Advanced tools for HEP analysis

andrea.rizzi@unipi.it

The material in this talk is mostly taken from ROOT Basic Course available at:
https://docs.google.com/presentation/d/189f0qsDEnMSk2R5KWLRPz2TdEV5kTfXH1VcuAra4cnU/edit#slide=id.g24ecd7c82b_0_94

Advanced tools for HEP analysis

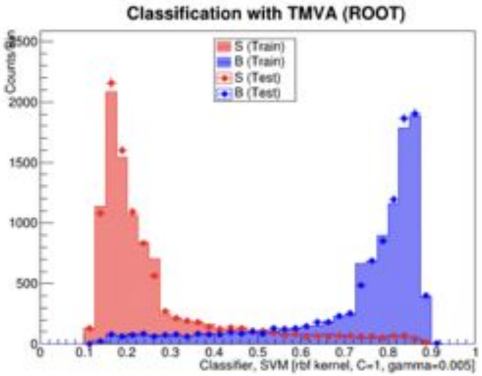
- Machine learning with ROOT (TMVA)
- Large data volumes with declarative syntax in ROOT (RDataFrame)
- Reading ROOT files without ROOT in python scientific environment

Machine learning

Machine Learning: TMVA

TMVA : Toolkit for Multi-Variate data Analysis in ROOT

- provides several built-in ML methods including:
  - Boosted Decision Trees
  - Deep Neural Networks
  - Support Vector Machines



- and interfaces to external ML tools
  - scikit-learn, Keras (Theano/Tensorflow), R

TMVA

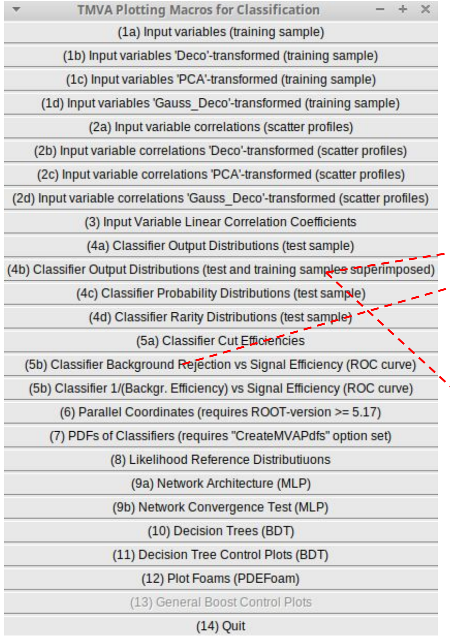
TMVA Test Suite

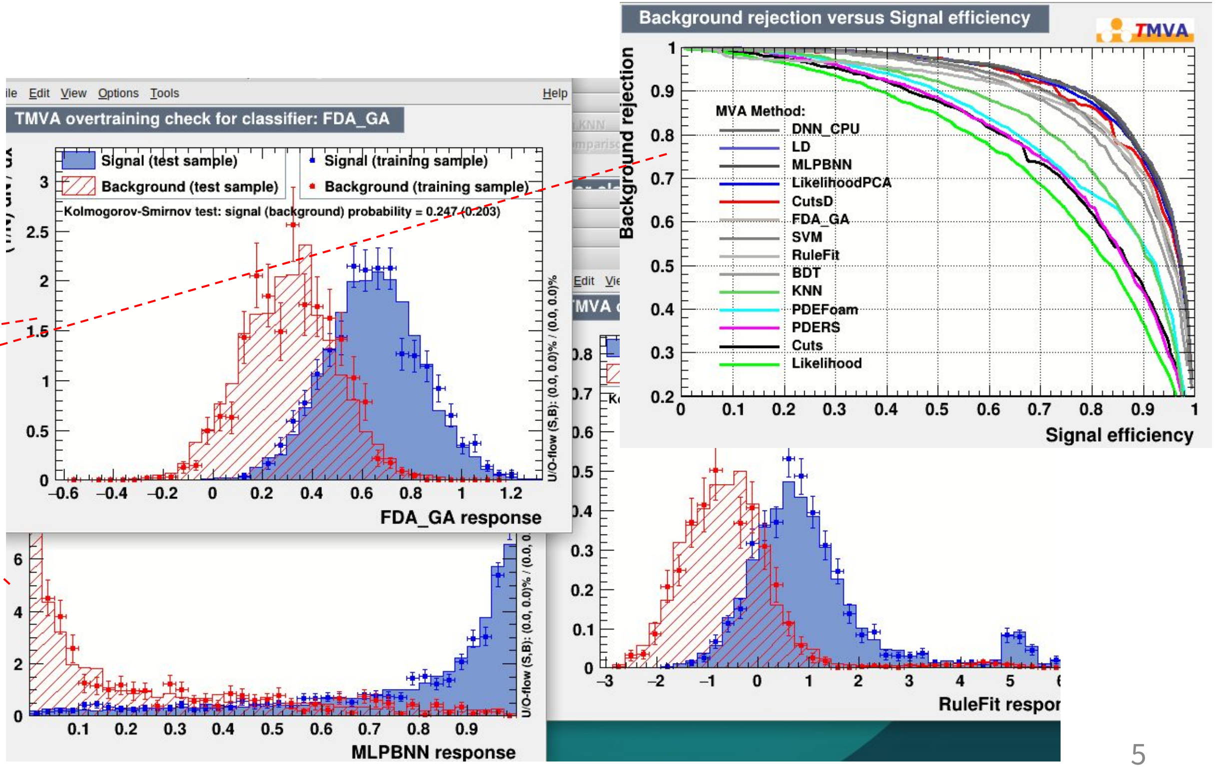
source /home/student/root/bin/thisroot.sh

