# cabinetry

cabinetry developers

Apr 01, 2023

# CONTENTS


1 Core concepts 3

2 Advanced concepts 7

3 Configuration schema 13

4 CLI 21

5 API 29

6 License 63

7 Presentations of cabinetry 65

8 Acknowledgements 67

Python Module Index 69

Index 71


cabinetry

CHAPTER

ONE

# CORE CONCEPTS

## 1.1 Inputs to cabinetry: ntuples or histograms

`cabinetry` supports two types of input files when building a workspace: ntuples containing columnar data and histograms. When using ntuple inputs, `cabinetry` needs to know not only where to find the input files for every template histogram it needs to build, but also what selections to apply, which column to extract and how to weight every event. The configuration schema lists the required options, see *Configuration schema* for more information. Less information is required when using histogram inputs: only the path to each histogram needs to be specified in this case.

### 1.1.1 Input file path specification for ntuples

Paths to ntuple input files for histogram production are specified with the mandatory `InputPath` setting in the `General` config section. If everything is in one file, the value should be the path to this file. It is common to have multiple input files, split across phase space regions or samples. For this purpose, the `InputPath` value can take two placeholders: `{RegionPath}` and `{SamplePath}`.

#### RegionPath

When building histograms for a specific region, the `{RegionPath}` placeholder takes the value specified in the `RegionPath` setting of the corresponding region. The value of `RegionPath` has to be a string.

#### SamplePath

The `{SamplePath}` placeholder takes the value given by `SamplePath` of the sample currently processed. This value can either be a string or a list of strings. If it is a list, multiple copies of `InputPath` are created, and in each of them the `{SamplePath}` placeholder takes the value of a different entry in the list. All input files are processed, and their contributions are summed together. The histogram created by `SamplePath: ["a.root", "b.root"]` is equivalent to the histogram created with `SamplePath: "a_plus_b.root"`, where `a_plus_b.root` is produced by merging both files.

### Systematics

It is possible to specify overrides for the `RegionPath` and `SamplePath` values in systematic templates. If those settings are specified in the `Up` or `Down` template section of a systematic uncertainty, then the corresponding values are used when building the path to the file used to construct the histogram for this specific template.

### An example

The following configuration file excerpt shows an example of specifying paths to input files.

```
General:
  InputPath: "inputs/{RegionPath}/{SamplePath}"

Regions:
  - Name: "Signal_region"
    RegionPath: "signal_region"

  - Name: "Control_region"
    RegionPath: "control_region"

Samples:
  - Name: "Data"
    SamplePath: "data.root"

  - Name: "Signal"
    SamplePath: ["signal_1.root", "signal_2.root"]

Systematics:
  - Name: "Signal_modeling"
    Up:
      SamplePath: "modeling_variation_up.root"
    Down:
      SamplePath: "modeling_variation_down.root"
    Samples: "Signal"
```

The following files will be read to create histograms:

- for *Signal_region*:
  - *Data*: `inputs/signal_region/data.root`
  - *Signal*: `inputs/signal_region/signal_1.root`, `inputs/signal_region/signal_2.root`
    - systematic uncertainty:
      - *up*: `inputs/signal_region/modeling_variation_up.root`
      - *down*: `inputs/signal_region/modeling_variation_down.root`
- for *Control_region*:
  - *Data*: `inputs/control_region/data.root`
  - *Signal*: `inputs/control_region/signal_1.root`, `inputs/control_region/signal_2.root`
    - systematic uncertainty:
      - *up*: `inputs/control_region/modeling_variation_up.root`

- *down*: `inputs/control_region/modeling_variation_down.root`

### 1.1.2 Input file path specification for histograms

The specification of paths to histograms works very similarly to the ntuple case. The `InputPath` setting in the `General` config section is still mandatory. It can again take placeholders: `{RegionPath}`, `{SamplePath}`, and `{VariationPath}`. The `VariationPath` setting will default to an empty string if not specified, but it can be set to another value (such as `"nominal"`) in the `General` block.

A major difference to the ntuple path construction is that the histogram path needs to not only include the path to the file containing a given histogram, but also to the histogram within the file. This is achieved by using a colon `:` to distinguish between both parts of the path: `folder/file.root:abc/h1` points to a histogram called `h1` located in a folder called `abc` which itself exists within a file called `file.root` which can be found in a folder called `folder`.

When using histogram inputs, use `cabinetry.templates.collect` instead of `cabinetry.templates.build` (which is used for ntuple inputs).

#### RegionPath

This works in the same way as it does for ntuples: the `RegionPath` setting in each region sets the value for the `{RegionPath}` placeholder. Note that the value cannot be overridden on a per-systematic basis in the histogram case.

#### SamplePath

The `SamplePath` setting sets the value for the `{SamplePath}` placeholder. In contrast to the ntuple case, this value cannot be a list of strings. It also cannot be overridden on a per-systematic basis, just like `RegionPath`.

#### VariationPath

Each systematic template can set the value for the `{VariationPath}` placeholder via the `VariationPath` setting. `RegionPath` and `SamplePath` settings cannot be overridden.

#### An example

The following shows an example, similar to the ntuple example.

```
General:
  InputPath: "inputs/{RegionPath}.root:{SamplePath}_{VariationPath}"
  VariationPath: "nominal"

Regions:
  - Name: "Signal_region"
    RegionPath: "signal_region"

  - Name: "Control_region"
    RegionPath: "control_region"

Samples:
  - Name: "Data"
    SamplePath: "data"
```

(continues on next page)

(continued from previous page)

```
  - Name: "Signal"
    SamplePath: "signal"

Systematics:
  - Name: "Signal_modeling"
    Up:
      VariationPath: "modeling_variation_up"
    Down:
      VariationPath: "modeling_variation_down"
    Samples: "Signal"
```

The following histograms will be read:

- for *Signal_region*:
  - *Data*: `inputs/signal_region.root:data_nominal`
  - *Signal*: `inputs/signal_region.root:signal_nominal`
    - systematic uncertainty:
      - *up*: `inputs/signal_region.root:signal_modeling_variation_up`
      - *down*: `inputs/signal_region:signal_modeling_variation_down`
- for *Control_region*:
  - *Data*: `inputs/control_region.root:data_nominal`
  - *Signal*: `inputs/control_region.root:signal_nominal`
    - systematic uncertainty:
      - *up*: `inputs/control_region.root:signal_modeling_variation_up`
      - *down*: `inputs/control_region:signal_modeling_variation_down`

CHAPTER

TWO

# ADVANCED CONCEPTS

## 2.1 Accessing vector branches

The transverse momentum of the first jet in a vector branch `jet_pT` is obtained via `jet_pT[0]` in `ROOT`. The `uproot` backend for ntuple reading treats expressions (such as what is written in `Filter` and `Weight` configuration file options) as Python code. The correct way to access the same information through `cabinetry` is `jet_pT[:,0]`, where the first index runs over events.

## 2.2 Overrides for template building

### 2.2.1 Introduction

It is possible to define functions that are called when `cabinetry` tries to construct a template histogram. Such functions need to accept four arguments in the following order:

- a dictionary with information about the region being processed,
- a dictionary with information about the sample being processed,
- a dictionary with information about the systematic being processed,
- the template being considered: a string `"Up"` / `"Down"` for variations, or `None` for the nominal template.

The function needs to return a boost-histogram Histogram. This histogram is then further processed in `cabinetry`.

### 2.2.2 Example

The example below defines a function `build_data_hist`. The decorator specifies that this function should be applied to all histograms for samples with name `ttbar`. It is also possible to specify `region_name`, `systematic_name` and `template` for the names of the region, systematic and template. Not specifying these options means not restricting the applicability of the function. When no user-defined function matches a given histogram that has to be produced, `cabinetry` falls back to use the default histogram creation methods.

```
from typing import Optional

import boost_histogram as bh
import numpy as np
import cabinetry

my_router = cabinetry.route.Router()
```

(continues on next page)

(continued from previous page)

```
# define a custom template builder function that is executed for data samples
@my_router.register_template_builder(sample_name="ttbar")
def build_data_hist(
    region: dict, sample: dict, systematic: dict, template: Optional[str]
) -> bh.Histogram:
    hist = bh.Histogram(
        bh.axis.Variable(region["Binning"], underflow=False, overflow=False),
        storage=bh.storage.Weight(),
    )
    yields = np.asarray([17, 12, 25, 20])
    variance = np.asarray([1.5, 1.2, 1.8, 1.6])
    hist[...] = np.stack([yields, variance], axis=-1)
    return hist


cabinetry.templates.build(
    cabinetry_config, method="uproot", router=my_router
)
```

The instance of `cabinetry.route.Router` is handed to `cabinetry.templates.build` to enable the use of `build_data_hist`.

The function `build_data_hist` in this example always returns the same histogram. Given that the dictionaries in the function signature provide additional information, it is for example possible to return different yields per region:

```
if region["Name"] == "Signal_region":
    yields = np.asarray([17, 12, 25, 20])
elif region["Name"] == "Background_region":
    yields = np.asarray([102, 121, 138, 154])
```

### 2.2.3 Wildcards and multiple requirements

It is also possible to use wildcards to specify which templates a function should be applied to. The implementation currently makes use of fnmatch. The following decorator

```
@my_router.register_template_builder(sample_name="ttbar_*")
```

means that the function will for example be applied if the sample name is *ttbar_ljets* or *ttbar_dil*, but not if it is *single_top*. All conditions need to be fulfilled to apply a user-defined function, so

```
@my_router.register_template_builder(
    region_name="signal_region",
    sample_name="signal",
    systematic="alpha_S",
    template="*",
)
```

means that for the decorated function to be executed, the region name needs to be *signal_region*, the sample needs to be called *signal*, the systematic needs to be *alpha_S*, but there is no restriction to the template name.

Since `template` can be a string or `None`, its behavior is slightly different:

- `template="*"` is the default, and means that any histogram matches (nominal, as well as variations),
- `template=None` matches only nominal histograms,
- `template=string`, where `string` is any string other than `"*"`, can never match the nominal template, but could match the systematic variations called `"Up"` and `"Down"`.

## 2.3 Fixed parameters

The `cabinetry` configuration file contains the `Fixed` option (in the `General` group of options), which allows for the creation of a workspace with parameters set to be constant.

```
Fixed:
  - Name: par_a
    Value: 2
  - Name: par_b
    Value: 1
```

The same can be written in a more compact way:

```
Fixed: [{"Name": "par_a", "Value": 2},{"Name": "par_b", "Value": 1}]
```

The associated `pyhf` workspace will contain the following:

```
{
  "measurements": [
    {
      "config": {
        "parameters": [
          {"fixed": true, "inits": [2], "name": "par_a"},
          {"fixed": true, "inits": [1], "name": "par_b"}
        ]
      }
    }
  ]
}
```

Fixed parameters are not allowed to vary in fits. Both their pre-fit and post-fit uncertainty are set to zero. This means that the associated nuisance parameters do not contribute to uncertainty bands in data/MC visualizations either. The impact of such parameters on the parameter of interest (for nuisance parameter ranking) is also zero.

## 2.4 Manually correlating systematics

Systematic uncertainties are correlated if the modifiers defining them in the `pyhf` workspace have the same names. The example below shows a modifier called *correlated_modifier*, correlated between two samples in a workspace.

```
[
  {
    "data": [25.0],
    "modifiers": [
      {
```

(continues on next page)

(continued from previous page)

```
        "data": {"hi": 1.05, "lo": 0.95},
        "name": "correlated_modifier",
        "type": "normsys"
      }
    ],
    "name": "Signal"
  },
  {
    "data": [55.0],
    "modifiers": [
      {
        "data": {"hi": 1.05, "lo": 0.95},
        "name": "correlated_modifier",
        "type": "normsys"
      }
    ],
    "name": "Background"
  }
]
```

The names of modifiers written to the workspace are by default picked up from the name of the associated systematic in the `cabinetry` configuration. Names of systematics in the configuration need to be unique, so it is not possible to define multiple systematics with the same name. Instead, the option `ModifierName` can be used to specify the name of the associated modifier(s) used in the workspace:

```
Systematics:
  - Name: "first_systematic"
    Up:
      Normalization: 0.05
    Down:
      Normalization: -0.05
    Type: "Normalization"
    Samples: "Signal"
    ModifierName: "correlated_modifier"

  - Name: "second_systematic"
    Up:
      Normalization: 0.05
    Down:
      Normalization: -0.05
    Type: "Normalization"
    Samples: "Background"
    ModifierName: "correlated_modifier"
```

This results in a workspace like the example shown above. Without `ModifierName`, the two modifiers would be uncorrelated and called *first_systematic* and *second_systematic*.

In this simple example, the following settings result in the same workspace:

```
Systematics:
  - Name: "correlated_modifier"
    Up:
      Normalization: 0.05
```

(continues on next page)

(continued from previous page)

```
  Down:
    Normalization: -0.05
  Type: "Normalization"
  Samples: ["Signal", "Background"]
```

The approach of manually correlating different systematics however allows to define systematics in different ways (e.g. different normalization effect per sample), while still keeping them correlated.

Internally, `cabinetry` refers to systematics by their unique name up until the workspace building stage. For statistical inference, information contained in the workspace is used and thus the original systematics names are replaced by the values set in `ModifierName` (if that option is used).

CHAPTER

THREE

# CONFIGURATION SCHEMA

The configuration schema for `cabinetry` is given below. It is defined via a json schema that can be found at src/cabinetry/schemas/config.json.

