SyedHassaanTauqeer 19-01-2019 Final Code Notebook Draft

In [1]:
```
import os, glob
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import collections
import datetime
import sklearn.cluster as skc
from sklearn.model_selection import train_test_split
from sklearn import svm
import pickle
import warnings
import itertools
warnings.filterwarnings("ignore")
plt.style.use('fast')
import statsmodels.api as sm
import operator
import matplotlib
%matplotlib inline
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'

import json
```

In [2]:
```
def file_write(df, path, name):
    #This is a simple function that writes a dataframe to a csv file
    return df.to_csv(path+name+'.csv', sep = ';', index = False)#Making index
 False helps ignore index column when writing file
```

In [3]:
```
def floatConv(arr):#This function takes in a python list/ numpy array and repl
aces any commas left in the thousand position in
    #numerical values. This stands as a safeguard to ensure all values are flo
at for easier future operations
    temp = []
    for i in range(len(arr)):
        temp.append( float(str(arr[i]).replace(",", "") ) )
    return temp
```

In [4]:
```
def perScale(arr):#This function is used to scale a list or numpy array to per
centage ratio.
    #This is done by taking the maximum value in that array and scaling every
 other value relatively. No trailing decimal places
    #left for the sake of ease in matching and querying at future stages
    scaled = []
    tempMax = max(arr)
    for i in range(len(arr)):
        scaled.append( round(float(arr[i]/tempMax)*100, 0) )
    return scaled
```

In [5]:
```
def minFil(arrList, win):#This function takes in a list/numpy array and applies a rolling window operation to get the minimum
    #value within that window.
    tempDiff = arrList
    appVal = arrList[len(arrList)-1]# To ensure the input and output lengths of the array are same, the last value of the input
    #array is repeated the number of times that matches one less than the window size
    tempDiff = np.insert(tempDiff, len(tempDiff), np.repeat(appVal, win-1))

    arrListDF = pd.DataFrame({"arr":tempDiff})#since the rolling function in pandas is really efficient, we convert the array
    #to Series data
    tempDiff = arrListDF.rolling(win).min().dropna()#all NaN values are dropped to not cause issues in future operations
    tempDiff = tempDiff['arr'].values

    return tempDiff
```

In [23]:
```
def diffIndexMaker(arr):#This function takes in an numpy array/list and computes the difference in between each consecutive value
    #To ensure that the input and output lengths of the array remain same, the first value of the array is pre-pended.
    #This doesn't affect the computation or any future operations but keeps length consistent
    temp = arr.item(0)
    arr = np.insert(arr, 0, temp)#pre-pending the first value to the array
    return np.diff(arr)
```

In [7]:
```
def TestPairGen(df):#This function follows the exact same pattern as the training pair generator function
    #It just requires the testing dataframe instead and does not produce a separate label array from the testing pair
    pair = []
    for index, rows in df.iterrows():
        pair.append( ( rows['scaledTime'].astype(float), rows['scaledPower'].astype(float),
                      rows['scaledPowMin'].astype(float), rows['scaledPowDiff'] ) )

    return pair
```

In [8]:
```
def thresher(DF):#This is the most important function for the heat phase detec
tion
    #It works on the assumption that the incoming dataframe has three main par
ts (heat, spin+end and noise)
    #The noise is the most occuring element and becomes an easy target for eli
mination
    #The spin+endspin is seperated by the sureity that the end always exists i
n the last time stamps
    #Whatever is left is thus the heating phase
    df = pd.DataFrame({"p": DF['pow'], "time": np.arange(len(DF['pow'].values
))})
    x_tag = df['time'].values

    print "length: ", len(df)

    X = df['p'].values.reshape(-1, 1)#single feature re-shaping for cluster pr
ediction - KMeans requires this shape
    model = skc.KMeans(n_clusters=3)#Base assumption that data always has the
 3 parts(heat, spin+end and noise)
    y_pred = model.fit_predict(X)

    #dictionary structure to store all predicted cluster labels against respec
tive time stamps
    refTsPowDict = collections.OrderedDict()

    for i in range(len(x_tag)):
        refTsPowDict[x_tag[i]] = y_pred[i] #populating dictionary

    clustCount = collections.Counter(y_pred)#counting labels to see which labe
ls has how many points
    #this is done because the segments are not always uniformly labelled
    #e.g heat phase might get label '1' in one case and the very next machine
 iteration may give heat phase a label of '0'
    looper = len(x_tag)
    for  i in range(looper): #deleting noise
        if refTsPowDict[x_tag[i]] == keywithmaxval(clustCount):#gets the clust
er key with most frequent value(noise timestamps)
            del refTsPowDict[x_tag[i]]#delete dictionary entry, theeby deletin
g time stamps of noise

    last_key = refTsPowDict.keys()[-1]#possible end spin. Gives the very last
 time stamp which has to be end spin in theory.
    H_timeArr = [] #time stamps for heating phase
    comb_timeArr = [] #time stamps for last phases
    for key, value in refTsPowDict.items():
        if value  ==  refTsPowDict[last_key]:#query all dict on this value to
 get keys for this cluster
            comb_timeArr.append(key)#if query matches then it is from the spin
+end group
        else:
            H_timeArr.append(key)#otherwise it's heat

    p_val = []
    for i in range(len(H_timeArr)): #create and return dataframe for heat phas
e
        p_val.append(df[df['time'] == H_timeArr[i]]['p'].values[0])
```

```
    hDF = pd.DataFrame({'time': H_timeArr, 'pow':p_val})

    return hDF
```

In [9]:
```
def keywithmaxval(d): #This function just takes in a dictionary and returns ke
y that corresponds to the maximum value among
    #the dictionary values
    #for example {A:2, B:6} then the function returns B
    v=list(d.values())
    k=list(d.keys())
    return k[v.index(max(v))]
```

In [10]:
```
def binarySegmenter(arr):# This function works with the "Segmenter" function.
 It's function is to create an array which contains
    #the time stamps where a positive prediction(where value is 1) occurs. It
 is used mainly by the predicted vectors.
    flag = 1#positive prediction
    Seg = []
    for i in range(len(arr)):
        if arr[i]==flag: #phase started
            Seg.append(i)
    return np.asarray(Seg)


def Segmenter(Seg): #This function is very important to find the togetherness
 of the clustered time stamps and at the same time
    #break up a single segment(timestamp array) which has a huge difference be
tween it's values into separate segments
    #for example a segment/timestamp array is as follows {12, 13,15, 18, 55, 6
0, 66, 69}
    #this array definitely needs to be broken up in to {12, 13,15, 18} and {5
5, 60, 66, 69}
    #to segment the labelled predictions. Usable for spins & end spin.

    #We also need to see if there are a lot of confident points nearby or if t
here is a sporadic scatter of timestamps.
    #in this case we need to get rid of them because they will act as noise an
d result in un-confident labels being considered
    #as actual labels.
    threshold = 3 #minimum number of points in a segment to qualify as a confi
dent prediction/group

    # deciding metric to  find breakers in sequence
    #https://www.researchgate.net/post/Can_anybody_suggests_me_how_to_split_a_
continuous_variable_into_high_and_low_value_group_for_testing_its_moderating_e
ffect

    diffArr = diffIndexMaker(Seg)#gets the differences between the timestamps
    iqr = np.percentile(diffArr, 75) - np.percentile(diffArr, 25)#inter-quarti
le range

    count = 0#counts segments
    breakers = []#contains the index of the timestamp where the break occurs

    buff = {}#dictionary that contains the
    prev = 0 #last marker for subsetting
    for i in range(len(diffArr)):

        if Seg[i] == 0:#only usable in the case of heat phase
            #if the timestamps carry a '0' point that means the heat phase was
 started at time 't0', which is not
            #possible because a cycle never starts from heat phase on 't0'. Th
is case is only possible if the previous
            #wash cycle was not completed and carried up in this cycle. We jus
t consider it as an invalid cycle.
            print 'INVALID CYCLE - Previous continuation'

        else:
```

```
            temp = float(diffArr[i])/float(Seg[i])*100

            if temp >(float(np.median(diffArr)) + float(iqr)):#filtering on me
dian and inter-quartile range
            #since data is assumed to be non-centered and un-symmetric

                breakers.append((diffArr[i], i))
                count +=1
            count += 1#this is the number of segments  ---|---|--- in this cas
e breakers=2 and segments=3
    if not breakers: #only one segment

        buff[0]  = Seg[prev:len(Seg)]

    if not (not (breakers)):#ensures that there is at least one breaker which
 means two segments
        for i in range(len(breakers)):
            buff[i] = Seg[prev:breakers[i][1]]

            if len(Seg[prev:breakers[i][1]])<threshold:#for entries less than
 thresh utilize the loop
                del buff[i]

            prev = breakers[i][1]

        buff[len(breakers)] = Seg[prev:len(Seg)]# last segment added manually

        if len(Seg[prev:len(Seg)])<threshold:#for entries less than thresh uti
lize the loop
            del buff[len(breakers)] #entries in segment less confidently predi
cted are deleted

            # e.g 3 points in a prediction segment are at least required for i
t to be labeled a separate segment

    return Seg, buff
```

In [11]:
```
def gapFinder(_dict, thresh):#This function finds gaps between dictionary entr
ies of segments, which are basically timestamps.
    #If the gaps are big enough it lets them be otherwise it fuses them togeth
er to make a shorter dictioanry
    #It works sort of in the opposite manner as the Segmenter function
    #e.g {A:[1,2,4,5,6], B:[10,12,14,17], C:[50,55,59,62]} will become {A:[1,
2,4,5,6,10,12,14,17], C:[50,55,59,62]}
    k = _dict.keys()

    if len(k) ==  1: # first case no pre return the only time markers
        return _dict
    else:
        newArr = []
        newDict  = collections.OrderedDict()
        firstPhase = _dict[k[0]] #load initial dictionary

        for i in range(len(k)):
            if i == 0: continue

            tempPrev = firstPhase
            lastPrev  = firstPhase[len(firstPhase)-1]
            tempCurrent = _dict[k[i]]
            firstCurrent = tempCurrent[0]
            gap = firstCurrent  - lastPrev
            # the gap has to be big enough relative to the last point in the p
revious array
            if ( float(gap)/float(lastPrev) )  < thresh:#acceptible diff means
 fuse this value with previous and delete this entry

                newArr = np.concatenate((tempPrev, tempCurrent),axis=None)

                firstPhase = newArr #set this as previous


            else:
                newDict[i] = tempPrev
                firstPhase = tempCurrent

        newDict[k[len(k)-1]+1] = newArr

        return newDict
```

In [12]:
```
def longestRecKey(_dict):# This function just finds the longest valued array i
n a dictionary and returns that value array
    _valLen = []
    k = _dict.keys()
    v = _dict.values()
    for items in v: _valLen.append(len(items))
    return  _dict[k[_valLen.index(max(_valLen))]]
```

