Improving the ROOT Data Analysis Framework

Julia Mathé
supervised by Enrico Guiraud and Ivan Donchev Kabadzhov

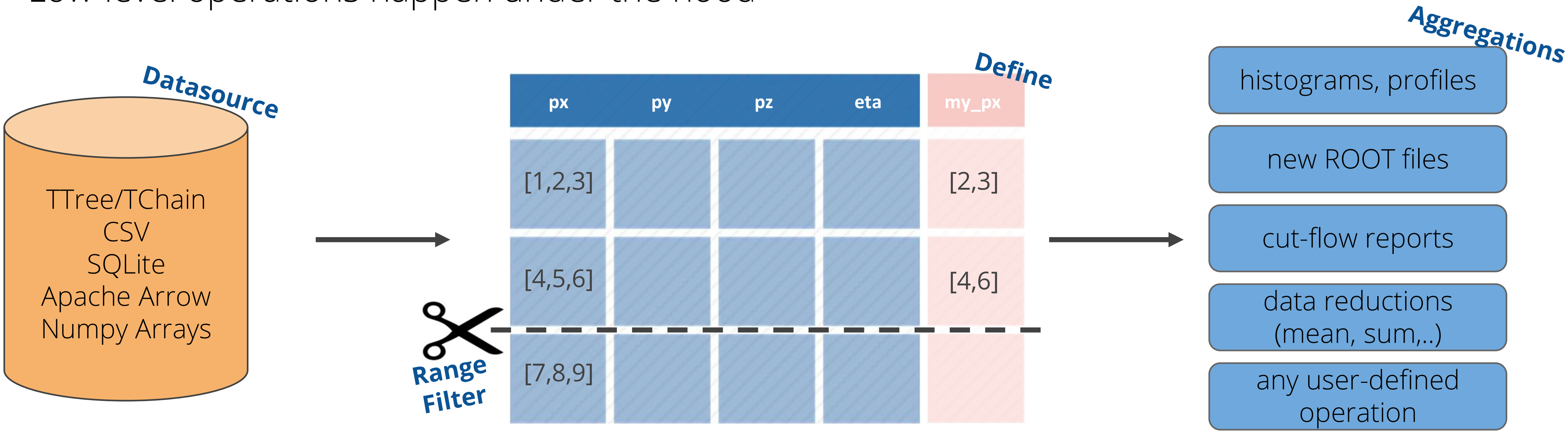
2022/09/05

Introduction into RDataFrame

RDataFrame is a class that provides a high-level interface for HEP data analysis usecases

Basic operations for the user are **Transformations** and **Actions**

Low-level operations happen under the hood

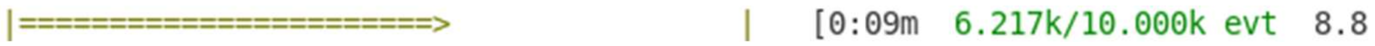


px	py	pz	eta	my_px
[1,2,3]				[2,3]
[4,5,6]				[4,6]
[7,8,9]				

A new feature: The RDF Progress bar

ROOT users

- How long will my analysis take? At which rate are my events being processed?



```
|=====================>                         |   [0:09m  6.217k/10.000k evt  8.8
```

→ Progress bar using a ProgressHelper class

- Project initiated by Stephan Hageboeck
- (Thread-safe) dataframe call-backs every n events
- Update print-out (progress bar + statistics) every m second
- Time estimation from running mean of events/sec, current event count and total number of events

Set up of the RDF Progress bar

- Old set up

C++

```
ROOT::RDataFrame df(“Events”, “f.root”);
ROOT::RDF::ProgressHelper progress{1000};
df.DefinePerSample(“_progressbar”,
                [&progress] (unsigned int slot, const ROOT::RDF::RSampleInfo & id) -> std::size_t{
progress.registerNewSample(slot, id); return progress.ComputeMaxEvents(); });
df.Count().OnPartialResultSlot(1000, [&](unsigned int slot, auto && arg){ progress(slot, arg); });
```

- Current set up

C++

```
ROOT::RDataFrame df(“Events”, “f.root”);
ROOT::RDF::Experimental::AddProgressbar(df);
```

Systematic variations in RDataFrame

Python

```
nominal_hx =
  df.Vary("pt", "RVecD{pt*0.9, pt*1.1}",  ["down", "up"])
    .Define("x", someFunc, ["pt"])
    .Histo1D("x")

hx = ROOT.RDF.VariationsFor(nominal_hx)
hx["nominal"].Draw()
hx["pt:down"].Draw("SAME")
```

Varied columns can be used in Defines, Filters, as histogram value/weights and anything else.
Variations automatically propagate to selections, derived quantities and results.
Multi-thread and **distributed** execution **just works**.

