Problem 1

Neutral pions decay into pairs of photons and have an invariant mass of 0.135 GeV (more information, also note that the *charged* pion weights slightly more and does not decay into two photons). Neutral pions are typically measured by combining pairs of photons detected in a single event. There is not tag saying that two particular photons came from a particular pion so all pairs are made and the invariant mass of each pair is histogramed. If the average number of pions in a particular event is N, then the average number of *pairs* in an event goes like N^2; so as N increases the signal (number of photon pairs from a real pion) decreases relative to the background (photon pairs where each photon is from a different pion). The signal pairs have an invariant mass that is peaked and can be described by a peak centered on 0.135 GeV and an area of $N \times S$, where S is the total number of events. The background pairs have a broad distribution in invariant mass distribution without a peak. The total distribution can be described as the sum of these two distributions.

the kinematics of photons from pion decay are available here:

- pi0s_1.txt
- pi0s_5.txt
- pi0s_25.txt

where the in the name "pi0s_N.txt" N is the number of pions per event (so each event contains 2N photons). The columns are labeled as event number, the x, y, z components of the momentum and the energy (both the momentum and energy are in units of GeV).

Useful Tip:

For each pair of photons, calculate their total four-momentum as $P = p_1 + p_2 = \left(E_1 + E_2,\ p_{x1} + p_{x2},\ p_{y1} + p_{y2},\ p_{z1} + p_{z2}\right)$ Then, the invariant mass is computed using the formula

$$m = \sqrt{(E_1 + E_2)^2 - \left[(p_{x1} + p_{x2})^2 + (p_{y1} + p_{y2})^2 + (p_{z1} + p_{z2})^2\right]}$$

Part 1

Using the avialable files here, create *all* pairs of photons with the same event number. This means that you create all possible pairs of photons within the same event and you do this for each event. If the event contains 4 photons there are six photon pairs: AB, AC, AD, BC, BD, BC (the order doesn't matter). Two of these pairs are from real pions and 4 are combinatoric. If you have 1000 events with 4 photons each, you have 2000 real photons and 4000 combinatoric pairs. Some of those combinatoric pairs will have a mass near the pion mass and some will be far away.