cd $ROOTSYS/tutorials/tmva
$ root -l TMVAClassification.C

You may need:

```
sudo apt-get install libblas3
```





TMVA: a simple example

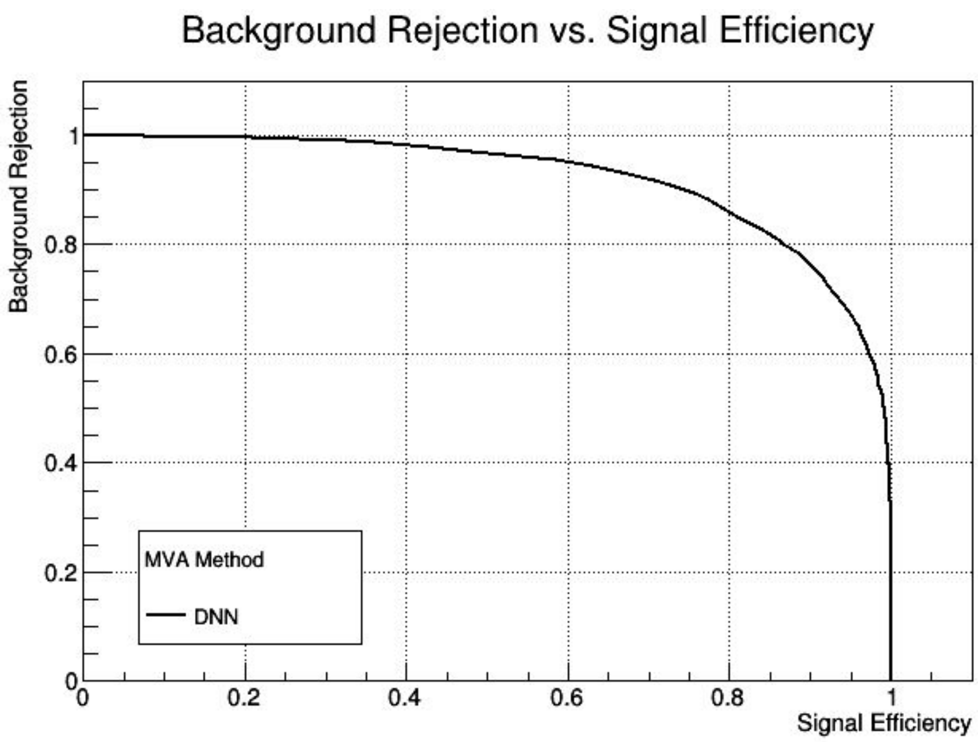
```
# Declare Factory
from ROOT import TMVA, TFile, TTree, TCut, TString

# Declare Variables in DataLoader
TMVA.Tools.Instance()

inputFile = TFile.Open("https://github.com/iml-wg/tmvatutorials/raw/master/inputdata.root")
outputFile = TFile.Open("TMVAOutputDNN.root", "RECREATE")

factory = TMVA.Factory("TMVAClassification", outputFile,
                       "!V:!Silent:Color:!DrawProgressBar:AnalysisType=Classification" )

# Declare Variables in DataLoader
loader = TMVA.DataLoader("dataset_dnn")

loader.AddVariable("var1")
loader.AddVariable("var2")
loader.AddVariable("var3")
loader.AddVariable("var4")
loader.AddVariable("var5 := var1-var3")
loader.AddVariable("var6 := var1+var2")

# Setup Dataset(s)
tsignal = inputFile.Get("Sig")
tbackground = inputFile.Get("Bkg")

loader.AddSignalTree(tsignal)
loader.AddBackgroundTree(tbackground)
loader.PrepareTrainingAndTestTree(TCut(""),
                                  "nTrain_Signal=1000:nTrain_Background=1000:SplitMode=Random:NormMode=NumEvents:!V")
```

TMVA: a simple example

```
# Configure Network Layout
# General layout
layoutString = TString("Layout=TANH|128,TANH|128,TANH|128,LINEAR");
# Training strategies
training0 = TString("LearningRate=1e-1,Momentum=0.9,Repetitions=1,"
                    "ConvergenceSteps=2,BatchSize=256,TestRepetitions=10,"
                    "WeightDecay=1e-4,Regularization=L2,"
                    "DropConfig=0.0+0.5+0.5+0.5, Multithreading=True")

training1 = TString("LearningRate=1e-2,Momentum=0.9,Repetitions=1,"
                    "ConvergenceSteps=2,BatchSize=256,TestRepetitions=10,"
                    "WeightDecay=1e-4,Regularization=L2,"
                    "DropConfig=0.0+0.0+0.0+0.0, Multithreading=True")

trainingStrategyString = TString("TrainingStrategy=")
trainingStrategyString += training0 + TString("|") + training1

# General Options
dnnOptions = TString("!H:!V:ErrorStrategy=CROSSENTROPY:VarTransform=N:"
        "WeightInitialization=XAVIERUNIFORM")
dnnOptions.Append(":")
dnnOptions.Append(layoutString)
dnnOptions.Append(":")
dnnOptions.Append(trainingStrategyString)

# Booking Methods
# Standard implementation, no dependencies.
stdOptions =  dnnOptions + ":Architecture=STANDARD"
factory.BookMethod(loader, TMVA.Types.kDNN, "DNN", stdOptions)
```

TMVA: a simple example

```python
# Train Methods
factory.TrainAllMethods()

# Test and Evaluate Methods
factory.TestAllMethods()
factory.EvaluateAllMethods()

# Plot ROC Curve
# %jsroot on
c = factory.GetROCCurve(loader)
c.Draw()
c.Print("result.png") # added
```

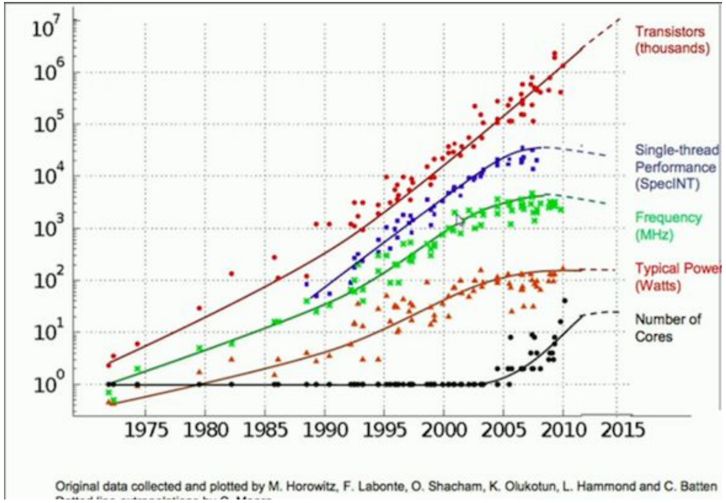


TMVA: simple inference example

```
//init
TMVA::Reader * reader = new TMVA::Reader("Silent");
float a;
reader->AddVariable("var1",&a);
reader->AddVariable("var2",&a);
reader->BookMVA("BDTG", "BDT.xml");

//per event
std::vector<float> f(2);
for(event loop) {
 f[0]=...
 f[1]=...
 float dnnresult= readers->EvaluateMVA(f,"BDTG");
}
```

Large data volumes

- Many HEP experiments need to handle millions to billions of events all the way down to analysis level
- Data reduction
  - “vertically”: reduce event content
  - “horizontally”: reduce number of events
- Computing derived quantities can be CPU expensive



