pyhf & cabinetry

Alexander Held[1]

[1] New York University

IRIS-HEP AGC Tools 2021 Workshop
https://indico.cern.ch/event/1076231/
Nov 3, 2021

This work was supported by the U.S. National Science Foundation (NSF) Cooperative Agreement OAC-1836650 (IRIS-HEP).

The HistFactory model

- **Binned template fits** are widely used for **statistical inference** at the LHC and beyond

- **HistFactory** is a statistical model for **binned template fits**
  - prescription for constructing probability density functions (pdfs) from small set of building blocks
  - covers wide range of use cases, extensively used in ATLAS
  - models can be serialized to *workspaces*

observed data

unconstrained parameters, e.g. POI

prediction (summed over samples)

constraint term (e.g. Gaussian)

$$p(\vec{n}, \vec{a} \mid \vec{k}, \vec{\theta}) = \prod_i \mathrm{Pois}(n_i \mid \nu_i(\vec{k}, \vec{\theta})) \cdot \prod_j c_j(a_j \mid \theta_j)$$

auxiliary data, e.g. from calibration measurement

constrained nuisance parameters

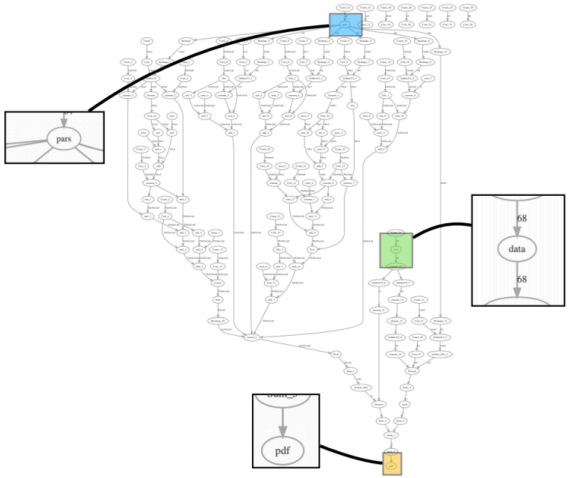
product over all bins in all channels

pyhf implements HistFactory (1)

- **Until 2018**, the **HistFactory** model had only been implemented in **ROOT**

- **pyhf** implements the **HistFactory** model in **pure Python**
  - leverages tensor backends: efficient vectorized calculations & hardware acceleration
    - can automatically differentiate through statistical model (computational graph)
      - exact gradients for minimizers
      - enables end-to-end analysis optimization: **neos**
    - backend-agnostic API (and CLI)

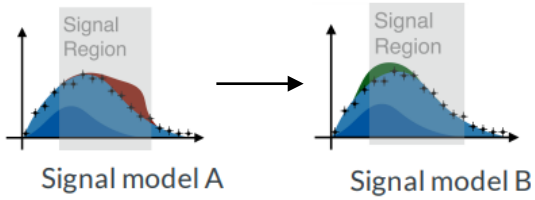
computational graph for HistFactory



pyhf implements HistFactory (2)

- **Until 2018**, the **HistFactory** model had only been implemented in **ROOT**

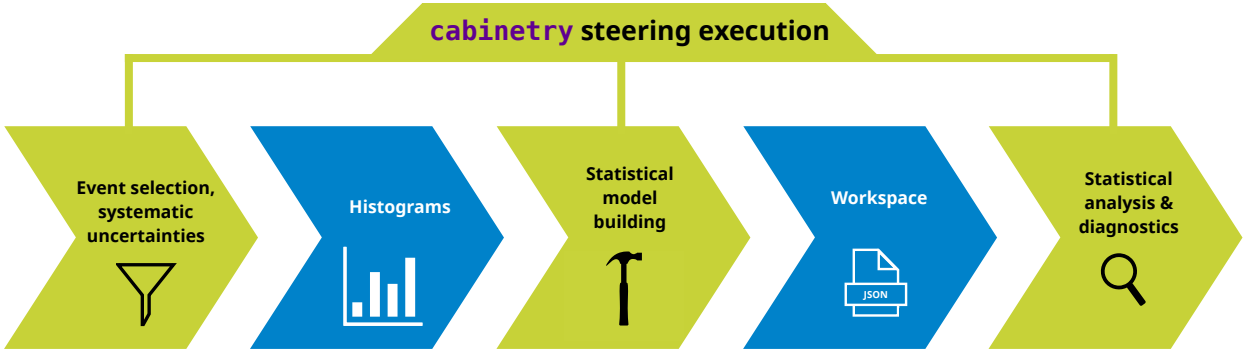
- **pyhf** implements the **HistFactory** model in **pure Python**
  - workspaces (statistical models) serialized to JSON → used by ATLAS to publish on HEPData (list of public models)
    - JSON Patch to swap out workspace components (e.g. signal model)
    - versioned JSON schema describes the declarative model




```
# Using CLI
$ pyhf cls example.json | jq .CLs_obs
0.053994246621274014

$ cat new_signal.json
[{
    "op": "replace",
    "path": "/channels/0/samples/0/data",
    "value": [10.0, 6.0]
}]

$ pyhf cls example.json --patch new_signal.json | jq .CLs_obs
0.3536906623262466
```

