

The Scikit-HEP project: overview and future

Eduardo Rodrigues
University of Liverpool



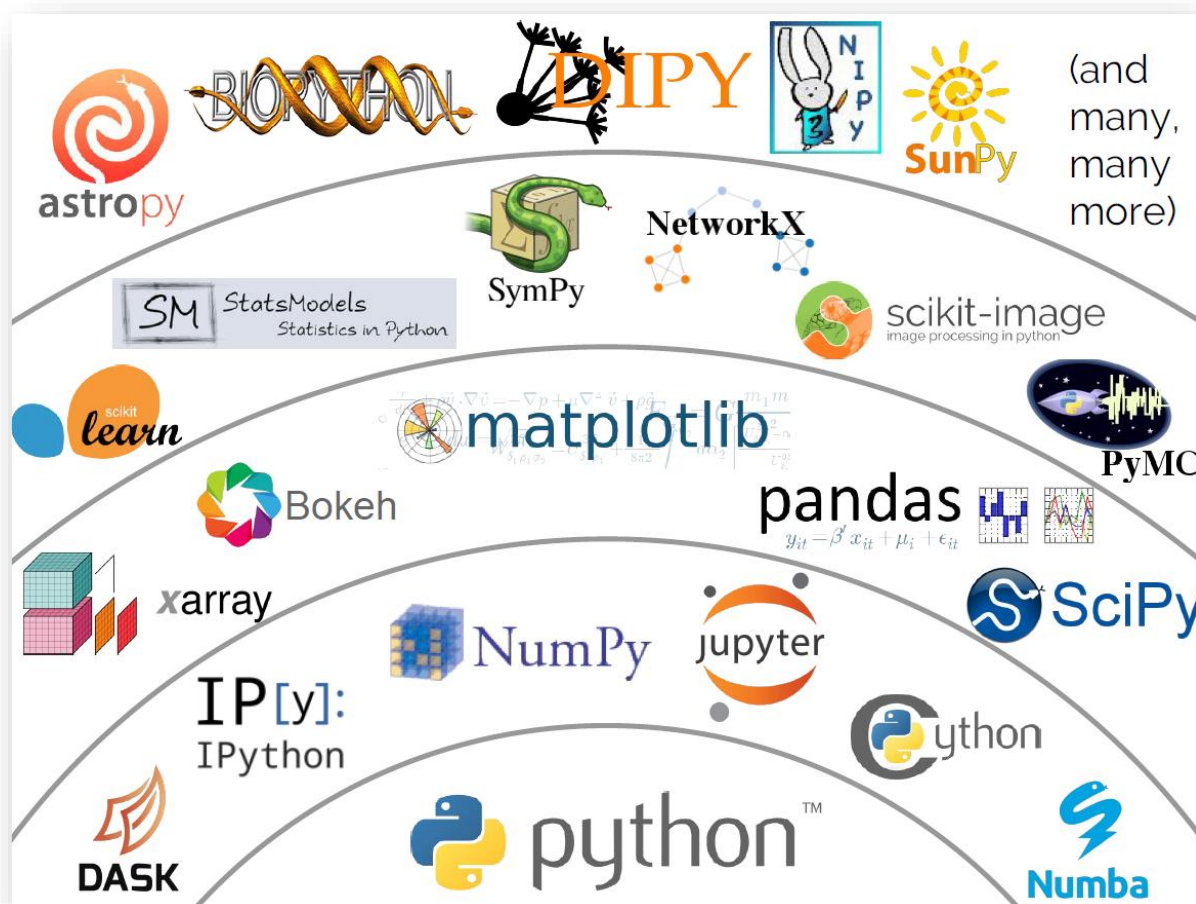
- Famous “shell diagram” of the Python scientific ecosystem outside HEP, back in the days:
- **What about HEP? Why was HEP not present back in 2017?**
- Sort of a philosophical question at this point. But there was a niche ...

Domain-specific

Python's

Scientific

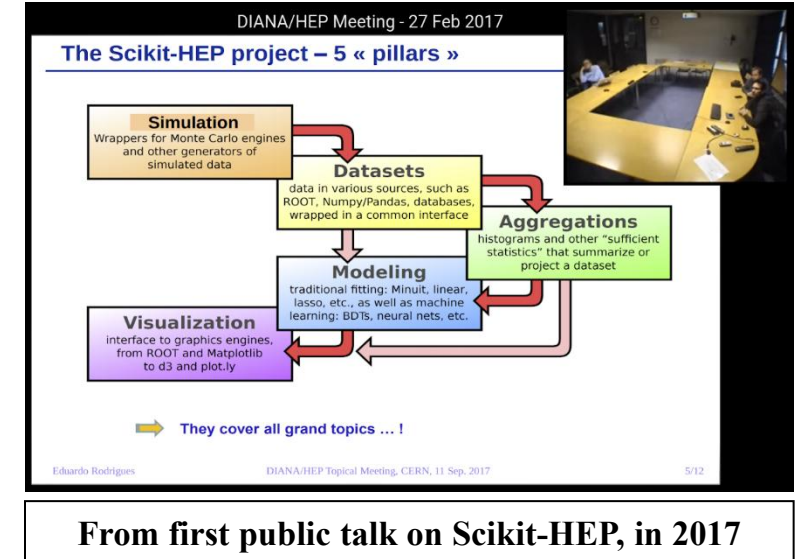
stack



Jake VanderPlas,
*The Unexpected Effectiveness
of Python in Science,*
PyCon 2017

What is Scikit-HEP?

- **Scikit-HEP started with the goal to:**
 - Improve the **interoperability** between HEP tools and the scientific Python ecosystem
 - **Build a community of developers and users** – be **community-driven and community-oriented**
 - Improve the **discoverability of relevant tools**
 - ...
- **Project started around concept of pillars:**
datasets, data aggregation, modelling, visualisation and simulation (and “utilities”)
- Initial approach quickly evolved towards a **toolset**, not a toolkit



