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

University of Bristol

10th July 2018

Science & Technology Facilities Council
UK Research and Innovation

Outline

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 *[făst, fahst]*

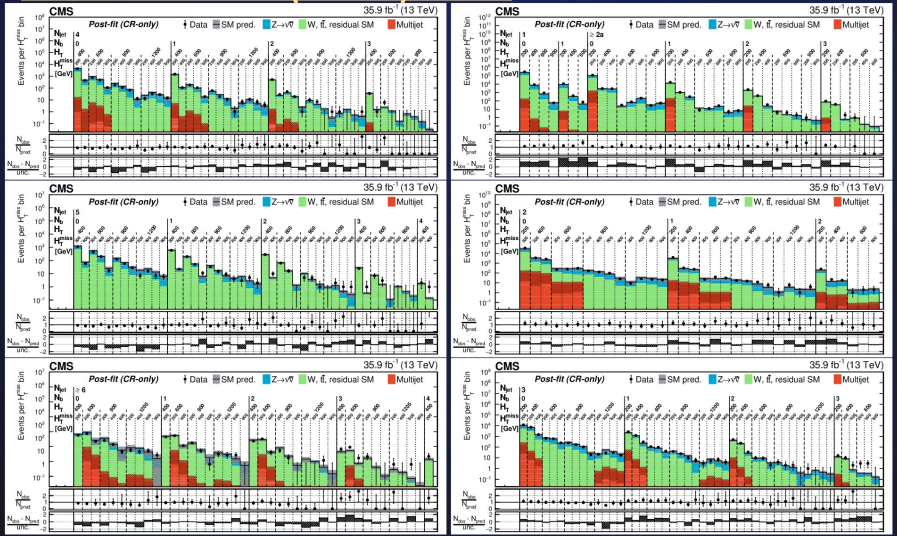
adjective, fast·er, fast·est.

