fault_detection

March 12, 2019

```
In [3]: import pandas as pd
        import pyarrow.parquet as pq
        import os
        import pyarrow
        import matplotlib.pyplot as plt
        import numpy as np
        import seaborn as sns
        from scipy.signal import *
        import statsmodels.api as sm
        from keras import backend as K
        import gc
        from sklearn.feature_selection import f_classif
        import lightgbm as lgbm
        from sklearn.model_selection import RandomizedSearchCV
        from scipy.stats import expon, uniform, norm
        from scipy.stats import randint, poisson
        from sklearn.metrics import confusion_matrix, make_scorer

        print(os.listdir('../input'))
```

```
['metadata_train.csv', 'train.parquet', 'train_subset_meta.csv', 'metadata_train_V2.csv', 'subse
```

1 Import data

```
In [4]: %%time

        print('load data...')
        train_df = pq.read_pandas("../input/train.parquet").to_pandas()
        #test_df = pq.read_pandas("../input/test.parquet").to_pandas()
        train_meta_df = pd.read_csv("../input/metadata_train.csv")
        #test_meta_df = pd.read_csv("../input/metadata_test.csv")
        print('import ok')
```

```
load data...
import ok
CPU times: user 55.6 s, sys: 11.6 s, total: 1min 7s
```

```
Wall time: 1min 8s
```

```
In [5]: train_meta_df.head(n=9)
```

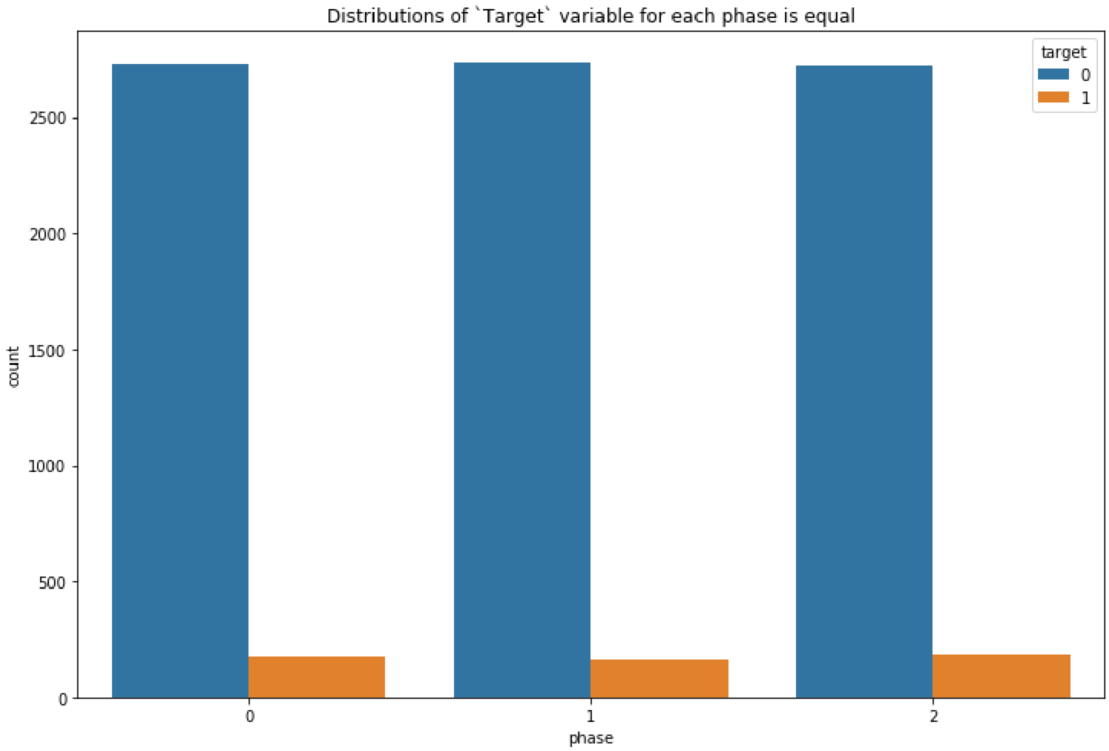
Out[5]:

	signal_id	id_measurement	phase	target
0	0	0	0	0
1	1	0	1	0
2	2	0	2	0
3	3	1	0	1
4	4	1	1	1
5	5	1	2	1
6	6	2	0	0
7	7	2	1	0
8	8	2	2	0

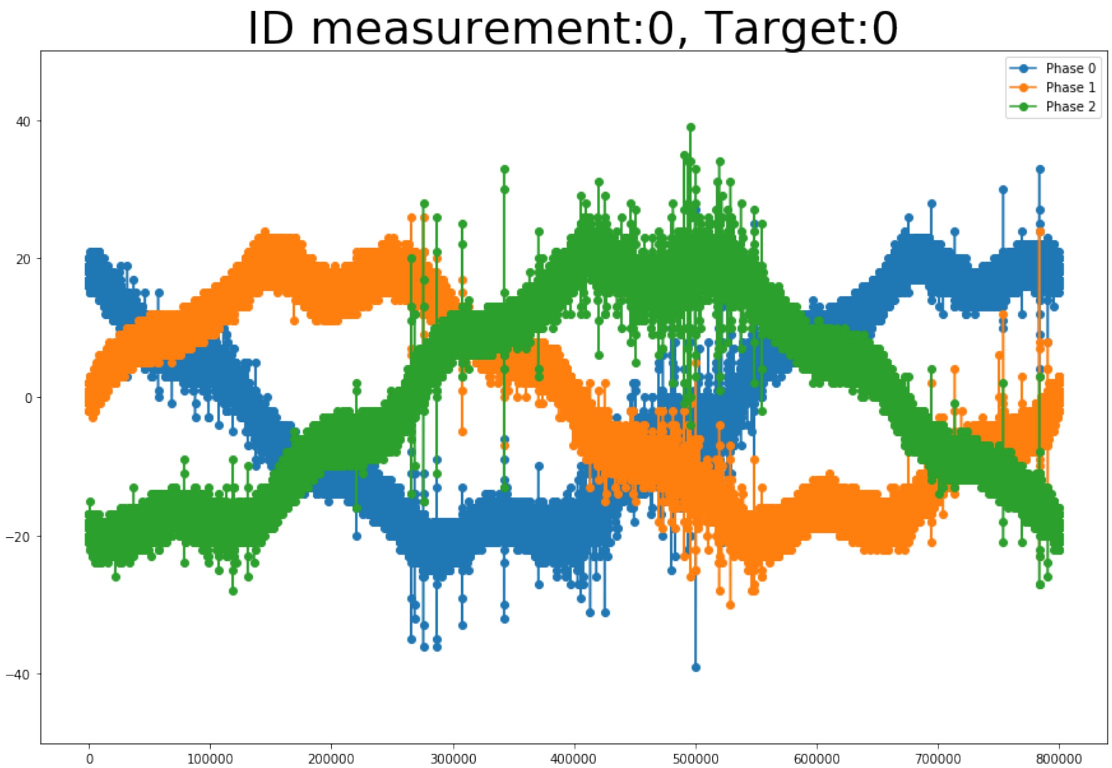
2 Exploratory data analysis

```
In [6]: # plot settings
        rand_seed = 135
        np.random.seed(rand_seed)
        xsize = 12.0
        ysize = 8.0
```

```
In [7]: fig, ax = plt.subplots()
        fig.set_size_inches(xsize, ysize)

        ax = sns.countplot(x="phase", hue="target", data=train_meta_df, ax=ax)
        ax.set_title("Distributions of `Target` variable for each phase is equal")
        plt.show()
```

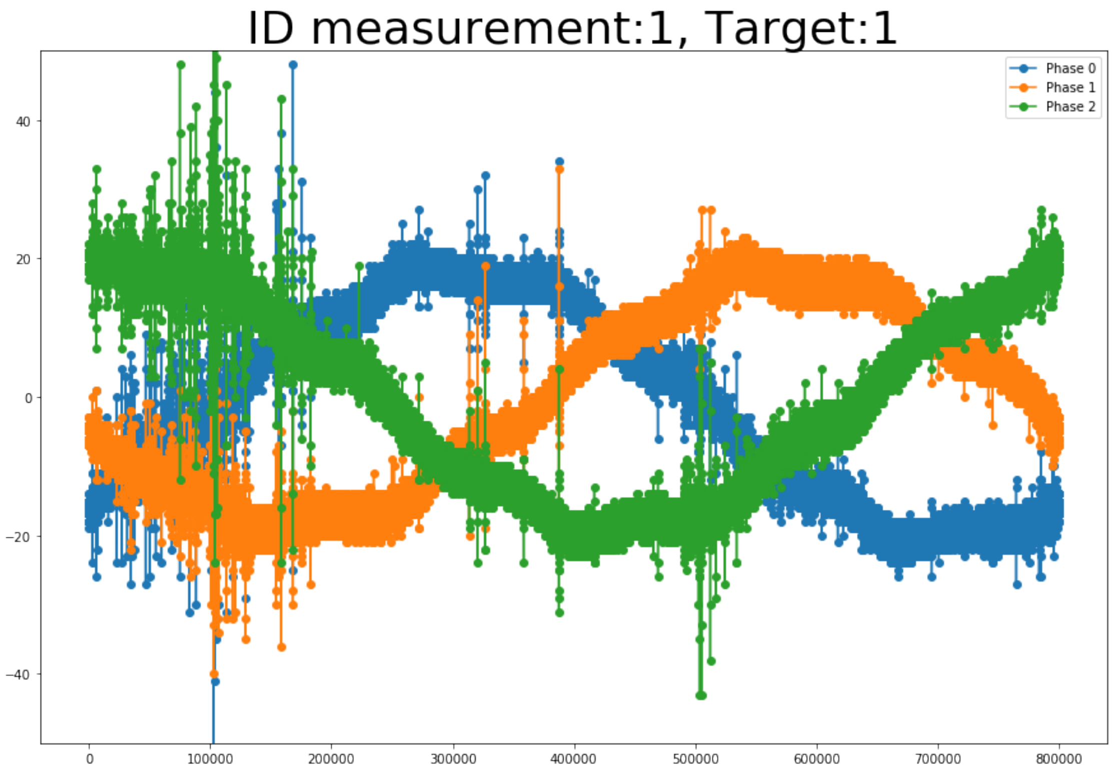



```
In [8]: %%time

        plt.figure(figsize=(15, 10))
        plt.title("ID measurement:0, Target:0", fontdict={'fontsize':36})
        plt.plot(train_df["0"].values, marker="o", label='Phase 0')
        plt.plot(train_df["1"].values, marker="o", label='Phase 1')
        plt.plot(train_df["2"].values, marker="o", label='Phase 2')
        plt.ylim(-50,50)
        plt.legend()
        plt.show()
```



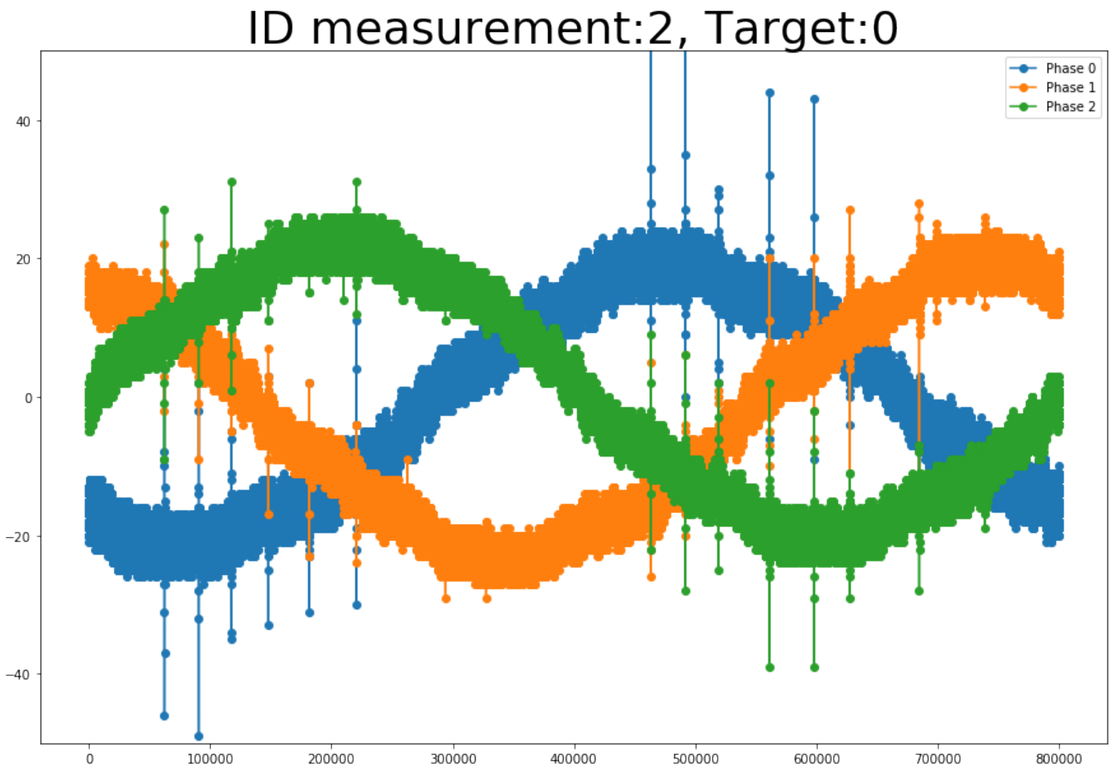

```
CPU times: user 18.3 s, sys: 364 ms, total: 18.6 s
Wall time: 10 s
```

In [9]:
```
%%time

plt.figure(figsize=(15, 10))
plt.title("ID measurement:1, Target:1", fontdict={'fontsize':36})
plt.plot(train_df["3"].values, marker="o", label='Phase 0')
plt.plot(train_df["4"].values, marker="o", label='Phase 1')
plt.plot(train_df["5"].values, marker="o", label='Phase 2')
plt.ylim(-50,50)
plt.legend()
plt.show()
```




```
CPU times: user 16.2 s, sys: 364 ms, total: 16.5 s
Wall time: 8.88 s
```

```
In [10]: %%time

         plt.figure(figsize=(15, 10))
         plt.title("ID measurement:2, Target:0", fontdict={'fontsize':36})
         plt.plot(train_df["6"].values, marker="o", label='Phase 0')
         plt.plot(train_df["7"].values, marker="o", label='Phase 1')
         plt.plot(train_df["8"].values, marker="o", label='Phase 2')
         plt.ylim(-50,50)
         plt.legend()
         plt.show()
```




```
CPU times: user 16.7 s, sys: 332 ms, total: 17 s
Wall time: 8.95 s
```

3 Save a subset of data

In order to speed up the code, we only run on a subset of the training data.