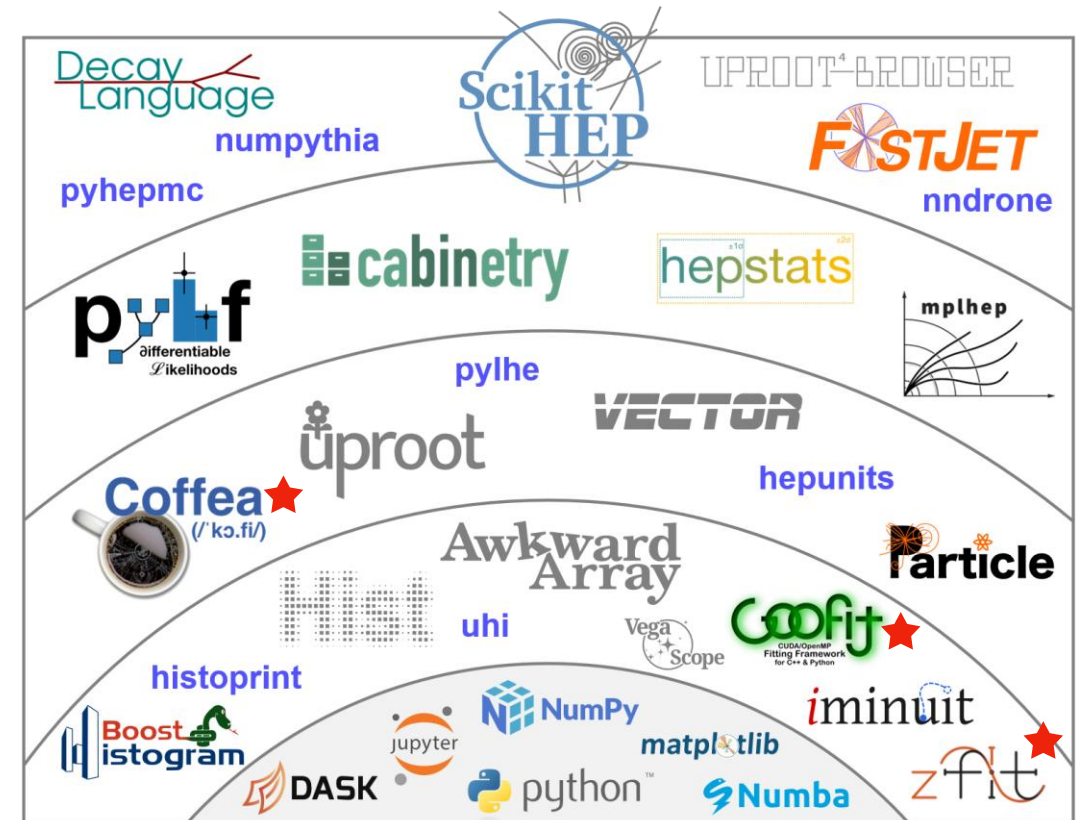
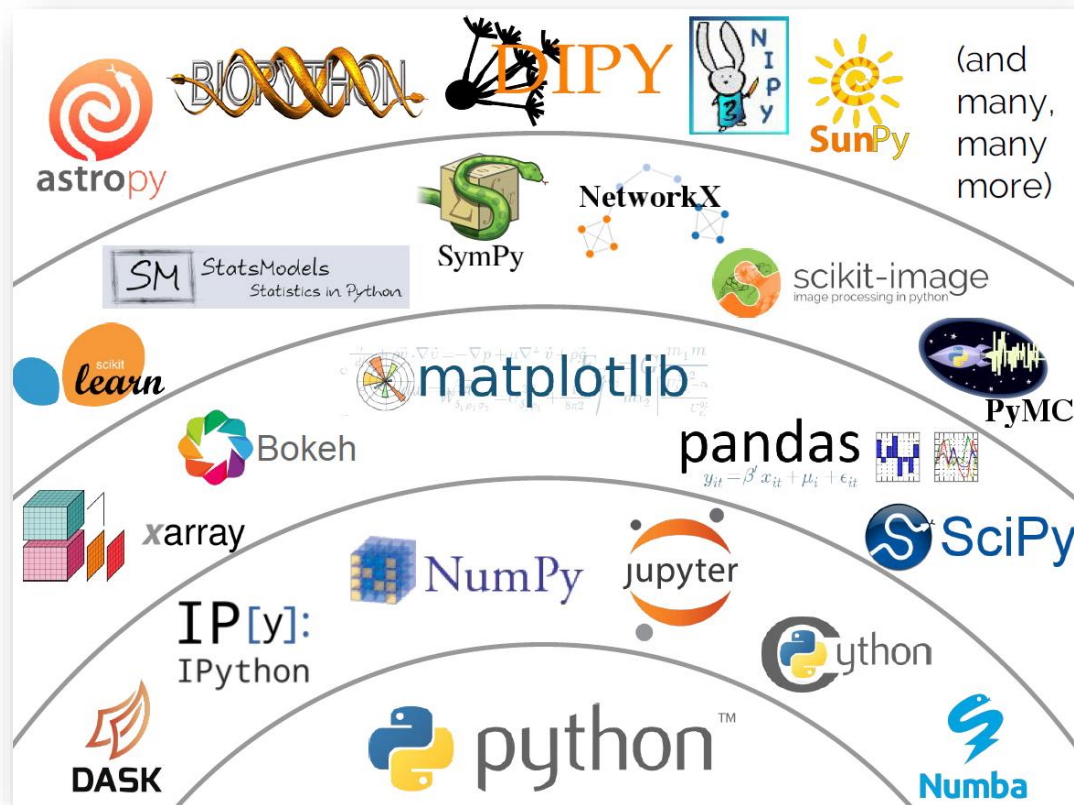
“Scikit-HEP is an open-source community-driven and community-oriented project with the goal of providing an ecosystem for particle physics data analysis in Python, fully integrated with the wider scientific Python ecosystem.

It expands the typical Python data analysis tools for particle physicists with packages spanning the spectrum from general scientific libraries for data manipulation to domain-specific libraries.”

- Fact often overlooked/misunderstood - some functionality is not available elsewhere ;-)
- More details about the project history / org were presented in the talk @ the WLCG/HSF workshop 2024, DESY, May 2024

Our contribution to the HEP domain-specific Python analysis ecosystem

- After several years we can proudly splash **our “NumPy fueled” shell ecosystem side-by-side ☺ !**
 - The full HEP ecosystem is of course wider, ROOT being prominent**, in particular – we tend to call it the **PyHEP ecosystem**



(In the mean time Coffea joined the org; it is no longer an affiliated package)



Overview of packages

- Not a description of packages
- Not attempting to be comprehensive in any aspect
- Some info left for the reader ...
- The following catches your attention?
Feel encouraged to look further at the repos ... 😊

Basics

Awkward(array)

