UNIVERSITY OF OXFORD

Worked example analysis

Standard Model W boson production

14/12/2021

Eimear Conroy – University of Oxford

ATLAS Open Data

Recipe for a particle physics analysis

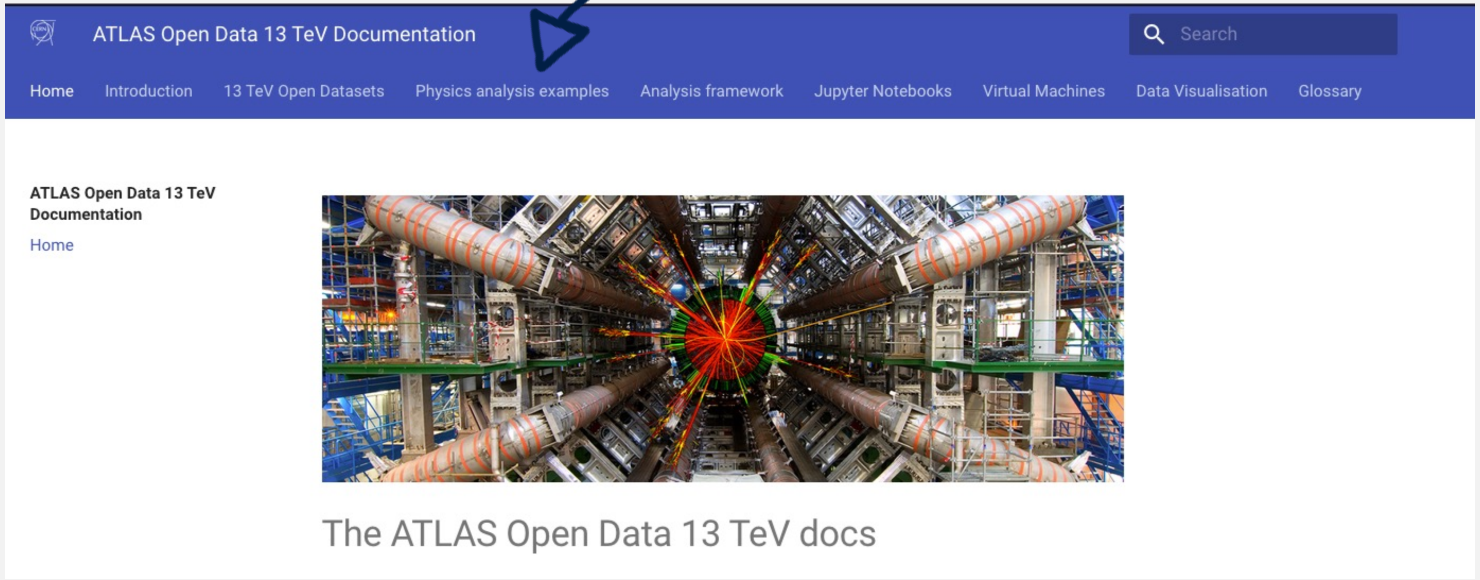
1. Go to the analysis prompt
2. What datasets do we need?
3. What plots will we need?
4. What selections do we need?
5. Implement our event loop
6. Execute our event loop
7. Plot our plots

- In our example analysis, we'll be measuring the Standard Model W boson production following the prompt here.
- For the full example code, look at the W_boson_worked_example notebook

Go to the analysis prompt

Go to the analysis prompt

- The ATLAS Open Data website has a range of skeleton analyses for you to try build yourself
- We're going to try measuring the Standard Model W-boson production in the single-lepton final state



