In [1]:
```
import uproot
import bokeh
from bokeh.plotting import show
bokeh.io.output_notebook()
```

Out[1]:


Out[1]:

Look around to see what's available

This is a freemium service: they block off network access, so I uploaded the git repo manually. We'll look at the (small) sample files in the tests.

In [2]:
```
!ls tests/samples
```

```
HZZ-lz4.root                       sample-5.25.02-zlib.root
HZZ-lzma.root                      sample-5.26.00-uncompressed.root
HZZ-uncompressed.root              sample-5.26.00-zlib.root
HZZ-zlib.root                      sample-5.27.02-uncompressed.root
HZZ.root                           sample-5.27.02-zlib.root
Zmumu-lz4.root                     sample-5.28.00-uncompressed.root
Zmumu-lzma.root                    sample-5.28.00-zlib.root
Zmumu-uncompressed.root            sample-5.29.02-uncompressed.root
Zmumu-zlib.root                    sample-5.29.02-zlib.root
Zmumu.root                         sample-5.30.00-lzma.root
foriter.root                       sample-5.30.00-uncompressed.root
foriter2.root                      sample-5.30.00-zlib.root
issue21.root                       sample-6.08.04-lzma.root
issue30.root                       sample-6.08.04-uncompressed.root
issue31.root                       sample-6.08.04-zlib.root
mc10events.root                    sample-6.10.05-lz4.root
nesteddirs.root                    sample-6.10.05-lzma.root
sample-5.23.02-uncompressed.root   sample-6.10.05-uncompressed.root
sample-5.23.02-zlib.root           sample-6.10.05-zlib.root
sample-5.24.00-uncompressed.root   simple.root
sample-5.24.00-zlib.root           small-evnt-tree-fullsplit.root
sample-5.25.02-uncompressed.root   small-flat-tree.root
```

In [3]:
```
uproot.open("tests/samples/Zmumu.root").keys()
```

Out[3]:
```
[b'events;1']
```

In [4]:
```
Zmumu = uproot.open("tests/samples/Zmumu.root")["events"]
```

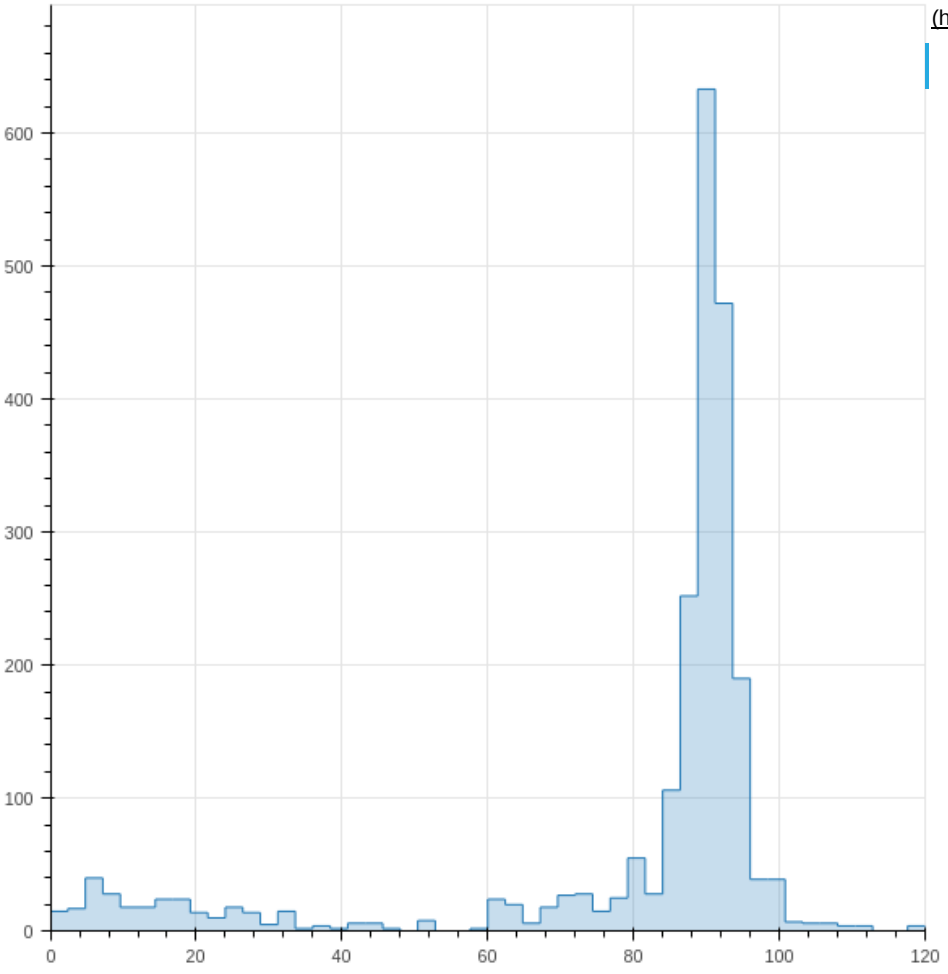
In [5]:
```
Zmumu.show()
```

```
Type                        (no streamer)              <uproot.interp.strings.asstrings obj
ect at 0x7fef5f2ed5c0>
Run                         (no streamer)              asdtype('>i4')
Event                       (no streamer)              asdtype('>i4')
E1                          (no streamer)              asdtype('>f8')
px1                         (no streamer)              asdtype('>f8')
py1                         (no streamer)              asdtype('>f8')
pz1                         (no streamer)              asdtype('>f8')
pt1                         (no streamer)              asdtype('>f8')
eta1                        (no streamer)              asdtype('>f8')
phi1                        (no streamer)              asdtype('>f8')
Q1                          (no streamer)              asdtype('>i4')
E2                          (no streamer)              asdtype('>f8')
px2                         (no streamer)              asdtype('>f8')
py2                         (no streamer)              asdtype('>f8')
pz2                         (no streamer)              asdtype('>f8')
pt2                         (no streamer)              asdtype('>f8')
eta2                        (no streamer)              asdtype('>f8')
phi2                        (no streamer)              asdtype('>f8')
Q2                          (no streamer)              asdtype('>i4')
M                           (no streamer)              asdtype('>f8')
```

