_decreasing Return boolean if values in the object are monotonic_decreasing.

is_monotonic_increasing Return boolean if values in the object are monotonic_increasing.

is_unique Return boolean if values in the object are unique.

itemsize Return the size of the dtype of the item of the underlying data.

ix A primarily label-location based indexer, with integer position fallback.

loc Access a group of rows and columns by label(s) or a boolean array.

name Return name of the Series.

nbytes Return the number of bytes in the underlying data.

ndim Number of dimensions of the underlying data, by definition 1.

real Return the real value of vector.

shape Return a tuple of the shape of the underlying data.

size Return the number of elements in the underlying data.

strides Return the strides of the underlying data.

timetuple

values Return Series as ndarray or ndarray-like depending on the dtype.

weight Sample weights.

Methods

abs()	Return a Series/DataFrame with absolute numeric value of each element.
add(other[, level, fill_value, axis])	Addition of series and other, element-wise (binary operator *add*).
add_prefix(prefix)	Prefix labels with string *prefix*.
add_suffix(suffix)	Suffix labels with string *suffix*.

Continued on next page

Table 12 – continued from previous page

agg(func[, axis])	Aggregate using one or more operations over the specified axis.
aggregate(func[, axis])	Aggregate using one or more operations over the specified axis.
align(other[, join, axis, level, copy, ...])	Align two objects on their axes with the specified join method for each axis Index.
all([axis, bool_only, skipna, level])	Return whether all elements are True, potentially over an axis.
any([axis, bool_only, skipna, level])	Return whether any element is True, potentially over an axis.
append(to_append[, ignore_index, ...])	Concatenate two or more Series.
apply(func[, convert_dtype, args])	Invoke function on values of Series.
argmax([axis, skipna])	Return the row label of the maximum value.
argmin([axis, skipna])	Return the row label of the minimum value.
argsort([axis, kind, order])	Overrides ndarray.argsort.
as_blocks([copy])	Convert the frame to a dict of dtype -> Constructor Types that each has a homogeneous dtype.
as_matrix([columns])	Convert the frame to its Numpy-array representation.
asfreq(freq[, method, how, normalize, ...])	Convert TimeSeries to specified frequency.
asof(where[, subset])	Return the last row(s) without any NaNs before *where*.
astype(dtype[, copy, errors])	Cast a pandas object to a specified dtype `dtype`.
at_time(time[, asof, axis])	Select values at particular time of day (e.g.
autocorr([lag])	Compute the lag-N autocorrelation.
between(left, right[, inclusive])	Return boolean Series equivalent to left <= series <= right.
between_time(start_time, end_time[, ...])	Select values between particular times of the day (e.g., 9:00-9:30 AM).
bfill([axis, inplace, limit, downcast])	Synonym for `DataFrame.fillna()` with `method='bfill'`.
bool()	Return the bool of a single element PandasObject.
cat	alias of `pandas.core.arrays.categorical.CategoricalAccessor`
clip([lower, upper, axis, inplace])	Trim values at input threshold(s).
clip_lower(threshold[, axis, inplace])	Trim values below a given threshold.
clip_upper(threshold[, axis, inplace])	Trim values above a given threshold.
combine(other, func[, fill_value])	Combine the Series with a Series or scalar according to *func*.
combine_first(other)	Combine Series values, choosing the calling Series's values first.
compound([axis, skipna, level])	Return the compound percentage of the values for the requested axis.
compress([nsamples])	Reduce the number of samples by discarding low-weights.
convert_objects([convert_dates, ...])	Attempt to infer better dtype for object columns.
copy([deep])	Make a copy of this object's indices and data.
corr(other[, method, min_periods])	Compute correlation with *other* Series, excluding missing values.
count([level])	Return number of non-NA/null observations in the Series.