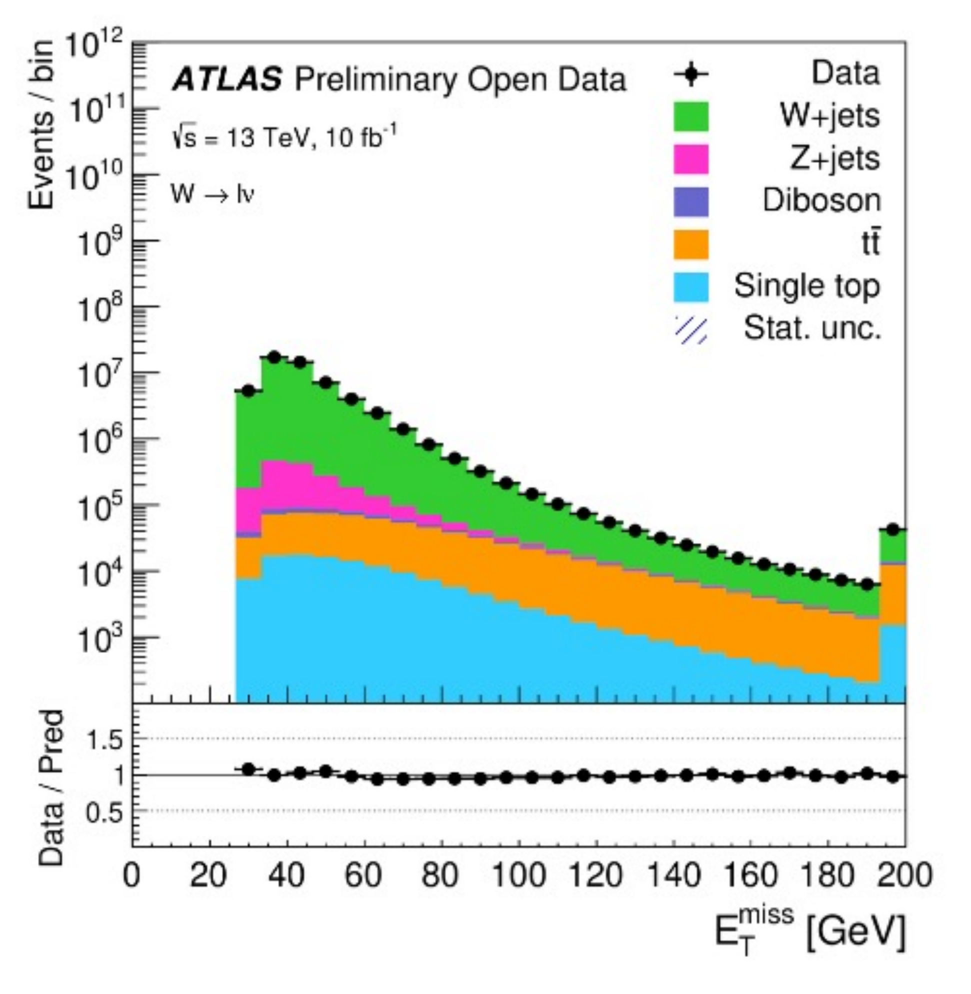
What datasets will we need?

What datasets will we need?

Looking at the plot we're trying to replicate can be helpful here!

Data
- Data with 1 lepton in the final state

Simulation
- Our W+jets **signal**
- **Background** processes that:
  - Also decay to 1 lepton
    - e.g. Single top
  - Can be mistaken for such if the detector gets the measurement slightly wrong
    - e.g. Dibosons



What datasets will we need - Data

Locate the data files and open with ROOT in a Jupyter notebook as in previous notebooks

Index of /atlas-opendata/samples/2020/1lep/Data

Name	Last modified	Size	Description
Parent Directory		-	
data_A.1lep.root	2020-11-03 15:21	1.5G	
data_B.1lep.root	2020-11-03 15:26	5.2G	

multiple files per dataset → store in a list

Data

In [2]:
```
#1 lepton data
data = [ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/Data/data_A.1lep.root")
        ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/Data/data_B.1lep.root")
        ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/Data/data_C.1lep.root")
        ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/Data/data_D.1lep.root")
]

data_trees = [f.Get("mini") for f in data]
```