A tutorial for systematic variations

Physics usecase: lepton scale factors

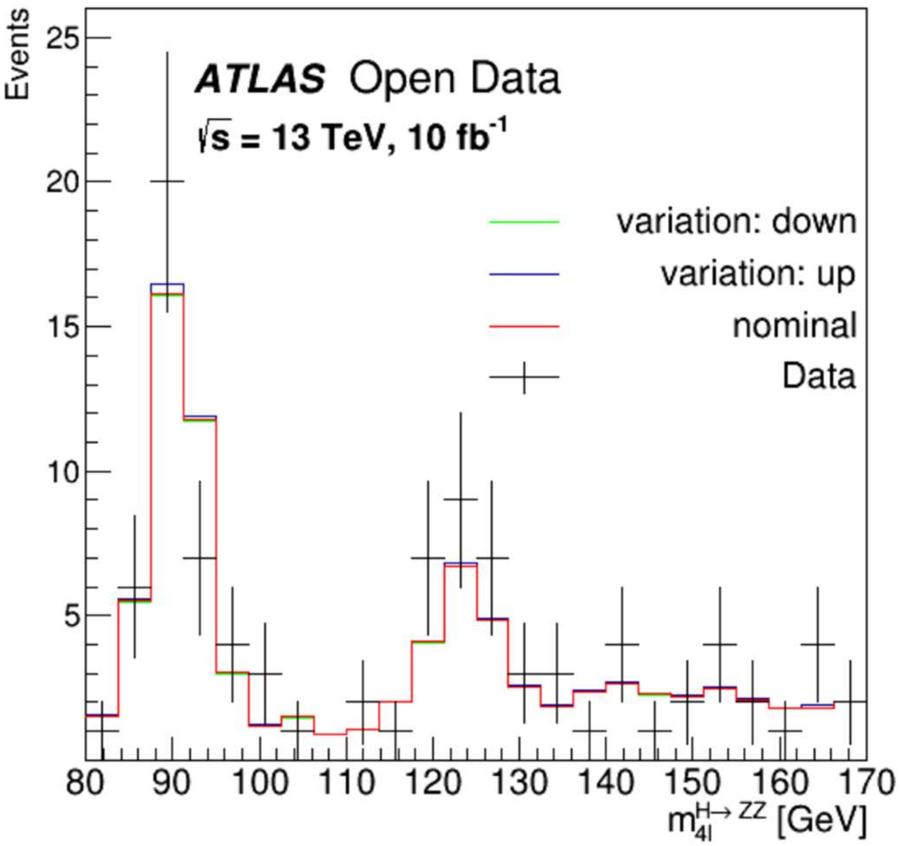
- Lepton scale factors account for differences in MC simulations vs. real data
- They vary with a lepton´s kinematics
- How does the **uncertainty in the lepton scale factors** vary with the invariant mass of a decay mode?

▶ Opportunity to “upcycle” the tutorial on the H → 4l decay

Varying the muon scale factors

- ✓ Made C++ version of H → 4l
- ✓ Applied systematic variations
- □ Difficult to find muon scale factor uncertainties
  - In particular: transverse momentum dependence