Go straight from TTree to plot

In [6]:
```
show(Zmumu.hist(50, 0, 120, "M").bokeh.fig())
```

Out[6]:



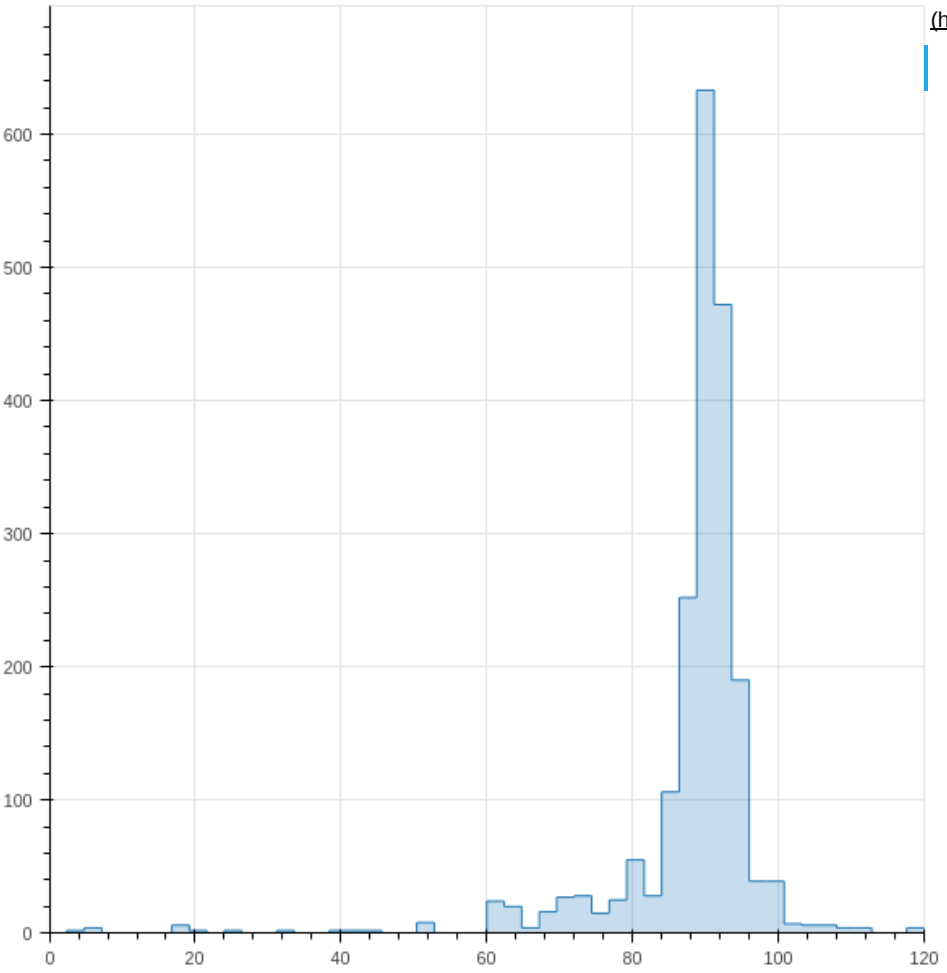
Out[6]:

Do the functional chain thing

I need to think of some better examples than this silly filter-and-plot!

