In [114…

```
import pandas as pd
import numpy as np


# Read the files
df_a = pd.read_csv('/Users/ayeshaferoz/Documents/Res=35k,noise=0/Files/charge separate columnsZ.csv')  # Replace 'file_a
df_b = pd.read_csv('/Users/ayeshaferoz/Documents/Res=35k,noise=1e5/adder.csv')  # Replace 'file_b.csv' with the actual 

# Filter entries in file b where DummyIndex=0
df_b_filtered = df_b[df_b['DummyIndex'] == 0]

# Define a function to calculate the mass difference in ppm
def calculate_ppm_diff(mass_a, mass_b):
    return abs(mass_a - mass_b) / mass_a * 1e6

# Initialize a list to store true positives
true_positives = []

# Iterate over each row in file b and compare with file a
for index_b, row_b in df_b_filtered.iterrows():
    mass_b = row_b['MonoisotopicMass']
    scan_b = row_b['ScanNum']

    # Filter entries in file a based on scan number
    df_a_filtered = df_a[df_a['ScanNum'] == scan_b]

    # Check for matching entries within ppm tolerance
    for index_a, row_a in df_a_filtered.iterrows():
        mass_a = row_a['MonoisotopicMass']
        ppm_diff = calculate_ppm_diff(mass_a, mass_b)

        if ppm_diff <= 10:
            true_positives.append(mass_a)  # Store the MonoisotopicMass values of the matching entries

# Print the true positives
#for mass in true_positives:
    #print("True positive: MonoisotopicMass =", mass)
```

In [114…

```
# Filter entries in file b where DummyIndex=0
df_b_filtered = df_b[df_b['DummyIndex'] == 0]
```

```
# Define a function to calculate the mass difference in ppm
def calculate_ppm_diff(mass_a, mass_b):
    return abs(mass_a - mass_b) / mass_a * 1e6

# Initialize lists to store true positives and false positives
true_positives = []
false_positives = []

# Iterate over each row in file b and compare with file a
for index_b, row_b in df_b_filtered.iterrows():
    mass_b = row_b['MonoisotopicMass']
    scan_b = row_b['ScanNum']

    # Filter entries in file a based on scan number
    df_a_filtered = df_a[df_a['ScanNum'] == scan_b]

    # Check for matching entries within ppm tolerance
    match_found = False
    for index_a, row_a in df_a_filtered.iterrows():
        mass_a = row_a['MonoisotopicMass']
        ppm_diff = calculate_ppm_diff(mass_a, mass_b)

        if ppm_diff <= 10:
            true_positives.append(mass_a)  # Store the MonoisotopicMass values of the matching entries
            match_found = True
            break

    if not match_found:
        false_positives.append(mass_b)  # Store the MonoisotopicMass values of the non-matching entries



# Print the false positives
#print("\nFalse positives:")
#for mass in false_positives:
    #print("MonoisotopicMass =", mass)
```

In [114…

```
# Print the number of true positives and false positives
print("Number of true positives:", len(true_positives))
print("Number of false positives:", len(false_positives))
```

```
Number of true positives: 589
Number of false positives: 697
```

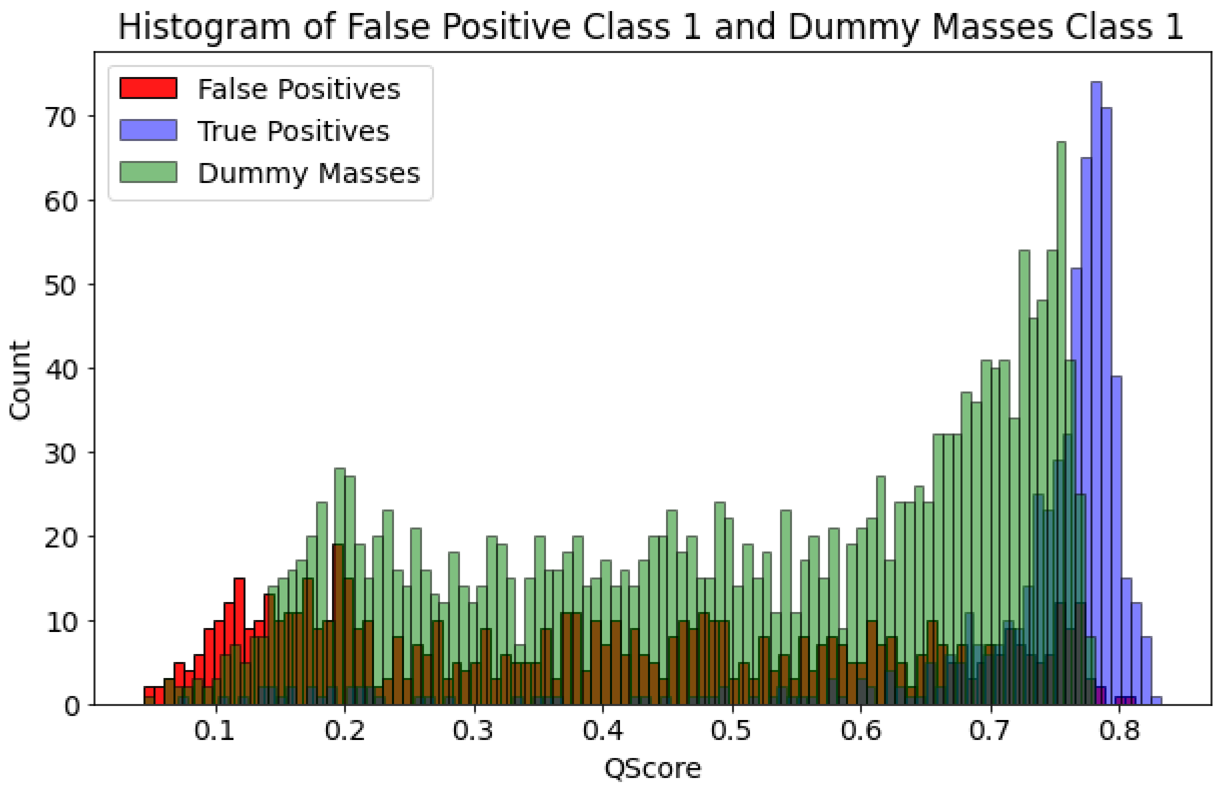
In [114…

```python
# Filter entries in file b where DummyIndex > 0
df_b_dummy = df_b[df_b['DummyIndex'] > 0]
dummy_masses = df_b_dummy['QScore'].tolist()  # Store the QScore values of the DummyMasses
```