- Propagation of uncertainty requires recomputation of derived quantities
- Moore’s law not scaling on “single core” performance
  - Many cores CPUs (e.g. AMD Epyc Rome 64core/128threads) or GPUs

Typical structure of analysis code

- Read from input all variables you *may* need
- Loop on all events
  - Loop on per-event collections
  - Derive event based quantities (possibly looping on different event interpretations)
  - Filter and select events
    - Make plots for various observables with different selection
- Repeat for each systematic uncertainty variation
- Save derived ntuples and/or histograms

More modern approach

- Avoid explicit loops
- Try to “declare” the transformations you want to do on data
  - slicing, projections, derived quantities
- Reduce the data into histograms and other per dataset information using optimized code
  - avoid multiple loops, avoid reading un-needed information

Examples

SQL like syntax

select average(muon_pt) from events;

SQL like syntax (each event is a "DB"):

select max(pt) from jets;

<u>Express what you want, not how to get it</u>

Data frames

- The concept of “data frame” is that of a “table” with heterogeneous columns that you can slice and dice
  - In ROOT: RDataFrame
  - In python: pandas DataFrame

RDataFrame Basics

Improved Interfaces

what we write

```
TTreeReader reader(data);
TTreeReaderValue<A> x(reader,"x");
TTreeReaderValue<B> y(reader,"y");
TTreeReaderValue<C> z(reader,"z");
while (reader.Next()) {
   if (IsGoodEntry(*x, *y, *z))
      h->Fill(*x);
}
```

what we *mean*

- full control over the event loop
- requires some boilerplate
- users implement common tasks again and again
- parallelisation is not trivial

RDataFrame: declarative analyses

```
RDataFrame d(data);
auto h = d.Filter(IsGoodEntry, {"x","y","z"})
          .Histo1D("x");
```

- full control over *the analysis*
- no boilerplate
- common tasks are already implemented

? parallelization is not trivial?

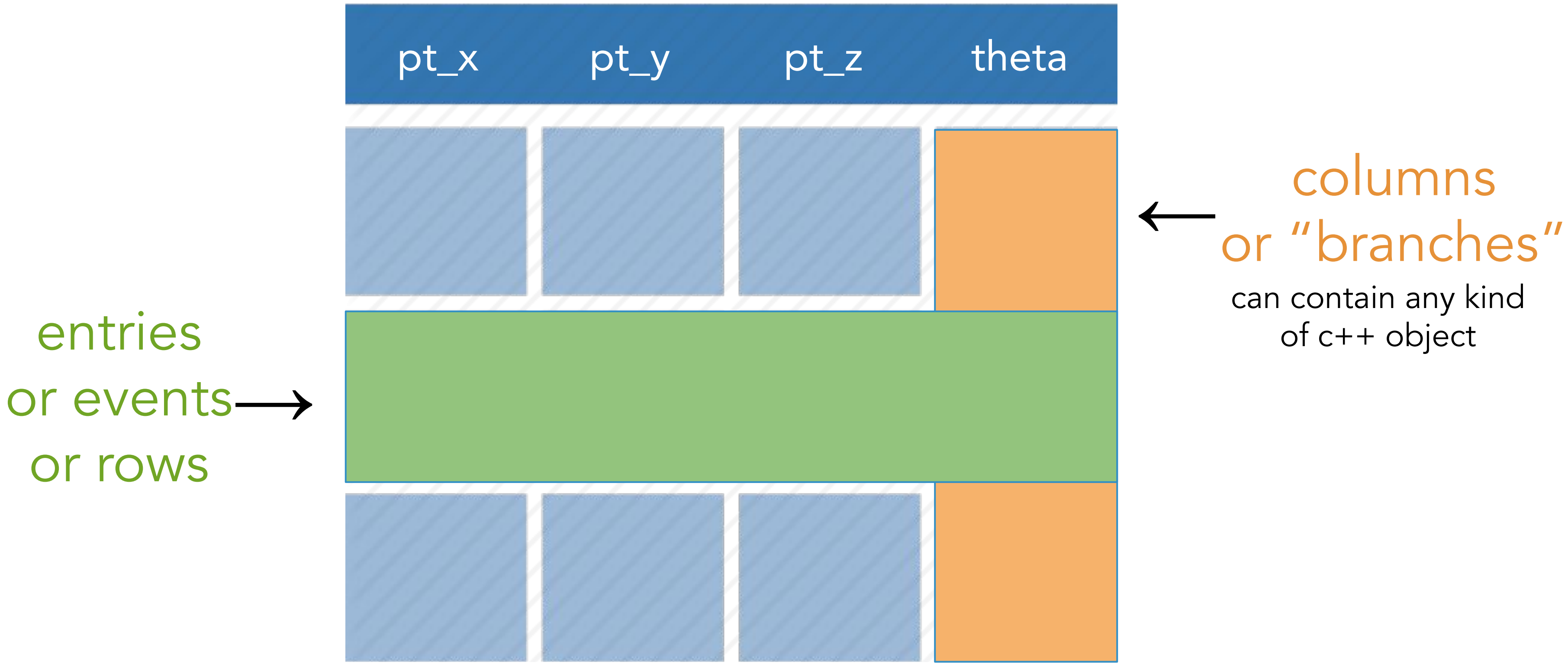
RDataFrame: declarative analyses

```
ROOT::EnableImplicitMT();
RDataFrame d(data);
auto h = d.Filter(IsGoodEntry, {"x","y","z"})
          .Histo1D("x");
```

- full control over *the analysis*
- no boilerplate
- common tasks are already implemented

? parallelization is not trivial?

Columnar Representation



RDataFrame: quick how-to

1. build a data-frame object by specifying your data-set
2. apply a series of transformations to your data
   - filter (e.g. apply some cuts) or
   - define new columns
3. apply actions to the transformed data to produce results

   (e.g. fill a histogram)

Creating a RDataFrame - 1 file

```
RDataFrame d1("treename", "file.root");

auto filePtr = TFile::Open("file.root");
RDataFrame d2("treename", filePtr);

TTree *treePtr = nullptr;
filePtr->GetObject("treename", treePtr);
RDataFrame d3(*treePtr); // by reference!
```

Three ways to create a RDataFrame that reads tree “treename” from file “file.root”

Creating a RDataFrame - more files

```
RDataFrame d1("treename", "file*.root");
RDataFrame d2("treename", {"file1.root","file2.root"});

std::vector<std::string> files = {"file1.root","file2.root"};
RDataFrame d3("treename", files);

TChain chain("treename");
chain.Add("file1.root); chain.Add("file2.root);
RDataFrame d4(chain); // passed by reference, not pointer!
```