Vector

hepunits

Data manipulation & interoperability

uproot

Coffea

uproot-browser

hepconvert

formulate

Histogramming

boost-histogram

Hist

histoprint

UHI

HEP specific libraries and interfaces to HEP libraries

Particle

DecayLanguage

fastjet

pylhe

pyhepmc

Fitting and Statistics

iminuit

pyhf

cabinetry

resample

hepstats

Visualisation

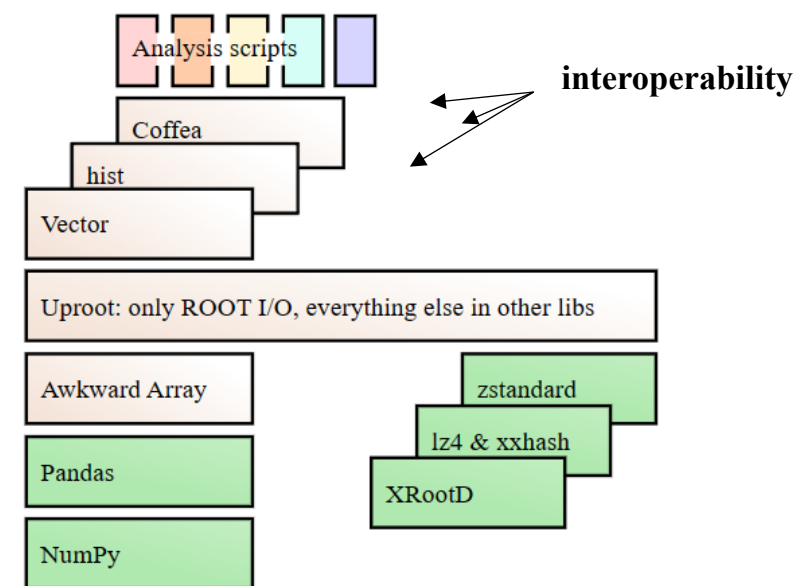
mplhep

Basics, data manipulation, interoperability

- **Data manipulation is key**
- **So is interoperability** to talk efficiently among HEP specific tools and leverage tools from the wider scientific ecosystem (NumPy-based)
- Uproot then awkward “filled the gap” and acted as an enabler / catalyst

uproot: reading and writing ROOT files in pure Python and NumPy

- Evolving towards a more robust infrastructure, including publishing via a trusted publisher
- Recent updates:
 - Several bug fixes and performance improvements have been integrated
 - In particular, **memory consumption reduced considerably**, as noticed for example in Coffea-based analyses
 - **RNtuple v1.0.0.0 read and write support** added
 - Improved integration with XRootD for enhanced stability
 - Optional support for adding C++ typenames to parameters in uproot.dask



awkward(array): library for nested, variable-sized data, including arbitrary-length lists, records, mixed types, and missing data, using NumPy-like idioms

- **Significant performance improvements** implemented, and various bug fixes released
 - Example: ~30% speedup in a full CMS analysis observed thanks to improvements in awkward (and Vector)
 - **JAX backend enhancement** and stability (important for IRIS-HEP's Analysis Grand Challenges with this backend)
 - Virtual arrays implementation
 - Named axis implementation
-
- Also, a lot of work to improve the stability of the Dask contributed package dask-awkward

Coffea: Columnar Object Framework For Effective Analysis

- The Lorentz vector math classes have been migrated to use Vector as the backend
- **Lots of developments, with direct implications for IRIS-HEP's Analysis Grand Challenges**

Vector: arrays of 2D, 3D, and Lorentz vectors

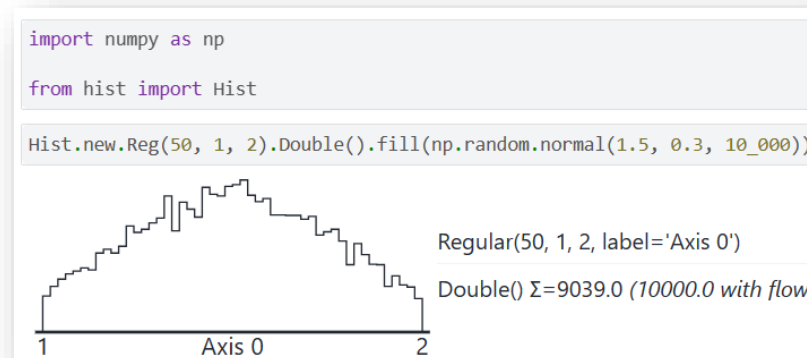
- **Optimizations for the Awkward backend**
 - Measured speedups in runtime of 2x to sometimes even >10x depending on operations
- A **new SymPy backend** to perform symbolic vector calculations
- Revamped documentation
- **Awkward v2 and dask-awkward support** added
- Allow users to extend vector classes (Coffea does it now)
- Several bug fixes and more formal physics conventions
- We published a **JOSS paper** on the package, for citation

Histogramming

- No HEP analysis without histogramming ;-) !
- **boost-histogram** – Python bindings to C++ **Boost.Histogram** triggered a lot of activity ...

boost-histogram and **Hist**: **Hist** is an analyst-friendly front-end for **boost-histogram**

- Both now **support non-uniform rebinning** (variable axis rebinning), first proposed as an extension of the Unified Histogram Interface (UHI package)



cuda-histogram:

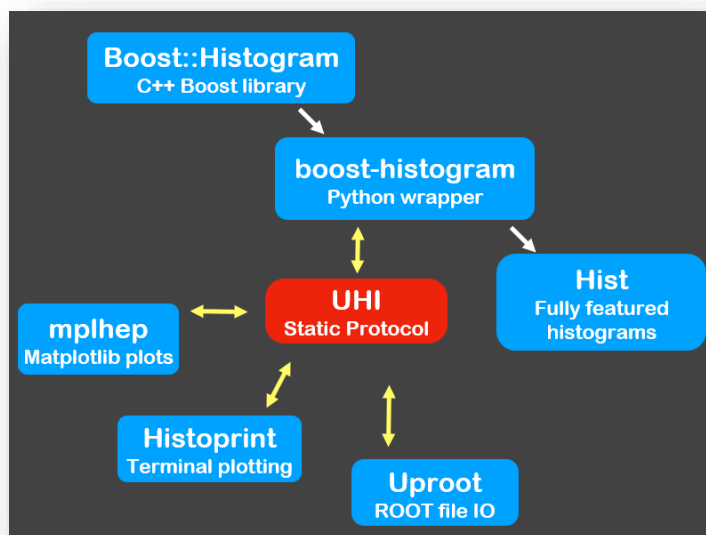
- A completely new Python package to fill histograms on GPUs (interoperable with **boost-histogram** and **Hist**)
- Experimental - still in early development but ready to be experimented with

UHI: Universal Histogram Interface package primarily for documenting histogram indexing and the PlottableHistogram Protocol (and any future cross-library standards)

- ⇒ packages can collaborate thanks to agreeing on a shared protocol !
- Very pleasing to see that ROOT is engaging with these meant-to-be common standards 👍 !
- Exchanges already triggered the development of tests for compliance
- Protocol followed by other org packages:
boost-histogram, Hist, mplhep, histoprint, uproot

PlottableHistogram

Defines expected behaviors for producers
Consumers expect these behaviors



Universal Histogramming Interface (UHI)

- [mplhep](#): python tool providing a set of helpers for matplotlib to more easily produce plots typically needed in HEP
- **UHI protocol:**
 - Draw ROOT histograms with [mplhep](#) (honouring the UHI protocol)
 - Indexing for histograms manipulation
- Both drawing and indexing available in ROOT 6.36.00
 - Thanks to Henry Schreiner for engaging with us about the less clear parts of the specification and for providing a testing suite for it!

```
import mplhep as hep
hep.style.use("LHCb2")
hep.histplot(h)
plt.title("MonHist")
```

a ROOT histogram

```
''' Access '''
v = h[b]
v = h[ROOT.loc(b)]
v = h[ROOT.loc(b) + 1]
v = h[ROOT.underflow]

''' Setting '''
h[b] = v
h[ROOT.loc(b)] = v
h[ROOT.underflow] = v
h[...] = array(...)
```

The plot shows a histogram with a title 'MonHist'. The x-axis ranges from 0 to 5, and the y-axis ranges from 0 to 6. The histogram bars are blue, and the plot is styled with 'LHCb2'.