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

Context: CMS SUSY AlphaT analysis

JHEP 1805 (2018) 025

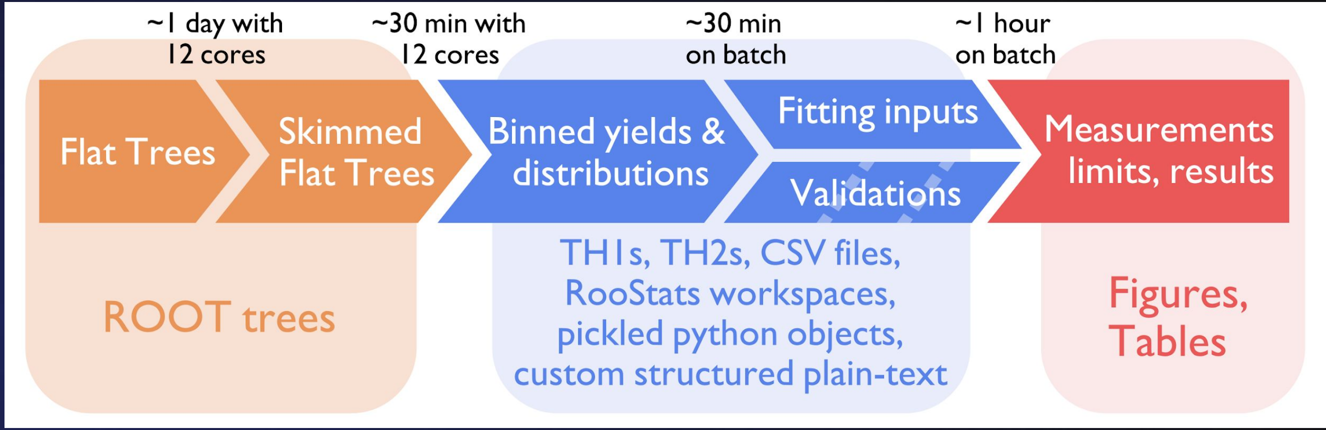


y-axis: bin yield →

x-axis: bin number →

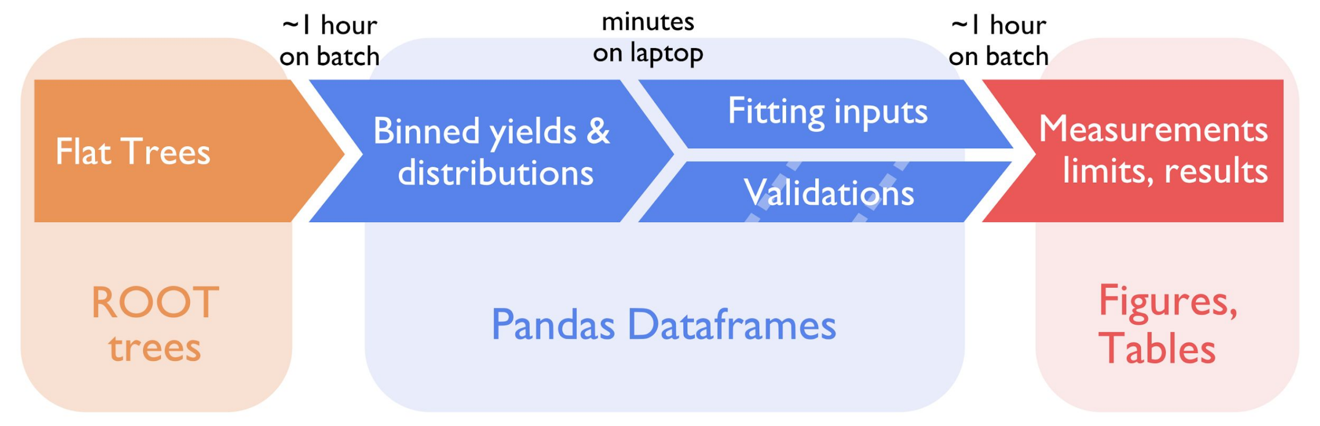
- Searching for SUSY in tails of distributions
  - eg. missing transverse energy
- The analysis is technically demanding:
  - A complex event selection
  - Signal region: 253 bins over 4 variables
  - 19 sources of systematic error considered

The original analysis framework



- 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/

```python
A = ['foo', 'bar', 'foo', 'bar']
B = ['one', 'one', 'two', 'three']
C = np.random.randn(4)
D = np.random.randn(4)

df = pd.DataFrame({"A": A, "B": B,
                   "C": C, "D": D})
```

```python
df
```

	A	B	C	D
0	foo	one	-0.678386	0.072926
1	bar	one	-0.338564	-1.038362
2	foo	two	0.527912	-0.478806
3	bar	three	-0.237991	-1.296666

```python
df.set_index(["A", "B"])
```

		C	D
A	B		
foo	one	-0.678386	0.072926
bar	one	-0.338564	-1.038362
foo	two	0.527912	-0.478806
bar	three	-0.237991	-1.296666

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/
  - 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

Summarising ROOT Trees: Config

```yaml
# Control the processing order
stages:
    - selection: {type: CutFlow}
    - DiMuon: {type: BinnedDataframe}

# Apply an event selection
selection:
    selection:
        All:
            - len(ev.Muon_Iso_Idx) >= 2
            - ev.triggerIsoMu24[0]
            - ev.Muon_Pt[0] > 25

# 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}
```