In [1]:
```
def preFunc(_dict):#This function analyses the possiblity of a pre-wash cycle
 existing, in case it does it identifies
    #the markers of the phase
    #Along that it also identifies the main heating phase timestamp array
    _dict = gapFinder(_dict, 0.5) # if the gap is half of previous segment's l
ast entry, keep it else it's not a wide enough gap
    preStart, preEnd = 0, 0
    preFlag = False
    k = _dict.keys()
    if len(k) ==  1: # first case no pre return the only time markers
        return preFlag, preStart, preEnd, _dict[k[0]]
    else:
        firstPhase = _dict[k[0]] #load initial dictionary
        longestPhase = longestRecKey(_dict)

        if np.array_equal(firstPhase, longestPhase):
            return preFlag, preStart, preEnd, longestPhase

        elif _dict.values().index(longestPhase) > _dict.values().index(firstPh
ase):
            preFlag = True
            preStart = firstPhase[0]
            preEnd  = firstPhase[len(firstPhase)-1]
            return preFlag, preStart, preEnd, longestPhase
```

In [14]:
```
def heatAnalysis(_dict, df):#This function takes in the heating phase datafram
e that comes after the cluster(KMeans)processing
    #and the dictionary that comes after the segmenting of the heating phase
    #This function provides the markers for the heating phase and the energy c
alculation by integration to stay precise
    Phase_start, Phase_end = 0, 0
    Energy = 0

    preFlag, preStart, preEnd, timeArr = preFunc(_dict)

    Phase_start = timeArr[0]
    Phase_end  = timeArr[len(timeArr)-1]

    plt.axvline(x=Phase_start, color='r')
    plt.axvline(x=Phase_end, color='r')

    #The energy is calculated by a simple formula
    #The power for each of the heat phase timestamps is added
    #The sum is multiplied by 4 because the timestamps have an average differe
nce of 4
    #and divided by 3600 to convert seconds to hours for the energy to be in W
att Hour.
    Energy = float(np.sum(df[df['time'].isin(timeArr)]['pow'].values)) * (floa
t(4)/float(3600))

    return Energy, Phase_start, Phase_end
```

In [15]:
```
def endAnalysis(_dict, df):#This function analyses the main dataframe for end
 cycle information.
    #It also takes in the end cycle prediction vector that has been segmented
 by the Segmenter function
    #generally assuming that even if more than one end spin detections then th
e longest one is relevant
    Phase_start, Phase_end = 0, 0
    Energy = 0
    timeArr = longestRecKey(_dict)
    Phase_start = timeArr[0]
    Phase_end  = timeArr[len(timeArr)-1]

    plt.axvline(x=Phase_start, color='g')
    plt.axvline(x=Phase_end, color='g')

    #The energy is calculated on the same principle as the heat phase
    #Each 4 second server ping is converted to hours by division by 3600
    #This is then multiplied to the sum of the end phase power sum to give the
 energy in Watt Hour
    Energy = float(np.sum(df[df['time'].isin(timeArr)]['pow'].values)) * (floa
t(4)/float(3600))

    return Energy, Phase_start, Phase_end
```

In [16]:
```
def spinAnalysis(_dict, df):#This function analyses the spins in a single cycl
e. It requires the main dataframe and the
    #dictionary that results from the segmentation of the spin predicted vecto
r.
    k = _dict.keys()
    Energies = []
    phaseStarts = []#all spin phase starts are stored here
    phaseEnds = []#all spin end phases are stored here
    spins = len(k)#gives the number of spins

    for i in range(spins):
        Energy = 0
        timeArr = _dict[k[i]]

        Phase_start = timeArr[0]
        Phase_end  = timeArr[len(timeArr)-1]
        phaseStarts.append(Phase_start)
        phaseEnds.append(Phase_end)

        plt.axvline(x=Phase_start, color='b')
        plt.axvline(x=Phase_end, color='b')

        Energy = float(np.sum(df[df['time'].isin(timeArr)]['pow'].values)) * (
float(4)/float(3600))
        Energies.append(Energy)
    #It returns the number of spins, their markers in two lists and their mean
 energies if more than one spin exist
    return spins, phaseStarts, phaseEnds, np.mean(Energies)
```

Testing Phase to begin from here

The test dataframe is read into the notebook. The models for spin and end spin detection are then loaded in. A JSON file composed of the entire analysis result is then generated after the analysis runs.

In [17]:
```
_file = 'C:\\Users\\Labyrinth\\JUPYTER NOTEBOOKS\\WeWash_Praktikum_TUM3sem\WeW
ash_Analysis_ver2\\Data\\intermediate\\labelled\\test\\UniLabeled_data_50_1432
00-143324.csv'
labtsDF = pd.read_csv(_file, delimiter=';')
labtsDF.head(2)
```

Out[17]:

	machine	pow	time	spins	end_spin
0	143200	0.07	0	NaN	NaN
1	143200	0.06	1	NaN	NaN

In [18]:
```
machines = labtsDF['machine'].unique() #Uniques machines are filtered based on
 id
print 'machines: ', len(machines)
```

```
machines:  50
```

In [19]:
```
testDF = pd.DataFrame()
for i in range(len(machines)):#Each of the machines has multiple records.
    tempDF = labtsDF[labtsDF['machine'] == machines[i]]#now that we have a sub
set of the dataframe filtered on
    #the machine, we'll add features to it and then append it to the test data
frame.
    tempDF['pow'] = floatConv(tempDF['pow'].values)#float conversions
    tempDF['time'] = floatConv(tempDF['time'].values)#float conversions
    tempDF['scaledPower'] = perScale(tempDF['pow'].values)#percentage scaling
    tempDF['scaledTime'] =  perScale(tempDF['time'].values)#percentage scaling
    tempDF['powMin'] = minFil(tempDF['pow'].values, win=5)#applying minimum fi
lter to main power values
    tempDF['scaledPowMin'] = minFil(tempDF['scaledPower'].values, win=5)#apply
ing minimum filter to scaled power values
    tempDF['scaledPowDiff'] = diffIndexMaker(tempDF['scaledPower'].values)#cre
ating the difference index from scaled power
    testDF = testDF.append(tempDF)
testDF.head(2)
```

Out[19]:

	machine	pow	time	spins	end_spin	scaledPower	scaledTime	powMin	scaledPowM
0	143200	0.07	0.0	NaN	NaN	0.0	0.0	0.06	0.0
1	143200	0.06	1.0	NaN	NaN	0.0	0.0	0.06	0.0

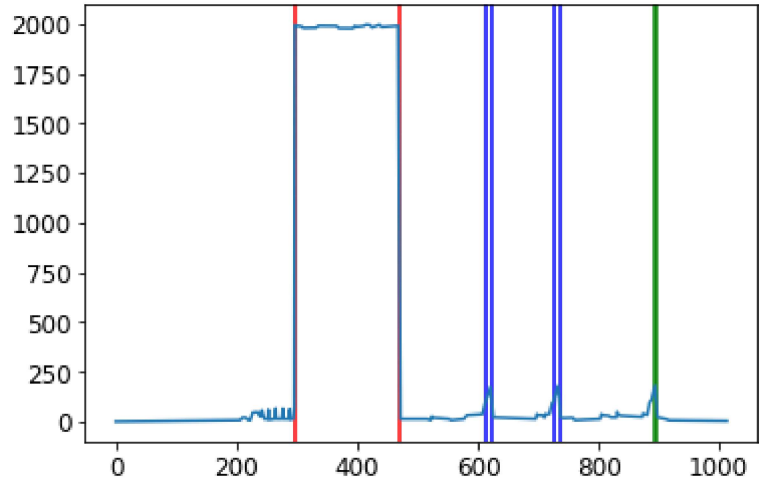
In [20]:
```
#It is extremely important to fill all NaN values because otherwise it will ha
mper the testing process
testDF['end_spin'] = testDF['end_spin'].fillna(0.0)
testDF['spins'] = testDF['spins'].fillna(0.0)
testDF['pow'] = testDF['pow'].fillna(0.0)
```

In [31]:
```
#Loading the SVM classifiers
_filePath = 'C:\\Users\\Labyrinth\\JUPYTER NOTEBOOKS\\WeWash_Praktikum_TUM3sem
\\WeWash_Analysis_ver2\\Models\\150_146_400_4f\\'
loaded_spinModel = pickle.load(open(_filePath+'SVM_spinModel4f_146-150-400.pk
l', 'rb'))
loaded_endModel = pickle.load(open(_filePath+'SVM_endModel4f_146-150-400.pkl',
 'rb'))
```

In [34]:
```
jsonDF  = pd.DataFrame(columns=['mac', 'pre', 'heat', 'spin', 'end', 'avgCycle
Energy'])

for i in range(len(machines)):#THe analysis is run on each machine sample indi
vidually
    testSampleDF = testDF[testDF['machine'] == machines[i]]

    print 'machine#', machines[i]
    sample = np.asarray(TestPairGen(testSampleDF))
#     print 'samp: ', sample
    TestPred_spin = loaded_spinModel.predict(sample)
    TestPred_end = loaded_endModel.predict(sample)


    #1 Check for end
    if np.count_nonzero(TestPred_end)>1:#If there is no  end spin,  there's no
 point analysing that cycle
    #1.1 if not end then invalid cycle
        tempLab, tempBuff = Segmenter(binarySegmenter(TestPred_end))

#         #2 If valid then run end analysis
        endDict = {}
        endEnergy, end_start, end_end = endAnalysis(tempBuff, testSampleDF)
        endDict['endEnergy'] = endEnergy
        endDict['end_start'] = end_start
        endDict['end_end'] = end_end
        print'End  Energy:', endEnergy, '\tEnd Start:', end_start, '\tEnd en
d:', end_end

        #2.1  segment hDF & run analysis
        hDF = thresher(testSampleDF)
        tempLab, tempBuff = Segmenter(hDF['time'].values)
        #2.2 check pre  && get markers
        preDict = collections.OrderedDict()
        prePhaseFlag, pre_start, pre_end, longestPhase = preFunc(tempBuff)
        preDict['preFlag'] = prePhaseFlag
        preDict['pre_start'] = pre_start
        preDict['pre_end'] = pre_end
        print'Pre-Phase:', prePhaseFlag, '\tprePhase Start:', pre_start, '\tpr
ePhase end:', pre_end

        #3Run heat analysis
        heatDict = {}
        heatEnergy, heat_start, heat_end = heatAnalysis(tempBuff, hDF)
        heatDict['heatEnergy'] = heatEnergy
        heatDict['heat_start'] = heat_start
        heatDict['heat_end'] = heat_end
        print'Heat  Energy:', heatEnergy, '\tHeat Start:', heat_start, '\tHeat
 end:', heat_end


        #4 segment spins
        spinDict = {}
        if np.count_nonzero(TestPred_spin)>1:
            tempLab, tempBuff = Segmenter(binarySegmenter(TestPred_spin))
            spinNo, spin_start, spin_end, avgSpinEnergy = spinAnalysis(tempBuf
```