In [11]:
```
train_subset_df = train_df.iloc[:,range(0,99)]
train_subset_meta_df = train_meta_df.iloc[range(0,99),:]
#train_subset_df.to_csv('../input/train_subset.csv')
#train_subset_meta_df.to_csv('../input/train_subset_meta.csv')

# uncomment to use the full dataset after
train_subset_df = train_df
train_subset_meta_df = train_meta_df
```

In [12]:
```
train_subset_meta_df.head(n=9)
```

Out[12]:

	signal_id	id_measurement	phase	target
0	0	0	0	0
1	1	0	1	0
2	2	0	2	0

3	3	1	0	1
4	4	1	1	1
5	5	1	2	1
6	6	2	0	0
7	7	2	1	0
8	8	2	2	0

4 Feature engineering

4.1 Mean, median and standard deviation

In [13]:
```
%%time

mean_list = train_subset_df.apply(np.mean)
median_list = train_subset_df.apply(np.median)
std_list = train_subset_df.apply(np.std)
```

```
CPU times: user 3min 38s, sys: 0 ns, total: 3min 38s
Wall time: 1min 3s
```

In [14]:
```
mean_signal_df = mean_list.to_frame()
mean_signal_df = mean_signal_df.reset_index()
mean_signal_df = mean_signal_df.drop("index",axis=1)

train_subset_meta_df = train_subset_meta_df.merge(mean_signal_df,"inner", left_index=Tr
train_subset_meta_df = train_subset_meta_df.rename(index=str, columns={0:"mean"})

median_signal_df = median_list.to_frame()

train_subset_meta_df = train_subset_meta_df.merge(median_signal_df,"inner", left_index=
train_subset_meta_df = train_subset_meta_df.rename(index=str, columns={0:"median"})

std_signal_df = std_list.to_frame()

train_subset_meta_df = train_subset_meta_df.merge(std_signal_df,"inner", left_index=Tru
train_subset_meta_df = train_subset_meta_df.rename(index=str, columns={0:"std_dev"})
```

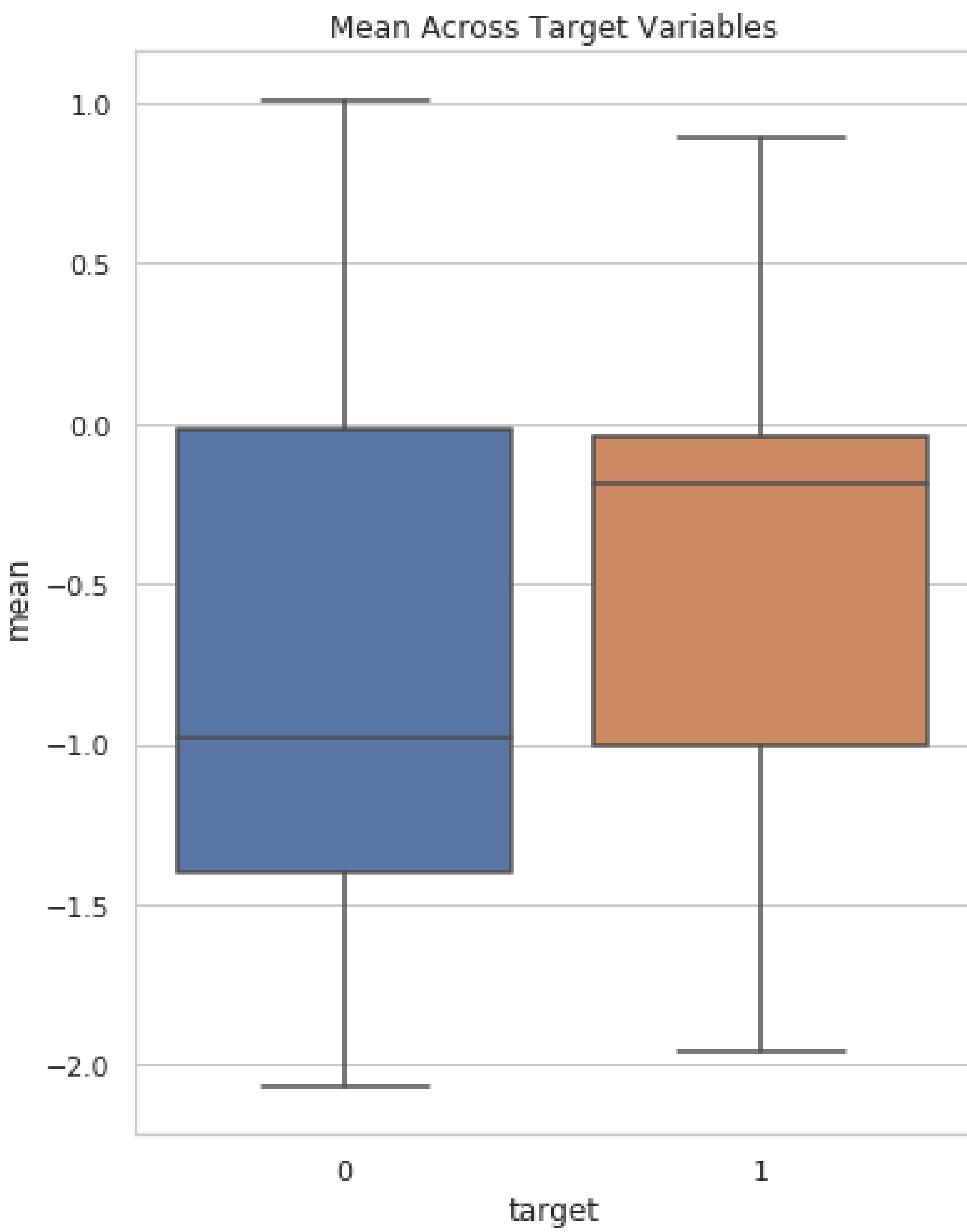
In [15]:
```
train_subset_meta_df.head(n=9)
```

Out[15]:

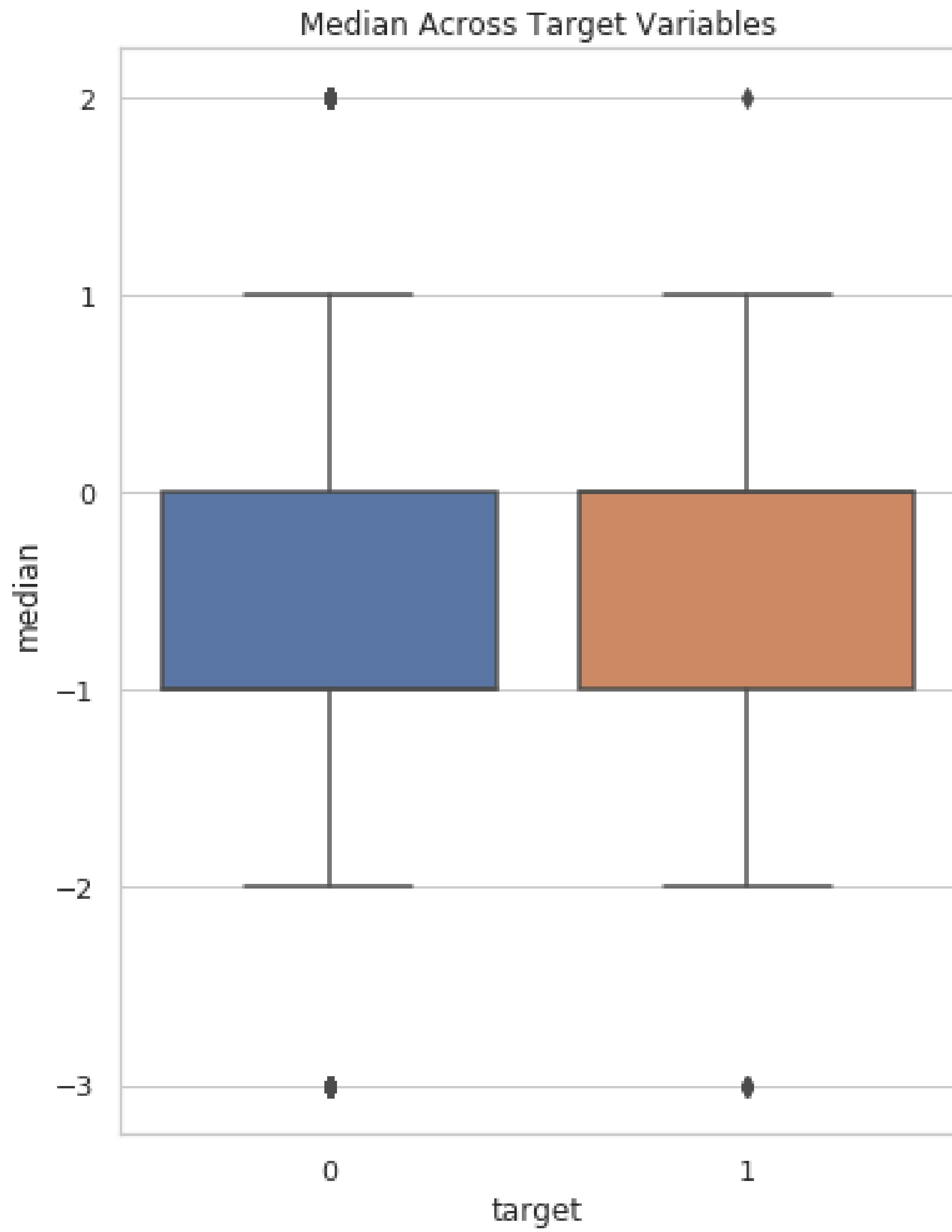
	signal_id	id_measurement	phase	target	mean	median	std_dev
0	0	0	0	0	-0.960271	-1.0	13.870724
1	1	0	1	0	-0.194125	0.0	13.037134
2	2	0	2	0	-0.043555	0.0	13.684282
3	3	1	0	1	-0.997401	-1.0	13.673630
4	4	1	1	1	-0.175586	0.0	12.938372
5	5	1	2	1	-0.036004	0.0	13.545777
6	6	2	0	0	-1.146185	-1.0	14.064211

7	7	2	1	0	-1.952695	-2.0	14.774424
8	8	2	2	0	0.873370	1.0	14.815668

In [16]:
```
plt.figure(figsize=(6,8))
sns.set(style="whitegrid")
plt.title("Mean Across Target Variables")
ax = sns.boxplot(x="target", y="mean", data=train_subset_meta_df)
```