In [114…

```python
# Filter entries in file b where DummyIndex=0
df_b_filtered = df_b[df_b['DummyIndex'] == 0]

# Define a function to calculate the mass difference in ppm
def calculate_ppm_diff(mass_a, mass_b):
    return abs(mass_a - mass_b) / mass_a * 1e6

# Initialize lists to store true positives, false positives, and DummyMasses with DummyIndex > 0
true_positives = []
false_positives = []
dummy_masses = []

# Iterate over each row in file b and compare with file a
for index_b, row_b in df_b_filtered.iterrows():
    mass_b = row_b['MonoisotopicMass']
    scan_b = row_b['ScanNum']
    qscore_b = row_b['QScore']

    # Filter entries in file a based on scan number
    df_a_filtered = df_a[df_a['ScanNum'] == scan_b]

    # Check for matching entries within ppm tolerance
    match_found = False
    for index_a, row_a in df_a_filtered.iterrows():
        mass_a = row_a['MonoisotopicMass']
        ppm_diff = calculate_ppm_diff(mass_a, mass_b)

        if ppm_diff <= 10:
            true_positives.append({'QScore': qscore_b})  # Store the QScore values of the true positives
            match_found = True
            break

    if not match_found:
        false_positives.append({'QScore': qscore_b})  # Store the QScore values of the false positives
```

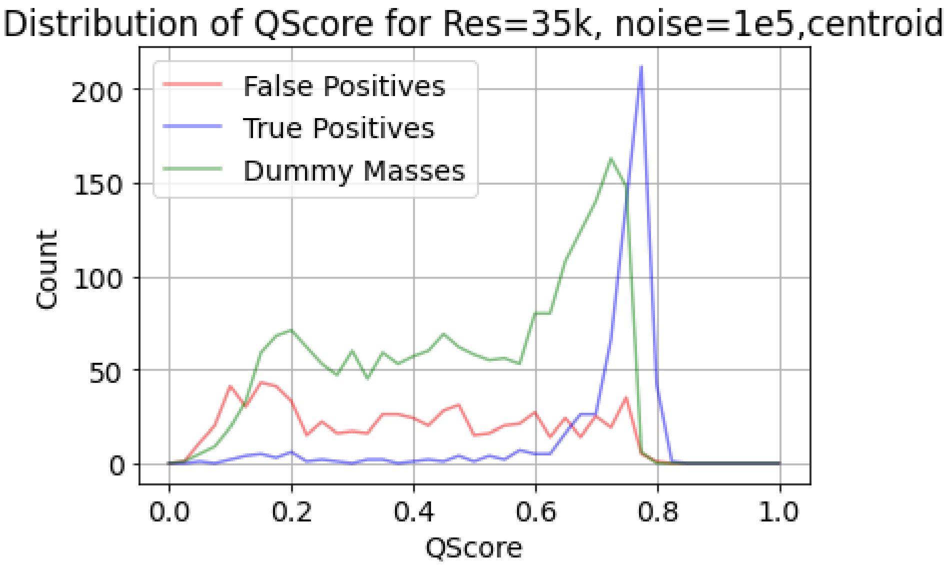
```
# Filter entries in file b where DummyIndex > 0
df_b_dummy = df_b[df_b['DummyIndex'] > 0]
dummy_masses = df_b_dummy['QScore'].tolist()  # Store the QScore values of the DummyMasses

# Convert lists to DataFrames
true_positives_df = pd.DataFrame(true_positives)
false_positives_df = pd.DataFrame(false_positives)
dummy_masses_df = pd.DataFrame({'QScore': dummy_masses})

# Plot the histograms
plt.figure(figsize=(10, 6))
plt.hist(false_positives_df['QScore'], bins=100, alpha=0.9, label='False Positives', color='red', edgecolor='black')
plt.hist(true_positives_df['QScore'], bins=100, alpha=0.5, label='True Positives', color='blue', edgecolor='black')
plt.hist(dummy_masses_df['QScore'], bins=100, alpha=0.5, label='Dummy Masses', color='green', edgecolor='black')
plt.xlabel('QScore')
plt.ylabel('Count')
plt.title('Histogram of False Positive Class 1 and Dummy Masses Class 1')
plt.legend()
plt.show()
```