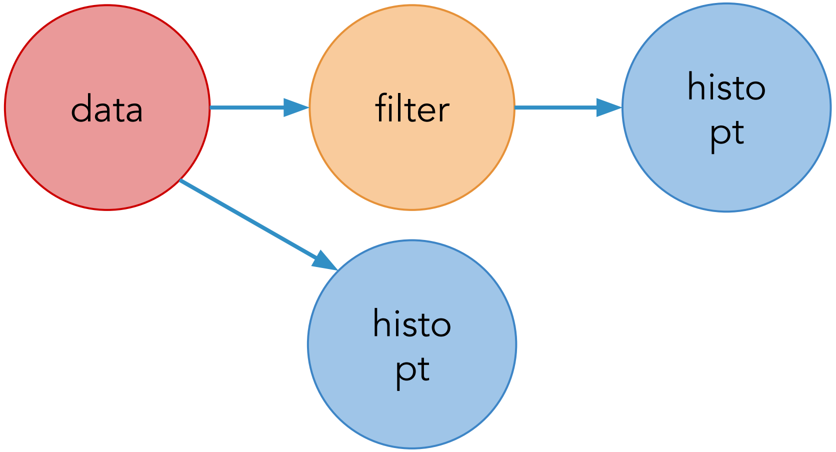
Here RDataFrame reads tree “treename” from files “file1.root” and “file2.root”

Cut on theta, fill histogram with pt

```
RDataFrame d("t", "f.root");
auto h = d.Filter("theta > 0").Histo1D("pt");
h->Draw(); // event loop is run here, when you access a result
           // for the first time
```

event-loop is run *lazily*, upon first access to the results

Think of your analysis as data-flow




```
auto h2 = d.Filter("theta > 0").Histo1D("pt");
auto h1 = d.Histo1D("pt");
```

Using callables instead of strings

```cpp
// define a c++11 lambda - an inline function - that checks "x>0"
auto IsPos = [](double x) { return x > 0.; };
// pass it to the filter together with a list of branch names
auto h = d.Filter(IsPos, {"theta"}).Histo1D("pt");
h->Draw();
```

any callable (function, lambda, functor class) can be used as a filter, as long as it returns a boolean

Filling multiple histograms

```
auto h1 = d.Filter("theta > 0").Histo1D("pt");
auto h2 = d.Filter("theta < 0").Histo1D("pt");
h1->Draw();          // event loop is run once here
h2->Draw("SAME"); // no need to run loop again here
```

Book all your actions upfront. The first time a result is accessed, RDataFrame will fill all booked results.

Define a new column

```
double m = d.Filter("x > y")
            .Define("z", "sqrt(x*x + y*y)")
            .Mean("z");
```

`Define` takes the name of the new column and its expression. Later you can use the new column as if it was present in your data.

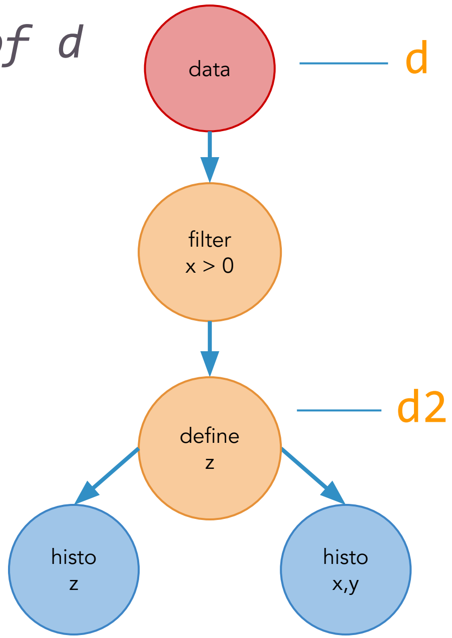
Define a new column

```
double SqrtSumSq(double, double) { return ... ; }
double m = d.Filter("x > y")
            .Define("z", SqrtSumSq, {"x","y"})
            .Mean("z");
```

Just like `Filter`, `Define` accepts any callable object (function, lambda, functor class...)

Think of your analysis as data-flow

```
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z", "x*x + y*y");

// make multiple histograms out of it
auto hz = d2.Histo1D("z");
auto hxy = d2.Histo2D("x","y");
```



You can store transformed data-frames in variables, then use them as you would use a RDataFrame.

Cutflow reports

```
d.Filter("x > 0", "xcut")
 .Filter("y < 2", "ycut");
d.Report();
```

```
// output
xcut      : pass=49        all=100      --   49.000 %
ycut      : pass=22        all=49       --   44.898 %
```

When called on the main TDF object, `Report` prints statistics for all filters *with a name*

Running on a range of entries #1

```
// stop after 100 entries have been processed
auto hz = d.Range(100).Histo1D("x");

// skip the first 10 entries, then process one every two until the end
auto hz = d.Range(10, 0, 2).Histo1D("x");
```

Ranges are only available in single-thread executions.
They are useful for quick initial data explorations.

Running on a range of entries #2

```
// ranges can be concatenated with other transformations
auto c = d.Filter("x > 0")
          .Range(100)
          .Count();
```

This `Range` will process the first 100 entries
that pass the filter

Saving data to file

```
auto new_df = df.Filter("x > 0")
                .Define("z", "sqrt(x*x + y*y)")
                .Snapshot("tree", "newfile.root");
```

We filter the data, add a new column, and then save everything to file. No boilerplate code at all.

Creating a new data-set

```
RDataFrame d(100);
auto new_d = d.Define("x", []() { return double(rand()) / RAND_MAX; })
              .Define("y", []() { return rand() % 10; })
              .Snapshot("tree", "newfile.root");
```

We create a special TDF with 100 (empty) entries, define some columns, save it to file

N.B. `rand()` is generally not a good way to produce uniformly distributed random numbers

Not Only ROOT Datasets

- TDataSource: Plug *any columnar* format in RDataFrame
- Keep the programming model identical!
- ROOT provides CSV data source
- More to come
  - TDataSource is a programmable interface!
  - E.g. https://github.com/bluehood/mdfds LHCb raw format - not in the ROOT repo