The `General` block holds general settings, followed by blocks that take lists of objects: regions, samples, normalization factors, and systematics. The `Regions`, `Samples` and `NormFactors` blocks are required, while `Systematics` is optional. Settings shown in bold are required.

## 3.1 General

| general settings |
| --- |
| *General settings* |

## 3.2 Regions

| list of regions | |
| --- | --- |
| type | *array* |
| items | a region |
| | *Region* |
| minItems | 1 |
| uniqueItems | True |

## 3.3 Samples

| list of samples | |
| --- | --- |
| type | *array* |
| items | a sample |
| | *Sample* |
| minItems | 1 |
| uniqueItems | True |

## 3.4 NormFactors

| list of normalization factors | |
|---|---|
| type | *array* |
| items | a normalization factor<br>*NormFactor* |
| minItems | 1 |
| uniqueItems | True |

## 3.5 Systematics

| list of systematics | |
|---|---|
| type | *array* |
| items | a systematic uncertainty<br>*Systematic* |
| minItems | 0 |
| uniqueItems | True |

## 3.6 Details about the setting blocks:

### 3.6.1 General settings

| | | | | |
|---|---|---|---|---|
| general settings | | | | |
| type | *object* | | | |
| properties | | | | |
| • **Measurement** | name of measurement | | | |
| | type | *string* | | |
| • POI | name of parameter of interest, defaults to empty string | | | |
| | type | *string* | | |
| • **InputPath** | path to input files | | | |
| | type | *string* | | |
| • **HistogramFolder** | folder to save histograms to and read histograms from | | | |
| | type | *string* | | |
| • Fixed | list of parameters to treat as constant in fits | | | |
| | type | *array* | | |
| | items | a fixed parameter | | |
| | | type | *object* | |
| | | properties | | |
| | | • **Name** | name of fixed parameter | |
| | | | type | *string* |
| | | • **Value** | value to fix parameter to | |
| | | | type | *number* |
| | | additionalProperties | False | |
| | minItems | 1 | | |
| | uniqueItems | True | | |
| • VariationPath | (part of) path to file containing variation (for histogram inputs), defaults to empty string | | | |
| | type | *string* | | |
| additionalProperties | False | | | |

### 3.6.2 Region

| | | | |
|---|---|---|---|
| a region of phase space | | | |
| required: Name + Variable + Binning (for ntuple inputs), Name (for histogram inputs) | | | |
| type | *object* | | |
| properties | | | |
| • Name | name of the region | | |
| | type | *string* | |
| • Variable | variable to bin in | | |
| | type | *string* | |
| • Binning | binning to use in histograms | | |
| | type | *array* | |
| | items | bins | |
| | | type | *number* |
| | minItems | 2 | |
| | uniqueItems | True | |
| • Filter | selection criteria to apply | | |
| | type | *string* | |
| • RegionPath | (part of) path to file containing region | | |
| | type | *string* | |
| additionalProperties | False | | |

### 3.6.3 Sample

| | | |
|---|---|---|
| a sample of a specific process or data | | |
| required: Name + Tree (for ntuple inputs), Name (for histogram inputs) | | |
| type | *object* | |
| properties | | |
| • Name | name of the sample | |
| | type | *string* |
| • Tree | name of tree | |
| | type | *string* |
| • Filter | selection criteria to apply (override for region setting) | |
| | type | *string* |
| • Weight | weight to apply to events | |
| | type | *string* |
| • SamplePath | (part of) path(s) to input file(s) | |
| | *SamplePath setting* | |
| • Data | if it is a data sample | |
| | type | *boolean* |
| • DisableStaterror | whether to disable the automatic inclusion of staterror modifiers for this sample, defaults to False | |
| | type | *boolean* |
| • Regions | region(s) that contain the sample, defaults to all regions | |
| | *Regions setting* | |
| additionalProperties | False | |

### 3.6.4 NormFactor

| | | | |
|---|---|---|---|
| a normalization factor affecting one or more samples | | | |
| type | *object* | | |
| properties | | | |
| • **Name** | name of the normalization factor | | |
| | type | *string* | |
| • Regions | region(s) that contain the normfactor, defaults to all regions | | |
| | *Regions setting* | | |
| • Samples | affected sample(s), defaults to all samples | | |
| | *Sample setting* | | |
| • Nominal | nominal value | | |
| | type | *number* | |
| • Bounds | lower and upper bound | | |
| | type | *array* | |
| | items | bounds | |
| | | type | *number* |
| | maxItems | 2 | |
| | minItems | 2 | |
| | uniqueItems | True | |
| additionalProperties | False | | |

### 3.6.5 Systematic

| | | |
|---|---|---|
| a systematic uncertainty | | |
| type | *object* | |
| properties | | |
| • **Name** | name of the systematic uncertainty | |
| | type | *string* |
| • **Type** | type of systematic uncertainty | |
| | type | *string* |
| | enum | Normalization, NormPlusShape |
| • **Up** | template for “up” variation | |
| | *Template* | |
| • **Down** | template for “down” variation | |
| | *Template* | |
| • Regions | region(s) that contain the systematic, defaults to all regions | |
| | *Regions setting* | |
| • Samples | affected sample(s), defaults to all samples | |
| | *Sample setting* | |
| • Smoothing | smoothing to apply | |
| | *Smoothing setting* | |
| • ModifierName | name of modifier in workspace, defaults to value set by Name | |
| | type | *string* |
| additionalProperties | False | |

## 3.7 Common options:

### 3.7.1 Template

| | |
|---|---|
| a systematic template (up/down) | |
| type | *object* |
| properties | |
| • Tree | name of tree (override for nominal setting) |
| | type *string* |
| • Weight | weight to apply (override for nominal setting) |
| | type *string* |
| • Variable | variable to bin in (override for nominal setting) |
| | type *string* |
| • Filter | selection criteria to apply (override for region / sample setting) |
| | type *string* |
| • RegionPath | (part of) path to file containing region (override for nominal setting) |
| | type *string* |
| • SamplePath | (part of) path(s) to input file(s) (override for nominal setting) |
| | *SamplePath setting* |
| • Normalization | normalization uncertainty to apply |
| | type *number* |
| • Symmetrize | whether to apply symmetrization |
| | type *boolean* |
| • VariationPath | (part of) path to file containing variation (for histogram inputs, override for general setting) |
| | type *string* |
| additionalProperties | False |

### 3.7.2 Sample setting

| | | |
|---|---|---|
| name(s) of affected sample(s) | | |
| oneOf | single affected sample | |
| | type | *string* |
| | multiple affected samples | |
| | type | *array* |
| | items | single affected sample |
| | | type *string* |
| | minItems | 1 |
| | uniqueItems | True |

### 3.7.3 SamplePath setting

| | | |
|---|---|---|
| path(s) to input file(s) for histogram production | | |
| a list of paths is only supported for ntuple inputs | | |
| oneOf | path to single file | |
| | type | *string* |
| | list of paths | |
| | type | *array* |
| | items | path to single file |
| | | type *string* |
| | minItems | 1 |
| | uniqueItems | True |

### 3.7.4 Regions setting

| | | |
|---|---|---|
| name(s) of region(s) | | |
| oneOf | single region | |
| | type | *string* |
| | list of regions | |
| | type | *array* |
| | items | single region |
| | | type *string* |
| | minItems | 1 |
| | uniqueItems | True |

### 3.7.5 Smoothing setting

| | | |
|---|---|---|
| smoothing settings for template histograms | | |
| type | *object* | |
| properties | | |
| • **Algorithm** | name of smoothing algorithm to use | |
| | type | *string* |
| | enum | 353QH, twice |
| • Regions | regions to apply smoothing in | |
| | *Regions setting* | |
| • Samples | sample(s) to apply smoothing to, defaults to all samples | |
| | *Sample setting* | |
| additionalProperties | False | |

CHAPTER

FOUR

# CLI

After installing `cabinetry`, a command line interface is available. Below is an example workflow that builds template histograms defined by the config file `config_example.yml`, and applies post-processing to them. A `pyhf` workspace is then constructed and a maximum likelihood fit is performed. The resulting correlation matrix and pull plot are saved to the default output folder `figures/`.

```
cabinetry templates config_example.yml
cabinetry postprocess config_example.yml
cabinetry workspace config_example.yml workspaces/example_workspace.json
cabinetry fit --pulls --corrmat workspaces/example_workspace.json
```

The `--help` flag can be used to obtain more information on the command line:

```
cabinetry --help
```

shows the available commands, while

```
cabinetry fit --help
```

shows what the `fit` command does, and which options it accepts.

It is possible to read the `cabinetry` config and workspaces from stdin, and to write workspaces to stdout:

```
# read config from stdin
cat config_example.yml | cabinetry workspace - workspaces/example_workspace.json
# read workspace from stdin
cat workspaces/example_workspace.json | cabinetry fit -
# write workspace to stdout
cabinetry workspace config_example.yml -
```

## 4.1 cabinetry

Entrypoint to the cabinetry CLI.

```
cabinetry [OPTIONS] COMMAND [ARGS]...
```

#### Options

**--version**
Show the version and exit.

### 4.1.1 data-mc

Visualizes distributions of fit model and observed data.

WS_SPEC: path to workspace

```
cabinetry data-mc [OPTIONS] WS_SPEC
```

#### Options

**--config** <config>
cabinetry configuration file

**--postfit**
visualize post-fit model (default: pre-fit model)

**--figfolder** <figfolder>
folder to save figures to (default: "figures")

#### Arguments

**WS_SPEC**
Required argument

### 4.1.2 fit

Fits a workspace and optionally visualizes the results.

WS_SPEC: path to workspace used in fit

```
cabinetry fit [OPTIONS] WS_SPEC
```

#### Options

**--asimov**
fit Asimov dataset (default: False)

**--minos** <minos>
run MINOS for a parameter (default: disabled)

**--goodness_of_fit**
calculate goodness-of-fit (default: False)

**--pulls**
produce pull plot (default: False)

**--corrmat**
produce correlation matrix (default: False)

**--figfolder** <figfolder>
folder to save figures to (default: "figures")

#### Arguments

**WS_SPEC**
Required argument

### 4.1.3 limit

Calculates upper limits and visualizes CLs distribution.

WS_SPEC: path to workspace used in fit

```
cabinetry limit [OPTIONS] WS_SPEC
```

#### Options

**--asimov**
fit Asimov dataset (default: False)

**--tolerance** <tolerance>
tolerance for convergence to CLs=1-confidence_level (default: 0.01)

**--confidence_level, --cl** <confidence_level>
confidence level for parameter limits (default: 0.95)

**--figfolder** <figfolder>
folder to save figures to (default: "figures")

#### Arguments

**WS_SPEC**
Required argument

### 4.1.4 modifier-grid

Visualizes modifier structure of a model.

WS_SPEC: path to workspace

```
cabinetry modifier-grid [OPTIONS] WS_SPEC
```

#### Options

**--split_by_sample**
split grids by sample (default: split by channel)

**--figfolder** <figfolder>
folder to save figures to (default: "figures")

#### Arguments

**WS_SPEC**
Required argument

### 4.1.5 postprocess

Post-processes template histograms.

CONFIG: path to cabinetry configuration file

```
cabinetry postprocess [OPTIONS] CONFIG
```

#### Arguments

**CONFIG**
Required argument

### 4.1.6 ranking

Ranks nuisance parameters and visualizes the result.

WS_SPEC: path to workspace used in fit

```
cabinetry ranking [OPTIONS] WS_SPEC
```

#### Options

**--asimov**
fit Asimov dataset (default: False)

**--max_pars** <max_pars>
maximum amount of parameters in plot (default: 10)

**--figfolder** <figfolder>
folder to save figures to (default: "figures")

#### Arguments

**WS_SPEC**
Required argument

### 4.1.7 scan

Performs and visualizes a likelihood scan over a parameter.

Parameter bounds are determined automatically, unless both the `lower_bound` and `upper_bound` parameters are provided.

WS_SPEC: path to workspace used in fit

PAR_NAME: name of parameter to scan over

```
cabinetry scan [OPTIONS] WS_SPEC PAR_NAME
```

#### Options

**--lower_bound** <lower_bound>
lower parameter bound in scan (default: auto)

**--upper_bound** <upper_bound>
upper parameter bound in scan (default: auto)

**--n_steps** <n_steps>
number of steps in scan (default: 11)

**--asimov**
fit Asimov dataset (default: False)

**--figfolder** <figfolder>
folder to save figures to (default: "figures")

#### Arguments

**WS_SPEC**
Required argument

**PAR_NAME**
Required argument

### 4.1.8 significance

Calculates observed and expected discovery significance.

WS_SPEC: path to workspace used in fit

```
cabinetry significance [OPTIONS] WS_SPEC
```

#### Options

**--asimov**
fit Asimov dataset (default: False)

#### Arguments

**WS_SPEC**
Required argument

### 4.1.9 templates

Produces template histograms.

CONFIG: path to cabinetry configuration file

```
cabinetry templates [OPTIONS] CONFIG
```

#### Options

**--method** <method>
backend for histogram production (default: uproot)

#### Arguments

**CONFIG**
Required argument

### 4.1.10 workspace

Produces a `pyhf` workspace.

CONFIG: path to cabinetry configuration file

WS_SPEC: where to save the workspace containing the fit model

```
cabinetry workspace [OPTIONS] CONFIG WS_SPEC
```

#### Arguments

**CONFIG**
Required argument

**WS_SPEC**
Required argument

### 4.1.11 yields

Creates yield tables of fit model and observed data.

