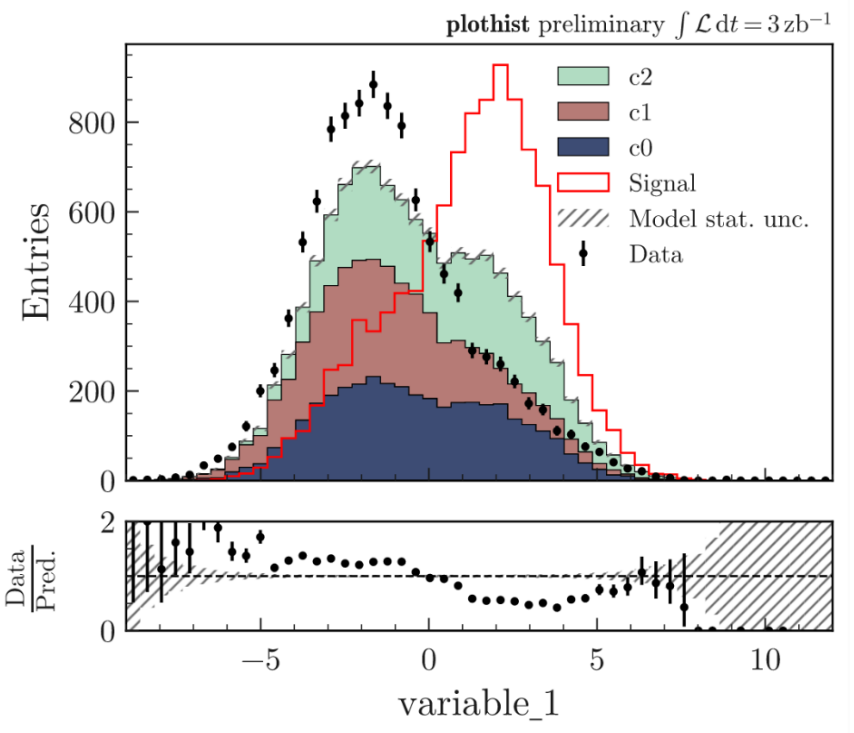
plothist package – scikit-HEP coordination

Plot and compare histograms in a scalable way and a beautiful style

Cyrille Praz, Tristan Fillinger

28/08/24

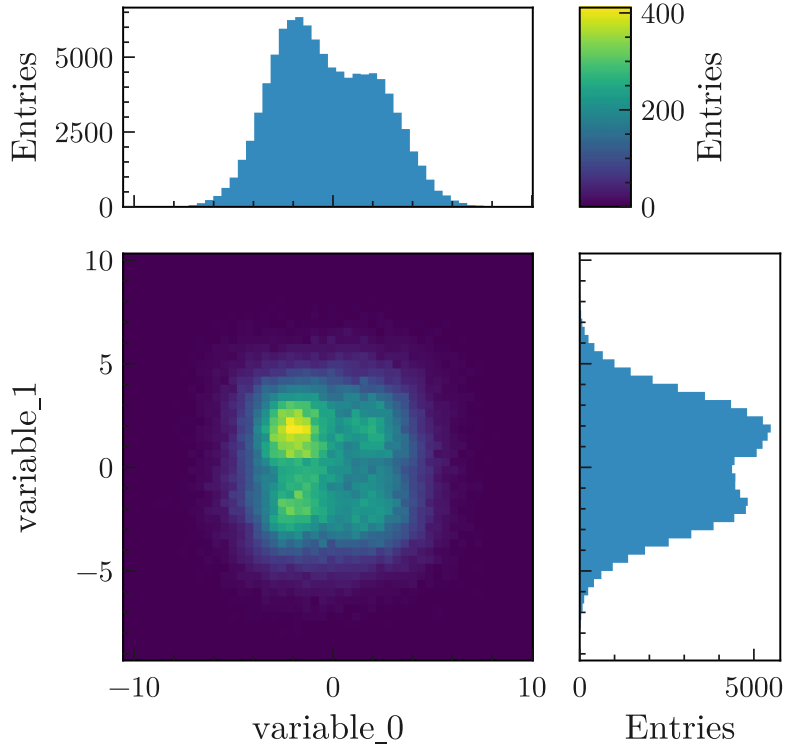
PyHEP.dev 2024



GitHub | pypi package 1.2.5 | docs main

Discussions Ask | DOI 10.5281/zenodo.12160362

License BSD 3-Clause | code style black



Concept

Goal of the package

- Provide tools to make standard high-energy physics plots in a scalable way and a publication-ready style, allowing analysts to focus on Physics rather than spending time on making and tuning plots.

Method

- Wrapper functions around matplotlib to plot `boost_histogram.Histogram` objects

Concept

Goal of the package

- Provide tools to make standard high-energy physics plots in a scalable way and a publication-ready style, allowing analysts to focus on Physics rather than spending time on making and tuning plots.

Method

- Wrapper functions around matplotlib to plot `boost_histogram.Histogram` objects

Main features

Style

- Default style is publication-ready (with little to no effort)

Concept

Goal of the package

- Provide tools to make standard high-energy physics plots in a scalable way and a publication-ready style, allowing analysts to focus on Physics rather than spending time on making and tuning plots.

Method

- Wrapper functions around matplotlib to plot `boost_histogram.Histogram` objects

Main features

Style

- Default style is publication-ready (with little to no effort)

Scalability

- Scalable wrt data size by separating histogram creation from plotting, allowing batching/parallelism
- Scalable wrt number of variables by storing plotting parameters in a variable registry

Concept

Goal of the package

- Provide tools to make standard high-energy physics plots in a scalable way and a publication-ready style, allowing analysts to focus on Physics rather than spending time on making and tuning plots.

Method

- Wrapper functions around matplotlib to plot `boost_histogram.Histogram` objects

Main features

Style

- Default style is publication-ready (with little to no effort)

Scalability

- Scalable wrt data size by separating histogram creation from plotting, allowing batching/parallelism
- Scalable wrt number of variables by storing plotting parameters in a variable registry

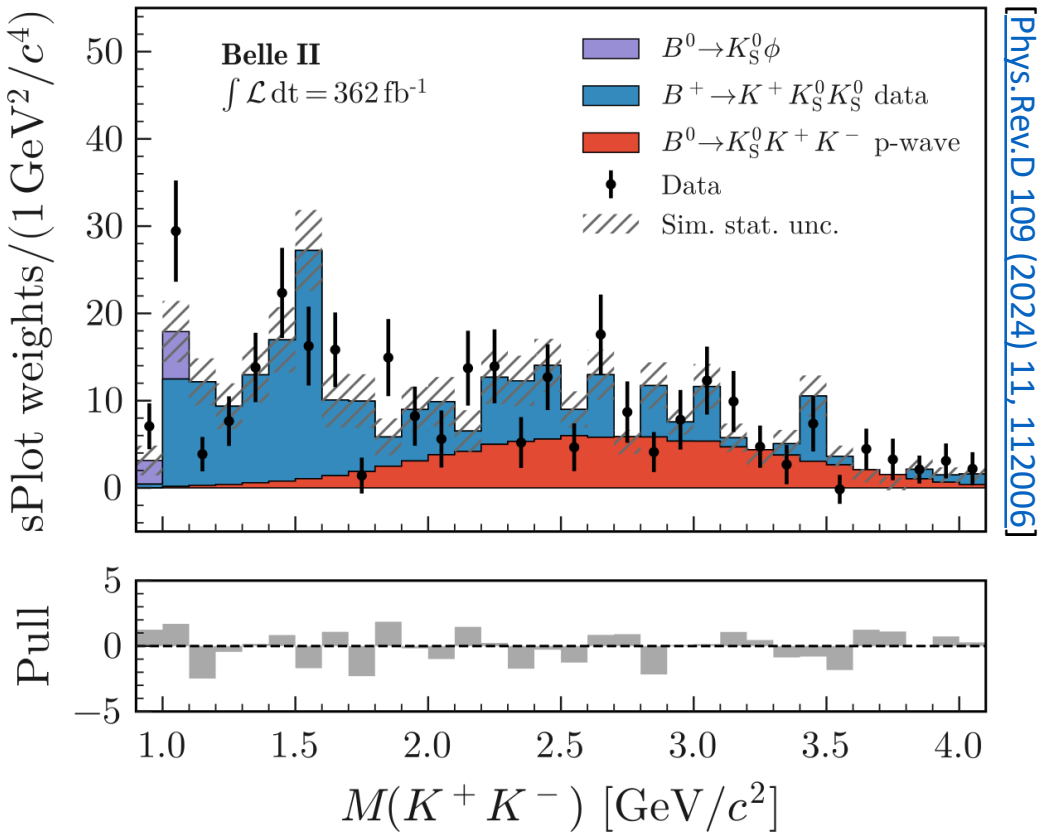
User-friendly

- A gallery of examples with complete codes
- Comprehensive and easy-to-navigate documentation
- Installable in one command line via pip: `pip3 install plothist`