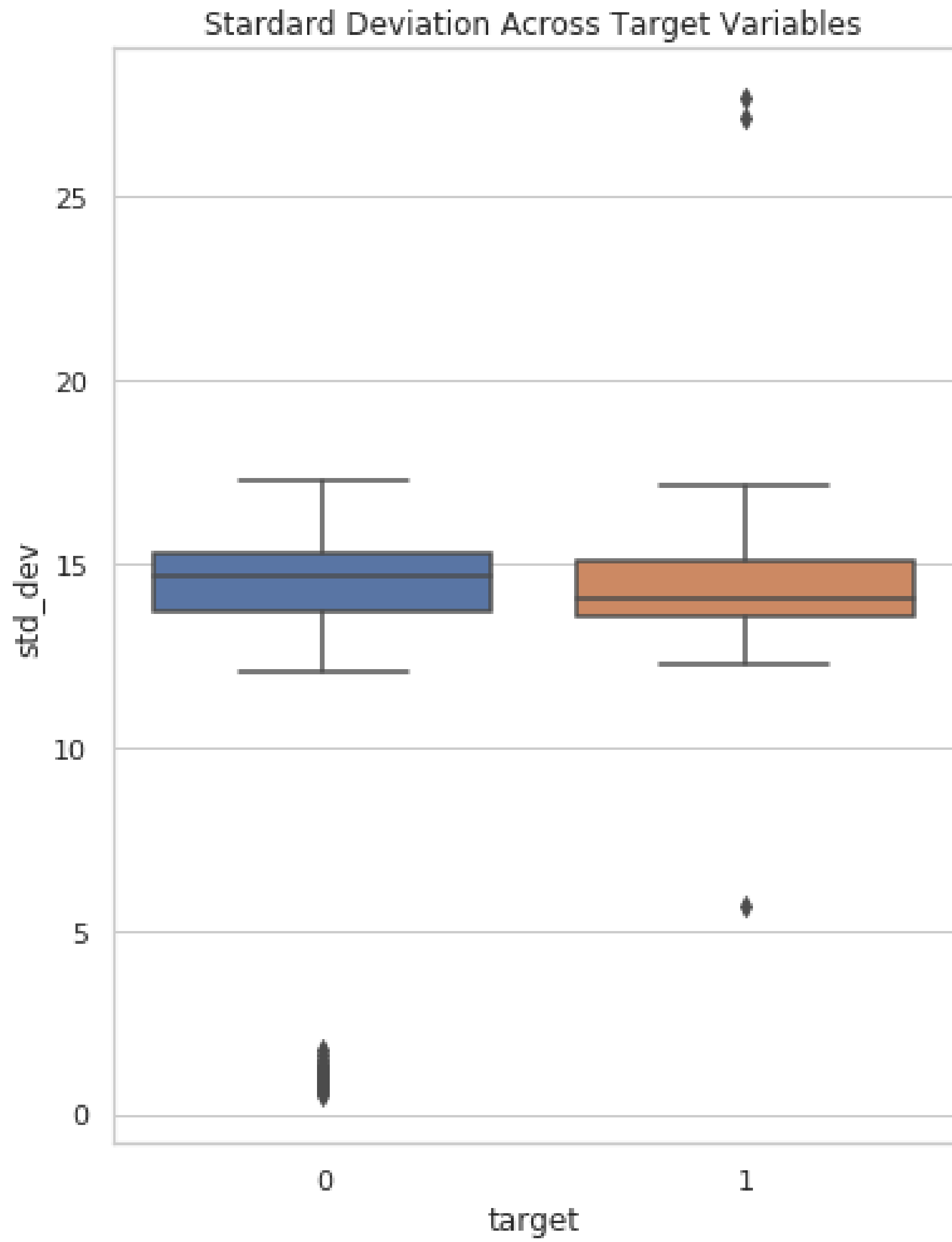


In [17]:
```
plt.figure(figsize=(6,8))
sns.set(style="whitegrid")
plt.title("Median Across Target Variables")
ax = sns.boxplot(x="target", y="median", data=train_subset_meta_df)
```



In [18]:
```
plt.figure(figsize=(6,8))
sns.set(style="whitegrid")
```

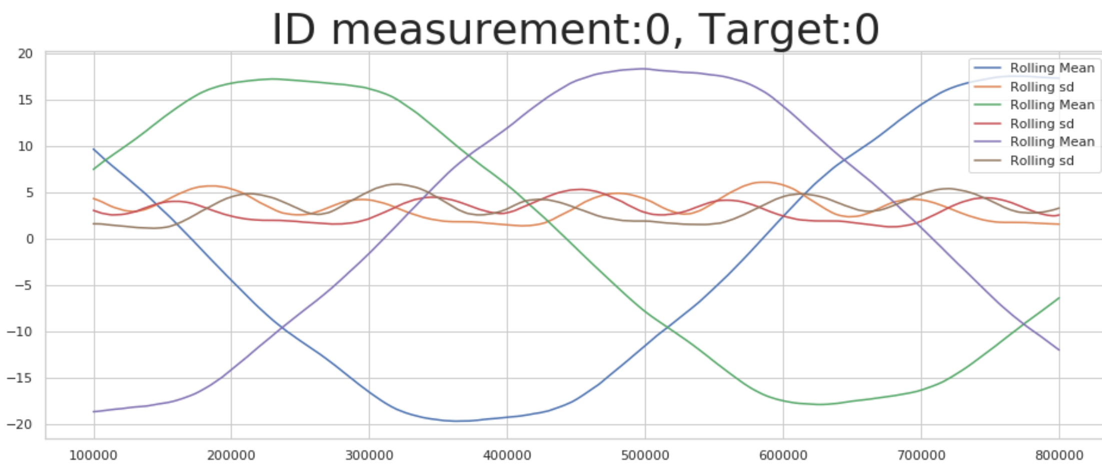
```
plt.title("Stardard Deviation Across Target Variables")
ax = sns.boxplot(x="target", y="std_dev", data=train_subset_meta_df)
```



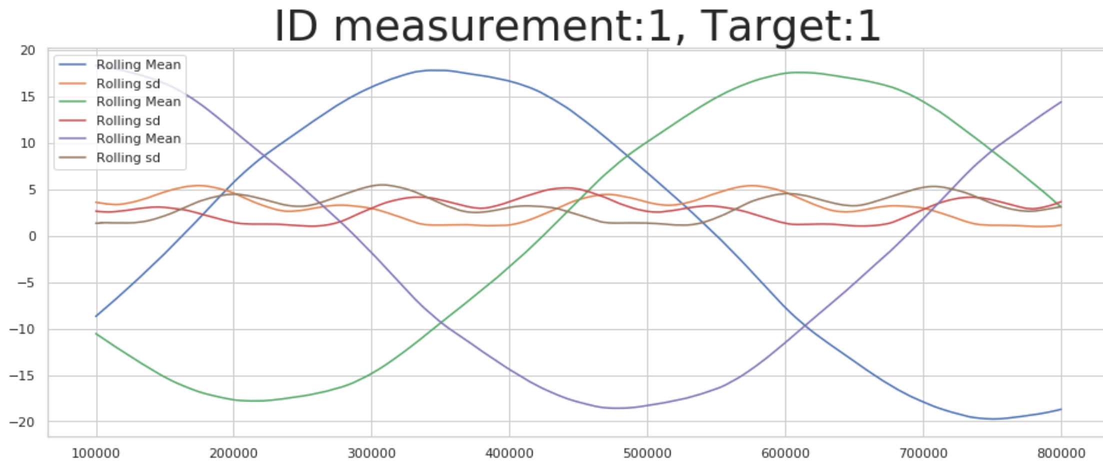
4.2 Amplitude of Rolling Series

Let's smooth the time series.

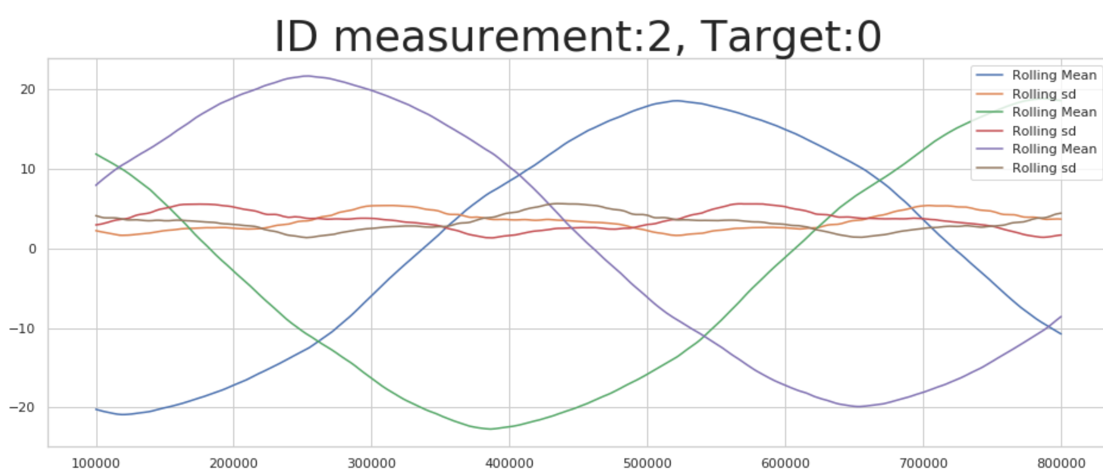
In [19]:
```
ts1 = train_df["0"]
ts2 = train_df["1"]
ts3 = train_df["2"]

plt.figure(figsize=(16,6))
plt.title("ID measurement:0, Target:0", fontdict={'fontsize':36})
plt.plot(ts1.rolling(window=100000,center=False).mean(),label='Rolling Mean');
plt.plot(ts1.rolling(window=100000,center=False).std(),label='Rolling sd');
plt.plot(ts2.rolling(window=100000,center=False).mean(),label='Rolling Mean');
plt.plot(ts2.rolling(window=100000,center=False).std(),label='Rolling sd');
plt.plot(ts3.rolling(window=100000,center=False).mean(),label='Rolling Mean');
plt.plot(ts3.rolling(window=100000,center=False).std(),label='Rolling sd');
plt.legend();
```



In [20]:
```
ts1 = train_df["3"]
ts2 = train_df["4"]
ts3 = train_df["5"]

plt.figure(figsize=(16,6))
plt.title("ID measurement:1, Target:1", fontdict={'fontsize':36})
plt.plot(ts1.rolling(window=100000,center=False).mean(),label='Rolling Mean');
plt.plot(ts1.rolling(window=100000,center=False).std(),label='Rolling sd');
plt.plot(ts2.rolling(window=100000,center=False).mean(),label='Rolling Mean');
plt.plot(ts2.rolling(window=100000,center=False).std(),label='Rolling sd');
plt.plot(ts3.rolling(window=100000,center=False).mean(),label='Rolling Mean');
plt.plot(ts3.rolling(window=100000,center=False).std(),label='Rolling sd');
plt.legend();
```




```
In [21]: ts1 = train_df["6"]
         ts2 = train_df["7"]
         ts3 = train_df["8"]

         plt.figure(figsize=(16,6))
         plt.title("ID measurement:2, Target:0", fontdict={'fontsize':36})
         plt.plot(ts1.rolling(window=100000,center=False).mean(),label='Rolling Mean');
         plt.plot(ts1.rolling(window=100000,center=False).std(),label='Rolling sd');
         plt.plot(ts2.rolling(window=100000,center=False).mean(),label='Rolling Mean');
         plt.plot(ts2.rolling(window=100000,center=False).std(),label='Rolling sd');
         plt.plot(ts3.rolling(window=100000,center=False).mean(),label='Rolling Mean');
         plt.plot(ts3.rolling(window=100000,center=False).std(),label='Rolling sd');
         plt.legend();
```



Let's look at the amplitude across each target group. To calculate the amplitude, I smooth the powerline signals to create a single wave then I subtract the lowest and highest point.

In [22]:
```
def calc_rolling_amp(row, window=100000):
    return np.max(row.rolling(window,center=False).mean()) - np.min(row.rolling(window=

rolling100k_amp = train_subset_df.apply(calc_rolling_amp)
```

In [23]:
```
rolling100k_amp_df = rolling100k_amp.to_frame()
train_subset_meta_df = train_subset_meta_df.merge(rolling100k_amp_df,"inner", left_inde
train_subset_meta_df = train_subset_meta_df.rename(index=str, columns={0:"rolling100k_a
```

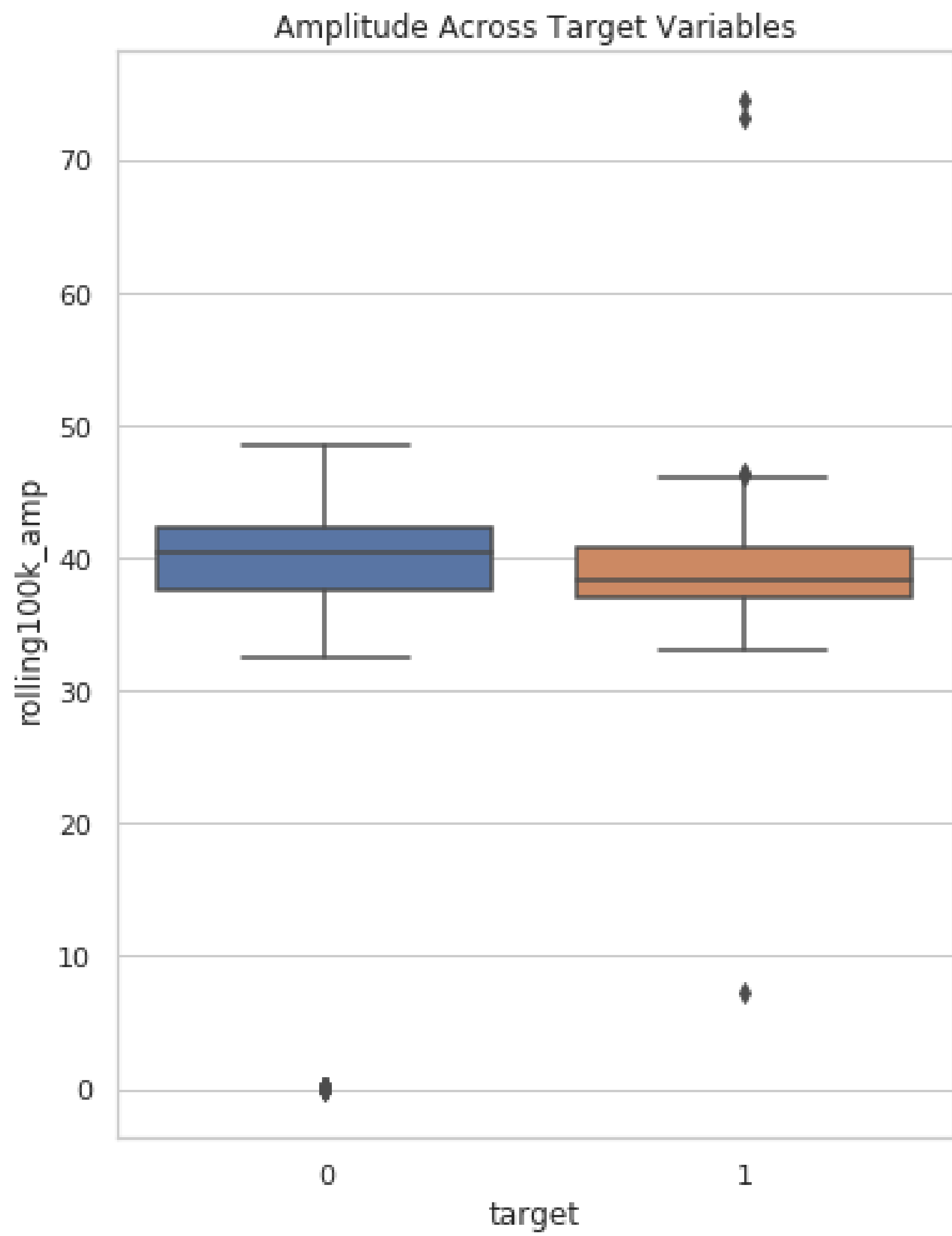
In [24]:
```
train_subset_meta_df.head(n=9)
```