Continued on next page

Table 12 – continued from previous page

cov(other[, min_periods])	Compute covariance with Series, excluding missing values.
cummax([axis, skipna])	Return cumulative maximum over a DataFrame or Series axis.
cummin([axis, skipna])	Return cumulative minimum over a DataFrame or Series axis.
cumprod([axis, skipna])	Return cumulative product over a DataFrame or Series axis.
cumsum([axis, skipna])	Return cumulative sum over a DataFrame or Series axis.
describe([percentiles, include, exclude])	Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
diff([periods])	First discrete difference of element.
div(other[, level, fill_value, axis])	Floating division of series and other, element-wise (binary operator *truediv*).
divide(other[, level, fill_value, axis])	Floating division of series and other, element-wise (binary operator *truediv*).
divmod(other[, level, fill_value, axis])	Integer division and modulo of series and other, element-wise (binary operator *divmod*).
dot(other)	Compute the dot product between the Series and the columns of other.
drop([labels, axis, index, columns, level, ...])	Return Series with specified index labels removed.
drop_duplicates([keep, inplace])	Return Series with duplicate values removed.
droplevel(level[, axis])	Return DataFrame with requested index / column level(s) removed.
dropna([axis, inplace])	Return a new Series with missing values removed.
dt	alias of pandas.core.indexes.accessors.CombinedDatetimelikeProperties
duplicated([keep])	Indicate duplicate Series values.
eq(other[, level, fill_value, axis])	Equal to of series and other, element-wise (binary operator *eq*).
equals(other)	Test whether two objects contain the same elements.
ewm([com, span, halflife, alpha, ...])	Provides exponential weighted functions.
expanding([min_periods, center, axis])	Provides expanding transformations.
factorize([sort, na_sentinel])	Encode the object as an enumerated type or categorical variable.
ffill([axis, inplace, limit, downcast])	Synonym for DataFrame.fillna() with method='ffill'.
fillna([value, method, axis, inplace, ...])	Fill NA/NaN values using the specified method.
filter([items, like, regex, axis])	Subset rows or columns of dataframe according to labels in the specified index.
first(offset)	Convenience method for subsetting initial periods of time series data based on a date offset.
first_valid_index()	Return index for first non-NA/null value.
floordiv(other[, level, fill_value, axis])	Integer division of series and other, element-wise (binary operator *floordiv*).
from_array(arr[, index, name, dtype, copy, ...])	Construct Series from array.
from_csv(path[, sep, parse_dates, header, ...])	Read CSV file.

Continued on next page

Table 12 – continued from previous page

ge(other[, level, fill_value, axis])	Greater than or equal to of series and other, element-wise (binary operator *ge*).
get(key[, default])	Get item from object for given key (DataFrame column, Panel slice, etc.).
get_dtype_counts()	Return counts of unique dtypes in this object.
get_ftype_counts()	Return counts of unique ftypes in this object.
get_value(label[, takeable])	Quickly retrieve single value at passed index label.
get_values()	Same as values (but handles sparseness conversions); is a view.
groupby([by, axis, level, as_index, sort, ...])	Group DataFrame or Series using a mapper or by a Series of columns.
gt(other[, level, fill_value, axis])	Greater than of series and other, element-wise (binary operator *gt*).
head([n])	Return the first *n* rows.
hist(*args, **kwargs)	Weighted histogram of the sampled distribution.
idxmax([axis, skipna])	Return the row label of the maximum value.
idxmin([axis, skipna])	Return the row label of the minimum value.
infer_objects()	Attempt to infer better dtypes for object columns.
interpolate([method, axis, limit, inplace, ...])	Interpolate values according to different methods.
isin(values)	Check whether *values* are contained in Series.
isna()	Detect missing values.
isnull()	Detect missing values.
item()	Return the first element of the underlying data as a python scalar.
items()	Lazily iterate over (index, value) tuples.
iteritems()	Lazily iterate over (index, value) tuples.
keys()	Alias for index.
kurt([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).
kurtosis([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).
last(offset)	Convenience method for subsetting final periods of time series data based on a date offset.
last_valid_index()	Return index for last non-NA/null value.
le(other[, level, fill_value, axis])	Less than or equal to of series and other, element-wise (binary operator *le*).
lt(other[, level, fill_value, axis])	Less than of series and other, element-wise (binary operator *lt*).
mad([axis, skipna, level])	Return the mean absolute deviation of the values for the requested axis.
map(arg[, na_action])	Map values of Series according to input correspondence.
mask(cond[, other, inplace, axis, level, ...])	Replace values where the condition is True.
max([axis, skipna, level, numeric_only])	Return the maximum of the values for the requested axis.
mean()	Weighted mean of the sampled distribution.
median()	Weighted median of the sampled distribution.
memory_usage([index, deep])	Return the memory usage of the Series.

