mpl-hep

- Our niche: plotting pre-binned histograms
  - Apparently no one else does this regularly
  - Discussed a bit in matplotlib issue 6669
  - Helper functions?
  - Gallery of examples?
- Packaging of the above
  - mpl developers amenable to hosting a ‘mpl-hep’ package
  - Third-party packages are a possible option
- Previous discussion
  - google doc including references to some existing libraries
  - gitter

Helper function signature

- Pass mpl Axes object and some data
- Communicate all styling by dict(s)
- Return all objects added to Axes

- Per mpl coding styles

```
def my_plotter(ax, data1, data2, param_dict):
    """
    A helper function to make a graph

    Parameters
    ----------
    ax : Axes
        The axes to draw to

    data1 : array
       The x data

    data2 : array
       The y data

    param_dict : dict
       Dictionary of kwargs to pass to ax.plot

    Returns
    -------
    out : list
        list of artists added
    """
    out = ax.plot(data1, data2, **param_dict)
    return out
```

Helper function signature

- My attempt at a general 1D plotting signature
  - Inside fnal_column_analysis_tools
  - Intertwined with the fcat Hist object

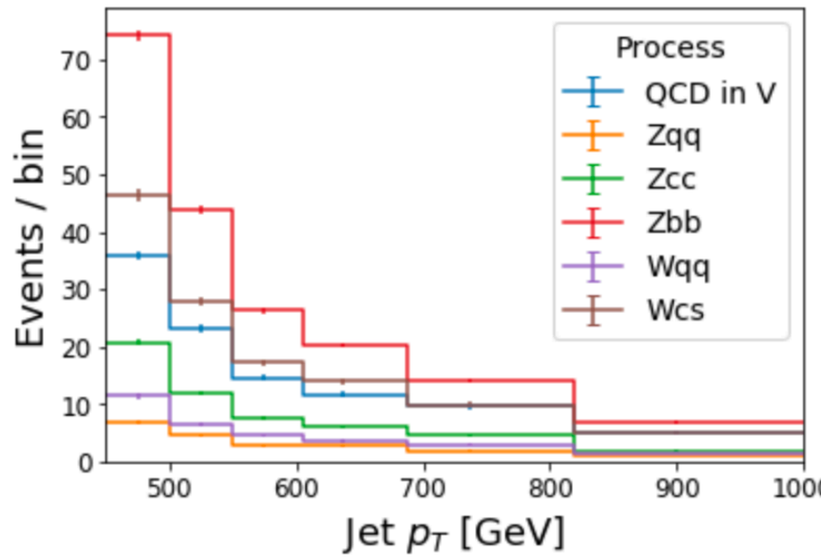
```
def plot1d(hist, ax=None, clear=True, overlay=None, stack=False, overflow='none', line_opts=None, fill_opts=None, error_opts=None,
overlay_overflow='none', density=False, binwnorm=None):
    """
        hist: Hist object with maximum of two dimensions
        ax: matplotlib Axes object (if None, one is created)
        clear: clear Axes before drawing (if passed); if False, this function will skip drawing the legend
        overlay: the axis of hist to overlay (remaining one will be x axis)
        stack: whether to stack or overlay the other dimension (if one exists)
        overflow: overflow behavior of plot axis (see Hist.sum() docs)

        The draw options are passed as dicts to the relevant matplotlib function, with some exceptions in case
        it is especially common or useful.  If none of *_opts is specified, nothing will be plotted!
        Pass an empty dict (e.g. line_opts={}) for defaults
            line_opts: options to plot a step without errors
                Special options interpreted by this function and not passed to matplotlib:
                    (none)

            fill_opts: to plot a filled area
                Special options interpreted by this function and not passed to matplotlib:
                    (none)

            error_opts: to plot an errorbar, with a step or marker
                Special options interpreted by this function and not passed to matplotlib:
                    'emarker' (default: '') marker to place at cap of errorbar


        overlay_overflow: overflow behavior of dense overlay axis, if one exists
        density: Convert sum weights to probability density (i.e. integrates to 1 over domain of axis) (NB: conflicts with binwnorm)
        binwnorm: Convert sum weights to bin-width-normalized, with units equal to supplied value (usually you want to specify 1.)
    """
```

Helper function signature

- The resulting API
- Is this in scope for mpl-hep?
  - Probably not
- How do we provide an API without becoming intertwined with the histogram filling and transformation library itself?
  - Maybe we just *don't*

- p.s. can you spot the bug in the figure?

```
to_plot = (a_histogram.project("doubleB", slice(0.8, None))
                      .project("msd")
          )
fig, ax, items = plot.plot1d(to_plot, overlay="process", error_opts={})
```



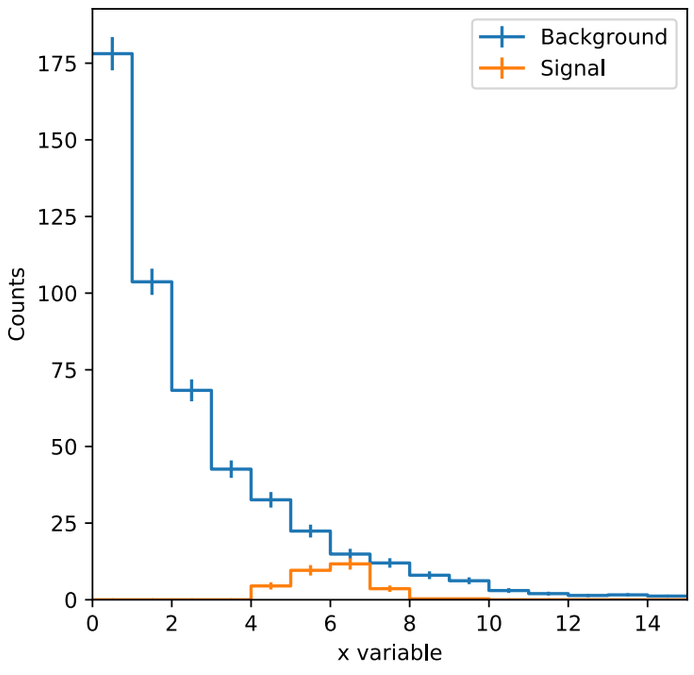

```
items
```

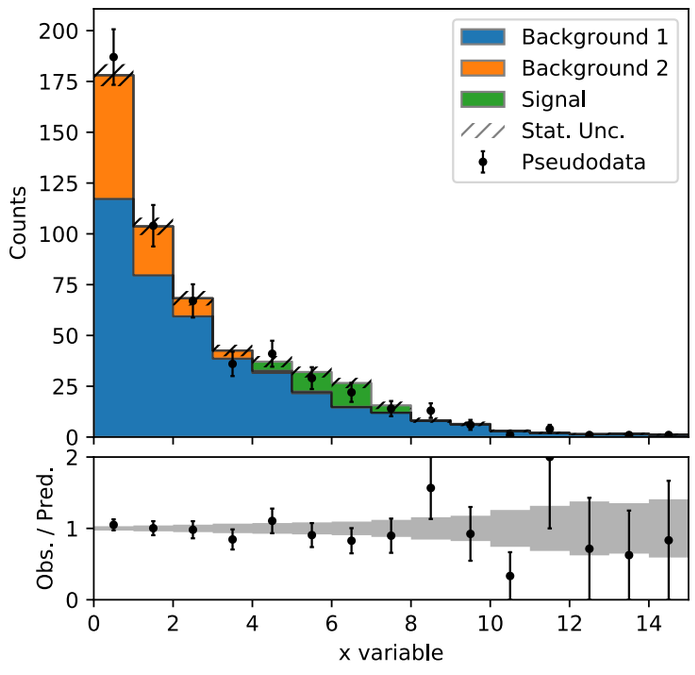
```
{'QCD in V': [(<ErrorbarContainer object of 3 artists>,
   <ErrorbarContainer object of 3 artists>)],
 'Zqq': [(<ErrorbarContainer object of 3 artists>,
   <ErrorbarContainer object of 3 artists>)],
 'Zcc': [(<ErrorbarContainer object of 3 artists>,
   <ErrorbarContainer object of 3 artists>)],
 'Zbb': [(<ErrorbarContainer object of 3 artists>,
   <ErrorbarContainer object of 3 artists>)],
 'Wqq': [(<ErrorbarContainer object of 3 artists>,
   <ErrorbarContainer object of 3 artists>)],
 'Wcs': [(<ErrorbarContainer object of 3 artists>,
   <ErrorbarContainer object of 3 artists>)],
 'legend': <matplotlib.legend.Legend at 0x7f291d5fe978>}
```

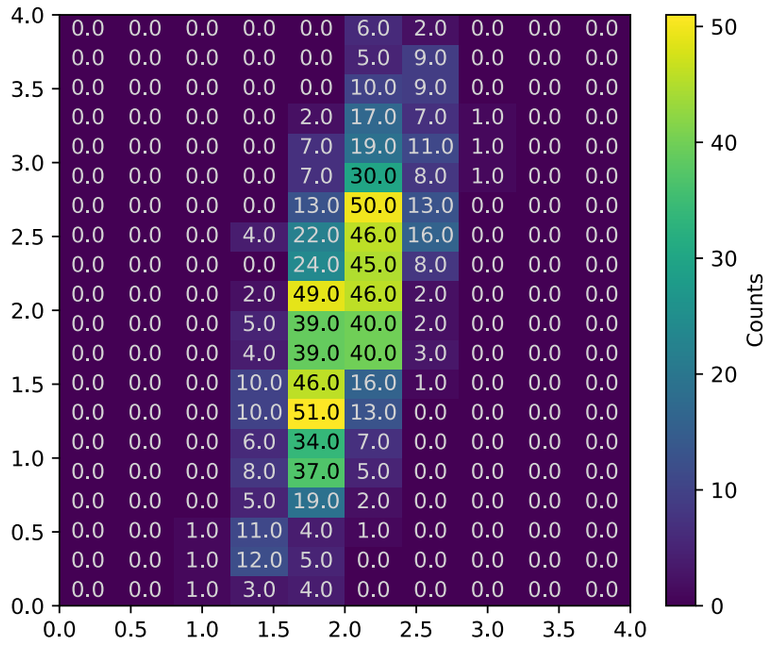

Fermilab

Gallery concept

- mpl is a library of plotting primitives
- mpl leaves users to fend for themselves
- mpl provides a gallery of example code
- Let's try that:
  - https://mybinder.org/v2/gh/nsmith-/mpl-hep/master?filepath=binder/gallery.ipynb
  - Please come contribute your nice HEP plots (or play code golf w/existing)
  - We probably need to learn how to use reStructured text + sphinx like mpl







Possible improvements

- Ask for (or make PR implementing) `step='edges'`

```
line1, = ax.step(x=bin_edges,
                 y=np.hstack([sum_bkg, sum_bkg[-1]]),
                 where='post'
                )
```

```
line1, = ax.step(x=bin_edges,
                 y=sum_bkg,
                 where='edges'
                )
```

Other ideas

- HEP styles
  - Recreate typical styles via mpl style sheets mechanism?
  - Largely experiment-specific (e.g. CMS ‘tdrStyle.C’)
- Many users experienced with ROOT
  - mpl-hep could provide a quickstart guide, translating terminology
  - Rosetta stone, e.g.

Matplotlib	*ROOT*
Figure	**TCanvas**
Axes	**TPad**
fig, axes = plt.subplots(n, m) **axes[i, j].plot(x)**	**c1.Divide(n, m)** **c1.cd(i)** **x.Draw()**
…	**…**

Summary

- Matplotlib can plot pre-binned data
  - With some workarounds
- mpl-hep exists to help bridge this gap
  - Please come contribute any mpl idioms you find useful for HEP!
- Unclear what level of API is needed
  - Most existing implementations leverage knowledge of histogram object, data model and presentation not well separated
- Concrete plan:
  - Upstream step='edges'
  - Continue to compile a gallery
  - Identify common patterns, abstract them into an API
  - Work on rosetta stone, styles