Lab 8

Ondrea Robinson

December 8, 2019

This final lab is a fuller analysis of the LHC data. We will look at simulated data to decide on event selection then we will apply this to more realistic pseudo data.

1 Part I (Lab 7)

Before we go into more detail, here's an overview of the Python packages we'll be using and how we'll be reading the data files for this lab.

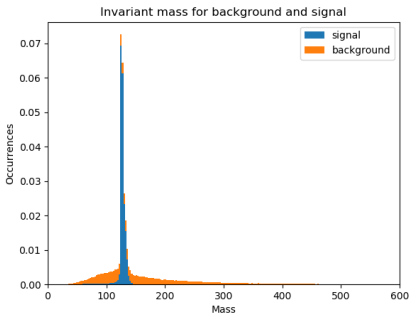
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import pylab
from pylab import *
import scipy
from scipy import stats
import pickle
import pandas

infile = open ("higgs_100000_pt_1000_1200.pkl",'rb')
backgfile = open("qcd_100000_pt_1000_1200.pkl", 'rb')
signal_dict = pickle.load(infile)
backg_dict = pickle.load(backgfile)
```

1.1

Before looking at the pseudo data (high and low luminosity data-sets), we'll be analyzing simulated LHC data (higgs and qcd data sets). First let's look at mass. We look at the two distributions of mass for the signal and background and it's very clear that the signal is peaked around one specific mass value while the background is more gently distributed over a wider range. Below is a stacked histogram of the two data sets:

```
figure()
x = [signal_dict['mass'], backg_dict['mass']]
```

```
labels = ['signal', 'background']
plt.hist(x, 300, stacked=True, density=True, label=labels)
plt.xlim(0, 600)
plt.title('Invariant mass for background and signal')
plt.xlabel('Mass')
plt.ylabel('Occurrences')
plt.legend()
plt.show()
```



Note that the tail of the signal histogram covers values less than the peak value. The tail in the direction of higher mass values is very short and maxes out very close to the peak. This indicates that for an event to be signal-like we should look at mass values that are *lower* than the peak rather than higher.

We want to be able to use these two plots to calculate the significance of a signal value without any event selection. We'll be using Poisson statistics. Here's the code for the significance. I chose the average signal value as our measurement:

```
#probability of getting val or lower using Poisson statistics
val = np.mean(signal_dict['mass'])
uncutmean = np.mean(backg_dict['mass'])
prob = stats.poisson.cdf(val, mu=uncutmean)
sig = stats.norm.ppf(1-prob)
print(sig)
```

We get an uncut significance of 4.08. This is fairly good but could be better. We also want to know the signal to noise ratio. We're going to use $\frac{N_{higgs}}{\sqrt{N_{qcd}}}$. For this number I got out 316.22. This means that there is a lot of signal data in comparison to our background. This is not what we would generally expect since the background should be constantly occurring and signal events happen infrequently.

1.2

From the mass histogram we can see a distinct difference in signal and background distributions but the signal is still relatively buried in the background. In order to see a clearer distinction, let's look at the other variables. We can do a scatter plot of mass versus each other variable to see if there's a common range for the signal mass versus the background mass. Here's the code and the 14 plots:

```
keys = signal_dict.keys()
backmass = backg_dict['mass']
sigmass = signal_dict['mass']
titles = ['Transverse Momentum', 'Psuedorapidity', 'Azimuthal Angle',
    'Invariant Mass', '2-point ECF Ratio', '3-Point ECFRatio',
    '3 to 2 point ECF Ratio', 'Angularity',
    't1','t2','t3','t21', 't32', 'KtDeltaR']
j=0
for i in keys:
    backg = backg_dict[i]
    signal = signal_dict[i]
    figure()
    plt.scatter(backmass, backg)
    plt.scatter(sigmass, signal)
    plt.xlabel('Mass')
    plt.title('Mass vs ' + titles[j])
    plt.ylabel('Occurences')
    plt.show()
    j=j+1
```

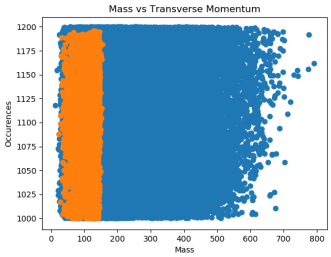


Figure 1

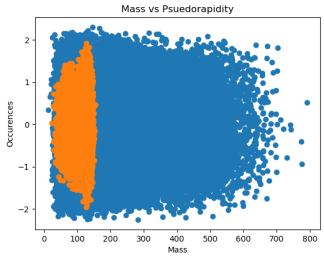


Figure 2

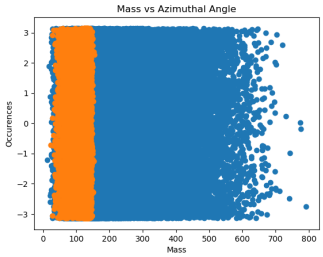


Figure 3

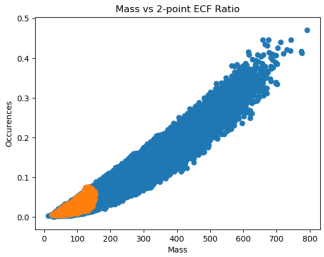


Figure 4

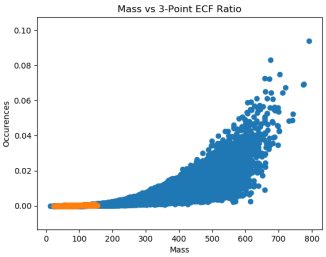


Figure 5

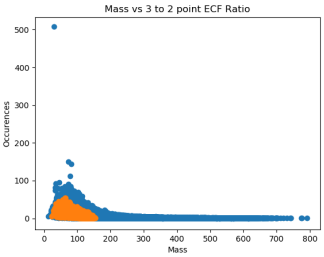


Figure 6

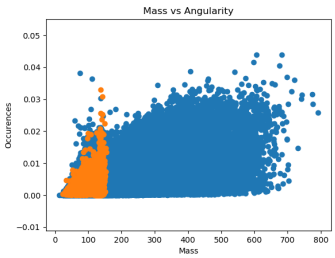


Figure 7

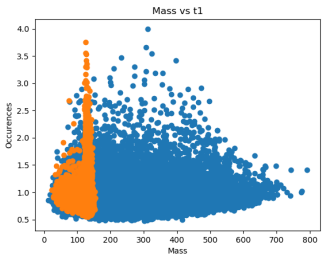


Figure 8

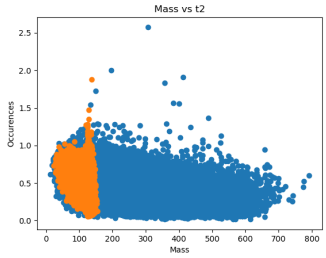


Figure 9

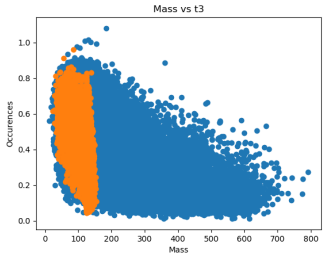


Figure 10

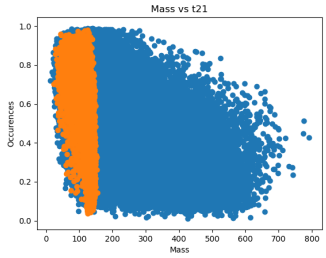


Figure 11

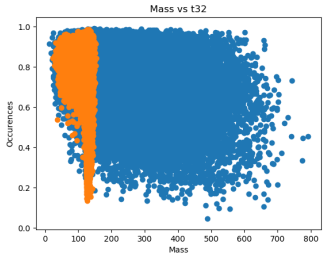


Figure 12

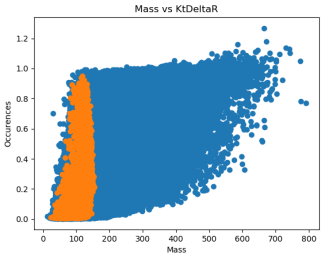


Figure 13

The clearest value where we can make a cut is at 150 mass units. Almost all signal mass values fall below this value with background primarily greater than that. I experimented with two different methods to find the optimal cut.

For both methods we pick different mass values x as our threshold and systematically change x to find the best significance. However, we still need to decide if we want to keep the data above or below our threshold. In the case where we keep the data below, we're decreasing the amount of high background values. This increases the significance of events that are defined so that higher values are more signal-like. In the second case, we keep the data above our threshold x. This increases the amount of background at high values and increases the significance of events that are defined so lower values are more signal-like.

Since we decided that lower values are more signal-like at first glance we should

go with the second method. I tested both ways and consistently got higher significance using the second method. I'm going to include the tests I did both ways below. First, the function below decides which indices to keep given an array of values and a certain threshold and returns an array of those indices. The test in the if statement can be changed easily to select for values above or below the threshold.

```
def keepIndices(array, threshold):
    indices = np.array([])
    for i in range(len(array)):
        if threshold <= array[i]:
            indices = np.append(indices, i)
    return indices
```

The method shown selects for events above the threshold value. For the opposite selection we replace the if test with:

```
(array[i] <= threshold)
```

Now that we know which indices to keep we can test which threshold provides the best significance. The code below tests values from 100 to 150 in steps of 10 mass units. We can only go up to 150 in this case because we're cutting data below the threshold and the highest signal value is between 150 and 160.

