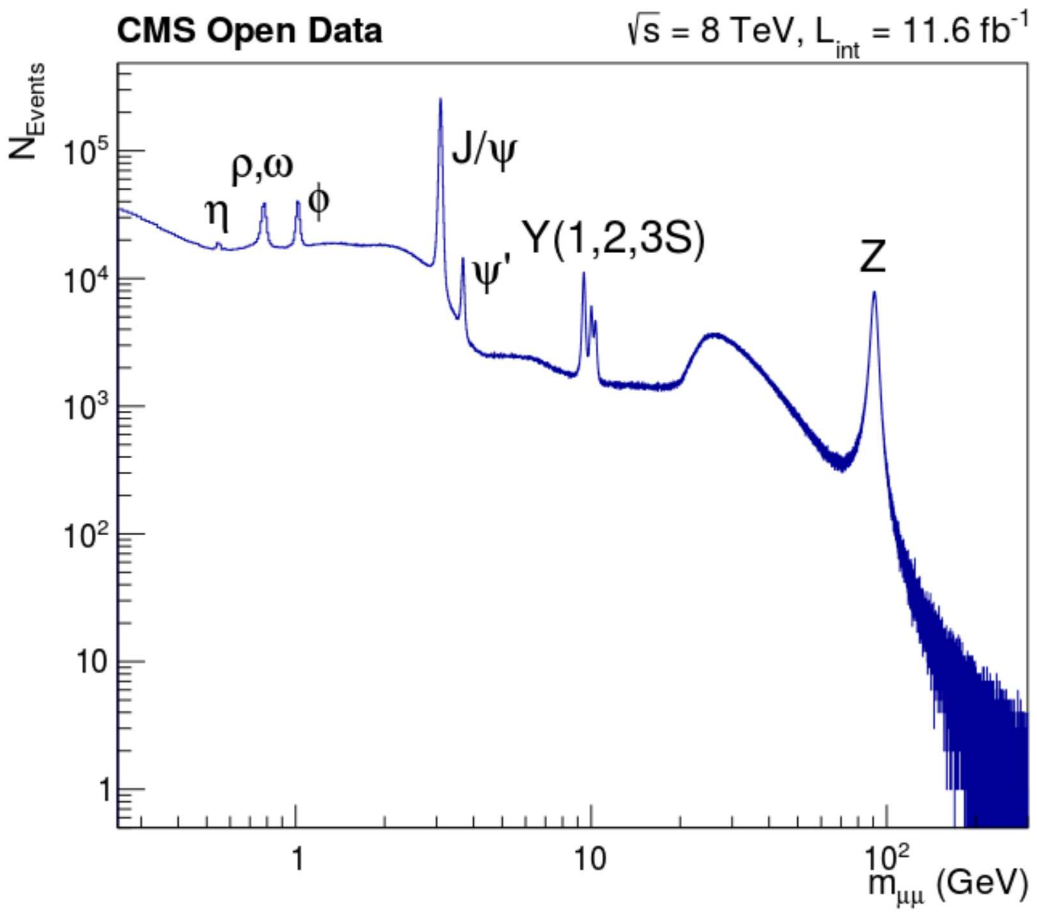
ROOT dataframe tutorial: Dimuon spectrum

This tutorial shows you how to analyze datasets using RDataFrame from a Python notebook. The example analysis performs the following steps:

- Connect a ROOT dataframe to a dataset containing 61 mio. events recorded by CMS in 2012
- Filter the events being relevant for your analysis
- Compute the invariant mass of the selected dimuon candidates
- Plot the invariant mass spectrum showing resonances up to the Z mass

This material is based on the analysis done by Stefan Wunsch, available here in CERN's Open Data portal.



```
In [4]: import ROOT
```

Create a ROOT dataframe in Python

First we will create a ROOT dataframe that is connected to a dataset named `Events` stored in a ROOT file. The file is pulled in via XRootD from EOS public, but note how it could also be stored in your CERNBox space or in any other EOS repository accessible from SWAN (e.g. the experiment ones).

