Pandas DataFrames for a F.A.S.T. binned analysis at CMS

Olivier Davignon, Lukasz Kreczko, Ben Krikler, Jacob Linacre, Emmanuel Olatunji Olaiya, Tai Sakuma

10th July 2018
The F.A.S.T. approach
Using modern tools for HEP analyses
What we’ve built, some examples, and lessons learnt
Faster Analysis Software Taskforce

**fast** [fäst, fahst]

adjective, fast·er, fast·est.

1. moving or able to move, operate, function, or take effect quickly; quick; swift; rapid: a fast horse; a fast pain reliever; a fast thinker.
2. done in comparatively little time; taking a comparatively short time: a fast race; fast work.

- Founded in May 2017
- A group of HEP scientists / researchers
  - All primarily working on CMS so far
  - All primarily working in UK institutions
- Regular hack-days for prototyping, experimenting
• Searching for SUSY in tails of distributions
  • e.g. missing transverse energy

• The analysis is technically demanding:
  • A complex event selection
  • Signal region: 253 bins over 4 variables
  • 19 sources of systematic error considered
The code has evolved organically
- Many heterogeneous internal formats
- Complicated data-reduction sequence over many packages

Very steep learning curve
- Ad-hoc documentation
- High “bus factor”: only one person knows how to run each step

Difficult to extend or adapt

Few weeks to process all datasets start to finish
- Improved by CMS new central NanoAOD format
The FAST analysis approach

- No skimmed tree stage
- Single, internal data format
- Use Pandas as a generalisation of multi-dimensional histograms:
  - Feature-rich toolkit to manipulate, combine, and visualise
- Could largely be run in Python notebooks
What is Pandas?

- A python package for handling tabular data
  - A Pandas dataframe == a programmatic table

- Feature rich:
  - Input / output in many formats (csv, hdf, excel, etc)
  - Table manipulations
  - Plotting

- [https://pandas.pydata.org/](https://pandas.pydata.org/)
FAST Objectives

1. Newcomers should produce useful research output within a week (i.e. fast)
2. Analysis code should allow for fast prototyping
3. Code accompanied by good documentation with code examples for fast lookup
4. Automated physics validation for fast bug detection on code changes
FAST Objectives

Learn how to use modern tools so physicists can ask more:

“What do I want to study”

and less: “how do I have to do this”

(similar to HSF CWP sentiment)
That’s enough talk...

Code and examples

Based on the open-access CMS HEP Tutorial: http://ippog.org/resources/2012/cms-hep-tutorial

Data and 8 MC components for 50 pb$^{-1}$ from 2011. Builds up to a tt-bar analysis
Summarising ROOT Trees

- **AlphaTwirl**
  - See detailed poster on AlphaTwirl
    [https://indico.cern.ch/event/587955/contributions/2937634/](https://indico.cern.ch/event/587955/contributions/2937634/)
  - Summarizes event-level data to binned data
  - Highly general, but our use:
    - Input: ROOT Trees
    - Output: Pandas dataframes, 1 row **per bin**

- **FAST adds a YAML control interface**
  - Condense analysis decisions (eg. what variable to cut on, binning widths) away from code
  - Easier to share, simpler to read, harder to make bugs
1. # Control the processing order
2. stages:
3.   - selection: {type: CutFlow}
4.   - DiMuon: {type: BinnedDataframe}
5. 
6. # Apply an event selection
7. selection:
8.   selection:
9.     All:
10.       - len(ev.Muon_Iso_Idx) >= 2
11.       - ev.triggerIsoMu24[0]
12.       - ev.Muon_Pt[0] > 25
13. 
14. # Define a binned dataframe: one discrete, one continuous variable which we bin
15. DiMuon:
16.   binning:
17.     - {in: comp_name, out: component}
18.     - {in: DiIsoMuon_Mass, out: dimu_mass, bins: {low: 60, high: 120, nbins: 60}}
19. weights: {weighted: EventWeight, unweighted: 1}
# Apply an event selection

Selection:

All:

- \(\text{len(ev.Muon}_{-}\text{Iso}_\text{Idx}) \geq 2\)
- \(\text{ev.triggerIsoMu24}[0]\)
- \(\text{ev.Muon}_{-}\text{Pt}[0] > 25\)
Example Binned Dataframe

```python
# Define a binned dataframe: one discrete, one continuous variable which we bin
DiMuon:
Binning:
- {in: comp_name, out: component}
- {in: DiIsoMuon_Mass, out: dimu_mass, bins: {low: 60, high: 120, nbins: 60}}
weights: {weighted: EventWeight, unweighted: 1}
```

- Only show first 5 bins of each data / MC component
- Columns:
  - Component, dimu_mass = bin labels
  - n = bin content
  - nvar = variance on bin
- n != nvar for MC due to event weighting

<table>
<thead>
<tr>
<th>component</th>
<th>dimu_mass</th>
<th>n</th>
<th>nvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>-inf</td>
<td>993</td>
<td>993</td>
</tr>
<tr>
<td>data</td>
<td>60</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>data</td>
<td>61</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>data</td>
<td>62</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>dy</td>
<td>-inf</td>
<td>655.571</td>
<td>1017.55</td>
</tr>
<tr>
<td>dy</td>
<td>60</td>
<td>23.9632</td>
<td>12.0911</td>
</tr>
<tr>
<td>dy</td>
<td>61</td>
<td>25.5728</td>
<td>13.0941</td>
</tr>
<tr>
<td>dy</td>
<td>62</td>
<td>29.2716</td>
<td>14.5514</td>
</tr>
<tr>
<td>dy</td>
<td>63</td>
<td>22.9417</td>
<td>11.5845</td>
</tr>
<tr>
<td>single_top</td>
<td>-inf</td>
<td>1.74104</td>
<td>0.100682</td>
</tr>
<tr>
<td>single_top</td>
<td>60</td>
<td>0.055288</td>
<td>0.00426256</td>
</tr>
<tr>
<td>single_top</td>
<td>61</td>
<td>0.058381</td>
<td>3.39996e-05</td>
</tr>
<tr>
<td>single_top</td>
<td>62</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>single_top</td>
<td>65</td>
<td>0.0937</td>
<td>0.00505474</td>
</tr>
<tr>
<td>ttbar</td>
<td>-inf</td>
<td>11.393</td>
<td>3.07205</td>
</tr>
<tr>
<td>ttbar</td>
<td>60</td>
<td>0.840432</td>
<td>0.23649</td>
</tr>
<tr>
<td>ttbar</td>
<td>61</td>
<td>0.319709</td>
<td>0.0759857</td>
</tr>
<tr>
<td>ttbar</td>
<td>62</td>
<td>0.274432</td>
<td>0.075313</td>
</tr>
<tr>
<td>ttbar</td>
<td>63</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Calculate the total predicted background by summing over all components

- Note: in this order the total errors are wrong
- This is why we store the variance instead

```python
df["err"] = np.sqrt(df.nvar)
df.groupby("component").nth(1)

<table>
<thead>
<tr>
<th>dimu_mass</th>
<th>err</th>
<th>n</th>
<th>nvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>60.0</td>
<td>6.164414</td>
<td>38.000000</td>
</tr>
<tr>
<td>dy</td>
<td>60.0</td>
<td>3.477322</td>
<td>23.963227</td>
</tr>
<tr>
<td>single_top</td>
<td>60.0</td>
<td>0.065288</td>
<td>0.065288</td>
</tr>
<tr>
<td>ttbar</td>
<td>60.0</td>
<td>0.486302</td>
<td>0.840432</td>
</tr>
<tr>
<td>wjets</td>
<td>60.0</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>ww</td>
<td>60.0</td>
<td>0.063284</td>
<td>0.063284</td>
</tr>
<tr>
<td>wz</td>
<td>60.0</td>
<td>0.037740</td>
<td>0.053328</td>
</tr>
<tr>
<td>zz</td>
<td>60.0</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

total = df[df.component != "data"].groupby("dimu_mass").sum()
total.head()
```

<table>
<thead>
<tr>
<th>dimu_mass</th>
<th>n</th>
<th>nvar</th>
<th>err</th>
</tr>
</thead>
<tbody>
<tr>
<td>-inf</td>
<td>0.73297897</td>
<td>10.2105132</td>
<td>34.894752</td>
</tr>
<tr>
<td>60.000000</td>
<td>24.985559</td>
<td>12.337321</td>
<td>4.129846</td>
</tr>
<tr>
<td>61.000000</td>
<td>26.000434</td>
<td>13.175767</td>
<td>3.975014</td>
</tr>
<tr>
<td>62.000000</td>
<td>29.653237</td>
<td>14.632247</td>
<td>4.186596</td>
</tr>
<tr>
<td>63.000000</td>
<td>23.145460</td>
<td>11.597231</td>
<td>3.525337</td>
</tr>
</tbody>
</table>
Depending on task, "wide-form" tables can be easier to work with.
Turning these into plots
Few lines of pure Pandas code: Dataframe $\rightarrow$ plot

```
data = df2.xs("data", level=1, axis=1).reset_index()
sims = df2.drop("data", axis=1, level=1).

ax = plt.subplot(111)
sims.plot.line(linestyle="steps-mid", stacked=True, ax=ax, zorder=-1, logy=True)
data.plot.scatter(x="dimu_mass", y="n", xerr="err", color="k", label="data", ax=ax)

plt.ylim([0.7, 1.04])
plt.xlabel(r"m$_{\text{dimu}}$")
plt.ylabel("Events")
plt.legend()
```
Using in Real Analysis