a quick way of populating a list using a for loop in Python

What datasets will we need - Simulation

Process	Unique “channelNumber”	Generator, hadronisation	Additional information
		Top-quark production	
$t\bar{t}$+jets	410000	Powheg-Box v2 [68] + Pythia 8 [69]	only 1ℓ and 2ℓ decays of $t\bar{t}$-system
single (anti)top t-channel	(410012) 410011	Powheg-Box v1 + Pythia 6 [70]	
single (anti)top Wt-channel	(410014) 410013	Powheg-Box v2 + Pythia 6	
single (anti)top s-channel	(410026) 410025	Powheg-Box v2 + Pythia 6	
		W/Z (+ jets) production	
$Z \to ee,\ \mu\mu,\ \tau\tau$	361106 – 361108	Powheg-Box v2 + Pythia 8	LO accuracy up to $N_{\text{jets}} = 1$
$W \to e\nu,\ \mu\nu,\ \tau\nu$	361100 – 361105	Powheg-Box v2 + Pythia 8	LO accuracy up to $N_{\text{jets}} = 1$
$W \to e\nu,\ \mu\nu,\ \tau\nu$ + jets	364156 – 364197	Sherpa 2.2 [71]	LO accuracy up to 3-jets final states
$Z \to ee,\ \mu\mu,\ \tau\tau$ + jets	364100 – 364141	Sherpa 2.2	LO accuracy up to 3-jets final states
		Diboson production	
WW	363359, 363360	Sherpa 2.2	$qq'\ell\nu$ final states
WW	363492	Sherpa 2.2	$\ell\nu\ell'\nu'$ final states
ZZ	363356	Sherpa 2.2	$qq'\ell^+\ell^-$ final states
ZZ	363490	Sherpa 2.2	$\ell^+\ell^-\ell'^+\ell'^-$ final states
WZ	363358	Sherpa 2.2	$qq'\ell^+\ell^-$ final states
WZ	363489	Sherpa 2.2	$\ell\nu qq'$ final states
WZ	363491	Sherpa 2.2	$\ell\nu\ell^+\ell^-$ final states
WZ	363493	Sherpa 2.2	$\ell\nu\nu\nu'$ final states
		SM Higgs production ($m_{\text{H}} = 125$ GeV)	
ggF, $H \to WW$	345324	Powheg-Box v2 + Pythia 8	$\ell\nu\ell'\nu'$ final states
VBF, $H \to WW$	345323	Powheg-Box v2 + Pythia 8	$\ell\nu\ell'\nu'$ final states
ggF, $H \to ZZ$	345060	Powheg-Box v2 + Pythia 8	$\ell^+\ell^-\ell'^+\ell'^-$ final states
VBF, $H \to ZZ$	344235	Powheg-Box v2 + Pythia 8	$\ell^+\ell^-\ell'^+\ell'^-$ final states
ZH, $H \to ZZ$	341947	Pythia 8	$\ell^+\ell^-\ell'^+\ell'^-$ final states
WH, $H \to ZZ$	341964	Pythia 8	$\ell^+\ell^-\ell'^+\ell'^-$ final states
ggF, $H \to \gamma\gamma$	343981	Powheg-Box v2 + Pythia 8	
VBF, $H \to \gamma\gamma$	345041	Powheg-Box v2 + Pythia 8	
$WH(ZH)$, $H \to \gamma\gamma$	345318, 345319	Powheg-Box v2 + Pythia 8	
$t\bar{t}H$, $H \to \gamma\gamma$	341081	aMC@NLO [72] + Pythia 8	
		BSM production	
$Z' \to t\bar{t}$	301325	Pythia 8	$m_{Z'} = 1$ TeV
$\tilde{\ell}\tilde{\ell}' \to \ell\tilde{\chi}_1^0\ell'\tilde{\chi}_1^{0\prime}$	392985	aMC@NLO + Pythia 8	$m_{\tilde{\ell}} = 600$ GeV, $m_{\tilde{\chi}_1^0} = 300$ GeV

- Once you know what simulated datasets you need, you can use this table to to match the process with a unique channel number

- This **channel number** will be in the name of the file you want to import

What datasets will we need - Simulation

Repeat file import for each simulated dataset

Monte-Carlo

```
In [3]: #Single top
single_top = [ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_410011.sing
              ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_410012.sing
              ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_410013.sing
              ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_410014.sing
              ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_410025.sing
              ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_410026.sing
]

single_top_trees = [f.Get("mini") for f in single_top]
```

```
In [4]: #T-tbar
ttbar = [ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363356.ZqqZll.1l
         ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363358.WqqZll.1l
         ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363359.WpqqWmlv.
         ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363360.WplvWmqq.
         ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363489.WlvZqq.1l
         ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363490.llll.1lep
         ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363491.lllv.1lep
         ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363492.llvv.1lep
         ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/1lep/MC/mc_363493.lvvv.1lep
]

ttbar_trees = [f.Get("mini") for f in ttbar]
```

