pyhf introduction

Lukas Heinrich, CERN

What's HistFactory?

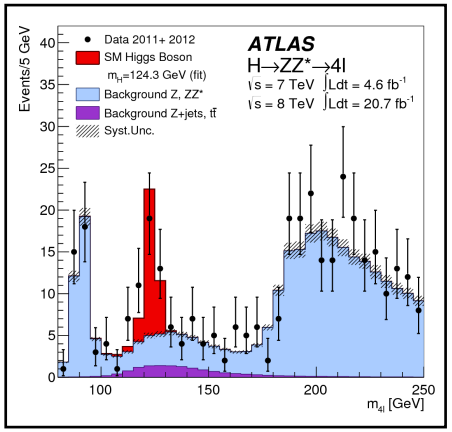
It's a way to express statistical models in a declarative way.

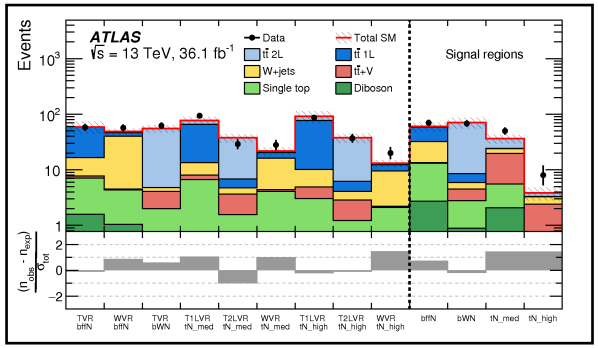
declarative: say what final model you want without specifying how this model is build.

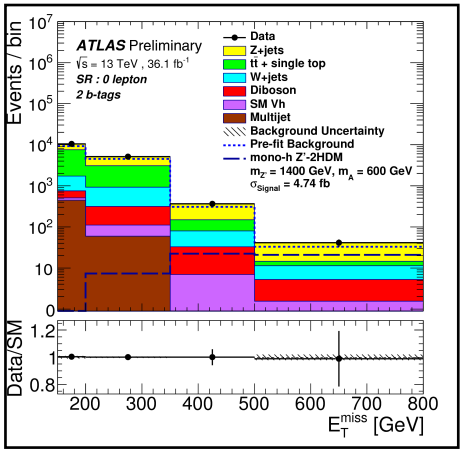
- **allows multiple different implementations in various: frameworks, languages, etc...**
- **allows implementations to optimize how to construct the model for you**

Where is HistFactory used?

all across ATLAS.







Higgs

SUSY

Exotics

If your plots look like this, there's a good chance your analysis is using HistFactory under the hood. Ask your resident statistics contact in your analysis team.

There are now two implementations of HistFactory

1. ROOT
   - this is what HistFitter, WSMaker, TRexFitter use by default

2. pyhf: a new python-only implementation
   - this talk
   - relatively new effort
   - useful when your analysis is using python a lot, or you want to combine ML + stats

The HistFactory template:

Basic Structure: simultaneous fit to

- **multiple channels (think: regions, histogram (stacks))**
- **each region can have multiple bins**
- **coupled to a set of constraint terms**

$$\mathcal{P}(n_c, x_e, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_{c \in \text{channels}} \left[\text{Pois}(n_c|\nu_c) \prod_{e=1}^{n_c} f_c(x_e|\boldsymbol{\alpha}) \right] \cdot G(L_0|\lambda, \Delta_L) \cdot \prod_{p \in \mathbb{S}+\boldsymbol{\Gamma}} f_p(a_p|\alpha_p)$$

Main Measurement

Constraint Terms

In what format can I define HistFactory?

1) Using ROOT Objects
2) XML Files + ROOT
3) JSON Files

new with pyhf

Example JSON:

