ROOT's Newest Pythonizations: UHI, RDataFrame and More

Silia Taider, Vincenzo Eduardo Padulano

PyHEP 2025
27-10-2025

https://root.cern

- **Python** is widely used in today's HEP analysis ecosystem
- ROOT evolves toward a **Pythonic** user experience by focusing on:

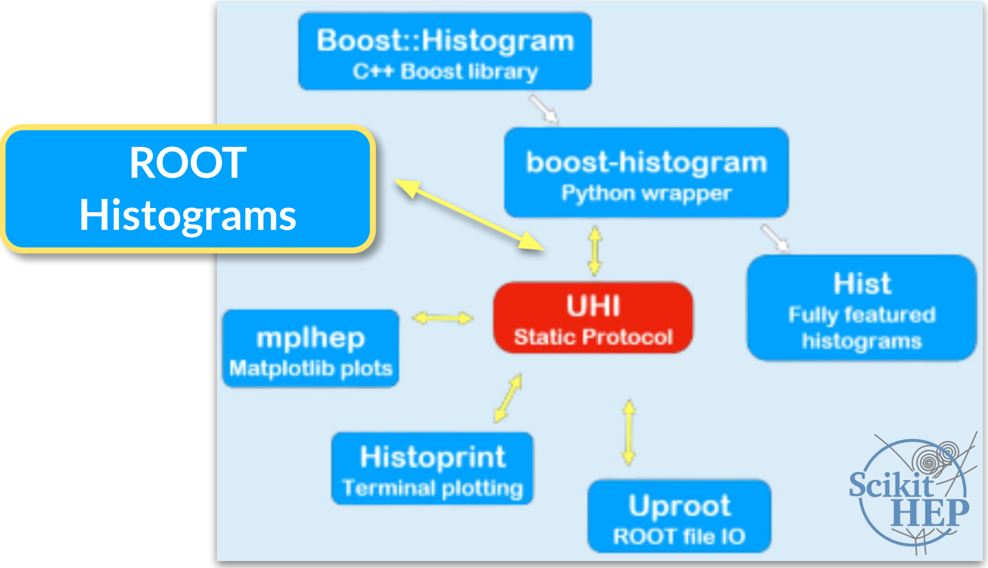
Interoperability | Usability | Distribution

*We will cover one example of each

Interoperability: UHI

ROOT Now Implements the UHI

- The Unified Histogram Interface (UHI) is a specification to unify histogram operations across Python libraries
- **ROOT adopted UHI in v6.36**
  - Thanks to H. Schreiner for engaging with us



WLCG/HSF Workshop 2024, E. Rodrigues, The Scikit-HEP project overview and future

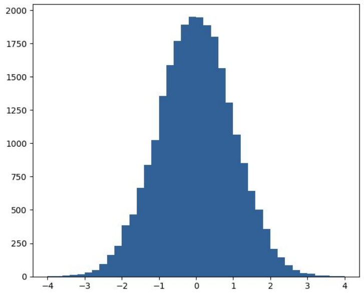
- UHI introduces **NumPy-like indexing for ROOT histograms, like slicing and projections**

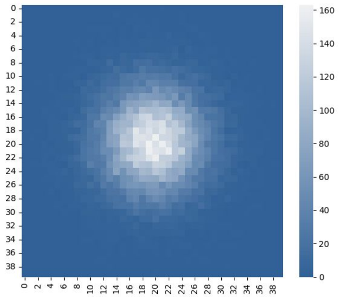
UHI Syntax	ROOT Syntax
`h1[underflow]`	`h1.GetBinContent(0)`
`h1[-1] = 3`	`h1.SetBinContent(nbins, 3)`
`h2[::, ::sum]`	`h.ProjectionX()`
`h2[:5, ...]`	❌

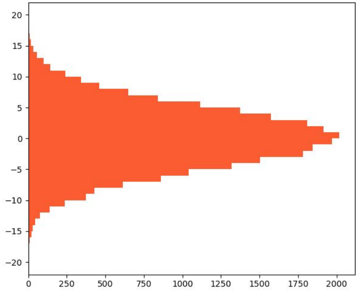
```
# slice the first 10 bins of X
h3[:10, ...]
# slice and rebin X, project Y
h3[10:20:rebin(2), ::sum, ::]
# rebin Z after selecting X = 3
h3[{2: np.s_[::rebin(5)]}]
```

Your histogram manipulation becomes expressive, intuitive, and Pythonic

- UHI enables **visualizing ROOT histograms using Python's plotting ecosystem**







Full code here

```python
# h2 is ROOT.TH2F
sns.heatmap(h2.values(), ...)
# projh2X is a ROOT.TH1D
hep.histplot(projh2X, ...)
...
```

You can now directly plot ROOT histograms with UHI-compatible libraries

On top of the existing

417 UHI tests

ROOT adopted the latest

[UHI testing helper](#) *-a set of cross-library tests-*

Thanks for the additional 72 tests!

Usability: RDataFrame

https://root.cern

Our Goal:

Enable fully Pythonic analysis With ROOT

-Today I show advancements in the RDataFrame Pythonic API-

RDataFrame Pythonic API Today

Legend: ✅ Supported ⚠️ Partial ❌ Not supported

- Use Numba to JIT-compile Python functions directly inside the RDataFrame event loop

```python
def invariant_mass(px, py, pz, E):
    return np.sqrt(E**2 - (px**2 + py**2 + pz**2))

rdf = (ROOT.RDataFrame("tree", "file.root")
       .Define("mass", invariant_mass))
```

Types of inputs and outputs	Support
Fundamental types	✅
C++ containers (ROOT::RVec, std::vector, std::array)	✅
ROOT Classes	⚠️ Limited by the cppyy -our dynamic Python-C++ bindings generator- numba_ext
NumPy arrays	✅
Awkward arrays	⚠️
User-defined Python classes	❌

Distribution: conda & pip

https://root.cern

ROOT Python Distributions

- **To make the python users experience smooth, we standardize the python distributions**
- **Conda: all ROOT releases are published on conda-forge, for Linux and MacOS.**
  - Thanks to the initial immense effort to bring this distribution into production by H. Schreiner and C. Burr

Try it yourself!

```
$ conda install -c conda-forge ROOT
```

- **pip: first presented at CHEP'24, since then:**
  - ROOT Python wheels for Linux x86-64 are built nightly by CI
  - Aiming to publish first pip-installable ROOT release with 6.38

Try it yourself!

```
$ pip install ROOT
```

Conclusion

https://root.cern

Conclusion

- We covered one **example** of each axis of ROOT's Pythonization efforts:
  - **Interoperability** through the UHI implementation
  - **Usability** through RDataFrame's Pythonic API
  - **Distribution** through Conda and pip

Many more efforts are ongoing across ROOT to make it feel natural and expressive in the Python ecosystem

Thoughts?

https://root.cern

Backup

RDataFrame Pythonic API Today

- Use functions from your library to define and filter RDataFrame columns

```
ROOT.gInterpreter.Declare("""
struct MyStruct {
   int value;
   MyStruct() : value(0) {}
   MyStruct(int v) : value(v) {}
};

template <typename T>
T make_struct_t(int x) {
   return T(x);
}
""")

make_mystruct = ROOT.make_struct_t[ROOT.MyStruct]
rdf = ROOT.RDataFrame(3) \
   .Define("x", "(int)rdfentry_") \
   .Define("mystruct_col", make_mystruct, ["x"])
```