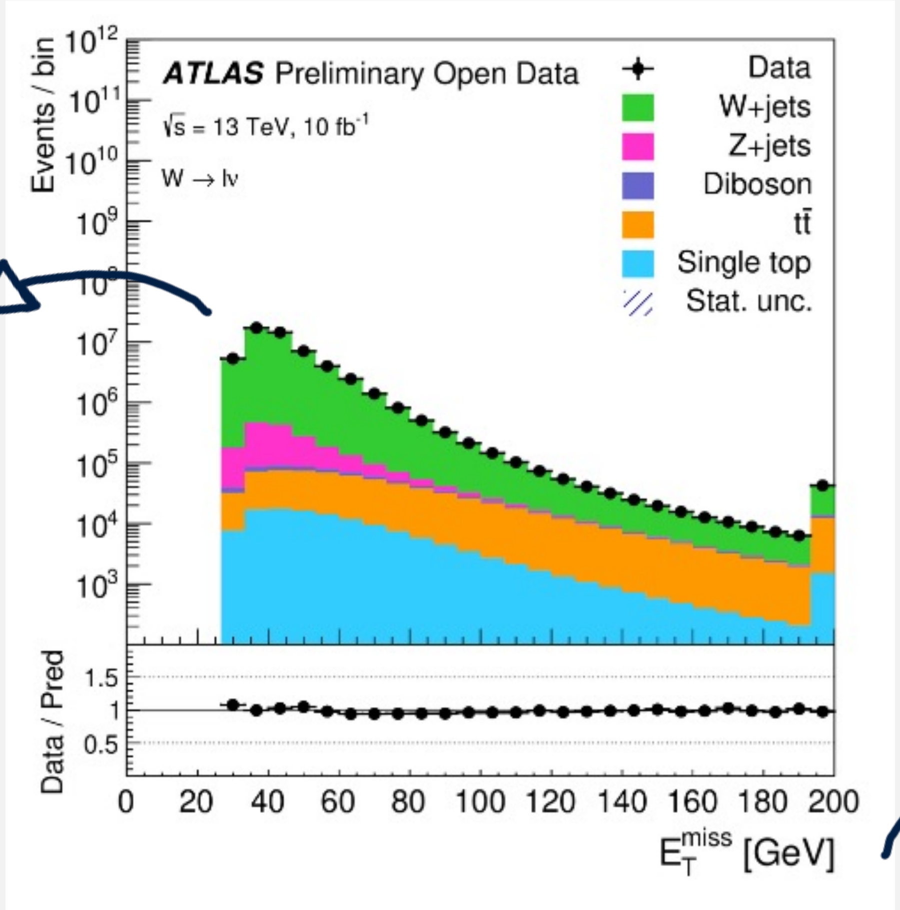
⋮

What plots will we need?

What plots will we need?

Going back to the prompt…

Simulated processes are shown as separate, stacked histograms – we'll want to do the same



Variable being plotted – E_T^{miss} (sometimes called MET) – energy escaping the detector, in this case, the energy of the neutrino

What plots will we need?

Set up your canvas and histograms as shown in previous notebooks

```
canvas = ROOT.TCanvas("Canvas","MET_canvas",800,600)
```

```
h_dat     = ROOT.TH1F("h_dat","MET; Missing energy E_{T}^{miss} [GeV] ; events",26,20,200)

#Want to plot all the simulated samples seperately
h_t       = ROOT.TH1F("h_t","MET; Missing energy E_{T}^{miss} [GeV] ; events",26,20,200)
h_ttbar   = ROOT.TH1F("h_ttbar","MET; Missing energy E_{T}^{miss} [GeV] ; events",26,20,200)
h_diboson = ROOT.TH1F("h_diboson","MET; Missing energy E_{T}^{miss} [GeV] ; events",26,20,200)
h_Zjets   = ROOT.TH1F("h_Zjets","MET; Missing energy E_{T}^{miss} [GeV] ; events",26,20,200)
h_Wjets   = ROOT.TH1F("h_Wjets","MET; Missing energy E_{T}^{miss} [GeV] ; events",26,20,200)
```

You can use the plot in the prompt to help decide the correct binnings

What selections will we need?

What selections will we need

According to the prompt…

Event

- Single-electron or single-muon trigger satisfied;

Event

- Exactly one light lepton (electron or muon) with $p_T > 35$ GeV;

Object

- Missing transverse momentum[3] larger than 30 GeV;

Event

- The transverse mass of the W-boson[4] (M_T^W): $M_T^W > 60$ GeV.