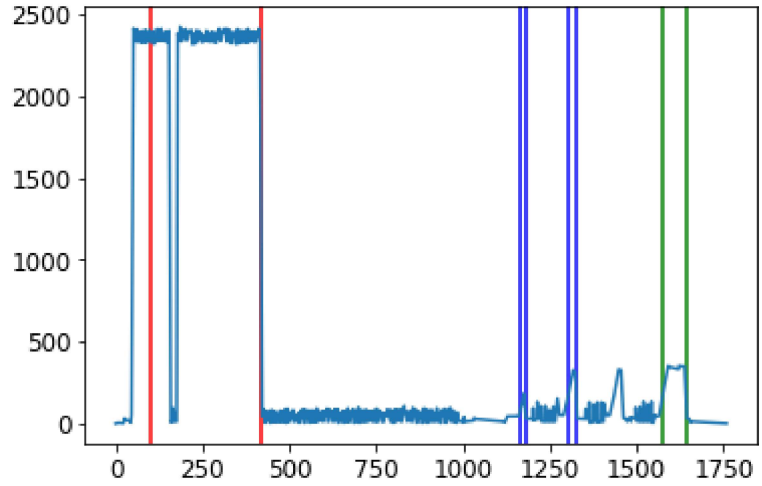
```
f, testSampleDF)
            spinDict['spins'] = spinNo
            spinDict['avgEnergy'] = avgSpinEnergy
            for j in range(spinNo):
                spinDict[j+1] = (spin_start[j],  spin_end[j])
            print'No. of Spins:', spinNo, '\tSpin Start:', spin_start, '\tSpin
 end:', spin_end, '\tAvg.  Spin Energy', avgSpinEnergy
        else:
            spinDict['spins'] = 0
            spinDict['avgEnergy'] = 0
            spinDict[0] = (0,  0)


        #4.1 plot all markers and sample
        plt.plot(np.arange(len(testSampleDF['powMin'].values)), testSampleDF[
'powMin'].values)


        #5 Avg cycle energy
        avgCycEnergy = np.mean(testSampleDF['pow'].values)
        print 'avg energy for cycle: ', avgCycEnergy

        #6 JSON export
        record = []
        record.append(machines[i])
        record.append(preDict)#pre
        record.append(heatDict)#heat
        record.append(spinDict)#spin
        record.append(endDict)#end
        record.append(avgCycEnergy)#avg cycle energy
        jsonDF.loc[len(jsonDF)] = record


    plt.show()
    print '\n'
```

```
machine# 143200
End  Energy: 0.438966666667     End Start: 894  End end: 897
length:  1014
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 395.591366667     Heat Start: 297         Heat end: 472
No. of Spins: 2         Spin Start: [613, 725]  Spin end: [624, 735]    Avg.
Spin Energy 1.3334055555555557
avg energy for cycle:  382.8820808678501
```



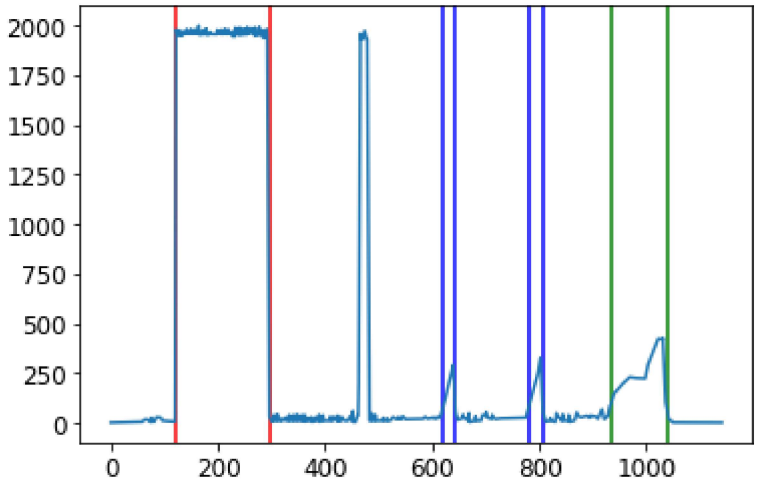

```
machine# 143201
```

```
machine# 143203
End  Energy: 24.5730444444      End Start: 1575         End end: 1642
length:  1760
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 811.083544444     Heat Start: 99  Heat end: 420
No. of Spins: 2         Spin Start: [1167, 1305]        Spin end: [1179, 132
6]      Avg.  Spin Energy 1.8955499999999996
avg energy for cycle:  559.9994431818182
```

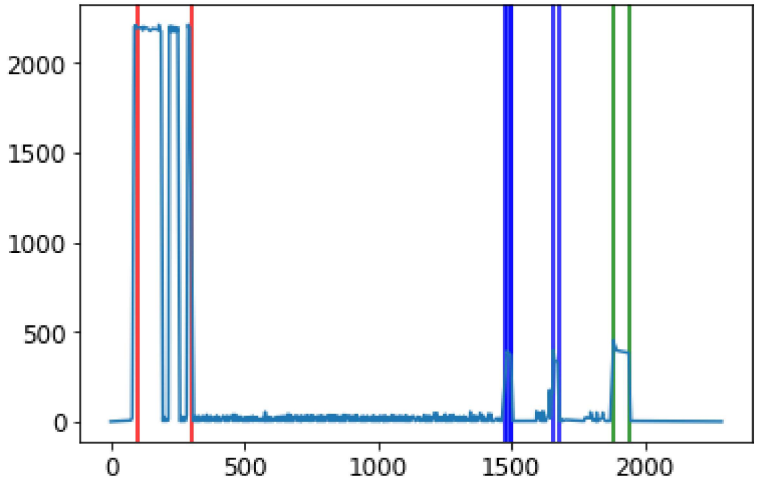


```
machine# 143208
```

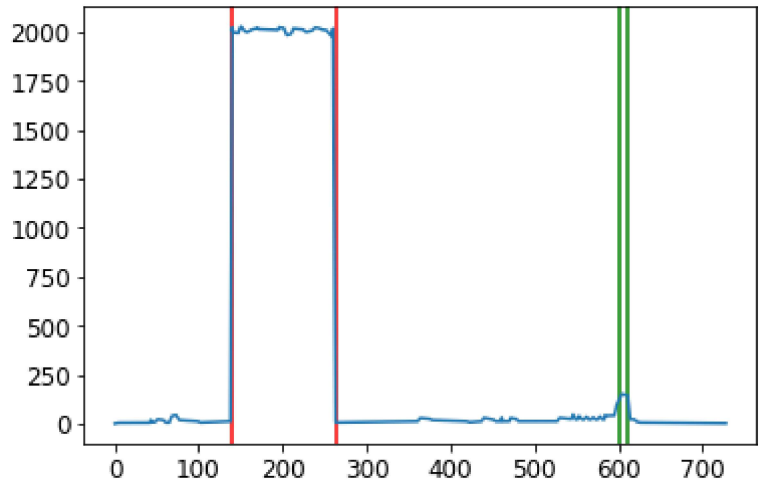
```
machine# 143211
End  Energy: 30.5979888889      End Start: 935  End end: 1039
length:  1142
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 388.288844444     Heat Start: 121         Heat end: 296
No. of Spins: 2         Spin Start: [620, 781]  Spin end: [643, 806]    Avg.
Spin Energy 4.936361111111111
avg energy for cycle:  398.3797635726794
```




```
machine# 143213
End  Energy: 25.2786222222      End Start: 1882         End end: 1944
length:  2289
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 367.487911111     Heat Start: 99  Heat end: 301
No. of Spins: 3         Spin Start: [1480, 1499, 1657]  Spin end: [1484, 150
2, 1677]        Avg.  Spin Energy 3.5522962962962965
avg energy for cycle:  208.86472695500217
```



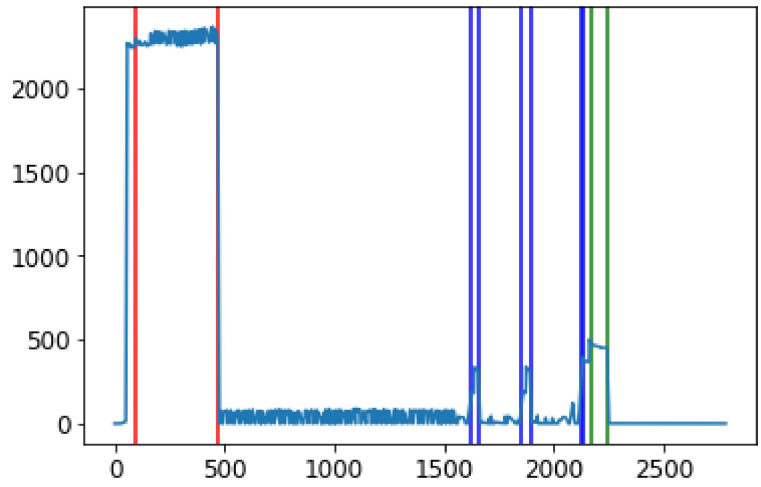
```
machine# 143216
End  Energy: 1.30528888889      End Start: 601  End end: 611
length:  729
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 285.919666667     Heat Start: 139         Heat end: 264
avg energy for cycle:  384.8243072702332
```



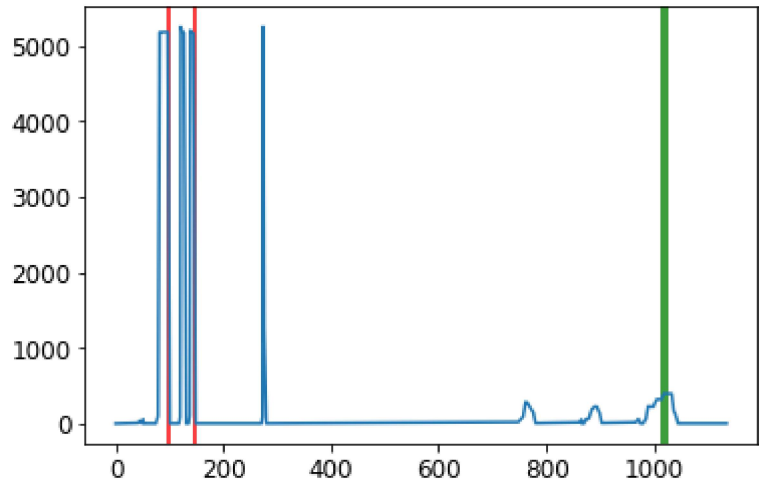

```
machine# 143217
```

```
machine# 143219
```

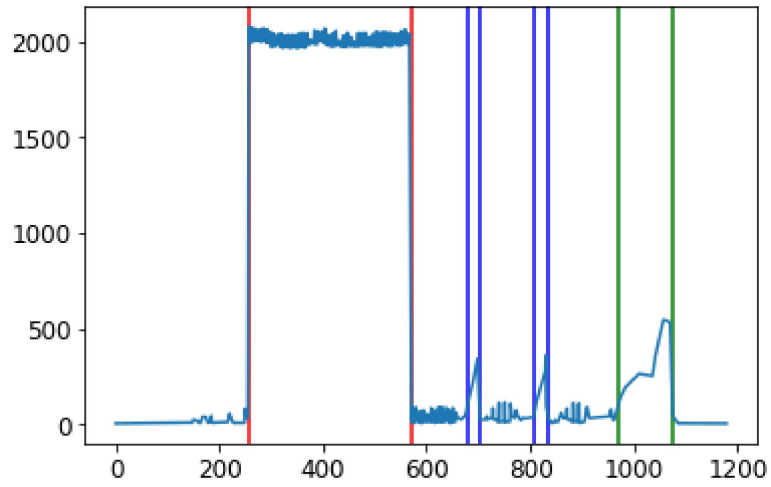
```
machine# 143223
End  Energy: 36.8869777778      End Start: 2166         End end: 2241
length:  2782
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 973.891933333     Heat Start: 99  Heat end: 474
No. of Spins: 3         Spin Start: [1618, 1853, 2121]  Spin end: [1659, 189
3, 2135]        Avg.  Spin Energy 5.2876962962962955
avg energy for cycle:  420.25621854780735
```