list of samples in region

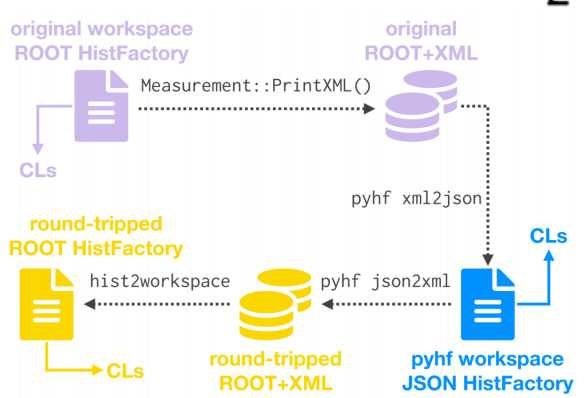
list of systematics free factors/

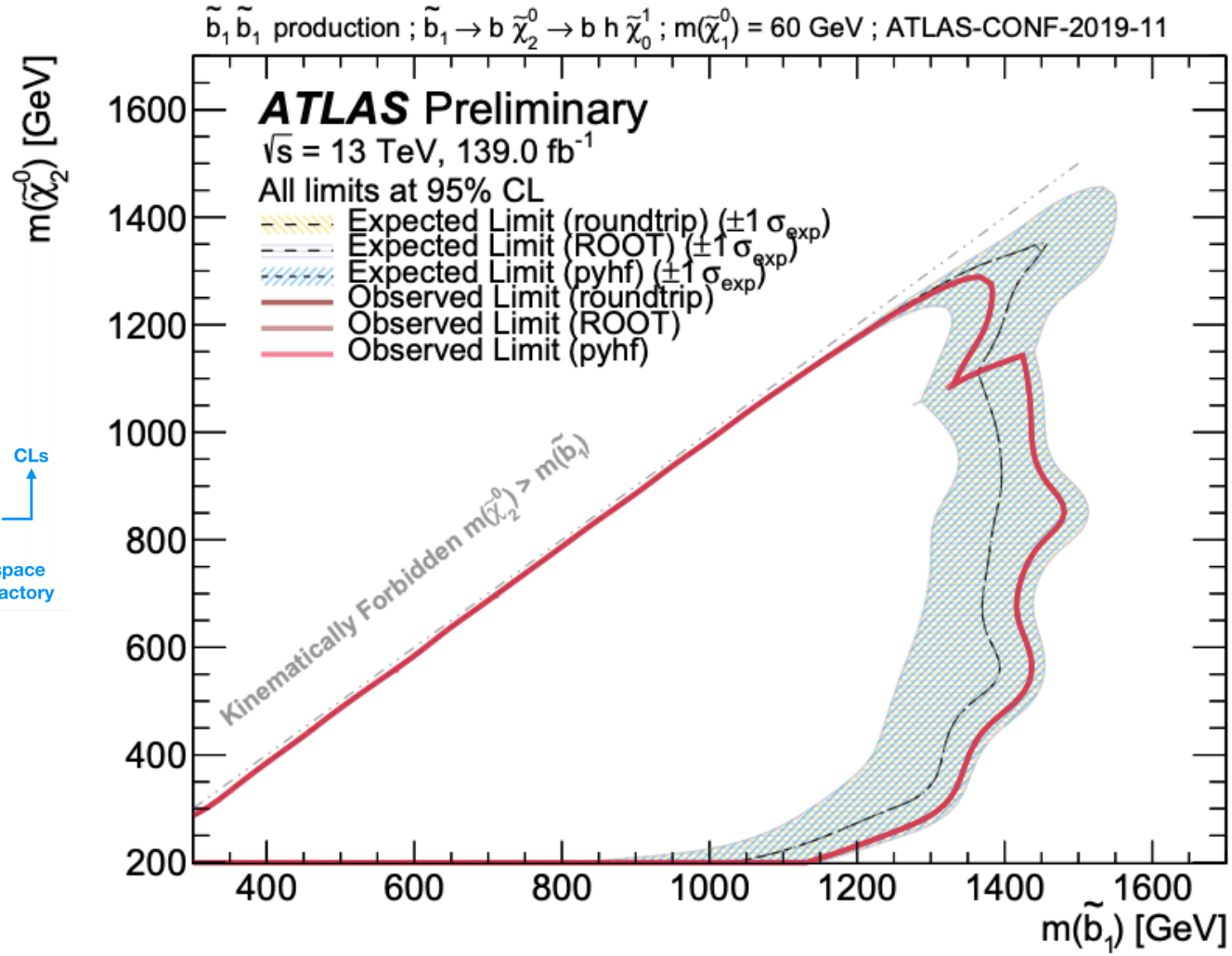
```
{
    "channels": [
        { "name": "singlechannel",
          "samples": [
            { "name": "signal",
              "data": [7.0, 2.0],
              "modifiers": [ { "name": "mu", "type": "normfactor", "data": null} ]
            },
            { "name": "background",
              "data": [50.0, 60.0],
              "modifiers": [ {"name": "uncorr_bkguncrt", "type": "shapesys", "data": [5.0,12.0]} ]
            }
          ]
        }
    ],
    "data": {
        "singlechannel": [50, 60]
    },
    "measurements": [
        { "name": "Measurement", "config": {"poi": "mu", "parameters": []} }
    ]
}
```

list of regions

observed data

The different formats / implementations produce the same results (of course)