Event – M_T^W is built from E_T^{miss}, which is a property of the whole event

Separate which selections dictate which objects we keep ("**object-level**"), and which are applied to the whole event ("**event-level**")

What selections will we need

Code up your object-level selections, event-level will come later

This piece of code will be used a lot, so it's helpful to write it as a function

```
def goodLeptons(tree):
    """
    A function to return the indices of 'good leptons' (electrons or muons) in an event. This follows
    many of the same steps as locateGoodPhotons() and photonIsolation() in Co-Creation3_HiggsSearch.

    Parameters
    ----------
    tree : TTree entry for this event
    """

    #Initialise (set up) the variables we want to return
    goodlepton_index = [] #Indices (position in list of event's leptons) of our good leptons

    ##Loop through all the leptons in the event
    for j in range(0,tree.lep_n):

        ##Check lepton pT
        if tree.lep_pt[j] > 35000:

            #Store lepton's index
            goodlepton_index.append(j)

    return goodlepton_index #return list of good lepton indices
```

Implement our event loop

Implement our event loop

Code taken from
Co-Creation3_HiggsSearch

Before we start, there are two last ingredients we'll need…

```
def trackProgress(n,m):
    """
    Function which prints the event loop progress every m events

    Parameters
    ----------
    n : Number of events processed so far

    m : Printout event interval

    """
    if n == 0:
        print("Event loop tracker")
        print("------------------")

    if(n%m==0):
        print("%d events processed" % n)
```

1. A progress tracker, to keep informed on how far along the loop is

Implement our event loop

Code taken from
Co-Creation3_HiggsSearch

Before we start, there are two last ingredients we'll need…

```
def mcWeights(tree,lumi=10):
    """
    When MC simulation is compared to data the contribution of each simulated event needs to be
    scaled ('reweighted') to account for differences in how some objects behave in simulation
    vs in data, as well as the fact that there are different numbers of events in the MC tree than
    in the data tree.

    Parameters
    ----------
    tree : TTree entry for this event
    """

    #These values don't change from event to event
    norm = lumi*(tree.XSection*1000)/tree.SumWeights

    #These values do change from event to event
    scale_factors = tree.scaleFactor_ELE*tree.scaleFactor_MUON*tree.scaleFactor_LepTRIGGER*tree.scaleFactor_PILEUP*t

    weight = norm*scale_factors
    return weight
```

2. We're comparing real data with simulation, so simulated events will have to be **reweighted** to account for different numbers of events generated, and differences in how certain tools (e.g. triggers) perform on simulated events vs real events

Combine everything into one function

```
def W_production(tree,hist,mode):
    """
    Function which executes the analysis flow for the W production measurement.

    Fills a histogram with MET of events which pass the full set of cuts

    Parameters
    ----------
    tree : A Ttree containing data / background information

    hist : The name of the histogram to be filled with mT(llvv) values

    mode : A flag to tell the function if it is looping over 'data' or 'mc'
    """

    n = 0
    for event in tree:

        ###############################
        ### Event-level requirements
        ###############################

        trackProgress(n,100000)
        n += 1

        #If event is MC: Reweight it
        if mode.lower() == 'mc': weight = mcWeights(tree)
        else: weight = 1
```

progress tracker

reweight MC

continued

Cuts: Event-level and object-level

```
#If the event passes either the electron or muon trigger
if tree.trigE or tree.trigM:

    ####Lepton preselections
    goodLeps = goodLeptons(tree) #If the datafiles were not already filtered by number of leptons

    ####################################
    ### Individual lepton requirements
    ####################################

    if len(goodLeps) == 1: #Exactly one good lepton...

        #####################
        ### MET requirements
        #####################

        #Initialse (set up) an empty 4 vector for the event's MET and fill from tree
        met_four_mom = ROOT.TLorentzVector()
        met_four_mom.SetPtEtaPhiE(tree.met_et,0,tree.met_phi,tree.met_et)

        #MET > 30 GeV
        if met_four_mom.Pt() > 30000:

            #################
            ### W requirements
            #################

            # Because of conservation of energy/momentum, the W boson 4-momentum
            # = (lepton 4-momentum + MET (i.e. neutrino) 4-momentum)

            #Initialse (set up) an empty 4 vector for dilepton system
            W_four_mmtm = ROOT.TLorentzVector()

            #Initialse (set up) an empty 4 vector for the lepton
            lep_four_mmtm = ROOT.TLorentzVector()
            lep_four_mmtm.SetPtEtaPhiE(tree.lep_pt[0], tree.lep_eta[0],tree.lep_phi[0],tree.lep_E[0])

            #Store lepton's 4 momentum
            W_four_mmtm = lep_four_mmtm + met_four_mom

            #W 'Transverse mass' > 60 GeV
            if W_four_mmtm.Mt() > 60000:

                #If Fill() is passed a second argument, the event is weighted by that amount
                hist.Fill(tree.met_et/1000,weight)
```