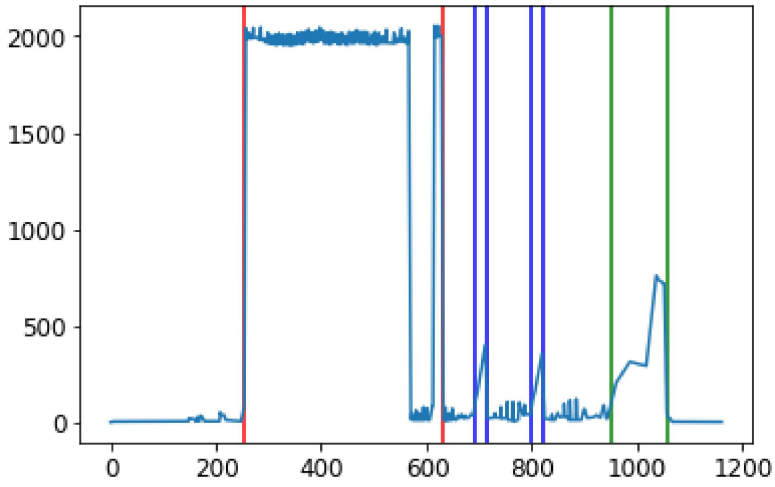
```
machine# 143224
End  Energy: 1.68832222222      End Start: 1016         End end: 1022
length:  1136
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 145.678055556     Heat Start: 99  Heat end: 149
avg energy for cycle:  309.8580193661972
```



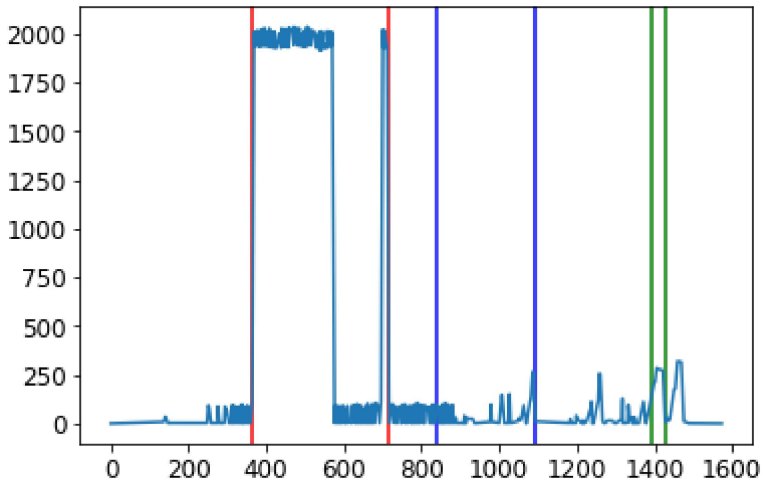

```
machine# 143225
End  Energy: 37.8396777778      End Start: 969  End end: 1076
length:  1180
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 718.517033333     Heat Start: 256         Heat end: 571
No. of Spins: 2         Spin Start: [679, 809]  Spin end: [702, 834]    Avg.
Spin Energy 5.873722222222223
avg energy for cycle:  611.2926186440677
```



```
machine# 143226
End  Energy: 44.0542111111      End Start: 949  End end: 1058
length:  1161
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 754.146322222     Heat Start: 255         Heat end: 632
No. of Spins: 2         Spin Start: [690, 800]  Spin end: [716, 823]    Avg.
Spin Energy 6.527677777777778
avg energy for cycle:  654.3586649440138
```

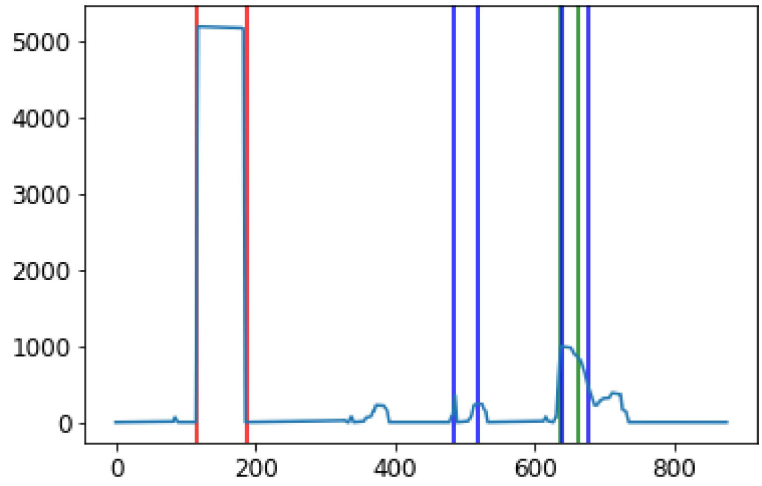



```
machine# 143228
End  Energy: 8.81798888889      End Start: 1393         End end: 1426
length:  1574
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 512.251277778     Heat Start: 365         Heat end: 715
No. of Spins: 1         Spin Start: [837]       Spin end: [1091]        Avg.
Spin Energy 3.1690444444444443
avg energy for cycle:  354.0400381194409
```

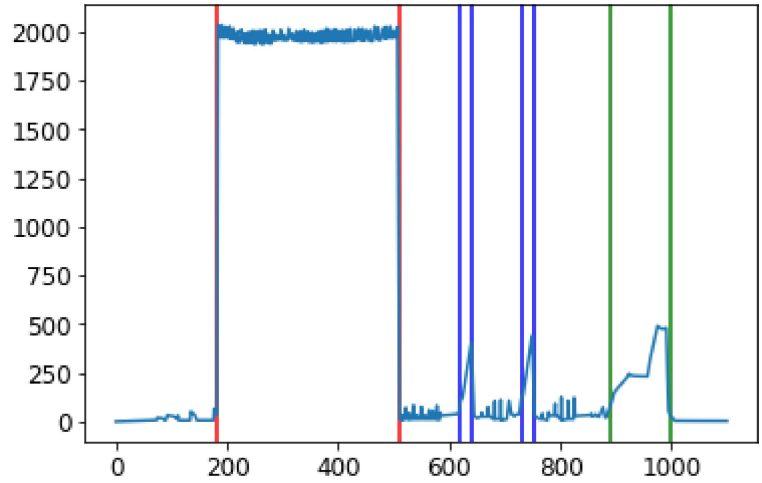


```
machine# 143229
```

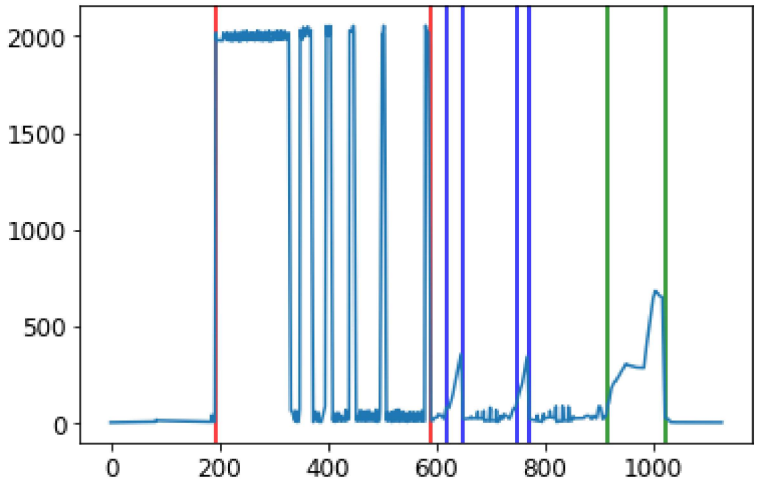
```
machine# 143231
End  Energy: 23.3460777778      End Start: 638  End end: 663
length:  876
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 420.029377778     Heat Start: 116         Heat end: 187
No. of Spins: 2         Spin Start: [484, 639]  Spin end: [519, 677]    Avg.
Spin Energy 4.352716666666667
avg energy for cycle:  557.3858675799087
```



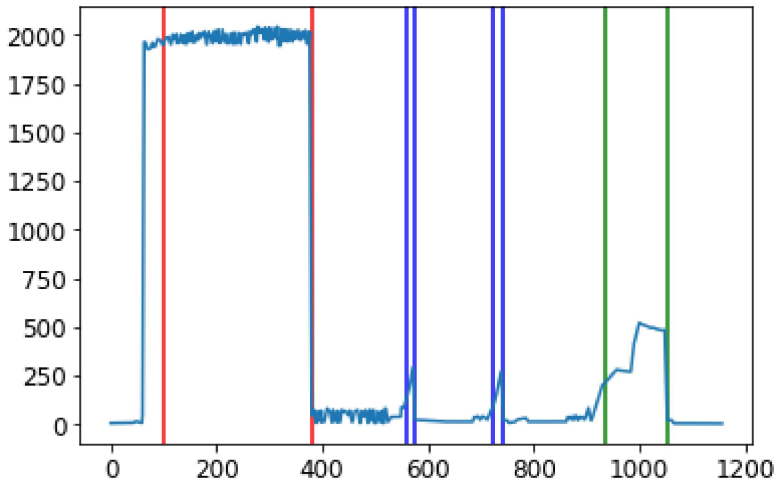

```
machine# 143233
End  Energy: 34.0572    End Start: 890  End end: 997
length:  1101
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 734.506088889     Heat Start: 183         Heat end: 511
No. of Spins: 2         Spin Start: [621, 730]  Spin end: [640, 753]    Avg.
Spin Energy 5.5652333333333335
avg energy for cycle:  664.6958855585832
```



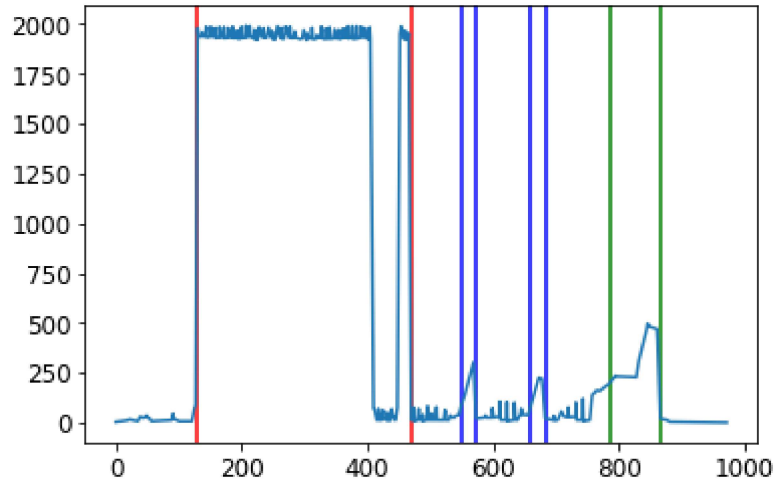
```
machine# 143237
End  Energy: 45.0442888889      End Start: 914  End end: 1023
length:  1126
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 485.455722222     Heat Start: 193         Heat end: 589
No. of Spins: 2         Spin Start: [619, 747]  Spin end: [649, 771]    Avg.
Spin Energy 6.447644444444444
avg energy for cycle:  462.08388099467146
```