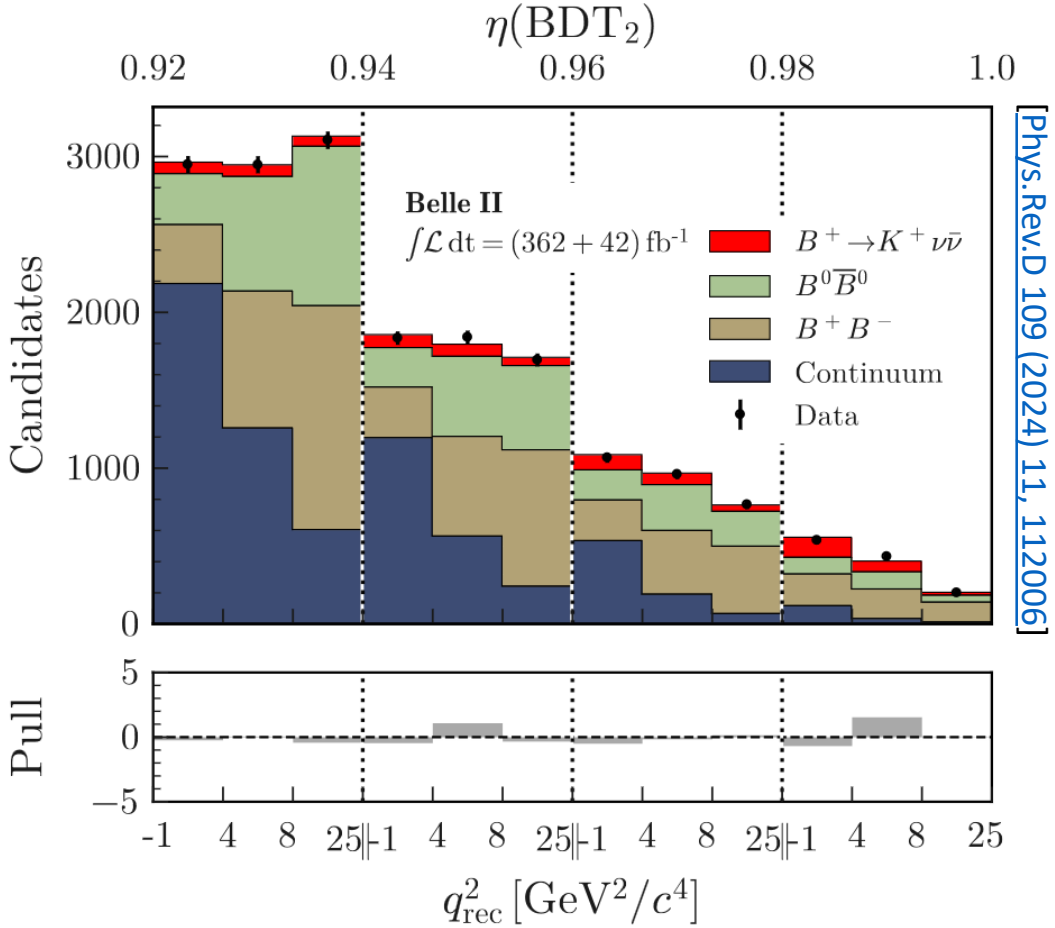
Default style

Style already compatible with **Physical Review Letters / Physical Review D** (with little to no effort)

Example from recent Belle II paper published in PRD



[Phys.Rev.D 109 (2024) 11, 112006]



[Phys.Rev.D 109 (2024) 11, 112006]

Functionality overview

Documentation: https://plothist.readthedocs.io

plothist

latest

Search docs

Installation and update

Font installation

SIMPLE EXAMPLES

Plot 1D histograms

Plot 2D histograms

Plot functions

ADVANCED EXAMPLES

Plot and compare model and data

Other advanced examples

UTILITIES

Variable registry

Style and colors

Utility functions

Plot result of a fit

DOCUMENTATION

Example gallery

Package references

Notes on statistics

Example gallery — Edit on GitHub

Example gallery

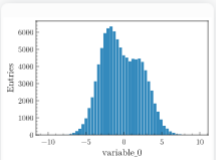
Gallery of images that are used in the doc.

Note

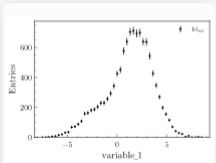
Click on an image to see the source code that generates it.

Plot 1D histograms

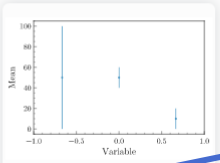
Gallery of images that are used in the 1D histogram section.



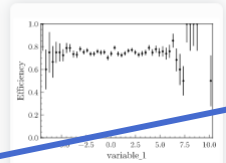
Histogram



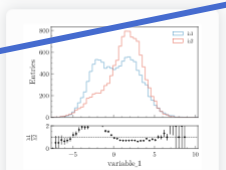
Histogram with uncertainties

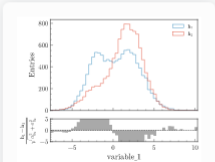


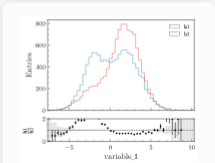
Mean histogram (profile)

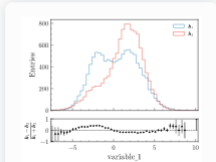


Comparison









Package references — Edit on GitHub

Package references

histogramming.py

plothist.histogramming.create_axis(*data*, *bins*, *range=None*)

Create an axis object for histogram binning based on the input data and parameters.

Parameters:
- **data** (*array-like*) – The input data for determining the axis range.
- **bins** (*int or array-like*) – The number of bins or bin edges for the axis.
- **range** (*None or tuple, optional*) – The range of the axis. If None, it will be determined based on the data.

Returns: An axis object for histogram binning.

Return type: Axis object

Raises: **ValueError** – If the range parameter is invalid or not finite.

plothist.histogramming.flatten_2d_hist(*hist*)

Flatten a 2D histogram into a 1D histogram.

Parameters: **hist** (*Histogram object*) – The 2D histogram to be flattened.

Returns: The flattened 1D histogram.

Return type: Histogram object

Raises: **ValueError** – If the input histogram is not 2D.

plothist.histogramming.make_2d_hist(*data*, *bins=(10, 10)*, *range=(None, None)*, *weights=1*)

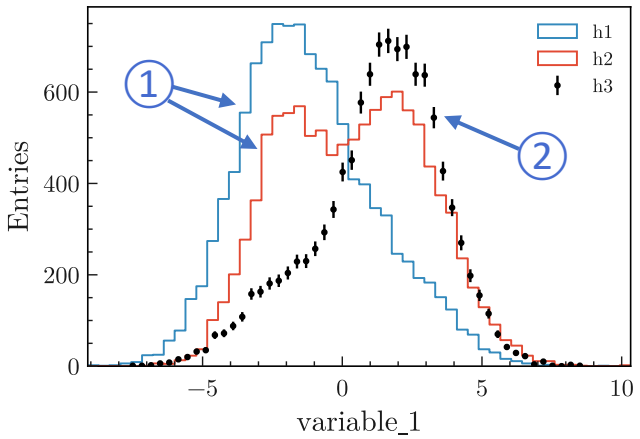
Create a 2D histogram object and fill it with the provided data.

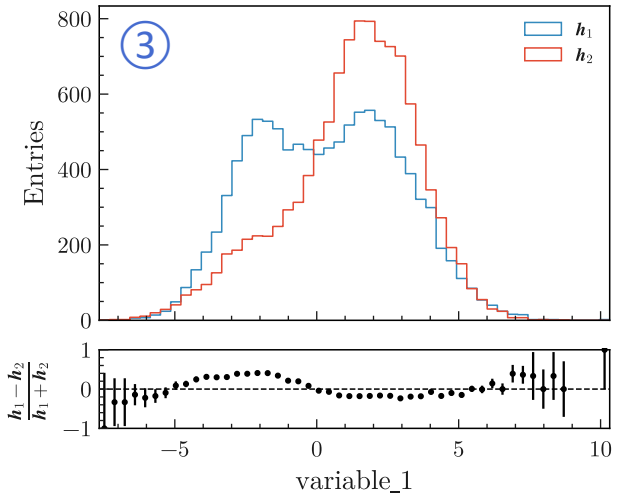
Parameters:
- **data** (*array-like*) – 2D array-like data used to fill the histogram.
- **bins** (*tuple, optional*) – Binning specification for each dimension of the histogram (default is (10, 10)). Each element of the tuple represents the number of bins for the corresponding dimension. Also support explicit bin edges specification (for non-constant bin size).

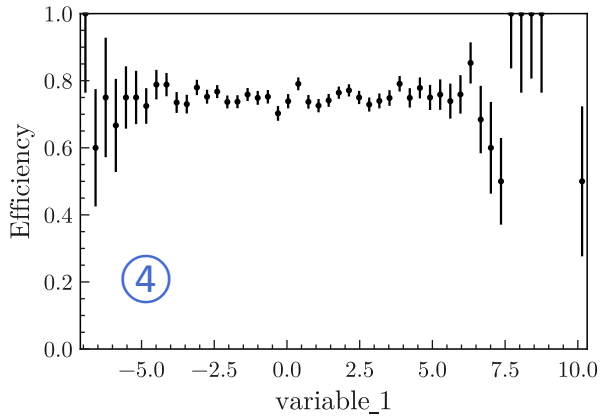
1D examples: overview

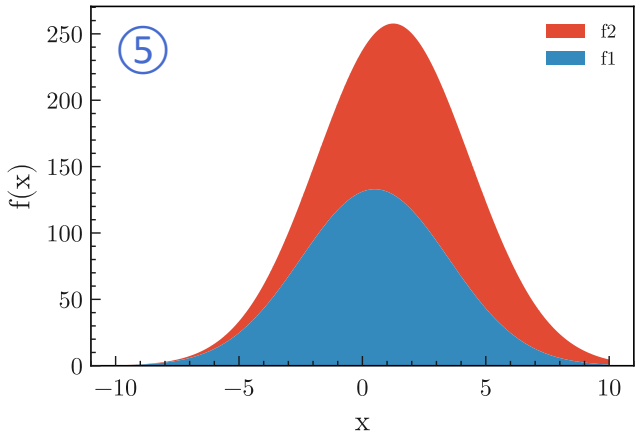
Create simple 1D histogram plots or compare them

`make_hist`	to create `boost_histogram` objects that are used in plothist
① `plot_hist`	to plot 1D histogram(s), takes `matplotlib` arguments for the style
② `plot_error_hist`	to plot 1D histogram with error bars (can be asymmetrical)
③ `plot_two_hist_comparison`	to compare 2 histograms
④ `plot_comparison`	to compare 2 histograms and only plot the comparison
⑤ `plot_function`	to plot 1D function(s), takes `matplotlib` arguments for the style









