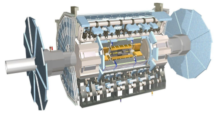
Introduction to Data Visualization in ROOT and MATPLOTLib

Chilufya Mwewa and Dominic Stafford

Introduction to the Terascale

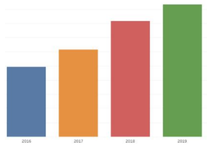
17 March 2025

Why do we need visualization in HEP?



HEP experiments generate enormous amounts of data (Petabytes!)

- ◆ Raw numbers alone may hide important trends in the data



ROOT and MATPLOTLib are important for the visual representation of the data

- ◆ Produce highly customizable, publication-ready plots

➔ With visualization tools;

- ◆ We emphasise the characteristics of the data
- ◆ We analyse its features
- ◆ We can compare and communicate results effectively
- ◆ We make informed decisions on the data

★ HEP discoveries come from identifying patterns in the data

ROOT and Matplotlib

ROOT	Matplotlib
Designed for HEP analysis	General purpose. Commonly used in data science
C++ (with python bindings)	Purely python
Optimized for large dataset storage and analysis (HEP data is stored in root format)	Uses tools like uproot to convert root files to numpy arrays
Widely used in HEP and supported by CERN	Becoming popular within the HEP community

Introduction to ROOT

ROOT
Data Analysis Framework

*Lot of material borrowed from Tadej's previous tutorials

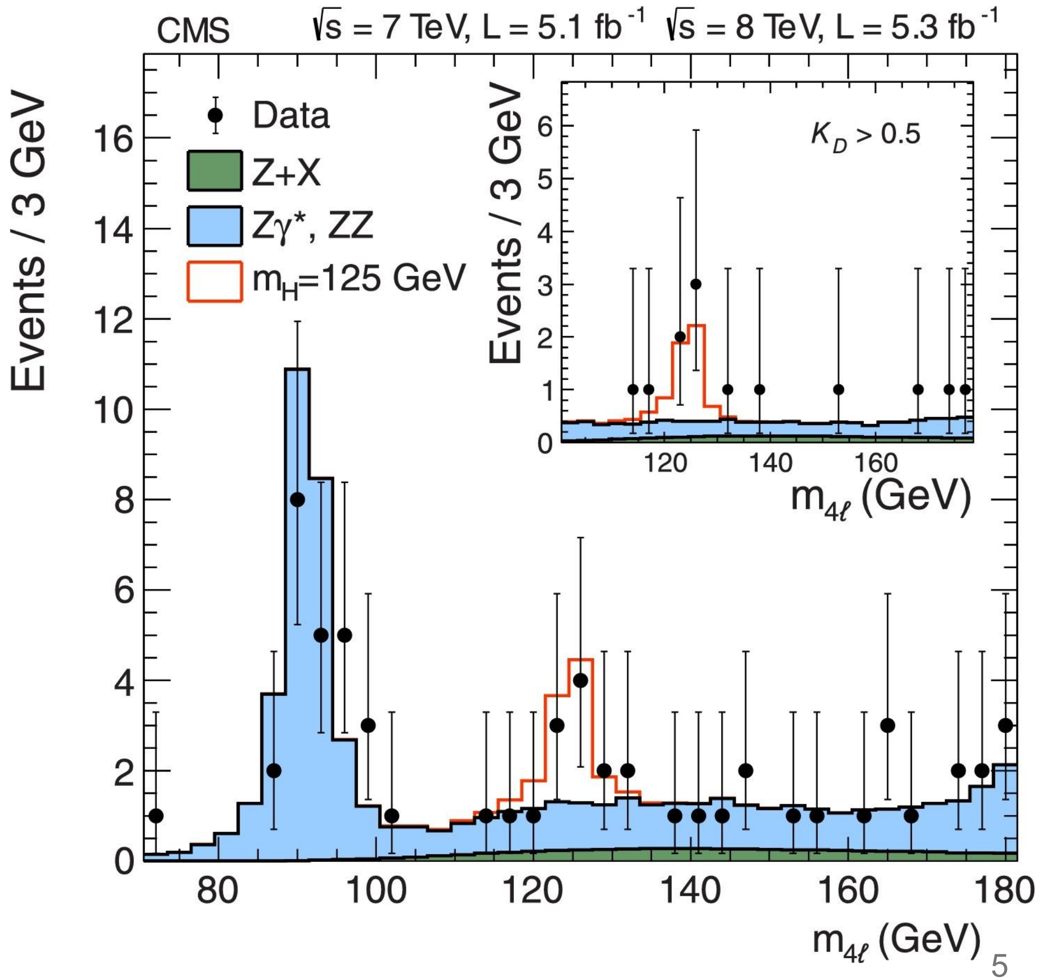
ROOT in a nutshell

❖ ROOT is a powerful data analysis framework designed for “big data” storage and analysis