You will notice that the files contain different numbers of photons per event and this will change the ratio of pions to background pairs. Histogram the mass distribution with an appropriate binning. Make a plot of the mass distribution for each data set.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
```

In [3]:
```
# Read data, here we use pandas and read the files as dataframes for latter operati
data_1 = pd.read_csv('pi0s_1.txt', sep=r'\s+', header=0, names=['event_number', "px
data_5 = pd.read_csv('pi0s_5.txt', sep=r'\s+', header=0, names=['event_number', "px
data_25 = pd.read_csv('pi0s_25.txt', sep=r'\s+', header=0, names=['event_number', "
```

In [4]:
```
# You can check that for data_1, there are only 1 pions in each event so only two p
data_1
```

Out[4]:

	event_number	px	py	pz	E
0	1	-1.630150	0.242347	0.041822	1.648600
1	1	-1.172580	0.165402	-0.084053	1.187170
2	2	4.034690	-4.959040	-0.412093	6.406290
3	2	0.925626	-1.041470	-0.072998	1.395270
4	3	-8.294120	5.451180	2.778070	10.306600
...	...	...	...	...	...
99995	49998	0.254672	-0.285421	-0.026357	0.383429
99996	49999	0.733212	1.019580	0.181932	1.268950
99997	49999	0.680370	0.783644	0.072178	1.040290
99998	50000	13.943600	-9.246750	-0.760801	16.748300
99999	50000	1.359950	-0.932515	-0.108837	1.652540

100000 rows × 5 columns

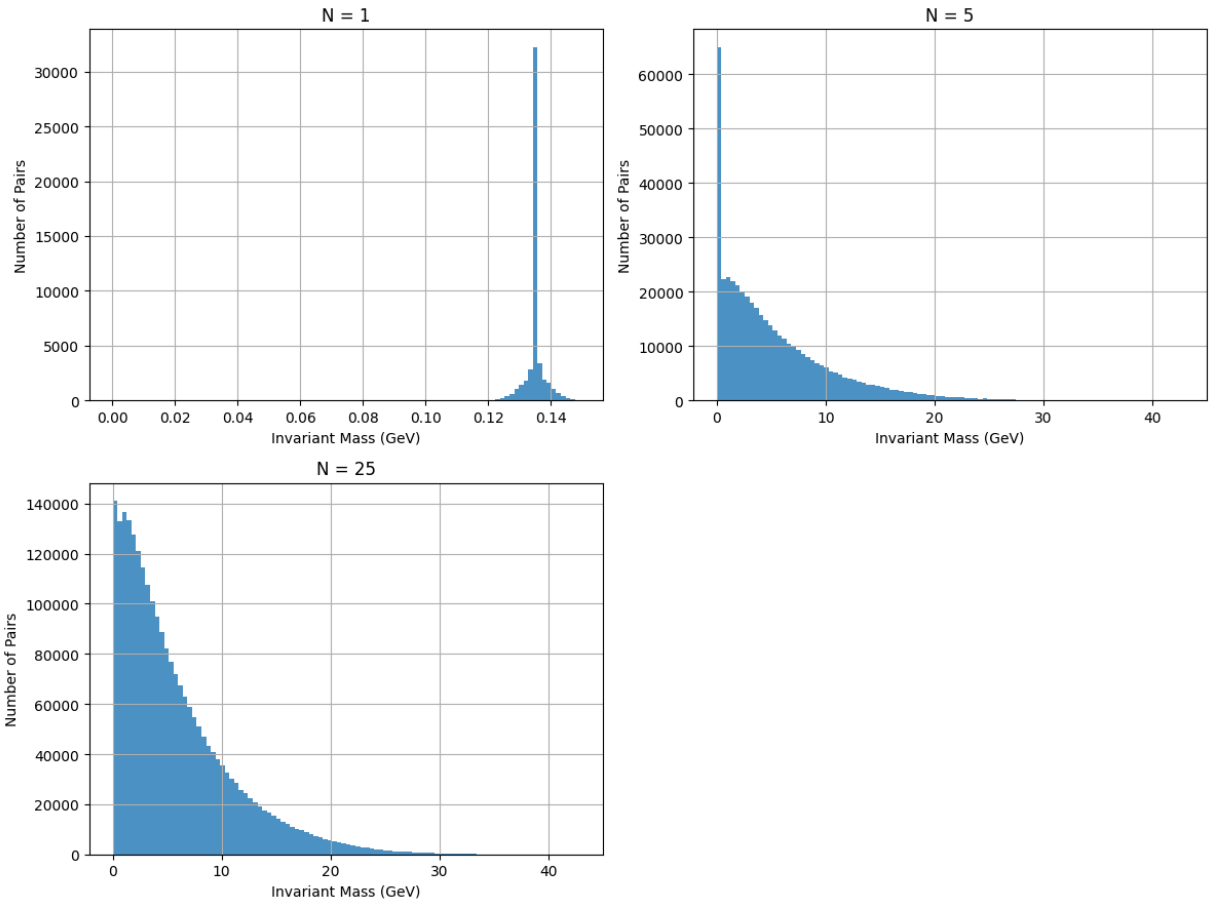
Define a function to calculate the invairant masses of possible photon pairs. Hint: Use "groupby" to group the data by *event_number*, and for entries with the same *event_number*, use "np.triu_indices" to avoid slow for loop and get possible combinators

In [5]:
```
from itertools import combinations

def calculate_invariant_mass_pairs(data):
    """
    Calculate the invariant mass for each photon pair in the dataset.

    Parameters
```

```
    ----------
    data : pandas.DataFrame
        DataFrame containing photon data with columns: 'event_number', 'E', 'px', '

    Returns
    -------
    mass_list : list of floats
        List of invariant mass values for each photon pair.
    """
    mass_list = []

    # Implement your code below
    # Hint: use "groupby" and group the dataframe by event_number

    for _, group in data.groupby('event_number'):
        photons = group[['E', 'px', 'py', 'pz']].values
        for (E1, px1, py1, pz1), (E2, px2, py2, pz2) in combinations(photons, 2):
            mass_squared = (E1 + E2)**2 - ((px1 + px2)**2 + (py1 + py2)**2 + (pz1 +
            if mass_squared >= 0:
                mass_list.append(np.sqrt(mass_squared))