J. Rembser | CERN EP-SFT | 6-5-25 WLCG/HSF Workshop | ROOT 16

pyhf: pure-Python HistFactory implementation with tensors and autodiff

- (Support for users can take a lot of bandwidth)
- Many developments, too many to cite here
- Related developments in cabinetry – in discussions with ATLAS and CMS, adding methods to quantify the impact of different sources of uncertainty (WIP)

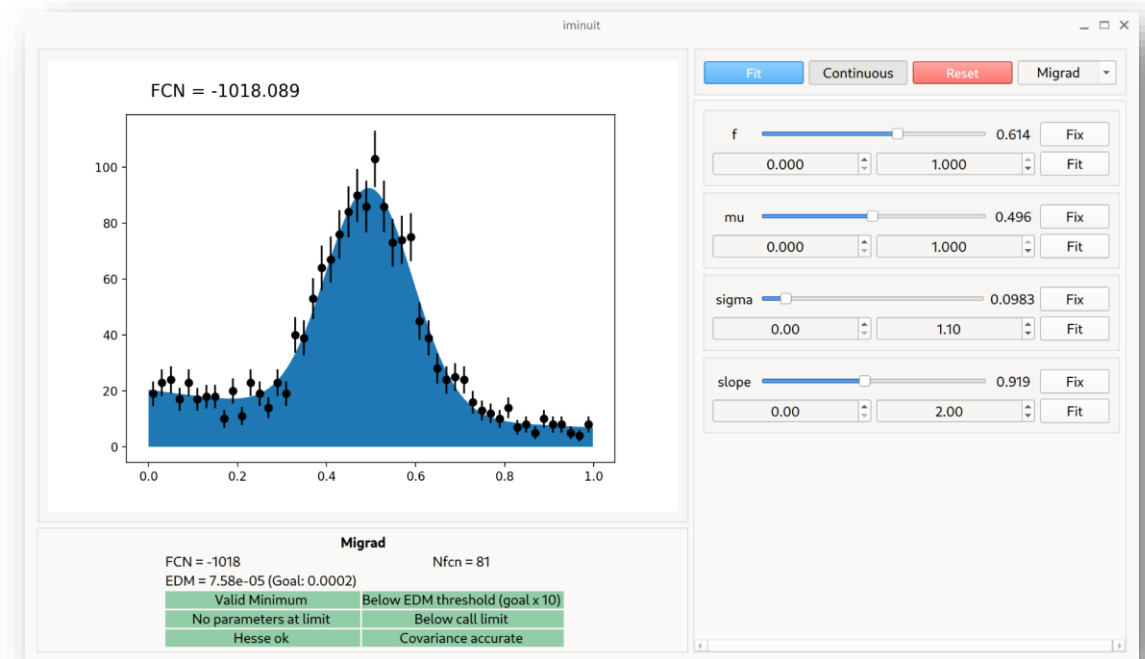
resample: resampling-based inference in Python based on data resampling and permutation

- One of few libraries that implements the "extended bootstrap" technique
- **Support for NumPy 2.x** added
- Added `resample.bootstrap.covariance` for estimators that produce a vector of outputs, which in contrast to `resample.bootstrap.variance`, also provides the correlations
- Add `resample.jackknife.cross_validation` for model selection (bias-variance trade-off)

iminuit: Jupyter-friendly Python interface for C++ MINUIT2

- Builds on the Minuit2 C++ code in ROOT
- Extensively used by other packages such as zfit, and also outside HEP, in Astro in particular
- Documentation improvements !
- Support for NumPy 2.x added
- Support for negative bin entries in cost function for weighted binned fits (important when using sWeighted histograms)

- **Minuit.interactive (interactive fitting)** now works **outside of Jupyter notebooks** based on a new Qt fitting widget
- E.g., you can call it to debug a fitting script !
- Example snapshot of this Qt-based interactive fitting window:



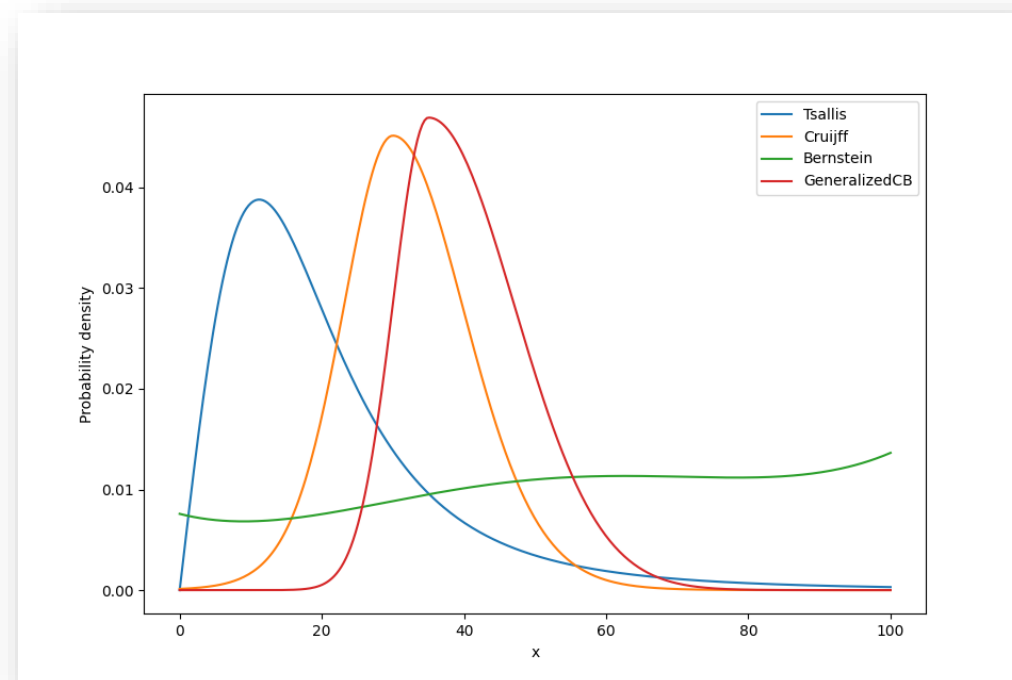


zfit: fitting library based on TensorFlow

- **New minimizers** (including Levenberg-Marquard)
- **New PDFs** to better capture commonly used shapes
- Many improvements in usability with the Python ecosystem, can easier fit arbitrary Python functions

zfit-physics: physical PDFs and more to extend zfit

- **Interconnection with RooFit, TF-PWA, pyhf, ComPWA:**
can minimize likelihood, can add likelihoods
- **New physics PDFs**
- **“zfit2” will be the future zfit: JAX mainly,**
WIP in collaboration with others



Some future directions:

- Working to adopt and contribute to the HS3 statistical representation to try to really unite the whole field on a common open statistical format
- Excellent to see a collaboration from

HS³
High Energy Physics
Statistics Serialization Standard



- Watch this space ... (A future HSF Seminar will be dedicated to the subject.)

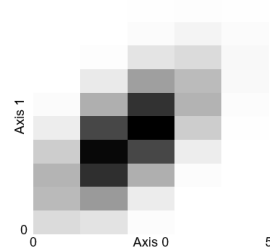
mplhep: easy matplotlib histogramming / styling for HEP plots

- **Convenient user interface for histogram plotting**

- Interface to matplotlib, fully integrated with Hist
- Can also ingest ROOT TH1/TH2

```
H # hist.Hist or ROOT.TH1/2
```

10



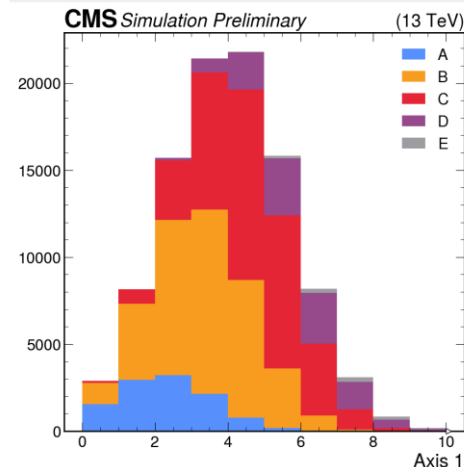
StrCategory(['A', 'B', 'C', 'D', 'E'], label='Axis 0')
Regular(10, 0, 10, label='Axis 1')

Weight() Σ =WeightedSum(value=98070, variance=98070) (WeightedSum(value=100000, variance=100000) with flow)

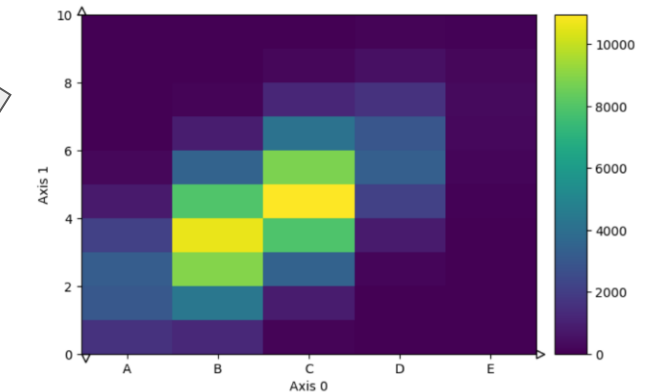
```
H.style.use("CMS")  
H.plot(stack=True, histtype='fill'); # mplhep.histplot([hid_0, ...])  
plt.legend();  
hep.cms.label("Preliminary");
```

Publication quality plots conforming to collaboration requirements (ATLAS, CMS, LHCb, ALICE) can be produced with 2 lines of code

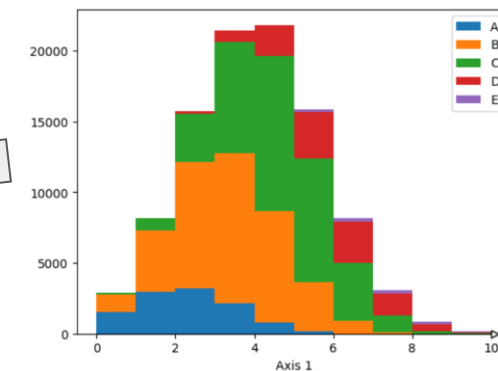
For CMS - CVD friendly color scheme (M. Petroff colors)



```
H.plot2d(); # mplhep.hist2dplot(...)
```