- Aux material for JHEP 1805 (2018) 025

<table>
<thead>
<tr>
<th>Event Selection</th>
<th>Benchmark Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T2bb</td>
</tr>
<tr>
<td>$m_{\text{SUSY}}$ (GeV)</td>
<td>550</td>
</tr>
<tr>
<td>$m_{\text{LSP}}$ (GeV)</td>
<td>450</td>
</tr>
<tr>
<td>$c_{\tau_0}$ (mm)</td>
<td>—</td>
</tr>
<tr>
<td>Before selection</td>
<td>100.0</td>
</tr>
<tr>
<td>Single isolated track, muon, electron, &amp; photon vetos</td>
<td>94.6</td>
</tr>
<tr>
<td>Event veto for jets failing ID</td>
<td>94.3</td>
</tr>
<tr>
<td>$p_T^j &gt; 100$ GeV</td>
<td>62.9</td>
</tr>
<tr>
<td>$0.1 &lt; f_{hh}^j &lt; 0.95$</td>
<td>59.8</td>
</tr>
<tr>
<td>$H_T &gt; 200$ GeV</td>
<td>49.5</td>
</tr>
<tr>
<td>$H_T^{\text{miss}} &gt; 200$ GeV</td>
<td>18.8</td>
</tr>
<tr>
<td>Event veto for forward jets ($</td>
<td>\eta</td>
</tr>
<tr>
<td>$H_T^{\text{miss}}/E_T^{\text{miss}} &lt; 1.25$</td>
<td>12.9</td>
</tr>
<tr>
<td>$n_{\text{jet}}^*$ and $H_T$-dependent $\alpha_T$ thresholds</td>
<td>8.3</td>
</tr>
<tr>
<td>$\Delta\phi_{\text{min}}^* &gt; 0.5$</td>
<td>5.7</td>
</tr>
</tbody>
</table>

- Includes full event selection, and all event weighting routines
Running a fit from Dataframes

- Final step of many analyses: extract signal strength via a fit
- Working on interface to fitting routines:
  - Starting with CMS Higgs Combine tool
- Validate against limit from JHEP 1805 (2018) 025
  - Can reproduce expected limit from left plot

10 July 2018, b.krikler@cern.ch
The analysis code’s ecosystem

- Use CERN GitLab
  - Runners for Continuous Integration
  - Access to EOS

- Run sections of analysis chain in CI
  - Unit testing of analysis code
  - Integration tests running full analysis chains on real data (stored on EOS)

- Auto-documentation-ing
  - Published to a CERN-hosted webpage
The analysis code’s ecosystem

Book-keeping of dataframes

- Stored inside the persisted Dataframes
- Dictionary of software versions, command-line options, important environment variables, provenance info
- If persisted as text: stored as YAML comment header
- If persisted as HD5: extra key in file (in development)
Conclusions

- Introduced you to F.A.S.T.
- Presented a case-study of reimplementing an analysis with a modern approach
- Big improvement over current approach
- Being adopted by several CMS analyses, interest from members of DUNE
Conclusions: Lessons Learnt

- A full analysis has many aspects
  - Some shortcomings in pure-python world:
    - Statistics that ROOT provides (e.g., TEfficiency)
    - Fitting routines
    - Plotting binned dataframes

- Power of Continuous Integration for analyses
  - Essentially free PhD students!
Conclusions: Next Steps

- Produce set of benchmarks and comparisons
  - Code and performance metrics
- Release code as an example or template for others
- Develop generic routines for manipulating binned dataframes
- Help plug identified shortcomings
  - Eg. Contributions to Pandas, Histbook, Seaborn, etc
WANNA TRY THIS NEW TOOL? IT'S REALLY FAST AND POWERFUL.

SORRY, NO TIME. I NEED TO CUT DOWN THIS TREE.
Back-ups
Turning these into plots

Few more lines: ROOT style plot

data = df2.xs("data", level=1, axis=1).reset_index()
sims = df2.drop("data", axis=1, level=1).n
sims = sims.fillna(0).cumsum(axis=1)
ax = plt.subplot(111)
ax.set_prop_cycle("color", "mcbgrybr")
def fill_coll(col, **kwargs):
    plt.fill_between(x=col.index.values, y1=col.values, label=col.name, **kwargs)
sims.iloc[:, 1:].apply(fill_coll, axis=0, step="mid")
data.plot.scatter(x="dimu_mass", y="n", yerr="err", color="k", label="data", ax=ax)
plt.ylim([0.7, 1e4])
plt.xlim([60, 120])
plt.xlabel("m_{#mu#nu}\text{ (GeV)}")
plt.ylabel("Events")
plt.legend(loc="upper left")
plt.yscale("log")

Our version

Tutorial’s version