Working with `cabinetry`

- **`cabinetry`** is a **Python library** for creating and operating HistFactory models
  - **design** and **construct statistical models** (workspaces) from instructions in **declarative configuration**
    - analyzers specify selections for signal/control regions, (Monte Carlo) samples, systematic uncertainties
    - **`cabinetry`** steers creation or collects provided template histograms (region ⊗ sample ⊗ systematic)
    - **`cabinetry`** produces **`HistFactory`** workspaces (serialized fit model)
  - perform **statistical inference**
    - including diagnostics and visualization tools to study and disseminate results



Designing a statistical model

- **Declarative configuration** (JSON/YAML/dictionary) specifies everything needed to build a workspace
  - can concisely capture complex region ⊗ sample ⊗ systematic structure

general settings

list of phase space regions (channels)

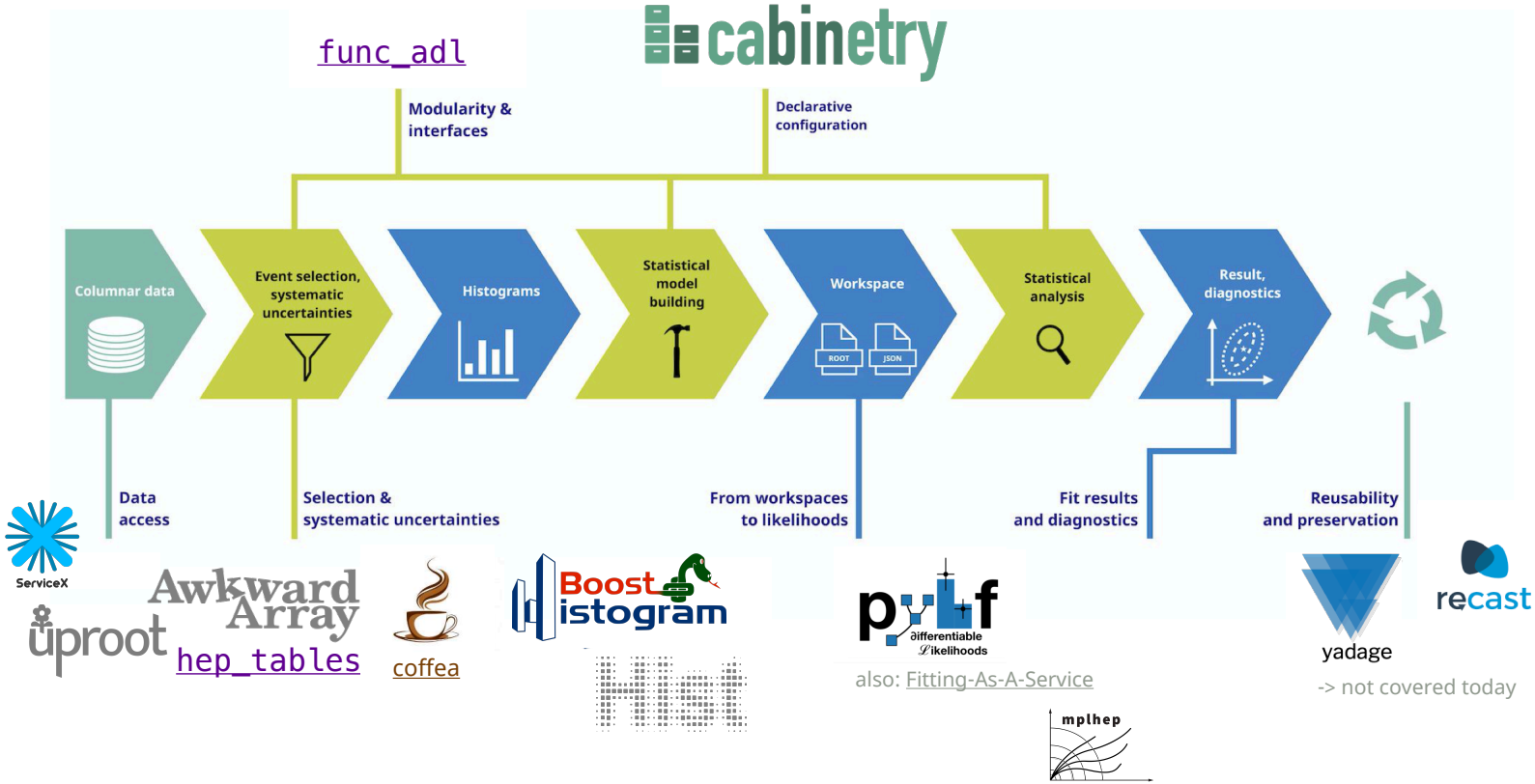
list of samples (MC/data)

```yaml
General:
  Measurement: "Example"
  InputPath: "input/{SamplePaths}"
  HistogramFolder: "histograms/"