Out[24]:

	signal_id	id_measurement	phase	target	mean	median	std_dev	\
0	0	0	0	0	-0.960271	-1.0	13.870724	
1	1	0	1	0	-0.194125	0.0	13.037134	
2	2	0	2	0	-0.043555	0.0	13.684282	
3	3	1	0	1	-0.997401	-1.0	13.673630	
4	4	1	1	1	-0.175586	0.0	12.938372	
5	5	1	2	1	-0.036004	0.0	13.545777	
6	6	2	0	0	-1.146185	-1.0	14.064211	
7	7	2	1	0	-1.952695	-2.0	14.774424	
8	8	2	2	0	0.873370	1.0	14.815668	

	rolling100k_amp
0	37.21537
1	35.10791
2	36.97624
3	37.53126
4	35.35856
5	36.87904
6	39.47469
7	41.58219
8	41.58396

In [25]:
```
plt.figure(figsize=(6,8))
sns.set(style="whitegrid")
plt.title("Amplitude Across Target Variables")
ax = sns.boxplot(x="target", y="rolling100k_amp", data=train_subset_meta_df)
```



4.3 Measuring Amount of Noisy Points

4.3.1 Number of points 1SD from the mean

```
In [26]: def count1SDfromTheMean(row):
             max_1sd = np.mean(row) + np.std(row)
             min_1sd = np.mean(row) - np.std(row)
```

```
        noise_points = [x for x in row if (x > max_1sd) or (x < min_1sd)]
        return (len(noise_points))
```

In [27]:
```
count1SDfromTheMean_list = train_subset_df.apply(count1SDfromTheMean)
count1SDfromTheMean_df = count1SDfromTheMean_list.to_frame()
train_subset_meta_df = train_subset_meta_df.merge(count1SDfromTheMean_df,"inner", left_
train_subset_meta_df = train_subset_meta_df.rename(index=str, columns={0:"count1SDfromT
```

In [28]:
```
train_subset_meta_df.head(n=9)
```

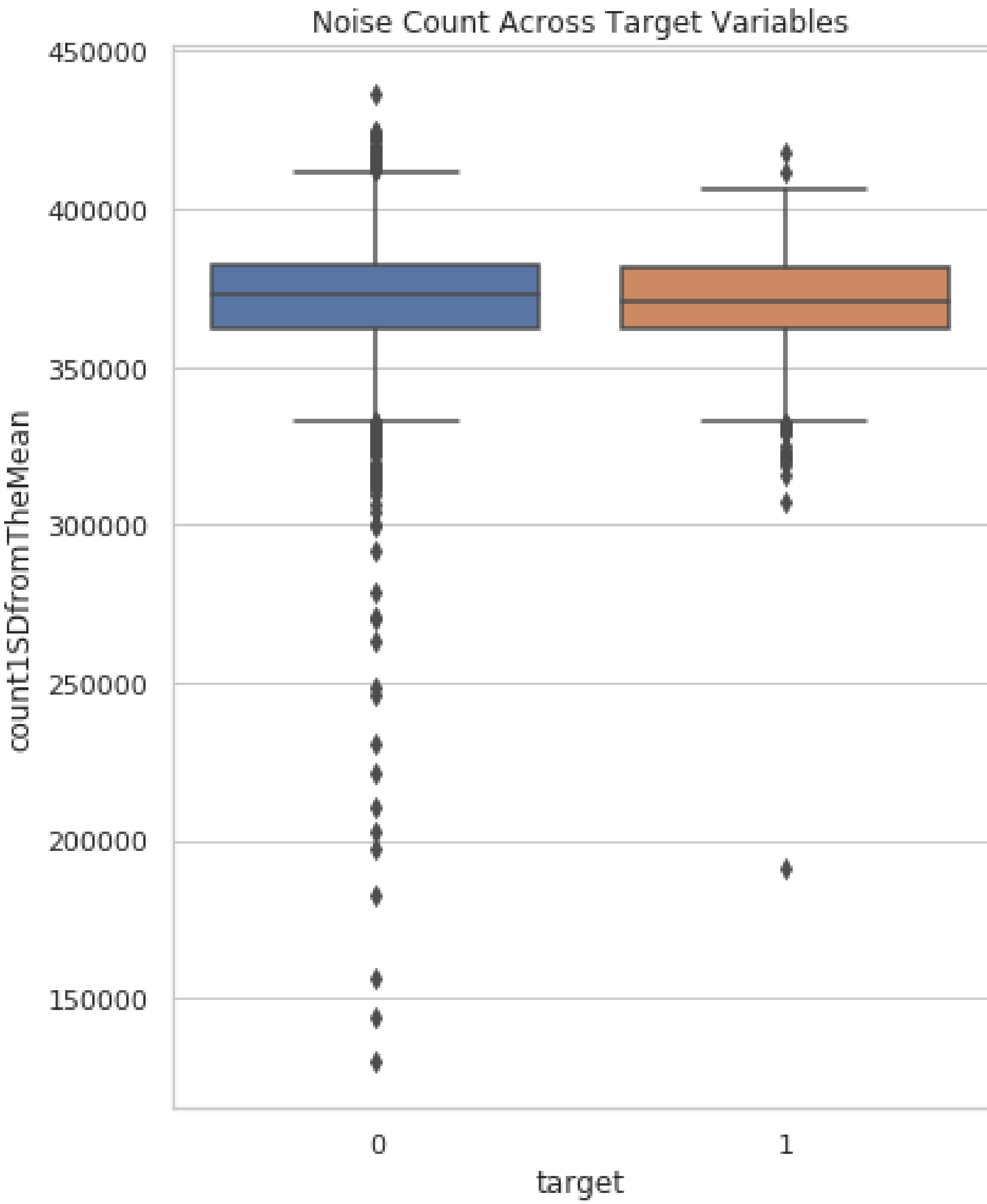
Out[28]:

	signal_id	id_measurement	phase	target	mean	median	std_dev
0	0	0	0	0	-0.960271	-1.0	13.870724
1	1	0	1	0	-0.194125	0.0	13.037134
2	2	0	2	0	-0.043555	0.0	13.684282
3	3	1	0	1	-0.997401	-1.0	13.673630
4	4	1	1	1	-0.175586	0.0	12.938372
5	5	1	2	1	-0.036004	0.0	13.545777
6	6	2	0	0	-1.146185	-1.0	14.064211
7	7	2	1	0	-1.952695	-2.0	14.774424
8	8	2	2	0	0.873370	1.0	14.815668

\

	rolling100k_amp	count1SDfromTheMean
0	37.21537	377353
1	35.10791	372859
2	36.97624	377776
3	37.53126	381716
4	35.35856	377552
5	36.87904	379631
6	39.47469	378400
7	41.58219	391669
8	41.58396	394043

In [29]:
```
plt.figure(figsize=(6,8))
sns.set(style="whitegrid")
plt.title("Noise Count Across Target Variables")
ax = sns.boxplot(x="target", y="count1SDfromTheMean", data=train_subset_meta_df)
```



4.3.2 Number of points 2SD from the mean

```
In [30]: def count2SDfromTheMean(row):
             max_1sd = np.mean(row) + (2 * np.std(row))
             min_1sd = np.mean(row) - (2 * np.std(row))
             noise_points = [x for x in row if (x > max_1sd) or (x < min_1sd)]
             return (len(noise_points))
```

In [31]:
```
count2SDfromTheMean_list = train_subset_df.apply(count2SDfromTheMean)
count2SDfromTheMean_df = count2SDfromTheMean_list.to_frame()
train_subset_meta_df = train_subset_meta_df.merge(count2SDfromTheMean_df,"inner", left_
train_subset_meta_df = train_subset_meta_df.rename(index=str, columns={0:"count2SDfromT
```

In [32]:
```
train_subset_meta_df.head()
```

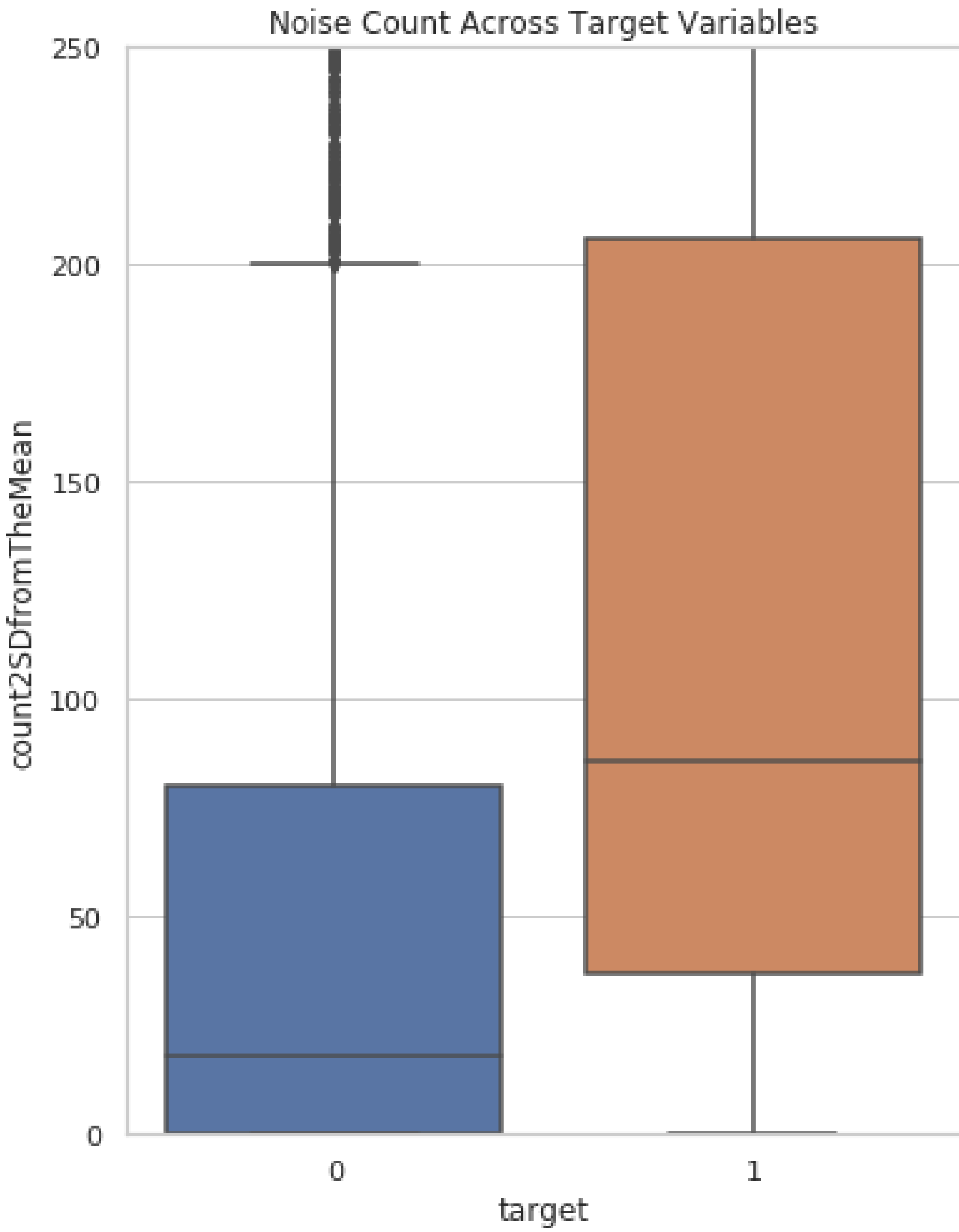
Out[32]:

	signal_id	id_measurement	phase	target	mean	median	std_dev	\
0	0	0	0	0	-0.960271	-1.0	13.870724	
1	1	0	1	0	-0.194125	0.0	13.037134	
2	2	0	2	0	-0.043555	0.0	13.684282	
3	3	1	0	1	-0.997401	-1.0	13.673630	
4	4	1	1	1	-0.175586	0.0	12.938372	

	rolling100k_amp	count1SDfromTheMean	count2SDfromTheMean
0	37.21537	377353	21
1	35.10791	372859	7
2	36.97624	377776	23
3	37.53126	381716	28
4	35.35856	377552	24

In [33]:
```
plt.figure(figsize=(6,8))
sns.set(style="whitegrid")
plt.title("Noise Count Across Target Variables")
ax = sns.boxplot(x="target", y="count2SDfromTheMean", data=train_subset_meta_df)
plt.ylim(0,250)
```

Out[33]: (0, 250)

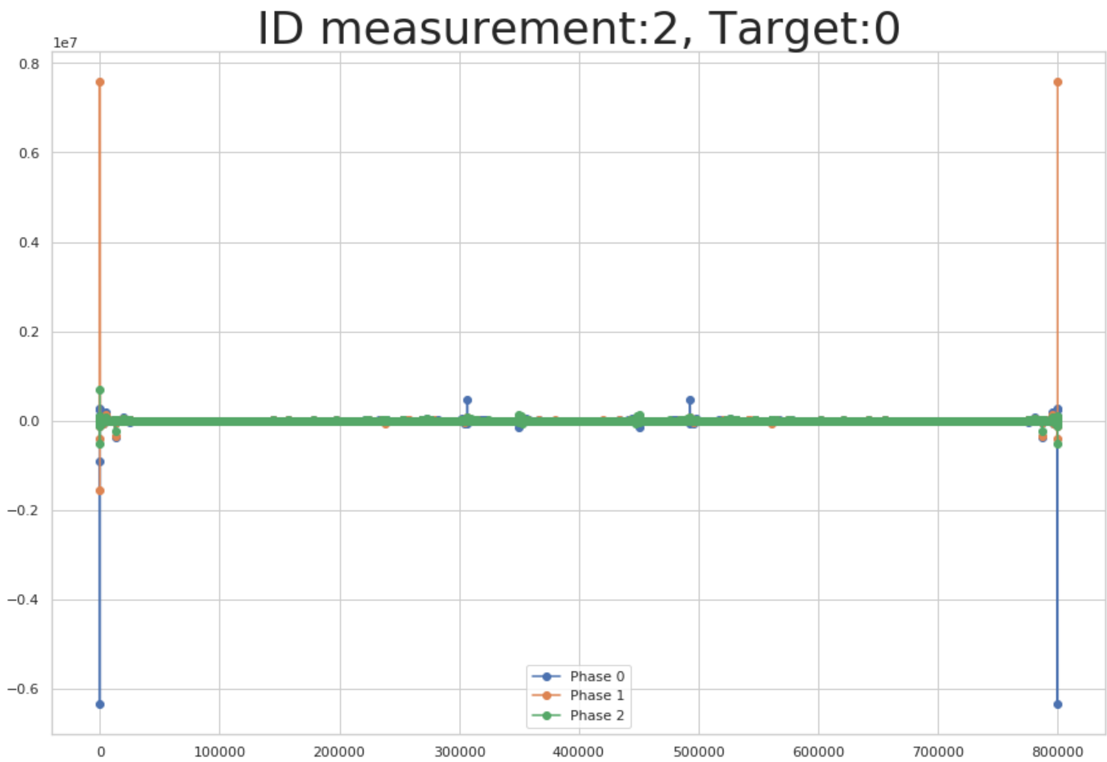


4.4 FFT

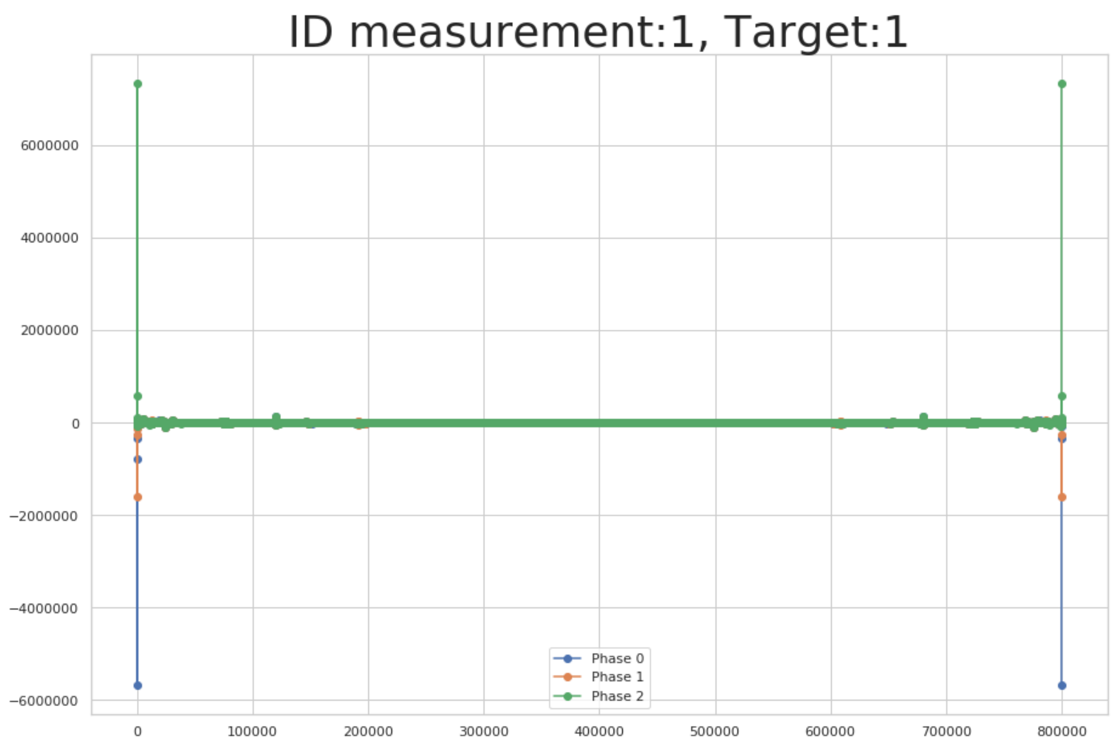
```
In [34]: plt.figure(figsize=(15, 10))
         plt.title("ID measurement:2, Target:0", fontdict={'fontsize':36})
         plt.plot(np.fft.fft(train_subset_df["6"]), marker="o", label='Phase 0')
         plt.plot(np.fft.fft(train_subset_df["7"]), marker="o", label='Phase 1')
         plt.plot(np.fft.fft(train_subset_df["8"]), marker="o", label='Phase 2')
```

```
plt.legend()
plt.show()
```

```
/home/pierre/bin/anaconda3/lib/python3.6/site-packages/numpy/core/numeric.py:538: ComplexWarning
  return array(a, dtype, copy=False, order=order)
```



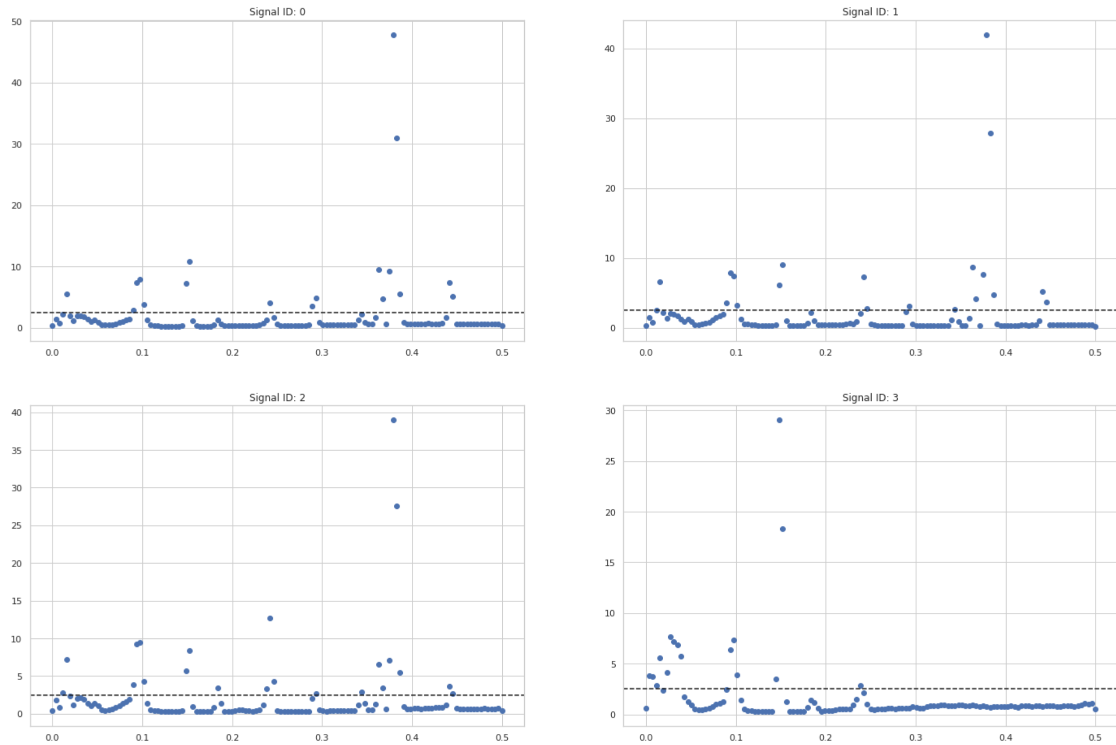

```
In [35]: plt.figure(figsize=(15, 10))
         plt.title("ID measurement:1, Target:1", fontdict={'fontsize':36})
         plt.plot(np.fft.fft(train_subset_df["3"]), marker="o", label='Phase 0')
         plt.plot(np.fft.fft(train_subset_df["4"]), marker="o", label='Phase 1')
         plt.plot(np.fft.fft(train_subset_df["5"]), marker="o", label='Phase 2')
         plt.legend()
         plt.show()
```



4.5 Power Spectral density using the Welch's method

In [36]:
```
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_size_inches(2.0*xsize, 2.0*ysize)
axes = axes.flatten()

f, Pxx = welch(train_subset_df["0"].values)
axes[0].plot(f, Pxx, marker="o", linestyle="none")
axes[0].set_title("Signal ID: 0")
axes[0].axhline(y=2.5, color="k", linestyle="--")

f, Pxx = welch(train_subset_df["1"].values)
axes[1].plot(f, Pxx, marker="o", linestyle="none")
axes[1].set_title("Signal ID: 1")
axes[1].axhline(y=2.5, color="k", linestyle="--")

f, Pxx = welch(train_subset_df["2"].values)
axes[2].plot(f, Pxx, marker="o", linestyle="none")
axes[2].set_title("Signal ID: 2")
axes[2].axhline(y=2.5, color="k", linestyle="--")

f, Pxx = welch(train_subset_df["3"].values)
axes[3].plot(f, Pxx, marker="o", linestyle="none")
```

```
axes[3].set_title("Signal ID: 3")
axes[3].axhline(y=2.5, color="k", linestyle="--")

plt.show()
```




```
In [37]: %%time

def welch_max_power_and_frequency(signal):
    f, Pxx = welch(signal)
    ix = np.argmax(Pxx)
    strong_count = np.sum(Pxx>2.5)
    avg_amp = np.mean(Pxx)
    sum_amp = np.sum(Pxx)
    std_amp = np.std(Pxx)
    median_amp = np.median(Pxx)
    return [Pxx[ix], f[ix], strong_count, avg_amp, sum_amp, std_amp, median_amp]

power_spectrum_summary = train_subset_df.apply(welch_max_power_and_frequency, result_ty
```

```
CPU times: user 26min 30s, sys: 24 ms, total: 26min 30s
Wall time: 4min 41s
```

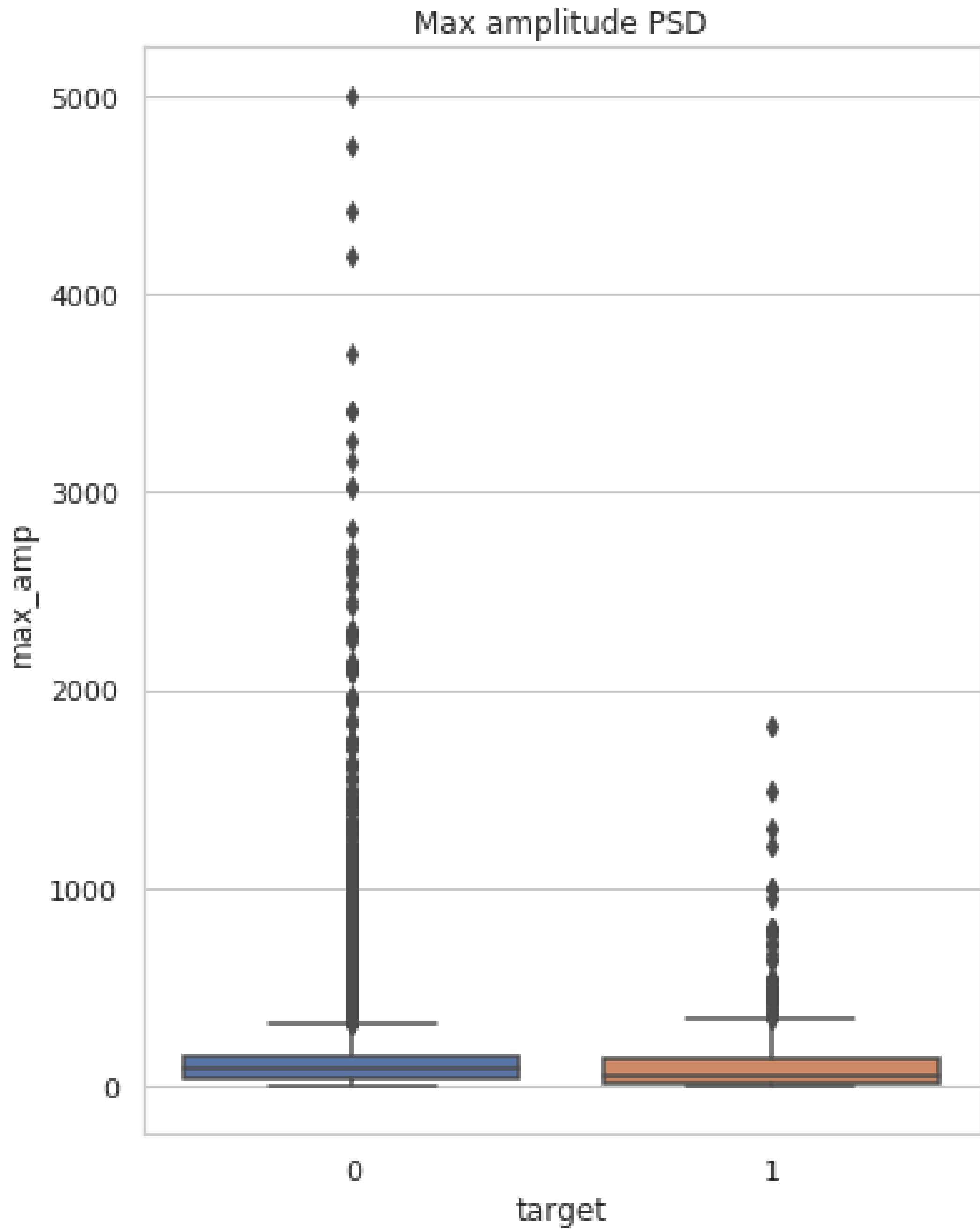
In [38]:
```
power_spectrum_summary = power_spectrum_summary.T.rename(columns={0:"max_amp", 1:"max_f
                                                                    4:"sum_amp", 5:"std_a

power_spectrum_summary.head()
```

Out[38]:
```
     max_amp  max_freq  strong_amp_count   avg_amp     sum_amp   std_amp  \
0  47.831062  0.378906              18.0  1.995473  257.415955  5.220994
1  41.982578  0.378906              20.0  1.836459  236.903198  4.641738
2  38.999954  0.378906              22.0  1.935738  249.710236  4.503104
3  29.060942  0.148438              16.0  1.590296  205.148132  3.224866
4  20.080660  0.148438              19.0  1.375932  177.495178  2.577763

   median_amp
0    0.612473
1    0.458779
2    0.652928
3    0.798159
4    0.439266
```

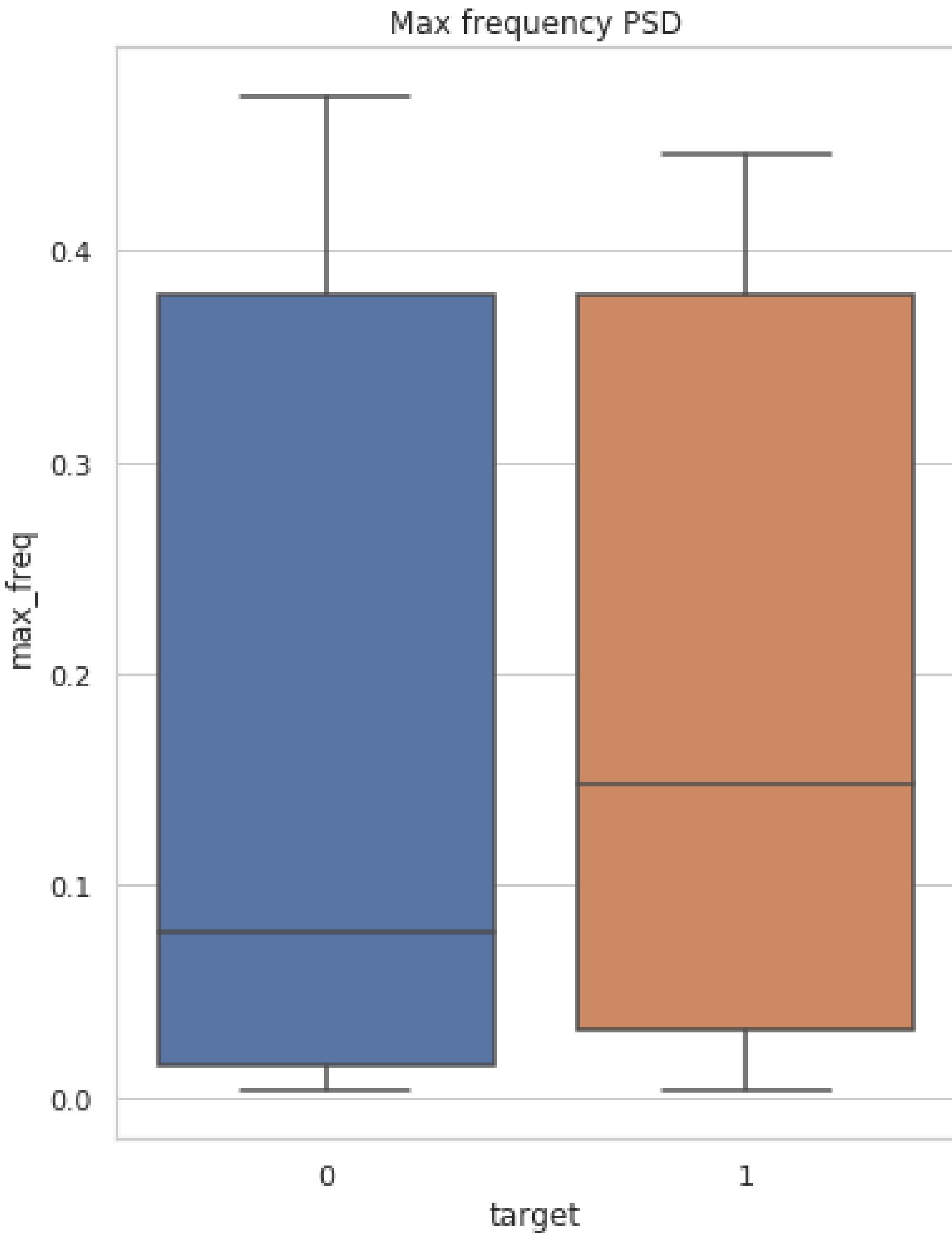
In [39]:
```
power_spectrum_summary.index = power_spectrum_summary.index.astype(int)

train_subset_meta_df = train_subset_meta_df.merge(power_spectrum_summary, left_on="sign
train_subset_meta_df.head()
```

Out[39]:
```
   signal_id  id_measurement  phase  target      mean  median    std_dev  \
0          0               0      0       0 -0.960271    -1.0  13.870724
1          1               0      1       0 -0.194125     0.0  13.037134
2          2               0      2       0 -0.043555     0.0  13.684282
3          3               1      0       1 -0.997401    -1.0  13.673630
4          4               1      1       1 -0.175586     0.0  12.938372

   rolling100k_amp  count1SDfromTheMean  count2SDfromTheMean    max_amp  \
0         37.21537               377353                   21  47.831062
1         35.10791               372859                    7  41.982578
2         36.97624               377776                   23  38.999954
3         37.53126               381716                   28  29.060942
4         35.35856               377552                   24  20.080660

   max_freq  strong_amp_count   avg_amp     sum_amp   std_amp  median_amp
0  0.378906              18.0  1.995473  257.415955  5.220994    0.612473
1  0.378906              20.0  1.836459  236.903198  4.641738    0.458779
2  0.378906              22.0  1.935738  249.710236  4.503104    0.652928
3  0.148438              16.0  1.590296  205.148132  3.224866    0.798159
4  0.148438              19.0  1.375932  177.495178  2.577763    0.439266
```

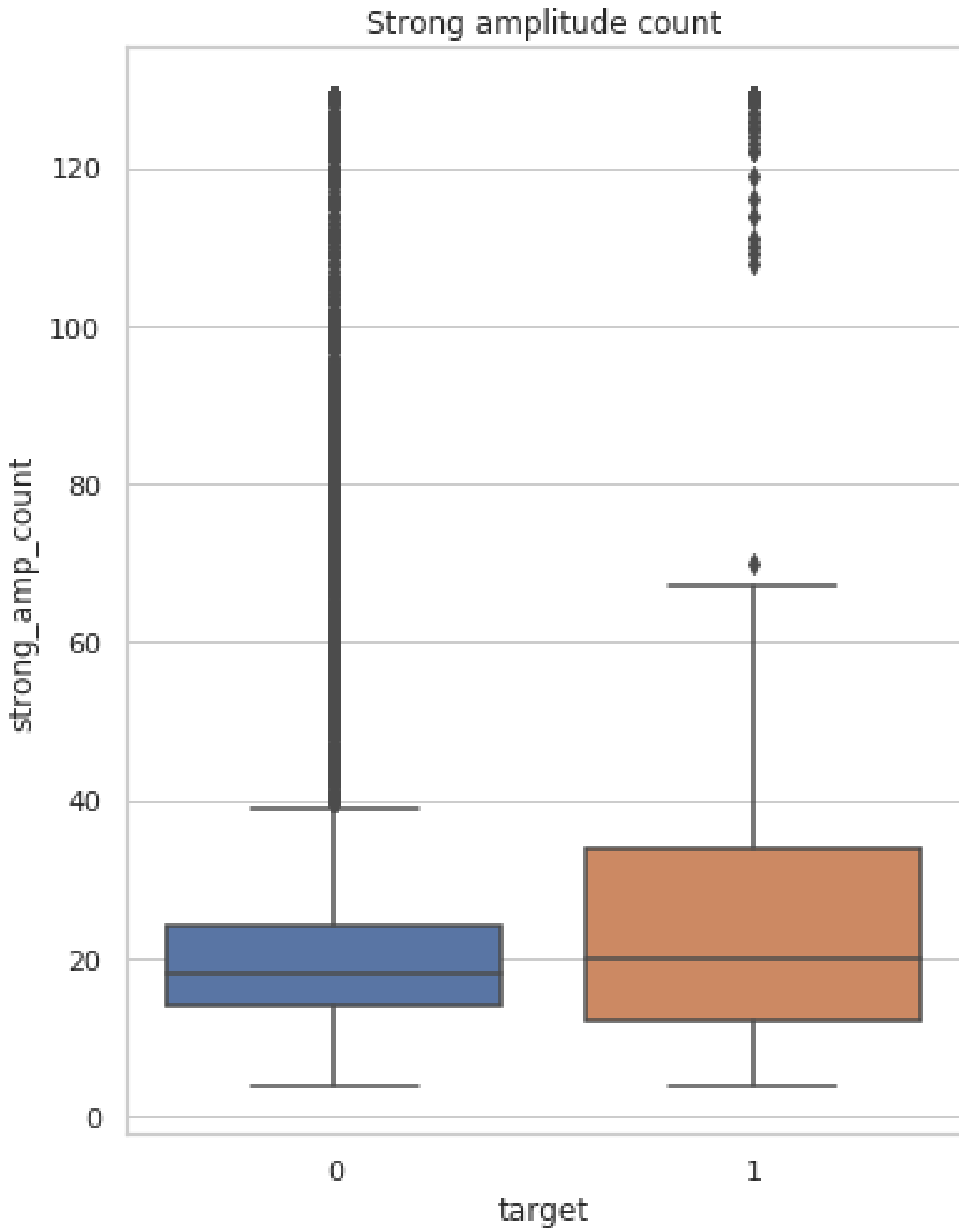
In [40]:
```
plt.figure(figsize=(6,8))
sns.set(style="whitegrid")
plt.title("Max amplitude PSD")
ax = sns.boxplot(x="target", y="max_amp", data=train_subset_meta_df)
```



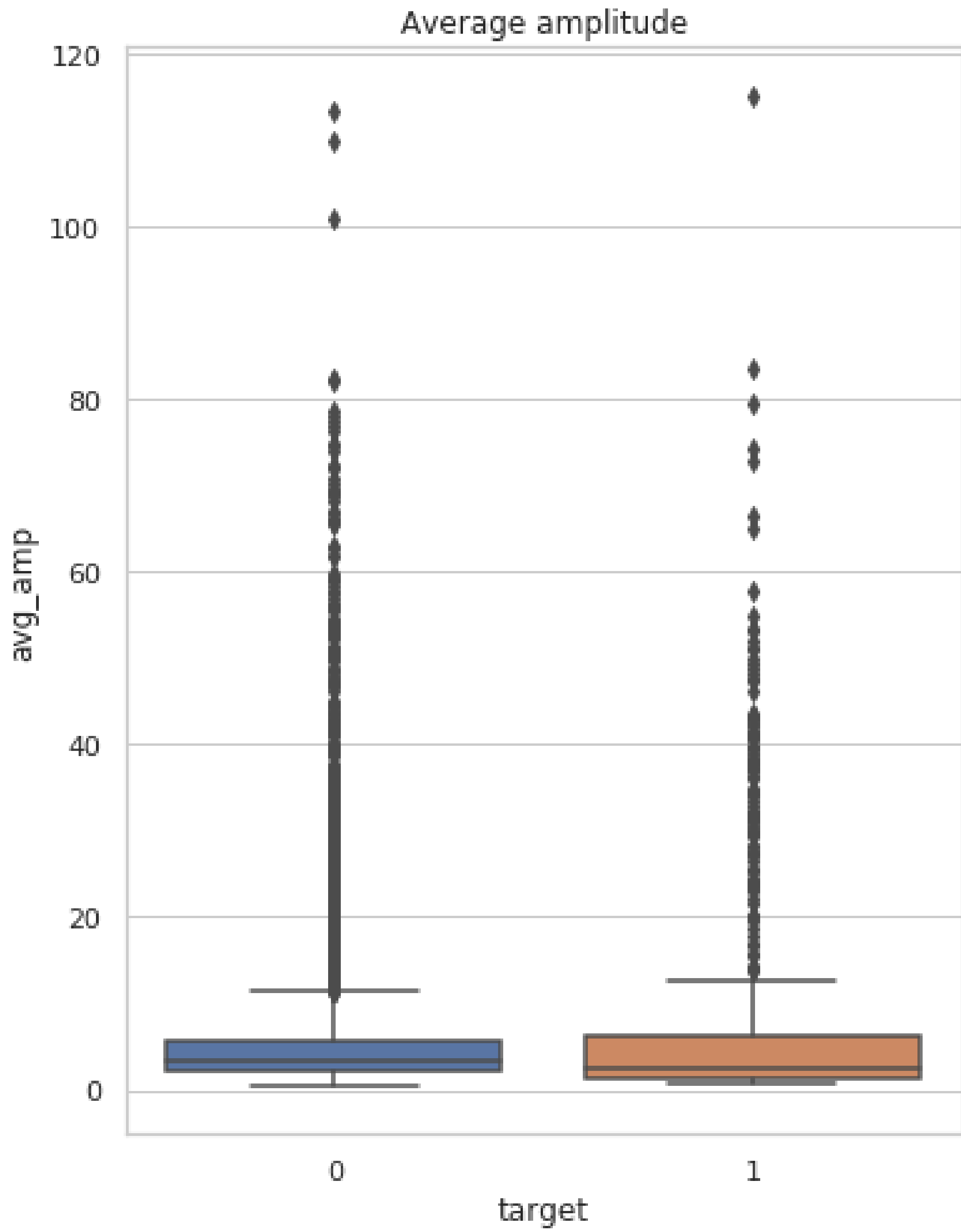

```
In [41]: plt.figure(figsize=(6,8))
         sns.set(style="whitegrid")
         plt.title("Max frequency PSD")
         ax = sns.boxplot(x="target", y="max_freq", data=train_subset_meta_df)
```



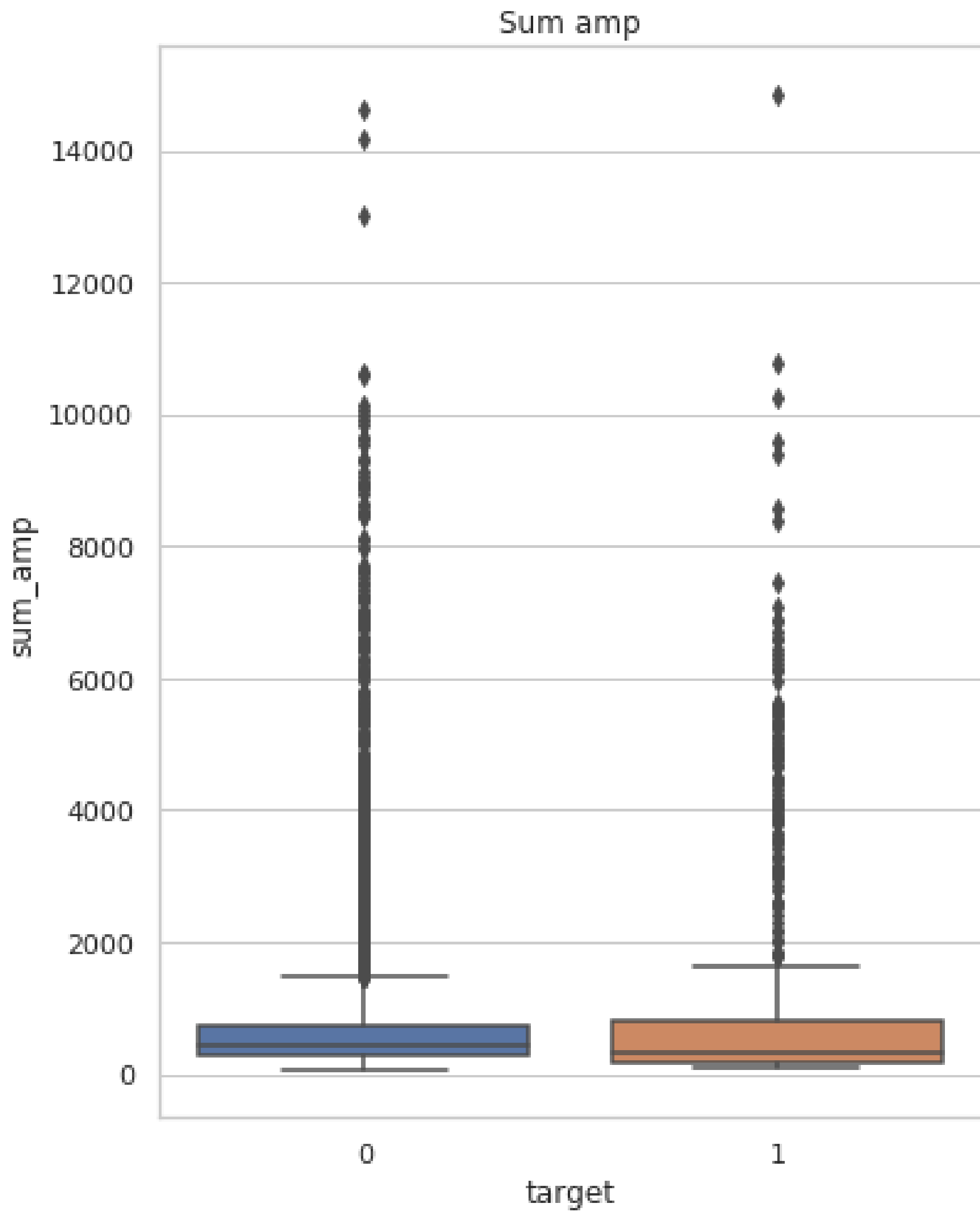

```
In [42]: plt.figure(figsize=(6,8))
         sns.set(style="whitegrid")
         plt.title("Strong amplitude count")
         ax = sns.boxplot(x="target", y="strong_amp_count", data=train_subset_meta_df)
```



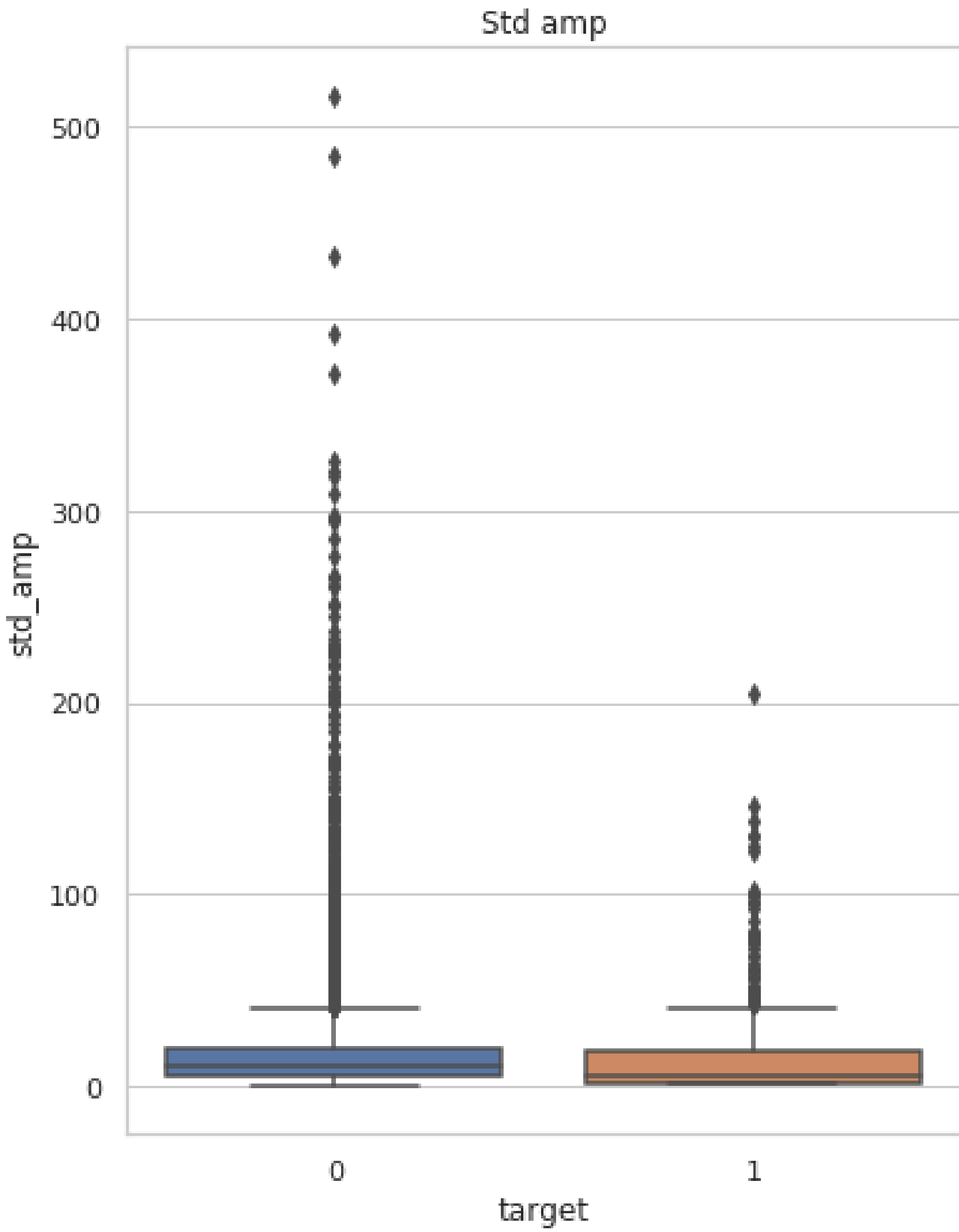

```
In [43]: plt.figure(figsize=(6,8))
         sns.set(style="whitegrid")
         plt.title("Average amplitude")
         ax = sns.boxplot(x="target", y="avg_amp", data=train_subset_meta_df)
```