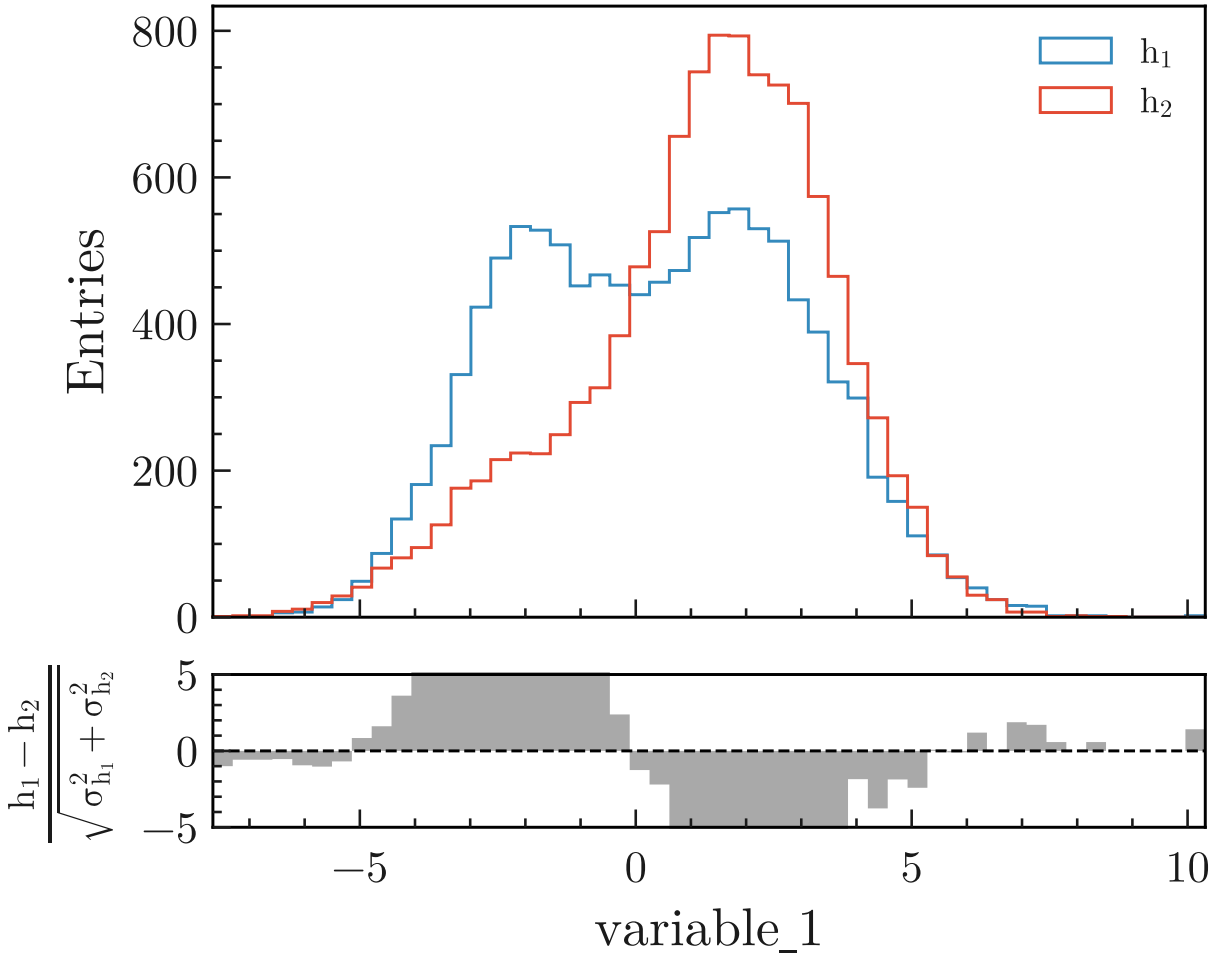
1D examples: histogram comparison

Example

```
from plothist import make_hist, plot_two_hist_comparison

h1 = make_hist(df["x1"], bins=50, range=[-7.5, 10.1])
h2 = make_hist(df["x2"], bins=50, range=[-7.5, 10.1])

fig, ax_main, ax_comparison = plot_two_hist_comparison(
    h1,
    h2,
    xlabel     ="variable_1",
    ylabel     ="Entries",
    h1_label   ="$h_1$",
    h2_label   ="$h_2$",
    comparison ="pull",
)
```



Available comparisons

Ratio: $\frac{h1}{h2}$

Difference: $h1 - h2$

Asymmetry: $\frac{h1-h2}{h1+h2}$

Pull: $\frac{h1-h2}{\sqrt{\sigma_{h1}^2+\sigma_{h2}^2}}$

Relative difference: $\frac{h1-h2}{h2}$

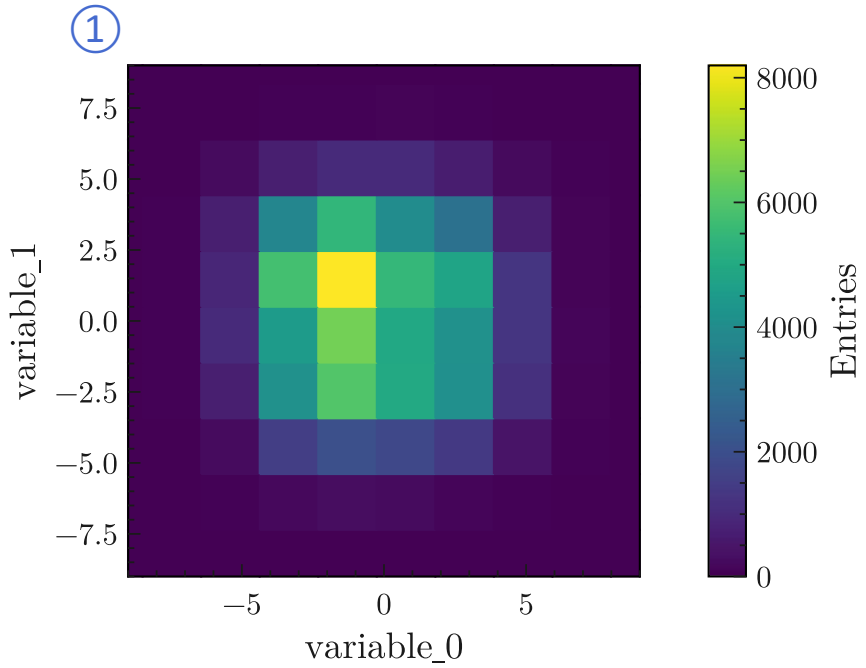
Efficiency: $\frac{h1}{h2}$ with $h1$ a subset of $h2$

2D examples: overview

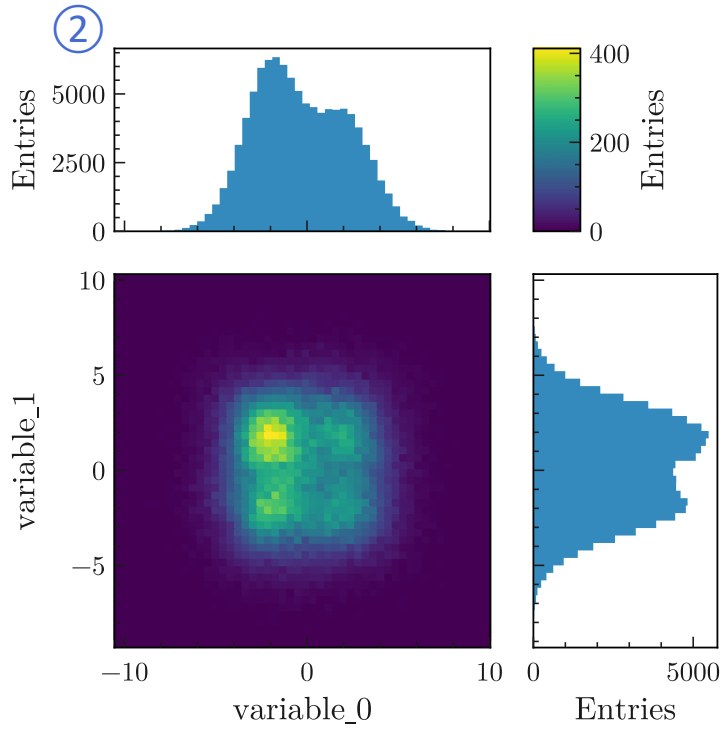
Create simple 2D histogram plots

	`make_2d_hist`	to create `boost_histogram` objects that are used in plothist
①	`plot_2d_hist`	to plot 2D histogram, takes `matplotlib` arguments for the style
②	`plot_2d_hist_with_projections`	to plot 2D hist with the 1D projections