In [7]:
```
show(Zmumu.filter("E1 > 10 and E2 > 10").hist(50, 0, 120, "M").bokeh.fig())
```

Out[7]:



Out[7]:

This Higgz to ZZ file has some jagged arrays in it

See Wikipedia (https://en.wikipedia.org/wiki/Jagged_array) for more about jagged arrays.

In [8]:
```
uproot.open("tests/samples/HZZ.root").keys()
```

Out[8]:
```
[b'events;1']
```

In [9]:
```
HZZ = uproot.open("tests/samples/HZZ.root")["events"]
```

In [10]:
```
HZZ.show()
```

```
NJet                        (no streamer)              asdtype('>i4')
Jet_Px                      (no streamer)              asjagged(asdtype('>f4'))
Jet_Py                      (no streamer)              asjagged(asdtype('>f4'))
Jet_Pz                      (no streamer)              asjagged(asdtype('>f4'))
Jet_E                       (no streamer)              asjagged(asdtype('>f4'))
Jet_btag                    (no streamer)              asjagged(asdtype('>f4'))
Jet_ID                      (no streamer)              asjagged(asdtype(dtype('bool')))
NMuon                       (no streamer)              asdtype('>i4')
Muon_Px                     (no streamer)              asjagged(asdtype('>f4'))
Muon_Py                     (no streamer)              asjagged(asdtype('>f4'))
Muon_Pz                     (no streamer)              asjagged(asdtype('>f4'))
Muon_E                      (no streamer)              asjagged(asdtype('>f4'))
Muon_Charge                 (no streamer)              asjagged(asdtype('>i4'))
Muon_Iso                    (no streamer)              asjagged(asdtype('>f4'))
NElectron                   (no streamer)              asdtype('>i4')
Electron_Px                 (no streamer)              asjagged(asdtype('>f4'))
Electron_Py                 (no streamer)              asjagged(asdtype('>f4'))
Electron_Pz                 (no streamer)              asjagged(asdtype('>f4'))
Electron_E                  (no streamer)              asjagged(asdtype('>f4'))
Electron_Charge             (no streamer)              asjagged(asdtype('>i4'))
Electron_Iso                (no streamer)              asjagged(asdtype('>f4'))
NPhoton                     (no streamer)              asdtype('>i4')
Photon_Px                   (no streamer)              asjagged(asdtype('>f4'))
Photon_Py                   (no streamer)              asjagged(asdtype('>f4'))
Photon_Pz                   (no streamer)              asjagged(asdtype('>f4'))
Photon_E                    (no streamer)              asjagged(asdtype('>f4'))
Photon_Iso                  (no streamer)              asjagged(asdtype('>f4'))
MET_px                      (no streamer)              asdtype('>f4')
MET_py                      (no streamer)              asdtype('>f4')
MChadronicBottom_px         (no streamer)              asdtype('>f4')
MChadronicBottom_py         (no streamer)              asdtype('>f4')
MChadronicBottom_pz         (no streamer)              asdtype('>f4')
MCleptonicBottom_px         (no streamer)              asdtype('>f4')
MCleptonicBottom_py         (no streamer)              asdtype('>f4')
MCleptonicBottom_pz         (no streamer)              asdtype('>f4')
MChadronicWDecayQuark_px    (no streamer)              asdtype('>f4')
MChadronicWDecayQuark_py    (no streamer)              asdtype('>f4')
MChadronicWDecayQuark_pz    (no streamer)              asdtype('>f4')
MChadronicWDecayQuarkBar_px
                            (no streamer)              asdtype('>f4')
MChadronicWDecayQuarkBar_py
                            (no streamer)              asdtype('>f4')
MChadronicWDecayQuarkBar_pz
                            (no streamer)              asdtype('>f4')
MClepton_px                 (no streamer)              asdtype('>f4')
MClepton_py                 (no streamer)              asdtype('>f4')
MClepton_pz                 (no streamer)              asdtype('>f4')
MCleptonPDGid               (no streamer)              asdtype('>i4')
MCneutrino_px               (no streamer)              asdtype('>f4')
MCneutrino_py               (no streamer)              asdtype('>f4')
MCneutrino_pz               (no streamer)              asdtype('>f4')
NPrimaryVertices            (no streamer)              asdtype('>i4')
triggerIsoMu24              (no streamer)              asdtype(dtype('bool'))
EventWeight                 (no streamer)              asdtype('>f4')
```

Making a plot the old fashioned way, by nested for loops

There ought to be a clever way to express this functionally, but for now we demonstrate that you can do it, if you want to.