The dataset Events is a TTree and has the following branches:

Branch name	Data type	Description
`nMuon`	`unsigned int`	Number of muons in this event
`Muon_pt`	`float[nMuon]`	Transverse momentum of the muons stored as an array of size `nMuon`
`Muon_eta`	`float[nMuon]`	Pseudo-rapidity of the muons stored as an array of size `nMuon`
`Muon_phi`	`float[nMuon]`	Azimuth of the muons stored as an array of size `nMuon`

Branch name	Data type	Description
`Muon_charge`	`int[nMuon]`	Charge of the muons stored as an array of size `nMuon` and either -1 or 1
`Muon_mass`	`float[nMuon]`	Mass of the muons stored as an array of size `nMuon`

In [5]:
```
treename = "Events"
filename = "root://eospublic.cern.ch//eos/opendata/cms/derived-data/AOD2NanoAODOutreachTool/Run2012BC_Dou
df = ROOT.RDataFrame(treename, filename)
```

Run only on a part of the dataset

The full dataset contains half a year of CMS data taking in 2012 with 61 mio events. For the purpose of this example, we use the Range node to run only on a small part of the dataset. This feature also comes in handy in the development phase of your analysis.

Feel free to experiment with this parameter!

In [6]:
```
# Take only the first 1M events
df_range = df.Range(1000000)
```

Filter relevant events for this analysis

Physics datasets are often general purpose datasets and therefore need extensive filtering of the events for the actual analysis. Here, we implement only a simple selection based on the number of muons and the charge to cut down the dataset in events that are relevant for our study.

In particular, we are applying two filters to keep:

1. Events with exactly two muons
2. Events with muons of opposite charge

In [7]:
```
# Change the first strings of both following operations to proper C++ expressions
# Use the points 1, 2 above as hints for what to write in your expression
df_2mu = df_range.Filter("nMuon == 2", "Events with exactly two muons")
df_oc = df_2mu.Filter("Muon_charge[0] != Muon_charge[1]", "Muons with opposite charge")
```

Perform complex operations in Python, efficiently!

Since we still want to perform complex operations in Python but plain Python code is prone to be slow and not thread-safe, you should use as much as possible C++ functions to do the work in your event loop during runtime. This mechanism uses the C++ interpreter `cling` shipped with ROOT, making this possible in a single line of code.

Note, that we are using here the `Define` node of the computation graph with a jitted function, calling into a function available in the ROOT library.

In [8]:
```
df_mass = df_oc.Define("Dimuon_mass", "ROOT::VecOps::InvariantMass(Muon_pt, Muon_eta, Muon_phi, Muon_mass
```

Make a histogram of the newly created column

In [9]:
```
# These are the parameters you would give to a histogram object constructor
# Put them in the right order inside the parentheses below
# You are effectively passing a tuple to the `Histo1D` operation as seen previously in other notebooks
nbins = 30000
low = 0.25
up = 300
histo_name = "Dimuon_mass"
histo_title = histo_name

h = df_mass.Histo1D((histo_name, histo_title, nbins, low, up), "Dimuon_mass")
```