    return mass_list
```

In [6]:
```
masses_1 = calculate_invariant_mass_pairs(data_1)
masses_5 = calculate_invariant_mass_pairs(data_5)
masses_25 = calculate_invariant_mass_pairs(data_25)
```

In [7]:
```
# Plot the invariant mass histogram
plt.figure(figsize=(12, 9))

plt.subplot(2, 2, 1)
plt.hist(masses_1, bins=100, alpha=0.8)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 1')
plt.grid(True)

plt.subplot(2, 2, 2)
plt.hist(masses_5, bins=100, alpha=0.8)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 5')
plt.grid(True)

plt.subplot(2, 2, 3)
plt.hist(masses_25, bins=100, alpha=0.8)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 25')
plt.grid(True)

plt.tight_layout()
plt.show()
```



Part 2

Look at the histograms. If the mass of a photon pair is far from the pion mass, it probably didn't come from a pion. However, if the mass is near the pion mass, either it came from a pion or it is a combinatoric pair.

Find a function to fit the invariant mass distribution. What is the signal to background in the region of the pion mass (answering this question requires defining a region around the pion mass to calculate your signal; make sure you provide some justification for your definition).

Requirement: state your signal and background model, plot the invariant mass distribution and fit result, clarify your region definition, and give signal to background ratio in that region.

Hint:

1. If gaussian cannot describe the peak well, try Breit-Wigner: $pdf(m) = A\frac{\Gamma^2}{(m-m_0)^2+\Gamma^2/4}$
2. For background a linear model should be sufficient.
3. Frist, define a fit_function first, which is pdf(signal)+pdf(backgrond)
4. Then define a χ^2 function you learned from lecture

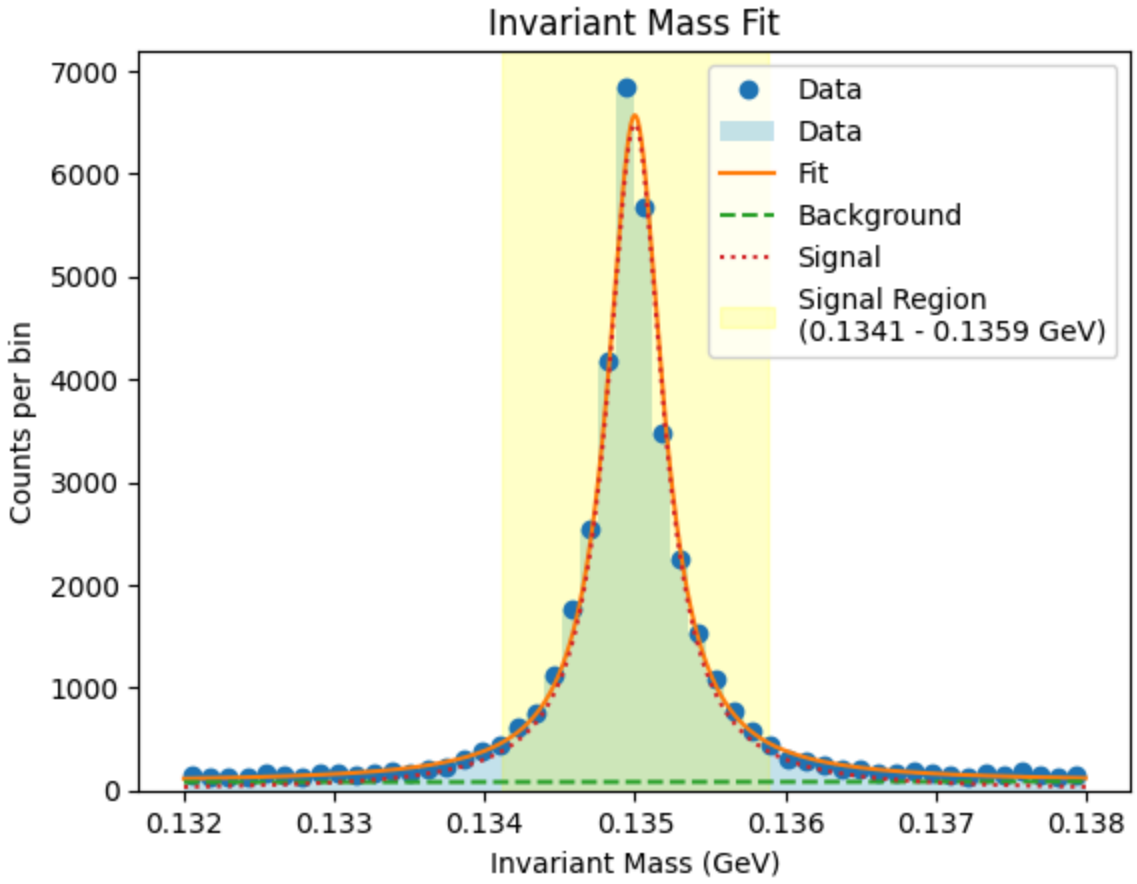
5. Use iminuit and fit the data with the fit_function, minimizing the χ^2
6. With the fitted parameters, calculate the number of backgrounds within the region you defined, and subtract it from actual counts to get the number of signals, then signal to background ratio is easy

In [28]:
```
from iminuit import Minuit
from scipy.special import erf