In [114…

```
# Create arrays of QScores for false positives, true positives, and dummy masses
false_positive_scores = false_positives_df['QScore'].values
true_positive_scores = true_positives_df['QScore'].values
dummy_masses_scores = dummy_masses_df['QScore'].values

# Define the bins for the histogram
bins = np.arange(0, 1.05, 0.025)

# Create a line plot for false positives
plt.plot(bins[:-1], np.histogram(false_positive_scores, bins=bins)[0], color='red', alpha=0.5)

# Create a line plot for true positives
plt.plot(bins[:-1], np.histogram(true_positive_scores, bins=bins)[0], color='blue', alpha=0.5)

# Create a line plot for dummy masses
plt.plot(bins[:-1], np.histogram(dummy_masses_scores, bins=bins)[0], color='green', alpha=0.5)
```

```
# Add axis labels, title, legend, and grid to the plot
plt.xlabel('QScore')
plt.ylabel('Count')
plt.title('Distribution of QScore for Res=35k, noise=1e5,centroid')
plt.grid(True)
plt.legend(['False Positives', 'True Positives', 'Dummy Masses'], loc='upper left')

# Display the plot
plt.show()
```



In [114…

```
# Filter entries in file b where DummyIndex = 0
df_b_filtered = df_b[df_b['DummyIndex'] == 0]

# Define a function to calculate the mass difference in ppm
def calculate_ppm_diff(mass_a, mass_b):
    return abs(mass_a - mass_b) / mass_a * 1e6

# Initialize lists to store true positives and false positives
true_positives = []
false_positives = []

# Iterate over each row in file b and compare with file a
for index_b, row_b in df_b_filtered.iterrows():
    mass_b = row_b['MonoisotopicMass']
```

```
        scan_b = row_b['ScanNum']

        # Filter entries in file a based on scan number
        df_a_filtered = df_a[df_a['ScanNum'] == scan_b]

        # Check for matching entries within ppm tolerance
        match_found = False
        for index_a, row_a in df_a_filtered.iterrows():
            mass_a = row_a['MonoisotopicMass']
            ppm_diff = calculate_ppm_diff(mass_a, mass_b)

            if ppm_diff <= 10:
                true_positives.append(mass_a)  # Store the MonoisotopicMass values of the matching entries
                match_found = True
                break

        if not match_found:
            false_positives.append(mass_b)  # Store the MonoisotopicMass values of the non-matching entries

    # Divide false positives into three classes based on conditions
    fp1 = []
    fp2 = []
    fp3 = []

    for mass_b in false_positives:
        scan_b = df_b[df_b['MonoisotopicMass'] == mass_b]['ScanNum'].values[0]

        # Filter entries in file a based on scan number
        df_a_filtered = df_a[df_a['ScanNum'] == scan_b]

        mass_a = df_a_filtered['MonoisotopicMass'].values[0]
        z_a = df_a_filtered['MinCharge'].values[0]
        z_b = row_b['MinCharge']

        mass_diff1 = (mass_b / z_b) - (mass_a / z_a)
        mass_diff2 = mass_b - mass_a

        if mass_diff1 <= 0.01:
            fp1.append(mass_b)
        elif mass_diff2 <= 2.0:
            fp3.append(mass_b)
        else:
            fp2.append(mass_b)
```

```python
# Print the false positives in each class
print("False positives:")
print("fp1 (Mass difference <= 0.1):", fp1)
print("fp2 (Mass difference > 0.1 and <= 2.2):", fp2)
print("fp3 (Mass difference > 2.2):", fp3)
```

```
False positives:
fp1 (Mass difference <= 0.1): [1146.485053, 4893.735543, 5074.070899, 5364.954631, 2682.527234, 3925.468258, 5652.01241
6, 2301.010936, 2674.731092, 3903.054697, 6620.521171, 3086.898652, 3936.279443, 2304.790069, 2896.790959, 4786.921224,
6585.377197, 2445.732546, 2640.678068, 4843.4793, 6064.416129, 3674.634381, 2293.490793, 5657.635085, 6721.608274, 5570.
794902, 6182.122761, 5228.663658, 2589.049055, 4454.121096, 5113.327112, 6720.16017, 4344.782853, 3619.738042, 6906.2384
3, 2219.380495, 4033.88894, 3425.616553, 4240.255949, 2204.45432, 2449.479916, 5227.820817, 2215.062148, 3951.688526, 59
46.23497, 3495.561353, 2996.190111, 4883.765808, 4071.585765, 1206.792342, 1136.990679, 1152.825225, 3804.58332, 4010.60
2386, 1986.700591, 2624.460221, 2528.413332, 3806.881404, 5606.003087, 7267.470405, 1351.174931, 2523.791496, 3760.2776
1, 4139.571238, 5488.047702, 5590.654983, 1140.930672, 2405.666946, 6744.162019, 6814.563739, 2648.783829, 5422.610965,