WS_SPEC: path to workspace

```
cabinetry yields [OPTIONS] WS_SPEC
```

#### Options

**--postfit**

show post-fit model (default: pre-fit model)

**--tablefolder** <tablefolder>

folder to save tables to (default: "tables")

**--tablefmt** <tablefmt>

format in which to save the table (default: "simple")

#### Arguments

**WS_SPEC**

Required argument

CHAPTER

# FIVE

# API


- *cabinetry.configuration*
- *cabinetry.route*
- *cabinetry.histo*
- *cabinetry.templates*
  - *cabinetry.templates.builder*
  - *cabinetry.templates.collector*
  - *cabinetry.templates.postprocessor*
  - *cabinetry.templates.utils*
- *cabinetry.workspace*
- *cabinetry.fit*
  - *cabinetry.fit.results_containers*
- *cabinetry.visualize*
  - *cabinetry.visualize.plot_model*
  - *cabinetry.visualize.plot_result*
  - *cabinetry.visualize.utils*
- *cabinetry.tabulate*
- *cabinetry.model_utils*
- *cabinetry.smooth*
- *cabinetry.contrib*
  - *cabinetry.contrib.histogram_creator*
  - *cabinetry.contrib.histogram_reader*

## 5.1 cabinetry.configuration

Provides utilities to handle the cabinetry configuration.

cabinetry.configuration.**histogram_is_needed**(*region: Dict[str, Any]*, *sample: Dict[str, Any]*, *systematic: Dict[str, Any]*, *template: Literal['Up', 'Down'] | None*) → bool

Determines whether a histogram is needed for a specific configuration.

The configuration is defined by the region, sample, systematic and template ("Up" or "Down", None for nominal).

**Parameters**

- **region** (*Dict[str, Any]*) – containing all region information
- **sample** (*Dict[str, Any]*) – containing all sample information
- **systematic** (*Dict[str, Any]*) – containing all systematic information
- **template** (*Optional[Literal["Up", "Down"]]*) – which template to consider: "Up", "Down", None for the nominal case

**Raises**

**NotImplementedError** – non-supported systematic variations based on histograms are requested

**Returns**

whether a histogram is needed

**Return type**

bool

cabinetry.configuration.**load**(*file_path_string: str | Path*) → Dict[str, Any]

Loads, validates, and returns a config file from the provided path.

**Parameters**

**file_path_string** (*Union[str, pathlib.Path]*) – path to config file

**Returns**

cabinetry configuration

**Return type**

Dict[str, Any]

cabinetry.configuration.**print_overview**(*config: Dict[str, Any]*) → None

Prints a compact summary of a config file.

**Parameters**

**config** (*Dict[str, Any]*) – cabinetry configuration

cabinetry.configuration.**region_contains_modifier**(*region: Dict[str, Any]*, *modifier: Dict[str, Any]*) → bool

Checks if a region contains a given modifier (Systematic, NormFactor).

A modifier affects all regions by default, and its "Regions" property can be used to specify a single region or list of regions that contain the modifier. This does not check whether the modifier only acts on samples which the region does not contain.

**Parameters**

- **region** (*Dict[str, Any]*) – containing all region information

- **modifier** (*Dict[str, Any]*) – containing all modifier information (a Systematic or a NormFactor)

**Returns**
True if region contains modifier, False otherwise

**Return type**
bool

cabinetry.configuration.**region_contains_sample**(*region: Dict[str, Any]*, *sample: Dict[str, Any]*) → bool

Checks if a region contains a given sample.

A sample enters all regions by default, and its "Regions" property can be used to specify a single region or list of regions that contain the sample.

**Parameters**

- **region** (*Dict[str, Any]*) – containing all region information
- **sample** (*Dict[str, Any]*) – containing all sample information

**Returns**
True if region contains sample, False otherwise

**Return type**
bool

cabinetry.configuration.**region_dict**(*config: Dict[str, Any]*, *region_name: str*) → Dict[str, Any]

Returns the dictionary for a region with the given name.

**Parameters**

- **config** (*Dict[str, Any]*) – cabinetry configuration file
- **region_name** (*str*) – name of region

**Raises**
**ValueError** – when region is not found in config

**Returns**
dictionary describing region

**Return type**
Dict[str, Any]

cabinetry.configuration.**sample_contains_modifier**(*sample: Dict[str, Any]*, *modifier: Dict[str, Any]*) → bool

Checks if a sample is affected by a given modifier (Systematic, NormFactor).

A modifier affects all samples by default, and its "Samples" property can be used to specify a single sample or list of samples on which the modifier acts.

**Parameters**

- **sample** (*Dict[str, Any]*) – containing all sample information
- **modifier** (*Dict[str, Any]*) – containing all modifier information (a Systematic or a NormFactor)

**Returns**
True if sample is affected, False otherwise

**Return type**
bool

cabinetry.configuration.**validate**(*config: Dict[str, Any]*) → bool

Returns True if the config file is validated, otherwise raises exceptions.

Checks that the config satisfies the json schema, and performs additional checks to validate the config further.

**Parameters**
**config** (*Dict[str, Any]*) – cabinetry configuration

**Raises**
- **NotImplementedError** – when more than one data sample is found
- **ValueError** – when region / sample / normfactor / systematic names are not unique

**Returns**
whether the validation was successful

**Return type**
bool

## 5.2 cabinetry.route

Provides features to apply functions to template histograms.

class cabinetry.route.**Router**

Holds user-defined processing functions and matches functions to templates.

Provides functions for matching a pattern to apply the right function to each template.

**template_builders**

user-defined processors for template building

**Type**
List[Dict[str, Any]]

**template_builder_wrapper**

wrapper to apply on user- defined template builders

**Type**
Optional[WrapperFunc]

**register_template_builder**(*, *region_name: str = '*'*, *sample_name: str = '*'*, *systematic_name: str = '*'*, *template: str | None = '*'*) → Callable[[Callable[[Dict[str, Any], Dict[str, Any], Dict[str, Any], str | None], Histogram]], Callable[[Dict[str, Any], Dict[str, Any], Dict[str, Any], str | None], Histogram]]

Decorator for registering a template builder function.

The function is added to the list stored in the `template_builders` member variable.

**Parameters**
- **region_name** (*str, optional*) – name of the region to apply the function to, defaults to "*" (apply to all regions)
- **sample_name** (*str, optional*) – name of the sample to apply the function to, defaults to "*" (apply to all samples)
- **systematic_name** (*str, optional*) – name of the systematic to apply the function to, defaults to "*" (apply to all systematics)

- **template** (*Optional[str], optional*) – name of the template to apply the function to (e.g. "Up" or "Down"), or None to apply to nominal only, defaults to "*" (apply to all templates, including nominal)

**Returns**
the function to register a processor

**Return type**
Callable[[UserTemplateFunc], UserTemplateFunc]

cabinetry.route.**apply_to_all_templates**(*config: Dict[str, Any]*, *default_func: Callable[[Dict[str, Any], Dict[str, Any], Dict[str, Any], Literal['Up', 'Down'] | None], None]*, *, *match_func: Callable[[str, str, str, Literal['Up', 'Down'] | None], Callable[[Dict[str, Any], Dict[str, Any], Dict[str, Any], Literal['Up', 'Down'] | None, None] | None] | None = None*) → None

Applies the supplied function `default_func` to all templates.

The templates are specified by the configuration file. The function takes four arguments in this order:

- the dict specifying region information
- the dict specifying sample information
- the dict specifying systematic information
- the template being considered: "Up", "Down", or None for the nominal template

In addition it is possible to specify a function that returns custom overrides. If one is found for a given template, it is used instead of the default.

**Parameters**

- **config** (*Dict[str, Any]*) – cabinetry configuration
- **default_func** (*ProcessorFunc*) – function to be called for every template by default
- **match_func** – (Optional[MatchFunc], optional): function that returns user-defined functions to override the call to `default_func`, defaults to None (then it is not used)

## 5.3 cabinetry.histo

Provides a histogram class based on boost-histogram.

class cabinetry.histo.**Histogram**(**axes: Axis | CppAxis | Histogram | Any*, *storage: Storage = Double()*, *metadata: Any | None = None*)

Holds histogram information, extends boost_histogram.Histogram.

property **bins: ndarray**

Returns the bin edges.

**Returns**
bin edges

**Return type**
np.ndarray

**classmethod from_arrays**(*bins: List[float] | ndarray*, *yields: List[float] | ndarray*, *stdev: List[float] | ndarray*) → H

Constructs a histogram from arrays of yields and uncertainties.

The input can be lists of ints or floats, or numpy.ndarrays.

**Parameters**

- **bins** (*Union[List[float], np.ndarray]*) – edges of histogram bins
- **yields** (*Union[List[float], np.ndarray]*) – yield per histogram bin
- **stdev** (*Union[List[float], np.ndarray]*) – statistical uncertainty of yield per bin

**Raises**

- **ValueError** – when amount of bins specified via bin edges and bin contents do not match
- **ValueError** – when length of yields and stdev do not match

**Returns**
the histogram instance

**Return type**
*cabinetry.histo.Histogram*

**classmethod from_config**(*histo_folder: str | Path*, *region: Dict[str, Any]*, *sample: Dict[str, Any]*, *systematic: Dict[str, Any]*, ***, *template: Literal['Up', 'Down'] | None = None*, *modified: bool = True*) → H

Loads a histogram, using information specified in the configuration file.

To find the histogram, need to provide the folder the histogram is located in and the relevant information from the config: region, sample, systematic, template.

**Parameters**

- **histo_folder** (*Union[str, patlib.Path]*) – folder containing all histograms
- **region** (*Dict[str, Any]*) – containing all region information
- **sample** (*Dict[str, Any]*) – containing all sample information
- **systematic** (*Dict[str, Any]*) – containing all systematic information
- **template** (*Optional[Literal["Up", "Down"]], optional*) – which template to consider: "Up", "Down", None for the nominal case, defaults to None
- **modified** (*bool, optional*) – whether to load the modified histogram (after post-processing), defaults to True

**Returns**
the loaded histogram

**Return type**
*cabinetry.histo.Histogram*

**classmethod from_path**(*histo_path: Path*, ***, *modified: bool = True*) → H

Builds a histogram from disk.

Loads the "modified" version of the histogram by default (which received post- processing).

**Parameters**

- **histo_path** (*pathlib.Path*) – where the histogram is located

- **modified** (*bool, optional*) – whether to load the modified histogram (after post-processing), defaults to True

**Returns**
the loaded histogram

**Return type**
*cabinetry.histo.Histogram*

**normalize_to_yield**(*reference_histogram: H*) → float

Normalizes a histogram to match the yield of a reference.

Returns the normalization factor used to normalize the histogram.

**Parameters**
**reference_histogram** (cabinetry.histo.Histogram) – reference histogram to normalize to

**Returns**
the yield ratio: un-normalized yield / normalized yield

**Return type**
float

**save**(*histo_path: Path*) → None

Saves a histogram to disk.

**Parameters**
**histo_path** (*pathlib.Path*) – where to save the histogram

**property stdev: ndarray**

Returns the stat. uncertainty per histogram bin.

**Returns**
stat. uncertainty per bin

**Return type**
np.ndarray

**validate**(*name: str*) → None

Runs consistency checks on a histogram.

Checks for empty bins and ill-defined statistical uncertainties. Logs warnings if issues are founds, but does not raise exceptions.

**Parameters**
**name** (*str*) – name of the histogram for logging purposes

**property yields: ndarray**

Returns the yields per histogram bin.

**Returns**
yields per bin

**Return type**
np.ndarray

cabinetry.histo.**name**(*region: Dict[str, Any]*, *sample: Dict[str, Any]*, *systematic: Dict[str, Any]*, ***, *template: Literal['Up', 'Down'] | None = None*) → str

Returns a unique name for each histogram.

If the template is not None, the systematic is required to have a name (guaranteed as long as it follows the config schema).

**Parameters**

- **region** (*Dict[str, Any]*) – containing all region information
- **sample** (*Dict[str, Any]*) – containing all sample information
- **systematic** (*Dict[str, Any]*) – containing all systematic information
- **template** (*Optional[Literal["Up", "Down"]], optional*) – which template to consider: "Up", "Down", None for the nominal case, defaults to None

**Returns**
unique name for the histogram

**Return type**
str

## 5.4 cabinetry.templates

High-level entry point to create, collect and post-process template histograms.

cabinetry.templates.**build**(*config: Dict[str, Any]*, *, *method: str = 'uproot'*, *router:* Router | *None = None*) → None

Produces all required histograms specified by the configuration file.

Inputs to the histogram production are ntuples containing columnar data. Uses either a default method specified via `method`, or a custom user-defined override through `router`.

**Parameters**

- **config** (*Dict[str, Any]*) – cabinetry configuration
- **method** (*str, optional*) – backend to use for histogram production, defaults to "uproot"
- **router** (*Optional[route.Router], optional*) – instance of cabinetry.route.Router that contains user-defined overrides, defaults to None

cabinetry.templates.**collect**(*config: Dict[str, Any]*, *, *method: str = 'uproot'*) → None

Collects all required histograms specified by the configuration file.

Histograms must already exist, and this collects and saves them in the format used for further processing. If no default for VariationPath is specified in the general settings, it defaults to an empty string.