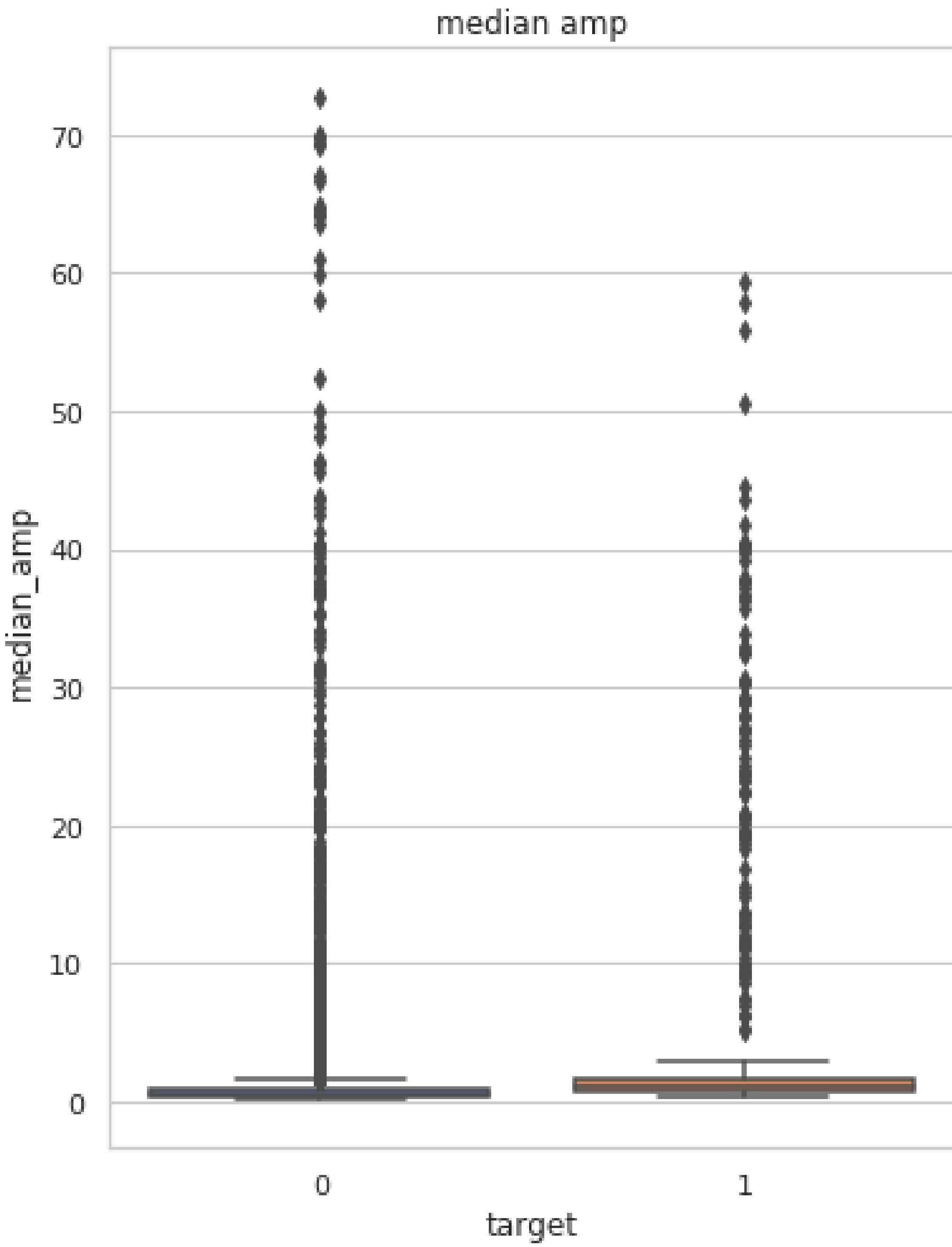

```
In [44]: plt.figure(figsize=(6,8))
         sns.set(style="whitegrid")
         plt.title("Sum amp")
         ax = sns.boxplot(x="target", y="sum_amp", data=train_subset_meta_df)
```




```
In [45]: plt.figure(figsize=(6,8))
         sns.set(style="whitegrid")
         plt.title("Std amp")
         ax = sns.boxplot(x="target", y="std_amp", data=train_subset_meta_df)
```




```
In [46]: plt.figure(figsize=(6,8))
         sns.set(style="whitegrid")
         plt.title("median amp")
         ax = sns.boxplot(x="target", y="median_amp", data=train_subset_meta_df)
```



5 Find the most important features

```
In [47]: X_cols = ["phase"] + train_subset_meta_df.columns[4:].tolist()
         X_cols

Out[47]: ['phase',
          'mean',
```

```
           'median',
           'std_dev',
           'rolling100k_amp',
           'count1SDfromTheMean',
           'count2SDfromTheMean',
           'max_amp',
           'max_freq',
           'strong_amp_count',
           'avg_amp',
           'sum_amp',
           'std_amp',
           'median_amp']
```

```
In [48]: Fvals, pvals = f_classif(train_subset_meta_df[X_cols], train_subset_meta_df["target"])

         print("F-value | P-value | Feature Name")
         print("--------------------------------")

         for i, col in enumerate(X_cols):
             print("%.4f"%Fvals[i]+" | "+"%.4f"%pvals[i]+" | "+col)
```

```
F-value | P-value | Feature Name
--------------------------------
0.0274 | 0.8686 | phase
9.2846 | 0.0023 | mean
7.0488 | 0.0079 | median
6.3229 | 0.0119 | std_dev
25.2459 | 0.0000 | rolling100k_amp
8.1104 | 0.0044 | count1SDfromTheMean
16.1775 | 0.0001 | count2SDfromTheMean
5.3612 | 0.0206 | max_amp
14.1640 | 0.0002 | max_freq
112.3396 | 0.0000 | strong_amp_count
62.2001 | 0.0000 | avg_amp
62.2001 | 0.0000 | sum_amp
3.9723 | 0.0463 | std_amp
250.2839 | 0.0000 | median_amp
```

So as expected phase is a useless feature on its own, but interestingly std_amp, median_amp, signal_std, max_amp may not be extremely useful variables because we cannot reject the null with a significance of 0.01 for these. However the features signal_mean, signal_sum, max_freq, strong_amp_count, avg_amp, and sum_amp all look like very useful features, even on their own.

```
In [49]: train_subset_meta_df.to_csv('../input/metadata_train_V2.csv')
         train_subset_meta_df.head(n=9)
```

```
Out[49]:    signal_id  id_measurement  phase  target      mean  median    std_dev  \
         0          0               0      0       0 -0.960271    -1.0  13.870724
```

```
1           1                0       1       0 -0.194125     0.0  13.037134
2           2                0       2       0 -0.043555     0.0  13.684282
3           3                1       0       1 -0.997401    -1.0  13.673630
4           4                1       1       1 -0.175586     0.0  12.938372
5           5                1       2       1 -0.036004     0.0  13.545777
6           6                2       0       0 -1.146185    -1.0  14.064211
7           7                2       1       0 -1.952695    -2.0  14.774424
8           8                2       2       0  0.873370     1.0  14.815668

   rolling100k_amp  count1SDfromTheMean  count2SDfromTheMean     max_amp  \
0         37.21537               377353                   21   47.831062
1         35.10791               372859                    7   41.982578
2         36.97624               377776                   23   38.999954
3         37.53126               381716                   28   29.060942
4         35.35856               377552                   24   20.080660
5         36.87904               379631                   68   23.466799
6         39.47469               378400                   22  120.421745
7         41.58219               391669                    1   81.411369
8         41.58396               394043                    6  123.972298

   max_freq  strong_amp_count   avg_amp     sum_amp    std_amp  median_amp
0  0.378906              18.0  1.995473  257.415955   5.220994    0.612473
1  0.378906              20.0  1.836459  236.903198   4.641738    0.458779
2  0.378906              22.0  1.935738  249.710236   4.503104    0.652928
3  0.148438              16.0  1.590296  205.148132   3.224866    0.798159
4  0.148438              19.0  1.375932  177.495178   2.577763    0.439266
5  0.148438              19.0  1.653171  213.259018   2.911619    0.874973
6  0.382812              17.0  4.404845  568.224976  15.451883    0.395055
7  0.382812              14.0  3.057476  394.414368  10.961865    0.334526
8  0.382812              16.0  3.072290  396.325439  12.887474    0.324272
```

6 Fit libGBM model and hyperparameter tuning

In [52]:
```
from keras import backend as K
from sklearn.metrics import matthews_corrcoef
```

In [54]:
```
#def mcc(y_true, y_pred, labels=None, sample_weight=None):
#    """ Matthew's coefficient correlation """
#    tn, fp, fn, tp = confusion_matrix(y_true, y_pred, labels=labels, sample_weight=sam
#    mcc = (tp*tn - fp*fn)/np.sqrt((tp + fp)*(tp + fn)*(tn + fp)*(tn + fn))
#    return mcc


#def mcc(y_true, y_pred):
#    '''Calculates the Matthews correlation coefficient measure for quality
#    of binary classification problems.
#    '''
```

```
#    y_pred_pos = K.round(K.clip(y_pred, 0, 1))
#    y_pred_neg = 1 - y_pred_pos
#
#    y_pos = K.round(K.clip(y_true, 0, 1))
#    y_neg = 1 - y_pos
#
#    tp = K.sum(y_pos * y_pred_pos)
#    tn = K.sum(y_neg * y_pred_neg)
#
#    fp = K.sum(y_neg * y_pred_pos)
#    fn = K.sum(y_pos * y_pred_neg)
#
#    numerator = (tp * tn - fp * fn)
#    denominator = K.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
#
#    return numerator / (denominator + K.epsilon())

def mcc(y_true, y_pred):
    return matthews_corrcoef(y_true, y_pred)

mcc_scorer = make_scorer(mcc)

lgbm_classifier = lgbm.LGBMClassifier(boosting_type='gbdt', max_depth=-1, subsample_for
                                      class_weight=None, min_split_gain=0.0, min_child_
                                      subsample_freq=0, random_state=rand_seed, n_jobs=

param_distributions = {
    "num_leaves": randint(16, 48),
    "learning_rate": expon(),
    "reg_alpha": expon(),
    "reg_lambda": expon(),
    "colsample_bytree": uniform(0.25, 1.0),
    "min_child_samples": randint(10, 30),
    "n_estimators": randint(50, 250)
}

clf = RandomizedSearchCV(lgbm_classifier, param_distributions, n_iter=100, scoring=mcc_
                         refit=True, cv=5, verbose=1, random_state=rand_seed, error_sco

clf.fit(train_subset_meta_df[X_cols], train_subset_meta_df["target"])
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/home/pierre/bin/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_validation.py:55
lightgbm.basic.LightGBMError: Check failed: feature_fraction <=1.0 at /tmp/pip-req-build-ztyh0k2
```

```
  FitFailedWarning)
/home/pierre/bin/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:543: Ru
  mcc = cov_ytyp / np.sqrt(cov_ytyt * cov_ypyp)
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:  3.0min finished
```

Out[54]:
```
RandomizedSearchCV(cv=5, error_score=-1.0,
          estimator=LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_b
        importance_type='split', learning_rate=0.1, max_depth=-1,
        min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
        n_estimators=100, n_jobs=1, num_leaves=31, objective='binary',
        random_state=135, reg_alpha=0.0, reg_lambda=0.0, silent=True,
        subsample=1.0, subsample_for_bin=200000, subsample_freq=0),
          fit_params=None, iid=True, n_iter=100, n_jobs=1,
          param_distributions={'num_leaves': <scipy.stats._distn_infrastructure.rv_froz
          pre_dispatch='2*n_jobs', random_state=135, refit=True,
          return_train_score=True, scoring=make_scorer(mcc), verbose=1)
```

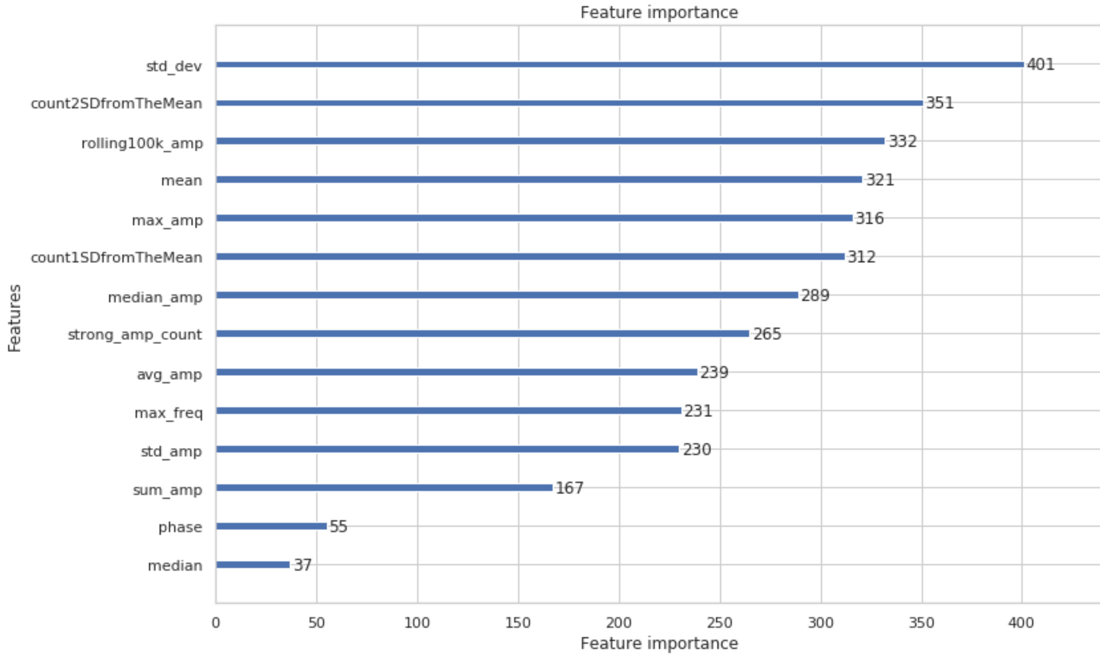
In [55]:
```
print(clf.best_score_)
```

```
0.5401876970150516
```

In [57]:
```
clf.best_estimator_
```

Out[57]:
```
LGBMClassifier(boosting_type='gbdt', class_weight=None,
        colsample_bytree=0.456616770229323, importance_type='split',
        learning_rate=0.4118819089418852, max_depth=-1,
        min_child_samples=24, min_child_weight=0.001, min_split_gain=0.0,
        n_estimators=197, n_jobs=1, num_leaves=19, objective='binary',
        random_state=135, reg_alpha=0.28274323494050946,
        reg_lambda=1.5600109217504974, silent=True, subsample=1.0,
        subsample_for_bin=200000, subsample_freq=0)
```

In [58]:
```
fig, ax = plt.subplots()
fig.set_size_inches(xsize, ysize)

lgbm.plot_importance(clf.best_estimator_, ax=ax)

plt.show()
```




```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```