fill the histogram

Execute our event loop

Execute our event loop

How do we execute an analysis over multiple datasets, each containing multiple files?

Gather our lists of TTrees together into a list a data samples and simulated samples

```
Data = [data_trees]
Sim  = [single_top_trees,ttbar_trees,Diboson_trees,Zjets_trees,Wjets_trees]
```

```
data_hists = [h_dat]
sim_hists  = [h_t,h_ttbar,h_diboson,h_Zjets,h_Wjets]
```

We're going to be using list indexing, so make sure to use the same order in both the TTree and histogram lists!

Do the same for our (currently empty) histograms

Execute our event loop

How do we execute an analysis over multiple datasets, each containing multiple files?

Loop over our sample lists, calling our event loop function on each TTree

```
#Data
for i in range(0,len(Data)):
    hist      = data_hists[i]
    tree_list = Data[i]
    for tree in tree_list:
        print("\nNumber of events in file: %d\n" % tree.GetEntries())
        W_production(tree,hist,'data')
```

Add some printouts to help keep track of progress

```
#MC
for i in range(0,len(Sim)):
    print("\n### Sample number: %d ###" % i)
    hist      = sim_hists[i]
    tree_list = Sim[i]
    for tree in tree_list:
        print("\nNumber of events in file: %d\n" % tree.GetEntries())
        W_production(tree,hist,'mc')
```

Remember to change the MC reweighting settings!

Execute our event loop

- Note that some very common processes, such as W production, can have **lots** of events
- Because of this, our analysis code can take a **long** time to execute
- Some good ideas for handling this are:
  - Leave your code running overnight
  - Do some research/speak to an advisor on how to run your code either
    - From the command line, backgrounded
    - As a python script instead of a Jupyter notebook

Plot our plots

Plot our plots

If you're unfamiliar with any of the functions or options used here, Google "ROOT *(function name)*" to find the documentation

First, some formatting…

```
#Some formatting options

h_dat.SetMarkerStyle(20) #Choose the shape for our markers
h_dat.SetMarkerSize(0.5) #Choose the size for our markers

h_t.SetFillColor(7)
h_ttbar.SetFillColor(5)
h_diboson.SetFillColor(9)
h_Zjets.SetFillColor(6)
h_Wjets.SetFillColor(3)

h_t.SetLineColor(1)
h_ttbar.SetLineColor(1)
h_diboson.SetLineColor(1)
h_Zjets.SetLineColor(1)
h_Wjets.SetLineColor(1)
```

Black dots are traditional for data points

Make each MC histogram a different colour, with a black outline

Plot our plots

If you're unfamiliar with any of the functions or options used here, Google "ROOT *(function name)*" to find the documentation

Our reference plot stacks the MC histograms on top of each other, let's do the same here using **ROOT.THStack()**

Set up an empty "stacked" histogram

```
#How to create a stacked histogram
hs1 = ROOT.THStack("hs1"," stacked")

hs1.Add(h_t)
hs1.Add(h_ttbar)
hs1.Add(h_diboson)
hs1.Add(h_Zjets)
hs1.Add(h_Wjets)
```

Add each MC histogram to the stacked histogram

Plot our plots

If you're unfamiliar with any of the functions or options used here, Google "ROOT *(function name)*" to find the documentation