**Parameters**

- **config** (*Dict[str, Any]*) – cabinetry configuration
- **method** (*str, optional*) – backend to use for histogram production, defaults to "uproot"

cabinetry.templates.**postprocess**(*config: Dict[str, Any]*) → None

Applies postprocessing to all histograms.

**Parameters**
**config** (*Dict[str, Any]*) – cabinetry configuration

### 5.4.1 cabinetry.templates.builder

Creates required template histograms from columnar data.

### 5.4.2 cabinetry.templates.collector

Collects required template histograms provided by user.

### 5.4.3 cabinetry.templates.postprocessor

Applies optional post-processing to template histograms.

cabinetry.templates.postprocessor.**apply_postprocessing**(*histogram:* Histogram, *name: str*, *, *smoothing_algorithm: str | None = None*, *nominal_histogram:* Histogram | *None = None*) → *Histogram*

Returns a new histogram with post-processing applied.

The histogram handed to the function stays unchanged. A copy of the histogram receives post-processing and is then returned. The postprocessing consists of a fix for NaN statistical uncertainties and optional smoothing.

**Parameters**

- **histogram** (*cabinetry.histo.Histogram*) – the histogram to postprocess
- **name** (*str*) – histogram name for logging
- **smoothing_algorithm** (*Optional[str]*) – name of smoothing algorithm to apply, defaults to None (no smoothing done)
- **nominal_histogram** (*Optional[cabinetry.histo.Histogram]*) – nominal histogram (needed for smoothing), defaults to None

**Returns**
the histogram with post-processing applied

**Return type**
*cabinetry.histo.Histogram*

### 5.4.4 cabinetry.templates.utils

Provides utilities for template histogram handling.

## 5.5 cabinetry.workspace

Constructs HistFactory workspaces in JSON format.

**class** cabinetry.workspace.**WorkspaceBuilder**(*config: Dict[str, Any]*)

Collects functionality to build a pyhf workspace.

**build**() → Dict[str, Any]

Constructs a *HistFactory* workspace in pyhf format.

**Returns**
pyhf-compatible *HistFactory* workspace

**Return type**
Dict[str, Any]

`channels`() → List[Dict[str, Any]]

Returns the channel information: yields per sample and modifiers.

**Returns**
channels for `pyhf` workspace

**Return type**
List[Dict[str, Any]]

`measurements`() → List[Dict[str, Any]]

Returns the measurements object for the workspace.

Constructs the measurements, including POI setting and parameter bounds, initial values and whether they are set to constant. Only supports a single measurement so far.

**Returns**
measurements for `pyhf` workspace

**Return type**
List[Dict[str, Any]]

`static normalization_modifier`(*systematic: Dict[str, Any]*) → Dict[str, Any]

Returns a normalization modifier (OverallSys in *HistFactory*).

**Parameters**
**`systematic`** (*`Dict[str, Any]`*) – systematic for which modifier is constructed

**Returns**
single *normsys* modifier for `pyhf` workspace

**Return type**
Dict[str, Any]

`normfactor_modifiers`(*region: Dict[str, Any]*, *sample: Dict[str, Any]*) → List[Dict[str, Any]]

Returns the list of NormFactor modifiers acting on a sample in a region.

**Parameters**

- **`region`** (*`Dict[str, Any]`*) – specific region to get NormFactor modifiers for
- **`sample`** (*`Dict[str, Any]`*) – specific sample to get NormFactor modifiers for

**Returns**
NormFactor modifiers for sample

**Return type**
List[Dict[str, Any]]

`normplusshape_modifiers`(*region: Dict[str, Any]*, *sample: Dict[str, Any]*, *systematic: Dict[str, Any]*) → List[Dict[str, Any]]

Returns modifiers for a correlated shape + normalization effect.

For a variation including a correlated shape + normalization effect, this provides the *histosys* and *normsys* modifiers for `pyhf` (in *HistFactory* language, this corresponds to a *HistoSys* and an *OverallSys*). Symmetrization could happen either at this stage (this is the case currently), or somewhere earlier, such as during template postprocessing.

**Parameters**

- **`region`** (*`Dict[str, Any]`*) – region the systematic variation acts in

- **sample** (*Dict[str, Any]*) – sample the systematic variation acts on
- **systematic** (*Dict[str, Any]*) – the systematic variation under consideration

**Returns**
a list with a pyhf *normsys* modifier and a *histosys* modifier

**Return type**
List[Dict[str, Any]]

**observations**() → List[Dict[str, Any]]

Returns the observations object (with data yields) for the workspace.

**Returns**
observations for pyhf workspace

**Return type**
List[Dict[str, Any]]

**sys_modifiers**(*region: Dict[str, Any]*, *sample: Dict[str, Any]*) → List[Dict[str, Any]]

Returns the list of all systematic modifiers acting on a sample in a region.

**Parameters**

- **region** (*Dict[str, Any]*) – region considered
- **sample** (*Dict[str, Any]*) – specific sample to get modifiers for

**Raises**
**NotImplementedError** – when unsupported modifiers act on sample

**Returns**
modifiers for pyhf workspace

**Return type**
List[Dict[str, Any]]

cabinetry.workspace.**build**(*config: Dict[str, Any]*, ***, *with_validation: bool = True*) → Dict[str, Any]

Returns a *HistFactory* workspace in pyhf format.

**Parameters**

- **config** (*Dict[str, Any]*) – cabinetry configuration
- **with_validation** (*bool, optional*) – validate workspace validity with pyhf, defaults to True

**Returns**
pyhf-compatible *HistFactory* workspace

**Return type**
Dict[str, Any]

cabinetry.workspace.**load**(*file_path_string: str | Path*) → Dict[str, Any]

Loads a workspace from a file.

**Parameters**
**file_path_string** (*Union[str, pathlib.Path]*) – path to the file to load the workspace from

**Returns**
pyhf-compatible *HistFactory* workspace

**Return type**
Dict[str, Any]

cabinetry.workspace.**save**(*ws: Dict[str, Any]*, *file_path_string: str | Path*) → None

Serializes a workspace to a file.

**Parameters**

- **ws** (*Dict[str, Any]*) – pyhf-compatible *HistFactory* workspace
- **file_path_string** (*Union[str, pathlib.Path]*) – path to the file to save the workspace in

cabinetry.workspace.**validate**(*ws: Dict[str, Any]*) → None

Validates a workspace with pyhf.

**Parameters**

**ws** (*Dict[str, Any]*) – the workspace to validate

## 5.6 cabinetry.fit

High-level entry point for statistical inference.

cabinetry.fit.**fit**(*model: Model*, *data: List[float]*, ***, *minos: str | List[str] | Tuple[str, ...] | None = None*, *goodness_of_fit: bool = False*, *init_pars: List[float] | None = None*, *fix_pars: List[bool] | None = None*, *par_bounds: List[Tuple[float, float]] | None = None*, *strategy: Literal[0, 1, 2] | None = None*, *maxiter: int | None = None*, *tolerance: float | None = None*, *custom_fit: bool = False*) → *FitResults*

Performs a maximum likelihood fit, reports and returns the results.

Depending on the custom_fit keyword argument, this uses either the pyhf.infer API or iminuit directly.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to use in fit
- **data** (*List[float]*) – data (including auxdata) the model is fit to
- **minos** (*Optional[Union[str, List[str], Tuple[str, ...]]], optional*) – runs the MINOS algorithm for all parameters specified, defaults to None (does not run MINOS)
- **goodness_of_fit** (*bool, optional*) – calculate goodness of fit with a saturated model (perfectly fits data with shapefactors in all bins), defaults to False
- **init_pars** (*Optional[List[float]], optional*) – list of initial parameter settings, defaults to None (use pyhf suggested inits)
- **fix_pars** (*Optional[List[bool]], optional*) – list of booleans specifying which parameters are held constant, defaults to None (use pyhf suggestion)
- **par_bounds** (*Optional[List[Tuple[float, float]]], optional*) – list of tuples with parameter bounds for fit, defaults to None (use pyhf suggested bounds)
- **strategy** (*Optional[Literal[0, 1, 2]], optional*) – minimization strategy used by Minuit, can be 0/1/2, defaults to None (then uses pyhf default behavior of strategy 0 with user-provided gradients and 1 otherwise)
- **maxiter** (*Optional[int], optional*) – allowed number of calls for minimization, defaults to None (use pyhf default of 100,000)
- **tolerance** (*Optional[float]), optional*) – tolerance for convergence, for details see iminuit.Minuit.tol (uses EDM < 0.002*tolerance), defaults to None (use iminuit default of 0.1)

- **custom_fit** (*bool, optional*) – whether to use the pyhf.infer API or iminuit, defaults to False (using pyhf.infer)

**Returns**
object storing relevant fit results

**Return type**
*FitResults*

cabinetry.fit.**limit**(*model: Model*, *data: List[float]*, ***, *bracket: List[float] | Tuple[float, float] | None = None*, *poi_tolerance: float = 0.01*, *maxsteps: int = 100*, *confidence_level: float = 0.95*, *poi_name: str | None = None*, *init_pars: List[float] | None = None*, *fix_pars: List[bool] | None = None*, *par_bounds: List[Tuple[float, float]] | None = None*, *strategy: Literal[0, 1, 2] | None = None*, *maxiter: int | None = None*, *tolerance: float | None = None*) → *LimitResults*

Calculates observed and expected upper parameter limits.

Limits are calculated for the parameter of interest (POI) defined in the model. Brent's algorithm is used to automatically determine POI values to be tested. The desired confidence level can be configured, and defaults to 95%. In order to support setting the POI directly without model recompilation, this temporarily changes the POI in the model configuration.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to use in fits
- **data** (*List[float]*) – data (including auxdata) the model is fit to
- **bracket** (*Optional[Union[List[float], Tuple[float, float]]], optional*) – the two POI values used to start the observed limit determination, the limit must lie between these values and the values must not be the same, defaults to None (then uses 0.1 as default lower value and the upper POI bound specified in the measurement as default upper value)
- **poi_tolerance** (*float, optional*) – tolerance in POI value for convergence to target CLs value (1-confidence_level), defaults to 0.01
- **maxsteps** (*int, optional*) – maximum number of steps for limit finding, defaults to 100
- **confidence_level** (*float, optional*) – confidence level for calculation, defaults to 0.95 (95%)
- **poi_name** (*Optional[str], optional*) – limit is calculated for this parameter, defaults to None (use POI specified in workspace)
- **init_pars** (*Optional[List[float]], optional*) – list of initial parameter settings, defaults to None (use pyhf suggested inits)
- **fix_pars** (*Optional[List[bool]], optional*) – list of booleans specifying which parameters are held constant, defaults to None (use pyhf suggestion)
- **par_bounds** (*Optional[List[Tuple[float, float]]], optional*) – list of tuples with parameter bounds for fit, defaults to None (use pyhf suggested bounds)
- **strategy** (*Optional[Literal[0, 1, 2]], optional*) – minimization strategy used by Minuit, can be 0/1/2, defaults to None (then uses pyhf default behavior of strategy 0 with user-provided gradients and 1 otherwise)
- **maxiter** (*Optional[int], optional*) – allowed number of calls for minimization, defaults to None (use pyhf default of 100,000)
- **tolerance** (*Optional[float]), optional*) – tolerance for convergence, for details see iminuit.Minuit.tol (uses EDM < 0.002*tolerance), defaults to None (use iminuit default of 0.1)

**Raises**

- **ValueError** – if no POI is found
- **ValueError** – if lower and upper bracket value are the same
- **ValueError** – if starting brackets do not enclose the limit

**Returns**
observed and expected limits, CLs values, and scanned points

**Return type**
*LimitResults*

cabinetry.fit.**print_results**(*fit_results:* FitResults) → None

Prints the best-fit parameter results and associated uncertainties.

**Parameters**
**fit_results** (FitResults) – results of fit to be printed

cabinetry.fit.**ranking**(*model: Model*, *data: List[float]*, **, fit_results:* FitResults | *None = None*, *poi_name: str | None = None*, *init_pars: List[float] | None = None*, *fix_pars: List[bool] | None = None*, *par_bounds: List[Tuple[float, float]] | None = None*, *strategy: Literal[0, 1, 2] | None = None*, *maxiter: int | None = None*, *tolerance: float | None = None*, *custom_fit: bool = False*) → *RankingResults*

Calculates the impact of nuisance parameters on the parameter of interest (POI).