Example Cut-flow Dataframe

component	depth	class	name	pass	total
data	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	16208	469384
		LambdaStr	ev: ev.triggerIsoMu24[0]	16208	16208
		LambdaStr	ev: ev.Muon_Pt[0] > 25	15995	16208
dy	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	37559	77729
		LambdaStr	ev: ev.triggerIsoMu24[0]	37559	37559
		LambdaStr	ev: ev.Muon_Pt[0] > 25	37263	37559
qcd	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	0	142
		LambdaStr	ev: ev.triggerIsoMu24[0]	0	0
		LambdaStr	ev: ev.Muon_Pt[0] > 25	0	0
single_top	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	111	5684
		LambdaStr	ev: ev.triggerIsoMu24[0]	111	111
		LambdaStr	ev: ev.Muon_Pt[0] > 25	110	
ttbar	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	226	
		LambdaStr	ev: ev.triggerIsoMu24[0]	206	
		LambdaStr	ev: ev.Muon_Pt[0] > 25	206	
wjets	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	1	
		LambdaStr	ev: ev.triggerIsoMu24[0]	1	
		LambdaStr	ev: ev.Muon_Pt[0] > 25	1	
ww	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	244	
		LambdaStr	ev: ev.triggerIsoMu24[0]	244	
		LambdaStr	ev: ev.Muon_Pt[0] > 25	243	
wz	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	623	
		LambdaStr	ev: ev.triggerIsoMu24[0]	623	
		LambdaStr	ev: ev.Muon_Pt[0] > 25	623	
zz	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	1235	
		LambdaStr	ev: ev.triggerIsoMu24[0]	1235	
		LambdaStr	ev: ev.Muon_Pt[0] > 25	1232	

```
# Apply an event selection
Selection:
    Selection:
        All:
            - len(ev.Muon_Iso_Idx) >= 2
            - ev.triggerIsoMu24[0]
            - ev.Muon_Pt[0] > 25
```

component	depth	class	name	pass	total
data	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	16208	469384
		LambdaStr	ev: ev.triggerIsoMu24[0]	16208	16208
		LambdaStr	ev: ev.Muon_Pt[0] > 25	15995	16208
dy	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	37559	77729
		LambdaStr	ev: ev.triggerIsoMu24[0]	37559	37559
		LambdaStr	ev: ev.Muon_Pt[0] > 25	37263	37559

Example Binned Dataframe

```
df.groupby("component").head().style.hide_index()
```

component	dimu_mass	n	nvar
data	-inf	993	993
data	60	38	38
data	61	25	25
data	62	22	22
data	63	28	28
dy	-inf	655.571	1017.55
dy	60	23.9632	12.0911
dy	61	25.5728	13.0941
dy	62	29.2716	14.5514
dy	63	22.9417	11.5845
single_top	-inf	1.74104	0.100682
single_top	60	0.065288	0.00426256
single_top	61	0.005831	3.39996e-05
single_top	62	0	0
single_top	65	0.0937	0.00505474
ttbar	-inf	11.393	3.07205
ttbar	60	0.840432	0.23649
ttbar	61	0.319709	0.0759857
ttbar	62	0.274432	0.075313
ttbar	63	0	0