  POI: "Signal_norm"

Regions:
  - Name: "Signal_region"
    Filter: "nJets >= 8"
    Variable: "jet_pt"
    Binning: [200, 300, 400, 500]

Samples:
  - Name: "Data"
    SamplePaths: "data.root"
    Tree: "events"
    Data: True

  - Name: "Signal"
    SamplePaths: "signal.root"
    Tree: "events"
    Weight: "weight_nominal"

  - Name: "Background"
    SamplePaths: "background.root"
    Tree: "events"
    Weight: "weight_nominal"

Systematics:
  - Name: "Luminosity"
    Up:
      Normalization: 0.05
    Down:
      Normalization: -0.05
    Samples: ["Signal", "Background"]
    Type: "Normalization"

  - Name: "ModelingVariation"
    Up:
      Tree: "events_up"
      Weight: "weight_modeling"
    Down:
      Tree: "events_down"
      Weight: "weight_modeling"
    Smoothing:
      Algorithm: "353QH, twice"
    Samples: "Background"
    Type: "NormPlusShape"

NormFactors:
  - Name: "Signal_norm"
    Samples: "Signal"
    Nominal: 1
    Bounds: [0, 10]
```

list of systematic uncertainties

list of normalization factors

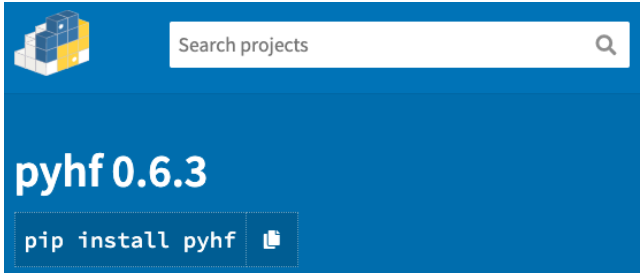
pyhf and cabinetry within the broader ecosystem