```
i=100
var = 'mass'
maxcut = i
maxsig = 0
while i <= 150:
    backgroundID = keepIndices(backg_dict[var], i)
    signalID = keepIndices(signal_dict[var], i)

    cutbackg = backg_dict[var]
    cutbackg = cutbackg[backgroundID]

    cutsignal = signal_dict[var]
    cutsignal = cutsignal[signalID]

    cutmean = np.mean(cutbackg)

    val = np.mean(cutsignal)
    prob = stats.poisson.cdf(val, cutmean)
    sig  = stats.norm.ppf(1-prob)

    if maxsig < sig:
        maxcut = i
        maxsig = abs(sig)
```

```
    print('cut at ', i , ' w/ significance ', sig)
    i = i + 10
```

Python gives us the significance at each cut:

cut at 100 w/ significance 5.554625846907397
cut at 110 w/ significance 6.091125193232889
cut at 120 w/ significance 6.712876106161238
cut at 130 w/ significance 6.848107218309519
cut at 140 w/ significance 6.784932958444955
cut at 150 w/ significance 6.649940202170263

We see the highest significance is around the mass value 130. The significance is in units of sigma and we can already see higher values after cutting than we did from the uncut distributions. We have a rough estimate for the best cut but we want to zoom in to see if we can optimize it more. We can run the same code but instead go from 120 to 140 in steps of 2 mass units. Our output looks like:

cut at 120 w/ significance 6.712876106161238
cut at 122 w/ significance 6.749694240734246
cut at 124 w/ significance 6.8753617181369915
cut at 126 w/ significance 6.917577366737428
cut at 128 w/ significance 6.878717209152588
cut at 130 w/ significance 6.848107218309519
cut at 132 w/ significance 6.896044209537601
cut at 134 w/ significance 6.869585992844234
cut at 136 w/ significance 6.843394779321498
cut at 138 w/ significance 6.81166741303017
cut at 140 w/ significance 6.784932958444955

From this we can take the value of 126 as the optimal cut. We could potentially optimize more but this is specific enough for now. Now let's look at selection with our other method. Recall in this method we're cutting so that we keep data that's below our threshold and we change our indices method to select for this. We're also defining signal-like to be high values. This means we have to change our while loop calculating the significance for each chosen threshold. The resulting code and output look like:

```
def keepIndices(array, threshold):
    indices = np.array([])
    for i in range(len(array)):
        if array[i] <= threshold:
            indices = np.append(indices, i)
    return indices

i=200
maxcut = i
```

```
maxsig = 0
var = 'mass'
while i <= 200:
    backgroundID = keepIndices(backg_dict[var], i)
    signalID = keepIndices(signal_dict[var], i)

    cutbackg = backg_dict[var]
    cutbackg = cutbackg[backgroundID]

    cutsignal = signal_dict[var]
    cutsignal = cutsignal[signalID]

    cutmean = np.mean(cutbackg)

    val = np.mean(cutsignal)
    prob = 1-stats.poisson.cdf(val, cutmean)
    sig  = stats.norm.ppf(prob)

    if maxsig < sig:
        maxcut = i
        maxsig = abs(sig)

    print('cut at ' , i , ' w/ significance ', sig)
    i = i + 10
```

```
cut at 100 w/ significance 0.2959887863661352
cut at 110 w/ significance 0.10457589626890214
cut at 120 w/ significance -0.3935751772919666
cut at 130 w/ significance -2.6614810888786637
cut at 140 w/ significance -2.320225348180627
cut at 150 w/ significance -1.8617230905744009
cut at 160 w/ significance -1.4470168617162864
cut at 170 w/ significance -1.0685744932849357
cut at 180 w/ significance -0.7351228592058063
cut at 190 w/ significance -0.4230836556578054
cut at 200 w/ significance -0.1411715522274427
```

With this style of cut even the absolute value of the significance is lower than even the uncut data. We clearly are better off using the first method and we can take the cut at 126 as the optimal cut.

1.3

So now that we've looked at mass, let's look through the rest of the features to see whether there are any obvious cuts we can make to improve significance. We're going to do stacked histograms for each feature. Here's the code and the

plots. The plots without event selection are on the left and the plots with event selection are on the right. Directly below is the code for each starting with the set without event selection.