The impact is given by the difference in the POI between the nominal fit, and a fit where the nuisance parameter is held constant at its nominal value plus/minus its associated uncertainty. The "pre-fit impact" is obtained by varying the nuisance parameters by their uncertainty given by their constraint term.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to use in fits
- **data** (*List[float]*) – data (including auxdata) the model is fit to
- **fit_results** (*Optional[FitResults], optional*) – nominal fit results to use for ranking, if not specified will repeat nominal fit, defaults to None
- **poi_name** (*Optional[str], optional*) – impact is calculated with respect to this parameter, defaults to None (use POI specified in workspace)
- **init_pars** (*Optional[List[float]], optional*) – list of initial parameter settings, defaults to None (use pyhf suggested inits)
- **fix_pars** (*Optional[List[bool]], optional*) – list of booleans specifying which parameters are held constant, defaults to None (use pyhf suggestion)
- **par_bounds** (*Optional[List[Tuple[float, float]]], optional*) – list of tuples with parameter bounds for fit, defaults to None (use pyhf suggested bounds)
- **strategy** (*Optional[Literal[0, 1, 2]], optional*) – minimization strategy used by Minuit, can be 0/1/2, defaults to None (then uses pyhf default behavior of strategy 0 with user-provided gradients and 1 otherwise)
- **maxiter** (*Optional[int], optional*) – allowed number of calls for minimization, defaults to None (use pyhf default of 100,000)
- **tolerance** (*Optional[float]), optional*) – tolerance for convergence, for details see iminuit.Minuit.tol (uses EDM < 0.002*tolerance), defaults to None (use iminuit default of 0.1)

- **custom_fit** (*bool, optional*) – whether to use the pyhf.infer API or iminuit, defaults to False (using pyhf.infer)

**Raises**
**ValueError** – if no POI is found

**Returns**
fit results for parameters, and pre- and post-fit impacts

**Return type**
*RankingResults*

cabinetry.fit.**scan**(*model: Model*, *data: List[float]*, *par_name: str*, ***, *par_range: Tuple[float, float] | None = None*, *n_steps: int = 11*, *init_pars: List[float] | None = None*, *fix_pars: List[bool] | None = None*, *par_bounds: List[Tuple[float, float]] | None = None*, *strategy: Literal[0, 1, 2] | None = None*, *maxiter: int | None = None*, *tolerance: float | None = None*, *custom_fit: bool = False*) → *ScanResults*

Performs a likelihood scan over the specified parameter.

If no parameter range is specified, center the scan around the best-fit result for the parameter that is being scanned, and scan over twice its uncertainty in each direction. The reported likelihood values are the differences between -2 log(L) at each point in the scan and the global minimum.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to use in fits
- **data** (*List[float]*) – data (including auxdata) the model is fit to
- **par_name** (*str*) – name of parameter to scan over
- **par_range** (*Optional[Tuple[float, float]], optional*) – upper and lower bounds of parameter in scan, defaults to None (automatically determine bounds)
- **n_steps** (*int, optional*) – number of steps in scan, defaults to 10
- **init_pars** (*Optional[List[float]], optional*) – list of initial parameter settings, defaults to None (use pyhf suggested inits)
- **fix_pars** (*Optional[List[bool]], optional*) – list of booleans specifying which parameters are held constant, defaults to None (use pyhf suggestion)
- **par_bounds** (*Optional[List[Tuple[float, float]]], optional*) – list of tuples with parameter bounds for fit, defaults to None (use pyhf suggested bounds)
- **strategy** (*Optional[Literal[0, 1, 2]], optional*) – minimization strategy used by Minuit, can be 0/1/2, defaults to None (then uses pyhf default behavior of strategy 0 with user-provided gradients and 1 otherwise)
- **maxiter** (*Optional[int], optional*) – allowed number of calls for minimization, defaults to None (use pyhf default of 100,000)
- **tolerance** (*Optional[float]), optional*) – tolerance for convergence, for details see iminuit.Minuit.tol (uses EDM < 0.002*tolerance), defaults to None (use iminuit default of 0.1)
- **custom_fit** (*bool, optional*) – whether to use the pyhf.infer API or iminuit, defaults to False (using pyhf.infer)

**Raises**
**ValueError** – if parameter is not found in model

**Returns**
includes parameter name, scanned values and 2*log(likelihood) offset

**Return type**
*ScanResults*

cabinetry.fit.**significance**(*model: Model*, *data: List[float]*, ***, *poi_name: str | None = None*, *init_pars: List[float] | None = None*, *fix_pars: List[bool] | None = None*, *par_bounds: List[Tuple[float, float]] | None = None*, *strategy: Literal[0, 1, 2] | None = None*, *maxiter: int | None = None*, *tolerance: float | None = None*) → *SignificanceResults*

Calculates the discovery significance of a positive signal.

Observed and expected p-values and significances are both calculated and reported.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to use in fits
- **data** (*List[float]*) – data (including auxdata) the model is fit to
- **poi_name** (*Optional[str], optional*) – significance is calculated for this parameter, defaults to None (use POI specified in workspace)
- **init_pars** (*Optional[List[float]], optional*) – list of initial parameter settings, defaults to None (use `pyhf` suggested inits)
- **fix_pars** (*Optional[List[bool]], optional*) – list of booleans specifying which parameters are held constant, defaults to None (use `pyhf` suggestion)
- **par_bounds** (*Optional[List[Tuple[float, float]]], optional*) – list of tuples with parameter bounds for fit, defaults to None (use `pyhf` suggested bounds)
- **strategy** (*Optional[Literal[0, 1, 2]], optional*) – minimization strategy used by Minuit, can be 0/1/2, defaults to None (then uses `pyhf` default behavior of strategy 0 with user-provided gradients and 1 otherwise)
- **maxiter** (*Optional[int], optional*) – allowed number of calls for minimization, defaults to None (use `pyhf` default of 100,000)
- **tolerance** (*Optional[float]), optional*) – tolerance for convergence, for details see `iminuit.Minuit.tol` (uses EDM < 0.002*tolerance), defaults to None (use `iminuit` default of 0.1)

**Returns**
observed and expected p-values and significances

**Return type**
*SignificanceResults*

### 5.6.1 cabinetry.fit.results_containers

Provides containers for inference results.

class cabinetry.fit.results_containers.**FitResults**(*bestfit: ndarray*, *uncertainty: ndarray*, *labels: List[str]*, *corr_mat: ndarray*, *best_twice_nll: float*, *goodness_of_fit: float = -1*, *minos_uncertainty: Dict[str, Tuple[float, float]] = {}*)

Collects fit results in one object.

**Parameters**

- **bestfit** (*np.ndarray*) – best-fit results of parameters

- **uncertainty** (*np.ndarray*) – uncertainties of best-fit parameter results, evaluated with Hessian
- **labels** (*List[str]*) – parameter labels
- **corr_mat** (*np.ndarray*) – parameter correlation matrix
- **best_twice_nll** (*float*) – -2 log(likelihood) at best-fit point
- **goodess_of_fit** (*float, optional*) – goodness-of-fit p-value, defaults to -1
- **minos_uncertainty** (*Dict[str, Tuple[float, float]]*) – uncertainties of best-fit parameter results indexed by parameter name, calculated with MINOS

**best_twice_nll: float**

Alias for field number 4

**bestfit: ndarray**

Alias for field number 0

**corr_mat: ndarray**

Alias for field number 3

**goodness_of_fit: float**

Alias for field number 5

**labels: List[str]**

Alias for field number 2

**minos_uncertainty: Dict[str, Tuple[float, float]]**

Alias for field number 6

**uncertainty: ndarray**

Alias for field number 1

class cabinetry.fit.results_containers.**LimitResults**(*observed_limit: float*, *expected_limit: ndarray*, *observed_CLs: ndarray*, *expected_CLs: ndarray*, *poi_values: ndarray*, *confidence_level: float*)

Collects parameter upper limit results in one object.

**Parameters**

- **observed_limit** (*float*) – observed limit
- **expected_limit** (*np.ndarray*) – expected limit, including 1 and 2 sigma bands
- **observed_CLs** (*np.ndarray*) – observed CLs values
- **expected_CLs** (*np.ndarray*) – expected CLs values, including 1 and 2 sigma bands
- **poi_values** (*np.ndarray*) – POI values used in scan
- **confidence_level** (*float*) – confidence level used for parameter limits

**confidence_level: float**

Alias for field number 5

**expected_CLs: ndarray**

Alias for field number 3

**expected_limit: ndarray**

Alias for field number 1

**observed_CLs: ndarray**

Alias for field number 2

**observed_limit: float**

Alias for field number 0

**poi_values: ndarray**

Alias for field number 4

class cabinetry.fit.results_containers.**RankingResults**(*bestfit: ndarray*, *uncertainty: ndarray*, *labels: List[str]*, *prefit_up: ndarray*, *prefit_down: ndarray*, *postfit_up: ndarray*, *postfit_down: ndarray*)

Collects nuisance parameter ranking results in one object.

The best-fit results per parameter, the uncertainties, and the labels should not include the parameter of interest, since no impact for it is calculated.

**Parameters**

- **bestfit** (*np.ndarray*) – best-fit results of parameters
- **uncertainty** (*np.ndarray*) – uncertainties of best-fit parameter results
- **labels** (*List[str]*) – parameter labels
- **prefit_up** (*np.ndarray*) – pre-fit impact in "up" direction
- **prefit_down** (*np.ndarray*) – pre-fit impact in "down" direction
- **postfit_up** (*np.ndarray*) – post-fit impact in "up" direction
- **postfit_down** (*np.ndarray*) – post-fit impact in "down" direction

**bestfit: ndarray**

Alias for field number 0

**labels: List[str]**

Alias for field number 2

**postfit_down: ndarray**

Alias for field number 6

**postfit_up: ndarray**

Alias for field number 5

**prefit_down: ndarray**

Alias for field number 4

**prefit_up: ndarray**

Alias for field number 3

**uncertainty: ndarray**

Alias for field number 1

class cabinetry.fit.results_containers.**ScanResults**(*name: str*, *bestfit: float*, *uncertainty: float*, *parameter_values: ndarray*, *delta_nlls: ndarray*)

Collects likelihood scan results in one object.

**Parameters**

- **name** (*str*) – name of parameter in scan
- **bestfit** (*float*) – best-fit parameter value from unconstrained fit
- **uncertainty** (*float*) – uncertainty of parameter in unconstrained fit
- **parameter_values** (*np.ndarray*) – parameter values used in scan
- **delta_nlls** (*np.ndarray*) – -2 log(L) difference at each scanned point

**bestfit: float**

Alias for field number 1

**delta_nlls: ndarray**

Alias for field number 4

**name: str**

Alias for field number 0

**parameter_values: ndarray**

Alias for field number 3

**uncertainty: float**

Alias for field number 2

**class** cabinetry.fit.results_containers.**SignificanceResults**(*observed_p_value: float*, *observed_significance: float*, *expected_p_value: float*, *expected_significance: float*)

Collects results from a discovery significance calculation in one object.

**Parameters**

- **observed_p_value** (*float*) – observed p-value
- **observed_significance** (*float*) – observed significance
- **expected_p_value** (*float*) – expected/median p-value
- **expected_significance** (*float*) – expected/median significance

**expected_p_value: float**

Alias for field number 2

**expected_significance: float**

Alias for field number 3

**observed_p_value: float**

Alias for field number 0

**observed_significance: float**

Alias for field number 1

## 5.7 cabinetry.visualize

High-level entry point for visualizing fit models and inference results.

cabinetry.visualize.**correlation_matrix**(*fit_results: FitResults*, **, figure_folder: str | Path = 'figures'*, *pruning_threshold: float = 0.0*, *close_figure: bool = True*, *save_figure: bool = True*) → Figure

Draws a correlation matrix.

**Parameters**

- **fit_results** (*fit.FitResults*) – fit results, including correlation matrix and parameter labels
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"
- **pruning_threshold** (*float, optional*) – minimum correlation for a parameter to have with any other parameters to not get pruned, defaults to 0.0
- **close_figure** (*bool, optional*) – whether to close figure, defaults to True
- **save_figure** (*bool, optional*) – whether to save figure, defaults to True

**Returns**
the correlation matrix figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.**data_mc**(*model_prediction: ModelPrediction*, *data: List[float]*, **, config: Dict[str, Any] | None = None*, *figure_folder: str | Path = 'figures'*, *log_scale: bool | None = None*, *log_scale_x: bool = False*, *channels: str | List[str] | None = None*, *close_figure: bool = False*, *save_figure: bool = True*) → List[Dict[str, Any]] | None

Draws pre- and post-fit data/MC histograms for a `pyhf` model and data.

The `config` argument is optional, but required to determine correct axis labels and binning. The information is not stored in the model, and default values are used if no `config` is supplied. This allows quickly plotting distributions for models that were not created with `cabinetry`, and for which no config exists.

**Parameters**

- **model_prediction** (model_utils.ModelPrediction) – model prediction to show
- **data** (*List[float]*) – data to include in visualization, can either include auxdata (the auxdata is then stripped internally) or only observed yields
- **config** (*Optional[Dict[str, Any]], optional*) – cabinetry configuration needed for binning and axis labels, defaults to None (uses a default binning and labels then)
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"
- **log_scale** (*Optional[bool], optional*) – whether to use logarithmic vertical axis, defaults to None (automatically determine whether to use linear/log scale)
- **log_scale_x** (*bool, optional*) – whether to use logarithmic horizontal axis, defaults to False
- **channels** (*Optional[Union[str, List[str]]], optional*) – name of channel to show, or list of names to include, defaults to None (uses all channels)

- **close_figure** (*bool, optional*) – whether to close each figure, defaults to False (enable when producing many figures to avoid memory issues, prevents automatic rendering in notebooks)
- **save_figure** (*bool, optional*) – whether to save figures, defaults to True

**Returns**

**list of dictionaries, where each dictionary**
contains a figure and the associated region name, or None if no figure was produced

**Return type**
Optional[List[Dict[str, Any]]]

cabinetry.visualize.**data_mc_from_histograms**(*config: Dict[str, Any]*, ***, *figure_folder: str | Path = 'figures'*, *log_scale: bool | None = None*, *log_scale_x: bool = False*, *close_figure: bool = False*, *save_figure: bool = True*) → List[Dict[str, Any]]

Draws pre-fit data/MC histograms, using histograms created by cabinetry.

The uncertainty band drawn includes only statistical uncertainties.

**Parameters**

- **config** (*Dict[str, Any]*) – cabinetry configuration
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"
- **log_scale** (*Optional[bool], optional*) – whether to use logarithmic vertical axis, defaults to None (automatically determine whether to use linear/log scale)
- **log_scale_x** (*bool, optional*) – whether to use logarithmic horizontal axis, defaults to False
- **close_figure** (*bool, optional*) – whether to close each figure, defaults to False (enable when producing many figures to avoid memory issues, prevents automatic rendering in notebooks)
- **save_figure** (*bool, optional*) – whether to save figures, defaults to True

**Returns**

**list of dictionaries, where each dictionary contains a**
figure and the associated region name

**Return type**
List[Dict[str, Any]]

cabinetry.visualize.**limit**(*limit_results:* LimitResults, ***, *figure_folder: str | Path = 'figures'*, *close_figure: bool = True*, *save_figure: bool = True*) → Figure

Visualizes observed and expected CLs values as a function of the POI.

**Parameters**

- **limit_results** (*fit.LimitResults*) – results of upper limit determination
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"
- **close_figure** (*bool, optional*) – whether to close figure, defaults to True
- **save_figure** (*bool, optional*) – whether to save figure, defaults to True

**Returns**
the CLs figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.**modifier_grid**(*model: Model*, ***, *figure_folder: str | Path = 'figures'*, *split_by_sample: bool = False*, *close_figure: bool = True*, *save_figure: bool = True*) → Figure

Visualizes the modifier structure of a model in a 2d grid.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to visualize
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"
- **split_by_sample** (*bool, optional*) – whether to use (channel, parameter) grids for each sample, defaults to False (if enabled, uses (sample, parameter) grids for each channel)
- **close_figure** (*bool, optional*) – whether to close figure, defaults to True
- **save_figure** (*bool, optional*) – whether to save figure, defaults to True

cabinetry.visualize.**pulls**(*fit_results: FitResults*, ***, *figure_folder: str | Path = 'figures'*, *exclude: str | List[str] | Tuple[str, ...] | None = None*, *close_figure: bool = True*, *save_figure: bool = True*) → Figure

Draws a pull plot of parameter results and uncertainties.

**Parameters**

- **fit_results** (*fit.FitResults*) – fit results, including correlation matrix and parameter labels
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"
- **exclude** (*Optional[Union[str, List[str], Tuple[str, ...]]], optional*) – parameter or parameters to exclude from plot, defaults to None (nothing excluded)
- **close_figure** (*bool, optional*) – whether to close figure, defaults to True
- **save_figure** (*bool, optional*) – whether to save figure, defaults to True

**Returns**
the pull figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.**ranking**(*ranking_results: RankingResults*, ***, *figure_folder: str | Path = 'figures'*, *max_pars: int | None = None*, *close_figure: bool = True*, *save_figure: bool = True*) → Figure

Produces a ranking plot showing the impact of parameters on the POI.

The parameters are shown in decreasing order of greatest post-fit impact.

**Parameters**

- **ranking_results** (*fit.RankingResults*) – fit results, and pre- and post-fit impacts
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"

- **max_pars** (*Optional[int], optional*) – number of parameters to include, defaults to None (which means all parameters are included)
- **close_figure** (*bool, optional*) – whether to close figure, defaults to True
- **save_figure** (*bool, optional*) – whether to save figure, defaults to True

**Returns**
the ranking figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.**scan**(*scan_results:* ScanResults, *, *figure_folder: str | Path = 'figures'*, *close_figure: bool = True*, *save_figure: bool = True*) → Figure

Visualizes the results of a likelihood scan.

**Parameters**

- **scan_results** (*fit.ScanResults*) – results of a likelihood scan
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"
- **close_figure** (*bool, optional*) – whether to close figure, defaults to True
- **save_figure** (*bool, optional*) – whether to save figure, defaults to True

**Returns**
the likelihood scan figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.**templates**(*config: Dict[str, Any]*, *, *figure_folder: str | Path = 'figures'*, *close_figure: bool = False*, *save_figure: bool = True*) → List[Dict[str, Any]]

Visualizes template histograms (after post-processing) for systematic variations.

The original template histogram for systematic variations (before post-processing) is also included in the visualization.

**Parameters**

- **config** (*Dict[str, Any]*) – cabinetry configuration
- **figure_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save figures in, defaults to "figures"
- **close_figure** (*bool, optional*) – whether to close each figure, defaults to False (enable when producing many figures to avoid memory issues, prevents automatic rendering in notebooks)
- **save_figure** (*bool, optional*) – whether to save figures, defaults to True

**Returns**

**list of dictionaries, where each dictionary contains a**
figure and the associated region / sample / systematic names

**Return type**
List[Dict[str, Any]]

### 5.7.1 cabinetry.visualize.plot_model

Visualizes fit models with matplotlib.

cabinetry.visualize.plot_model.**data_mc**(*histogram_dict_list: List[Dict[str, Any]]*, *total_model_unc: ndarray*, *bin_edges: ndarray*, ***, *figure_path: Path | None = None*, *log_scale: bool | None = None*, *log_scale_x: bool = False*, *label: str = ''*, *close_figure: bool = False*) → Figure

Draws a data/MC histogram with uncertainty bands and ratio panel.

**Parameters**

- **histogram_dict_list** (*List[Dict[str, Any]]*) – list of samples (with info stored in one dict per sample)
- **total_model_unc** (*np.ndarray*) – total model uncertainty, if specified this is used instead of calculating it via sum in quadrature, defaults to None
- **bin_edges** (*np.ndarray*) – bin edges of histogram
- **figure_path** (*Optional[pathlib.Path], optional*) – path where figure should be saved, or None to not save it, defaults to None
- **log_scale** (*Optional[bool], optional*) – whether to use a logarithmic vertical axis, defaults to None (automatically determine whether to use linear or log scale)
- **log_scale_x** (*bool, optional*) – whether to use logarithmic horizontal axis, defaults to False
- **label** (*str, optional*) – label written on the figure, defaults to “”
- **close_figure** (*bool, optional*) – whether to close each figure immediately after saving it, defaults to False (enable when producing many figures to avoid memory issues, prevents rendering in notebooks)

**Raises**

**ValueError** – when total model yield is negative in any bin

**Returns**

the data/MC figure

**Return type**

matplotlib.figure.Figure

cabinetry.visualize.plot_model.**modifier_grid**(*grid_list: List[ndarray]*, *axis_labels: List[List[str]]*, *category_map: Dict[int, str]*, *figure_path: Path | None = None*, *close_figure: bool = False*) → Figure

Draws a grid of modifiers per channel, sample and parameter.

**Parameters**

- **grid_list** (*List[np.ndarray]*) – list of 2d grids with modifier information
- **axis_labels** (*List[List[str]]*) – list with axis labels for the three axes in order (first axis is grid label, second and third are axes per grid)
- **category_map** (*Dict[int, str]*) – translation of integer values in grid to labels
- **figure_path** (*Optional[pathlib.Path], optional*) – path where figure should be saved, or None to not save it, defaults to None

- **close_figure** (*bool, optional*) – whether to close each figure immediately after saving it, defaults to False (enable when producing many figures to avoid memory issues, prevents rendering in notebooks)

cabinetry.visualize.plot_model.**templates**(*nominal_histo: Dict[str, ndarray]*, *up_histo_orig: Dict[str, ndarray]*, *down_histo_orig: Dict[str, ndarray]*, *up_histo_mod: Dict[str, ndarray]*, *down_histo_mod: Dict[str, ndarray]*, *bin_edges: ndarray*, *variable: str*, ***, *figure_path: Path | None = None*, *label: str = ''*, *close_figure: bool = False*) → Figure

Draws a nominal template and the associated up/down variations.

If a variation template is an empty dict, it is not drawn.

**Parameters**

- **nominal_histo** (*Dict[str, np.ndarray]*) – the nominal template
- **up_histo_orig** (*Dict[str, np.ndarray]*) – original "up" variation
- **down_histo_orig** (*Dict[str, np.ndarray]*) – original "down" variation
- **up_histo_mod** (*Dict[str, np.ndarray]*) – "up" variation after post-processing
- **down_histo_mod** (*Dict[str, np.ndarray]*) – "down" variation after post-processing
- **bin_edges** (*np.ndarray*) – bin edges of histogram
- **variable** (*str*) – variable name for the horizontal axis
- **figure_path** (*Optional[pathlib.Path], optional*) – path where figure should be saved, or None to not save it, defaults to None
- **label** (*str, optional*) – label written on the figure, defaults to ""
- **close_figure** (*bool, optional*) – whether to close each figure immediately after saving it, defaults to False (enable when producing many figures to avoid memory issues, prevents rendering in notebooks)

**Returns**
the template figure

**Return type**
matplotlib.figure.Figure

### 5.7.2 cabinetry.visualize.plot_result

Visualizes inference results with matplotlib.

cabinetry.visualize.plot_result.**correlation_matrix**(*corr_mat: ndarray*, *labels: List[str] | ndarray*, ***, *figure_path: Path | None = None*, *close_figure: bool = False*) → Figure

Draws a correlation matrix.

**Parameters**

- **corr_mat** (*np.ndarray*) – the correlation matrix to plot
- **labels** (*Union[List[str], np.ndarray]*) – names of parameters in the correlation matrix
- **figure_path** (*Optional[pathlib.Path], optional*) – path where figure should be saved, or None to not save it, defaults to None

- **close_figure** (*bool, optional*) – whether to close each figure immediately after saving it, defaults to False (enable when producing many figures to avoid memory issues, prevents rendering in notebooks)

**Returns**
the correlation matrix figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.plot_result.**limit**(*observed_CLs: ndarray*, *expected_CLs: ndarray*, *poi_values: ndarray*, *cls_target: float*, ***, *figure_path: Path | None = None*, *close_figure: bool = False*) → Figure

Draws observed and expected CLs values as function of the parameter of interest.

**Parameters**

- **observed_CLs** (*np.ndarray*) – observed CLs values
- **expected_CLs** (*np.ndarray*) – expected CLs values, including 1 and 2 sigma bands
- **poi_values** (*np.ndarray*) – parameter of interest values used in scan
- **cls_target** (*float*) – target CLs value to visualize as horizontal line
- **figure_path** (*Optional[pathlib.Path], optional*) – path where figure should be saved, or None to not save it, defaults to None
- **close_figure** (*bool, optional*) – whether to close each figure immediately after saving it, defaults to False (enable when producing many figures to avoid memory issues, prevents rendering in notebooks)

**Returns**
the CLs figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.plot_result.**pulls**(*bestfit: ndarray*, *uncertainty: ndarray*, *labels: List[str] | ndarray*, ***, *figure_path: Path | None = None*, *close_figure: bool = False*) → Figure

Draws a pull plot.

**Parameters**

- **bestfit** (*np.ndarray*) – best-fit parameter results
- **uncertainty** (*np.ndarray*) – parameter uncertainties
- **labels** (*Union[List[str], np.ndarray]*) – parameter names
- **figure_path** (*Optional[pathlib.Path], optional*) – path where figure should be saved, or None to not save it, defaults to None
- **close_figure** (*bool, optional*) – whether to close each figure immediately after saving it, defaults to False (enable when producing many figures to avoid memory issues, prevents rendering in notebooks)

**Returns**
the pull figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.plot_result.**ranking**(*bestfit: ndarray*, *uncertainty: ndarray*, *labels: List[str] | ndarray*, *impact_prefit_up: ndarray*, *impact_prefit_down: ndarray*, *impact_postfit_up: ndarray*, *impact_postfit_down: ndarray*, ***, *figure_path: Path | None = None*, *close_figure: bool = False*) → Figure

Draws a ranking plot.

**Parameters**

- **bestfit** (*np.ndarray*) – best-fit parameter results
- **uncertainty** (*np.ndarray*) – parameter uncertainties
- **labels** (*Union[List[str], np.ndarray]*) – parameter labels
- **impact_prefit_up** (*np.ndarray*) – pre-fit impact in "up" direction per parameter
- **impact_prefit_down** (*np.ndarray*) – pre-fit impact in "down" direction per parameter
- **impact_postfit_up** (*np.ndarray*) – post-fit impact in "up" direction per parameter
- **impact_postfit_down** (*np.ndarray*) – post-fit impact in "down" direction per parameter
- **figure_path** (*Optional[pathlib.Path], optional*) – path where figure should be saved, or None to not save it, defaults to None
- **close_figure** (*bool, optional*) – whether to close each figure immediately after saving it, defaults to False (enable when producing many figures to avoid memory issues, prevents rendering in notebooks)

**Returns**
the ranking figure

**Return type**
matplotlib.figure.Figure

cabinetry.visualize.plot_result.**scan**(*par_name: str*, *par_mle: float*, *par_unc: float*, *par_vals: ndarray*, *par_nlls: ndarray*, ***, *figure_path: Path | None = None*, *close_figure: bool = False*) → Figure

Draws a figure showing the results of a likelihood scan.

**Parameters**

- **par_name** (*str*) – name of parameter used in scan
- **par_mle** (*float*) – best-fit result for parameter
- **par_unc** (*float*) – best-fit parameter uncertainty
- **par_vals** (*np.ndarray*) – values used in scan over parameter
- **par_nlls** (*np.ndarray*) – -2 log(L) offset at each scan point
- **figure_path** (*Optional[pathlib.Path], optional*) – path where figure should be saved, or None to not save it, defaults to None
- **close_figure** (*bool, optional*) – whether to close each figure immediately after saving it, defaults to False (enable when producing many figures to avoid memory issues, prevents rendering in notebooks)

**Returns**
the likelihood scan figure

**Return type**
matplotlib.figure.Figure

### 5.7.3 cabinetry.visualize.utils

Provides visualization utilities.

## 5.8 cabinetry.tabulate

Creates yield tables.

cabinetry.tabulate.**yields**(*model_prediction: ModelPrediction*, *data: List[float]*, ***, *channels: str | List[str] | None = None*, *per_bin: bool = True*, *per_channel: bool = False*, *table_folder: str | Path = 'tables'*, *table_format: str = 'simple'*, *save_tables: bool = True*) → Dict[str, List[Dict[str, Any]]]

Generates yield tables, showing model prediction and data.

Channels can be filtered via the optional `channels` argument. Either yields per bin, or yields per channel, or both can be shown.

**Parameters**

- **model_prediction** (model_utils.ModelPrediction) – model prediction to show in table
- **data** (*List[float]*) – data to include in table, can either include auxdata (the auxdata is then stripped internally) or only observed yields
- **channels** (*Optional[Union[str, List[str]]], optional*) – name of channel to show, or list of names to include, defaults to None (uses all channels)
- **per_bin** (*bool, optional*) – whether to show a table with yields per bin, defaults to True
- **per_channel** (*bool, optional*) – whether to show a table with yields per channel, defaults to False
- **table_folder** (*Union[str, pathlib.Path], optional*) – path to the folder to save tables in, defaults to "tables"
- **table_format** (*str, optional*) – format in which to save the tables, can be any of the formats `tabulate` supports (e.g. html, latex, plain, simple, tsv), defaults to "simple"
- **save_tables** (*bool, optional*) – whether to save tables, defaults to True

**Returns**
dictionary with yield tables for use with the `tabulate` package

**Return type**
Dict[str, List[Dict[str, Any]]]

## 5.9 cabinetry.model_utils

Provides utilities for pyhf models.

class cabinetry.model_utils.**ModelPrediction**(*model: Model*, *model_yields: List[List[List[float]]]*, *total_stdev_model_bins: List[List[List[float]]]*, *total_stdev_model_channels: List[List[float]]*, *label: str*)

Model prediction with yields and total uncertainties per bin and channel.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to which prediction corresponds to
- **model_yields** (*List[List[List[float]]]*) – yields per sample, channel and bin, indices: channel, sample, bin
- **total_stdev_model_bins** (*List[List[List[float]]]*) – total yield uncertainty per channel, sample and bin, indices: channel, sample, bin (last sample: sum over samples)
- **total_stdev_model_channels** (*List[List[float]]*) – total yield uncertainty per channel and sample, indices: channel, sample (last sample: sum over samples)
- **label** (*str*) – label for the prediction, e.g. "pre-fit" or "post-fit"

**label: str**

Alias for field number 4

**model: Model**

Alias for field number 0

**model_yields: List[List[List[float]]]**

Alias for field number 1

**total_stdev_model_bins: List[List[List[float]]]**

Alias for field number 2

**total_stdev_model_channels: List[List[float]]**

Alias for field number 3

cabinetry.model_utils.**asimov_data**(*model: Model*, ***, *fit_results:* FitResults | *None = None*, *poi_name: str | None = None*, *poi_value: float | None = None*, *include_auxdata: bool = True*) → List[float]

Returns the Asimov dataset (optionally with auxdata) for a model.

Initial parameter settings for normalization factors in the workspace are treated as the default settings for that parameter. Fitting the Asimov dataset will recover these initial settings as the maximum likelihood estimate for normalization factors. Initial settings for other modifiers are ignored. If the fit_results keyword argument is used, the Asimov dataset is built to recover the fit results given when fitted again.

**Parameters**

- **model** (*pyhf.Model*) – the model from which to construct the dataset
- **fit_results** (*Optional[FitResults], optional*) – parameter configuration to use when building the Asimov dataset (using the best-fit result), defaults to None (then a pre-fit Asimov dataset is built)
- **poi_name** (*Optional[str], optional*) – name of parameter to set to a custom value via poi_value, defaults to None (use POI specified in workspace)
- **poi_value** (*Optional[float], optional*) – custom value to set POI specified via poi_name to, defaults to None (no custom value set)
- **include_auxdata** (*bool, optional*) – whether to also return auxdata, defaults to True

**Returns**
the Asimov dataset

**Return type**
List[float]

cabinetry.model_utils.**asimov_parameters**(*model: Model*) → ndarray

Returns a list of Asimov parameter values for a model.

For normfactors and shapefactors, initial parameter settings (specified in the workspace) are treated as nominal settings. This ignores custom auxiliary data set in the measurement configuration in the workspace.

**Parameters**
**model** (*pyhf.pdf.Model*) – model for which to extract the parameters

**Returns**
the Asimov parameters, in the same order as `model.config.suggested_init()`

**Return type**
np.ndarray

cabinetry.model_utils.**match_fit_results**(*model: Model*, *fit_results:* FitResults) → *FitResults*

Matches results from a fit to a model by adding or removing parameters as needed.

If the fit results contain parameters missing in the model, these parameters are not included in the returned fit results. If the fit results do not include parameters used in the model, they are added to the fit results. The best-fit value for such parameters are the Asimov values as returned by `asimov_parameters` (initial parameter settings for unconstrained parameters), and the associated uncertainties as given by `prefit_uncertainties` (zero uncertainty for unconstrained or fixed parameters). These parameters furthermore are assumed to have no correlation with any other parameters. If required, parameters are re-ordered to match the target model.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to match fit results to
- **fit_results** (FitResults) – fit results to be updated in order to match model

**Returns**
fit results matching the model

**Return type**
*FitResults*

cabinetry.model_utils.**model_and_data**(*spec: Dict[str, Any]*, ***, *asimov: bool = False*, *include_auxdata: bool = True*) → Tuple[Model, List[float]]

Returns model and data for a `pyhf` workspace specification.

**Parameters**

- **spec** (*Dict[str, Any]*) – a `pyhf` workspace specification
- **asimov** (*bool, optional*) – whether to return the Asimov dataset, defaults to False
- **include_auxdata** (*bool, optional*) – whether to also return auxdata, defaults to True

**Returns**

- a HistFactory-style model in `pyhf` format
- the data (plus auxdata if requested) for the model

**Return type**
Tuple[pyhf.pdf.Model, List[float]]

cabinetry.model_utils.**prediction**(*model: Model*, ***, *fit_results:* FitResults | *None = None*, *label: str* | *None = None*) → *ModelPrediction*

Returns model prediction, including model yields and uncertainties.

If the optional fit result is not provided, the pre-fit Asimov yields and uncertainties are calculated. If the fit result is provided, the best-fit parameter values, uncertainties, and the parameter correlations are used to obtain the post- fit model and its associated uncertainties.

**Parameters**

- **model** (*pyhf.pdf.Model*) – model to evaluate yield prediction for
- **fit_results** (*Optional[FitResults], optional*) – parameter configuration to use, includes best-fit settings and uncertainties, as well as correlation matrix, defaults to None (then the pre-fit configuration is used)
- **label** (*Optional[str], optional*) – label to include in model prediction, defaults to None (then will use "pre-fit" if fit results are not included, and "post- fit" otherwise)

**Returns**
model, yields and uncertainties per bin and channel

**Return type**
*ModelPrediction*

cabinetry.model_utils.**prefit_uncertainties**(*model: Model*) → ndarray

Returns a list of pre-fit parameter uncertainties for a model.

For unconstrained parameters the uncertainty is set to 0. It is also set to 0 for fixed parameters (similarly to how the post-fit uncertainties are defined to be 0).

**Parameters**
**model** (*pyhf.pdf.Model*) – model for which to extract the parameters

**Returns**
pre-fit uncertainties for the parameters, in the same order as `model.config.suggested_init()`

**Return type**
np.ndarray

cabinetry.model_utils.**unconstrained_parameter_count**(*model: Model*) → int

Returns the number of unconstrained parameters in a model.

The number is the sum of all independent parameters in a fit. A shapefactor that affects multiple bins enters the count once for each independent bin. Parameters that are set to constant are not included in the count.

**Parameters**
**model** (*pyhf.pdf.Model*) – model to count parameters for

**Returns**
number of unconstrained parameters

**Return type**
int

cabinetry.model_utils.**yield_stdev**(*model: Model*, *parameters: ndarray*, *uncertainty: ndarray*, *corr_mat: ndarray*) → Tuple[List[List[List[float]]], List[List[float]]]

Calculates symmetrized model yield standard deviation per channel / sample / bin.

Returns both the uncertainties per bin (in a list of channels and samples), and the uncertainty of the total yield per channel (again, for a list of channels and samples). To calculate the uncertainties for the total yield per channel, the function internally treats the sum of yields per channel like another channel with one bin. Similarly, the sum over samples is treated as another sample. The results of this function are cached to speed up subsequent calls with the same arguments.

**Parameters**

- **model** (*pyhf.pdf.Model*) – the model for which to calculate the standard deviations for all bins
- **parameters** (*np.ndarray*) – central values of model parameters
- **uncertainty** (*np.ndarray*) – uncertainty of model parameters
- **corr_mat** (*np.ndarray*) – correlation matrix

**Returns**

- list of channels, each channel is a list of samples, and each sample a list of standard deviations per bin (the last sample corresponds to a sum over all samples)
- list of standard deviations per channel, each channel is a list containing the standard deviations per sample (the last sample corresponds to a sum over all samples)

**Return type**
Tuple[List[List[List[float]]], List[List[float]]]

## 5.10 cabinetry.smooth

Implements histogram smoothing algorithms.

cabinetry.smooth.**smooth_353qh_twice**(*hist: T*) → T

Runs the 353QH algorithm twice and returns smooth version of the input.

For documentation see these proceedings https://cds.cern.ch/record/186223/ on page 292. The algorithm runs twice to avoid over-smoothing peaks and valleys. The algorithm is not aware of statistical uncertainties per entry in the array. See also https://root.cern.ch/doc/master/classTH1.html#aeb935cae10dbf9cd484bee1b6a549f83 for the ROOT implementation.

**Parameters**
**hist** (*Union[list, np.ndarray]*) – array to smooth

**Returns**
smooth version of input

**Return type**
Union[list, np.ndarray]

## 5.11 cabinetry.contrib

Basic implementations of functionality that can be provided by external tools.

### 5.11.1 cabinetry.contrib.histogram_creator

Creates histograms from ntuples with uproot.

cabinetry.contrib.histogram_creator.**with_uproot**(*ntuple_paths: List[Path]*, *pos_in_file: str*, *variable: str*, *bins: ndarray*, ***, *weight: str | None = None*, *selection_filter: str | None = None*) → Histogram

Reads an ntuple with uproot, fills and returns a histogram with the observable.

The paths may contain wildcards.

**Parameters**

- **ntuple_paths** (*List[pathlib.Path]*) – list of paths to ntuples
- **pos_in_file** (*str*) – name of tree within ntuple
- **variable** (*str*) – variable to bin histogram in
- **bins** (*np.ndarray*) – bin edges for histogram
- **weight** (*Optional[str], optional*) – event weight to extract, defaults to None (no weights applied)
- **selection_filter** (*Optional[str], optional*) – filter to be applied on events, defaults to None (no filter)

**Returns**
histogram containing data

**Return type**
bh.Histogram

### 5.11.2 cabinetry.contrib.histogram_reader

Reads histograms with uproot.

cabinetry.contrib.histogram_reader.**with_uproot**(*histo_path: str*) → Histogram

Reads a histogram with uproot and returns it.

**Parameters**
**histo_path** (*str*) – path to histogram, use a colon to distinguish between path to file and path to histogram within file (example: `file.root:h1`)

**Returns**
histogram containing data

**Return type**
bh.Histogram

CHAPTER

SIX

# LICENSE



`cabinetry` is a Python library for building and steering binned template fits. It interfaces many other libraries to make it easy for an analyzer to run their statistical inference pipeline.

This documentation can be viewed on Read the Docs or locally via

```
sphinx-build docs docs/_build
open docs/_build/index.html
```

See the README on github for a simple example of using `cabinetry`, and the cabinetry-tutorials repository for a more in-depth version.

CHAPTER

SEVEN

# PRESENTATIONS OF CABINETRY

- vCHEP 2021
  - short talk: https://indico.cern.ch/event/948465/contributions/4324154/
  - associated paper: 10.1051/epjconf/202125103067
- PyHEP 2021
  - notebook talk: https://indico.cern.ch/event/1019958/contributions/4421868/

CHAPTER

EIGHT

# ACKNOWLEDGEMENTS

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

# PYTHON MODULE INDEX

## C

cabinetry.configuration, 30
cabinetry.contrib, 60
cabinetry.contrib.histogram_creator, 60
cabinetry.contrib.histogram_reader, 61
cabinetry.fit, 40
cabinetry.fit.results_containers, 44
cabinetry.histo, 33
cabinetry.model_utils, 56
cabinetry.route, 32
cabinetry.smooth, 60
cabinetry.tabulate, 56
cabinetry.templates, 36
cabinetry.templates.builder, 37
cabinetry.templates.collector, 37
cabinetry.templates.postprocessor, 37
cabinetry.templates.utils, 37
cabinetry.visualize, 48
cabinetry.visualize.plot_model, 52
cabinetry.visualize.plot_result, 53
cabinetry.visualize.utils, 56
cabinetry.workspace, 37

# INDEX

## Symbols


--asimov
    cabinetry-fit command line option, 22
    cabinetry-limit command line option, 23
    cabinetry-ranking command line option, 24
    cabinetry-scan command line option, 25
    cabinetry-significance command line option, 26
--cl
    cabinetry-limit command line option, 23
--confidence_level
    cabinetry-limit command line option, 23
--config
    cabinetry-data-mc command line option, 22
--corrmat
    cabinetry-fit command line option, 22
--figfolder
    cabinetry-data-mc command line option, 22
    cabinetry-fit command line option, 23
    cabinetry-limit command line option, 23
    cabinetry-modifier-grid command line option, 24
    cabinetry-ranking command line option, 24
    cabinetry-scan command line option, 25
--goodness_of_fit
    cabinetry-fit command line option, 22
--lower_bound
    cabinetry-scan command line option, 25
--max_pars
    cabinetry-ranking command line option, 24
--method
    cabinetry-templates command line option, 26
--minos
    cabinetry-fit command line option, 22
--n_steps
    cabinetry-scan command line option, 25
--postfit
    cabinetry-data-mc command line option, 22
    cabinetry-yields command line option, 27
--pulls
    cabinetry-fit command line option, 22
--split_by_sample
    cabinetry-modifier-grid command line option, 24
--tablefmt
    cabinetry-yields command line option, 27
--tablefolder
    cabinetry-yields command line option, 27
--tolerance
    cabinetry-limit command line option, 23
--upper_bound
    cabinetry-scan command line option, 25
--version
    cabinetry command line option, 22


## A


apply_postprocessing() (*in module cabinetry.templates.postprocessor*), 37
apply_to_all_templates() (*in module cabinetry.route*), 33
asimov_data() (*in module cabinetry.model_utils*), 57
asimov_parameters() (*in module cabinetry.model_utils*), 57


## B


best_twice_nll (*cabinetry.fit.results_containers.FitResults attribute*), 45
bestfit (*cabinetry.fit.results_containers.FitResults attribute*), 45
bestfit (*cabinetry.fit.results_containers.RankingResults attribute*), 46
bestfit (*cabinetry.fit.results_containers.ScanResults attribute*), 47
bins (*cabinetry.histo.Histogram property*), 33
build() (*cabinetry.workspace.WorkspaceBuilder method*), 37
build() (*in module cabinetry.templates*), 36
build() (*in module cabinetry.workspace*), 39


## C


cabinetry command line option
    --version, 22

cabinetry.configuration
    module, 30
cabinetry.contrib
    module, 60
cabinetry.contrib.histogram_creator
    module, 60
cabinetry.contrib.histogram_reader
    module, 61
cabinetry.fit
    module, 40
cabinetry.fit.results_containers
    module, 44
cabinetry.histo
    module, 33
cabinetry.model_utils
    module, 56
cabinetry.route
    module, 32
cabinetry.smooth
    module, 60
cabinetry.tabulate
    module, 56
cabinetry.templates
    module, 36
cabinetry.templates.builder
    module, 37
cabinetry.templates.collector
    module, 37
cabinetry.templates.postprocessor
    module, 37
cabinetry.templates.utils
    module, 37
cabinetry.visualize
    module, 48
cabinetry.visualize.plot_model
    module, 52
cabinetry.visualize.plot_result
    module, 53
cabinetry.visualize.utils
    module, 56
cabinetry.workspace
    module, 37
cabinetry-data-mc command line option
    --config, 22
    --figfolder, 22
    --postfit, 22
    WS_SPEC, 22
cabinetry-fit command line option
    --asimov, 22
    --corrmat, 22
    --figfolder, 23
    --goodness_of_fit, 22
    --minos, 22
    --pulls, 22
    WS_SPEC, 23
cabinetry-limit command line option
    --asimov, 23
    --cl, 23
    --confidence_level, 23
    --figfolder, 23
    --tolerance, 23
    WS_SPEC, 23
cabinetry-modifier-grid command line option
    --figfolder, 24
    --split_by_sample, 24
    WS_SPEC, 24
cabinetry-postprocess command line option
    CONFIG, 24
cabinetry-ranking command line option
    --asimov, 24
    --figfolder, 24
    --max_pars, 24
    WS_SPEC, 25
cabinetry-scan command line option
    --asimov, 25
    --figfolder, 25
    --lower_bound, 25
    --n_steps, 25
    --upper_bound, 25
    PAR_NAME, 25
    WS_SPEC, 25
cabinetry-significance command line option
    --asimov, 26
    WS_SPEC, 26
cabinetry-templates command line option
    --method, 26
    CONFIG, 26
cabinetry-workspace command line option
    CONFIG, 26
    WS_SPEC, 26
cabinetry-yields command line option
    --postfit, 27
    --tablefmt, 27
    --tablefolder, 27
    WS_SPEC, 27
channels() (*cabinetry.workspace.WorkspaceBuilder method*), 38
collect() (*in module cabinetry.templates*), 36
confidence_level (*cabinetry.fit.results_containers.LimitResults attribute*), 45
CONFIG
    cabinetry-postprocess command line option, 24
    cabinetry-templates command line option, 26
    cabinetry-workspace command line option, 26

corr_mat (*cabinetry.fit.results_containers.FitResults attribute*), 45
correlation_matrix() (*in module cabinetry.visualize*), 48
correlation_matrix() (*in module cabinetry.visualize.plot_result*), 53

## D

data_mc() (*in module cabinetry.visualize*), 48
data_mc() (*in module cabinetry.visualize.plot_model*), 52
data_mc_from_histograms() (*in module cabinetry.visualize*), 49
delta_nlls (*cabinetry.fit.results_containers.ScanResults attribute*), 47

## E

expected_CLs (*cabinetry.fit.results_containers.LimitResults attribute*), 45
expected_limit (*cabinetry.fit.results_containers.LimitResults attribute*), 45
expected_p_value (*cabinetry.fit.results_containers.SignificanceResults attribute*), 47
expected_significance (*cabinetry.fit.results_containers.SignificanceResults attribute*), 47

## F

fit() (*in module cabinetry.fit*), 40
FitResults (*class in cabinetry.fit.results_containers*), 44
from_arrays() (*cabinetry.histo.Histogram class method*), 33
from_config() (*cabinetry.histo.Histogram class method*), 34
from_path() (*cabinetry.histo.Histogram class method*), 34

## G

goodness_of_fit (*cabinetry.fit.results_containers.FitResults attribute*), 45

## H

Histogram (*class in cabinetry.histo*), 33
histogram_is_needed() (*in module cabinetry.configuration*), 30

## L

label (*cabinetry.model_utils.ModelPrediction attribute*), 57
labels (*cabinetry.fit.results_containers.FitResults attribute*), 45
labels (*cabinetry.fit.results_containers.RankingResults attribute*), 46
limit() (*in module cabinetry.fit*), 41
limit() (*in module cabinetry.visualize*), 49
limit() (*in module cabinetry.visualize.plot_result*), 54
LimitResults (*class in cabinetry.fit.results_containers*), 45
load() (*in module cabinetry.configuration*), 30
load() (*in module cabinetry.workspace*), 39

## M

match_fit_results() (*in module cabinetry.model_utils*), 58
measurements() (*cabinetry.workspace.WorkspaceBuilder method*), 38
minos_uncertainty (*cabinetry.fit.results_containers.FitResults attribute*), 45
model (*cabinetry.model_utils.ModelPrediction attribute*), 57
model_and_data() (*in module cabinetry.model_utils*), 58
model_yields (*cabinetry.model_utils.ModelPrediction attribute*), 57
ModelPrediction (*class in cabinetry.model_utils*), 56
modifier_grid() (*in module cabinetry.visualize*), 50
modifier_grid() (*in module cabinetry.visualize.plot_model*), 52
module
    cabinetry.configuration, 30
    cabinetry.contrib, 60
    cabinetry.contrib.histogram_creator, 60
    cabinetry.contrib.histogram_reader, 61
    cabinetry.fit, 40
    cabinetry.fit.results_containers, 44
    cabinetry.histo, 33
    cabinetry.model_utils, 56
    cabinetry.route, 32
    cabinetry.smooth, 60
    cabinetry.tabulate, 56
    cabinetry.templates, 36
    cabinetry.templates.builder, 37
    cabinetry.templates.collector, 37
    cabinetry.templates.postprocessor, 37
    cabinetry.templates.utils, 37
    cabinetry.visualize, 48
    cabinetry.visualize.plot_model, 52
    cabinetry.visualize.plot_result, 53
    cabinetry.visualize.utils, 56
    cabinetry.workspace, 37

## N

name (*cabinetry.fit.results_containers.ScanResults attribute*), 47
name() (*in module cabinetry.histo*), 35
normalization_modifier() (*cabinetry.workspace.WorkspaceBuilder static method*), 38
normalize_to_yield() (*cabinetry.histo.Histogram method*), 35
normfactor_modifiers() (*cabinetry.workspace.WorkspaceBuilder method*), 38
normplusshape_modifiers() (*cabinetry.workspace.WorkspaceBuilder method*), 38

## O

observations() (*cabinetry.workspace.WorkspaceBuilder method*), 39
observed_CLs (*cabinetry.fit.results_containers.LimitResults attribute*), 46
observed_limit (*cabinetry.fit.results_containers.LimitResults attribute*), 46
observed_p_value (*cabinetry.fit.results_containers.SignificanceResults attribute*), 47
observed_significance (*cabinetry.fit.results_containers.SignificanceResults attribute*), 47

## P

PAR_NAME
    cabinetry-scan command line option, 25
parameter_values (*cabinetry.fit.results_containers.ScanResults attribute*), 47
poi_values (*cabinetry.fit.results_containers.LimitResults attribute*), 46
postfit_down (*cabinetry.fit.results_containers.RankingResults attribute*), 46
postfit_up (*cabinetry.fit.results_containers.RankingResults attribute*), 46
postprocess() (*in module cabinetry.templates*), 36
prediction() (*in module cabinetry.model_utils*), 58
prefit_down (*cabinetry.fit.results_containers.RankingResults attribute*), 46
prefit_uncertainties() (*in module cabinetry.model_utils*), 59
prefit_up (*cabinetry.fit.results_containers.RankingResults attribute*), 46
print_overview() (*in module cabinetry.configuration*), 30
print_results() (*in module cabinetry.fit*), 42
pulls() (*in module cabinetry.visualize*), 50
pulls() (*in module cabinetry.visualize.plot_result*), 54

## R

ranking() (*in module cabinetry.fit*), 42
ranking() (*in module cabinetry.visualize*), 50
ranking() (*in module cabinetry.visualize.plot_result*), 54
RankingResults (*class in cabinetry.fit.results_containers*), 46
region_contains_modifier() (*in module cabinetry.configuration*), 30
region_contains_sample() (*in module cabinetry.configuration*), 31
region_dict() (*in module cabinetry.configuration*), 31
register_template_builder() (*cabinetry.route.Router method*), 32
Router (*class in cabinetry.route*), 32

## S

sample_contains_modifier() (*in module cabinetry.configuration*), 31
save() (*cabinetry.histo.Histogram method*), 35
save() (*in module cabinetry.workspace*), 40
scan() (*in module cabinetry.fit*), 43
scan() (*in module cabinetry.visualize*), 51
scan() (*in module cabinetry.visualize.plot_result*), 55
ScanResults (*class in cabinetry.fit.results_containers*), 46
significance() (*in module cabinetry.fit*), 44
SignificanceResults (*class in cabinetry.fit.results_containers*), 47
smooth_353qh_twice() (*in module cabinetry.smooth*), 60
stdev (*cabinetry.histo.Histogram property*), 35
sys_modifiers() (*cabinetry.workspace.WorkspaceBuilder method*), 39

## T

template_builder_wrapper (*cabinetry.route.Router attribute*), 32
template_builders (*cabinetry.route.Router attribute*), 32
templates() (*in module cabinetry.visualize*), 51
templates() (*in module cabinetry.visualize.plot_model*), 53
total_stdev_model_bins (*cabinetry.model_utils.ModelPrediction attribute*), 57
total_stdev_model_channels (*cabinetry.model_utils.ModelPrediction attribute*), 57

## U

uncertainty (*cabinetry.fit.results_containers.FitResults attribute*), 45
uncertainty (*cabinetry.fit.results_containers.RankingResults attribute*), 46
uncertainty (*cabinetry.fit.results_containers.ScanResults attribute*), 47
unconstrained_parameter_count() (*in module cabinetry.model_utils*), 59

## V

validate() (*cabinetry.histo.Histogram method*), 35
validate() (*in module cabinetry.configuration*), 31
validate() (*in module cabinetry.workspace*), 40

## W

with_uproot() (*in module cabinetry.contrib.histogram_creator*), 60
with_uproot() (*in module cabinetry.contrib.histogram_reader*), 61
WorkspaceBuilder (*class in cabinetry.workspace*), 37
WS_SPEC
    cabinetry-data-mc command line option, 22
    cabinetry-fit command line option, 23
    cabinetry-limit command line option, 23
    cabinetry-modifier-grid command line option, 24
    cabinetry-ranking command line option, 25
    cabinetry-scan command line option, 25
    cabinetry-significance command line option, 26
    cabinetry-workspace command line option, 26
    cabinetry-yields command line option, 27

## Y

yield_stdev() (*in module cabinetry.model_utils*), 59
yields (*cabinetry.histo.Histogram property*), 35
yields() (*in module cabinetry.tabulate*), 56