wjets			
wjets	60	0	0
ww	-inf	3.60022	0.221474
ww	60	0.063284	0.00400488
ww	61	0.102053	0.00561706
ww	62	0.068484	0.00469004
ww	63	0.194258	0.0126013
wz	-inf	0.320914	0.00784179
wz	60	0.053328	0.00142432
wz	61	0	0
wz	62	0.038697	0.000843894
wz	63	0	0
zz	-inf	0.360053	0.00298072
zz	60	0	0
zz	63	0.009475	8.97719e-05
zz	64	0.00954	6.65446e-05
zz	65	0.004153	1.7248e-05

```
# 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

Manipulating DataFrames

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

	dimu_mass	err	n	nvar
component				
data	60.0	6.164414	38.000000	38.000000
dy	60.0	3.477232	23.963227	12.091140
single_top	60.0	0.065288	0.065288	0.004263
ttbar	60.0	0.486302	0.840432	0.236490
wjets	60.0	0.000000	0.000000	0.000000
ww	60.0	0.063284	0.063284	0.004005
wz	60.0	0.037740	0.053328	0.001424
zz	60.0	0.000000	0.000000	0.000000

Convert the variance to the error

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

	n	nvar	err
dimu_mass			
-inf	673.297897	1021.051322	34.894752
60.000000	24.985559	12.337321	4.129846
61.000000	26.000434	13.175767	3.975014
62.000000	29.653237	14.632247	4.186596
63.000000	23.145460	11.597231	3.525337

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

Manipulating DFs: Long to wide form

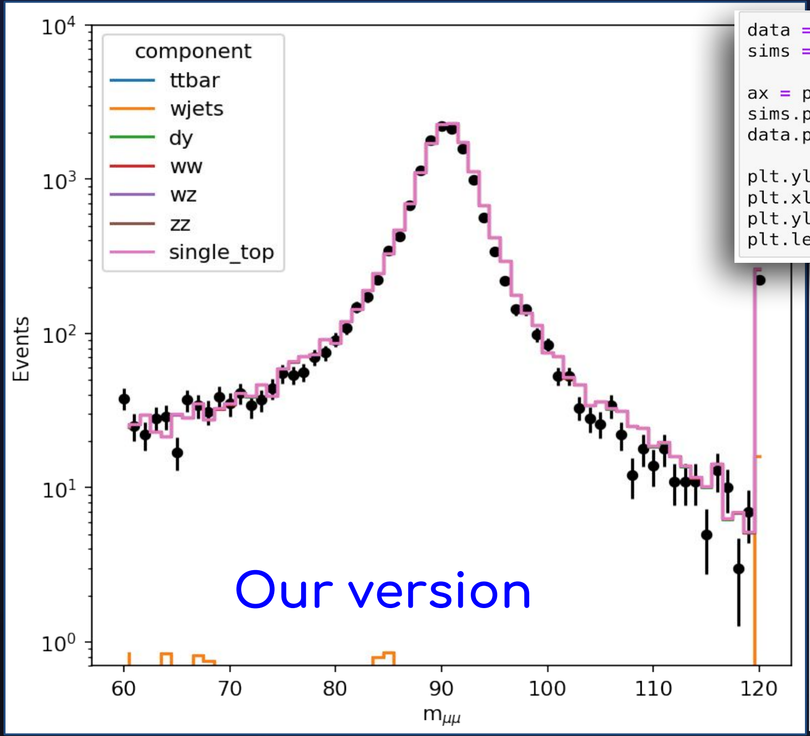
Depending on task, “wide-form” tables can be easier to work with

```
# Convert variance --> error
df["err"] = np.sqrt(df.nvar)

# Switch to long-form
df2 = df.pivot_table(index="dimu_mass", columns="component", values=["n", "err"])
df2 = df2.sort_index(axis=1, ascending=False)

# Sort components to match tutorial
order = ["data", "ttbar", "wjets", "dy", "ww", "wz", "zz", "qcd", "single_top"]
df2 = df2.reindex(order, axis=1, level="component")

# Show first 10 rows
df2.head(10)
```

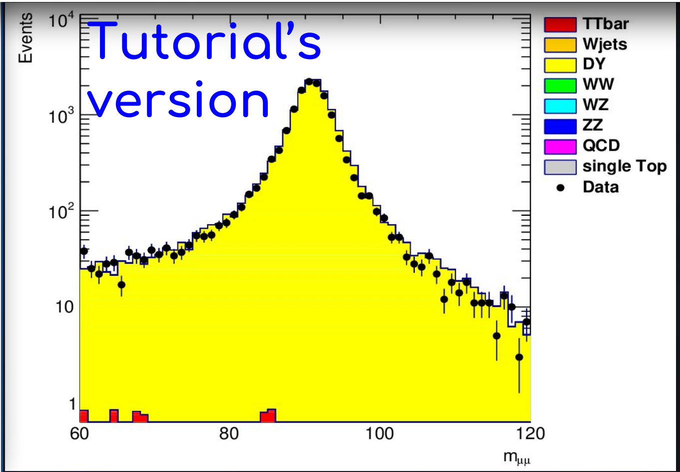
	n								err			
component	**data**	**ttbar**	**wjets**	**dy**	**ww**	**wz**	**zz**	**single_top**	**data**	**ttbar**	**wjets**	**d**
dimu_mass												
-inf	993.0	11.392980	0.311917	655.570771	3.600221	0.320914	0.360053	1.741041	31.511903	1.752727	0.311917	3
60.000000	38.0	0.840432	0.000000	23.963227	0.063284	0.053328	0.000000	0.065288	6.164414	0.486302	0.000000	[illegible]
61.000000	25.0	0.319709	NaN	25.572841	0.102053	0.000000	NaN	0.005831	5.000000	0.275655	NaN	[illegible]
62.000000	22.0	0.274432	NaN	29.271624	0.068484	0.038697	NaN	0.000000	4.690416	0.274432	NaN	[illegible]
63.000000	28.0	0.000000	NaN	22.941727	0.194258	0.000000	0.009475	NaN	5.291503	0.000000	NaN	[illegible]
64.000000	29.0	0.847224	NaN	20.534599	0.065338	0.081642	0.009540	NaN	5.385165	0.490427	NaN	[illegible]
65.000000	17.0	0.352667	NaN	29.464412	0.130224	0.000000	0.004153	0.093700	4.123106	0.282423	NaN	[illegible]
66.000000	37.0	0.570011	NaN	27.861013	0.128668	0.059988	0.015375	0.000000	6.082763	0.403615	NaN	[illegible]

Turning these into plots

Few lines of pure Pandas code: Dataframe → plot




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

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", yerr="err", color="k", label="data", ax=ax)

plt.ylim([0.7, 1e4])
plt.xlabel(r"m$_{\mu\mu}$")
plt.ylabel("Events")
plt.legend()
```