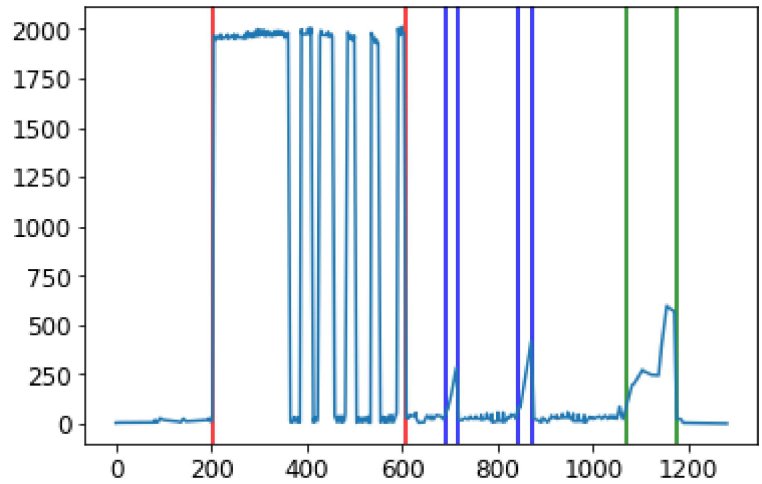

```
machine# 143246
End  Energy: 51.1375888889      End Start: 933  End end: 1052
length:  1156
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 632.167422222     Heat Start: 99  Heat end: 380
No. of Spins: 2         Spin Start: [560, 721]  Spin end: [575, 742]    Avg.
Spin Energy 3.8300666666666663
avg energy for cycle:  634.1686505190312
```



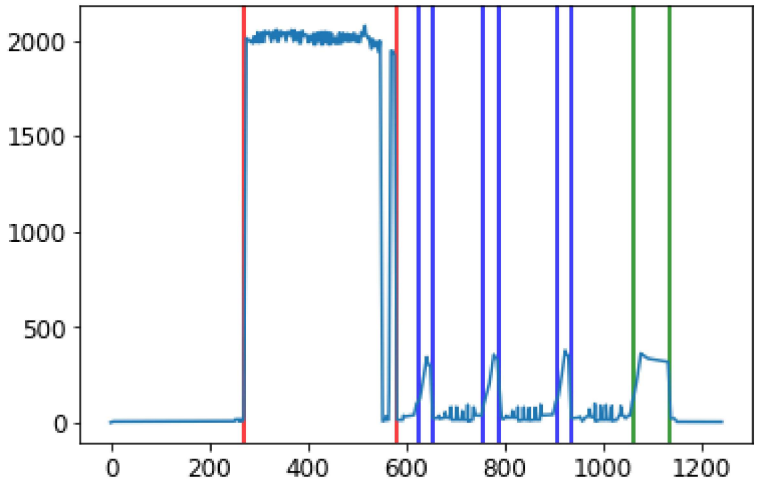
```
machine# 143249
End  Energy: 28.7423111111      End Start: 787  End end: 867
length:  973
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 669.085433333     Heat Start: 128         Heat end: 470
No. of Spins: 2         Spin Start: [551, 660]  Spin end: [573, 683]    Avg.
Spin Energy 4.944816666666666
avg energy for cycle:  684.1379445015416
```



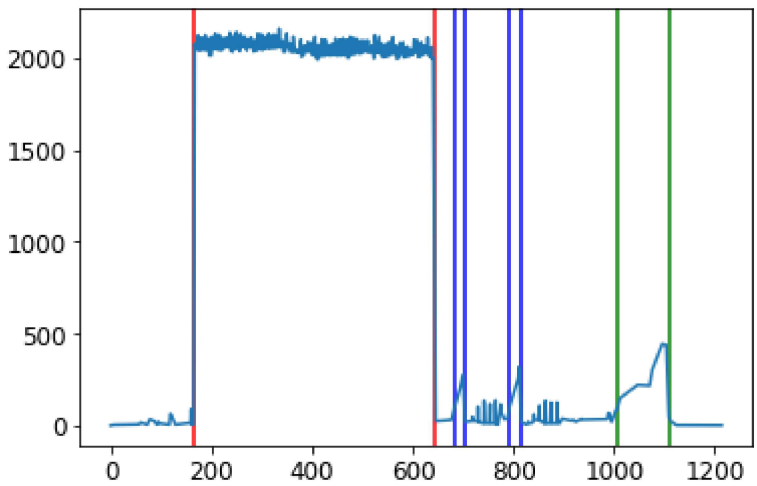

```
machine# 143250
End  Energy: 39.0557666667      End Start: 1071         End end: 1176
length:  1282
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 629.2579  Heat Start: 205         Heat end: 608
No. of Spins: 2         Spin Start: [692, 845]  Spin end: [718, 873]    Avg.
Spin Energy 5.865261111111112
avg energy for cycle:  502.15613884555376
```



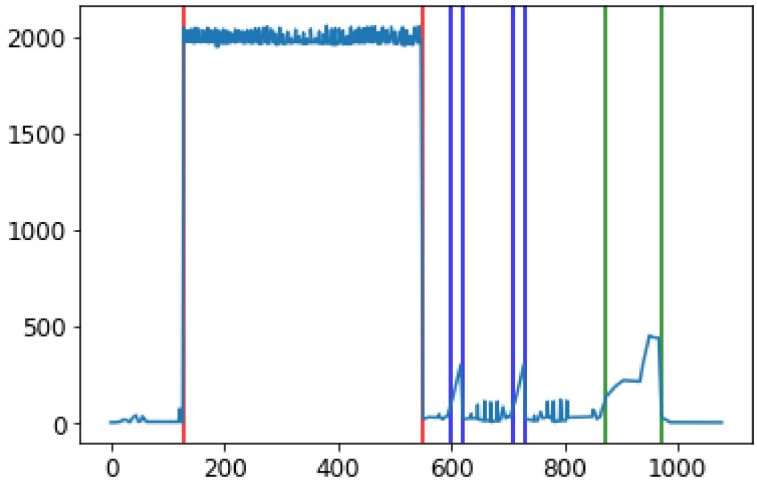
```
machine# 143251
End  Energy: 26.5148222222      End Start: 1063         End end: 1138
length:  1243
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 669.360655556     Heat Start: 272         Heat end: 581
No. of Spins: 3         Spin Start: [628, 756, 907]     Spin end: [654, 791,
935]    Avg.  Spin Energy 6.076959259259259
avg energy for cycle:  544.5513837489943
```




```
machine# 143252
End  Energy: 29.8143555556      End Start: 1007         End end: 1109
length:  1215
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 1121.96748889     Heat Start: 166         Heat end: 645
No. of Spins: 2         Spin Start: [684, 792]  Spin end: [704, 815]    Avg.
Spin Energy 5.00565
avg energy for cycle:  884.4452592592593
```

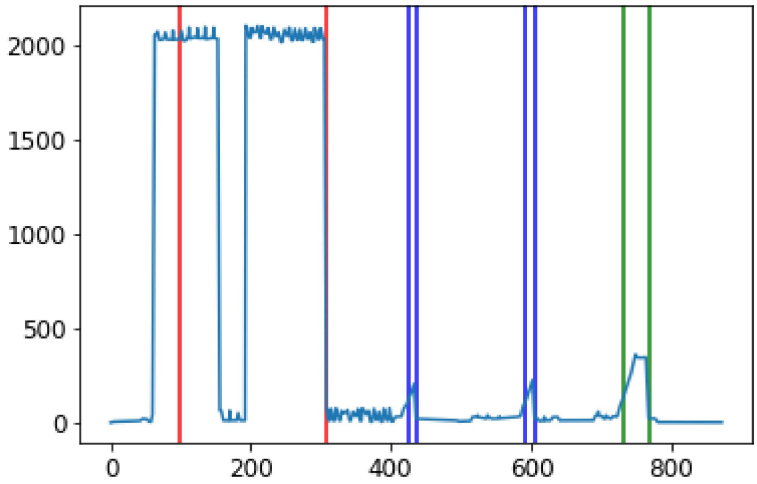


```
machine# 143253
End  Energy: 30.7052    End Start: 871  End end: 971
length:  1078
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 958.501177778     Heat Start: 128         Heat end: 551
No. of Spins: 2         Spin Start: [600, 710]  Spin end: [621, 732]    Avg.
Spin Energy 5.003905555555555
avg energy for cycle:  855.6886734693878
```

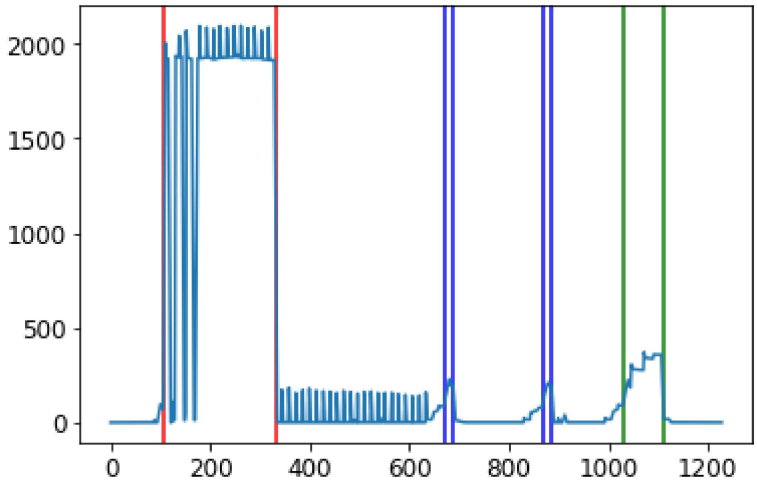



```
machine# 143254
```

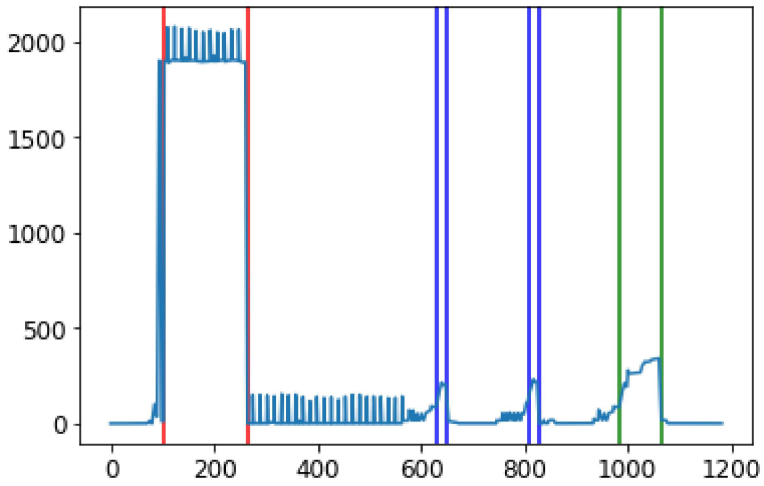
```
machine# 143258
End  Energy: 12.2339333333      End Start: 733  End end: 768
length:  873
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 407.127177778     Heat Start: 99  Heat end: 309
No. of Spins: 2         Spin Start: [425, 591]  Spin end: [436, 606]    Avg.
Spin Energy 2.4811944444444443
avg energy for cycle:  559.5319702176404
```