5285.086968, 2368.295163, 3166.275204, 6383.50695, 3079.279578, 6235.478774, 7066.276678, 6906.090911, 6253.350478, 274
6.843031, 2761.151259, 3957.291123, 1164.59779, 2400.6526, 2802.074221, 2301.744545, 4238.86391, 2178.543105, 6229.28765
8, 2133.305866, 2673.829064, 2265.993701, 3103.53532, 7230.100046, 3945.675661, 4189.309607, 5690.211858, 6582.461937, 3
546.001301, 841.409982, 3555.945429, 2410.134779, 6322.945773, 1065.449906, 1339.670401, 1339.757814, 3653.53951, 5068.5
96078, 959.25636, 1277.962287, 5869.433918, 2156.930851, 2784.678943, 6894.265466, 5895.722651, 4160.277858, 4489.89979
7, 6733.624087, 5628.464703, 4037.23732, 5322.155143, 6737.233826, 1330.259903, 2232.814319, 2971.942835, 1249.312622, 2
609.84979, 3017.121615, 4828.137533, 3513.648263, 3073.084212, 3961.568495, 4009.77683, 2268.773668, 2649.523642, 3417.1
58372, 5256.873385, 2333.417031, 3805.291708, 6257.386182, 4381.163923, 7146.917352, 3841.989684, 5257.65182, 1064.06918
6, 4258.760345, 1059.492464, 1347.413348, 4022.678555, 2720.574627, 3358.151013, 5509.736289, 6034.81897, 2260.975213, 5
676.324797, 2208.768149, 3984.345411, 5548.05239, 7096.010239, 1120.868734, 1974.938695, 2066.550582, 5654.677122, 2681.
113963, 2701.76557, 4817.280577, 6442.598225, 2667.31693, 5701.715055, 2233.612604, 1104.851044, 1358.372781, 2970.3417
6, 3807.939183, 5186.804848, 3583.224056, 3520.790942, 1689.489504, 2221.938088, 3842.545083, 4023.943727, 2414.364666,
5698.197242, 2642.598758, 3124.821948, 1763.315114, 1581.837066, 4383.071105, 5709.676445, 1217.024029, 4587.620843, 393
2.100839, 1112.727976, 4660.937173, 1961.862176, 5595.564163, 3835.09573, 2940.726944, 3738.321652, 862.359822, 2374.790
428, 4773.783589, 2210.0924, 2453.542704, 3079.529039, 3631.600678, 3839.329349, 6725.231904, 806.097846, 4513.129773, 2
463.143843, 5238.012417, 3716.433209, 7006.038417, 3186.709165, 5327.809957, 2957.257099, 3014.137014, 6559.825532, 652
4.384792, 3390.813982, 4624.86422, 5895.244134, 6760.780719, 1531.275273, 5682.329052, 2209.987066, 7192.309374, 1362.90
8155, 3471.330075, 8464.263657, 5856.919849, 6732.932775, 1116.021869, 4308.300322]
fp2 (Mass difference > 0.1 and <= 2.2): [22084.453501, 24984.193215, 39752.743903, 39757.346904, 36004.885883, 37296.880
879, 37299.693369, 35729.077454, 27400.778821, 33538.634629, 34753.950646, 31477.744548, 35675.608013, 26143.961388, 334
84.385708, 35911.19328, 36807.503166, 36810.388162, 32449.536161, 35828.848966, 35833.007594, 28220.443735, 28500.42412
1, 36051.089443, 30504.454353, 38188.334358, 76385.757045, 20363.294424, 25404.740389, 50576.072343, 26213.312561, 3934
3.327774, 21257.812188, 26715.145791, 35538.368862, 20715.697288, 35469.02093, 35473.238376, 19236.617788, 21322.66574,
20521.31839, 26753.277822, 20007.713508, 11599.340008, 24325.071885, 25339.141433, 35470.321548, 11622.179358, 21497.079
672, 22886.944779, 26962.073434, 25582.282324, 32892.184823, 25402.722392, 30545.811642, 26715.151426, 39745.682195, 429
98.385468, 43003.370173, 41929.778553, 16072.168132, 16092.585978, 21663.20219, 38116.593998, 26787.113265, 31921.69928
4, 32745.028789, 31963.413895, 34537.399322, 41088.645746, 33481.168581, 22718.417972, 28499.421785, 30504.452184, 1659
3.053105, 13055.759382, 15339.005151, 17286.207929, 30911.3833, 26753.27904, 35471.105315, 27636.744643, 32340.670595, 3
2344.648035, 33936.373059, 34572.592201, 28411.884021, 33537.096677, 36664.086983, 24336.603281, 24339.598064, 35045.508
```

```
403, 35227.760682, 39437.384421, 30343.136306, 29406.436485, 33616.789625, 33852.524952, 28387.954435, 29003.705431, 309
36.584293, 31303.182158, 39667.295758, 29714.356919, 40037.316183, 42555.335583, 34690.959056, 37622.989201, 37627.08604
1, 46279.155958, 37174.865388, 37798.097503, 38433.861787, 48781.185548, 49252.30251, 49254.843099, 70461.989734, 24832.
051479, 36256.611429, 43547.846789, 43549.932867, 32647.930461, 34365.567657, 34779.879482, 48320.159026, 67910.354569,
30544.814605, 35022.612904, 38676.671143, 76612.772559, 49597.309714, 35473.406439, 73853.241162, 32483.05405, 58323.691
625, 27801.037187, 25583.443903, 26545.537659, 26215.327367, 8751.09309, 27505.676611, 12342.6283, 26428.736251, 27589.1
64681, 8059.942215, 21117.962031, 26143.977491, 27618.740589, 8243.334368, 12881.132975, 10235.502889, 10629.891364, 255