def fit_signal_background(mass_list, mass_min, mass_max, n_bins):
    """
    Fit the invariant mass histogram using a Breit-Wigner signal plus linear backgr
    and return the estimated signal and background counts in the signal region.
    """

    # Implement your code below
    # Create the histogram from mass_list
    counts, bin_edges = np.histogram(mass_list, bins=n_bins, range=(mass_min, mass_
    bin_width = (bin_edges[-1] - bin_edges[0]) / n_bins
    bin_centers = 0.5 * (bin_edges[1:] + bin_edges[:-1])
    # print(len(bin_centers))
    # print(bin_centers)

    # Define the fit function
    def fit_function(m, A, m0, gamma, a, b):
        signal = A * gamma**2 / ((m - m0)**2 + gamma**2/4)
        background = a + b * (m - m0)
        return signal + background

    # Define the chi-square function to minimize.
    def chi2(A, m0, gamma, a, b):
        expected = fit_function(bin_centers, A, m0, gamma, a, b) * bin_width
        mask = counts > 0
        return np.sum(((counts[mask] - expected[mask])**2) / counts[mask])

    # Implement the Minuit fitting below
    m = Minuit(chi2, A=np.max(counts) * bin_width, m0=0.135, gamma=0.01, a=np.mean(
    m.limits["A"] = (0, None)
    m.limits["gamma"] = (0, None)
    m.limits["a"] = (0, None)
    m.fixed["m0"] = True
    m.migrad()
    A_fit, m0_fit, gamma_fit, a_fit, b_fit = m.values["A", "m0", "gamma", "a", "b"]

    # Calculate Signal and Background counts in the signal region
    signal_region_min = m0_fit - 2 * gamma_fit
    signal_region_max = m0_fit + 2 * gamma_fit
    signal_region = (bin_centers >= signal_region_min) & (bin_centers <= signal_reg

    all_signals = np.sum(counts[signal_region])
    backgrounds = np.sum(fit_function(bin_centers[signal_region], 0, m0_fit, gamma_
    signals = all_signals - backgrounds

    m_fine = np.linspace(mass_min, mass_max, 1000)
```

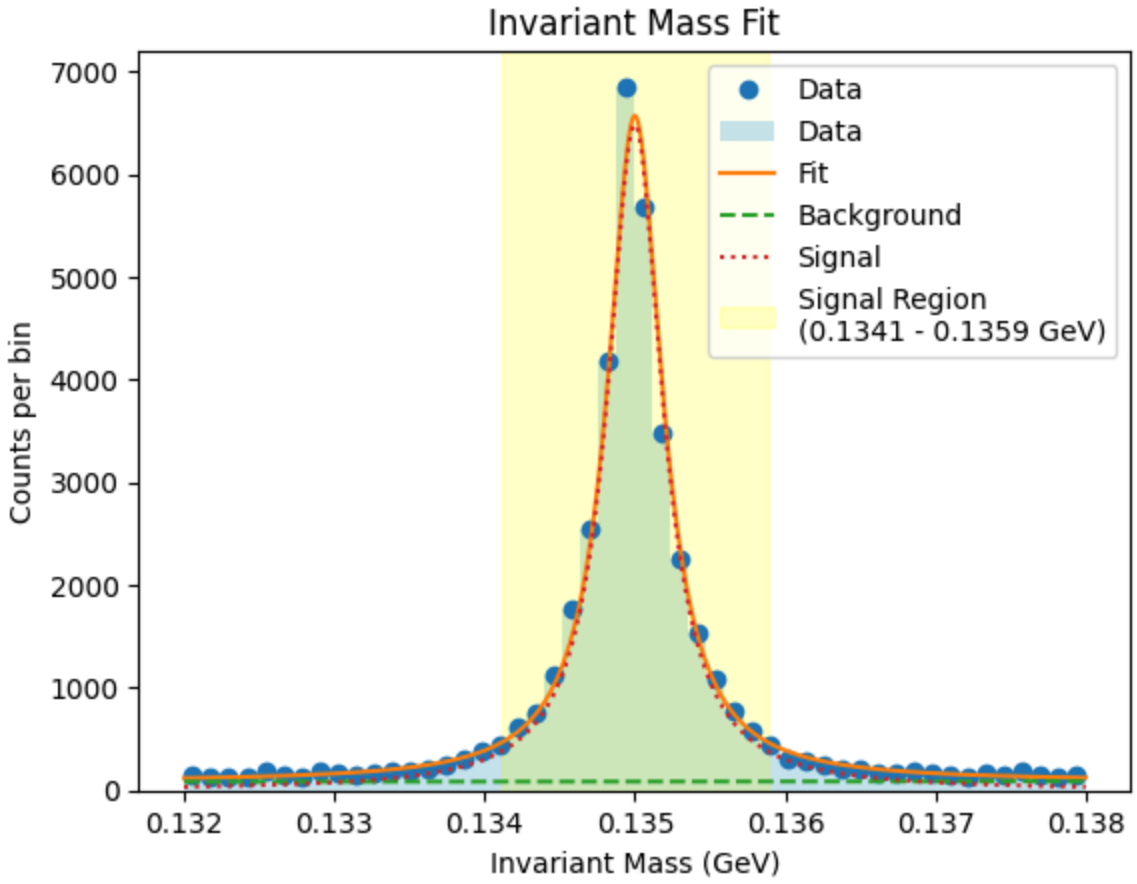
```
    plt.plot(bin_centers, counts, 'o', label='Data')
    plt.hist(mass_list, bins=n_bins, range=(mass_min, mass_max), alpha=0.7,
             label='Data', color='lightblue', density=False)
    plt.plot(m_fine, fit_function(m_fine, A_fit, m0_fit, gamma_fit, a_fit, b_fit) *
    plt.plot(m_fine, fit_function(m_fine, 0, m0_fit, gamma_fit, a_fit, b_fit) * bin
    plt.plot(m_fine, fit_function(m_fine, A_fit, m0_fit, gamma_fit, 0, 0) * bin_wid
    plt.axvspan(signal_region_min, signal_region_max, alpha=0.2, color='yellow',
                label=f'Signal Region\n({signal_region_min:.4f} - {signal_region_ma
    plt.xlabel('Invariant Mass (GeV)')
    plt.ylabel('Counts per bin')
    plt.title('Invariant Mass Fit')
    plt.legend()
    # plt.yscale('log')
    plt.show()

    print(f"Fit results:")
    print(f"A = {A_fit}")
    print(f"m0 = {m0_fit}")
    print(f"gamma = {gamma_fit}")
    print(f"a = {a_fit}")
    print(f"b = {b_fit}")