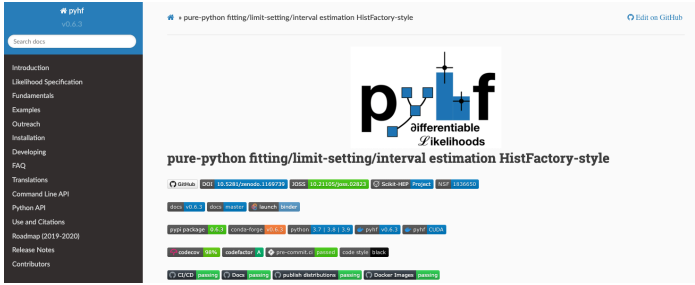


Backup

Links to pyhf

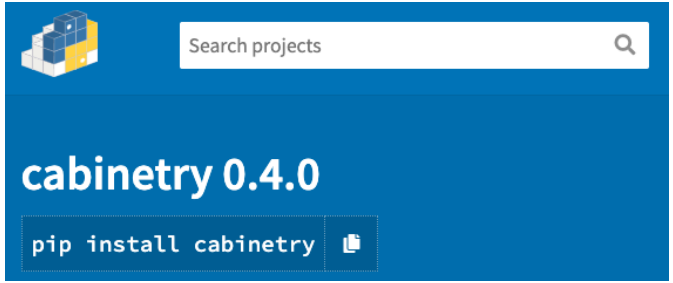
- **`pyhf`:**
  - can be installed via `$ pip install pyhf`
    - `pyhf[backends]` for all tensor backends
  - is open source and
    - developed on GitHub: scikit-hep/pyhf
    - published on PyPI
    - documented on Read the Docs
      - contains links to talks / paper as well
    - part of IRIS-HEP
    - provides tutorials: pyhf.github.io/pyhf-tutorial





Links to `cabinetry`

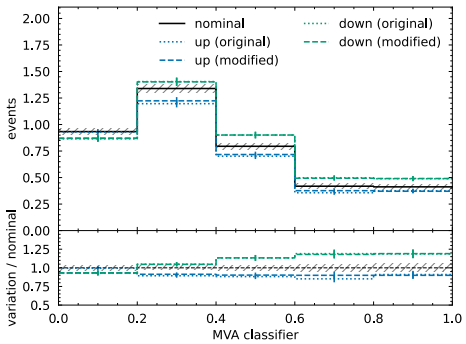
- **`cabinetry`:**
  - can be installed via `$ pip install cabinetry`
    - `cabinetry[contrib]` for extra features
  - is open source and
    - developed on GitHub: scikit-hep/cabinetry
    - published on PyPI
    - documented on Read the Docs
      - contains links to talks / paper as well
    - part of IRIS-HEP
    - provides tutorials: cabinetry/cabinetry-tutorials



Template histograms and workspace building

- **Workspaces construction** happens in three steps:
  1) create template histograms from columnar data following config instructions
     - backends execute instructions (default: **uproot**, experimental: **coffea**)
     - alternatively: collect existing user-provided histograms
  2) optional: apply post-processing to templates (e.g. smoothing)
  3) assemble templates into workspace (JSON file)

visualization of individual template histograms

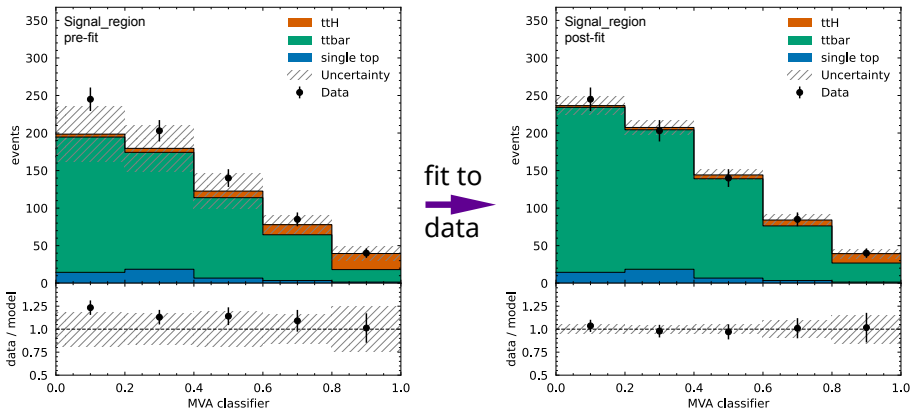


- Utilities provided to **visualize and debug** fit model

event yield table

sample	Control region	Signal region
single top	44.74	0.35
ttbar	635.98	13.28
z_ttH	30.90	1.80
total	711.61 ± 28.28	15.43 ± 2.69
data	713.00	14.00

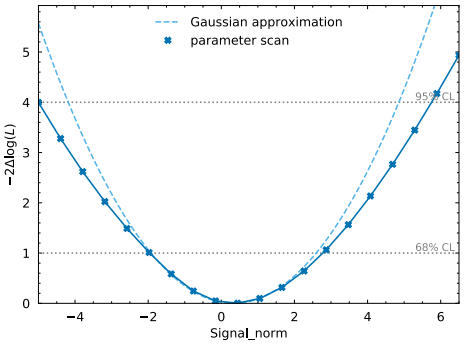
fit model visualization



Statistical inference with `cabinetry`

- Implementations for all **common inference tasks** exist
  - includes associated visualizations

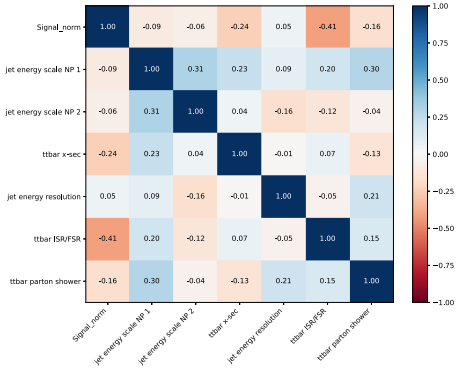
likelihood scans



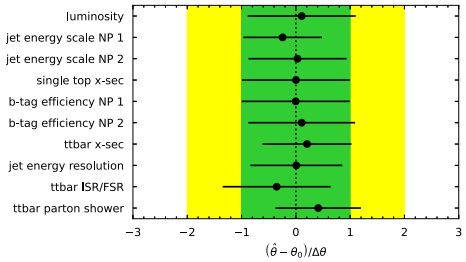
discovery significance

```
$ cabinetry significance workspaces/example_workspace.json
INFO - cabinetry.fit - calculating discovery significance
INFO - cabinetry.fit - observed p-value: 1.13053295%
INFO - cabinetry.fit - observed significance: 2.280
INFO - cabinetry.fit - expected p-value: 0.42110716%
INFO - cabinetry.fit - expected significance: 2.635
```

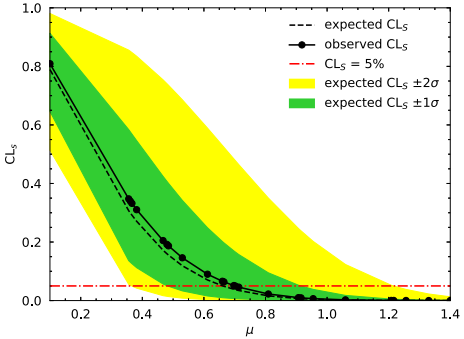
parameter correlations



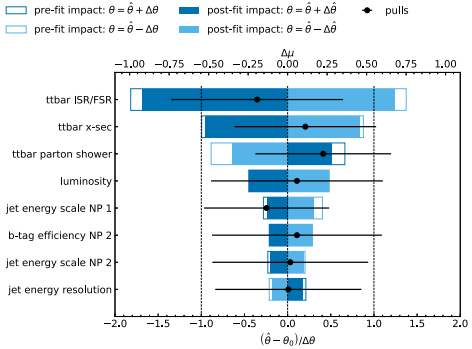
nuisance parameter pulls



upper parameter limits



nuisance parameter impacts



Statistical analysis: the `HistFactory` model

- **`HistFactory`** is the standard model used in ATLAS for **binned statistical analysis**
  - **`pyhf`** is a python implementation of this model
  - the **`HistFactory`** model specifies how to construct the likelihood function
  - **`cabinetry`** turns a declarative specification about cuts, systematics etc. into a statistical model
  - **`pyhf`** turns that model into a likelihood function

unconstrained parameters, e.g. POI

observed data

prediction (summed over samples)

constraint term (e.g. Gaussian)

$$p(\vec{n}, \vec{a} \mid \vec{k}, \vec{\theta}) = \prod_i \mathrm{Pois}(n_i \mid \nu_i(\vec{k}, \vec{\theta})) \cdot \prod_j c_j(a_j \mid \theta_j)$$

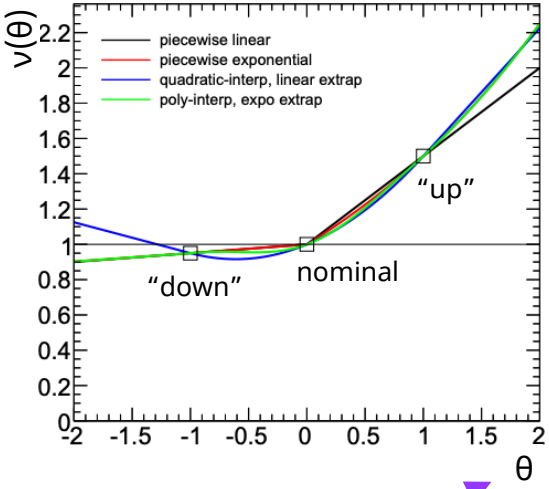
auxiliary data, e.g. from calibration measurement

constrained nuisance parameters

product over all bins in all channels

Systematic uncertainties with `HistFactory`

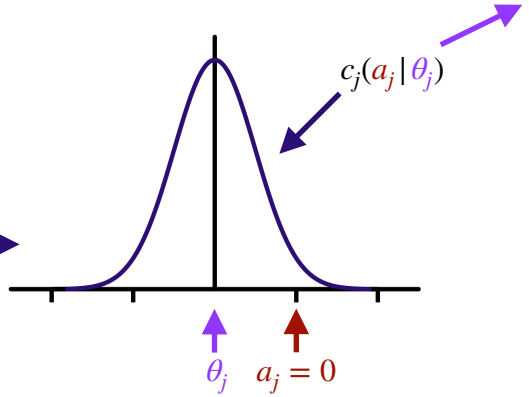
- common **systematic uncertainties** specified with **two template histograms**
  - "up variation": model prediction for $\theta = +1$
  - "down variation": model prediction for $\theta = -1$
  - interpolation & extrapolation provides model predictions ν for *any* $\vec{\theta}$



CERN-OPEN-2012-016

- **Gaussian constraint terms** used to model auxiliary measurements (in most cases)
  - centered around nuisance parameter (NP)
  - normalized width ($\sigma = 1$) and mean (auxiliary data $a_j = 0$)
  - penalty for pulling NP away from best-fit auxiliary measurement value

$$p(\vec{n}, \vec{a} \mid \vec{k}, \vec{\theta}) = \prod_i \mathrm{Pois}(n_i \mid \nu_i(\vec{k}, \vec{\theta})) \cdot \prod_j c_j(a_j \mid \theta_j)$$

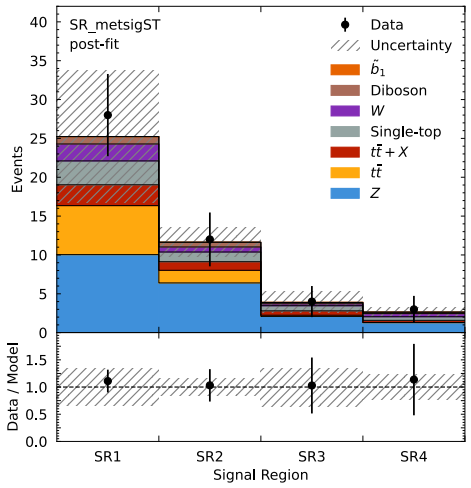


Working with an unknown workspace

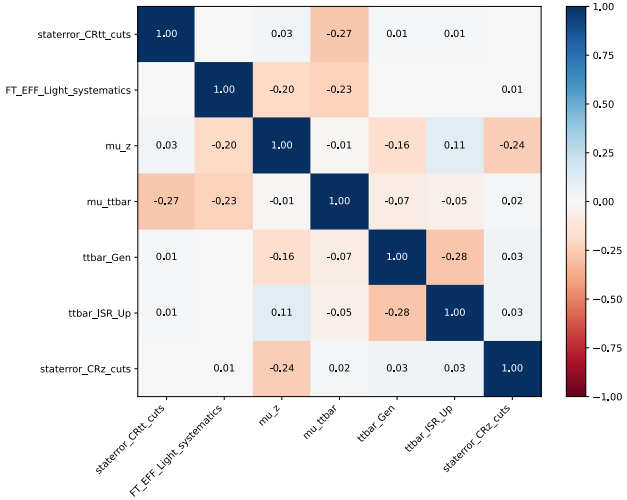
- Pick a **workspace** from **HEPData**: 10.17182/hepdata.89408.v3 (analysis: JHEP 12 (2019) 060)
  - download workspace with **`pyhf`**
  - perform inference and visualize results with **`cabinetry`**
- See arXiv:2109.04981 and try it on Binder!

Search for bottom-squark pair production with the ATLAS detector in final states containing Higgs bosons, *b*-jets and missing transverse momentum

```python
import json
import cabinetry
import pyhf
from cabinetry.model_utils import prediction
from pyhf.contrib.utils import download

# download the ATLAS bottom-squarks analysis probability models from HEPData
download("https://www.hepdata.net/record/resource/1935437?view=true", "bottom-squarks")

# construct a workspace from a background-only model and a signal hypothesis
bkg_only_workspace = pyhf.Workspace(json.load(open("bottom-squarks/RegionC/BkgOnly.json")))
patchset = pyhf.PatchSet(json.load(open("bottom-squarks/RegionC/patchset.json")))
workspace = patchset.apply(bkg_only_workspace, "sbottom_600_280_150")

# construct the probability model and observations
model, data = cabinetry.model_utils.model_and_data(workspace)

# produce visualizations of the pre-fit model and observed data
prefit_model = prediction(model)
cabinetry.visualize.data_mc(prefit_model, data)

# fit the model to the observed data
fit_results = cabinetry.fit.fit(model, data)

# produce visualizations of the post-fit model and observed data
postfit_model = prediction(model, fit_results=fit_results)
cabinetry.visualize.data_mc(postfit_model, data)
```

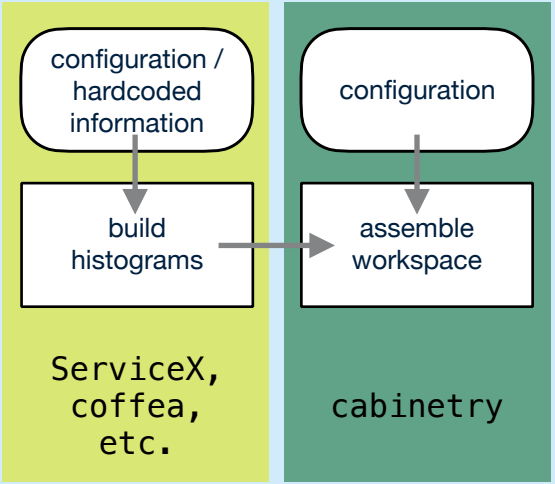


(workspace contains additional channels not shown here)



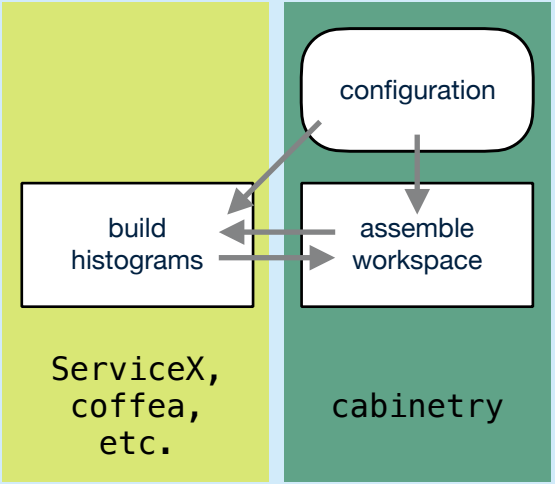
Control flow & duplication of information

demonstrated today



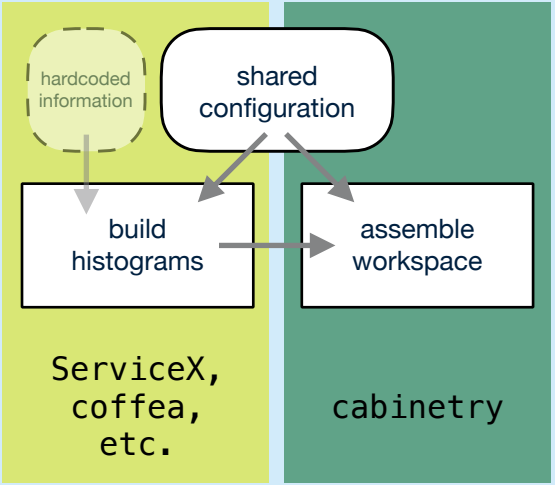
1) user builds histograms
2) `cabinetry` assembles workspace

execution steered by `cabinetry`



1) `cabinetry` determines & builds required histograms
2) `cabinetry` assembles workspace

the future (?)



1) user builds histograms using info in shared config
2) `cabinetry` assembles workspace