82.413048, 26121.303431, 26753.273475, 26712.684338, 35470.96453, 25083.507197, 38477.881038, 25837.851666, 27976.46394
3, 38236.78566, 24454.329403, 33377.251664, 44576.450509, 44579.587566, 44581.987105, 26091.561435, 27290.736423, 37393.
091743, 41337.929008, 17233.053475, 27680.201539, 41039.894133, 10471.987514, 36110.76489, 36112.793828, 36541.700498, 5
6984.406613, 30930.405901, 37603.631331, 36342.163764, 28707.086757, 33552.734605, 36490.18524, 71745.222096, 28627.0630
6, 37413.363491, 31876.115782, 33101.129626, 39309.685084, 44920.016611, 32497.551257, 36195.933367, 44148.973326, 4430
6.334316, 35912.330563, 39508.996489, 30400.860271, 31139.015491, 36417.321358, 34097.462671, 34100.617804, 34102.63558
2, 45870.206116, 34485.958791, 27208.972818]
fp3 (Mass difference > 2.2): [8791.605561, 10047.250765, 25533.381616, 34902.346139, 38192.61183, 21681.175024, 25782.07
2833, 7420.376189, 9762.998665, 25966.208573, 33241.596555, 37149.77893, 25680.252073, 26979.241181, 44135.728496, 1375
5.13605, 22782.57396, 8354.03223, 23185.192999, 31041.792789, 35329.785766, 35332.529943, 15758.504205, 17736.62229, 177
38.629797, 28854.61633, 46278.663961, 26700.479112, 12546.44223, 15615.399972, 21390.018628, 34873.572654, 8362.414015,
9813.561425, 31587.255708, 32713.747499, 45512.465276, 21550.178403, 28989.56468, 33985.95594, 20952.78542, 21059.89839,
22352.929458, 24554.003642, 9067.821869, 11057.804343, 9208.244673, 12006.450779, 24979.449005, 27618.749563, 17738.6553
94, 7481.109072, 8223.839684, 8975.971275, 7750.152726, 13674.535767, 8576.708989, 8595.530712, 10564.182364, 7413.51677
3, 8625.309634, 10612.436435, 16940.29237, 34362.51836, 45914.607671, 9476.158174, 20694.831756, 21390.027568, 9538.8149
98, 15591.079062, 19166.565647, 20628.0466, 7447.299115, 20883.441356, 34778.12632, 34781.654562, 9794.467965, 28854.601
683, 12474.140532, 16631.694906, 24326.639119, 11733.506923, 22352.944541, 9919.906062, 17957.366009, 11155.829657, 830
5.380741, 7581.326434, 8157.770514, 27618.737001, 8170.250305, 10216.862791, 19868.459106, 39013.545439, 39016.426234, 4
0534.10915, 50546.608088, 7484.814927, 7738.895564, 7530.697229, 20963.759547, 30619.822339, 31447.24843, 20525.203958,
26700.474288, 30534.763829, 44505.617856, 16365.536505, 17894.100954, 23301.962756, 22522.872749, 33313.013411, 35471.96
5223, 21874.178727, 33374.642545, 43547.456559, 49506.054383, 7433.962067, 22437.143962, 10135.73866, 18828.597358, 2111
2.172759, 45181.984794, 45185.323381, 49261.618018, 29297.224531, 33420.195065, 33422.242917, 35375.77755, 41961.111628,
41963.562323, 30401.655518, 32319.958275, 33986.557004, 9309.969638, 17391.545144, 13844.761392, 14468.102589, 8502.4983
79, 44790.783777, 10825.028859, 38307.415556, 14559.499181, 9153.807078, 22838.93288, 17051.462251, 12034.980268, 34780.
257295, 25083.518504, 35469.282607, 35473.183151, 7591.551048, 21567.057968, 25848.27415, 26211.766168, 26962.999876, 72
74.54043, 7328.142383, 25081.665874, 11600.351424, 23674.695808, 28217.425293, 28219.439818, 7313.771313, 24444.231536,
21322.614478, 25403.750417, 15052.641562, 17540.09662, 11554.350274, 15591.125524, 34780.930921, 12402.92274, 8275.83584
7, 12740.504161, 9050.183617, 12096.327374, 12288.544297, 7527.424589, 27618.727294, 8431.657751, 22228.877348, 26540.89
5801, 8361.354896, 11676.651112, 23348.302729, 32117.454341, 35066.351197, 36831.919709, 10775.158047, 16136.410967, 731
6.694973, 10287.949239, 18066.424883, 29564.586371, 29883.044145, 31493.18964, 32845.102128, 48518.326141, 21320.849328,
27256.471649, 27831.387814, 47209.659485, 33306.286032, 17767.871582, 7915.660269, 9357.032289, 18710.899698, 7523.08838
6, 24864.067839, 7281.303252, 7887.296754, 35456.336973, 9148.653327, 11522.610094, 25442.690293, 11203.213501, 10453.46
5725, 18139.443416, 8506.524, 31044.281873, 33143.172796, 43466.234133, 9265.280866, 9395.167772, 14763.153987, 14887.90
6868, 33720.775102, 37826.852776, 40146.246475, 9253.601003, 21501.229815, 24061.586847, 8963.766641, 15836.570088, 2226
0.164762, 36170.547312, 45194.785555, 27375.49628, 34639.495347, 41522.716544, 41525.152348, 44330.599601, 22156.812784,
22962.972151, 19075.193616, 25433.895594, 9152.191697, 11527.089139, 17584.097741, 7463.457476, 14731.582544, 14784.1639
89]
```