Using in Real Analysis

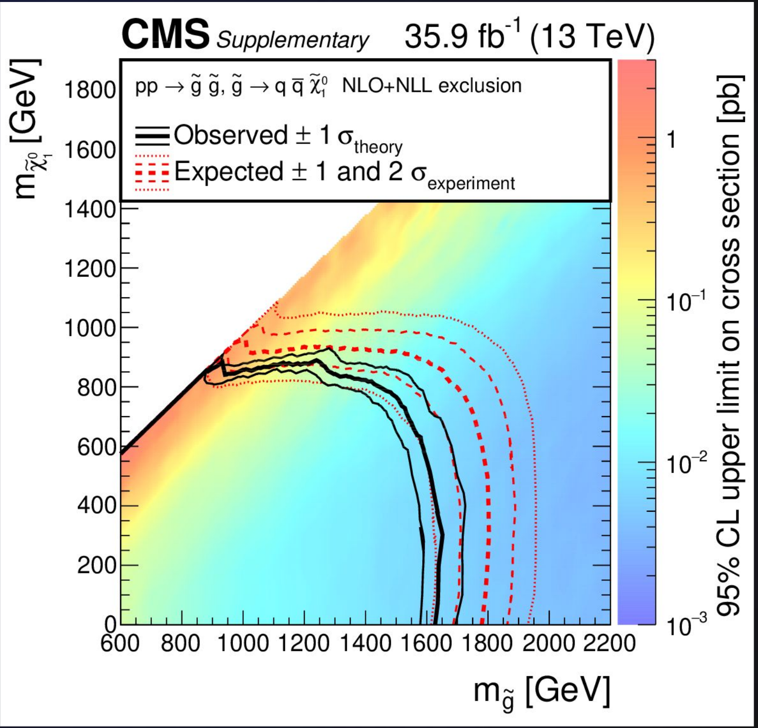
- Aux material for [JHEP 1805 (2018) 025]