①



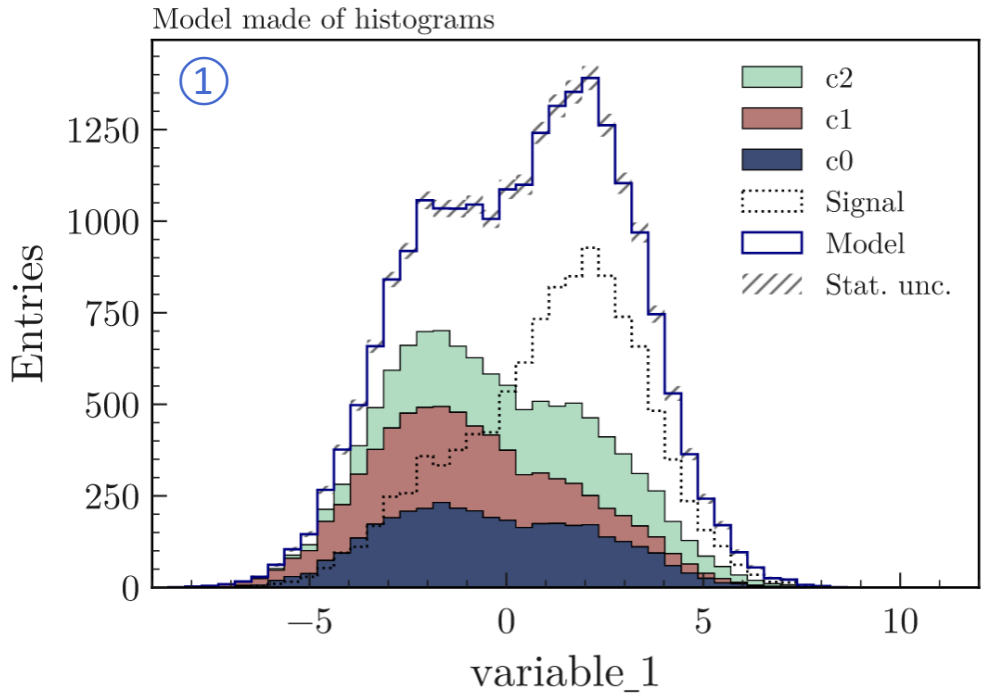
②

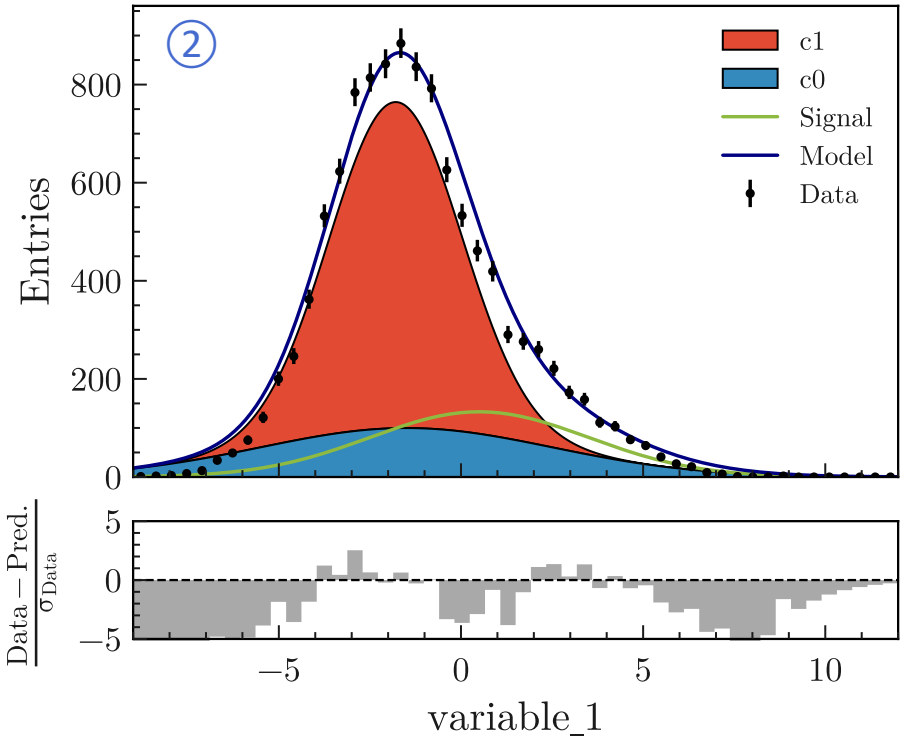


High-energy physics examples: overview

Create Data/model plots in a few lines of code

① `plot_model`	to plot stacked and/or unstacked histograms or functions together
② `plot_data_model_comparison`	to compare stacked and/or unstacked histograms or functions with data





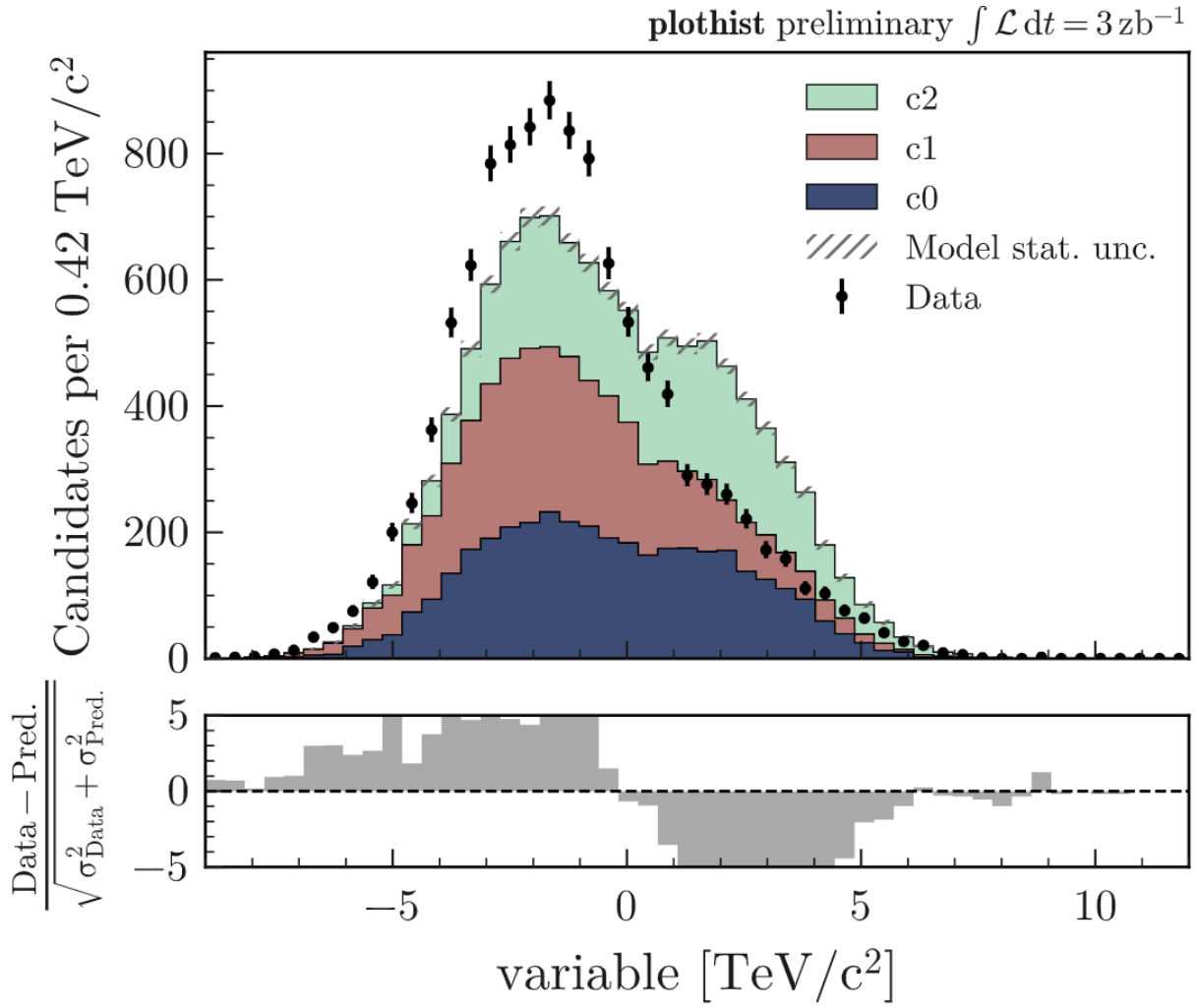
High-energy physics examples: Data vs Model comparison

Example

```
from plothist import plot_data_model_comparison, add_luminosity

fig, ax_main, ax_comparison = plot_data_model_comparison(
    data_hist          =data_hist,
    stacked_components =background_hists,
    stacked_labels     =background_categories_labels,
    stacked_colors     =background_categories_colors,
    xlabel             ="variable [TeV/$c^2$]",
    ylabel             ="Candidates per 0.42 [TeV/$c^2$]",
    comparison         ="pull"
)

add_luminosity(
      collaboration="plothist",
      ax=ax_main,
      lumi=3,
      lumi_unit="zb",
      preliminary=True
)
```



Available comparisons

Ratio: $\frac{h1}{h2}$

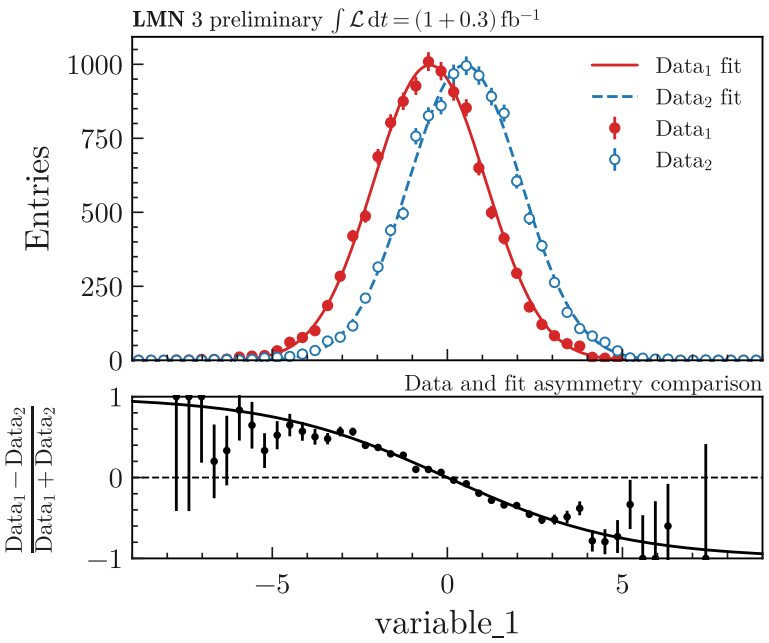
Pull: $\frac{h1-h2}{\sqrt{\sigma_{h1}^2+\sigma_{h2}^2}}$ or $\frac{h1-h2}{\sigma_{h1}}$

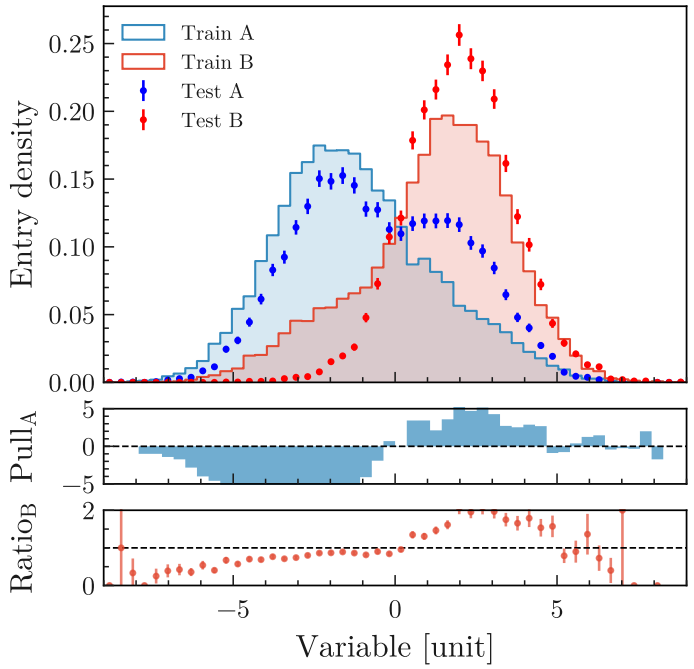
Difference: $h1-h2$

Relative difference: $\frac{h1-h2}{h2}$

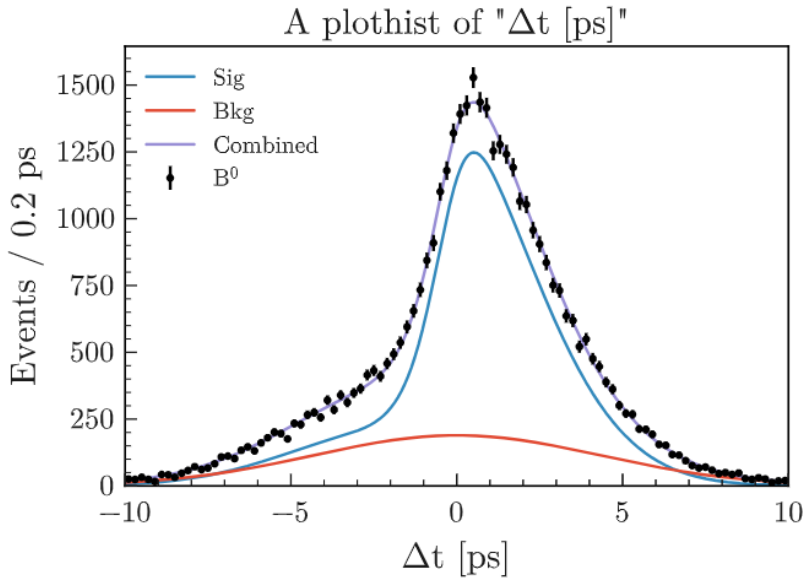
And more!

Complex examples, code still simple and easy-to-navigate





Tutorial to transfer RooFit, zfit or pyhf plot to plothist



Utility functions

`install_latin_modern_fonts`	from the terminal to install the LaTeX fonts
`add_luminosity`	to easily add luminosity + collaboration text on the plot
`get_color_palette`	to sample any color palette
`add_text`	to easily add text on a plot
`set_fitting_ylabel_fontsize`	to automatically set the ylabel font size to fit the plot
`And more!`	

Variable registry

Manage any number of variables using unique identifiers (keys) in a YAML file

```yaml
variable_0:
  name: variable_0
  bins: 50
  range:
   - -10.55227774892869 # min(df["variable_0"])
   - 10.04658448558009  # max(df["variable_0"])
  label: variable_0
  log: false
  legend_location: best
  ...

variable_1:
    ...
```

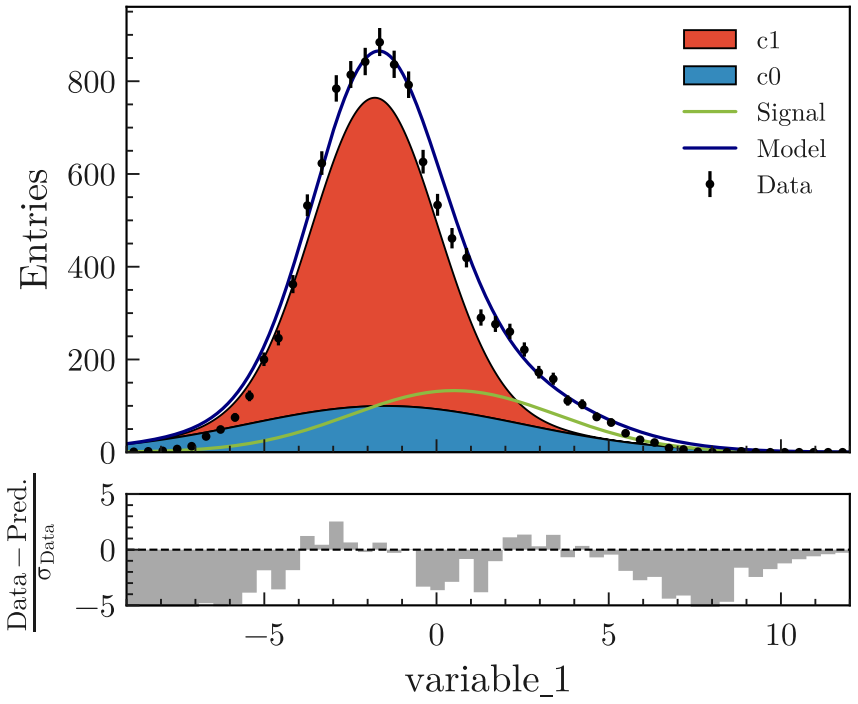
plothist outlook

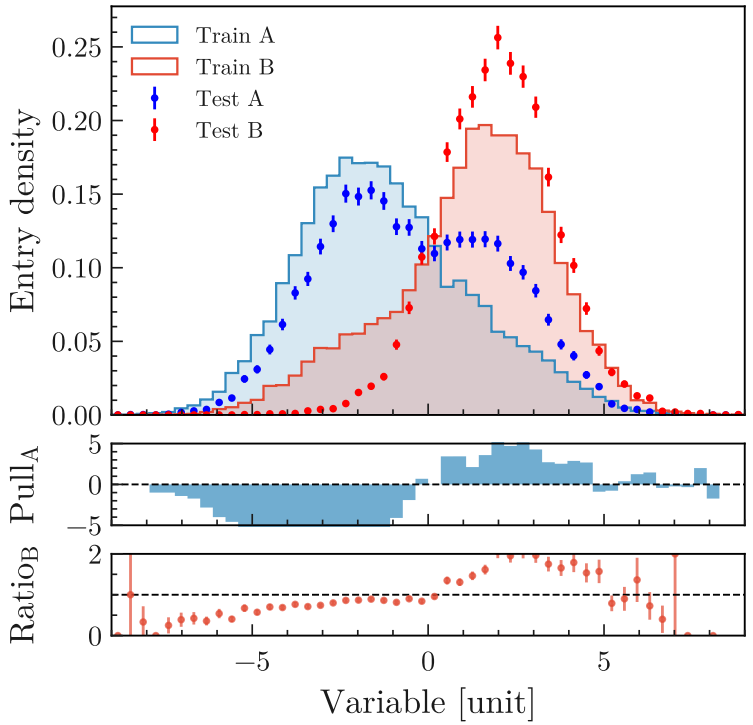
- Every example shown here (and many others) available in the doc:

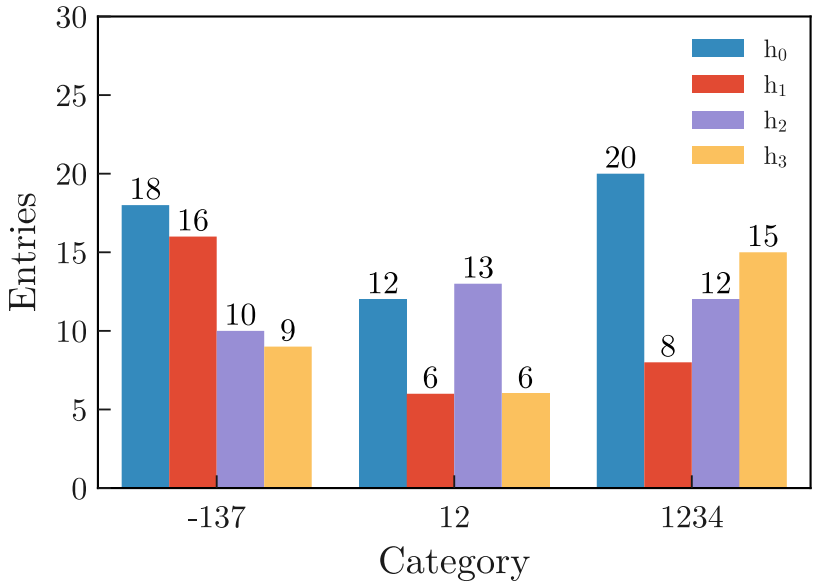
  https://plothist.readthedocs.io
  https://plothist.readthedocs.io/en/latest/example_gallery

- plothist already used by collaborators of multiple experiments
  Main feedback: plothist is a time saver, so they can spend more time on physics than on making and tuning plots

- **What's next**: coordination with scikit-HEP (see next slides)







Comparison/overlapping with mplhep

	mplhep	plothist
Histogramming	Plot any histogram (numpy, Hist /boost-histogram, ROOT Histogram) via extended PlottableProtocol histograms	Only plot Hist / boost-histogram histograms
Plotting	Only simple plots, no histogram comparison or data-model comparison	Provide high-level functions to create out-of-the-box data-model comparisons
Style	Supports multiple collaboration styles (LHC)	One default style, compatible with Physical Review Letters / Physical Review D and Belle II
Utilities	Various utility functions	Variable registry Various utility functions

Coordinating the **plothist** and **mplhep** packages

- Already multiple exchanges and meetings with **Andrzej Novak** and **Jonas Eschle**
- 2 scenarios considered so far (see next 2 slides)

Option 1: two orthogonal packages

Idea: 2 packages, which are orthogonal in terms of functionalities

plothist

Plot and **compare** models made of **histograms and/or functions** and stays **independent** of **HEP**

In the package:

- **plotting functions** (simple plots, comparison between data and model made of hists or functions, 2d with projections…)
- Some plotting **utility functions**

mplhep

Uses plothist functions **to plot** and has all the **HEP utilities and styles**

In the package:

- All the **HEP experiment styles**
- **HEP** plotting **utility functions**
- **Variable registry**

Work needed

- UHI compatibility for plothist functions, remove boost-histogram dependency
- Overload some plothist plot functions with HEP functionalities (currently in histplot of mplhep)

For the user

- HEP users will only use mplhep
- non-HEP users can use plothist

Option 2: merging

Idea: merge the 2 packages into a new one that can be used by HEP users

Work needed

Take from plothist

Simple 1D and 2D plot:

Take **plot_hist** and **plot_2d_hist** method to plot, make them UHI compatible and overload them with → **histplot** & **hist2dplot** HEP arguments

All the **other plotting functions** (simple comparisons, comparison between data and model made of hists or functions, 2d with projections…)

Keep the **style as the default one**

All the **utility functions**

need to decide for the overlapping ones

Variable registry

Take from mplhep

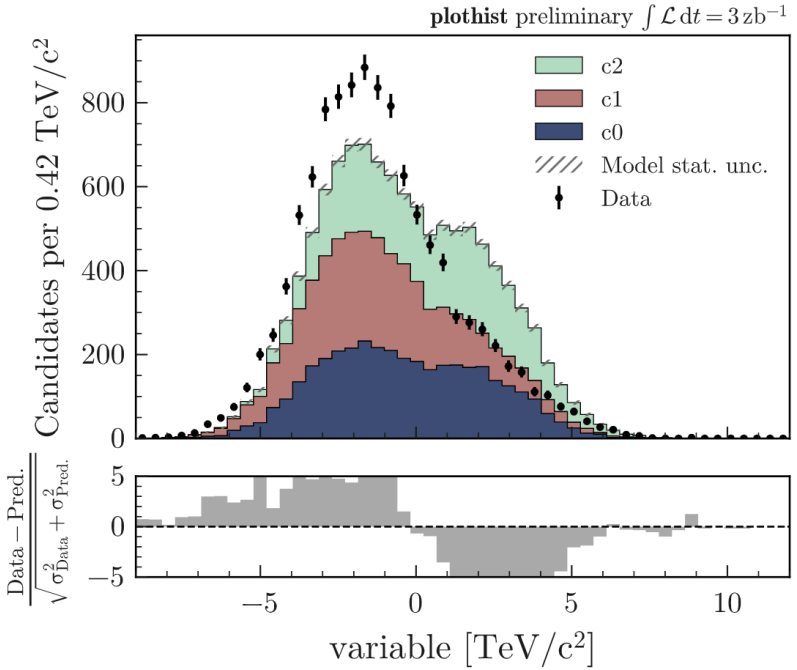
All the **HEP experiment styles**

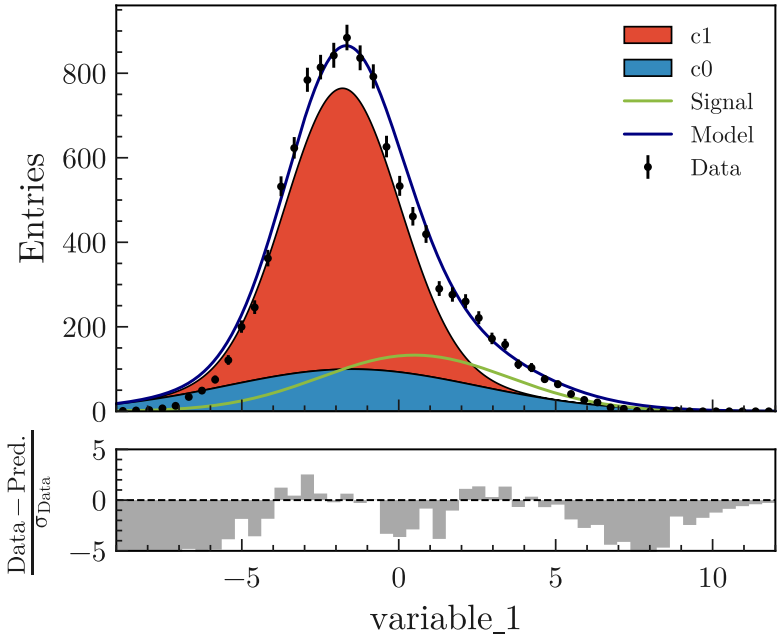
as mplstyle + python files, easily callable

All the **utility functions**

Summary

- plothist provides tools to plot Hist / boost-histogram objects as well as high-level data model comparisons





- Two scenarios considered so far to coordinate with mplhep:
  - plothist focuses on plotting, mplhep on the HEP functionalities
  - plothist and mplhep are merged in one new package

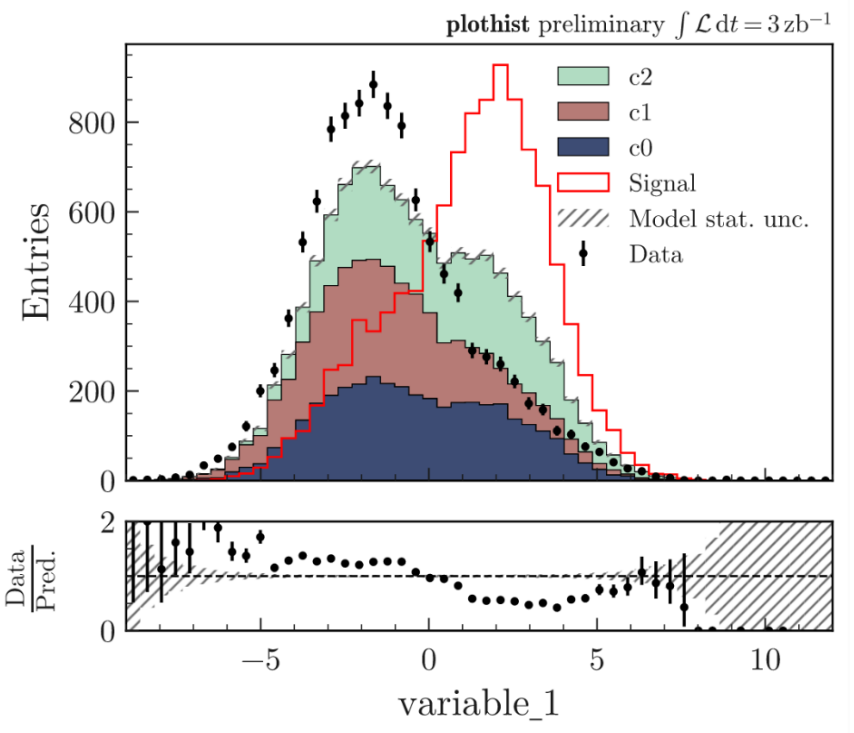
- As a first step, we plan to make plothist functions UHI compatible and drop dependency on boost-histogram

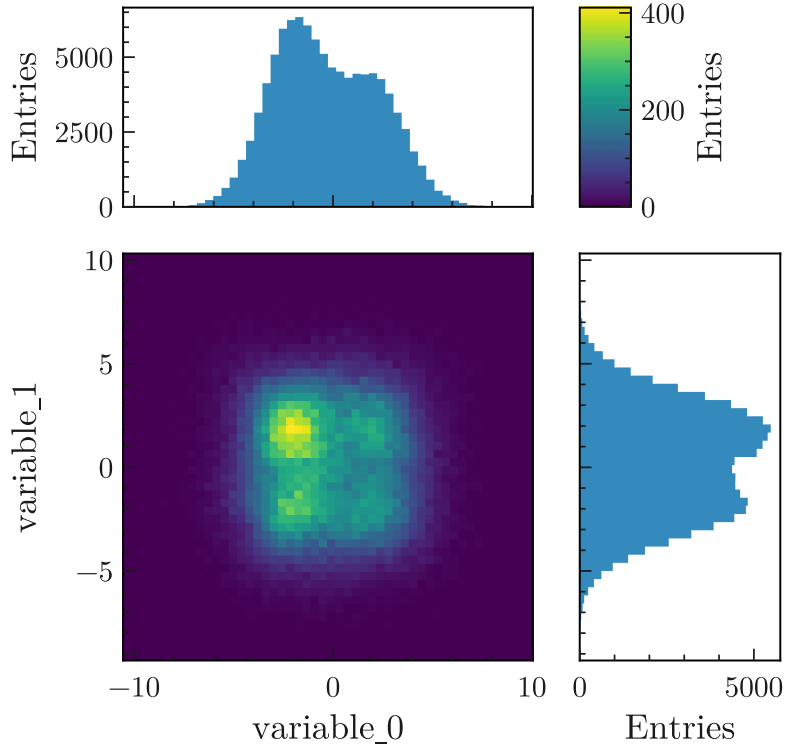
Thank you for your attention!

Tristan Fillinger, Cyrille Praz

28/08/24

PyHEP.dev 2024



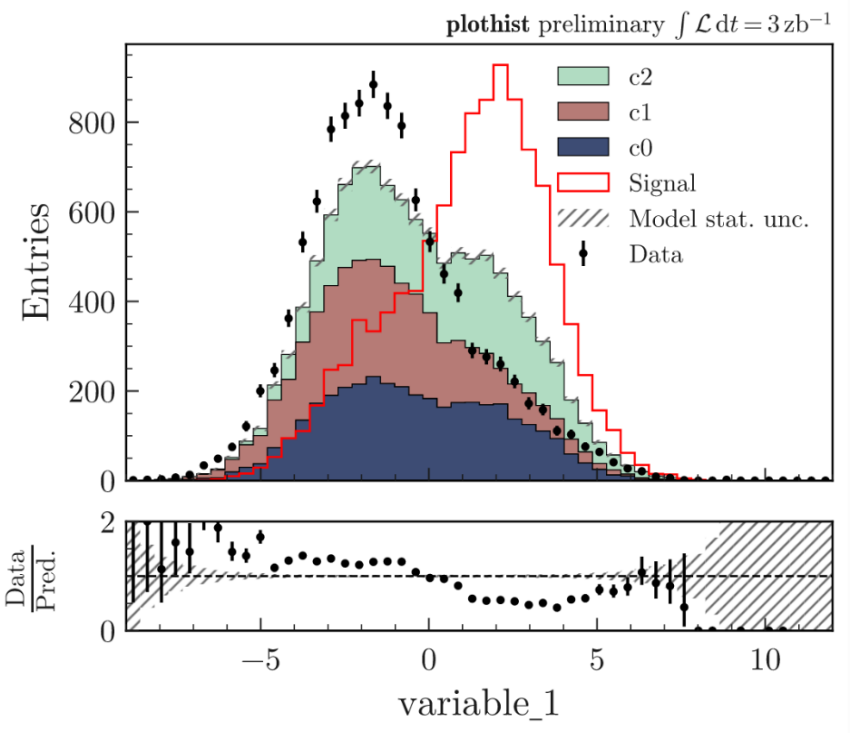


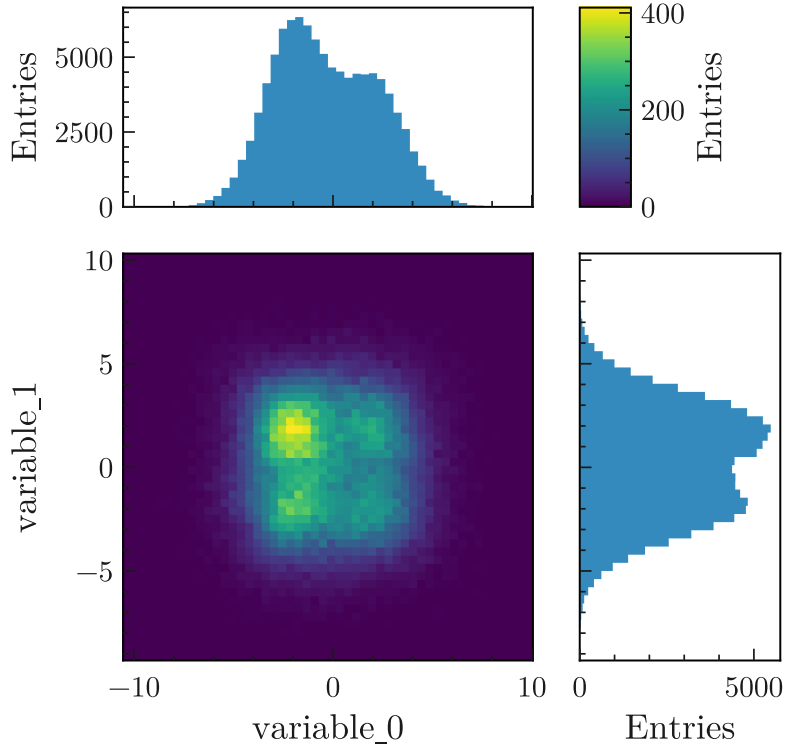
Backup

Tristan Fillinger, Cyrille Praz

28/08/24

PyHEP.dev 2024





Live demonstration of the package  

- Interactive Jupyter notebooks
  - High-energy physics examples
    - model made of functions
    - model made of histograms
  - 2D histograms with variable registry

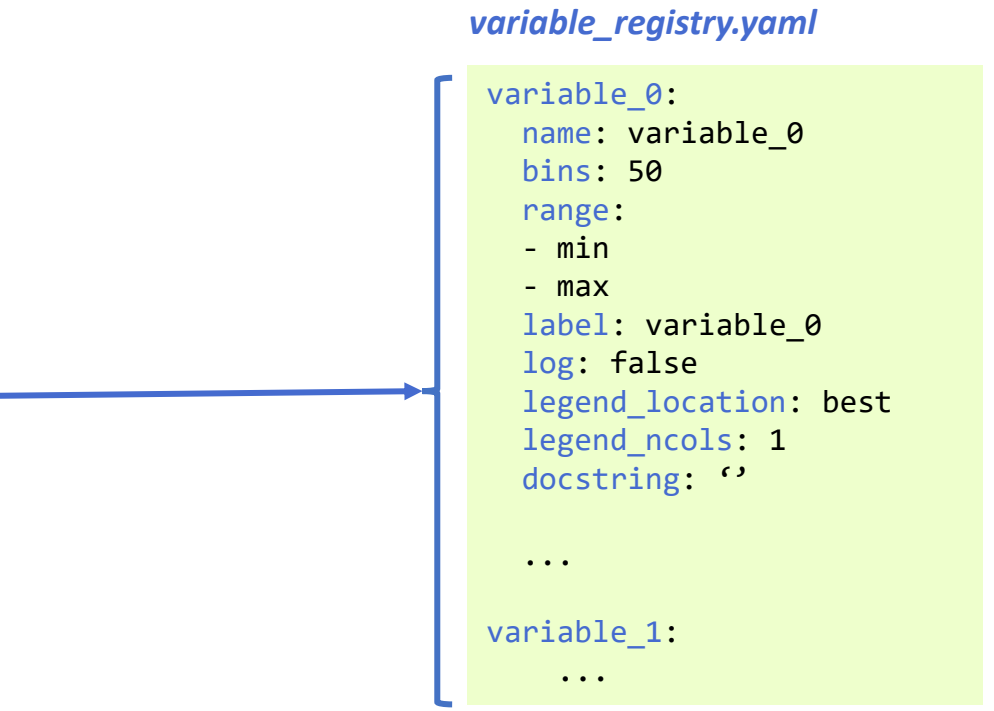
Variable registry

Functionalities

- Manage any number of variable using unique identifiers (keys)
- Store any information in a database (YAML file)

```
variable_keys = ["variable_0", "variable_1", "variable_2"]

create_variable_registry(variable_keys)
```



Variable registry

Functionalities

- Manage any number of variable using unique identifiers (keys)
- Store any information in a database (YAML file)
- Retrieve information with only the keys

```python
variable_keys = ["variable_0", "variable_1", "variable_2"]

create_variable_registry(variable_keys)

variable = get_variable_from_registry("variable_0")

# variable is a dictionary
# Get the name: variable["name"]
# Get the range: variable["range"]
# …
```

variable_registry.yaml

```yaml
variable_0:
  name: variable_0
  bins: 50
  range:
  - min
  - max
  label: variable_0
  log: false
  legend_location: best
  legend_ncols: 1
  docstring: ‘’

  ...

variable_1:
    ...
```

Variable registry

Functionalities

- Manage any number of variable using unique identifiers (keys)
- Store any information in a database (YAML file)
- Retrieve information with only the keys
- Update or add automatically information (like the range)

```
variable_keys = ["variable_0", "variable_1", "variable_2"]

create_variable_registry(variable_keys)

variable = get_variable_from_registry("variable_0")

update_variable_registry_ranges(df, variable_keys)
```

variable_registry.yaml

```
variable_0:
  name: variable_0
  bins: 50
  range:
   - -10.55227774892869 # min(df["variable_0"])
   - 10.04658448558009  # max(df["variable_0"])
  label: variable_0
  log: false
  legend_location: best
  legend_ncols: 1
  docstring: ‘’

  ...

variable_1:
    ...
```

mplhep functions: checking the overlap with plothist

Basic functionalities

- histplot = plot_hist but with UHI compatibilities and some utility arguments
- hist2dplot = plot_2d_hist but with UHI compatibilities and some utility arguments

Experiment label helpers

- [exp].lumitext ≈ add_luminosity
- [exp].text ≈ add_luminosity
- [exp].label = add_luminosity

Axes helpers

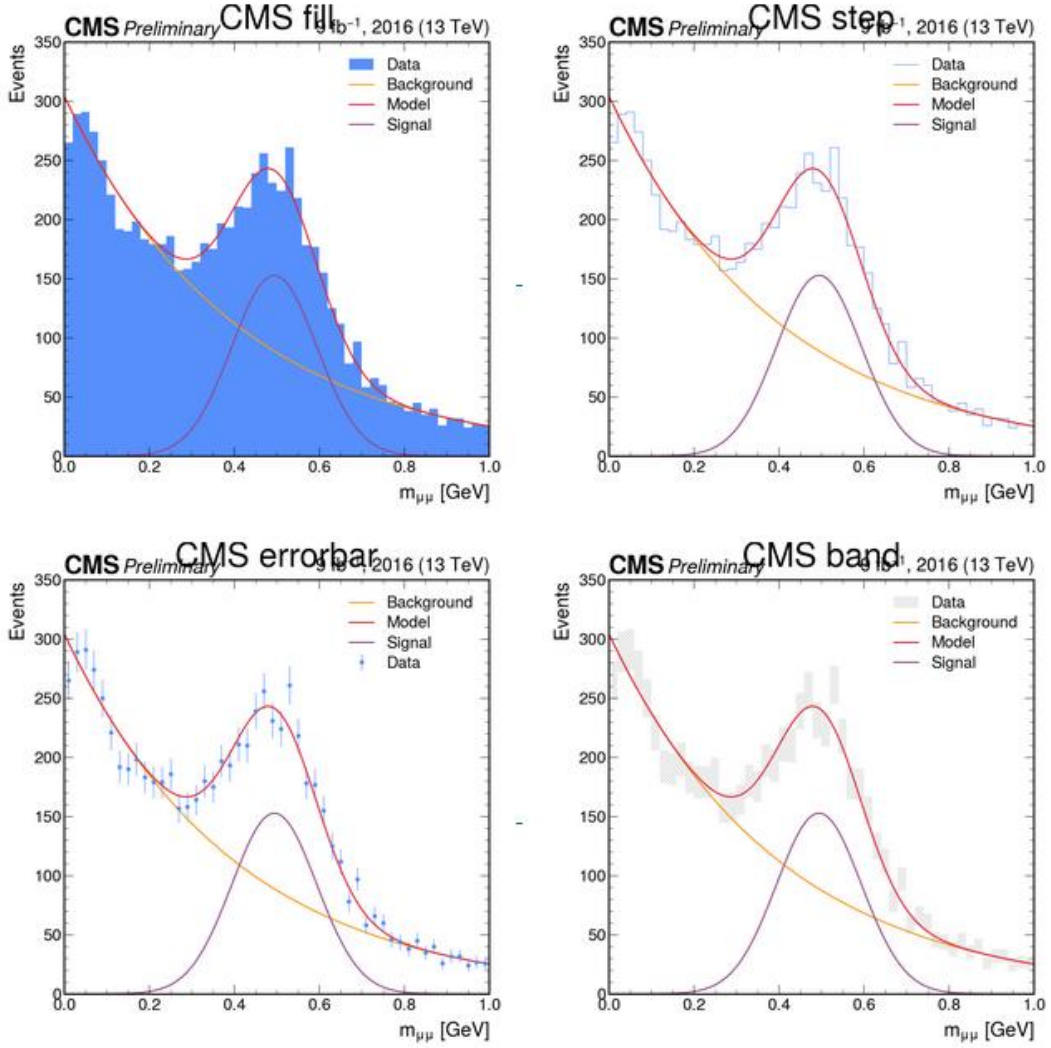
- yscale_legend: really nice (might want to add modifying also x axis range)
- ylow: nice to have
- yscale_anchored_text = set_ylabel_fontsize
- mpl_magic = set_ylabel_fontsize + ylow + yscale_legend