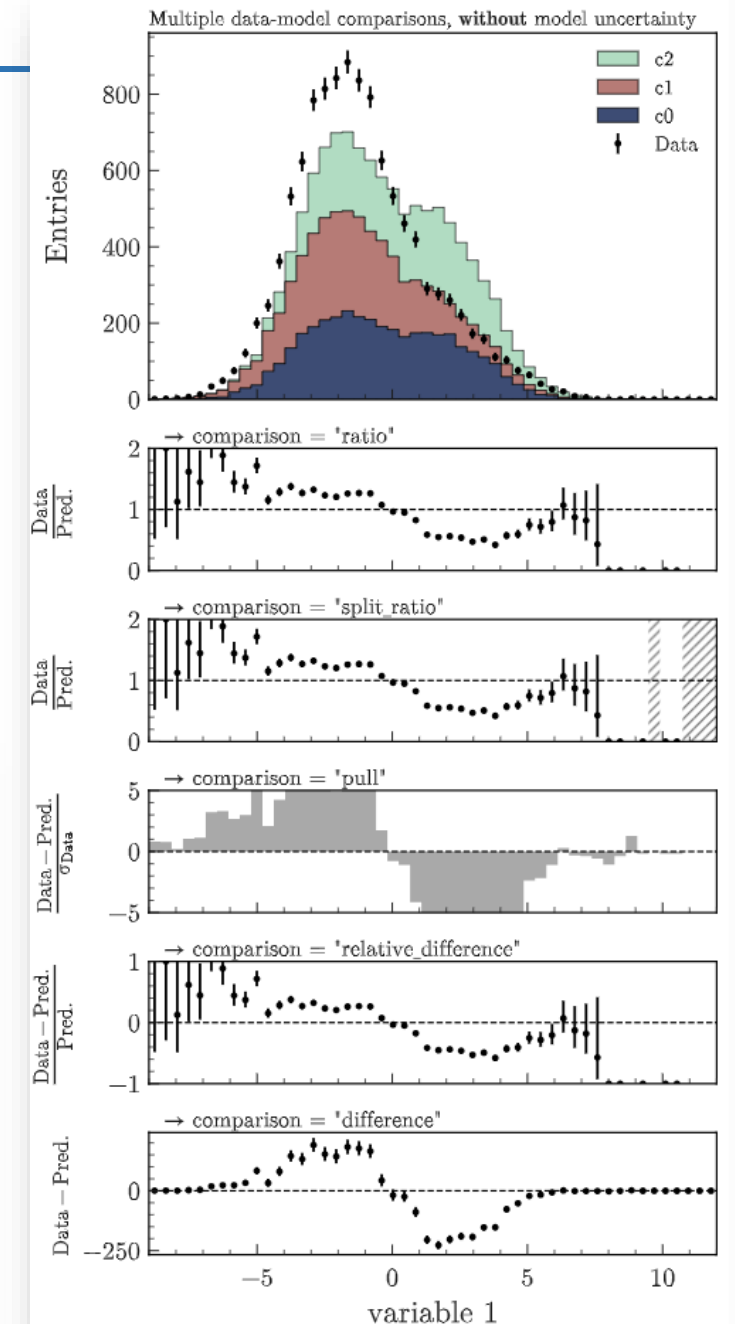
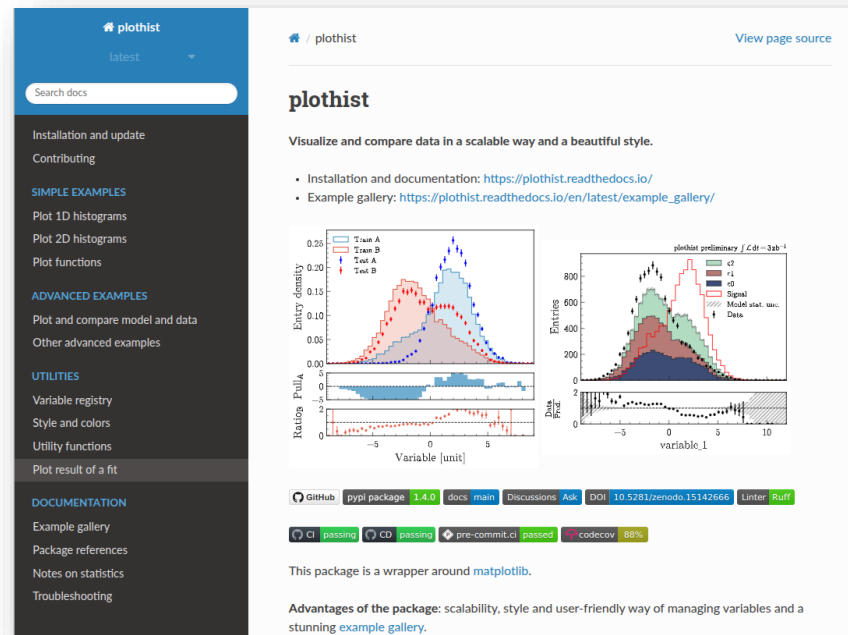


```
H.plot(stack=True, histtype='fill'); # mplhep.histplot([hid_0, ...])  
plt.legend();
```



mplhep: easy matplotlib histogramming / styling for HEP plots

- **Convenient user interface for histogram plotting**
 - Interface to matplotlib, fully integrated with Hist
 - Can also ingest ROOT TH1/TH2
- **Merging with the `plothist` package (From Belle-II colleagues) is WIP**
 - This brings the large functionality of `plothist` into `mplhep`, mostly nice out-of-the-box plots and nice documentation
 - Removes deduplications in both libraries, full UHL compatibility, clean-up also in `mplhep`
 - Brings Belle II closer into the HEP ecosystem
 - This merging/enhancing brings also 1 new maintainer 😊 !



HEP domain-specific libraries

pylhe

Lightweight Python interface to read
Les Houches Event (LHE) files

```
# Use an example LHE file from package scikit-hep-testdata
from skhep_testdata import data_path

lhe_file = data_path("pylhe-drell-yan-ll-lhe.gz")

arr = pylhe.to_awkward(pylhe.read_lhe_with_attributes(lhe_file))

arr

[{'eventinfo': {'nparticles': 4, 'pid': 1, ...}, 'particles': [...]},
 {'eventinfo': {'nparticles': 5, 'pid': 1, ...}, 'particles': [...]}]
```

fastjet

Python bindings for the C++ FastJet library.
Contains 2 interfaces – the Classic and the Awkward(array)

Find all pseudoscalar charm mesons known to the PDG:

```
Particle.findall(lambda p: p.pdgid.is_meson and p.pdgid.has_charm and p.spin_type==SpinType.PseudoScalar)
```

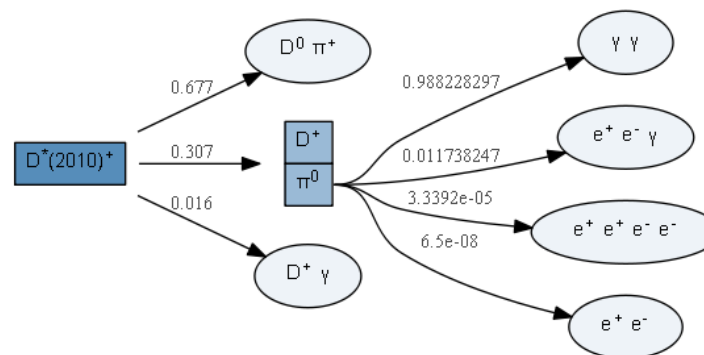
```
[<Particle: name="D+", pdgid=411, mass=1869.65 ± 0.05 MeV>,
 <Particle: name="D-", pdgid=-411, mass=1869.65 ± 0.05 MeV>,
 <Particle: name="D0", pdgid=421, mass=1864.83 ± 0.05 MeV>,
 <Particle: name="D~0", pdgid=-421, mass=1864.83 ± 0.05 MeV>,
 <Particle: name="D(s)+", pdgid=431, mass=1968.34 ± 0.07 MeV>,
 <Particle: name="D(s)-", pdgid=-431, mass=1968.34 ± 0.07 MeV>,
 <Particle: name="eta(c)(1S)", pdgid=441, mass=2983.9 ± 0.5 MeV>,
 <Particle: name="B(c)+", pdgid=541, mass=6274.9 ± 0.8 MeV>,
 <Particle: name="B(c)-", pdgid=-541, mass=6274.9 ± 0.8 MeV>,
 <Particle: name="eta(c)(2S)", pdgid=100441, mass=3637.5 ± 1.1 MeV>]
```

Particle

PDG particle data
and identification codes, etc.