Continued on next page

Table 12 – continued from previous page

min([axis, skipna, level, numeric_only])	Return the minimum of the values for the requested axis.
mod(other[, level, fill_value, axis])	Modulo of series and other, element-wise (binary operator *mod*).
mode([dropna])	Return the mode(s) of the dataset.
mul(other[, level, fill_value, axis])	Multiplication of series and other, element-wise (binary operator *mul*).
multiply(other[, level, fill_value, axis])	Multiplication of series and other, element-wise (binary operator *mul*).
ne(other[, level, fill_value, axis])	Not equal to of series and other, element-wise (binary operator *ne*).
neff()	Effective number of samples.
nlargest([n, keep])	Return the largest *n* elements.
nonzero()	Return the *integer* indices of the elements that are non-zero.
notna()	Detect existing (non-missing) values.
notnull()	Detect existing (non-missing) values.
nsmallest([n, keep])	Return the smallest *n* elements.
nunique([dropna])	Return number of unique elements in the object.
pct_change([periods, fill_method, limit, freq])	Percentage change between the current and a prior element.
pipe(func, *args, **kwargs)	Apply func(self, *args, **kwargs).
plot	alias of pandas.plotting._core.SeriesPlotMethods
pop(item)	Return item and drop from frame.
pow(other[, level, fill_value, axis])	Exponential power of series and other, element-wise (binary operator *pow*).
prod([axis, skipna, level, numeric_only, ...])	Return the product of the values for the requested axis.
product([axis, skipna, level, numeric_only, ...])	Return the product of the values for the requested axis.
ptp([axis, skipna, level, numeric_only])	Returns the difference between the maximum value and the
put(*args, **kwargs)	Applies the *put* method to its *values* attribute if it has one.
quantile([q])	Weighted quantile of the sampled distribution.
radd(other[, level, fill_value, axis])	Addition of series and other, element-wise (binary operator *radd*).
rank([axis, method, numeric_only, ...])	Compute numerical data ranks (1 through n) along axis.
ravel([order])	Return the flattened underlying data as an ndarray.
rdiv(other[, level, fill_value, axis])	Floating division of series and other, element-wise (binary operator *rtruediv*).
rdivmod(other[, level, fill_value, axis])	Integer division and modulo of series and other, element-wise (binary operator *rdivmod*).
reindex([index])	Conform Series to new index with optional filling logic, placing NA/NaN in locations having no value in the previous index.
reindex_axis(labels[, axis])	Conform Series to new index with optional filling logic.

Continued on next page

Table 12 – continued from previous page

reindex_like(other[, method, copy, limit, ...])	Return an object with matching indices as other object.
rename([index])	Alter Series index labels or name.
rename_axis([mapper, index, columns, axis, ...])	Set the name of the axis for the index or columns.
reorder_levels(order)	Rearrange index levels using input order.
repeat(repeats[, axis])	Repeat elements of a Series.
replace([to_replace, value, inplace, limit, ...])	Replace values given in *to_replace* with *value*.
resample(rule[, how, axis, fill_method, ...])	Resample time-series data.
reset_index([level, drop, name, inplace])	Generate a new DataFrame or Series with the index reset.
rfloordiv(other[, level, fill_value, axis])	Integer division of series and other, element-wise (binary operator *rfloordiv*).
rmod(other[, level, fill_value, axis])	Modulo of series and other, element-wise (binary operator *rmod*).
rmul(other[, level, fill_value, axis])	Multiplication of series and other, element-wise (binary operator *rmul*).
rolling(window[, min_periods, center, ...])	Provides rolling window calculations.
round([decimals])	Round each value in a Series to the given number of decimals.
rpow(other[, level, fill_value, axis])	Exponential power of series and other, element-wise (binary operator *rpow*).
rsub(other[, level, fill_value, axis])	Subtraction of series and other, element-wise (binary operator *rsub*).
rtruediv(other[, level, fill_value, axis])	Floating division of series and other, element-wise (binary operator *rtruediv*).
sample([n, frac, replace, weights, ...])	Return a random sample of items from an axis of object.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
select(crit[, axis])	Return data corresponding to axis labels matching criteria.
sem([axis, skipna, level, ddof, numeric_only])	Return unbiased standard error of the mean over requested axis.
set_axis(labels[, axis, inplace])	Assign desired index to given axis.
set_value(label, value[, takeable])	Quickly set single value at passed label.
shift([periods, freq, axis, fill_value])	Shift index by desired number of periods with an optional time *freq*.
skew([axis, skipna, level, numeric_only])	Return unbiased skew over requested axis Normalized by N-1.
slice_shift([periods, axis])	Equivalent to *shift* without copying data.
sort_index([axis, level, ascending, ...])	Sort Series by index labels.
sort_values([axis, ascending, inplace, ...])	Sort by the values.
sparse	alias of pandas.core.arrays.sparse.SparseAccessor
squeeze([axis])	Squeeze 1 dimensional axis objects into scalars.
std()	Weighted standard deviation of the sampled distribution.
str	alias of pandas.core.strings.StringMethods

Continued on next page

Table 12 – continued from previous page

sub(other[, level, fill_value, axis])	Subtraction of series and other, element-wise (binary operator *sub*).
subtract(other[, level, fill_value, axis])	Subtraction of series and other, element-wise (binary operator *sub*).
sum([axis, skipna, level, numeric_only, ...])	Return the sum of the values for the requested axis.
swapaxes(axis1, axis2[, copy])	Interchange axes and swap values axes appropriately.
swaplevel([i, j, copy])	Swap levels i and j in a MultiIndex.
tail([n])	Return the last *n* rows.
take(indices[, axis, convert, is_copy])	Return the elements in the given *positional* indices along an axis.
to_clipboard([excel, sep])	Copy object to the system clipboard.
to_csv(*args, **kwargs)	Write object to a comma-separated values (csv) file.
to_dense()	Return dense representation of NDFrame (as opposed to sparse).
to_dict([into])	Convert Series to {label -> value} dict or dict-like object.
to_excel(excel_writer[, sheet_name, na_rep, ...])	Write object to an Excel sheet.
to_frame([name])	Convert Series to DataFrame.
to_hdf(path_or_buf, key, **kwargs)	Write the contained data to an HDF5 file using HDFStore.
to_json([path_or_buf, orient, date_format, ...])	Convert the object to a JSON string.
to_latex([buf, columns, col_space, header, ...])	Render an object to a LaTeX tabular environment table.
to_list()	Return a list of the values.
to_msgpack([path_or_buf, encoding])	Serialize object to input file path using msgpack format.
to_numpy([dtype, copy])	A NumPy ndarray representing the values in this Series or Index.
to_period([freq, copy])	Convert Series from DatetimeIndex to PeriodIndex with desired frequency (inferred from index if not passed).
to_pickle(path[, compression, protocol])	Pickle (serialize) object to file.
to_sparse([kind, fill_value])	Convert Series to SparseSeries.
to_sql(name, con[, schema, if_exists, ...])	Write records stored in a DataFrame to a SQL database.
to_string([buf, na_rep, float_format, ...])	Render a string representation of the Series.
to_timestamp([freq, how, copy])	Cast to datetimeindex of timestamps, at *beginning* of period.
to_xarray()	Return an xarray object from the pandas object.
tolist()	Return a list of the values.
transform(func[, axis])	Call func on self producing a Series with transformed values and that has the same axis length as self.
transpose(*args, **kwargs)	Return the transpose, which is by definition self.
truediv(other[, level, fill_value, axis])	Floating division of series and other, element-wise (binary operator *truediv*).
truncate([before, after, axis, copy])	Truncate a Series or DataFrame before and after some index value.
tshift([periods, freq, axis])	Shift the time index, using the index's frequency if available.

Continued on next page

Table 12 – continued from previous page

tz_convert(tz[, axis, level, copy])	Convert tz-aware axis to target time zone.
tz_localize(tz[, axis, level, copy, ...])	Localize tz-naive index of a Series or DataFrame to target time zone.
unique()	Return unique values of Series object.
unstack([level, fill_value])	Unstack, a.k.a.
update(other)	Modify Series in place using non-NA values from passed Series.
valid([inplace])	Return Series without null values.
value_counts([normalize, sort, ascending, ...])	Return a Series containing counts of unique values.
var()	Weighted variance of the sampled distribution.
view([dtype])	Create a new view of the Series.
where(cond[, other, inplace, axis, level, ...])	Replace values where the condition is False.
xs(key[, axis, level, drop_level])	Return cross-section from the Series/DataFrame.

compress(*nsamples=None*)
Reduce the number of samples by discarding low-weights.

Parameters

neff: int, optional effective number of samples after compression. If not supplied, then reduce to the channel capacity (theoretical optimum compression). If <=0, then compress so that all weights are unity.

hist(**args*, ***kwargs*)
Weighted histogram of the sampled distribution.

mean()
Weighted mean of the sampled distribution.

quantile(*q=0.5*)
Weighted quantile of the sampled distribution.

var()
Weighted variance of the sampled distribution.

7.8 Module contents

Anesthetic: nested sampling post-processing.

Key routines:

- MCMCSamples.build
- MCMCSamples.read
- NestedSamples.build
- NestedSamples.read

CHAPTER 8

anesthetic: nested sampling visualisation

anesthetic nested sampling visualisation

Author Will Handley

Version 1.2.3

Homepage https://github.com/williamjameshandley/anesthetic

Documentation http://anesthetic.readthedocs.io/

`anesthetic` brings together tools for processing nested sampling chains, leveraging standard scientific python libraries.

You can see example usage and plots in the plot gallery, or in the corresponding Jupyter notebook.

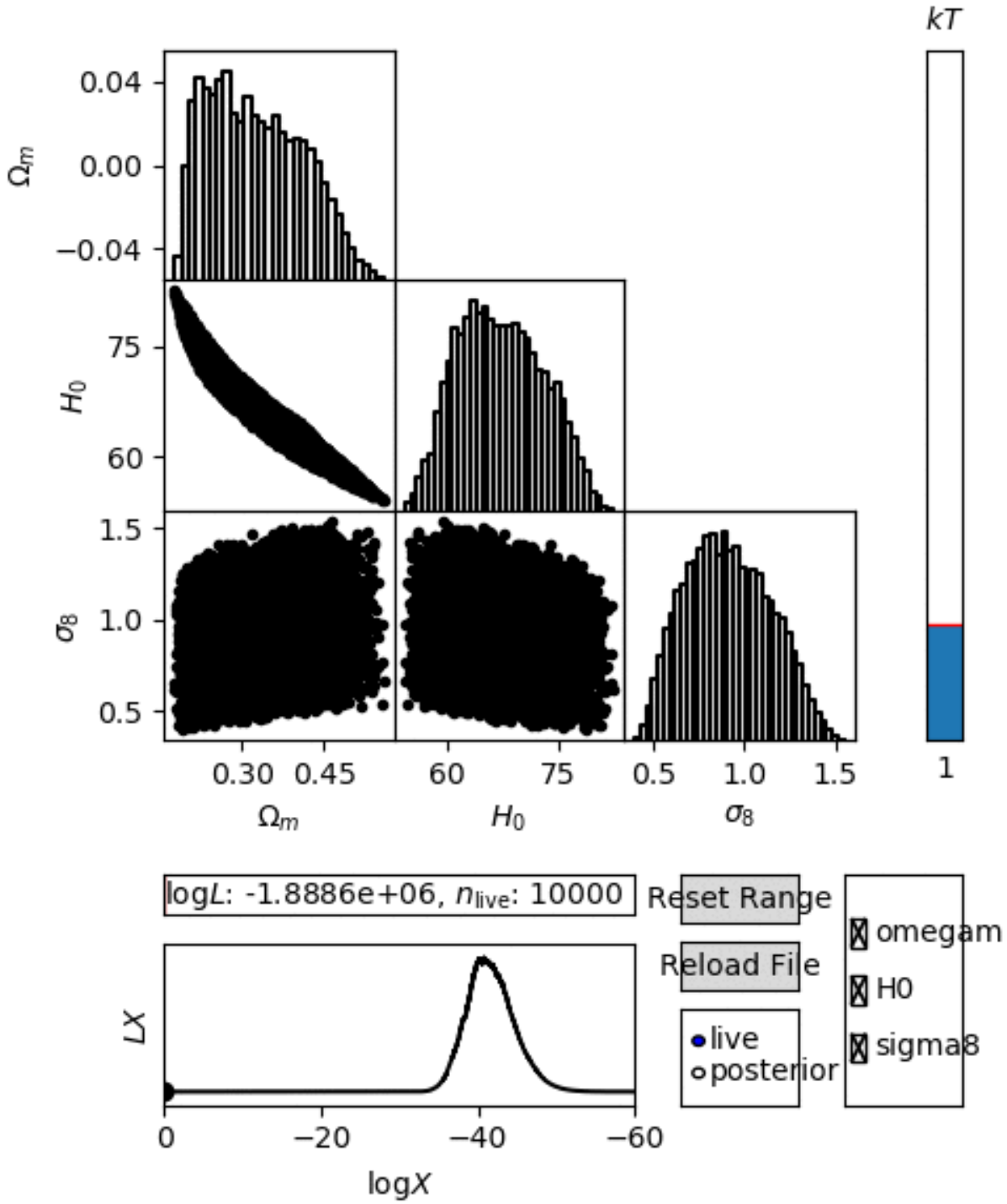
Current functionality includes:

- Computation of Bayesian evidences, Kullback-Liebler divergences and Bayesian model dimensionalities.
- Marginalised 1d and 2d plots.
- Dynamic replaying of nested sampling.

This tool was designed primarily for use with nested sampling outputs, although it can be used for normal MCMC chains.

For an interactive view of a nested sampling run, you can use the `anesthetic` script.

```
$ anesthetic <ns file root>
```



8.1 Features

- Both samples and plotting axes are stored as a `pandas.DataFrame`, with parameter names as indices, which makes for easy access and modification.

- Sensible color scheme for plotting nearly flat distributions.
- For easy extension/modification, uses the standard python libraries: numpy, scipy, matplotlib and pandas.

8.2 Installation

`anesthetic` can be installed via pip

```
pip install anesthetic
```

or via the setup.py

```
git clone https://github.com/williamjameshandley/anesthetic
cd anesthetic
python setup.py install --user
```

You can check that things are working by running the test suite:

```
export MPLBACKEND=Agg      # only necessary for OSX users
python -m pytest
flake8 anesthetic tests
pydocstyle --convention=numpy anesthetic
```

8.2.1 Dependencies

Basic requirements:

- Python 2.7 or 3.5+
- matplotlib
- numpy
- scipy
- pandas
- fastKDE

Documentation:

- sphinx
- numpydoc

Tests:

- pytest

8.3 Documentation

Full Documentation is hosted at ReadTheDocs. To build your own local copy of the documentation you'll need to install sphinx. You can then run:

```
cd docs
make html
```

8.4 Citation

If you use `anesthetic` to generate plots for a publication, please cite as:

```
Handley, (2019). anesthetic: nested sampling visualisation. Journal of Open
Source Software, 4(37), 1414, https://doi.org/10.21105/joss.01414
```

or using the BibTeX:

```
@article{anesthetic,
    doi = {10.21105/joss.01414},
    url = {http://dx.doi.org/10.21105/joss.01414},
    year  = {2019},
    month = {Jun},
    publisher = {The Open Journal},
    volume = {4},
    number = {37},
    pages = {1414},
    author = {Will Handley},
    title = {anesthetic: nested sampling visualisation},
    journal = {The Journal of Open Source Software}
}
```

8.5 Contributing

There are many ways you can contribute via the GitHub repository.

- You can open an issue to report bugs or to propose new features.
- Pull requests are very welcome. Note that if you are going to propose major changes, be sure to open an issue for discussion first, to make sure that your PR will be accepted before you spend effort coding it.

8.6 Questions/Comments

8.6.1 Another posterior plotting tool?

> This is my posterior plotter. There are many like it, but this one is mine.

There are several excellent tools for plotting marginalised posteriors:

- getdist
- corner
- pygtc
- dynesty
- MontePython

Why create another one? In general, any dedicated user of software will find that there is some functionality that in their use case is lacking, and the designs of previous codes make such extensions challenging. In my case this was:

1. For large numbers of samples, kernel density estimation is slow, or inaccurate (particularly for samples generated from nested sampling). There are kernel density estimators, such as fastKDE, which ameliorate many of these difficulties.

2. Existing tools can make it difficult to define new parameters. For example, the default cosmomc chain defines `omegabh2`, but not `omegab`. The transformation is easy, since `omegab = omegabh2/ (H0/100)**2`, but implementing this transformation in existing packages is not so trivial. `anesthetic` solves this issue by storing the samples as a pandas array, for which the relevant code for defining the above new parameter would be

```
from anesthetic import MCMCSamples
samples = MCMCSamples(root=file_root)                    # Load the samples
samples['omegab'] = samples.omegabh2/(samples.H0/100)**2 # Define omegab
samples.tex['omegab'] = '$\Omega_b$'                     # Label omegab
samples.plot_1d('omegab')                                # Simple 1D plot
```

3. Many KDE plotting tools have conventions that don't play well with uniformly distributed parameters, which presents a problem if you are trying to plot priors along with your posteriors. `anesthetic` has a sensible mechanism, by defining the contours by the amount of iso-probability mass they contain, but colouring the fill in relation to the probability density of the contour.

8.6.2 What's in a name?

There is an emerging convention for naming nested sampling packages with words that have nest in them (nestle and dynesty, nestorflow). Doing a UNIX grep:

```
grep nest /usr/share/dict/words
```

yields a lot of superlatives (e.g. greenest), but a few other cool names for future projects:

- amnesty
- defenestrate
- dishonestly
- inestimable
- minestrone
- rhinestone

I chose `anesthetic` because I liked the soft 'th', and in spite of the US spelling.

Python Module Index

a

Index

M

N

P

Q

R

S

T

U

V

W