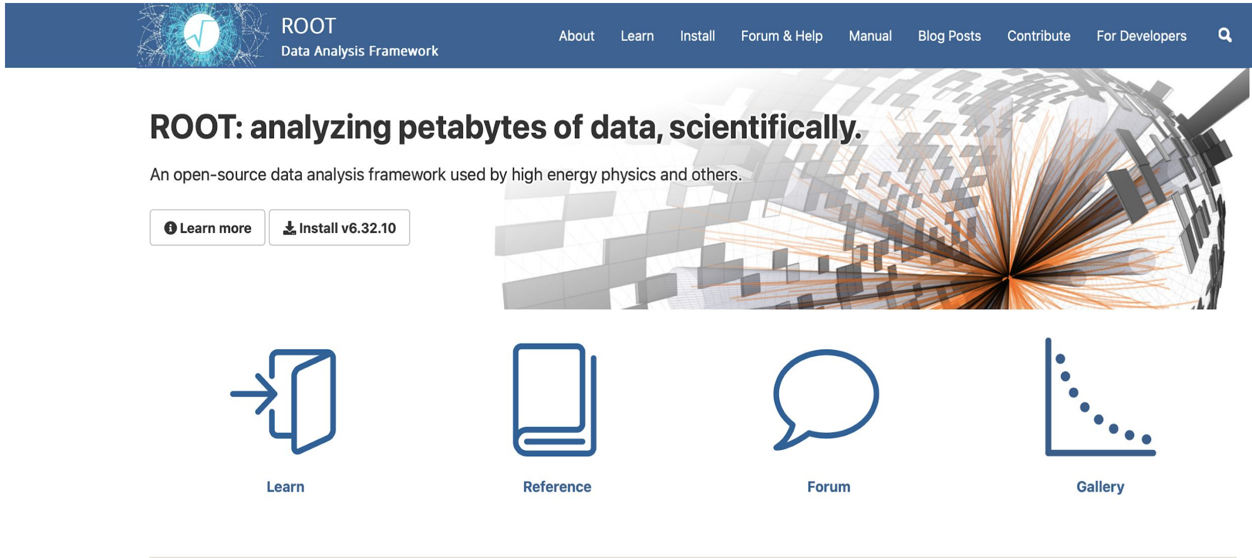
❖ It’s main functions include;

- ➔ Storage
  - ◆ Data from HEP experiments are stored in `.root` files
- ➔ Data analysis
  - ◆ Math libraries
  - ◆ Libraries for statistical analysis
  - ◆ Machine learning libraries
- ➔ Data visualization
  - ◆ Plotting tools

❖ The Higgs boson was discovered with ROOT!



ROOT resources



- ➔ ROOT website: https://root.cern
  - ◆ Main reference for all ROOT users
- ➔ Documentation: https://root.cern/doc/master/
- ➔ ROOT forum: https://root-forum.cern.ch
  - ◆ Active platform for any questions you might have

Our goals for today

- ➔ Use the ROOT prompt to inspect data stored in a `.root` file
- ➔ Use the ROOT python wrapper (pyroot) to inspect the data
- ➔ Use pyroot to make plots and perform some basic analysis of the data
- ➔ Learn how to style and save plots
- ➔ Learn how to navigate the ROOT documentation

Practical information for our task

- ➔ Log in to one of the NAF nodes (or stay on your local laptop area if you successfully installed ROOT)
  - ◆ $ ssh -Y school60@naf-school.desy.de
- ➔ Create directory where you will work from
  - ◆ $ mkdir <your_directory>
- ➔ Navigate to that directory
  - ◆ $ cd <your_directory>
- ➔ Get the root file which contains events with muon pairs from a Z boson decay
  - ◆ $ wget http://desy.de/~mwewachi/ROOT_tutorial_2025/Zmumu.root

