The FAST-HEP toolset: Using YAML to make tables out of trees

Benjamin Edward Krikler[1,*], *Olivier* Davignon[2], *Lukasz* Kreczko[1], and *Jacob* Linacre[3]

[1]University of Bristol, UK
[2]Laboratoire Leprince-Ringuet, CNRS/IN2P3, France
[3]Rutherford Appleton Laboratory, UK

Abstract. The Faster Analysis Software Taskforce (FAST) is a small, European group of HEP researchers that have been investigating and developing modern software approaches to improve HEP analyses. We present here an overview of the key product of this effort: a set of packages that allows a complete implementation of an analysis using almost exclusively YAML files. Serving as an analysis description language (ADL), this toolset builds on top of the evolving technologies from the Scikit-HEP and IRIS-HEP projects as well as industry-standard libraries such as Pandas and Matplotlib. Data processing starts with event-level data (the trees) and can proceed by adding variables, selecting events, performing complex user-defined operations and binning data, as defined in the YAML description. The resulting outputs (the tables) are stored as Pandas dataframes which can be programmatically manipulated and converted to plots or inputs for fitting frameworks. No longer just a proof-of-principle, these tools are now being used in CMS analyses, the LUX-ZEPLIN experiment, and by students on several other experiments. In this talk we will showcase these tools through examples, highlighting how they address the different experiments' needs, and compare them to other similar approaches.

1 Introduction

Producing high-quality research papers in High-Energy Physics (HEP) involves processing petabytes of data, applying the latest knowledge for the specific experiment and the statistical evaluation of the end-results and their uncertainties. This process often involves the use of experiment specific software frameworks, community packages as well as researcher-written code. All components have to undergo rigorous testing and the result is required to be fully reproducible. The key aims of the Faster Analysis Software Taskforce (FAST) are to: a) reduce the amount of researcher-written code to minimize mistakes, b) lower the entry requirements for new researchers, c) make it easier to share, and d) provide an abstraction between the analysis itself and the processing system that actually runs over the data. We started by investigating how industry-standard tools, in particular the Pandas library [1], could help with some of these challenges, which we presented previously [2]. Here we describe how this has led to a complete set of high-level analysis tools.

[*]Speaker. e-mail: fast-hep@cern.ch

2 Key design considerations

To simplify both use and setup of the FAST analysis tools, the software dependencies are significantly reduced with respect to common HEP analyses.To achieve this we build on Python packages provided by Scikit-HEP (e.g. uproot) and the wider Python community (e.g. pandas). Furthermore, to start analysing data only basic knowledge of YAML is required, while the description of selections and other algorithms is kept close to their mathematical form. The definition of an analysis can then be split into three distinct steps, each addressing a different part of the analysis, and with corresponding FAST tools:

1. What input datasets do you need and what are their analysis-specific metadata? Handled by: `fast-curator`
2. What do you want to do with this data? Handled by `fast-carpenter`
3. What do you want to show of this data? Handled by `fast-plotter`

To use Our primary interface to these tools right now is through YAML [3] configuration files, which are widely used for the description of processing steps for tools like continuous integration (e.g. on gitlab, Travis, Azure), container orchestration (e.g. Ansible, Kubernetes), and within particle physics is used for preservation of data on the HEPData platform. As a superset of JSON, it lacks any flow control structures but makes it easier to read for example by reducing the number of brackets and quotation marks in the document, preferring indentation for item separation and adds options to reuse sections with a reference and anchor system. These features make it natural to use in a declarative approach where the user describes *what* to do, but lets the system take care of *how* to achieve this—addressing the separation between processing system and actual analysis decisions. Furthermore, it allows for various accelerating techniques to happen behind the scenes, such as memoization or the omission of unnecessary analysis steps.

3 Describing input data: `fast-curator`

As an analysis evolves, the input data might change too. New datasets can become available as a physics run continues, simulations are extended, or existing data is reprocessed through early stages. Alternatively, metadata of a given data set might be updated, such as various scale factors or cross sections.

Input data are described as one or more data sets in YAML files that are generated and interpreted using the `fast-curator` package. Listing 1 shows such an example: data sets and their meta data can be defined in either a single YAML file (e.g. `data` and `DY` definitions, lines 2-12) or imported from other YAML files (`WW` and `WZ` data sets, lines 19+). Note that some of the information there is redundant, e.g. the number of events in the files. This information is useful to keep track of changes to the analysis' input datasets as new samples become available. Input files can be either provided individually or via wildcards to the `fast-curator` command which can work on local files or via the XRootD [4] protocol. This mechanism can be extended to support experiment-specific file catalogues, with CMS' DAS [5] already included, and with plans to add a Rucio [6] interface in the near future.

4 Describing data processing: `fast-carpenter`

The core of any particle physics analysis is the processing of event-level data.

In `fast-carpenter` the data processing steps are defined as "stages" in a YAML file. The example shown in listing 2 uses three built-in stages: one to define new variables (lines 9

```
datasets:
  - files:
      - input_files/HEPTutorial/files/data.root
    eventtype: data
    name: data
    nevents: 469384
  - files:
      - input_files/HEPTutorial/files/dy.root
      - input_files/HEPTutorial/files/dy_2.root
    name: dy
    nevents: 77729
    nfiles: 2

defaults:
  eventtype: mc
  nfiles: 1
  tree: events

import:
  - "{this_dir}/WW.yml"
  - "{this_dir}/WZ.yml"
```

Listing 1: Dataset description for fast-curator. Contains datasets for real data and one simulation (dy) and imports for two more (WW and WZ). Note that some of the redundant information is only there to assist book-keeping and validating changes to the input datasets as new samples become available. They are not all required by `fast-carpenter` and later stages.

to 15), another to remove events based on a series of cuts (lines 30 to 38), and two instances of binned dataframe production i.e. histograms (lines 23 to 28 and 39 to 43). Beyond these, additional modules produce dataframes with full event-level information and helpers handle systematic weight variables and parameters that define phase-space regions. In addition, the mechanism used to load stages easily extensible; if `fast-carpenter` does not provide a stage that you need for your analysis, it is easy to write it and include it in your workflow. Such a stage is included in the example Listing 2 on lines 4 and then configured on line 17.

Both `CutFlow` and `BinnedDataframe` stages write outputs to disk in the form of a Pandas DataFrame, defaulting to CSV format, although any Pandas-supported format can be used. Binned dataframes generalise histograms to multiple dimensions. Each bin captures the raw number of events within its boundaries and, if one ore more are provided, the sum of each weight and square of weights.

The example in listing 2 starts by defining new variables, such as the transverse momentum of each muon (`Muon_Pt`), whether or not each muon is considered isolated (`IsoMuon_Idx`), how many muons are isolated (`NIsoMuon`) and several variables for di-muon pairs using ean "external" user module, `cms_hep_tutorial.DiObjectMass`. We obtain the 2D distribution for the number of muons and isolated muons in each event, followed by a selection (`EventSelection`), removing events with fewer than 2 isolated muons, failing a trigger requirement, or with a leading muon of `Muon_Pt` less than 25 GeV. The last step, `DiMuonMass`, produces another binned dataframe for the invariant mass of muon pairs.

A key advantage of this approach is that the same processing description can be used on many different processing systems. At the time of writing this includes the use of local multi-core processing and interfaces to the SGE and HTCondor batch systems. Recently an

```
stages:
  - BasicVars: Define # Define new variables
    # A custom class to form the invariant mass of a two-object system
  - DiMuons: cms_hep_tutorial.DiObjectMass
  - NumberMuons: BinnedDataframe # Filled a binned dataframe
  - EventSelection: CutFlow # Select events by applying cuts
  - DiMuonMass: BinnedDataframe # Fill another binned dataframe

BasicVars:
  variables:
    - Muon_Pt: "sqrt(Muon_Px ** 2 + Muon_Py ** 2)"
    - IsoMuon_Idx: (Muon_Iso / Muon_Pt) < 0.10
    # This next variable will create a single number for each event,
    # using a set of inputs whose length varies for each event
    - NIsoMuon: {reduce: count_nonzero, formula: IsoMuon_Idx}
# Custom module specific to this analysis uses one an optional parameter
DiMuons: {mask: IsoMuon_Idx}
# Make a binned dataframe with a column for:
#  - the dataset name
#  - the number of muons
#  - the number of muons considered "isolated"
#  and weight everything using the EventWeight variable
NumberMuons:
  dataset_col: true
  binning:
    - {in: NMuon, out: nMuons}
    - {in: NIsoMuon, out: nIsoMuons}
  weights: {weighted: EventWeight}
# Subsequent stages only see events that pass the following requirements
EventSelection:
  selection:
    All:
      - NIsoMuon >= 2
      - triggerIsoMu24 == 1
      - {reduce: 0, formula: Muon_Pt > 25}
  # Weight events in the resulting cut flow table
  weights: {weighted: EventWeight}
# Binned dataframe using the dataset and the binned DiMuon_mass
DiMuonMass:
  dataset_col: true
  binning:
    - {in: DiMuon_Mass, out: dimu_mass, bins: {low: 60, high: 120, nbins: 60}}
  weights: {weighted: EventWeight}
```

Listing 2: Data processing description example for fast-carpenter. Firstly, we define what stages we want to apply and then we provide descriptions for each stage.

interface to Coffea [7] has been included. Although this only uses the local multiprocessing options for now, we anticipate expanding it to include other systems as well, e.g. Spark. The user can control which back-end is used via a command-line flag.

5 Describing visualisation: `fast-plotter`

To turn the binned dataframes produced by `fast-carpenter` into plots, the `fast-plotter` tool is provided. The goal with this package is to provide useful defaults which make reasonable-looking plots easily, but to write this from many small functions which themselves are useful in a user's custom scripts. The example in Listing 3 controls the labels for the axes and plots, the way the legend is displayed, which annotations are added to the figure, and the range for the y-axis.

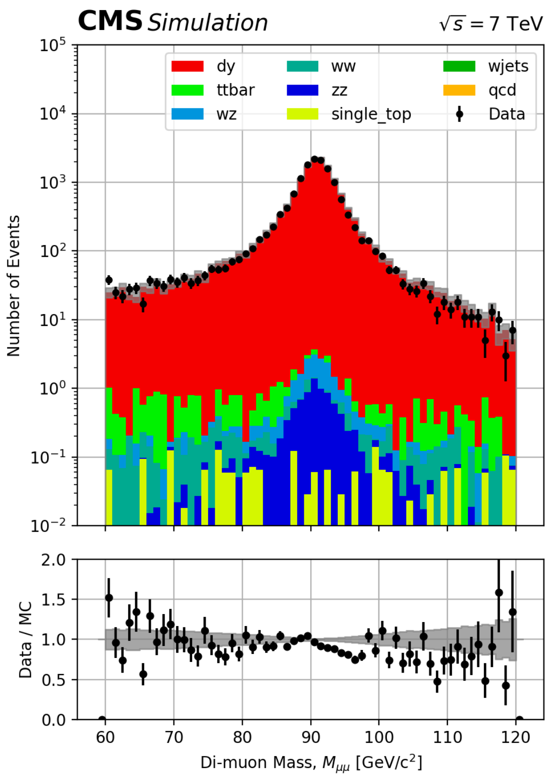
6 Interoperability of the FAST-HEP tools

Figure 1 shows the interplay between a user's repository and the various FAST-HEP packages. A user should typically only write their YAML configuration files, and possibly custom `fast-carpenter` extension stages and additional post-processing code. These will be passed to the corresponding tool to produce the final analysis results. In addition, to the tools

```yaml
weights: [weighted]
yscale: linear
ylabel: Number of Events
figsize: [5, 7]
legend:
    ncol: 3
    loc: 'upper right'
limits: {y: [1e-2, 1e5]}

annotations:
    - text: CMS
      position: [0, 1.03]
      fontweight: bold
      fontsize: xx-large
    - text: Simulation
      fontsize: x-large
      fontstyle: italic
      position: [0.15, 1.03]
    - text: '$\sqrt{s} = 7$ TeV'
      position: [1, 1.03]
      horizontalalignment: right
      fontsize: large

bin_variable_replacements:
    dimu_mass: >-
        Di-muon Mass,
        $M_{\mu\mu}$
        [GeV/c$^{2}$]
```



Listing 3: Left: Example visualisation description for `fast-plotter`; Right: the result of using this description on the binned dataframe for the `DiMuonMass` stage from 2.

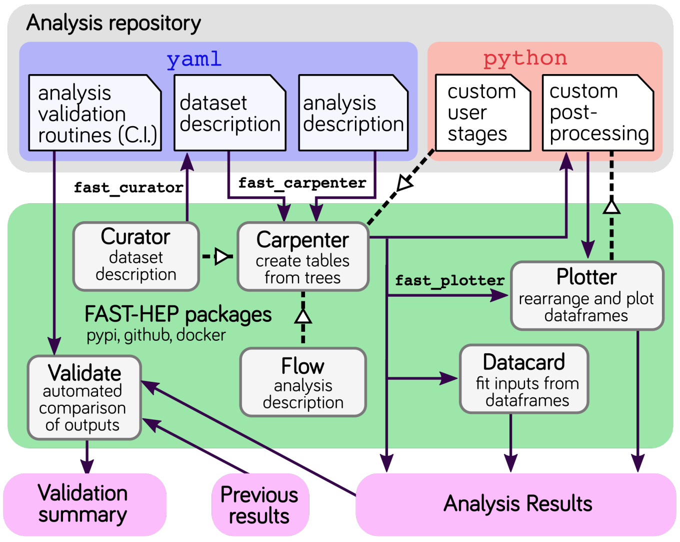


Figure 1: The interplay between FAST-HEP packages, a user's repository, and the analysis results. A user's repository contains mainly YAML files and some custom python code, which are used to control the FAST-HEP packages. Command-line programs are shown in typewriter fonts along arrows.

mentioned above, the FAST-HEP toolkit also provides the `scikit-validate` package to help validate analysis results in a continuous integration (CI) pipeline.

All of these tools are available to download from the Python package index (pypi). Documentation is available online for `fast-carpenter` and `fast-plotter` and for the overall project at fast-hep.web.cern.ch.

The YAML-based interfaces described above are the primary way to use these tools, however we have started investigating other approaches, in particular, using the packages' python APIs directly from Jupyter notebooks.

References

[1] W. McKinney, *Data Structures for Statistical Computing in Python*, in *Proceedings of the 9th Python in Science Conference* (2010), pp. 51 – 56

[2] B.E. Krikler, O. Davignon, L. Kreczko, J. Linacre, E.O. Olaiya, T. Sakuma, EPJ Web Conf. **214**, 06035 (2019)

[3] *YAML website*, http://yaml.org/, accessed: 2020-03-13

[4] *XRootD website*, http://xrootd.org/, accessed: 2020-03-13

[5] V. Kuznetsov, D. Evans, S. Metson, Procedia Computer Science **1**, 1535 (2010), iCCS 2010

[6] M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron, D. Christidis, D. Ciangottini, G. Dimitrov, M. Elsing et al., Computing and Software for Big Science **3**, 11 (2019)

[7] L. Gray, N. Smith, A. Novak, D. Taylor, P. Gessinger, J. Pata, A. Woodard, M. Verzetti, Andreas, dnoonan08 et al., *CoffeaTeam/coffea: Release v0.6.35* (2020), `https://doi.org/10.5281/zenodo.3708080`