In [114…

```
print (len(fp1))
```

```
237
```

In [114…

```
print (len(fp2))
```

```
207
```

In [115…

```
print (len(fp3))
```

```
253
```

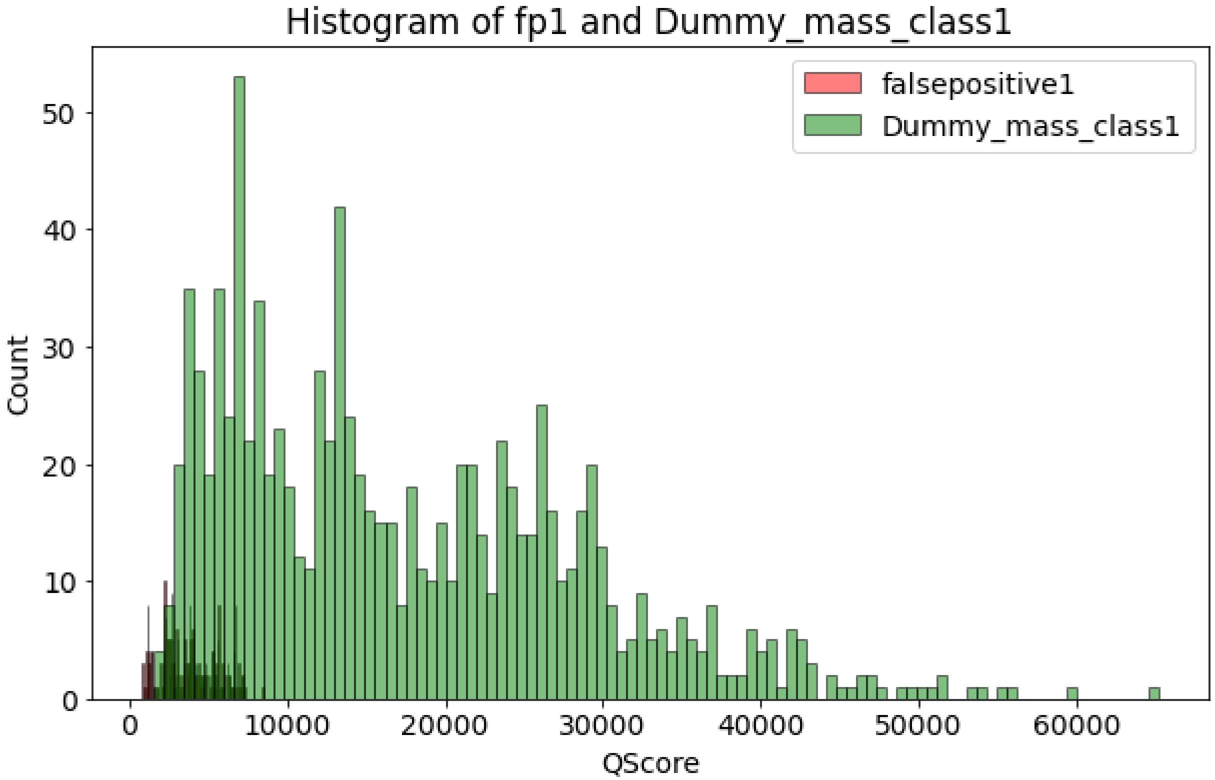
In [115…

```
print (len(false_positives))
```

```
697
```