Systematic variations of the muon scale factor



Improve RDF Sum and Means methods

- RDF Sum action

C++

```cpp
for (auto &m: summands)
        sum += m;
```

- Compensated summation

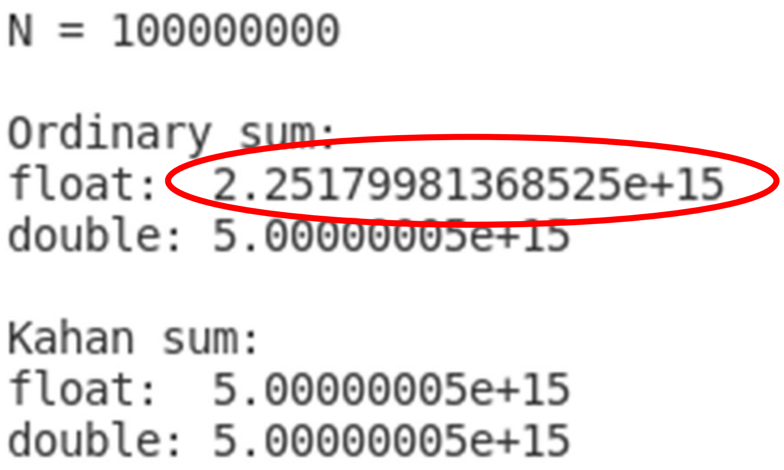
C++

```cpp
double sum(0);
double compensation(0);
double y(0);
double t(0);
for (auto &m: summands) {
        y = m – compensation;
        t = sum + y;
        compensation = (t – sum) – y;
        sum = t;
}
```

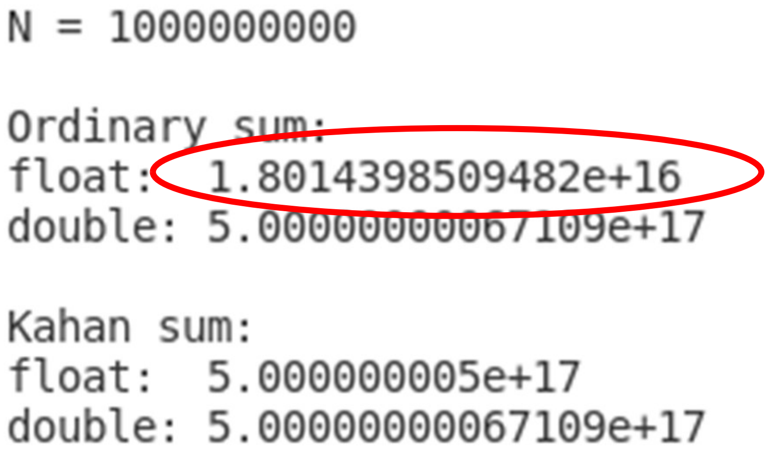
Example: Sum vs. Kahan Sum

C++

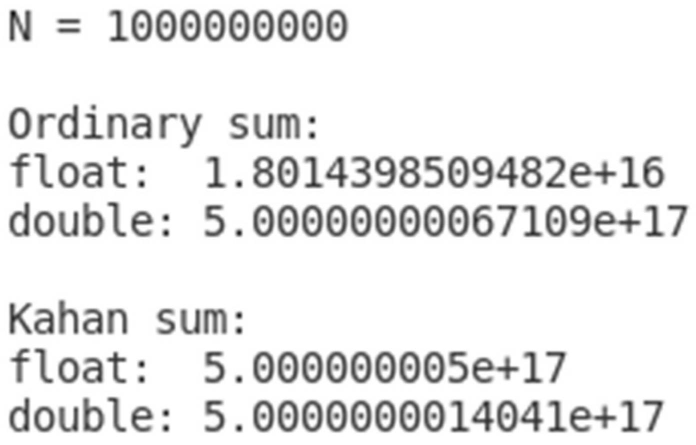
```
ROOT::RDataFrame df(N);
ROOT::RDataFrame dd(N);
auto ddf = df.Define(“x”, “float(rdfentry_ + 1)”);
auto ddd = dd.Define(“x”, “double(rdfentry_ + 1)”);
```



```
N = 100000000

Ordinary sum:
float:  2.25179981368525e+15
double: 5.00000005e+15

Kahan sum:
float:  5.00000005e+15
double: 5.00000005e+15
```



```
N = 1000000000

Ordinary sum:
float:  1.8014398509482e+16
double: 5.00000000067109e+17

Kahan sum:
float:  5.000000005e+17
double: 5.00000000067109e+17
```



```
N = 1000000000

Ordinary sum:
float:  1.8014398509482e+16
double: 5.00000000067109e+17

Kahan sum:
float:  5.000000005e+17
double: 5.0000000014041e+17
```

The TensorFlowCEvaluator

Why do we need this?