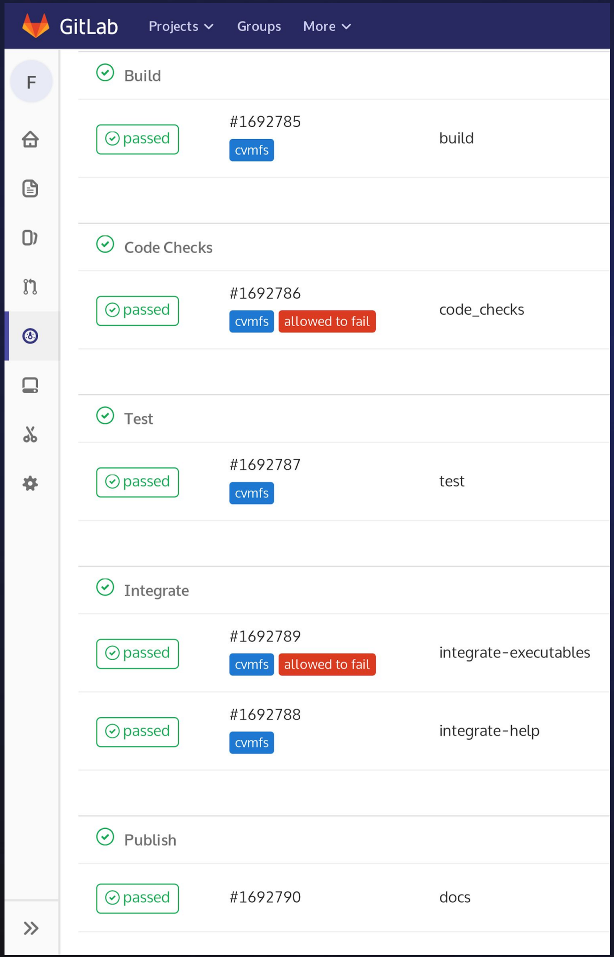
Event Selection		Benchmark Model			
		T2bb	T2cc	T1bbbb	T1qqqqLL
	m_{SUSY} (GeV)	550	500	1900	1800
	m_{LSP} (GeV)	450	480	100	200
	$c\tau_0$ (mm)	–	–	–	1
Before selection		100.0	100.0	100.0	100.0
Single isolated track, muon, electron, & photon vetos		94.6	97.2	99.4	86.0
Event veto for jets failing ID		94.3	96.8	98.7	85.7
$p_{\mathrm{T}}^{\mathrm{j}_1} > 100$ GeV		62.9	84.3	98.7	85.7
$0.1 < f_{\mathrm{h}^\pm}^{\mathrm{j}_1} < 0.95$		59.8	77.4	93.9	82.1
$H_{\mathrm{T}} > 200$ GeV		49.5	64.5	93.9	82.1
$H_{\mathrm{T}}^{\mathrm{miss}} > 200$ GeV		18.8	48.3	88.5	77.4
Event veto for forward jets ($\|\eta\| > 2.4$)		13.6	35.8	69.9	63.7
$H_{\mathrm{T}}^{\mathrm{miss}} / E_{\mathrm{T}}^{\mathrm{miss}} < 1.25$		12.9	34.1	69.3	60.3
n_{jet}- and H_{T}-dependent α_{T} thresholds		8.3	24.9	69.2	60.1
$\Delta\phi^*_{\mathrm{min}} > 0.5$		5.7	20.5	25.1	22.9

- Includes full event selection, and all event weighting routines
    - https://cms-results.web.cern.ch/cms-results/public-results/publications/SUS-16-038/index.html

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





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)

```
#% ===== HEADER (lines = 18)
# ancestors:
# - outputs/tbl_n.component.jet_pt.txt
# - outputs/tbl_cfg_component_phasespace_process.txt
# command_line:
# - t2df_combine_mc_components
# - outputs/backgrounds/
# filename: /home/ben/CMS/FAST/CMS_HEP_tutorial/outputs/tbl_n.proc
# project_dir: /home/ben/CMS/FAST/FAST-RA1/fast_ra1
# software:
#   ROOT_version: v6-08-06
#   pandas_version: 0.23.0
#   python_version: 2.7.13
#   rootpy_version: 1.0.1
#   yaml_version: '3.12'
# version: 0.1.0
# working_dir: /home/ben/CMS/FAST/180701_CMS_HEP_tutorial/
#
#% ===== dataframe (lines = 416)
component    dimu_mass            n             nvar
      data          -inf   993.000000   9.930000e+02
```