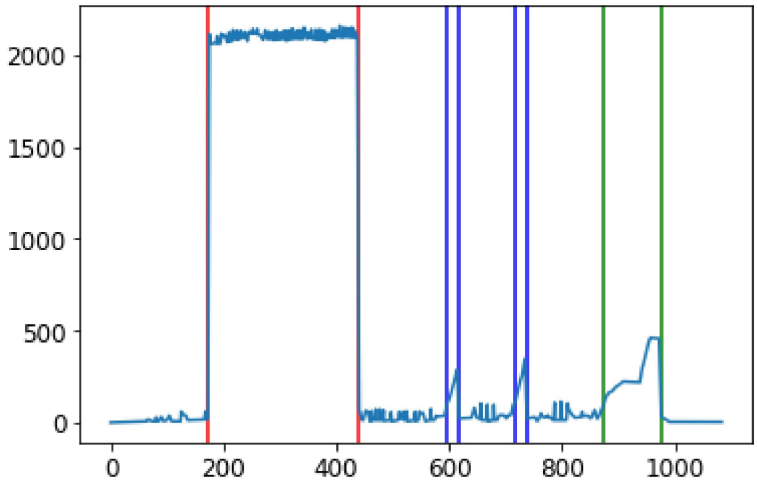
```
machine# 143259
End  Energy: 25.7709222222      End Start: 1030         End end: 1110
length:  1228
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 483.694588889     Heat Start: 107         Heat end: 333
No. of Spins: 2         Spin Start: [670, 867]  Spin end: [688, 885]    Avg.
Spin Energy 3.9102888888888887
avg energy for cycle:  439.4138110749185
```



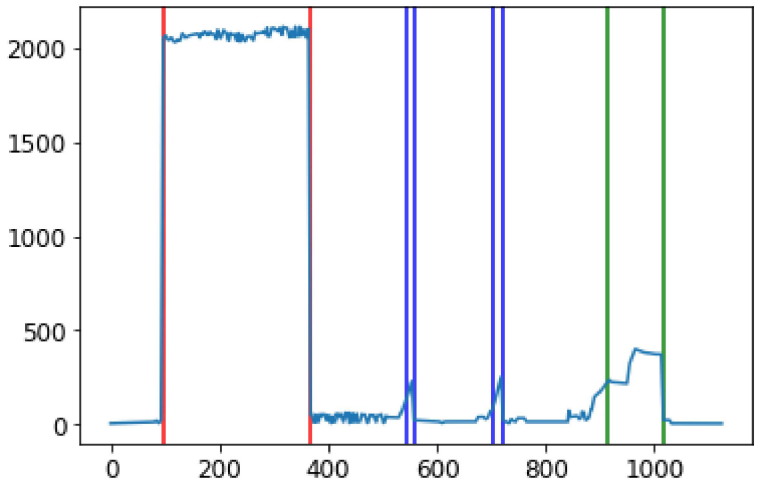

```
machine# 143260
End  Energy: 25.3465    End Start: 985  End end: 1065
length:  1183
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 362.503911111     Heat Start: 104         Heat end: 265
No. of Spins: 2         Spin Start: [631, 808]  Spin end: [652, 828]    Avg.
Spin Energy 4.427333333333333
avg energy for cycle:  372.97077768385464
```



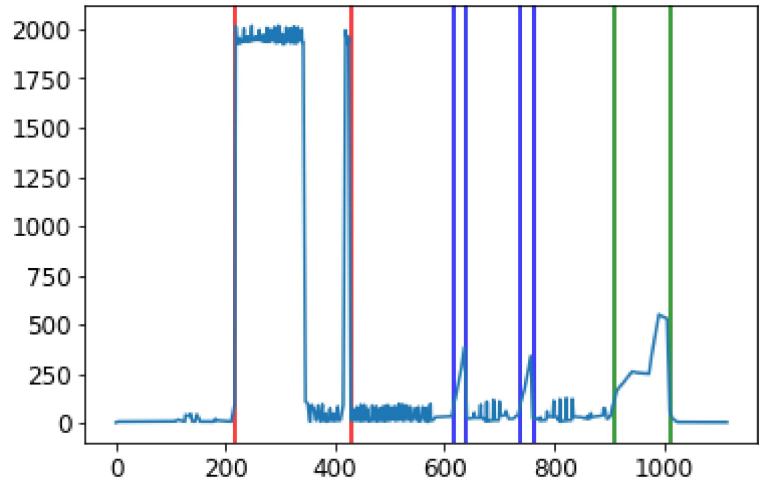
```
machine# 143262
End  Energy: 31.6368444444      End Start: 872  End end: 977
length:  1084
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 635.644288889     Heat Start: 174         Heat end: 441
No. of Spins: 2         Spin Start: [597, 718]  Spin end: [618, 738]    Avg.
Spin Energy 5.616649999999998
avg energy for cycle:  589.8456826568265
```



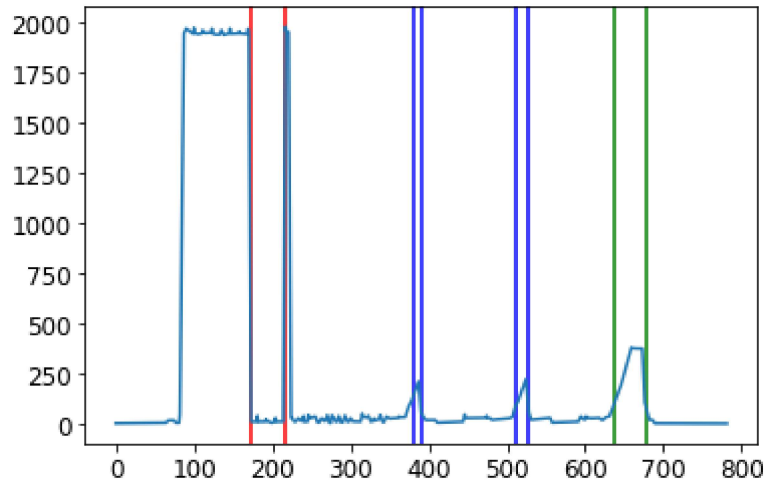

```
machine# 143265
End  Energy: 37.4841888889      End Start: 913  End end: 1017
length:  1125
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 627.341544444     Heat Start: 99  Heat end: 368
No. of Spins: 2         Spin Start: [546, 702]  Spin end: [558, 722]    Avg.
Spin Energy 2.235622222222222
avg energy for cycle:  575.9193333333334
```



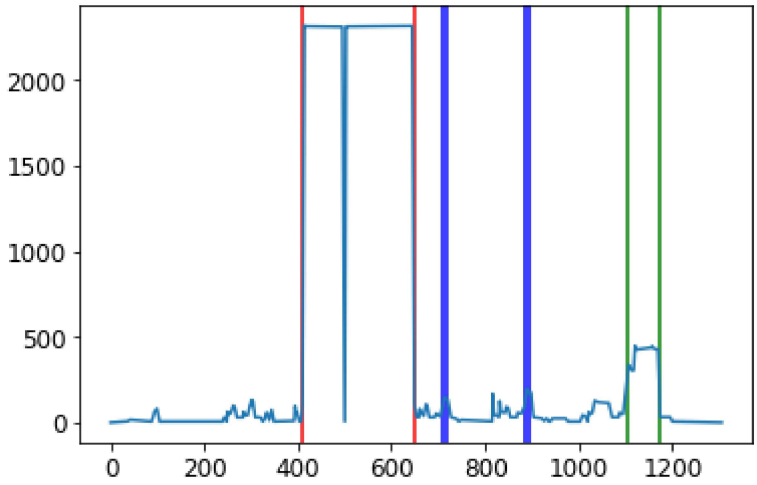
```
machine# 143270
End  Energy: 36.7940666667      End Start: 909  End end: 1013
length:  1116
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 317.224288889     Heat Start: 218         Heat end: 429
No. of Spins: 2         Spin Start: [616, 737]  Spin end: [640, 762]    Avg.
Spin Energy 5.963505555555556
avg energy for cycle:  333.0838709677419
```



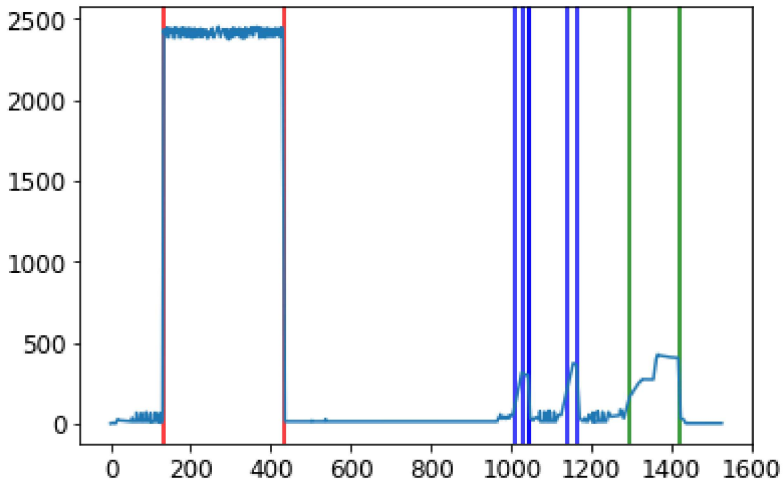

```
machine# 143271
End  Energy: 13.3649333333      End Start: 638  End end: 677
length:  782
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 188.373555556     Heat Start: 216         Heat end: 174
No. of Spins: 2         Spin Start: [380, 511]  Spin end: [391, 528]    Avg.
Spin Energy 2.202738888888889
avg energy for cycle:  305.4442455242966
```



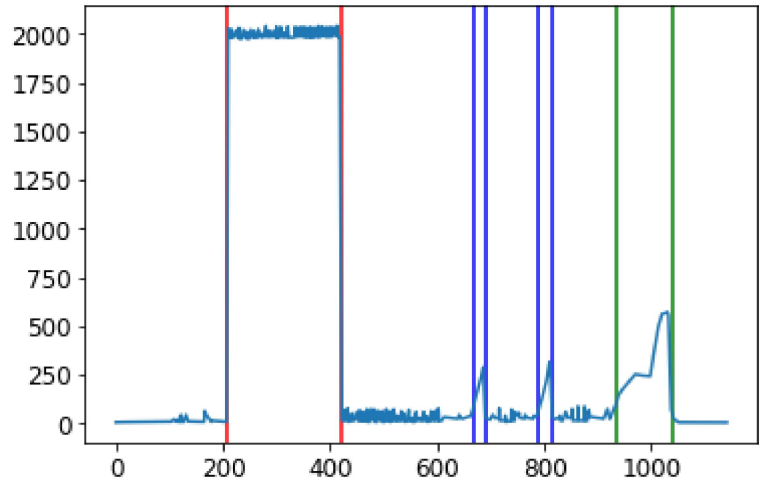
```
machine# 143273
End  Energy: 32.1100222222      End Start: 1102         End end: 1173
length:  1306
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 628.903355556     Heat Start: 411         Heat end: 648
No. of Spins: 2         Spin Start: [709, 886]  Spin end: [718, 893]    Avg.
Spin Energy 1.0572222222222223
avg energy for cycle:  504.72368300153147
```