Not Only ROOT Datasets

```
auto fileName = "tdf014_CsvDataSource_MuRun2010B.csv";
auto tdf = ROOT::Experimental::TDF::MakeCsvDataFrame(fileName);

auto filteredEvents =
tdf.Filter("Q1 * Q2 == -1")
.Define("m", "sqrt(pow(E1 + E2, 2) - (pow(px1 + px2, 2) + pow(py1 + py2, 2) + pow(pz1 + pz2, 2)))");

auto invMass =
filteredEvents.Histo1D({"invMass", "CMS Opendata: #mu#mu mass;mass [GeV];Events", 512, 2, 110}, "m");
```

tdf014_CsvDataSource_MuRun2010B.csv:

```
Run,Event,Type1,E1,px1,py1,pz1,pt1,eta1,phi1,Q1,Type2,E2,px2,py2,pz2,pt2,eta2,phi2,Q2,M
146436,90830792,G,19.1712,3.81713,9.04323,-16.4673,9.81583,-1.28942,1.17139,1,T,5.43984,-0.362592,2.62699,-
4.74849,2.65189,-1.34587,1.70796,1,2.73205
146436,90862225,G,12.9435,5.12579,-3.98369,-11.1973,6.4918,-1.31335,-0.660674,-1,G,11.8636,4.78984,-6.26222,
-8.86434,7.88403,-0.966622,-0.917841,1,3.10256
```

RDataFrame
Extra features

Caching

```cpp
RDataFrame d("mytree", "myFile.root");
auto cached_d = d.Cache();
```

All the content of the TDF is now in (contiguous) memory.
Analysis as fast as it can be (vectorisation possible too).

N.B. It is always possible to selectively cache columns to save some memory!

Creating a new data-set - parallel

```
ROOT::EnableImplicitMT();
RDataFrame d(100);
auto new_d = d.Define("x", []() { return double(rand()) / RAND_MAX; })
              .Define("y", []() { return rand() % 10; })
              .Snapshot("tree", "newfile.root");
```

We create a special TDF with 100 (empty) entries, define some columns, save it to file -- in parallel

N.B. `rand()` is generally [not a good way]() to produce uniformly distributed random numbers

More on histograms #1

```
auto h = d.Histo1D("x","w");
```

TDF can produce *weighted* TH1D, TH2D and TH3D.
Just pass the extra column name.

More on histograms #2

```
auto h = d.Histo1D({"h","h",10,0.,1.},"x", "w");
```

You can specify a model histogram with a set axis range, a name and a title (optional for TH1D, mandatory for TH2D and TH3D)

Filling histograms with arrays

```
auto h = d.Histo1D("pt_array", "x_array");
```

If `pt_array` and `x_array` are an array or an STL container (e.g. std::vector), TDF fills histograms with all of their elements. `pt_array` and `x_array` are required to have equal size for each event.

C++ / JIT / PyROOT

Pure C++

```
d.Filter([](double t) { return t > 0.; }, {"th”})
 .Snapshot<vector<float>>("t","f.root",{"pt_x"});
```

C++ and JIT-ing with CLING

```
d.Filter("th > 0").Snapshot("t","f.root","pt*");
```

pyROOT -- just leave out the ;

```
d.Filter("th > 0").Snapshot("t","f.root","pt*")
```

Convert to Numpy or Pandas

```
import ROOT

ROOT.EnableImplicitMT()

df = ROOT.RDataFrame('myTree', 'file.root')

# Perform computations with RDataFrame
np_arrays = df.Filter('x > 0')
              .Define('z', 'x*y')
              .AsNumpy()
```

Get dataset columns as NumPy arrays

```
# Wrap data with pandas
import pandas
pdf = pandas.DataFrame(np_arrays)
```

Use python functions

```
import ROOT

@ROOT.Numba.Declare(['float', 'int'], 'float')
def pypow(x, y):
    return x**y

# Use the callable within an RDataFrame workflow
data = ROOT.RDataFrame('tree', 'file.root')
            .Define('x_pow3', 'Numba::pypow(x, 3)')
            .AsNumpy()
```

Operations on a RDF

Transformation	Description
Define	Creates a new column in the dataset.
DefineSlot	Same as `Define`, but the user-defined function must take an extra `unsigned int slot` as its first parameter. `slot` will take a different value, `0` to `nThreads - 1`, for each thread of execution. This is meant as a helper in writing thread-safe `Define` transformation when using `RDataFrame` after `ROOT::EnableImplicitMT()`. `DefineSlot` works just as well with single-thread execution: in that case `slot` will always be `0`.
DefineSlotEntry	Same as `DefineSlot`, but the entry number is passed in addition to the slot number. This is meant as a helper in case some dependency on the entry number needs to be honoured.
Filter	Filter the rows of the dataset.
Range	Creates a node that filters entries based on range of entries

Instant action	Description
Foreach	Execute a user-defined function on each entry. Users are responsible for the thread-safety of this lambda when executing with implicit multi-threading enabled.
ForeachSlot	Same as `Foreach`, but the user-defined function must take an extra `unsigned int slot` as its first parameter. `slot` will take a different value, `0` to `nThreads - 1`, for each thread of execution. This is meant as a helper in writing thread-safe `Foreach` actions when using `RDataFrame` after `ROOT::EnableImplicitMT()`. `ForeachSlot` works just as well with single-thread execution: in that case `slot` will always be `0`.
Snapshot	Writes processed data-set to disk, in a new `TTree` and `TFile`. Custom columns can be saved as well, filtered entries are not saved. Users can specify which columns to save (default is all). Snapshot, by default, overwrites the output file if it already exists. `Snapshot` can be made *lazy* setting the appropriate flage in the snapshot options.

Lazy action	Description
Aggregate	Execute a user-defined accumulation operation on the processed column values.
Book	Book execution of a custom action using a user-defined helper object.
Cache	Caches in contiguous memory columns' entries. Custom columns can be cached as well, filtered entries are not cached. Users can specify which columns to save (default is all).
Count	Return the number of events processed.
Display	Obtains the events in the dataset for the requested columns. The method returns a RDisplay instance which can be queried to get a compressed tabular representation on the standard output or a complete representation as a string.
Fill	Fill a user-defined object with the values of the specified branches, as if by calling `Obj.Fill(branch1, branch2, ...).
Graph	Fills a **TGraph** with the two columns provided. If Multithread is enabled, the order of the points may not be the one expected, it is therefore suggested to sort if before drawing.
Histo{1D,2D,3D}	Fill a {one,two,three}-dimensional histogram with the processed branch values.
Max	Return the maximum of processed branch values. If the type of the column is inferred, the return type is `double`, the type of the column otherwise.
Mean	Return the mean of processed branch values.
Min	Return the minimum of processed branch values. If the type of the column is inferred, the return type is `double`, the type of the column otherwise.
Profile{1D,2D}	Fill a {one,two}-dimensional profile with the branch values that passed all filters.
Reduce	Reduce (e.g. sum, merge) entries using the function (lambda, functor...) passed as argument. The function must have signature `T(T,T)` where T is the type of the branch. Return the final result of the reduction operation. An optional parameter allows initialization of the result object to non-default values.
Report	Obtains statistics on how many entries have been accepted and rejected by the filters. See the section on named filters for a more detailed explanation. The method returns a RCutFlowReport instance which can be queried programmatically to get information about the effects of the individual cuts.
StdDev	Return the unbiased standard deviation of the processed branch values.
Sum	Return the sum of the values in the column. If the type of the column is inferred, the return type is `double`, the type of the column otherwise.
Take	Extract a column from the dataset as a collection of values. If the type of the column is a C-style array, the type stored in the return container is a **`ROOT::VecOps::RVec`**<T> to guarantee the lifetime of the data involved.

Other RDF functions

Operation	Description
Alias	Introduce an alias for a particular column name.
GetColumnNames	Get the names of all the available columns of the dataset.
GetDefinedColumnNames	Get the names of all the defined columns
GetColumnType	Return the type of a given column as a string.
GetColumnTypeNamesList	Return the list of type names of columns in the dataset.
GetFilterNames	Get all the filters defined. If called on a root node, all filters will be returned. For any other node, only the filters upstream of that node.
Display	Provides an ASCII representation of the columns types and contents of the dataset printable by the user.
SaveGraph	Store the computation graph of an **RDataFrame** in graphviz format for easy inspection.
GetNRuns	Get the number of event loops run by this **RDataFrame** instance.

The inner loops

- Ok, we can get rid of the "event loop", but how about the "inner loops"
  - e.g. "loop on all particles and find the highest pt"
- RDF sees "arrays" as RVec<...> (similar to std::vector, with some benefits)

```
RVec ROOT::VecOps::RVec< T >::operator[] ( const RVec< V > & conds ) const
```

- RDF has predefined "VecOps"
  - https://root.cern/doc/master/classROOT_1_1VecOps_1_1RVec.html

VecOps

template<typename T >
auto **All** (const RVec< T > &v) -> decltype(v[0]==false)
Return true if all of the elements equate to true, return false otherwise. More...

template<typename T >
auto **Any** (const RVec< T > &v) -> decltype(v[0]==true)
Return true if any of the elements equates to true, return false otherwise. More...

template<typename T >
std::size_t **ArgMax** (const RVec< T > &v)
Get the index of the greatest element of an **RVec** In case of multiple occurrences of the maximum values, the index corresponding to the first occurrence is returned. More...

template<typename T >
std::size_t **ArgMin** (const RVec< T > &v)
Get the index of the smallest element of an **RVec** In case of multiple occurrences of the minimum values, the index corresponding to the first occurrence is returned. More...

template<typename T >
RVec< typename RVec< T >::size_type > **Argsort** (const RVec< T > &v)
Return an **RVec** of indices that sort the input **RVec**. More...

template<typename T >
RVec< RVec< typename RVec< T >::size_type > > **Combinations** (const RVec< T > &v, const typename RVec< T >::size_type n)
Return the indices that represent all unique combinations of the elements of a given **RVec**. More...

template<typename T1 , typename T2 >
RVec< RVec< typename RVec< T1 >::size_type > > **Combinations** (const RVec< T1 > &v1, const RVec< T2 > &v2)
Return the indices that represent all combinations of the elements of two RVecs. More...

RVec< RVec< std::size_t > > **Combinations** (const std::size_t size1, const std::size_t size2)
Return the indices that represent all combinations of the elements of two RVecs. More...

VecOps

template<typename T , typename... Args_t>

RVec< T > **Construct** (const RVec< Args_t > &... args)
Build an **RVec** of objects starting from RVecs of input to their constructors. More...

template<typename T >

RVec< T > **DeltaPhi** (const RVec< T > &**v1**, const RVec< T > &**v2**, const T **c=M_PI**)
Return the angle difference $\Delta\phi$ in radians of two vectors. More...

template<typename T >

RVec< T > **DeltaPhi** (const RVec< T > &**v1**, T **v2**, const T **c=M_PI**)
Return the angle difference $\Delta\phi$ in radians of a vector and a scalar. More...

template<typename T >

RVec< T > **DeltaPhi** (T **v1**, const RVec< T > &**v2**, const T **c=M_PI**)
Return the angle difference $\Delta\phi$ in radians of a scalar and a vector. More...

template<typename T >

T **DeltaPhi** (T **v1**, T **v2**, const T **c=M_PI**)
Return the angle difference $\Delta\phi$ of two scalars. More...

template<typename T >

RVec< T > **DeltaR** (const RVec< T > &eta1, const RVec< T > &eta2, const RVec< T > &phi1, const RVec< T > &phi2, const T **c=M_PI**)
Return the distance on the η- ϕ plane (ΔR) from the collections eta1, eta2, phi1 and phi2. More...

template<typename T >

T **DeltaR** (T eta1, T eta2, T phi1, T phi2, const T **c=M_PI**)
Return the distance on the η- ϕ plane (ΔR) from the scalars eta1, eta2, phi1 and phi2. More...

VecOps

template<typename T , typename V >

auto **Dot** (const RVec< T > &v0, const RVec< V > &v1) -> decltype(v0[0] *v1[0])

Inner product. More...

template<typename T , typename F >

RVec< T > **Filter** (const RVec< T > &v, F &&f)

Create a new collection with the elements passing the filter expressed by the predicate. More...

template<typename T >

RVec< T > **Intersect** (const RVec< T > &v1, const RVec< T > &v2, bool v2_is_sorted=false)

Return the intersection of elements of two RVecs. More...

template<typename T >

T **InvariantMass** (const RVec< T > &pt, const RVec< T > &eta, const RVec< T > &phi, const RVec< T > &mass)

Return the invariant mass of multiple particles given the collections of the quantities transverse momentum (pt), rapidity (eta), azimuth (phi) and mass. More...

template<typename T >

RVec< T > **InvariantMasses** (const RVec< T > &pt1, const RVec< T > &eta1, const RVec< T > &phi1, const RVec< T > &mass1, const RVec< T > &pt2, const RVec< T > &eta2, const RVec< T > &phi2, const RVec< T > &mass2)

Return the invariant mass of two particles given the collections of the quantities transverse momentum (pt), rapidity (eta), azimuth (phi) and mass. More...

template<typename... Args>

auto **Map** (Args &&... args) -> decltype(ROOT::Detail::VecOps::MapFromTuple(std::forward_as_tuple(args...), std::make_index_sequence< sizeof...(args) - 1 >()))

Create new collection applying a callable to the elements of the input collection. More...

template<typename T >

T **Max** (const RVec< T > &v)

Get the greatest element of an RVec. More...

VecOps

double **Mean** (const **RVec**< T > &v)
Get the mean of the elements of an **RVec**. More...

template<typename T >
T **Min** (const **RVec**< T > &v)
Get the smallest element of an **RVec**. More...

template<typename T >
RVec< typename **RVec**< T >::size_type > **Nonzero** (const **RVec**< T > &v)
Return the indices of the elements which are not zero. More...

template<class T >
std::ostream & **operator<<** (std::ostream &os, const **RVec**< T > &v)
Print a **RVec** at the prompt: More...

template<typename T >
RVec< T > **Reverse** (const **RVec**< T > &v)
Return copy of reversed vector. More...

template<typename T >
RVec< T > **Sort** (const **RVec**< T > &v)
Return copy of **RVec** with elements sorted in ascending order. More...

template<typename T , typename Compare >
RVec< T > **Sort** (const **RVec**< T > &v, **Compare** &&c)
Return copy of **RVec** with elements sorted based on a comparison operator. More...

template<typename T >
double **StdDev** (const **RVec**< T > &v)
Get the standard deviation of the elements of an **RVec**. More...

template<typename T >
T **Sum** (const **RVec**< T > &v)

VecOps

template<typename T >
void swap (RVec< T > &lhs, RVec< T > &rhs)

template<typename T >
RVec< T > Take (const RVec< T > &v, const int n)
Return first or last n elements of an RVec. More...

template<typename T >
RVec< T > Take (const RVec< T > &v, const RVec< typename RVec< T >::size_type > &i)
Return elements of a vector at given indices. More...

template<typename T >
double Var (const RVec< T > &v)
Get the variance of the elements of an RVec. More...

template<typename T >
RVec< T > Where (const RVec< int > &c, const RVec< T > &v1, const RVec< T > &v2)
Return the elements of v1 if the condition c is true and v2 if the condition c is false. More...

template<typename T >
RVec< T > Where (const RVec< int > &c, const RVec< T > &v1, T v2)
Return the elements of v1 if the condition c is true and sets the value v2 if the condition c is false. More...

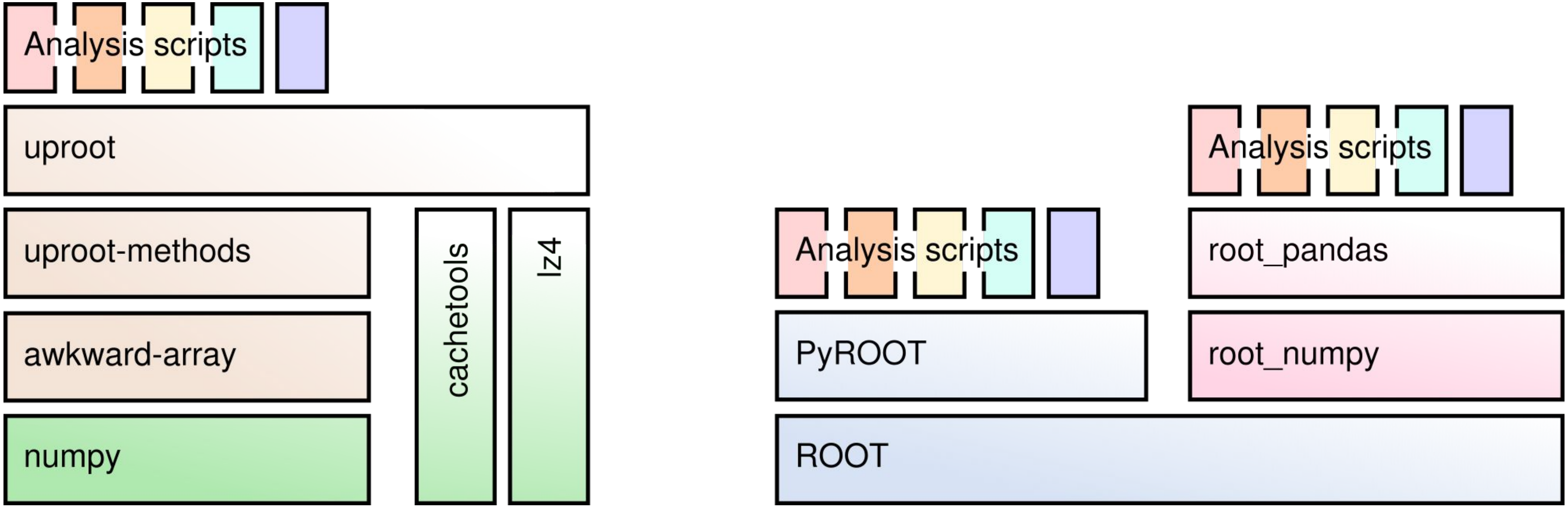
template<typename T >
RVec< T > Where (const RVec< int > &c, T v1, const RVec< T > &v2)
Return the elements of v2 if the condition c is false and sets the value v1 if the condition c is true. More...

template<typename T >
RVec< T > Where (const RVec< int > &c, T v1, T v2)
Return a vector with the value v2 if the condition c is false and sets the value v1 if the condition c is true. More...

Examples

```
using namespace ROOT::VecOps;

RVec<float> vf {1.f, 2.f, 3.f, 4.f};
auto vf_1 = Take(vf, {1, 3}); // The content is {2.f, 4.f}
auto vf_2 = Take(vf, 2); // The content is {1.f, 2.f}
auto vf_3 = Take(vf, -3); // The content is {2.f, 3.f, 4.f}

auto vf_4 = vf[vf >2]; // The content is {3.f, 4.f}


RVec<double> v0 {9., 7., 8.};
auto v1_indices = Argsort(v0); // The content of v1_indices is {1, 2, 0}.
v1 = Take(v0, v1_indices);

RVec<float> pts = {15.5, 34.32, 12.95};
RVec<float> etas = {0.3, -2.2, 1.32};
RVec<float> phis = {0.1, 3.02, 2.2};
RVec<float> masses = {105.65, 105.65, 105.65};

auto fwd_pts = pts[etas > 0];
auto most_fwd_pts = pts[ArgMax(etas)];

auto pt2 = Map(pts , [](double x){return x*x;} );

auto fourVecs = Construct<ROOT::Math::PtEtaPhiMVector>(pts, etas, phis, masses);
```

ROOT data without ROOT



Analysing ROOT data with python tools

- ROOT provides python bindings to call ROOT tools and functions (TTree draw, TH manipulation, TMVA, RooFit, RDataFrame etc…)
- What if I want to use other (python native) analysis tools?
- Use “**uproot**” (or “root_numpy”) and then feed data to:
  - numpy
  - matplotlib
  - panda dataframe
  - python machine learning algorithms (keras, tensorflow)
  - minimization tools (e.g. **zfit**)
  - …any other python goodies (see https://github.com/scikit-hep)

uproot

```
pip install uproot awkward
```

```
import uproot
import os
if not os.path.exists('DataSet highstat.root'):
   os.system('wget -O DataSet_highstat.root cern.ch/arizzi/out.root')


tree = uproot.open("DataSet_highstat.root")["Events"]

massdata=tree["Dimuon_mass"].array()
print(massdata)

#Draw with matplot lib
import matplotlib.pyplot as plt
n, bins, patches = plt.hist(massdata, 50, (0,5) )
plt.show()
```

zfit (https://github.com/zfit/zfit)

```
pip3 install zfit --user
sudo apt-get install python3-tk
```

Similar to RooFit

- i.e. handles PDFs
- Decoupled from ROOT
- Supports multiple backends
  - Minuit
  - TensorFlow
  - scipy tools

```python
obs = zfit.Space('x', limits=(-10, 10))

# create the model
mu    = zfit.Parameter("mu"   , 2.4, -1, 5)
sigma = zfit.Parameter("sigma", 1.3,  0, 5)
gauss = zfit.pdf.Gauss(obs=obs, mu=mu, sigma=sigma)

# load the data
data_np = np.random.normal(size=10000)
data = zfit.Data.from_numpy(obs=obs, array=data_np)

# build the loss
nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

# minimize
minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

# calculate errors
param_errors = result.error()
```

Scikit HEP

More HEP tools in python

https://scikit-hep.org/

Basics:

Awkward Array	Manipulate arrays of complex data structures as easily as Numpy.	
hepunits	Units and constants in the HEP system of units.	

Data manipulation and interoperability:

formulate	Easy conversions between different styles of expressions.	
root_numpy	Interface between ROOT and NumPy.	
root_pandas	Module for conveniently loading/saving ROOT files as pandas DataFrames.	
uproot	Minimalist ROOT I/O in pure Python and Numpy.	
uproot-methods	Pythonic behaviours for non-I/O related ROOT classes.	

Histogramming:

aghast	Convert between histogram representations	
Boost histogram	Python bindings for the C++14 Boost::Histogram library.	
hist	Hist is a analyst friendly front-end for boost-histogram, designed for Python 3.6+.	

Particles and decays:

DecayLanguage	Describe and convert particle decays between digital representations.	
Particle	PDG particle data and identification codes.	

Fitting:

GooFit	GPU/OpenMP fitting in Python and C++.	Affiliated
iminuit	MINUIT from Python - Fitting like a boss.	
probfit	Cost function builder. For fitting distributions.	Deprecated
zfit	Scalable Pythonic fitting	Affiliated

Statistics:

hepstats	Statistics tools and utilities.	
pyhf (differentiable likelihoods)	pure-Python implementation of HistFactory models.	

Interface to HEP libraries:

numpythia	Interface between Pythia and NumPy.	
pyhepmc	Next generation Python bindings for HepMC3.	
pyjet	Interface between FastJet and NumPy.	
pylhe	Lightweight Python interface to read Les Houches Event (LHE) files.	

Afternoon part - exercises

Time For Exercises

▶️ Draw a plot of px + py for every pz between -2 and 2 using the $ROOTSYS/tutorials/hsimple.root file

- Use RDataFrame
- Compare with the other approaches: number of lines, readability

RDataFrame Exercise (in python or in C++)

- Open CMS di-muon open data file

`root://eospublic.cern.ch//eos/root-eos/cms_opendata_2012_nanoaod/Run2012B_DoubleMuParked.root`

- Check the list of available “columns”
  - Please note that some columns have variable length (e.g. Muon_pt) per event
- Create a RDataFrame object
- Select events where the first muon has pt > 20 GeV (units for E/p in CMS data is GeV)
- Write a C++ lambda function that computes the invariant mass of a pair of particles given pt,eta(= -log(tan(theta/2)), phi of the particle
- Select events where the first two muons have opposite charge
- Compute the invariant mass
- Make an histogram
- Store it in a “Snapshot”

Exercise solution

```
import ROOT
rdf = ROOT.RDataFrame("Events",
                "root://eospublic.cern.ch//eos/root-eos/cms_opendata_2012_nanoaod/Run2012B_DoubleMuParked.root");

#list available columns
print(rdf.GetColumnNames())

sel0=rdf.Filter("nMuon>=2","two muons").Range(10000) #restrict to first 10k events with at least two muons
sel1=sel0.Filter("Muon_pt[0]>20","leading mu pt")
sel2=sel1.Filter("Muon_charge[0]*Muon_charge[1]<0","opposite charge")

#create a C++ function to compute the mass
cppcode='''
float mass(float pt1, float eta1, float phi1, float pt2,float eta2, float phi2)
{
 TLorentzVector mu1,mu2;
 mu1.SetPtEtaPhiM(pt1,eta1,phi1,0.106);
 mu2.SetPtEtaPhiM(pt2,eta2,phi2,0.106);
 return (mu1+mu2).M();
}
'''
ROOT.gInterpreter.ProcessLine(cppcode)

#add mass
mass=sel2.Define("Dimuon_mass","mass(Muon_pt[0],Muon_eta[0],Muon_phi[0],Muon_pt[1],Muon_eta[1],Muon_phi[1])")

outCols=ROOT.vector("std::string")() #this creates a c++ std::vector<std::string> and wrap it in python
outCols.push_back("Dimuon_mass")

mass.Snapshot("Events","out.root",outCols)

import pandas
print(pandas.DataFrame(mass.AsNumpy(["Dimuon_mass","event","run"])))
```

Exercise 2

- Select events with two muons passing a selection
  - pt > 20, abs(eta) < 2.0
- Extend it to the case of more than two muons, taking the first two with opposite charge

```
twoOCMuons=rdf.Define("goodmu","Muon_pt > 20 && abs(Muon_eta) < 2.0")
                      .Filter("Sum(goodmu)>=2")
                      .Define("mu0","Nonzero(goodmu)[0]")
                      .Define("oppositeCharge"," goodmu && Muon_charge[mu0]*Muon_charge<0 ")
                      .Filter( "Sum(oppositeCharge)>0")
                      .Define("mu1","Nonzero(oppositeCharge)[0]")
```