Conclusions

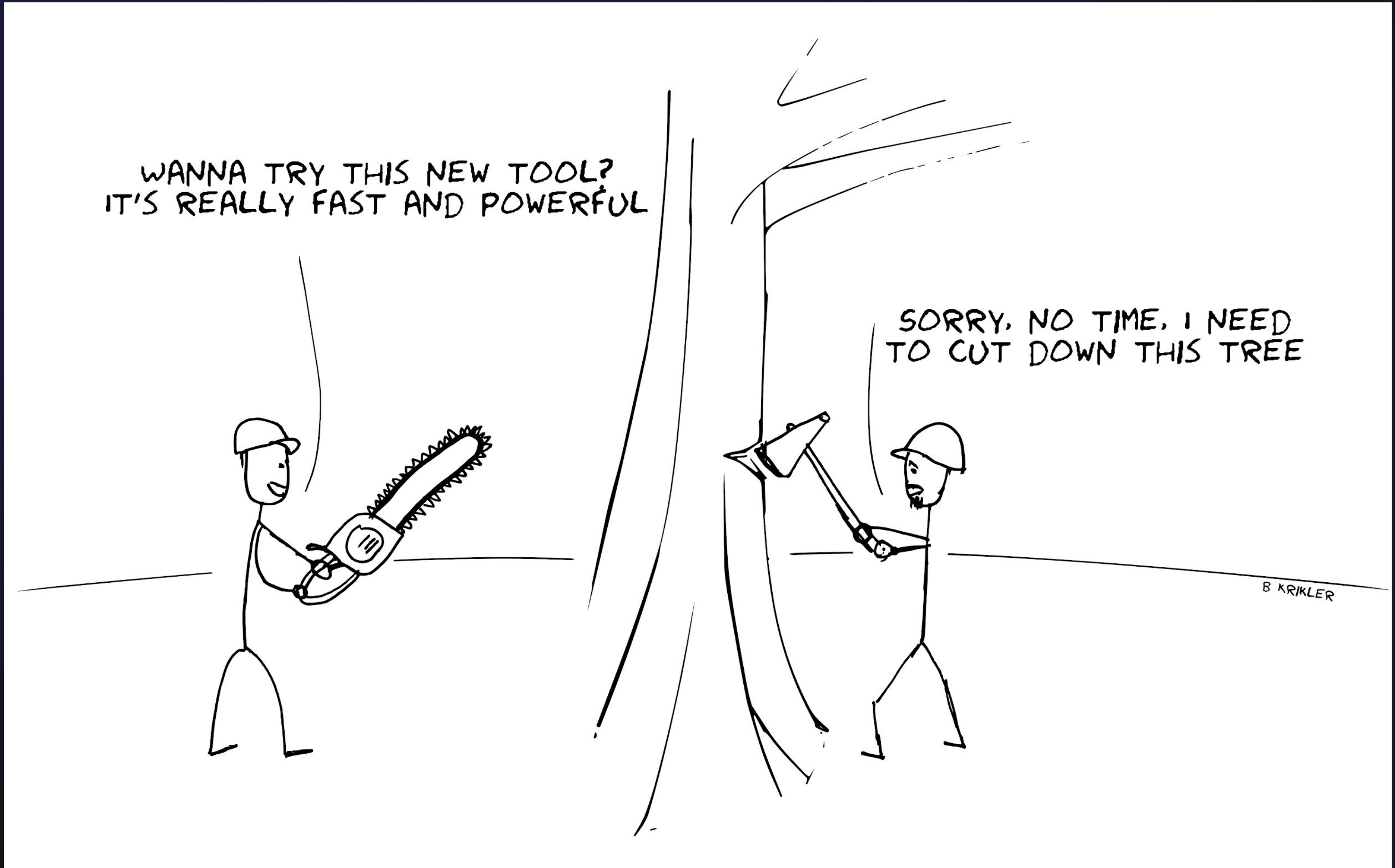
- Introduced you to F.A.S.T.
- Presented a case-study of reimplementing an analysis with a modern approach
- Big improvement over current approach
- Being adopted by several CMS analyses, interest from members of DUNE

Conclusions: Lessons Learnt

- A full analysis has many aspects
  - Some shortcomings in pure-python world:
    - Statistics that ROOT provides (eg 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
B KRIKLER

Thank you!

Back-ups

Turning these into plots

Few more lines:
ROOT style plot

```python
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", "mbcgybr")
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, 119])
plt.xlabel(r"m$_{\mu\mu}$")
plt.ylabel("Events")
plt.legend(loc="upper left")
plt.yscale("log")
```