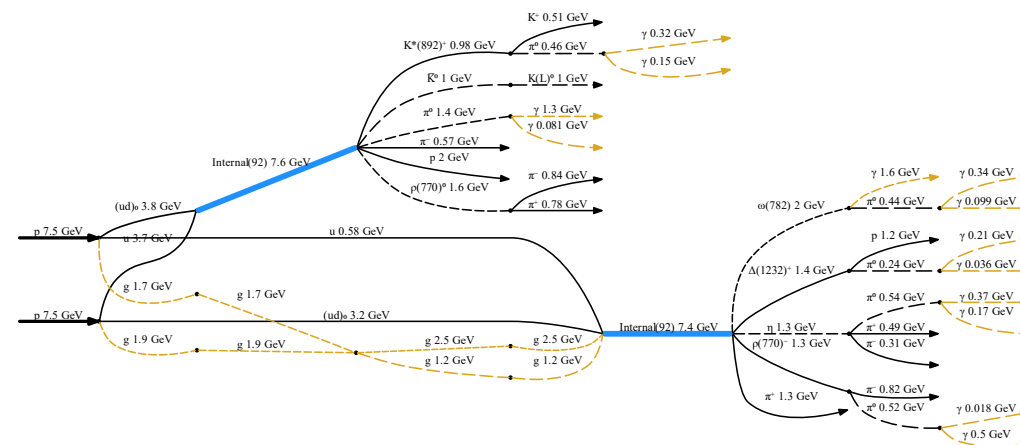
DecayLanguage

Describe, manipulate and convert particle decays



pyhepmc

Easy-to-use Python bindings for HepMC3



In- and out-of-HEP community engagement

Engagement via training, training material, workshops/conferences

- **Training material on Scikit-HEP packages contributed to the HSF (Training group) courses**
- Maintaining and developing such material also helps maintaining the packages and ensuring they evolve the direction the community prefers
- Worth emphasising fact – it's community colleagues who take time to deliver these courses on ROOT, Scikit-HEP, etc. !




<https://github.com/hsf-training/hsf-training-scikit-hep-webpage>

- **Active participation in and in fact organisation of the PyHEP(.dev) workshops**
 - **Excellent fora to discuss and plan a coherent “PyHEP ecosystem” roadmap and make priorities for the upcoming year**

Engagement in the wider scientific Python community

- Check out <https://learn.scientific-python.org/>



Repo-Review

You can check the style of a GitHub repository below. Enter any repository, such as `scikit-hep/hist`, and the branch you want to check, such as `main` (it must exist). This will produce a list of results - green checkmarks mean this rule is followed, red errors mean the rule is not. A yellow warning sign means that the check was skipped because a previous required check failed. Some checks will fail, that's okay - the goal is bring all possible issues to your attention, not to force compliance with arbitrary checks.

You can also run [this tool](#) locally (Python 3.10+ required):

```
pipx run 'sp-repo-review[cli]' <path to repo>
```

Org/Repo
e.g. scikit-hep/hist

Branch
e.g. main

I→

Enter a GitHub repo and branch to review. Runs Python entirely in your browser using WebAssembly. Built with React, MaterialUI, and Pyodide. Packages: repo-review~=0.10.0 sp-repo-review==2024.03.10 validate-pyproject-schema-store==2024.03.11 validate-pyproject[all]~=0.16.0

Related Resources

This guide does *not* cover the basics of Python itself or the scientific Python libraries; it focuses on making or maintaining a package. We recommend the [Scientific Python Lectures](#) if you want info.

This guide also does not cover version control, but it is essential to have a basic facility with git to use these tools successfully. We recommend the [Software Carpentry lesson on Git](#).

HISTORY

This guide (along with cookie & repo-review) started in [Scikit-HEP](#) in 2020. It was merged with the [NSLS-II](#) guidelines and moved to Scientific Python at the [2023 Scientific Python Developer Summit](#), along with many updates. Improved support for compiled components supported in part by NSF grant [OAC-2209877](#).

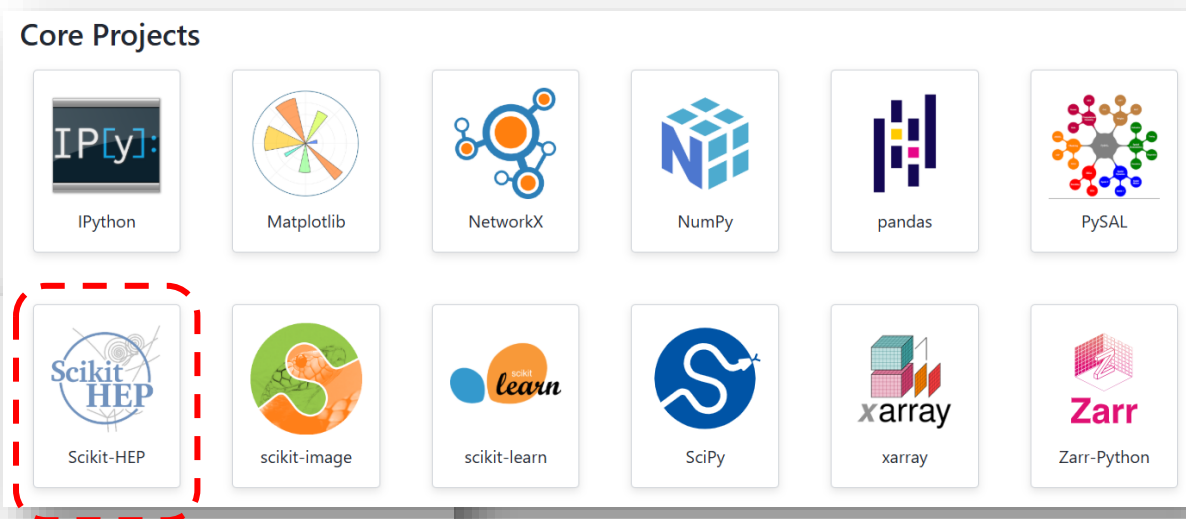
- Does your library follow the guidelines ?
- Repo-Review is your friend ...

Engagement in the wider scientific Python community

Scientific Python 

<https://scientific-python.org/>

- Ongoing engagement with Scientific Python (also SciPy)
- **April 2025:**
Scikit-HEP joins the Scientific Python Ecosystem Coordination Core Projects as 1 of the first 2 domain specific stacks accepted
- **Present list of core projects,**
see Scientific Python - SPEC Core Projects:



Scientific Python Ecosystem Coordination > SPEC Core Projects

SPEC Core Projects

Description

Core Projects are depended upon by many other projects, and often provide basic data structures, drawing primitives, implementations of fundamental algorithms, or are metapackages that represent a particular scientific field. Due to their central position in the ecosystem, the policies, practices, and tooling used by the Core Projects are widely seen by the ecosystem and impact many other projects. The [Steering Committee](#) maintains the list of Core Projects.