The ROOT prompt

- ➔ Open ROOT interactively
  - ◆ $ root
- ➔ ROOT provides a C++ interpreter
  - ◆ Interactive C++, no need for a compiler
- ➔ Use C++ syntax when using the prompt

```
root [0] double N, x = 0.5
(double) 0.50000000
root [1] for (int i=0;i<10;++i) N += pow(x,i)
root [2] N
(double) 1.9980469
```

The Tfile and TTree class

➔ The TFile class manages I/O in ROOT (e.g creation and reading of `.root` files)

pyroot: file = TFile.Open("Zmumu.root") (or: with TFile("Zmumu.root") as f:)

C++: TFile::Open("Zmumu.root")

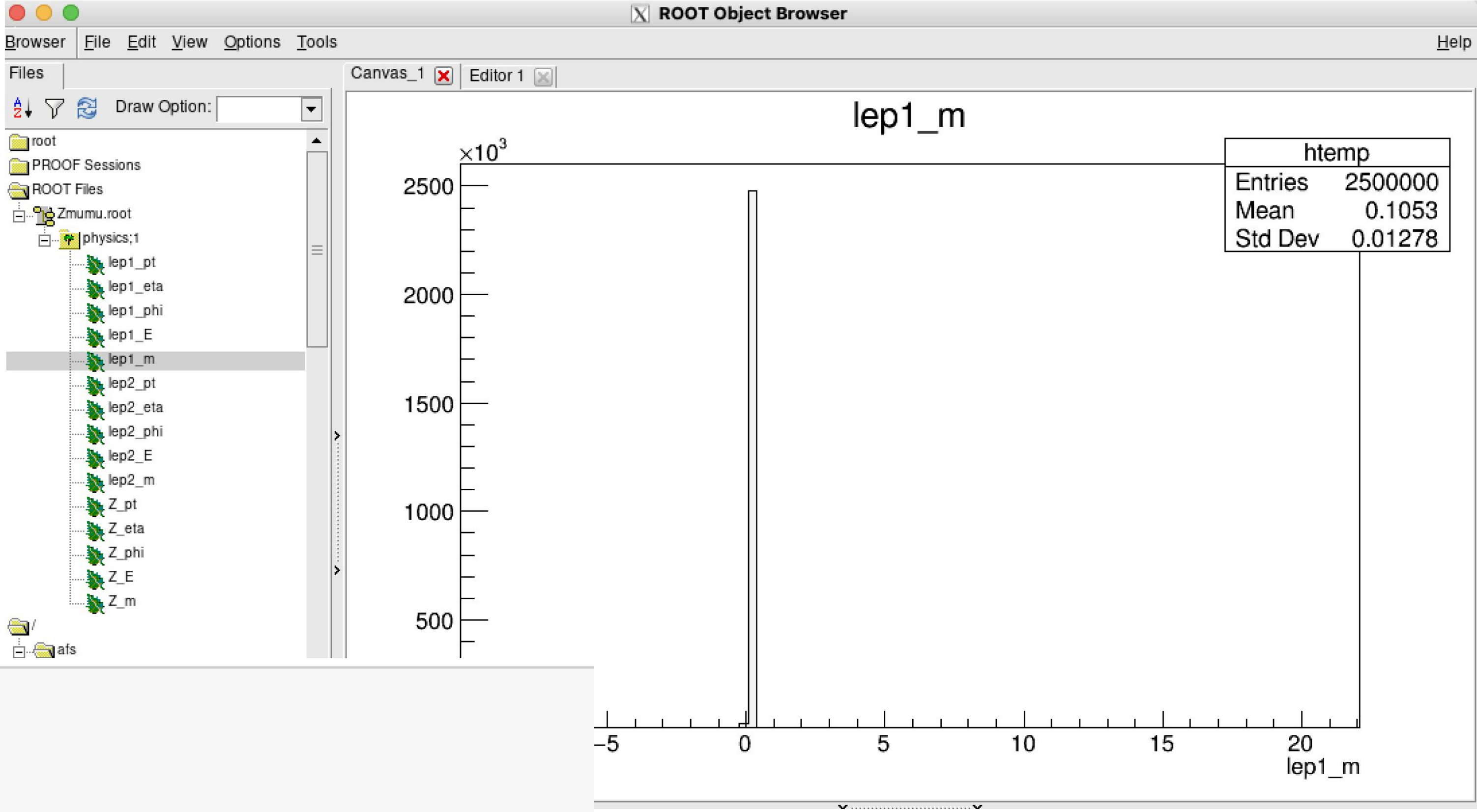
➔ A TTree (or simply "tree") represents the data structure of `.root` files

- Independent columns with identical data structures
- To load a specific tree from the TFile:

my_tree = file.Get("<tree name>")

The ROOT browser (Tbrowser class)

- ➔ Nice way to inspect the contents and structure of data in a TFile
- ➔ Quit ROOT for a second and re-open it with the root file attached
  - ◆ .q
  - ◆ root --web=off Zmumu.root
- ➔ Call the TBrowser to look at the structure and contents of the file
  - ◆ new TBrowser




```python
import ROOT
Zmumu_file = ROOT.TFile("Zmumu.root")
physics_tree = Zmumu_file.Get("physics")

print(f"Number of events in this root file: {physics_tree.GetEntries()}")
physics_tree.Print()
```

Histogram classes (TH)

➔ Show distribution/frequency of the data

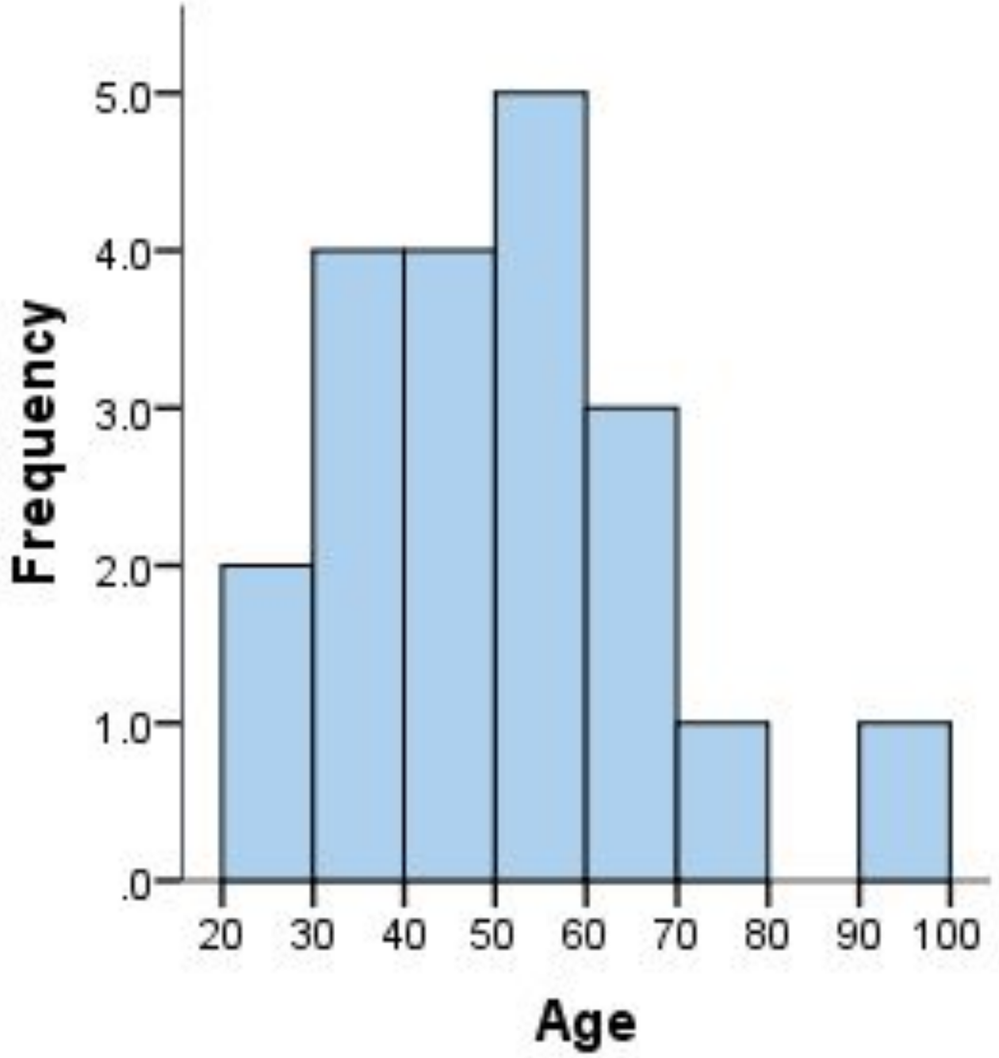
- ◆ Information is sorted in bins

➔ ROOT classes:

- ◆ 1 dimensional histogram: TH1
- ◆ 2 dimensional histogram: TH2
- ◆ 3 dimensional histogram: TH3

➔ Histogram types:

- ◆ TH1F: bins of type "float"
- ◆ TH1D: bins of type "double"
- ◆ TH1I: bins of type "integer"



More in [TH1 class reference]

Summary of relevant ROOT classes

```python
import ROOT
```

- TFile: class for reading/writing files

```python
#Opening a ROOT file
file = ROOT.TFile("Zmumu.root")
```

- TTree: basic storage format in ROOT

```python
#Get a tree from a ROOT file and store it in a TTree object
tree = file.Get("physics")
```

- TH<x>: base class for histograms
  - <x> = 1,2,3-D histograms

```python
#1-D histogram
my_histogram = ROOT.TH1D("myHist", "my first ROOT histogram", 100, 0, 1000)
```

```python
#Filling a histogram directly from an existing tree's branch
tree.Draw("variable >> myHist")
```

```python
#Change the color of a histogram
my_histogram.SetLineColor(ROOT.kRed)
```

- TCanvas: class for graphical display

```python
my_canvas = ROOT.TCanvas()
```

```python
#Draw the histogram on the canvas
my_histogram.Draw()
```

```python
#Save the canvas as a pdf
my_canvas.SaveAs("histogram.pdf")
```

- TLegend: add legend to a canvas

```python
legend = TLegend(0.7, 0.75, 0.90, 0.87)
legend.AddEntry(hist1, hist1.GetTitle(), "l")
```

Exercises

Exercise 1: Histogram Drawing

➔ Write a python macro ExerciseHist.py

1. Open the Zmumu.root file and load the physics tree
2. Create two histograms with 40 bins ranging from 0 to 200 to plot the transverse momentum (pt) of the muons
3. Fill the histograms with leading and subleading muon pT from branches lep1_pt and lep2_pt
4. Calculate the mean value and RMS of each histogram
5. Calculate the integral of each histogram