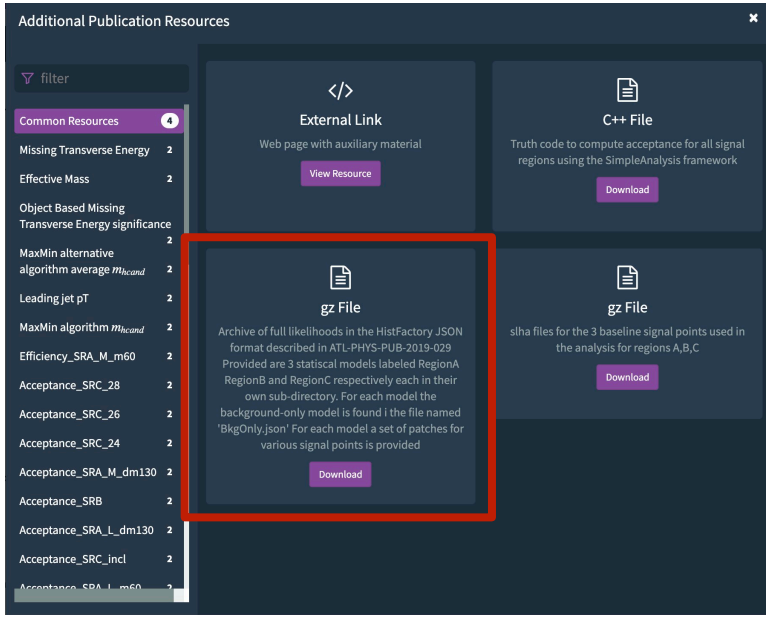


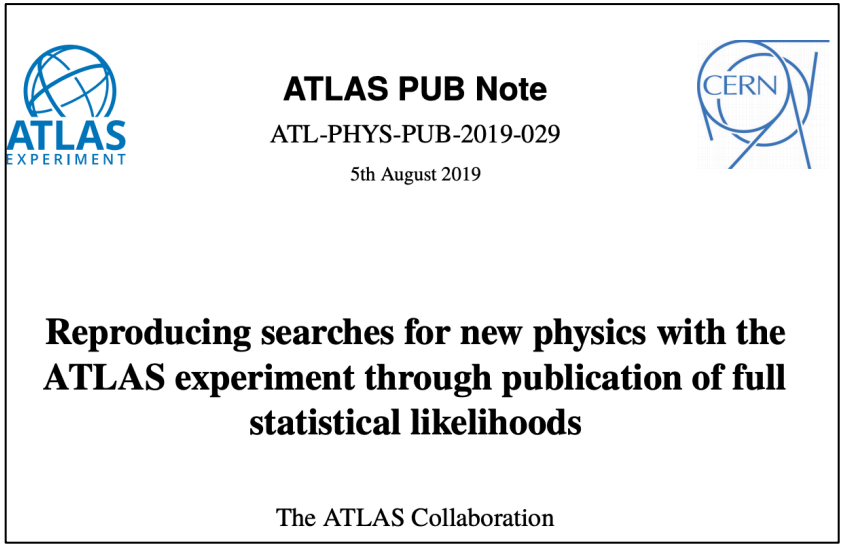


Advantages of JSON format

- human, machine-readable
- software-independent, good for long-term archiving, sharing.