Remove the default box with statistical information, and add a legend

```
h_dat.SetStats(0) #Remove the stats box

legend=ROOT.TLegend(0.75,0.7,0.9,0.9) #Add a legend instead
legend.AddEntry(h_dat,"Data","p")
legend.AddEntry(h_Wjets,"W+jets","f")
legend.AddEntry(h_Zjets,"Z+jets","f")
legend.AddEntry(h_diboson,"Diboson","f")
legend.AddEntry(h_ttbar,"ttbar","f")
legend.AddEntry(h_t,"Single top","f")
```

- "p": Appears in legend as a point
- "f": Appears in legend as a filled box

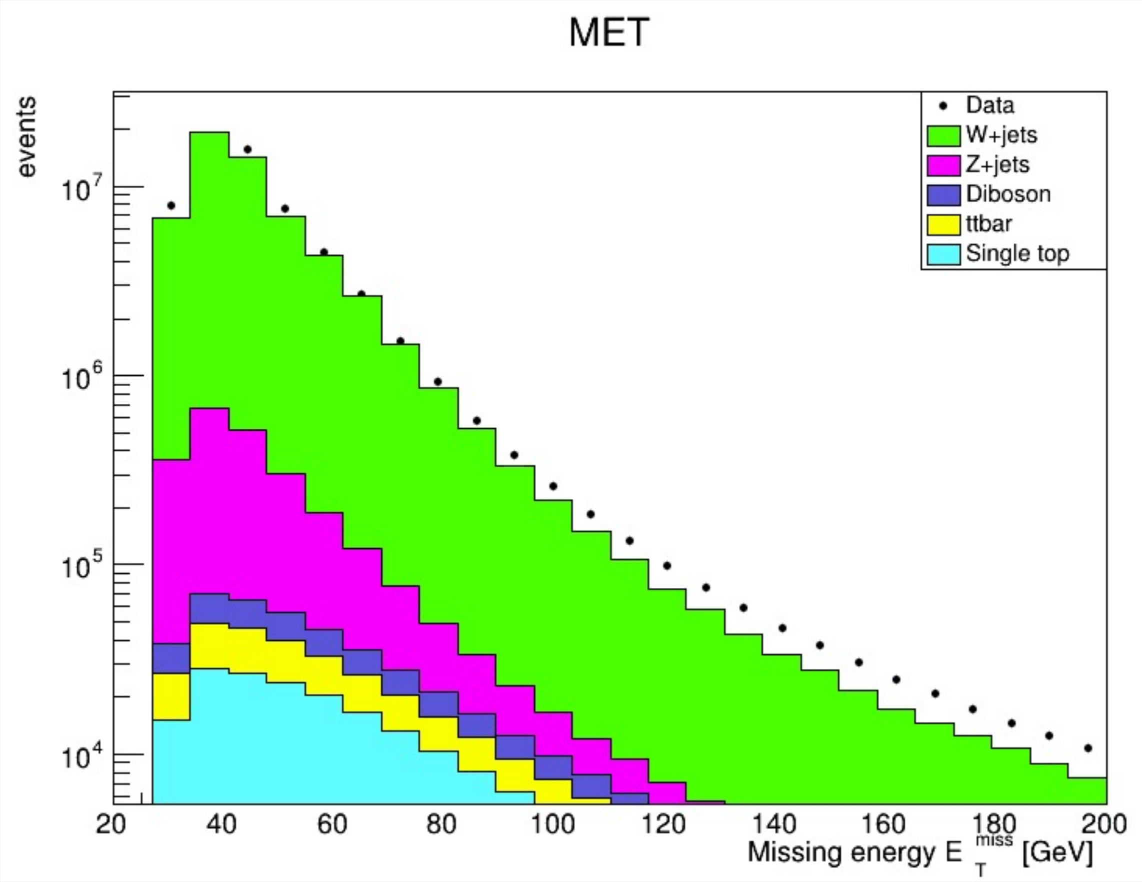
Plot our plots

If you're unfamiliar with any of the functions or options used here, Google "ROOT *(function name)*" to find the documentation

Finally, get drawing!!

```
h_dat.Draw("P") #Plot bin markers
hs1.Draw("histsame")
legend.Draw()

canvas.SetLogy() #Set y axis to log scale
canvas.Draw()
```



Congratulations!

- You now know how to create a particle physics analysis from scratch using an ATLAS Open Data prompt!

- Your turn: Choose another prompt and replicate that, or use your new skills to get answering your own particle physics questions.