```
j=0
for i in keys:
    x = [signal_dict[i], backg_dict[i]]
    figure()
    labels = ['signal', 'background']
    plt.hist(x, 300, stacked=True, density=True, label=labels)
    plt.xlabel(titles[j])
    plt.ylabel('Occurences')
    plt.title('Histogram of ' + titles[j] + ' for Signal and Background')
    plt.legend()
    plt.show()
    j=j+1
```

After event selection of the mass we get a second set of plots using the code:

```
signalcut = keepIndices(signal_dict['mass'], 126)
backgcut = keepIndices(backg_dict['mass'], 126)

j = 0
for i in keys:
    cutsignal = signal_dict[i]
    cutsignal = cutsignal[signalcut]

    cutbackg = backg_dict[i]
    cutbackg = cutbackg[backgcut]

    figure()
    x = [cutsignal, cutbackg]
    labels = ['signal', 'background']
    plt.hist(x, 300, stacked=True, density=True, label=labels)
    plt.title('Histogram of ' + titles[j] + ' for Signal and
    Background (cut)')
    plt.legend()
    plt.show()
    j= j+1
```

And the plots:

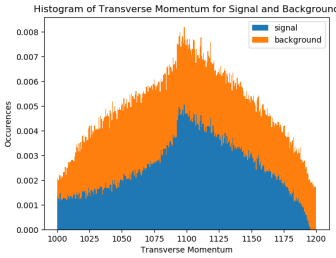


Figure 14

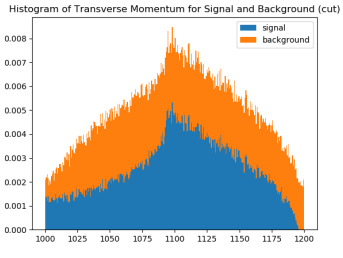


Figure 15

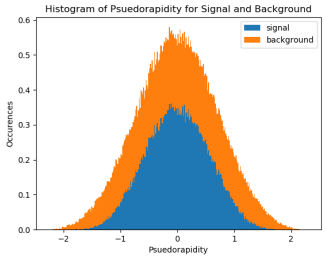


Figure 16

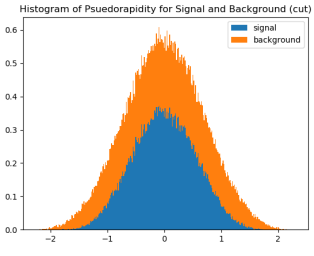


Figure 17

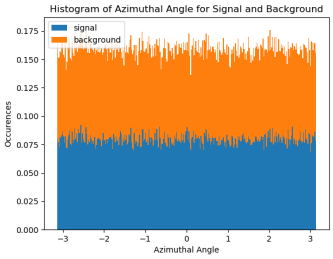


Figure 18

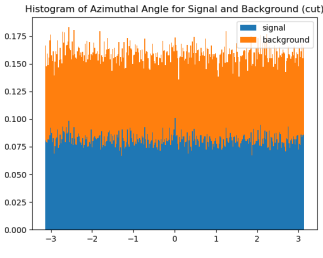


Figure 19

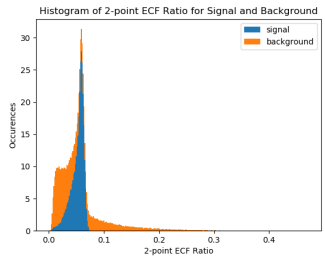


Figure 20

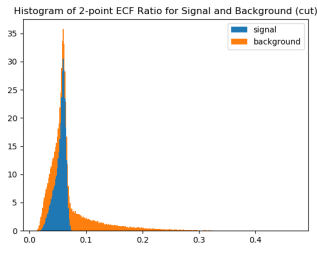


Figure 21