The performance of this code should be exactly the same as writing a C program, compiling it, and running it, because that's essentially what's happening with the `@numba.njit`. The beauty of this is that you don't have to leave Python, set up the compilation, etc., etc. You put these functions inline with the rest of your Python-based analysis code.

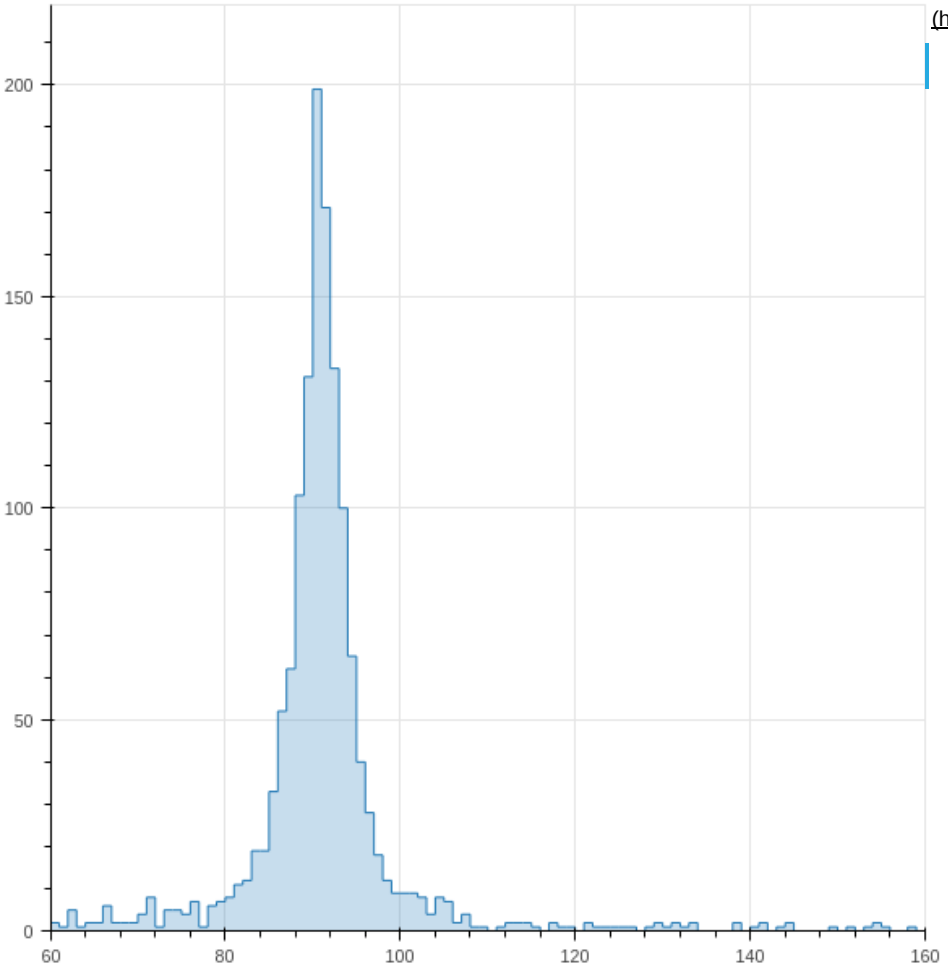
In [15]:
```
import numba
from math import sqrt

@numba.njit
def fillhist(dimuon_hist, quadmuon_hist, NMuon, Muon_Px, Muon_Py, Muon_Pz, Muon_E):   # ha
ve to pass in the histograms to be able to modify them
    for event_i in range(len(NMuon)):
        totE  = 0.0
        totPx = 0.0
        totPy = 0.0
        totPz = 0.0
        for muon_i in range(NMuon[event_i]):                                          # ra
nge(jagged array member) is an optimized function
            for muon_j in range(muon_i + 1, NMuon[event_i]):
                E  = Muon_E[event_i][muon_i]  + Muon_E[event_i][muon_j]               # su
barray extraction is optimized, too
                Px = Muon_Px[event_i][muon_i] + Muon_Px[event_i][muon_j]
                Py = Muon_Py[event_i][muon_i] + Muon_Py[event_i][muon_j]
                Pz = Muon_Pz[event_i][muon_i] + Muon_Pz[event_i][muon_j]
                dimuon_hist.fill(sqrt(E**2 - Px**2 - Py**2 - Pz**2))                  # hi
stogram filling (and fillw for weights) are optimized
            totE  += Muon_E[event_i][muon_i]
            totPx += Muon_Px[event_i][muon_i]
            totPy += Muon_Py[event_i][muon_i]
            totPz += Muon_Pz[event_i][muon_i]
            quadmuon_hist.fill(totE**2 - totPx**2 - totPy**2 - totPz**2)
    return dimuon_hist, quadmuon_hist                                                 # ha
ve to return the histograms to get the updates
```