Figure helpers

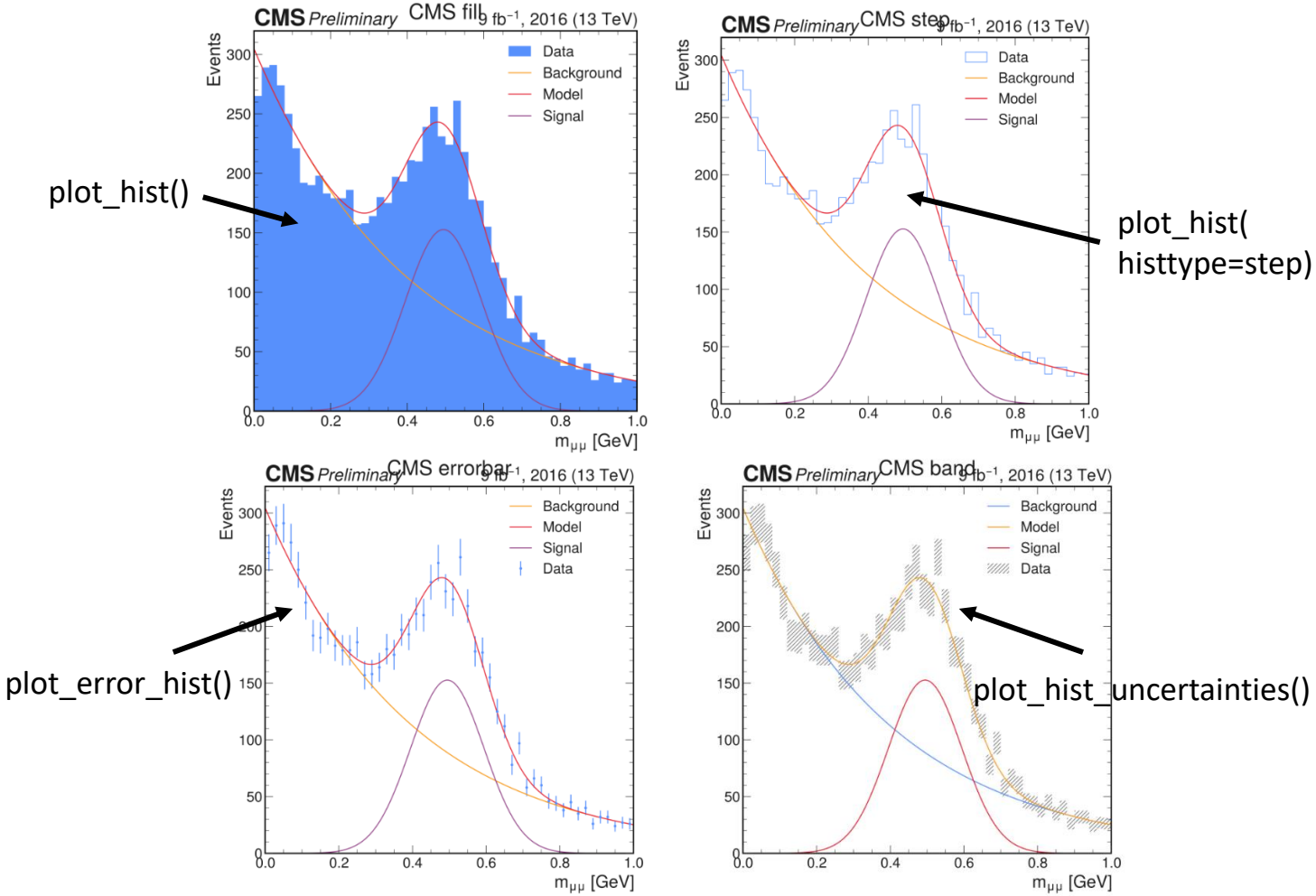
- append_axes: nice to have
- box_aspect: same idea as plothist has for 2d hist with projection plot
- make_square_add_cbar: same idea as with plothist square_ax option
- rescale_to_axessize: same idea as in savefig
- sort_legend = plot_reordered_legend

mplhep style compatibility

From mplhep doc



Using plothist functions to plot the histogram
+ applying mplhep CMS style (mplhep.style.CMS + mplhep.cms.label)

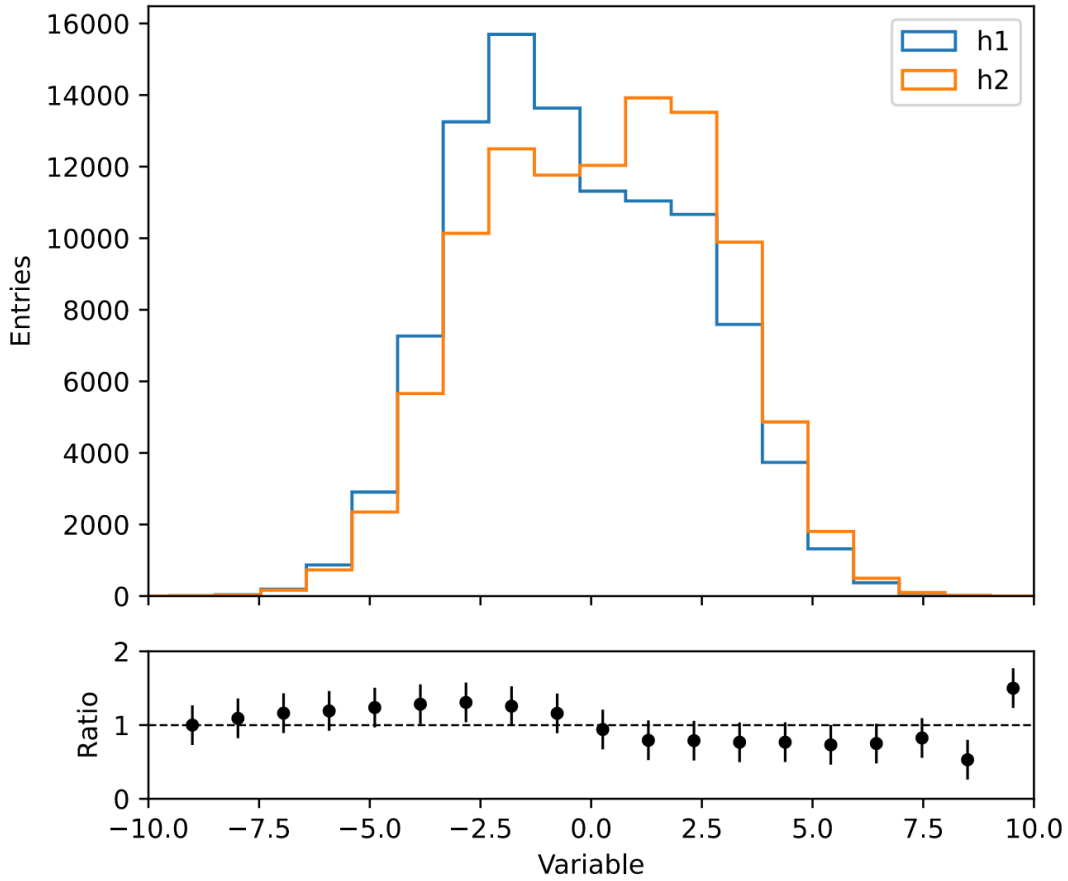


- Applying LHC style works out of the box
- (almost) no conflicts

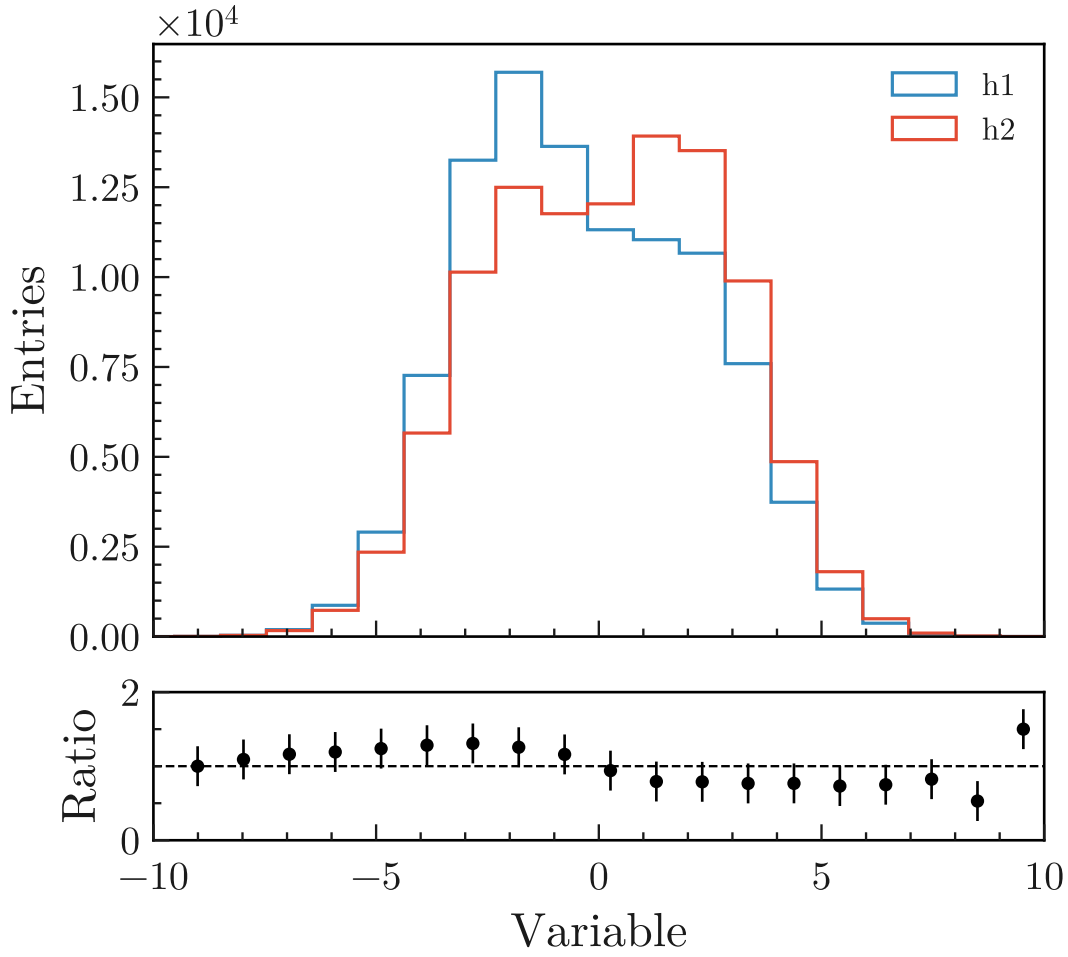
Default plothist style without using plothist functions

If you just want the style, add `import plothist` to your python script

Simple matplotlib script



Same script, just `import plothist` added



plothist style

Fonts

Latin Modern (LaTeX)

Colors

Provide a function to sample colors from:

- Default palette
- Cubehelix palette
- Any matplotlib palettes