This format is used to make statistical models publicly available




ATLAS PUB Note

ATL-PHYS-PUB-2019-029

5th August 2019

Reproducing searches for new physics with the ATLAS experiment through publication of full statistical likelihoods

The ATLAS Collaboration

What to do with a pyhf JSON:

Full interface is still developing but setting limits is one of the most common tasks for searches, so that's easy in pyhf.

```
$> pip install pyhf
$> pyhf cls workspace.json
{
    "CLs_exp": [
        0.008897411763217407,
        0.03524468002619176,
        0.1243148689002353,
        0.3514186235832989,
        0.6941411699405086
    ],
    "CLs_obs": 0.03607409335946063
}
```

Try it yourself:

launch binder

pyhf uses the wider scientific python ecosystem

NumPy

SciPy

also optionally uses numeric libraries from the Machine Learning World to get e.g. faster fits using GPUs, more precise minimization

TensorFlow

PYTORCH

The basic object of HistFactory is the statistical model (or pdf)

$$\mathcal{P}(n_c, x_e, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_{c \in \text{channels}} \left[\text{Pois}(n_c|\nu_c) \prod_{e=1}^{n_c} f_c(x_e|\boldsymbol{\alpha}) \right] \cdot G(L_0|\lambda, \Delta_L) \cdot \prod_{p \in \mathbb{S}+\boldsymbol{\Gamma}} f_p(a_p|\alpha_p)$$

Math

$$\text{logLikelihood} = \ln L(\theta|x)$$

Code

```
model.logpdf(parameters, data)
```

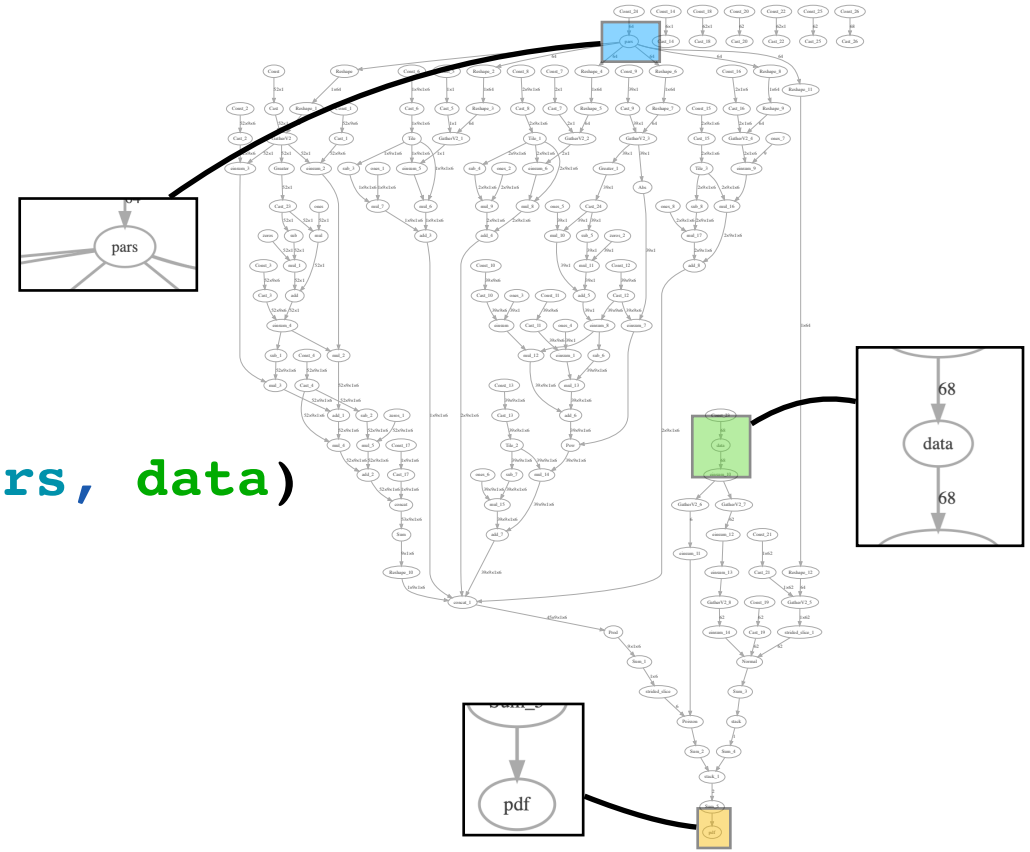
you can create such a model for simple cases right from python

```
[28]: import  pyhf

model = pyhf.simplemodels.hepdata_like([5,10],[50,30],[2,3])
data = [50,30] + model.config.auxdata
parameters = [1.0,1.0,1.0] # nominal parameters

model.logpdf(parameters,data) # evaluate log likelihood

[28]: array([-14.46326506])
```

Internally, the pdf is represented as a computation with a bunch of NumPy arrays.

- the very bottom node is the final log likelihood value

```
value = model.logpdf(pars, data)
```



You only rarely evaluating the likelihood by hand

Common Task: Maximum Likelihood Fitting!

Minimize Objective Function

$$\log \lambda(\theta) = -2 \ln L(\theta|x)$$

(maximizes likelihood)

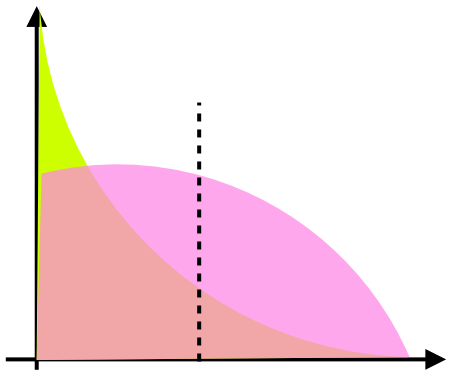
$$\hat{\theta} = \text{argmax } L(\theta|d)$$

```
[41]: import  pyhf

      model = pyhf.simplemodels.hepdata_like([5,10],[50,30],[2,3])
      data = [55,33] + model.config.auxdata
      parameters = [1.0,1.0,1.0] # nominal parameters

      model.logpdf(parameters,data) # evaluate log likelihood

[41]: array([-13.60586994])

[42]: bestfit_pars = pyhf.optimizer.unconstrained_bestfit(
          pyhf.utils.loglambdav, # -2*logL
          data,
          model,
          model.config.suggested_init(),
          model.config.suggested_bounds(),
      )
      bestfit_pars

[42]: array([0.41202114, 1.00420257, 0.99222999])

[43]:  #we've maximized the loglikelihood (or minimized the neg. log likelihood)
      model.logpdf(bestfit_pars,data)

[43]: array([-13.04446523])
```

Often you don't even do that.
you just want to compute e.g. a CLs value using the profile likelihood

$$\ln \lambda(\mu) = -2 \frac{L(\mu, \hat{\hat{\theta}})}{L(\hat{\mu}, \hat{\theta})}$$

In pyhf that's also easy:



$$\mathrm{CL}_s = \frac{\mathrm{CL}_{s+b}}{\mathrm{CL}_b}$$

```
[48]: cls = pyhf.utils.hypotest(1.0,data,model)
      cls

[48]: array([0.22800433])
```

You can also get the expected CLs values easily

In [4]:
```
cls = pyhf.utils.hypotest(1.0,data,model)
cls
```

Out[4]:
```
array([0.22800433])
```

In [5]:
```
cls_obs, cls_exp = pyhf.utils.hypotest(1.0,data,model, return_expected = True)
cls_obs, cls_exp
```

Out[5]:
```
(array([0.22800433]), array([0.10248917]))
```

In [17]:
```
import numpy as np
results = []
poivals = np.linspace(0,5)
for mu in poivals:
    cls_obs, cls_exp = pyhf.utils.hypotest(mu,data,model, return_expected_set = True)
    results.append([ cls_obs[0] ] + [x[0] for x in cls_exp])

results = np.asarray(results)
```

[42]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(poivals,results[:,0], c = 'black')
plt.plot(poivals,results[:,[1,-1]], linestyle = 'dotted', c = 'black')
plt.plot(poivals,results[:,[2,-2]], linestyle = 'dotted', c = 'black')
plt.fill_between(poivals,results[:,1],results[:,-1], color = 'yellow')
```

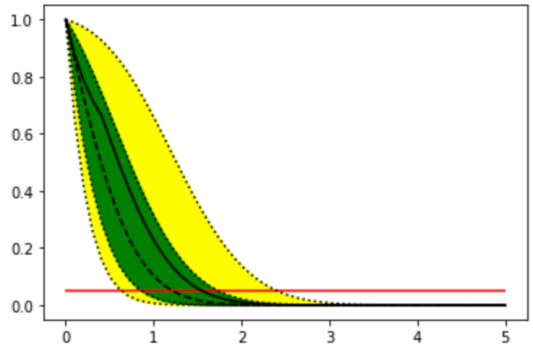
Sometimes you want to do a scan over the parameter of interest: e.g. to get an upper limit

[83]:
```
import numpy as np
results = []
poivals = np.linspace(0,5)
for mu in poivals:
    cls_obs, cls_exp = pyhf.utils.hypotest(mu,data,model, return_expected_set = True)
    results.append([ cls_obs[0] ] + [x[0] for x in cls_exp])

results = np.asarray(results)
print('Upper Limit (obs): μ = {:0.3}'.format(np.interp(0.05,results[:,0][::-1],poivals[::-1])))
print('Upper Limit (exp): μ = {:0.3}'.format(np.interp(0.05,results[:,3][::-1],poivals[::-1])))
```

```
Upper Limit (obs): μ = 1.55
Upper Limit (exp): μ = 1.22
```

[84]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(poivals,results[:,0], c = 'black')
plt.plot(poivals,results[:,[1,-1]], linestyle = 'dotted', c = 'black')
plt.plot(poivals,results[:,[2,-2]], linestyle = 'dotted', c = 'black')
plt.plot(poivals,results[:,3], linestyle = 'dashed', c = 'black')
plt.fill_between(poivals,results[:,1],results[:,-1], color = 'yellow')
plt.fill_between(poivals,results[:,2],results[:,-2], color = 'green')
plt.hlines(0.05,0,5, color = 'r')
```

[84]:
```
<matplotlib.collections.LineCollection at 0x7f03117c1ac8>
```



Hopefully this looks nice / appealing.
Goal was to make the user interface natural, easy to use.

How do you get started beyond the simple models?

If you have a ROOT HistFactory Workspace you can just export the XML again:

```
jovyan@jupyter-lukasheinrich-2datlaspyhftutorial-2dg4tpwbwn:~$ root -b root/example_UsingPy_combined_meas_model.root
   ------------------------------------------------------------
  | Welcome to ROOT 6.18/04                    https://root.cern |
  |                                (c) 1995-2019, The ROOT Team |
  | Built for linuxx8664gcc on Sep 28 2019, 10:56:00            |
  | From tag , 11 September 2019                                |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
   ------------------------------------------------------------

root [0]
Attaching file root/example_UsingPy_combined_meas_model.root as _file0...

RooFit v3.60 -- Developed by Wouter Verkerke and David Kirkby
                Copyright (C) 2000-2013 NIKHEF, University of California & Stanford University
                All rights reserved, please read http://roofit.sourceforge.net/license.txt

(TFile *) 0x560251657500
root [1] .ls
TFile**         root/example_UsingPy_combined_meas_model.root
 TFile*         root/example_UsingPy_combined_meas_model.root
  KEY: RooWorkspace     combined;1      combined
  KEY: TProcessID       ProcessID0;1    7a7a3df6-f70d-11e9-910f-77280c0abeef
  KEY: TDirectoryFile   channel1_hists;1        channel1_hists
  KEY: RooStats::HistFactory::Measurement       meas;1  meas
root [2] meas->PrintXML()
Printing XML Files for measurement: meas
Printing XML Files for channel: channel1
Finished printing XML files
Finished printing XML files
root [3] .q
jovyan@jupyter-lukasheinrich-2datlaspyhftutorial-2dg4tpwbwn:~$ pyhf xml2json meas.xml
```

Hopefully this looks nice / appealing.
Goal was to make the user interface natural, easy to use.

How do you get started beyond the simple models?

If you have the XML + ROOT files for your HistFactory analysis you can just use an automated script to convert to JSON

```
jovyan@jupyter-lukasheinrich-2datlaspyhftutorial-2dg4tpwbwn:~$ pyhf xml2json meas.xml > workspace.json
  - modifier SigXsecOverSM(NormFactor): : 3modifier [00:00, 5724.71modifier/s]
  - modifier syst2(OverallSys): : 2modifier [00:00, 13819.78modifier/s]
  - modifier syst3(OverallSys): : 3modifier [00:00, 13203.48modifier/s]
  - sample background2: 100%|██████████████████████████████████████████████| 3/3 [00:00<00:00, 26.93sample/s]
Processing ./meas_channel1.xml: 100%|═══════════════════════════════════════| 1/1 [00:00<00:00,  8.89channel/s]
jovyan@jupyter-lukasheinrich-2datlaspyhftutorial-2dg4tpwbwn:~$ pyhf cls workspace.json
{
    "CLs_exp": [
        0.08709494712648695,
        0.18895596600071107,
        0.3782612958476278,
        0.6505311336012763,
        0.8886239643568519
    ],
    "CLs_obs": 0.3782590536787565
}
```