- TensorFlow is a powerful ML and AI library
- Bamboo trains their models on HEP data stored in ROOT files
- User-friendly: directly evaluate a model on RDF columns

C++

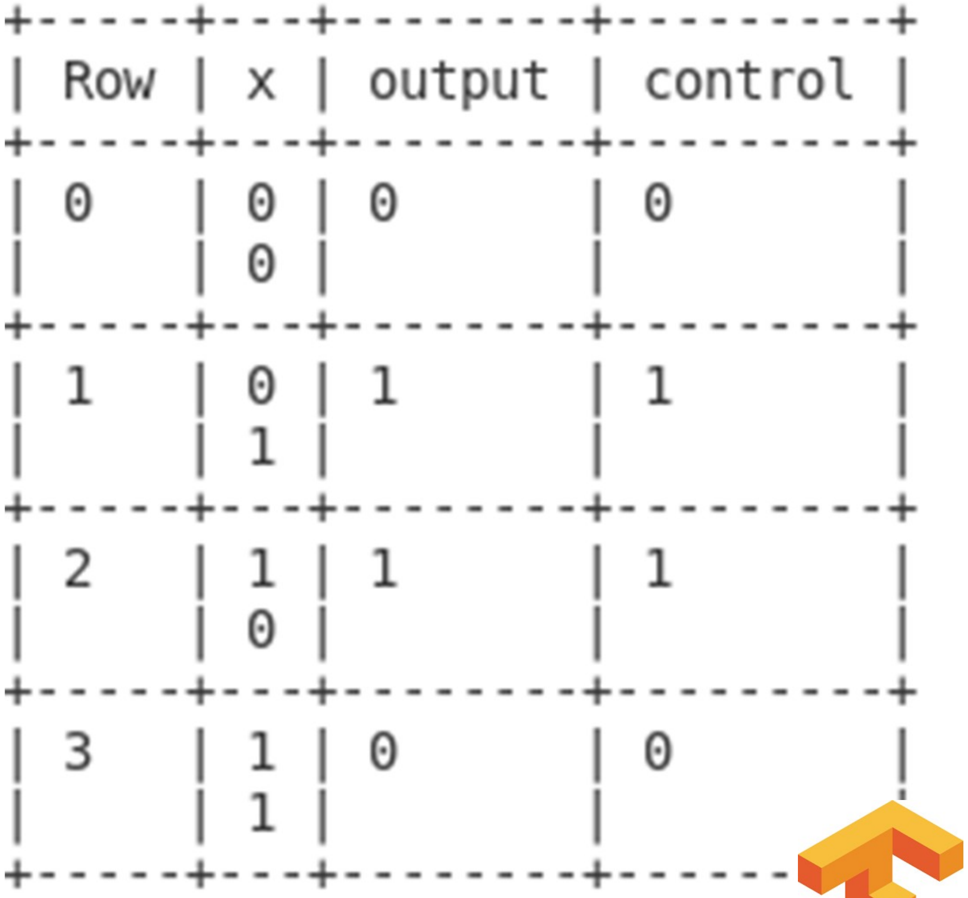
```
df.Define(“output”, [&model] (ROOT::RVecF & x_in) {return model.evaluate(x_in); },  {“input”});
```

→ **Integrate TensorFlowCEvaluator class from bamboo into RDataFrame**

Testing the TensorFlowCEvaluator

- Figure out how the bamboo TensorFlowCEvaluator works
  - Reads the model's graph
  - Needs to know input and output nodes → find way to read them from any model
- Make a model to be read and tested
  - Save everything in a **pb** file

    → "Model freezing"



```
+-----+---+---------+---------+
| Row | x | output  | control |
+-----+---+---------+---------+
| 0   | 0 | 0       | 0       |
|     | 0 |         |         |
+-----+---+---------+---------+
| 1   | 0 | 1       | 1       |
|     | 1 |         |         |
+-----+---+---------+---------+
| 2   | 1 | 1       | 1       |
|     | 0 |         |         |
+-----+---+---------+---------+
| 3   | 1 | 0       | 0       |
|     | 1 |         |
+-----+---+---------+-----
```

Results and Outlook

My projects & PRs:

- Tutorial fixes
- Progress bar
- Systematic variations
- Kahan Sum & Mean
- TensorFlowCEvaluator

My to-do’ s:

□ Finalize Vary tutorial

□ Finalize TensorFlowCEvaluator

□ Improve progress bar

□ Feature requests for RVec

Learned a lot: ROOT, RDataFrame, GitHub, debugging, testing, C++, TensorFlow