Book a Report of the dataframe filters

In [10]:
```
report = df.Report()
```

Start data processing

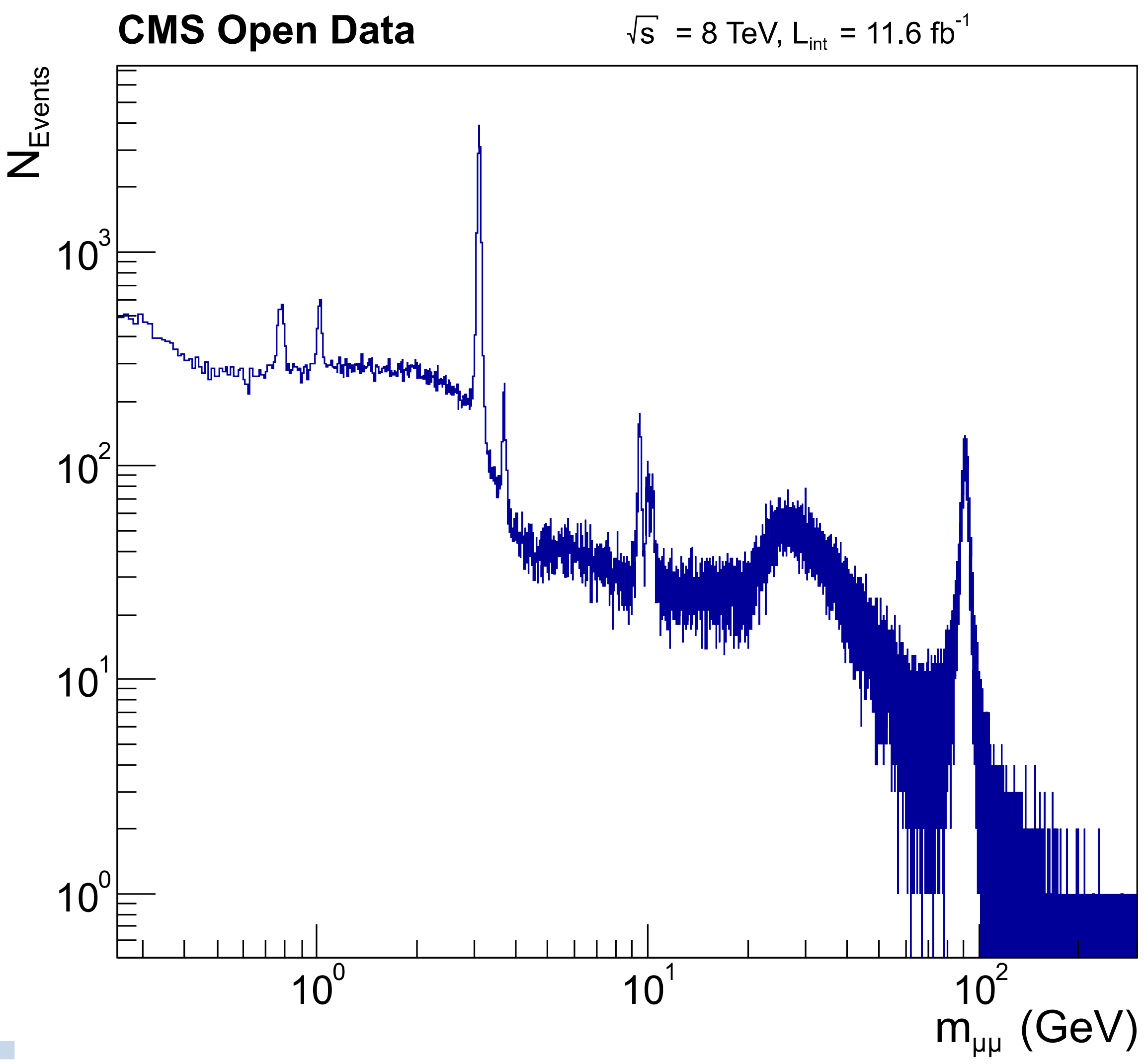
This is the final step of the analysis: retrieving the result. We are expecting to see a plot of the mass of the dimuon spectrum similar to the one shown at the beginning of this exercise (remember we are running on fewer entries in this exercise). Finally in the last cell we should see a report of the filters applied on the dataset.

In [11]:
```
%%time

ROOT.gStyle.SetOptStat(0)
ROOT.gStyle.SetTextFont(42)
c = ROOT.TCanvas("c", "", 800, 700)
c.SetLogx()
c.SetLogy()
h.SetTitle("")
h.GetXaxis().SetTitle("m_{#mu#mu} (GeV)")
h.GetXaxis().SetTitleSize(0.04)
h.GetYaxis().SetTitle("N_{Events}")
h.GetYaxis().SetTitleSize(0.04)
h.Draw()

label = ROOT.TLatex()
label.SetNDC(True)
label.SetTextSize(0.040)
label.DrawLatex(0.100, 0.920, "#bf{CMS Open Data}")
label.SetTextSize(0.030)
label.DrawLatex(0.500, 0.920, "#sqrt{s} = 8 TeV, L_{int} = 11.6 fb^{-1}")
```

```
CPU times: user 11.8 s, sys: 262 ms, total: 12.1 s
Wall time: 12 s
```

Out[11]:
```
<cppyy.gbl.TLatex object at 0xfe01cf0>
```

In [12]:
```
%jsroot on
c.Draw()
```



In [13]:
```
report.Print()
```

```
Events with exactly two muons: pass=489473     all=1000000    -- eff=48.95 % cumulative eff=48.95 %
Muons with opposite charge: pass=371508     all=489473     -- eff=75.90 % cumulative eff=37.15 %
```