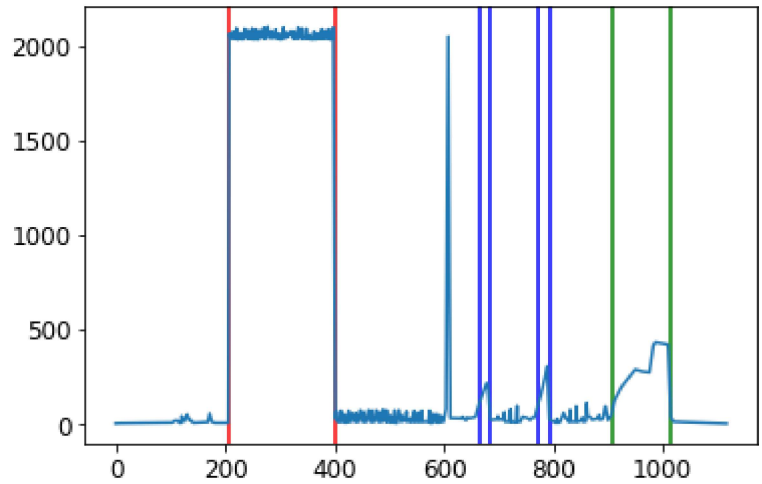

```
machine# 143276
End  Energy: 46.2689555556      End Start: 1295         End end: 1420
length:  1525
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 823.266555556     Heat Start: 131         Heat end: 433
No. of Spins: 3         Spin Start: [1008, 1042, 1137]  Spin end: [1028, 104
5, 1166]        Avg.  Spin Energy 4.5460370370370375
avg energy for cycle:  560.2703081967213
```



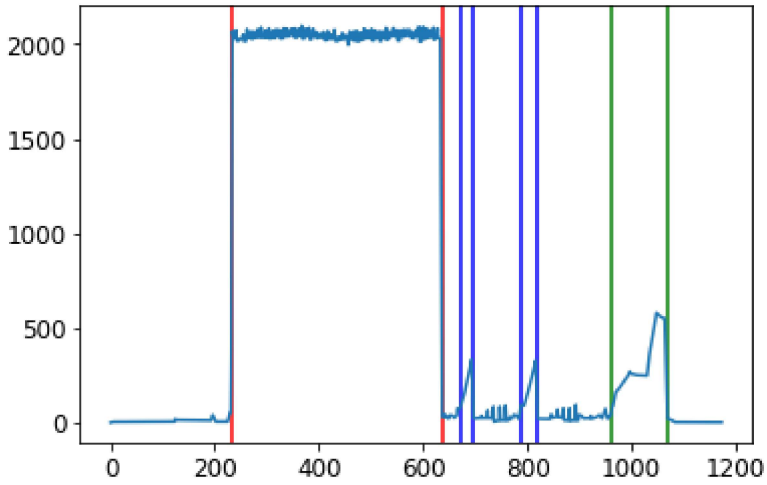
```
machine# 143284
End  Energy: 36.8446444444      End Start: 935  End end: 1039
length:  1142
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 482.731744444     Heat Start: 209         Heat end: 422
No. of Spins: 2         Spin Start: [669, 790]  Spin end: [690, 814]    Avg.
Spin Energy 4.925127777777778
avg energy for cycle:  446.17916812609457
```



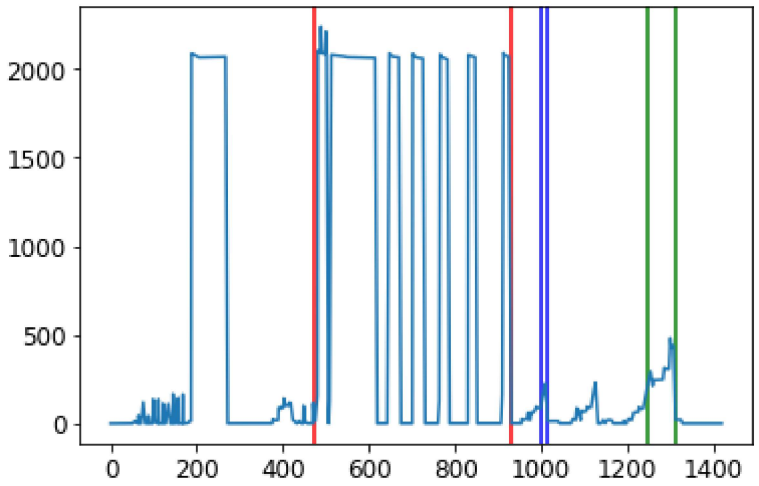

```
machine# 143285
End  Energy: 35.6499111111      End Start: 909  End end: 1016
length:  1118
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 454.096   Heat Start: 207         Heat end: 402
No. of Spins: 2         Spin Start: [665, 772]  Spin end: [684, 793]    Avg.
Spin Energy 4.331894444444445
avg energy for cycle:  446.42118962432914
```



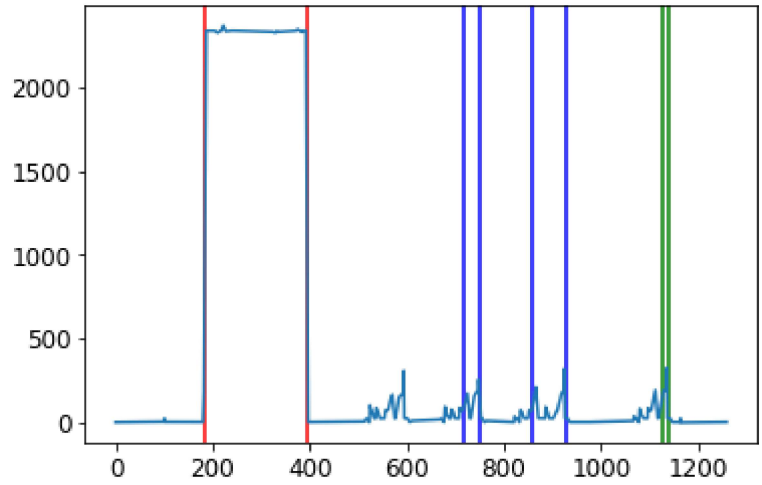
```
machine# 143286
End  Energy: 38.4462888889      End Start: 962  End end: 1069
length:  1174
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 934.601166667     Heat Start: 233         Heat end: 637
No. of Spins: 2         Spin Start: [674, 789]  Spin end: [697, 820]    Avg.
Spin Energy 6.251083333333334
avg energy for cycle:  773.9974701873937
```




```
machine# 143290
End  Energy: 18.9058555556      End Start: 1246         End end: 1310
length:  1419
Pre-Phase: True         prePhase Start: 188     prePhase end: 271
Heat  Energy: 626.323788889     Heat Start: 472         Heat end: 928
No. of Spins: 1         Spin Start: [999]       Spin end: [1013]        Avg.
Spin Energy 3.1554777777777776
avg energy for cycle:  597.6602818886539
```



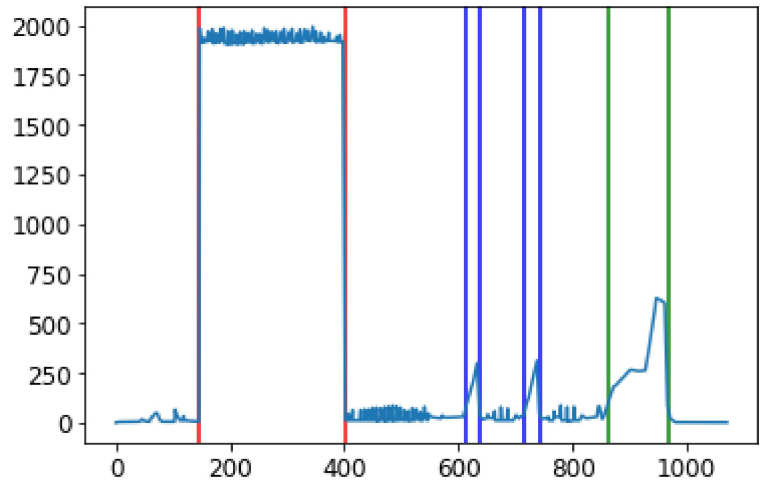
```
machine# 143294
End  Energy: 2.56174444444      End Start: 1127         End end: 1140
length:  1260
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 586.143744444     Heat Start: 183         Heat end: 396
No. of Spins: 2         Spin Start: [719, 860]  Spin end: [752, 928]    Avg.
Spin Energy 2.402105555555556
avg energy for cycle:  501.8638253968254
```



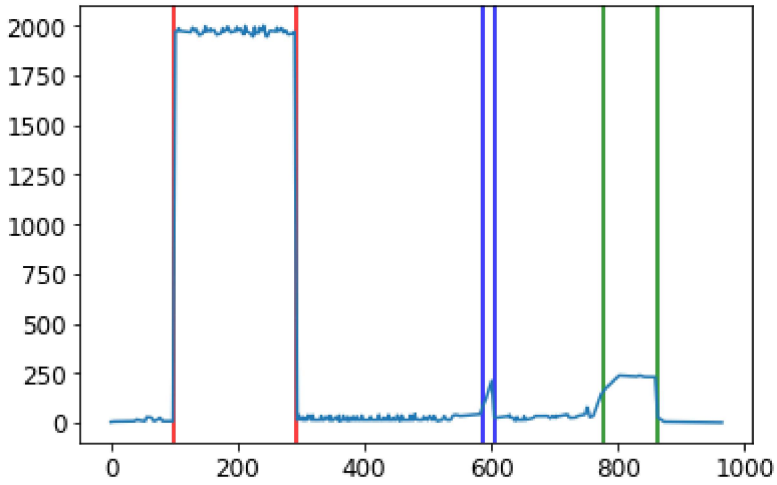

```
machine# 143299
```

```
machine# 143301
```

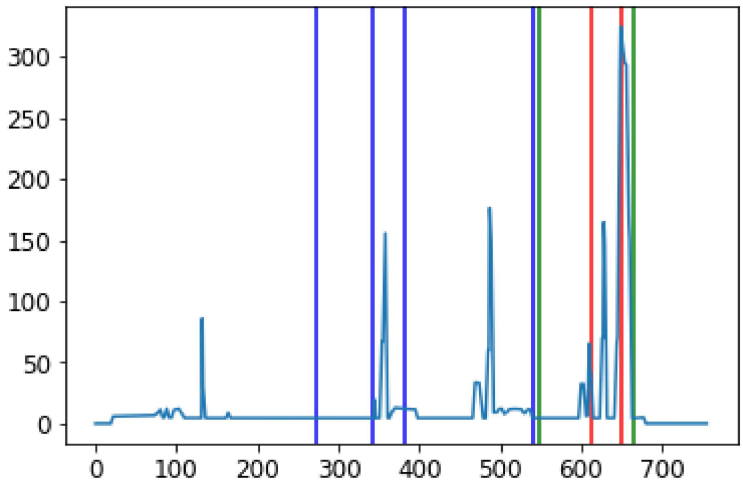
```
machine# 143302
End  Energy: 38.7699444444      End Start: 862  End end: 970
length:  1072
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 560.454822222     Heat Start: 147         Heat end: 402
No. of Spins: 2         Spin Start: [614, 717]  Spin end: [638, 743]    Avg.
Spin Energy 5.1884
avg energy for cycle:  541.5200746268657
```