In [19]:
```
dimuon_hist, quadmuon_hist = fillhist(uproot.hist(100, 60, 160), uproot.hist(100, 60, 160
), *HZZ.arrays(["NMuon", "Muon_Px", "Muon_Py", "Muon_Pz", "Muon_E"], outputtype=tuple))
```

In [20]:
```
show(dimuon_hist.bokeh.fig())
```

Out[20]:

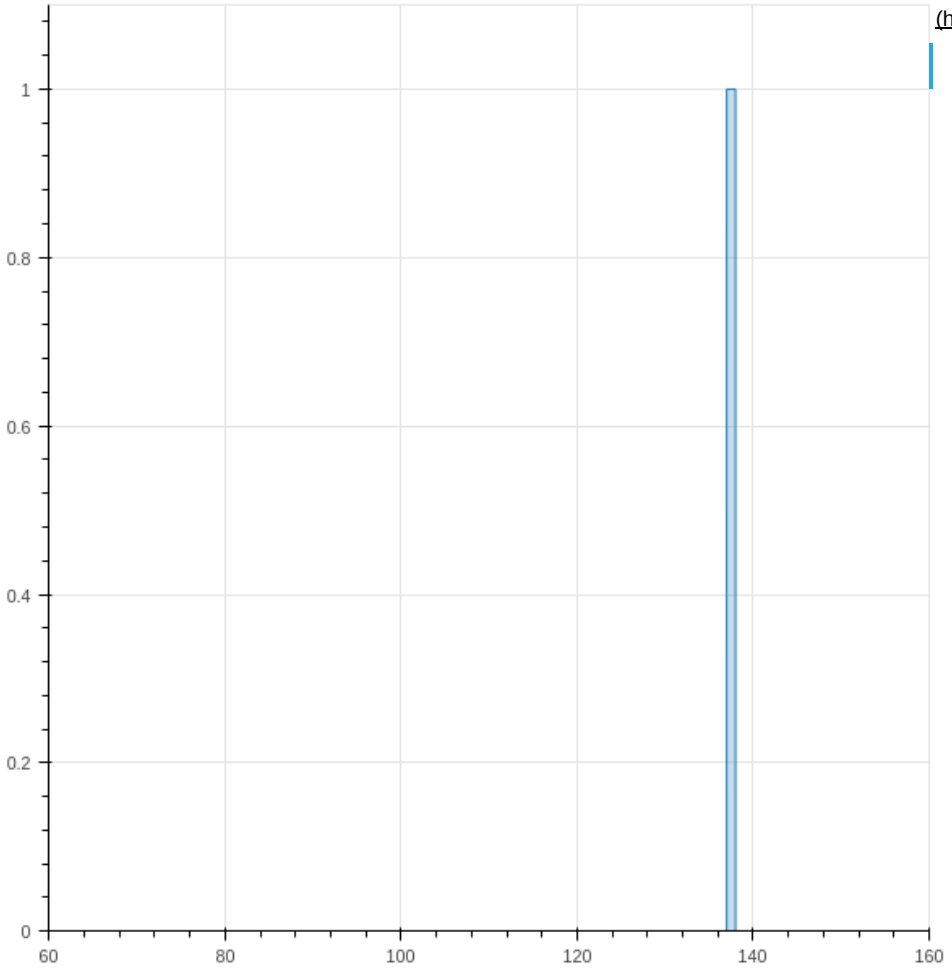


(https://bokeh.pydata.org/)

Out[20]:

In [21]:
```
show(quadmuon_hist.bokeh.fig())
```

Out[21]:



(https://bokeh.pydata.org/)

Out[21]:

I guess this is Monte Carlo!