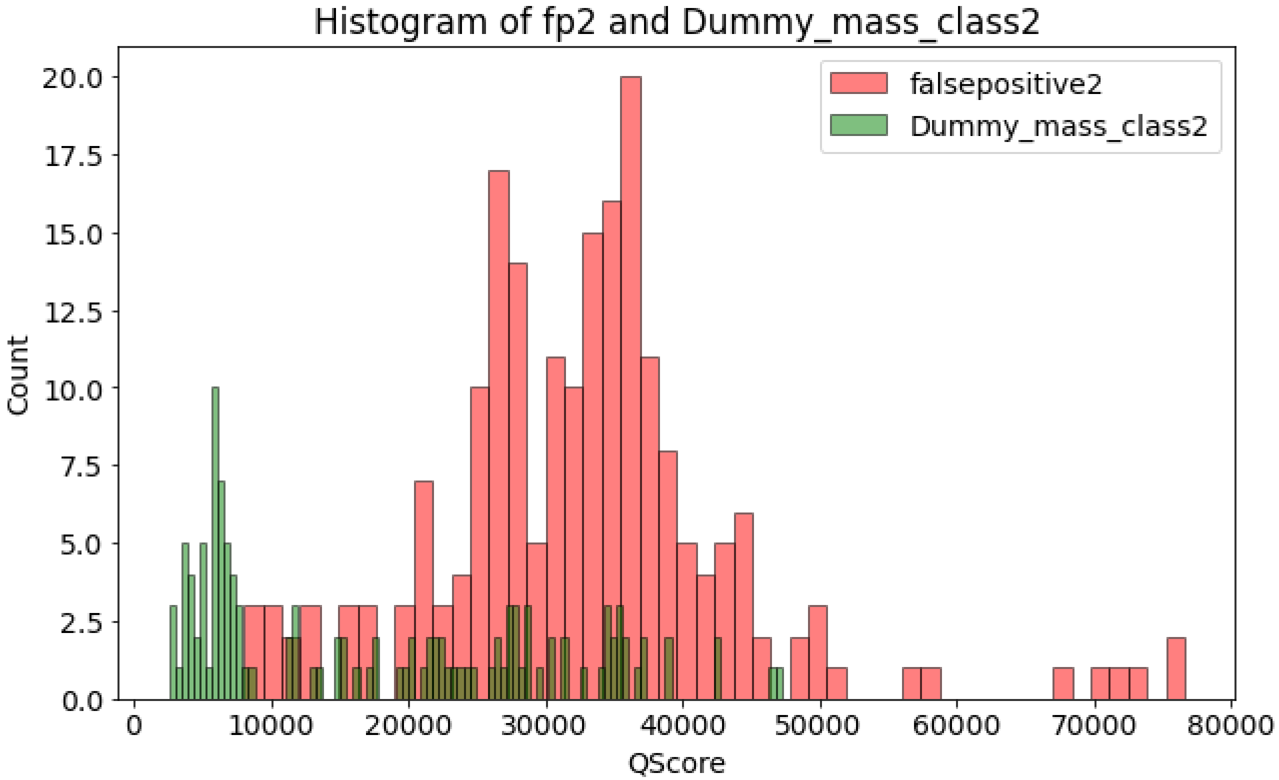
In [115…

```
# Filter Dummy_mass_class1 (where DummyIndex == 1)
dummy_mass_class1 = df_b[df_b['DummyIndex'] == 1]['MonoisotopicMass']

# Plot histogram of fp1 and Dummy_mass_class1
plt.figure(figsize=(10, 6))
plt.hist(fp1, bins=100, alpha=0.5, label='falsepositive1', color='red', edgecolor='black')
plt.hist(dummy_mass_class1, bins=100, alpha=0.5, label='Dummy_mass_class1', color='green', edgecolor='black')
plt.xlabel('QScore')
plt.ylabel('Count')
plt.title('Histogram of fp1 and Dummy_mass_class1')
plt.legend()
plt.show()
```