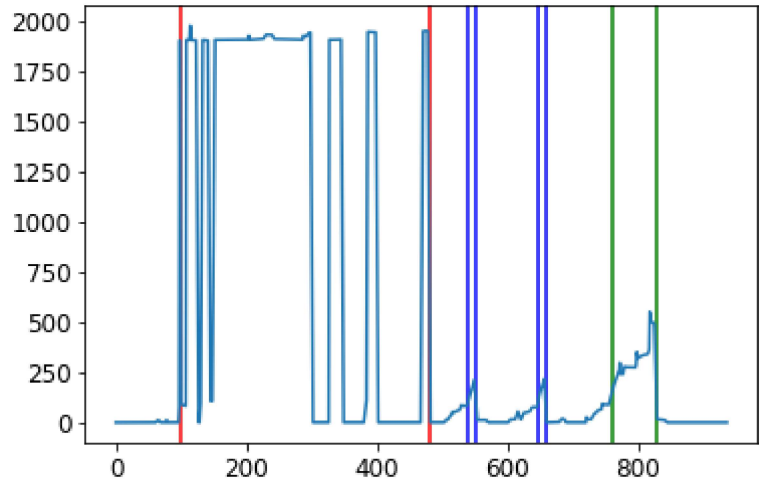
```
machine# 143306
End  Energy: 19.9987555556      End Start: 777  End end: 863
length:  965
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 433.154555556     Heat Start: 99  Heat end: 294
No. of Spins: 1         Spin Start: [587]       Spin end: [605]         Avg.
Spin Energy 2.1386
avg energy for cycle:  447.9397409326425
```



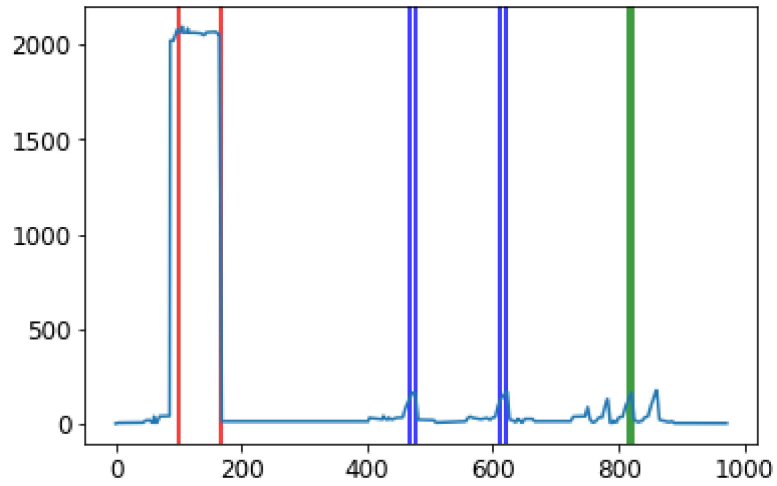

```
machine# 143311
End  Energy: 1.99531111111      End Start: 549  End end: 665
length:  756
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 4.42066666667     Heat Start: 613         Heat end: 650
No. of Spins: 2         Spin Start: [272, 383]  Spin end: [343, 541]    Avg.
Spin Energy 2.5295500000000004
avg energy for cycle:  77.05010582010583
```



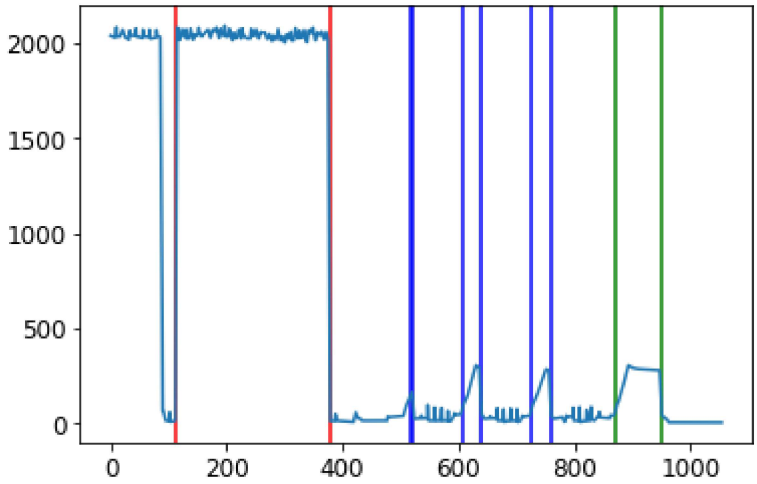
```
machine# 143312
End  Energy: 23.9149888889      End Start: 761  End end: 829
length:  937
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 547.405188889     Heat Start: 99  Heat end: 482
No. of Spins: 2         Spin Start: [539, 646]  Spin end: [553, 659]    Avg.
Spin Energy 2.577061111111111
avg energy for cycle:  592.1617716115261
```



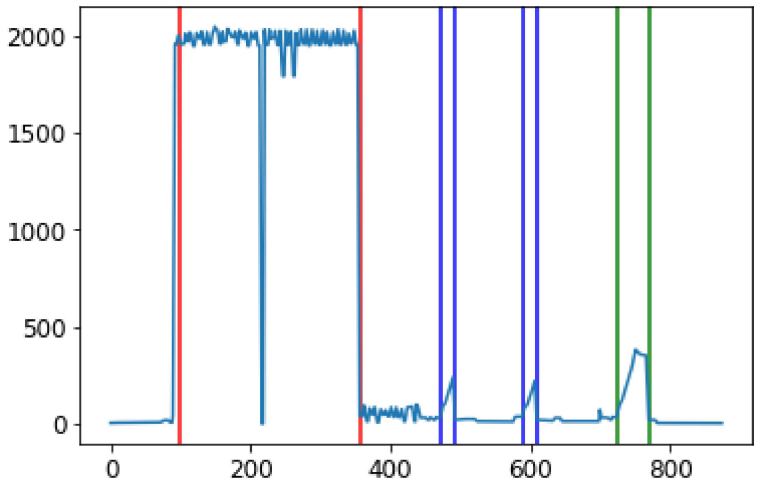

```
machine# 143315
End  Energy: 0.902033333333     End Start: 817  End end: 821
length:  973
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 164.328811111     Heat Start: 99  Heat end: 169
No. of Spins: 2         Spin Start: [468, 610]  Spin end: [477, 620]    Avg.
Spin Energy 1.444377777777778
avg energy for cycle:  219.67606372045222
```



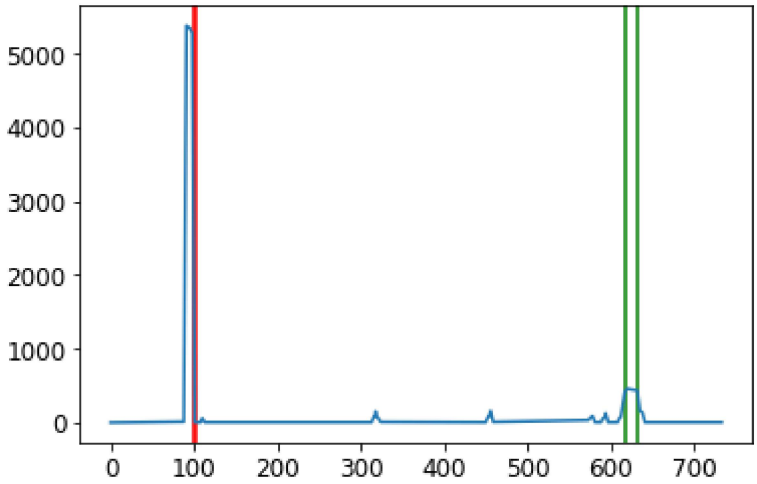
```
machine# 143316
End  Energy: 22.8763444444      End Start: 871  End end: 951
length:  1055
INVALID CYCLE - Previous continuation
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 615.698933333     Heat Start: 113         Heat end: 379
No. of Spins: 3         Spin Start: [516, 609, 725]     Spin end: [520, 640,
760]    Avg.  Spin Energy 4.59182962962963
avg energy for cycle:  763.5202274881517
```




```
machine# 143320
End  Energy: 13.4081111111      End Start: 725  End end: 770
length:  874
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 579.758288889     Heat Start: 99  Heat end: 357
No. of Spins: 2         Spin Start: [473, 590]  Spin end: [493, 610]    Avg.
Spin Energy 3.5288166666666663
avg energy for cycle:  665.7684210526317
```



```
machine# 143323


machine# 143324
End  Energy: 5.30802222222      End Start: 618  End end: 632
length:  734
Pre-Phase: False        prePhase Start: 0       prePhase end: 0
Heat  Energy: 23.8769888889     Heat Start: 99  Heat end: 102
avg energy for cycle:  191.35309264305175
```



In [35]:
```
jsonDF.head()
```

Out[35]:

	mac	pre	heat	spin	end	avgCycleEnergy
0	143200	{u'preFlag': False, u'pre_start': 0, u'pre_end...	{u'heat_end': 472, u'heatEnergy': 395.59136666...	{1: (613, 624), 2: (725, 735), u'avgEnergy': 1...	{u'end_start': 894, u'end_end': 897, u'endEner...	382.882081
1	143203	{u'preFlag': False, u'pre_start': 0, u'pre_end...	{u'heat_end': 420, u'heatEnergy': 811.08354444...	{1: (1167, 1179), 2: (1305, 1326), u'avgEnergy...	{u'end_start': 1575, u'end_end': 1642, u'endEn...	559.999443
2	143211	{u'preFlag': False, u'pre_start': 0, u'pre_end...	{u'heat_end': 296, u'heatEnergy': 388.28884444...	{1: (620, 643), 2: (781, 806), u'avgEnergy': 4...	{u'end_start': 935, u'end_end': 1039, u'endEne...	398.379764
3	143213	{u'preFlag': False, u'pre_start': 0, u'pre_end...	{u'heat_end': 301, u'heatEnergy': 367.48791111...	{1: (1480, 1484), 2: (1499, 1502), 3: (1657, 1...	{u'end_start': 1882, u'end_end': 1944, u'endEn...	208.864727
4	143216	{u'preFlag': False, u'pre_start': 0, u'pre_end...	{u'heat_end': 264, u'heatEnergy': 285.91966666...	{0: (0, 0), u'avgEnergy': 0, u'spins': 0}	{u'end_start': 601, u'end_end': 611, u'endEner...	384.824307

In [36]:
```
_file = 'C:\\Users\\Labyrinth\\JUPYTER NOTEBOOKS\\WeWash_Praktikum_TUM3sem\\WeWash_Analysis_ver2\\Models\\FinalAnalysis_50samples.JSON'
jsonDF.to_json(_file, orient='records')
```