    return signals, backgrounds
```

Our signal model follows the Breit-Wigner distribution, and our background model is linear as specified. We use a region definition of +/- our fitted gamma value, as this covers a reasonable spread (83%) of the Breit-Wigner distribution. Additionally, if the signal truly followed a Gaussian distribution, this would also cover a reasonable spread (95%) to identify our signal/background pions.

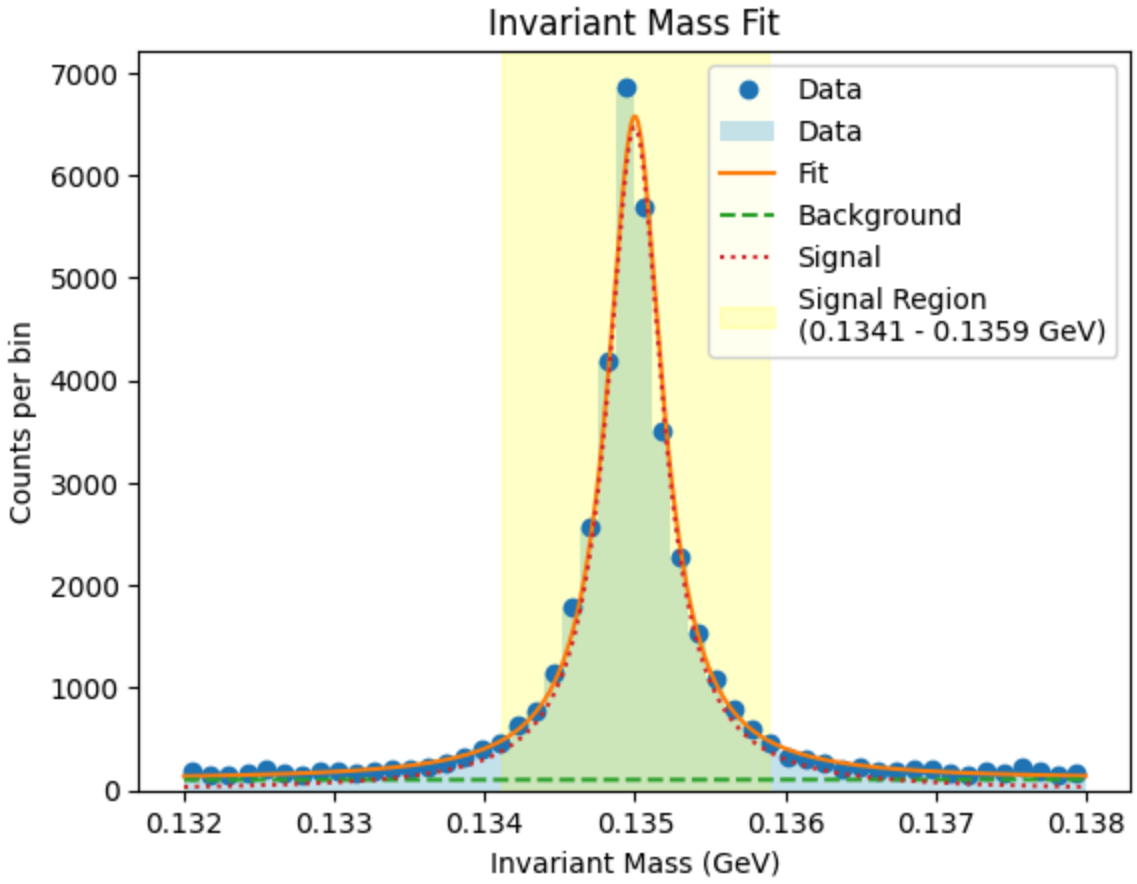
In [29]:
```
mass_dict = {'N=1': masses_1,
             'N=5': masses_5,
             'N=25': masses_25
            }
for key in mass_dict.keys():
    S, B = fit_signal_background(mass_dict[key], 0.132, 0.138, 50)
    print (f'{key}, S=%.2f, B=%.2f, S/B=%.2f, uncertainty=%.4f%%'%(S, B, S/B, 100*n
```




```
Fit results:
A = 13510636.817696448
m0 = 0.135
gamma = 0.00044289652788331857
a = 690607.3713048493
b = 7333726.194268326
N=1, S=32027.78, B=1160.22, S/B=27.60, uncertainty=0.5688%
```




```
Fit results:
A = 13506643.66132939
m0 = 0.135
gamma = 0.0004429318665997162
a = 720336.7840025837
b = 5856624.78469176
N=5, S=32021.83, B=1210.17, S/B=26.46, uncertainty=0.5693%
```




```
Fit results:
A = 13483529.502568156
m0 = 0.135
gamma = 0.0004441746149872472
a = 871131.6890654316
b = 4604274.024694325
N=25, S=32027.50, B=1463.50, S/B=21.88, uncertainty=0.5714%
```

Part 3

Find the number of pions in each of the histograms (making sure to remove the background). How does the precision of the number of pions vary with the signal to background? Explain what you find.

From the output of the previous function:

```
N=1, S=32027.78, B=1160.22, S/B=27.60, uncertainty=0.5688%
N=5, S=32021.83, B=1210.17, S/B=26.46, uncertainty=0.5693%
N=25, S=32027.50, B=1463.50, S/B=21.88, uncertainty=0.5714%
```

We can see as S/B increases, uncertainty decreases, or in other terms, precision increases. This makes sense as if the majority of pions are from the signal, it is easier to be sure that these pions differ from the background.