Bonus tasks

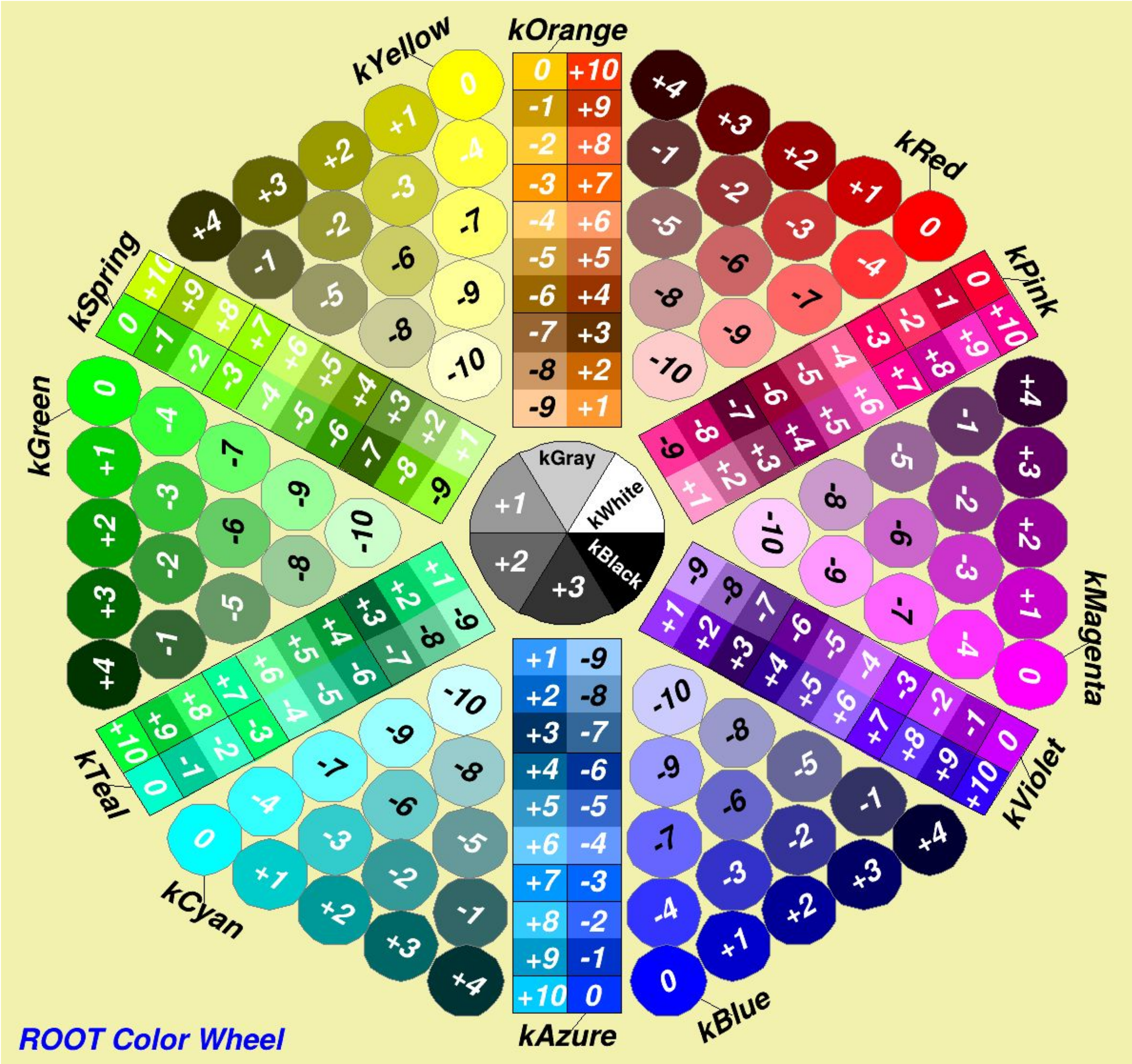
1. Change the color of each histogram
2. Write the output to a file

Exercise 2: Canvas and Legends

➔ Write a python macro ExerciseCanvas.py

1. Create two histograms with 50 bins ranging from -3 to 3 with two different names.
2. Fill first histogram with Gaussian distribution with 10000 entries.
3. Fill second histogram with a second order polynomial and 5000 entries (hint: hist.FillRandom("pol2", 5000)).
4. Set the line color of the first histogram to kRed and second to kBlue.
5. Draw both histograms on the same canvas.
6. Clone both histograms and normalise them (scale with inverse of the integral).
7. Draw normalised histograms on a different canvas.
8. Draw a legend on both canvases at position (0.7, 0.75, 0.90, 0.87)
9. Save both canvases as PDF.

ROOT Color Wheel



MATPLOTLib and the Scikit-HEP ecosystem

matplotlib

mplhep

Scikit HEP

NumPy

Awkward Array

uproot

Data processing in python

- Python has its own ecosystem of data processing tools, e.g.:
  - numpy, scipy
  - ML tools (keras, pytorch, tensorflow)
  - matplotlib
- Advantages:
  - Easier to use and maintain (python experience more common than C++)
  - Faster than Python for loops
  - industry supported tools (good documentation)
- Challenges:
  - Accessing data in ROOT-based formats
  - HEP data is usually “ragged” (different numbers of particles per event)
  - Matplotlib/numpy histograms are not usually sufficient for HEP

The Scikit-HEP ecosystem

- The Scikit-HEP project extends these tools to HEP needs:
  - Uproot opens root files
  - Awkward offers numpy-like processing for “ragged” HEP data
  - Hist provides improved histogramming
  - MPL-HEP adds utilities for plot styles
- These packages are increasingly being used for a wide range of analyses
  - Easy to learn, particularly if already familiar with numpy, etc.
  - Fast processing times
- However not suited to all applications, particularly at earlier processing stages

Exercise

Open SHARE/Intro_to_Terascale/Matplotlib_plotting_exercise.ipynb
on jupyter.desy.de, and follow the exercises