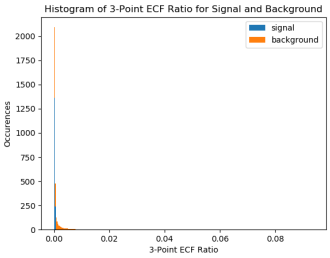


Figure 22

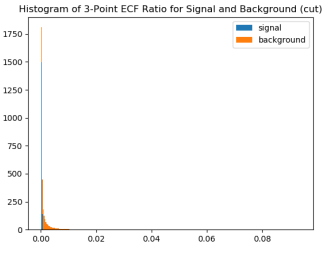


Figure 23

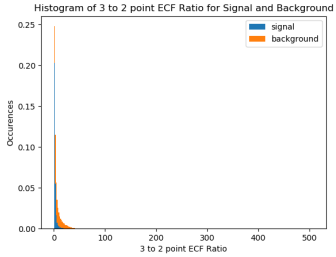


Figure 24

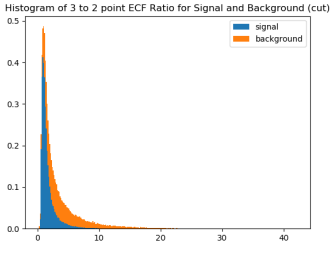


Figure 25

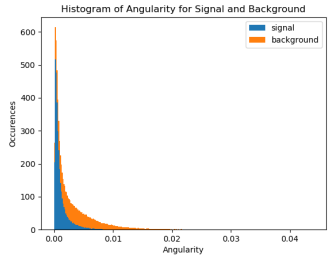


Figure 26

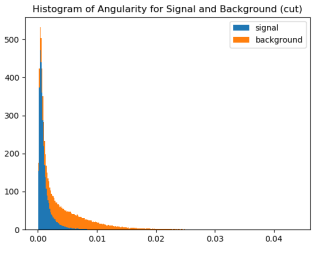


Figure 27

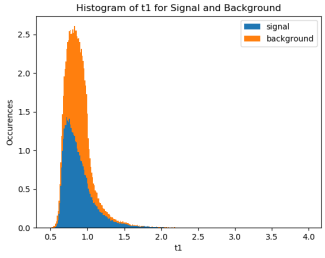


Figure 28

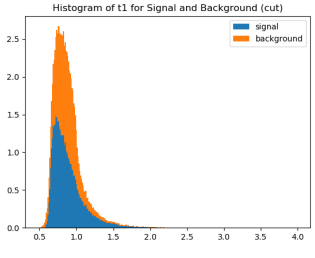


Figure 29

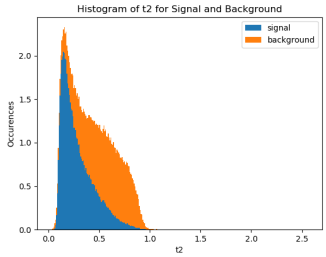


Figure 30

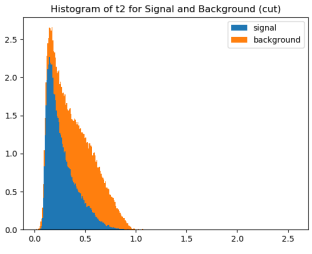


Figure 31

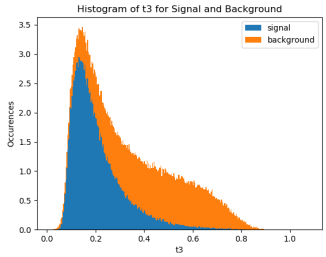


Figure 32

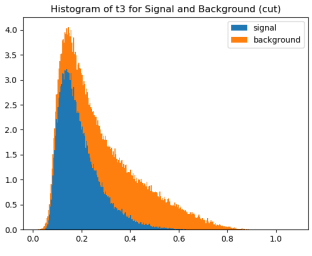


Figure 33

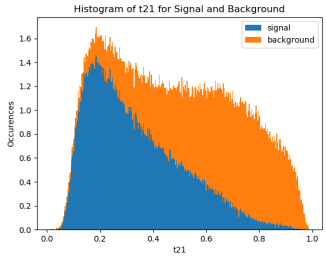


Figure 34

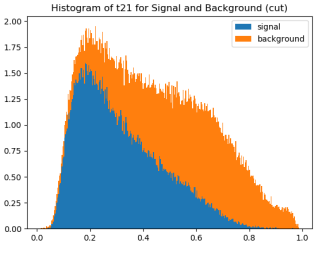


Figure 35

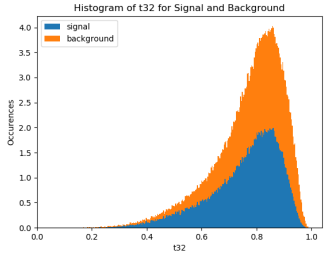


Figure 36

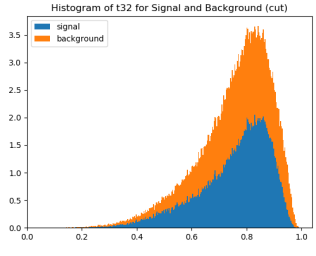


Figure 37

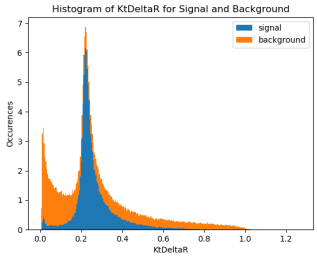


Figure 38

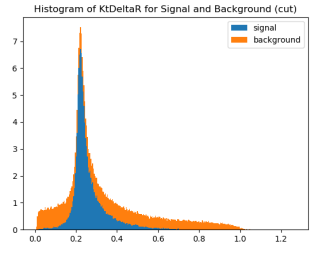


Figure 39

I can't see a feature quite as discriminatory as mass but the other feature that seemed to differentiate itself more after cutting was the ee2 factor. After trying several cuts (below) I couldn't find much of an improvement in significance so decided to stick with only one feature cut for this data set.

2 Part II (Lab 8)

For this part of the lab we'll be looking at a high-luminosity data set and a low-luminosity data set. Here's the set-up:

```
#pseudo data exploration
low = h5py.File('data_lowLumi_pt_1000_1200.h5', 'r')
high = h5py.File('data_highLumi_pt_1000_1200.h5', 'r')
low_data = np.array(low.get('data'))
high_data = np.array(high.get('data'))

low_axis0 = np.array(low.get('data').get('axis0'))
low_axis1 = np.array(low.get('data').get('axis1'))
low_names = np.array(low.get('data').get('block0_items'))
low_values = np.array(low.get('data').get('block0_values'))

hi_axis0 = np.array(high.get('data').get('axis0'))
hi_axis1 = np.array(high.get('data').get('axis1'))
hi_names = np.array(high.get('data').get('block0_items'))
hi_values = np.array(high.get('data').get('block0_values'))

low_mass = np.array([])
hi_mass = np.array([])
for i in range(len(low_values)):
    low_mass = np.append(low_mass, low_values[i][3])
for i in range(len(hi_values)):
    hi_mass = np.append(hi_mass, hi_values[i][3])
```

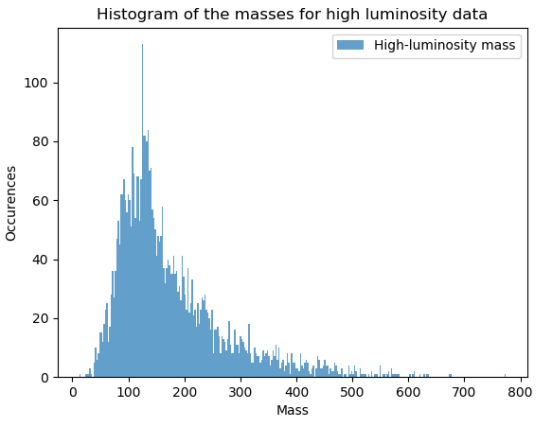
The mass data is stored in two arrays now so we can start exploring what this data looks like. We're only going to be looking at the mass data since we did

not implement any event selection for other features. Note the high luminosity data has 4066 data values while the low luminosity data only has 442. These numbers will be necessary when we need to normalize our distributions from the simulated data set.

2.1 High luminosity data

First let's histogram the data:

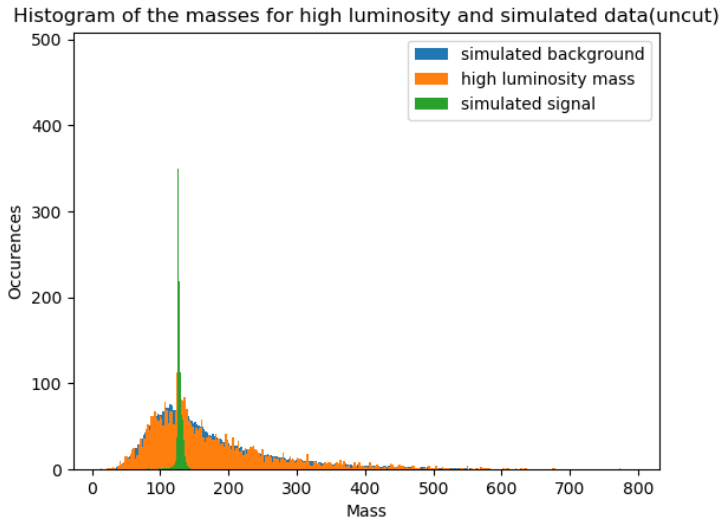
```
figure()
plt.title('Histogram of the masses for high luminosity data')
plt.hist(hi_mass, density=True, bins=300,
label='High-luminosity mass', alpha=0.7)
plt.xlabel('Mass')
plt.ylabel('Occurences')
plt.legend()
plt.show()
```



At first glance we can see that the distribution follows a more gradual slope than the signal data did from the simulated set. First we'll want to compare this histogram with our histograms from our simulated data set. To do this we need to normalize our simulated data with the *observed* yield. Also we need to keep in mind that we should expect $N_{higgs} = 50$ and $N_{qcd} = 2000$. This means we will expect to see 40 times more background data then signal data. In this case we normalize assuming the background has the same expected yield as the observed data and the signal data is normalized with an extra factor of $\frac{1}{40}$. See below for the implementation and the plot:

```
backgweight= (len(hi_mass)/len(signal_dict['mass']))
sigweight = (len(hi_mass)/len(signal_dict['mass']))
```

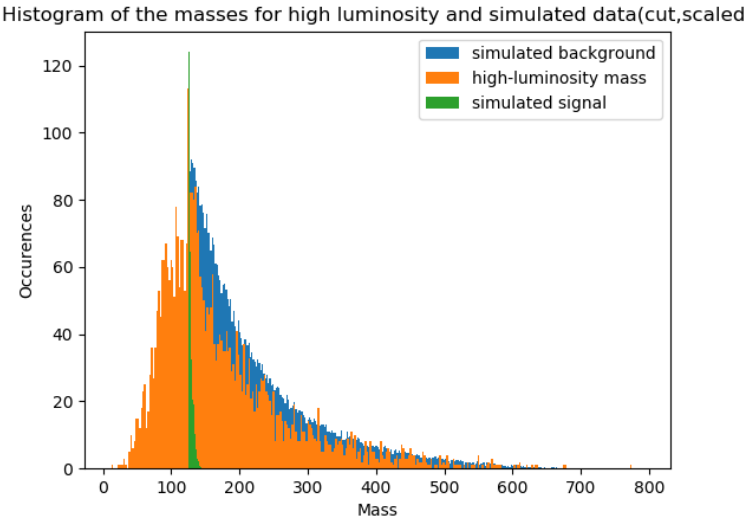
```
figure()
plt.hist(backmass, weights=backgweight*np.ones(len(backmass)),
         label='simulated background', bins=300)
plt.hist(hi_mass, bins=300, label='Low-luminosity mass')
plt.hist(sigmass, weights=sigweight*np.ones(len(sigmass)),
         label='simulated signal', bins=300)
plt.xlabel('Mass')
plt.ylabel('Occurences')
plt.title('Histogram of the masses for high luminosity and
simulated data(uncut)')
plt.legend()
plt.show()
```



Now let's look at the data including event selection:

```
bestcut = 126
sigcut = keepIndices(signal_dict['mass'], bestcut)
backcut = keepIndices(backg_dict['mass'], bestcut)

cutsig = signal_dict['mass']
cutsig = cutsig[sigcut]

cutback = backg_dict['mass']
cutback = cutback[backcut]

backgweight= (len(high_mass)/len(cutback))
sigweight = (len(high_mass)/len(cutsig))
```

```
plt.hist(cutback, weights=backgweight*np.ones(len(cutback)),
label='simulated background', bins=300)
plt.hist(high_mass, bins=300, label='high-luminosity mass')
plt.hist(cutsig, weights=sigweight*np.ones(len(cutsig)),
label='simulated signal', bins=300)
plt.xlabel('Mass')
plt.ylabel('Occurences')
plt.title('Histogram of the masses for high luminosity and
simulated data(cut)')
plt.legend()
plt.show()
```



Now we want to calculate the expected significance. We'll use the same Poisson statistics we used earlier. Here's the implementation:

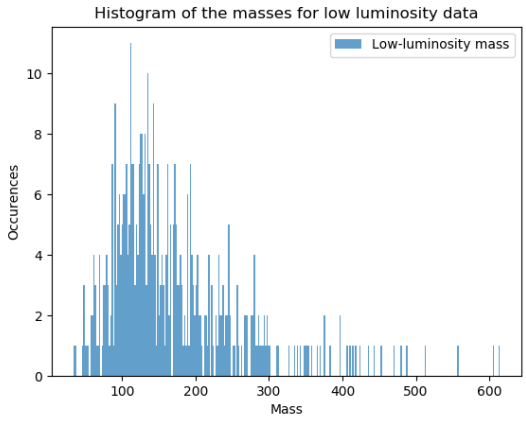
```
cutmean = np.mean(cutback)
val = np.mean(low_mass)
prob = stats.poisson.cdf(val, cutmean)
sig = stats.norm.ppf(1-prob)
print(sig)
```

We get the cut significance as 3.72. This is higher than the uncut significance of 0.12.

2.2 Low luminosity data

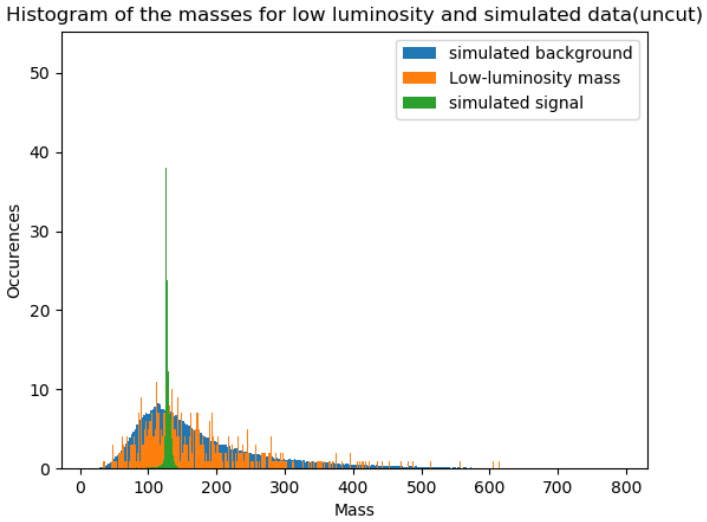
In this section we're going to follow the exact same process but with our low luminosity data. Histogram of the data:

```
figure()
plt.title('Histogram of the masses for low luminosity data')
plt.hist(low_mass, bins=300, label='Low-luminosity mass', alpha=0.7)
plt.xlabel('Mass')
plt.ylabel('Occurences')
plt.legend()
plt.show()
```



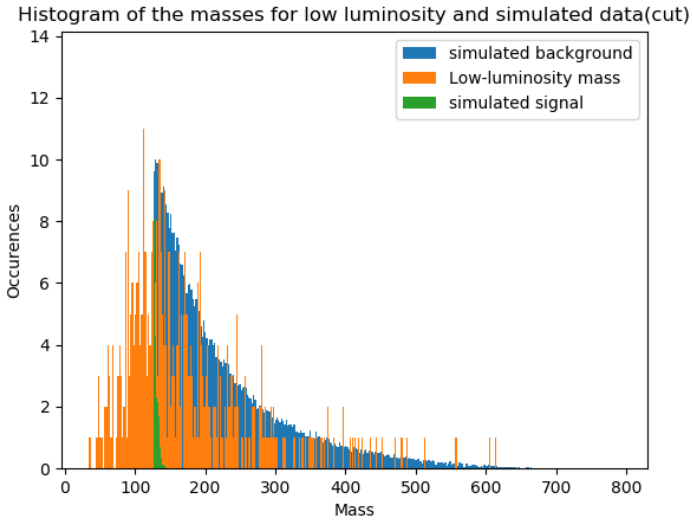
The histogram has a similar shape as the high luminosity data, the only difference is there are far fewer data points in the low luminosity data set. Now we want to plot the same overlapping histograms. First the set without event selection:

```
ratio = 50/2000 #how much more background than signal
backgweight= (len(low_mass)/len(signal_dict['mass']))
sigweight = (len(low_mass)/len(signal_dict['mass']))

figure()
plt.hist(backmass, weights=backgweight*np.ones(len(backmass)),
label='simulated background', bins=300)
plt.hist(low_mass, bins=300, label='Low-luminosity mass')
plt.hist(sigmass, weights=sigweight*np.ones(len(sigmass)),
label='simulated signal', bins=300)
plt.title('Histogram of the masses for low luminosity and simulated data(uncut)'
plt.xlabel('Mass')
plt.ylabel('Occurences')
plt.legend()
plt.show()
```



Now the set with event selection:

```
bestcut = 126
sigcut = keepIndices(signal_dict['mass'], bestcut)
backcut = keepIndices(backg_dict['mass'], bestcut)

cutsig = signal_dict['mass']
cutsig = cutsig[sigcut]

cutback = backg_dict['mass']
cutback = cutback[backcut]

ratio = 50/2000 #how much more background than signal
backgweight= (len(low_mass)/len(cutback))
sigweight = (len(low_mass)/len(cutsig))

plt.hist(cutback, weights=backgweight*np.ones(len(cutback)),
label='simulated background', bins=300)
plt.hist(low_mass, bins=300, label='Low-luminosity mass')
plt.hist(cutsig, weights=sigweight*np.ones(len(cutsig)),
label='simulated signal', bins=300)
plt.xlabel('Mass')
plt.ylabel('Occurences')
plt.title('Histogram of the masses for low luminosity and
simulated data(cut)')
plt.legend()
plt.show()
```



Now we want to calculate the expected significance. We'll use the same Poisson statistics we used earlier. Here's the implementation:

```
cutmean = np.mean(cutback)
val = np.mean(hi_mass)
prob = stats.poisson.cdf(val, cutmean)
sig  = stats.norm.ppf(1-prob)
print(sig)
```

After cutting the expected significance is 3.71 which is higher than the significance before cutting, 0.42.

2.3 95% Confidence Levels of signal yields

To calculate the 95% upper bound on signal yield we'll use the implementation below. The method will be the same for expected and observed we'll just change the original array we feed in.

```
occurences = len(signal_dict['mass'])
sortsig = np.sort(signal_dict['mass'])

count = 0
i=0
while i < occurences:
    x = signal_dict['mass'][i]
    count = count + 1
    if abs(count/occurences - 0.95) < 0.001:
        print(x)
        i = occurences
```

i = i+1

The upper bound we expect from this method is 128 for 95% confidence bound. Now for the observed bound we feed in the low_mass and high_mass arrays. For the low luminosity data the expected bound is 150.04. And for the high luminosity data the bound is 136.89 We can see that for both the observed yield is higher than the expected.