Core Projects endorse SPEC documents. During the endorsement stage of the SPEC process, Core Project contributors propose, discuss, and review SPEC documents with the goal of developing a coherent implementation plan suitable for all the Core Projects. Often SPECs are coauthored by contributors from several Core Projects as well as other community members (e.g., contributors to other ecosystem projects).

Aspects of maintainability and sustainability

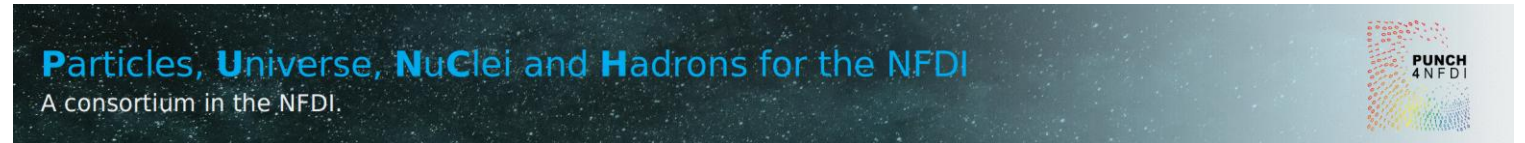
Measuring relevance & impact – «projects» (in HEP) using Scikit-HEP



Sustainable software ↔ sustain/maintain the people that develop the software!

- When the project started in late 2016, much of the (initial) challenge was in attracting (1) people and (2) relevant packages
- Today, much of the challenge remains to attract people and especially (new, younger) maintainers and developers !
- This is, of course, rather general an issue, not specific to the Scikit-HEP project
- It is the ecosystem surrounding a piece of software, built not only by original authors but also by community members, that forms a long-term hence sustainable solution in a continuously evolving space
- Without adequate support, career path and funding, nothing is sustainable !
- Small package granularity – a set of “building blocks” and well-scoped generic or domain-specific libraries providing tools that address a well-defined class of problems at one level of abstraction has been a success story
 - Increasing number of projects/libraries build on top of these
- BTW, sustaining a “toolset project” does not require to maintain and keep all packages alive “forever”
- Along the way, several packages got archived, superseded by better products !
 - Example: root_numpy and root_pandas replaced (years ago) with uproot + awkward
 - It helped enormously that packages are scoped, hence an “deprecate, archive and replace” hasn’t been a killer for users to the best of our knowledge

Concluding remarks – Scikit-HEP and PUNCH4NFDI Consortium



- **Scikit-HEP** very likely provides useful libraries and functionality to you
- We would be most **happy to receive feedback**
- Conversely, **you can be a contributor** – do not hesitate to look around at the repositories and get in touch 😊 !
- **We're very much interested in your contributions !**

Thank you for listening !

Showcasing how the various packages work together

- **SciPy 2024 contribution paper describing to a broad audience how a large scientific collaboration leverages the power of the Scientific Python ecosystem to tackle domain-specific challenges and advance our understanding of the Cosmos**
 - Through a simplified example of the renowned Higgs boson discovery
 - With a **Jupyter notebook** showing the various steps from data cleaning, statistical interpretation and visualization at scale, with many **Scikit-HEP packages and others as well (Dask, etc.)**

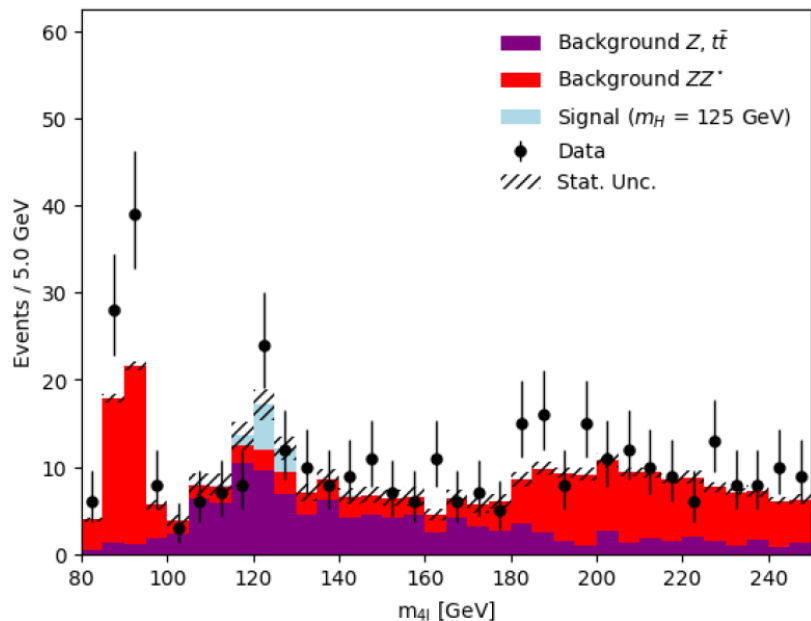


Figure 3. Using `mplhep`, `hist`, and `matplotlib` the post-processed histograms of the simulation and the data are visualized in advance of any statistical inference of best-fit model parameters.



SciPy 2024

July 8 - July 14, 2024

Proceedings of the 23rd
Python in Science Conference
ISSN: 2575-9752

How the Scientific Python ecosystem helps answer fundamental questions of the Universe

Matthew Feickert¹, Nikolai Hartmann², Lukas Heinrich³,
Alexander Held¹, Vangelis Kourlitis³, Nils Krumnack⁴, Giordon Stark⁵,
Matthias Vigl³, and Gordon Watts⁶

¹University of Wisconsin--Madison, ²Ludwig Maximilians Universitat, ³Technical University of Munich, ⁴Iowa State University, ⁵Santa Cruz Institute for Particle Physics, ⁶University of Washington

Abstract

The ATLAS experiment at CERN explores vast amounts of physics data to answer the most fundamental questions of the Universe. The prevalence of Python in scientific computing motivated ATLAS to adopt it for its data analysis workflows while enhancing users' experience. This paper will describe to a broad audience how a large scientific collaboration leverages the power of the Scientific Python ecosystem to tackle domain-specific challenges and advance our understanding of the Cosmos. Through a simplified example of the renowned Higgs boson discovery, attendees will gain insights into the utilization of Python libraries to discriminate a signal in immersive noise, through tasks such as data cleaning, feature engineering, statistical interpretation and visualization at scale.

<https://doi.org/10.25080/KMXN4784>

<https://github.com/ekourlit/scipy2024-ATLAS-demo>

A nice-to-see example of the “PyHEP ecosystem”

- **Code snippet with ROOT, some of Scikit-HEP, and seaborn (wider scientific Python library)**
 - BTW, you could mix/swap which libraries to use for each snippet “section”