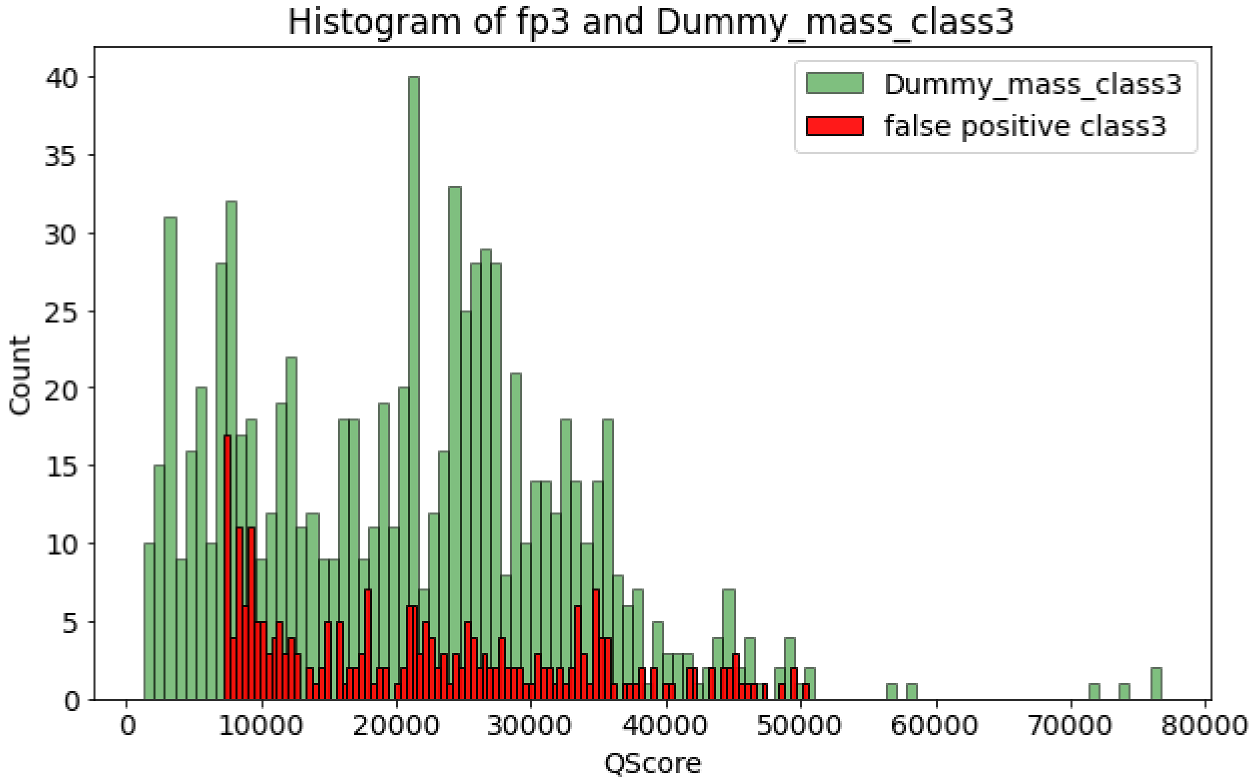
In [115…

```
# Filter Dummy_mass_class1 (where DummyIndex == 1)
dummy_mass_class2 = df_b[df_b['DummyIndex'] == 2]['MonoisotopicMass']

# Plot histogram of fp1 and Dummy_mass_class1
plt.figure(figsize=(10, 6))
plt.hist(fp2, bins=50, alpha=0.5, label='falsepositive2', color='red', edgecolor='black')
plt.hist(dummy_mass_class2, bins=100, alpha=0.5, label='Dummy_mass_class2', color='green', edgecolor='black')
plt.xlabel('QScore')
plt.ylabel('Count')
plt.title('Histogram of fp2 and Dummy_mass_class2')
plt.legend()
plt.show()
```



In [115…

```
# Filter Dummy_mass_class1 (where DummyIndex == 1)
dummy_mass_class3 = df_b[df_b['DummyIndex'] == 3]['MonoisotopicMass']

# Plot histogram of fp1 and Dummy_mass_class1
plt.figure(figsize=(10, 6))

plt.hist(dummy_mass_class3, bins=100, alpha=0.5, label='Dummy_mass_class3', color='green', edgecolor='black')
plt.hist(fp3, bins=100, alpha=0.9, label='false positive class3', color='red', edgecolor='black')
plt.xlabel('QScore')
plt.ylabel('Count')
plt.title('Histogram of fp3 and Dummy_mass_class3')
plt.legend()
plt.show()
```


Histogram of fp3 and Dummy_mass_class3
Dummy_mass_class3
false positive class3
Count
QScore
0
5
10
15
20
25
30
35
40
0
10000
20000
30000
40000
50000
60000
70000
80000

In [ ]:

In [ ]:

In [ ]:

In [ ]: