(a)Accesing the Dataset

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import operator
import statistics
import seaborn as sns
```

In [2]:

```
bending1_te1 = pd.read_csv('/Users/hp1/Desktop/AREM/bending1/dataset1.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending1_te2 = pd.read_csv('/Users/hp1/Desktop/AREM/bending1/dataset2.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending1_tr1 = pd.read_csv('/Users/hp1/Desktop/AREM/bending1/dataset3.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending1_tr2 = pd.read_csv('/Users/hp1/Desktop/AREM/bending1/dataset4.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending1_tr3 = pd.read_csv('/Users/hp1/Desktop/AREM/bending1/dataset5.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending1_tr4 = pd.read_csv('/Users/hp1/Desktop/AREM/bending1/dataset6.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending1_tr5 = pd.read_csv('/Users/hp1/Desktop/AREM/bending1/dataset7.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])

bending2_te1 = pd.read_csv('/Users/hp1/Desktop/AREM/bending2/dataset1.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending2_te2 = pd.read_csv('/Users/hp1/Desktop/AREM/bending2/dataset2.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending2_tr1 = pd.read_csv('/Users/hp1/Desktop/AREM/bending2/dataset3.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending2_tr2 = pd.read_csv('/Users/hp1/Desktop/AREM/bending2/dataset4.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending2_tr3 = pd.read_csv('/Users/hp1/Desktop/AREM/bending2/dataset5.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
bending2_tr4 = pd.read_csv('/Users/hp1/Desktop/AREM/bending2/dataset6.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])

cycling_te1 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset1.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_te2 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset2.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_te3 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset3.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_tr1 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset4.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_tr2 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset5.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_tr3 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset6.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_tr4 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset7.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_tr5 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset8.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_tr6 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset9.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
cycling_tr7 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset10.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
cycling_tr8 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset11.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
cycling_tr9 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset12.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
cycling_tr10 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset13.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
cycling_tr11 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset14.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
cycling_tr12 = pd.read_csv('/Users/hp1/Desktop/AREM/cycling/dataset15.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
```

```
lying_te1 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset1.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_te2 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset2.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_te3 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset3.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr1 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset4.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr2 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset5.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr3 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset6.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr4 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset7.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr5 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset8.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr6 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset9.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr7 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset10.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr8 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset11.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr9 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset12.csv',header=4,usecols=[0, 1, 2, 3,
4, 5, 6])
lying_tr10 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset13.csv',header=4,usecols=[0, 1, 2, 3
, 4, 5, 6])
lying_tr11 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset14.csv',header=4,usecols=[0, 1, 2, 3
, 4, 5, 6])
lying_tr12 = pd.read_csv('/Users/hp1/Desktop/AREM/lying/dataset15.csv',header=4,usecols=[0, 1, 2, 3
, 4, 5, 6])

sitting_te1 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset1.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_te2 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset2.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_te3 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset3.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_tr1 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset4.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_tr2 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset5.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_tr3 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset6.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_tr4 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset7.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_tr5 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset8.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_tr6 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset9.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
sitting_tr7 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset10.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
sitting_tr8 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset11.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
sitting_tr9 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset12.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
sitting_tr10 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset13.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
sitting_tr11 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset14.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
sitting_tr12 = pd.read_csv('/Users/hp1/Desktop/AREM/sitting/dataset15.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])

standing_te1 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset1.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_te2 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset2.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_te3 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset3.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr1 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset4.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr2 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset5.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr3 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset6.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr4 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset7.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
```

```
standing_tr5 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset8.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr6 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset9.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr7 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset10.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr8 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset11.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr9 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset12.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
standing_tr10 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset13.csv',header=4,usecols=[0, 1
, 2, 3, 4, 5, 6])
standing_tr11 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset14.csv',header=4,usecols=[0, 1
, 2, 3, 4, 5, 6])
standing_tr12 = pd.read_csv('/Users/hp1/Desktop/AREM/standing/dataset15.csv',header=4,usecols=[0, 1
, 2, 3, 4, 5, 6])

walking_te1 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset1.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_te2 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset2.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_te3 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset3.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_tr1 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset4.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_tr2 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset5.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_tr3 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset6.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_tr4 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset7.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_tr5 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset8.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_tr6 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset9.csv',header=4,usecols=[0, 1, 2,
3, 4, 5, 6])
walking_tr7 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset10.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
walking_tr8 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset11.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
walking_tr9 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset12.csv',header=4,usecols=[0, 1, 2
, 3, 4, 5, 6])
walking_tr10 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset13.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
walking_tr11 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset14.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
walking_tr12 = pd.read_csv('/Users/hp1/Desktop/AREM/walking/dataset15.csv',header=4,usecols=[0, 1,
2, 3, 4, 5, 6])
```

(b)Splitting the data into test and train

In [3]:

```
test = pd.concat([bending1_te1,bending1_te2,bending2_te1,bending2_te2,
                  cycling_te1,cycling_te2,cycling_te3,lying_te1,lying_te2,lying_te3,
                  sitting_te1,sitting_te2,sitting_te3,standing_te1,standing_te2,standing_te3,
                  walking_te1,walking_te2,walking_te3],ignore_index=True,sort=False)
train = pd.concat([bending1_tr1,bending1_tr2,bending1_tr3,bending1_tr4,bending1_tr5,
                   bending2_tr1,bending2_tr2,bending2_tr3,bending2_tr4,
                   cycling_tr1,cycling_tr2,cycling_tr3,cycling_tr4,cycling_tr5,cycling_tr6,
                   cycling_tr7,cycling_tr8,cycling_tr9,cycling_tr10,cycling_tr11,cycling_tr12,
                   lying_tr1,lying_tr2,lying_tr3,lying_tr4,lying_tr5,lying_tr6,
                   lying_tr7,lying_tr8,lying_tr9,lying_tr10,lying_tr11,lying_tr12,
                   sitting_tr1,sitting_tr2,sitting_tr3,sitting_tr4,sitting_tr5,sitting_tr6,
                   sitting_tr7,sitting_tr8,sitting_tr9,sitting_tr10,sitting_tr11,sitting_tr12,
                   standing_tr1,standing_tr2,standing_tr3,standing_tr4,standing_tr5,standing_tr6,
                   standing_tr7,standing_tr8,standing_tr9,standing_tr10,standing_tr11,standing_tr12
,
                   walking_tr1,walking_tr2,walking_tr3,walking_tr4,walking_tr5,walking_tr6,
                   walking_tr7,walking_tr8,walking_tr9,walking_tr10,walking_tr11,walking_tr12],ignc
re_index=True,sort=False)
Data_test_train = pd.concat([test, train],ignore_index=True,sort=False)
print(Data_test_train)
```

```
       # Columns: time  avg_rss12  var_rss12  avg_rss13  var_rss13  avg_rss23  \
```

```
       # Columns: time  avg_rss12  var_rss12  avg_rss13  var_rss13  avg_rss23  \
0                   0      39.25       0.43      22.75       0.43      33.75
1                 250      39.25       0.43      23.00       0.00      33.00
2                 500      39.25       0.43      23.25       0.43      33.00
3                 750      39.50       0.50      23.00       0.71      33.00
4                1000      39.50       0.50      24.00       0.00      33.00
5                1250      39.25       0.43      24.00       0.00      33.00
6                1500      39.25       0.43      24.00       0.00      33.00
7                1750      39.00       0.00      23.75       0.43      33.00
8                2000      39.50       0.50      24.00       0.00      33.00
9                2250      39.50       0.50      23.00       0.00      33.00
10               2500      39.50       0.50      23.25       0.43      33.00
11               2750      39.50       0.50      23.50       0.50      32.75
12               3000      39.50       0.50      23.75       0.43      32.50
13               3250      39.67       0.47      23.75       0.43      33.00
14               3500      39.50       0.50      24.00       0.00      33.00
15               3750      39.50       0.50      23.25       0.43      33.00
16               4000      39.50       0.50      22.50       0.50      33.00
17               4250      39.50       0.50      22.00       0.71      33.00
18               4500      40.25       0.83      21.00       0.00      33.00
19               4750      40.50       0.50      18.67       1.70      33.00
20               5000      40.67       0.47      15.50       0.87      33.00
21               5250      40.50       0.50      15.00       0.00      33.00
22               5500      40.50       0.50      15.00       0.00      33.00
23               5750      40.33       0.47      16.00       0.00      33.00
24               6000      40.50       0.50      16.00       0.71      33.00
25               6250      40.50       0.50      16.50       0.87      33.00
26               6500      40.50       0.50      17.50       0.87      33.00
27               6750      40.50       0.50      19.75       1.09      33.75
28               7000      40.75       0.83      21.00       0.00      33.00
29               7250      40.75       0.83      21.00       0.00      33.00
...               ...        ...        ...        ...        ...        ...
42209          112500      39.75       1.30      16.00       3.46      19.25
42210          112750      39.25       2.38      17.00       2.74      16.50
42211          113000      34.50       9.60       9.25       6.30      17.75
42212          113250      25.75       2.95      12.50       1.50      16.75
42213          113500      27.25       6.38      16.25       3.63      15.50
42214          113750      36.75       4.44      16.33       5.79      15.33
42215          114000      32.67       0.47      19.50       4.56      19.00
42216          114250      32.25       5.76      19.00       4.53      16.00
42217          114500      42.33       4.50      19.00       1.41      15.00
42218          114750      29.40       4.96      18.40       1.85      18.50
42219          115000      41.25       7.19      14.50       3.35      20.50
42220          115250      40.25       1.30      16.00       1.22      21.25
42221          115500      38.75       2.38      16.00       4.42      13.75
42222          115750      37.50       9.07      10.67       4.78      16.25
42223          116000      33.25       1.79      14.75       7.01      16.33
42224          116250      30.33       2.49      17.00       2.16      19.25
42225          116500      37.75       5.07      13.00       7.65      12.25
42226          116750      35.50       4.15      15.50       3.28      14.00
42227          117000      36.67       4.78      16.00       2.35      17.33
42228          117250      35.00      10.05      16.00       3.16      12.50
42229          117500      26.00       5.79      16.25       5.97      11.00
42230          117750      28.75       7.89      11.25       3.77      15.75
42231          118000      37.50       3.20      15.75       5.54      17.50
42232          118250      31.50       2.60      13.25       5.93      19.00
42233          118500      36.25       5.63      17.00       2.16      18.50
42234          118750      34.50       6.18       9.00       3.56      12.67
42235          119000      25.75       6.02      13.75       2.05      16.00
42236          119250      31.50       3.35      10.25       5.12      16.25
42237          119500      33.75       2.77      14.00       3.24      13.75
42238          119750      37.00       1.41      18.25       3.70      11.00

       var_rss23
0           1.30
1           0.00
2           0.00
3           0.00
4           0.00
5           0.00
6           0.00
7           0.00
8           0.00
9           0.00
10          0.00
11          0.43
12          0.50
13          0.00
```

```
14         0.00
15         0.00
16         0.00
17         0.00
18         0.00
19         0.00
20         0.00
21         0.00
22         0.00
23         0.00
24         0.00
25         0.00
26         0.00
27         1.30
28         0.00
29         0.00
...         ...
42209      2.49
42210      1.50
42211      4.44
42212      3.70
42213      3.57
42214      1.25
42215      3.56
42216      4.85
42217      8.04
42218      3.20
42219      0.87
42220      0.43
42221      7.66
42222      2.95
42223      3.09
42224      4.02
42225      2.49
42226      5.34
42227      3.30
42228      1.80
42229      5.34
42230      2.59
42231      3.91
42232      2.74
42233      0.87
42234      4.19
42235      1.58
42236      2.95
42237      0.43
42238      4.32

[42239 rows x 7 columns]
```

(C)Feature Extraction

(i) and (ii)

In [35]:

```
#TABLE after extracting time domain features
Name = ['Instance', 'Min1', 'Max1', 'Mean1', 'Median1', 'std1', '1st quart_1', '3rd quart_1',
         'Min2', 'Max2', 'Mean2', 'Median2', 'std2', '1st quart_2', '3rd quart_2',
         'Min3', 'Max3', 'Mean3', 'Median3', 'std3', '1st quart_3', '3rd quart_3',
            'Min4', 'Max4', 'Mean4', 'Median4', 'std4', '1st quart_4', '3rd quart_4',
      'Min5', 'Max5', 'Mean5', 'Median5', 'std5', '1st quart_5', '3rd quart_5',
         'Min6', 'Max6', 'Mean6', 'Median6', 'std6', '1st quart_6', '3rd quart_6'
        ]
Inst_Table = pd.DataFrame(np.zeros((88, 43)), columns=Name)
Table = pd.DataFrame(np.zeros((88, 43)), columns=Name)
val = -1
for i in range(0,88):
    Table.iloc[i, 0] = int(i+1)
def time_instance(activity):
    global val
    val+=1
    list_dir1 =
```

```
    list_dir1 =
[bending1_te1,bending1_te2,bending1_tr1,bending1_tr2,bending1_tr3,bending1_tr4,bending1_tr5,
               bending2_te1,bending2_te2,bending2_tr1,bending2_tr2,bending2_tr3,bending2_tr4,
               cycling_te1,cycling_te2,cycling_te3,
               cycling_tr1,cycling_tr2,cycling_tr3,cycling_tr4,cycling_tr5,cycling_tr6,
               cycling_tr7,cycling_tr8,cycling_tr9,cycling_tr10,cycling_tr11,cycling_tr12,
               lying_te1,lying_te2,lying_te3,
               lying_tr1,lying_tr2,lying_tr3,lying_tr4,lying_tr5,lying_tr6,
               lying_tr7,lying_tr8,lying_tr9,lying_tr10,lying_tr11,lying_tr12,
               sitting_te1,sitting_te2,sitting_te3,
               sitting_tr1,sitting_tr2,sitting_tr3,sitting_tr4,sitting_tr5,sitting_tr6,
               sitting_tr7,sitting_tr8,sitting_tr9,sitting_tr10,sitting_tr11,sitting_tr12,
               standing_te1,standing_te2,standing_te3,
               standing_tr1,standing_tr2,standing_tr3,standing_tr4,standing_tr5,standing_tr6,
               standing_tr7,standing_tr8,standing_tr9,standing_tr10,standing_tr11,standing_tr12,
               walking_te1,walking_te2,walking_te3,
               walking_tr1,walking_tr2,walking_tr3,walking_tr4,walking_tr5,walking_tr6,
               walking_tr7,walking_tr8,walking_tr9,walking_tr10,walking_tr11,walking_tr12]
    j=0
    k=1
    for k in range(1, 7):
        Table.iloc[val, j+1] = list_dir1[activity].iloc[:, k].min()
        Table.iloc[val, j+2] = list_dir1[activity].iloc[:, k].max()
        Table.iloc[val, j+3] = list_dir1[activity].iloc[:, k].mean()
        Table.iloc[val, j+4] = list_dir1[activity].iloc[:, k].median()
        Table.iloc[val, j+5] = list_dir1[activity].iloc[:, k].std()
        Table.iloc[val, j+6] = list_dir1[activity].iloc[:, k].quantile(q=.25)
        Table.iloc[val, j+7] = list_dir1[activity].iloc[:, k].quantile(q=.75)
        j = j+7

for temp in range(0,88):
    time_instance(temp)



Inst_Table = Table
print(Inst_Table)
Inst_Table['class'] = 0
Inst_Table.iloc[:13, -1] = 1
```

```
    Instance   Min1   Max1       Mean1  Median1      std1  1st quart_1  \
0        1.0  37.25  45.00  40.624792   40.500  1.476967      39.2500
1        2.0  38.00  45.67  42.812812   42.500  1.435550      42.0000
2        3.0  35.00  47.40  43.954500   44.330  1.558835      43.0000
3        4.0  33.00  47.75  42.179813   43.500  3.670666      39.1500
4        5.0  33.00  45.75  41.678063   41.750  2.243490      41.3300
5        6.0  37.00  48.00  43.454958   43.250  1.386098      42.5000
6        7.0  36.25  48.00  43.969125   44.500  1.618364      43.3100
7        8.0  12.75  51.00  24.562958   24.250  3.737514      23.1875
8        9.0   0.00  42.75  27.464604   28.000  3.583582      25.5000
9       10.0  21.00  50.00  32.586208   33.000  6.238143      26.1875
10      11.0  27.50  33.00  29.881938   30.000  1.153837      29.0000
11      12.0  19.00  45.50  30.938104   29.000  7.684146      26.7500
12      13.0  25.00  47.50  31.058250   29.710  4.829794      27.5000
13      14.0  24.25  45.00  37.177042   36.250  3.581301      34.5000
14      15.0  28.75  44.75  37.561188   36.875  3.226507      35.2500
15      16.0  22.00  44.67  37.058708   36.000  3.710180      34.5000
16      17.0  19.00  44.00  36.228396   36.000  3.528617      34.0000
17      18.0  26.50  44.33  36.687292   36.000  3.529404      34.2500
18      19.0  25.33  45.00  37.114312   36.250  3.710385      34.5000
19      20.0  26.75  44.75  36.863375   36.330  3.555787      34.5000
20      21.0  26.25  44.25  36.957458   36.290  3.434863      34.5000
21      22.0  27.75  44.67  37.144833   36.330  3.758904      34.0000
22      23.0  27.00  45.00  36.819521   36.000  3.900459      33.7500
23      24.0  27.00  44.33  36.541667   36.000  4.018922      33.2500
24      25.0  18.50  44.25  35.752354   36.000  4.614802      33.0000
25      26.0  19.00  43.75  35.879875   36.000  4.614878      33.0000
26      27.0  23.33  43.50  36.244083   36.750  3.822016      33.4575
27      28.0  24.25  45.00  37.177042   36.250  3.581301      34.5000
28      29.0  23.50  30.00  27.716375   27.500  1.442253      27.0000
29      30.0  24.75  48.33  44.182937   48.000  7.495615      48.0000
..       ...    ...    ...        ...      ...       ...          ...
58      59.0  33.33  48.00  44.334729   45.000  2.476940      42.2500
59      60.0  35.50  46.25  43.174938   43.670  1.989052      42.5000
60      61.0  32.75  47.00  42.760563   44.500  3.398919      41.3300
61      62.0  30.00  46.67  42.648521   42.750  2.395338      41.5000
62      63.0  36.00  47.50  43.720021   45.000  2.384105      43.0000
```

63	64.0	34.50	47.75	44.471146	45.000	1.772553	45.0000
64	65.0	35.50	48.00	46.224937	46.000	1.748315	45.2500
65	66.0	29.75	48.00	46.932208	47.500	1.832665	47.2375
66	67.0	36.33	47.67	45.399625	45.500	1.328121	45.0000
67	68.0	36.00	45.80	42.419917	42.670	2.520129	41.3300
68	69.0	37.00	48.25	42.516958	42.500	2.195751	41.0000
69	70.0	36.25	45.50	42.959354	42.670	1.500878	42.0000
70	71.0	36.00	47.33	42.674583	43.670	2.384170	40.0000
71	72.0	36.25	45.75	43.187521	44.750	2.491162	39.7500
72	73.0	36.00	47.33	44.441187	45.000	2.417797	44.6275
73	74.0	19.33	43.50	34.227771	35.500	4.889576	30.5000
74	75.0	12.50	45.00	33.509729	34.125	4.850923	30.5000
75	76.0	15.00	46.75	34.660583	35.000	5.315110	31.0000
76	77.0	18.00	46.00	35.193333	36.000	4.751868	32.0000
77	78.0	20.75	46.25	34.763333	35.290	4.742208	31.6700
78	79.0	21.50	51.00	34.935813	35.500	4.645944	32.0000
79	80.0	18.33	47.67	34.333042	34.750	4.948770	31.2500
80	81.0	18.33	45.75	34.599875	35.125	4.731790	31.5000
81	82.0	15.50	43.67	34.225875	34.750	4.441798	31.2500
82	83.0	21.50	51.25	34.253521	35.000	4.940741	30.9375
83	84.0	19.50	45.33	33.586875	34.250	4.650935	30.2500
84	85.0	19.75	45.50	34.322750	35.250	4.752477	31.0000
85	86.0	19.50	46.00	34.546229	35.250	4.842294	31.2500
86	87.0	23.50	46.25	34.873229	35.250	4.531720	31.7500
87	88.0	19.25	44.00	34.473188	35.000	4.796705	31.2500

	3rd quart_1	Min2	Max2	...	std5	1st quart_5	3rd quart_5	\
0	42.0000	0.0	1.30	...	2.188449	33.0000	36.0000	
1	43.6700	0.0	1.22	...	1.995255	32.0000	34.5000	
2	45.0000	0.0	1.70	...	1.999604	35.3625	36.5000	
3	45.0000	0.0	3.00	...	3.849448	30.4575	36.3300	
4	42.7500	0.0	2.83	...	2.411026	28.4575	31.2500	
5	45.0000	0.0	1.58	...	2.488862	22.2500	24.0000	
6	44.6700	0.0	1.50	...	3.318301	20.5000	23.7500	
7	26.5000	0.0	6.87	...	3.693786	20.5000	27.0000	
8	30.0000	0.0	7.76	...	5.053642	15.0000	20.7500	
9	34.5000	0.0	9.90	...	5.032424	17.6700	23.5000	
10	30.2700	0.0	1.00	...	1.745970	17.0000	19.0000	
11	38.0000	0.0	6.40	...	5.845911	15.0000	20.8125	
12	31.8125	0.0	6.38	...	7.853427	9.0000	18.3125	
13	40.2500	0.0	8.58	...	2.890347	17.9500	21.7500	
14	40.2500	0.0	9.91	...	2.727377	18.0000	21.5000	
15	40.0625	0.0	14.17	...	3.537144	16.0000	21.0000	
16	39.0000	0.0	12.28	...	3.166655	14.0000	18.0625	
17	39.3725	0.0	12.89	...	2.978238	14.6700	18.5000	
18	40.2500	0.0	10.84	...	2.847876	14.7500	18.5000	
19	39.7500	0.0	11.68	...	2.655906	15.0000	18.6700	
20	40.2500	0.0	8.64	...	2.851673	14.0000	18.2500	
21	40.5000	0.0	10.76	...	2.689291	15.0000	18.7500	
22	40.2500	0.0	10.47	...	2.781030	15.5000	19.2700	
23	39.8125	0.0	10.43	...	3.088141	15.0000	19.5000	
24	39.3300	0.0	12.60	...	3.120057	14.0000	18.0625	
25	39.5000	0.0	11.20	...	3.537635	14.7500	19.6900	
26	39.2500	0.0	9.71	...	3.617702	15.7500	21.0000	
27	40.2500	0.0	8.58	...	2.890347	17.9500	21.7500	
28	29.0000	0.0	1.79	...	4.074511	5.5000	10.7500	
29	48.0000	0.0	3.11	...	3.274539	2.0000	5.5425	
..	...	...	...	...	...	...	...	
58	46.5000	0.0	3.90	...	5.401794	9.3300	17.7500	
59	44.5000	0.0	2.12	...	2.983976	12.7500	16.5000	
60	45.3725	0.0	3.34	...	4.296574	13.0000	18.5650	
61	45.0000	0.0	2.95	...	3.141679	10.6275	14.2500	
62	45.0000	0.0	1.92	...	3.289138	11.3100	15.5425	
63	45.2500	0.0	2.18	...	2.612390	12.0000	14.8125	
64	48.0000	0.0	4.50	...	2.931581	12.0000	15.2500	
65	47.7500	0.0	4.60	...	3.134822	11.6700	15.5000	
66	46.3300	0.0	1.66	...	3.374095	11.2500	14.5000	
67	44.6175	0.0	2.12	...	3.722074	7.6275	12.0000	
68	44.5000	0.0	2.12	...	3.623557	12.6275	17.5000	
69	44.3300	0.0	2.60	...	2.702605	14.0000	16.6900	
70	44.7500	0.0	2.17	...	3.261617	12.7500	16.5000	
71	45.0000	0.0	2.83	...	3.566038	16.5000	21.0000	
72	45.7500	0.0	4.50	...	3.414454	11.0000	14.6700	
73	37.7500	0.0	14.50	...	3.092094	14.7500	18.6700	
74	36.7500	0.0	13.05	...	3.133564	14.6275	18.7500	
75	38.2500	0.0	13.44	...	3.155015	14.2500	18.5000	
76	38.7500	0.0	16.20	...	3.207642	14.2500	18.5000	

```
77      38.2500   0.0  12.68     ...       3.174681      14.2500      18.3300
78      38.0625   0.0  12.21     ...       3.192058      14.2375      18.2500
79      38.0000   0.0  12.48     ...       3.000493      13.7500      18.0000
80      38.0000   0.0  15.37     ...       2.905688      14.0000      18.2500
81      37.2500   0.0  17.24     ...       2.992920      14.3300      18.2500
82      37.7500   0.0  13.55     ...       3.116627      13.7500      18.0000
83      37.0000   0.0  14.67     ...       3.283983      13.7300      18.2500
84      38.0000   0.0  13.47     ...       3.119856      13.5000      17.7500
85      37.8125   0.0  12.47     ...       2.823124      14.0000      17.7500
86      38.2500   0.0  14.82     ...       3.131076      13.7500      18.0000
87      38.0000   0.0  13.86     ...       3.156320      13.7300      17.7500

    Min6   Max6     Mean6  Median6      std6  1st quart_6  3rd quart_6
0   0.00   1.92  0.570583    0.430  0.582915       0.0000       1.3000
1   0.00   3.11  0.571083    0.430  0.601010       0.0000       1.3000
2   0.00   1.79  0.493292    0.430  0.513506       0.0000       0.9400
3   0.00   2.18  0.613521    0.500  0.524317       0.0000       1.0000
4   0.00   1.79  0.383292    0.430  0.389164       0.0000       0.5000
5   0.00   5.26  0.679646    0.500  0.622534       0.4300       0.8700
6   0.00   2.96  0.555313    0.490  0.487826       0.0000       0.8300
7   0.00   4.97  0.700188    0.500  0.693720       0.4300       0.8700
8   0.00   6.76  1.122125    0.830  1.012342       0.4700       1.3000
9   0.00  13.61  1.162042    0.830  1.332980       0.4700       1.3000
10  0.00   6.40  0.701625    0.710  0.481103       0.4700       0.9400
11  0.00   6.73  1.107354    0.830  1.080842       0.4700       1.3000
12  0.00   4.92  1.098104    0.940  0.831480       0.5000       1.3000
13  0.00   9.34  2.921729    2.500  1.852600       1.5000       3.9000
14  0.00   9.62  2.765896    2.450  1.769203       1.4100       3.7700
15  0.00   8.55  2.983750    2.570  1.815730       1.5000       4.1500
16  0.00   9.98  3.480687    3.340  1.827769       2.1025       4.5500
17  0.00   8.19  3.073312    2.690  1.629675       1.9125       4.0875
18  0.00   9.50  3.076354    2.770  1.824534       1.7000       4.0375
19  0.00   8.81  2.773312    2.590  1.569919       1.6400       3.6325
20  0.00   8.34  2.934625    2.525  1.631380       1.6600       4.0300
21  0.00   8.75  2.822437    2.590  1.637183       1.5800       3.7400
22  0.00   8.99  2.887562    2.525  1.723094       1.5600       3.7700
23  0.00   9.18  3.225458    2.870  1.769758       1.8850       4.2625
24  0.00   9.39  3.069667    2.770  1.748326       1.7975       4.0600
25  0.00   8.50  3.093021    2.930  1.626034       1.8900       4.0600
26  0.00  11.15  3.530500    3.110  1.963685       2.1700       4.6175
27  0.00   9.34  2.921729    2.500  1.852600       1.5000       3.9000
28  0.00   4.50  0.734271    0.710  0.613688       0.4300       1.0000
29  0.00   3.91  0.692771    0.500  0.675781       0.3225       0.9400
..   ...    ...       ...      ...       ...          ...          ...
58  0.00   5.02  0.933000    0.830  0.673609       0.4700       1.2500
59  0.00   5.72  0.911979    0.830  0.666161       0.4700       1.2200
60  0.00   5.73  0.842271    0.710  0.722165       0.4300       1.0900
61  0.00   4.64  0.917354    0.830  0.709638       0.4700       1.1200
62  0.00   6.18  1.039688    0.830  0.916657       0.4700       1.2200
63  0.00   4.32  0.927375    0.830  0.756436       0.4700       1.2200
64  0.00   6.00  0.882583    0.830  0.668423       0.4700       1.1200
65  0.00   6.58  0.991125    0.830  0.855329       0.4700       1.2200
66  0.00   4.50  0.795104    0.820  0.503007       0.4700       1.0000
67  0.00   6.65  1.226271    1.090  0.891988       0.5000       1.5850
68  0.00   6.85  0.977417    0.830  0.853280       0.4700       1.2200
69  0.00   4.00  0.748479    0.820  0.461152       0.4300       0.9500
70  0.00   3.77  0.702042    0.500  0.567451       0.4300       0.9400
71  0.00   3.83  0.645458    0.500  0.567419       0.4300       0.8300
72  0.00   5.91  1.155083    0.940  0.842087       0.5000       1.5000
73  0.00   9.74  3.394125    3.100  1.792090       2.1050       4.4250
74  0.00   8.96  3.378479    3.085  1.787360       2.0600       4.4400
75  0.00   8.99  3.244396    3.000  1.630983       2.1200       4.2400
76  0.00   8.50  3.241958    3.015  1.769182       1.8850       4.4400
77  0.00   9.39  3.288271    3.270  1.647528       2.0500       4.3050
78  0.00  10.21  3.280021    3.015  1.700918       2.1200       4.5000
79  0.00   8.01  3.261583    2.980  1.617290       2.0500       4.3200
80  0.00   8.86  3.289542    3.015  1.680170       2.1200       4.2600
81  0.00   9.42  3.479542    3.270  1.761146       2.2400       4.5375
82  0.00   8.32  3.500750    3.285  1.692378       2.1800       4.5575
83  0.00   8.32  3.259729    3.110  1.640243       2.0500       4.3225
84  0.00   9.67  3.432562    3.200  1.732727       2.1575       4.5650
85  0.00  10.00  3.338125    3.080  1.656742       2.1600       4.3350
86  0.00   9.51  3.424646    3.270  1.690960       2.1700       4.5000
87  0.43   9.00  3.340458    3.090  1.699114       2.1200       4.3750

[88 rows x 43 columns]
```

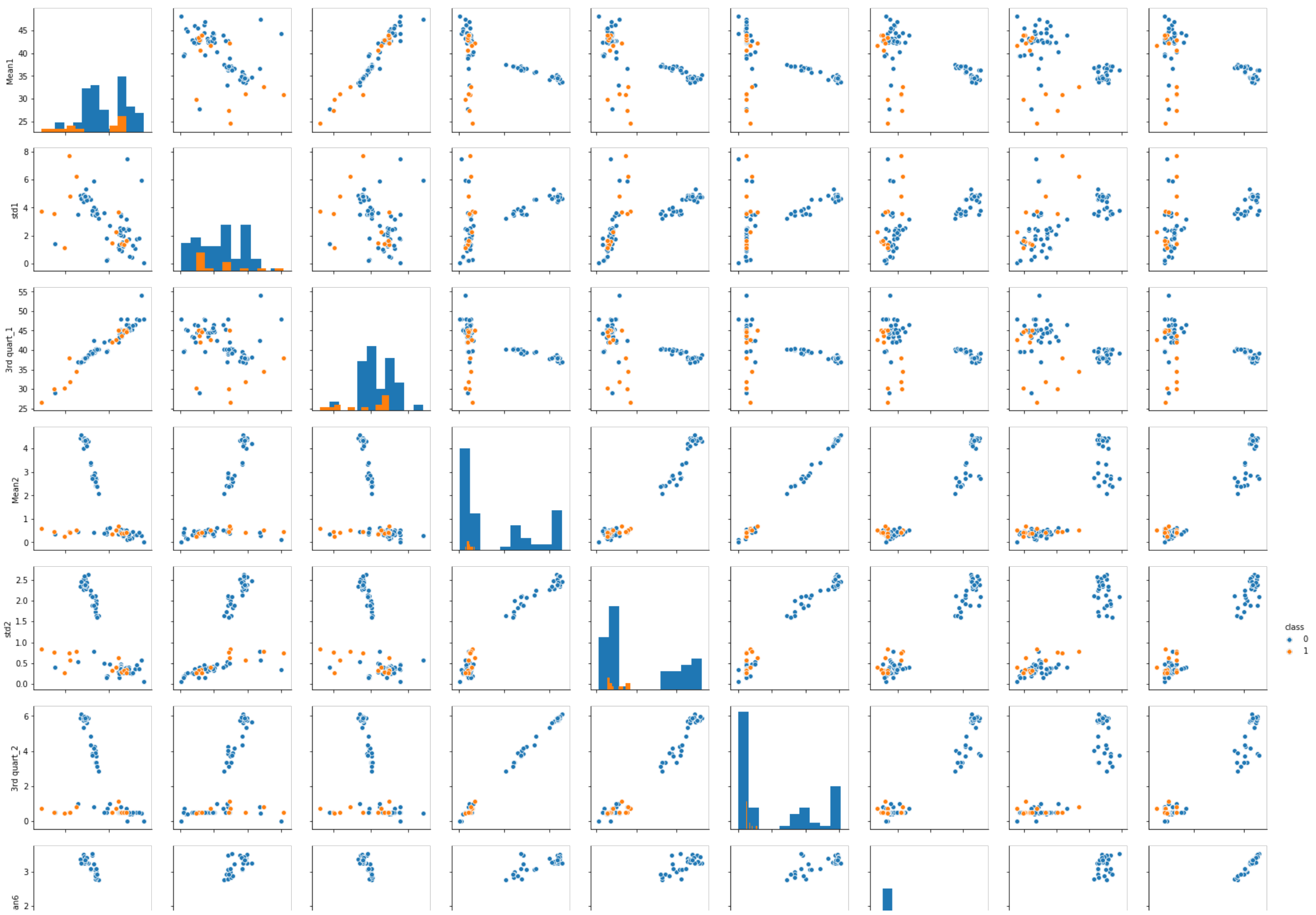
(iii) Mean, standard deviation and 3rd quartile are the most important time domain features the 7 features.

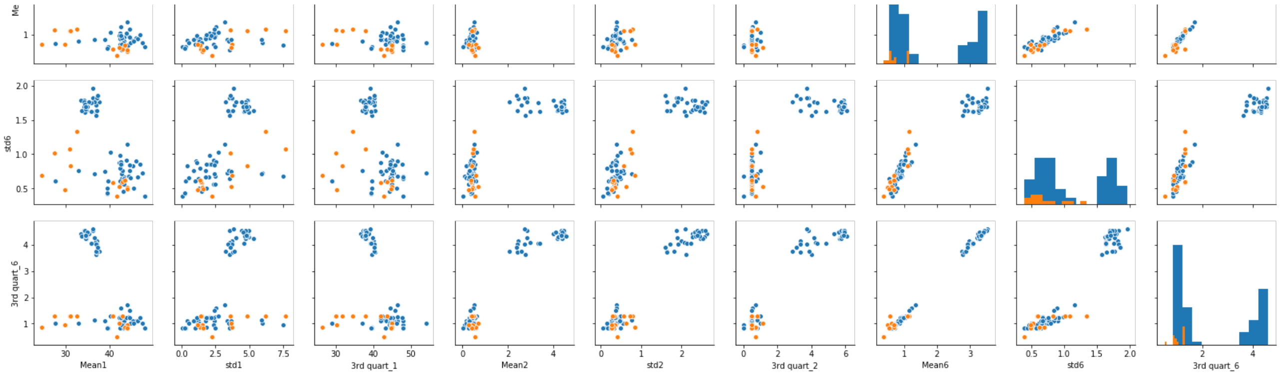
(d) Binary Classification Using Logistic Regression

(i)

In [36]:

```
#Train test table
test_inst = pd.concat([Inst_Table.iloc[0:2], Inst_Table.iloc[7:9], Inst_Table.iloc[13:17], Inst_Tab
le.iloc[28:32],
                       Inst_Table.iloc[43:46], Inst_Table.iloc[58:62],
Inst_Table.iloc[73:76]])

train_inst = Inst_Table.loc[~Inst_Table.index.isin(test_inst.index)]

train_test_inst = pd.concat([test_inst, train_inst], ignore_index = True)



Imp_features = train_test_inst[['Instance', 'Mean1', 'std1', '3rd quart_1', 'Mean2', 'std2', '3rd
quart_2',
                    'Mean6', 'std6', '3rd quart_6', 'class']]


y = train_test_inst.iloc[:, -1]
y_te = train_test_inst.iloc[:4, -1]
y_tr = y.loc[~y.index.isin(y_te.index)]
#Scatterplot
import seaborn as sns
grid = sns.pairplot(data=Imp_features, kind='scatter', vars =['Mean1', 'std1', '3rd quart_1', 'Mean
2', 'std2', '3rd quart_2',
           'Mean6', 'std6', '3rd quart_6'], hue='class', diag_kind='hist')
plt.show()
```

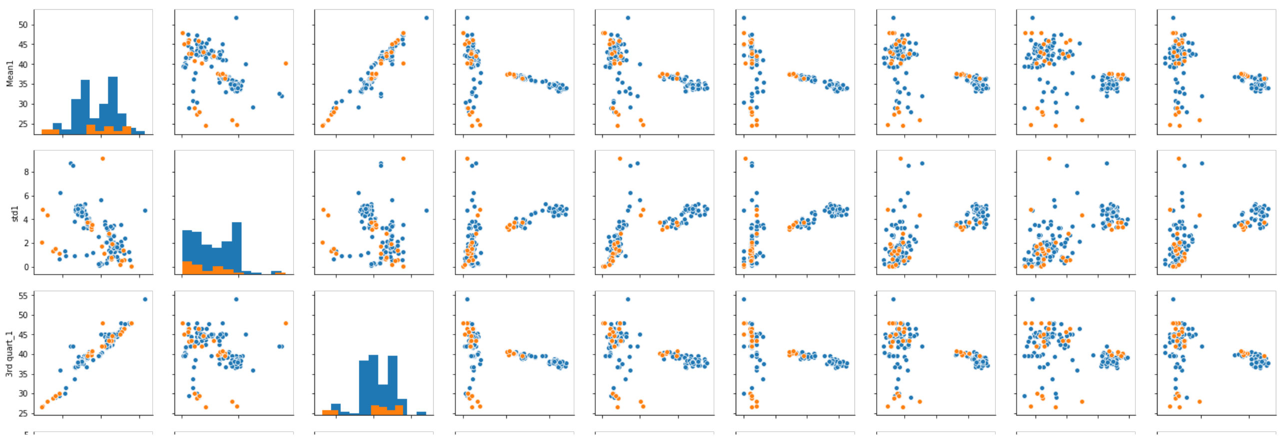


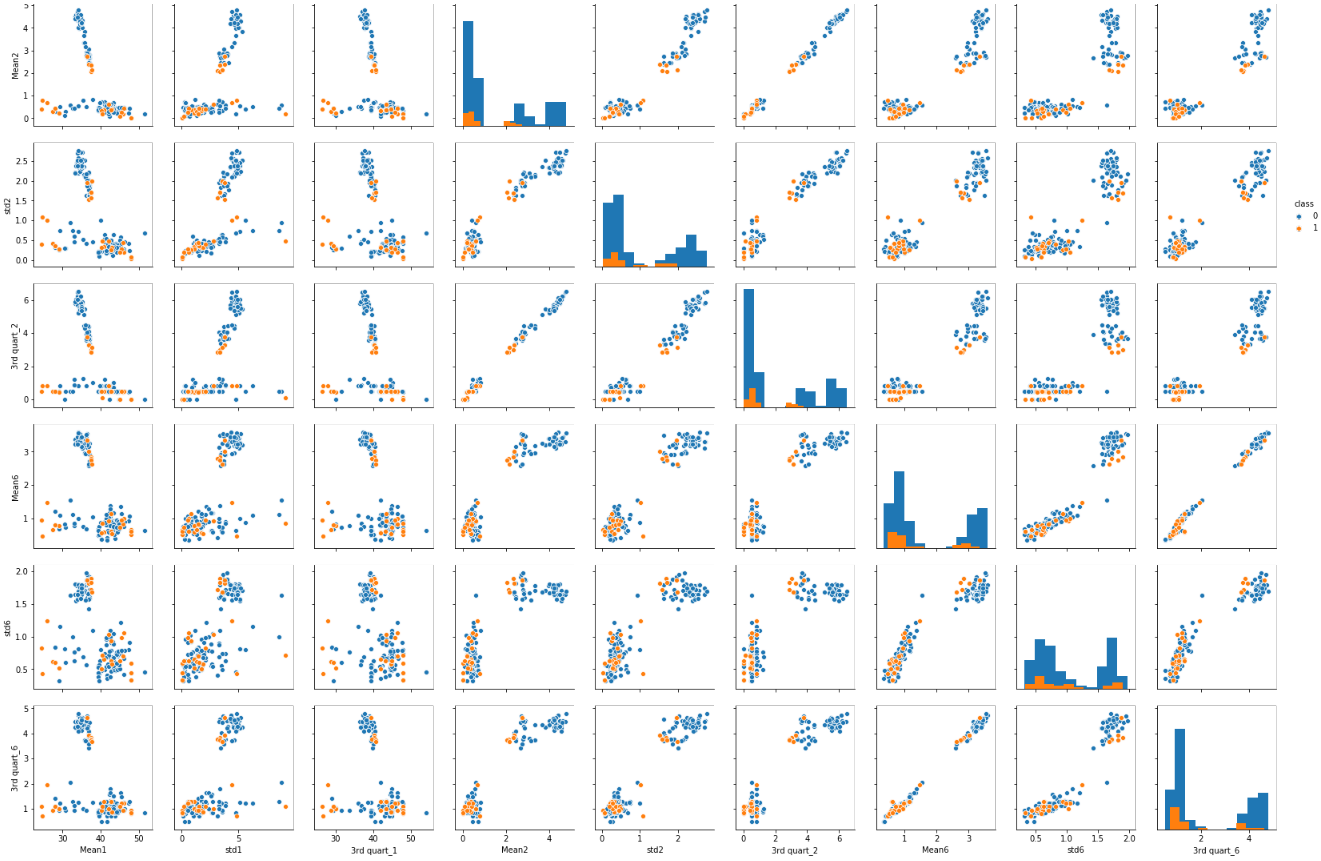


d (iii)

In [92]:

```
Feature2 = [ 'Mean1', 'std1', '3rd quart_1', 'Mean2', 'std2', '3rd quart_2',
                     'Mean6', 'std6', '3rd quart_6', 'class']
def split_inst(df, l,flag):
    length = int(480/l)
    table = pd.DataFrame(np.zeros((int(len(df)/length)+1, 43)), columns=Name)
    table['class'] = 0
    if(flag==0):
        table.iloc[:13*l, -1] = 1
    else:
        table.iloc[:7*l, -1] = 1
        table.iloc[7*l:13*l, -1] = 2
        table.iloc[13*l:28*l, -1] = 3
        table.iloc[28*l:43*l, -1] = 4
        table.iloc[43*l:58*l, -1] = 5
        table.iloc[58*l:73*l, -1] = 6
        table.iloc[73*l:88*l, -1] = 7
    t= 0
    for i in range(0, len(df), length):
        instance = df.iloc[i:i+length]
        j = 0
        for k in range(1, 7):
            table.iloc[t, j+1] = instance.iloc[0:length, k].min()
            table.iloc[t, j+2] = instance.iloc[0:length, k].max()
            table.iloc[t, j+3] = instance.iloc[0:length, k].mean()
            table.iloc[t, j+4] = instance.iloc[0:length, k].median()
            table.iloc[t, j+5] = instance.iloc[0:length, k].std()
            table.iloc[t, j+6] = instance.iloc[0:length, k].quantile(q=.25)
            table.iloc[t, j+7] = instance.iloc[0:length, k].quantile(q=.75)
            j+=7
        t+=1
    return table

Newtable = split_inst(Data_test_train, 2,0)
New_tdf = Newtable[Feature2]

grid = sns.pairplot(data=New_tdf, kind ='scatter', vars = [ 'Mean1', 'std1', '3rd quart_1','Mean2',
'std2', '3rd quart_2',
                     'Mean6', 'std6', '3rd quart_6'] , hue ='class', diag_kind='hist')
plt.show()
```





(iii)RFECV used instead of p-values

In [86]:

```
def tetr_split(df, l):
   te = pd.concat([df.iloc[0:2*l], df.iloc[7*l:9*l], df.iloc[13*l:16*l], df.iloc[28*l:31*l],
                   df.iloc[43*l:46*l], df.iloc[58*l:61*l], df.iloc[73*l:76*l]])
   tr = df.loc[~df.index.isin(test_inst.index)]
   te_tr_table = pd.concat([te, tr], ignore_index = True)
   X_test = df.iloc[:19*l, 0:-1]
   X_train = te_tr_table.loc[~te_tr_table.index.isin(X_test.index)]
   Y = te_tr_table.iloc[:, -1]
   y_test =  Y.iloc[:19*l]
   y_train = Y.loc[~Y.index.isin(y_test.index)]
   X_train = X_train.drop(['Instance', 'class'], axis=1)
   X_test = X_test.drop(['Instance'], axis=1)
   data = [X_train, y_train, X_test, y_test]
   return data


from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFECV, RFE
from sklearn import metrics
from imblearn.under_sampling import RandomUnderSampler
def feat_select(flag):
    best_l = 0
    best_score = 0
    for l in range(1,21):
         matrix = split_inst(Data_test_train, l,0)
         datasets = tetr_split(matrix, l)
         X_train = datasets[0]
         y_train = datasets[1]

         if(flag ==0):
             model = LogisticRegression(C=1e5,solver='warn')
             X_subs = X_train
             y_subs = y_train
         else:
                RUS = RandomUnderSampler()
                model = LogisticRegression()
                X_train_rus, y_train_rus = RUS.fit_sample(X_train, y_train)
                X_subs = X_train_rus
                y_subs = y_train_rus
```

```
                y_subs = y_train_fus

#                 array = np.array(X_train_cc.columns.values)



         rfecv_tr = RFECV(model , cv=5, scoring='accuracy')
         rfecv_tr.fit(X_subs, y_subs)
         X_tr_cols = np.array(X_train.columns.values)
         cv_score = max(rfecv_tr.grid_scores_)
         if cv_score > best_score:
                best_score = cv_score
                best_l = l
                no_optimal_feature = rfecv_tr.n_features_
                opt_f = X_tr_cols[rfecv_tr.support_]
                y_train_opt = y_subs
                X_train_opt = X_subs[opt_f]
                y_test_opt = datasets[3]
                X_test_opt = datasets[2][opt_f]

feat_select(0)
```

In [80]:

```
print('Optimal Feature', opt_f)
print ('The optimal l is', best_l)
print ('The cross validation accuracy for optimal l is', best_score)
log_tr = model.fit(X_train_opt, y_train_opt)
```

```
Optimal Feature ['Mean4' '3rd quart_4']
The optimal l is 1
The cross validation accuracy for optimal l is 0.8956043956043956
```

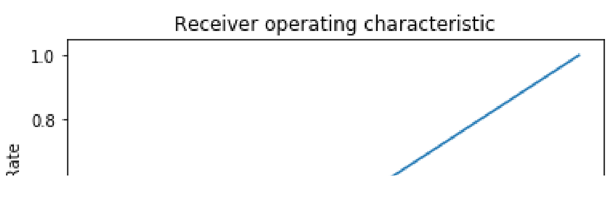
```
C:\Users\hp1\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Defa
ult solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

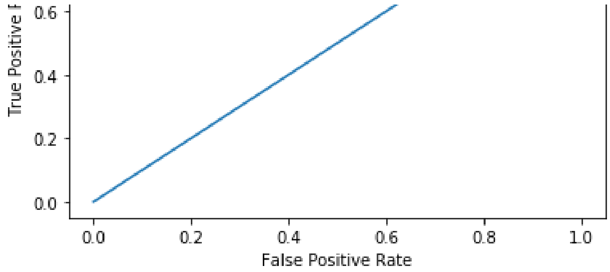
d (iv)

In [81]:

```
# On train data
import statsmodels.api as sm
lgt = sm.Logit(y_train_opt, X_train_opt)
print("TRAINING DATA:")
print('Betas', log_tr.coef_)
print('Confusion matrix ', metrics.confusion_matrix(y_train_opt, log_tr.predict(X_train_opt)))
print("accuracy", log_tr.score(X_train_opt, y_train_opt))

false_pr, true_pr, threshold = metrics.roc_curve(y_train_opt, log_tr.predict(X_train_opt))
plt.plot(false_pr, true_pr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
AUC_tr = metrics.roc_auc_score(y_train_opt, log_tr.predict(X_train_opt))
```

```
TRAINING DATA:
Betas [[ 5.47660608 -4.20552845]]
Confusion matrix  [[57  0]
 [ 9  0]]
accuracy 0.8636363636363636
```





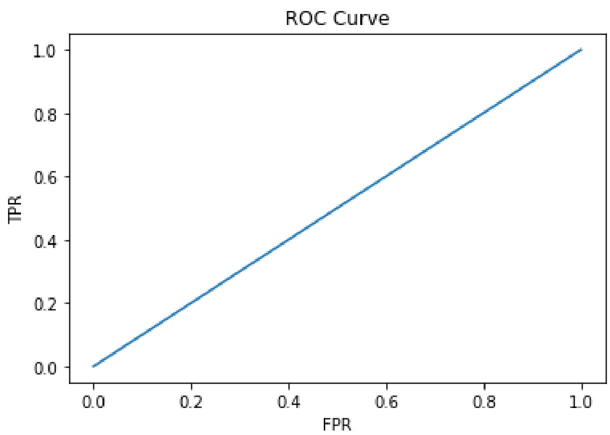
d (v)

In [82]:

```
false_pr, true_pr, threshold = metrics.roc_curve(y_test_opt, log_tr.predict(X_test_opt))
plt.plot(false_pr, true_pr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC Curve')
AUC_te = metrics.roc_auc_score(y_test_opt, log_tr.predict(X_test_opt))


import statsmodels.api as sm
lgt_ts = sm.Logit(y_test_opt, X_test_opt)
smy_ts = lgt_ts.fit(maxiter=5)
smy_ts.summary2()
log_ts = log_tr.predict(X_test_opt)


print("TESTING DATA:")
print('Betas', log_tr.coef_)
print('Confusion matrix', metrics.confusion_matrix(y_test_opt, log_tr.predict(X_test_opt)))
print('AUC', AUC_te)
print('accuracy', log_tr.score(X_test_opt, y_test_opt))
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.412141
         Iterations: 5
TESTING DATA:
Betas [[ 5.47660608 -4.20552845]]
Confusion matrix [[15  0]
 [ 4  0]]
AUC 0.5
accuracy 0.7894736842105263
```

```
C:\Users\hp1\Anaconda3\lib\site-packages\statsmodels\base\model.py:508: ConvergenceWarning:
Maximum Likelihood optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```



d(vi)- The classes seem to be well seperated from the confusion matrix and accuracy of logisitc regression model to cause instability in calculating logistic regression parameters.

d(vii)- the classes are imbalanced thus we use case control sampling for this problem.

In [87]:

```
feat_select(1)
```

```
C:\Users\hp1\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Defa
ult solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [88]:

```
print('The optimal features are', best_features)
```

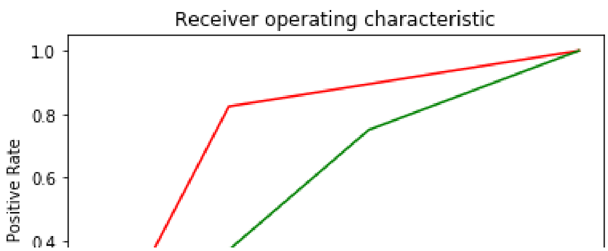
```
The optimal features are ['Mean1' 'Median1' 'Min2' 'Median2' '3rd quart_2' 'Mean4' 'std4'
 '3rd quart_4' 'std6' '1st quart_6' '3rd quart_6']
```

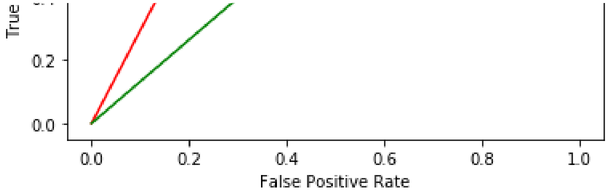
In [89]:

```
#For train data
RUS = RandomUnderSampler()
X_train_optimal_cc, y_train_optimal_cc = RUS.fit_sample(X_train_optimal_cc, y_train_optimal_cc)
log_tr_cc = model.fit(X_train_optimal_cc, y_train_optimal_cc)
print('The confusion matrix', metrics.confusion_matrix(y_train_optimal_cc,
log_tr_cc.predict(X_train_optimal_cc)))
false_UPR, true_UPR, threshold = metrics.roc_curve(y_train_optimal_cc,
log_tr_cc.predict(X_train_optimal_cc))
plt.title('Receiver operating characteristic')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(false_UPR, true_UPR, color='red', label='Train ROC')
AUCU = metrics.roc_auc_score(y_train_optimal_cc, log_tr_cc.predict(X_train_optimal_cc))
print('The AUC train data is', AUCU)

#For test data
print('The confusion matrix is', metrics.confusion_matrix(y_test_optimal_cc, log_tr_cc.predict(X_t
est_optimal_cc)))
false_UPR, true_UPR, threshold = metrics.roc_curve(y_test_optimal_cc,
log_tr_cc.predict(X_test_optimal_cc))
plt.title('Receiver operating characteristic')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(false_UPR, true_UPR, color='green', label='Test ROC')
AUCU = metrics.roc_auc_score(y_test_optimal_cc, log_tr_cc.predict(X_test_optimal_cc))
print('The AUC for test data is', AUCU)
```

```
C:\Users\hp1\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Defa
ult solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
The confusion matrix [[143  56]
 [ 35 164]]
The AUC train data is 0.771356783919598
The confusion matrix is [[110 145]
 [ 17  51]]
The AUC for test data is 0.5906862745098039
```





E(i)Binary Classication Using L1-penalized logistic regression:

In [91]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import cross_val_score, StratifiedKFold
best_score_l1 = 0
best_C_l1 = 0
C = []
for l in range(1,21):
     matrix_l1 = split_inst(Data_test_train, l,0)
     ls = tetr_split(matrix_l1, l)
     X_train_l1 = ls[0]
     y_train_l1 = ls[1]
     stdl1 = StandardScaler()
#Standardization of data
     X_train_l1 = stdl1.fit_transform(X_train_l1)
#Standardization of data
     model = LogisticRegressionCV(penalty='l1', solver='liblinear', cv=5)
     logreg = model.fit(X_train_l1, y_train_l1)
     cv_score_l1 = cross_val_score(logreg, X_train_l1, y_train_l1, cv=StratifiedKFold(5), scoring='
accuracy')
     cv_l1 = np.mean(cv_score_l1)
     if cv_l1 > best_score_l1:
        best_score_l1 = cv_score
        best_l_l1 = l
        C.append(logreg.C_[0])
optimal_C = min(C)                         #Inverse of lambda
#Summarization of attributes
print ('The optimal l is', best_l_l1)
print ('The cross validation accuracy for optimal l is', best_score_l1)
```

E(ii)From cross validation accuracy it can be seen that varaible selection using rfecv performs better but L1-penalized is easier to implement.

F)i)Multi-class Classication:

In [93]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
def multicalss(flag):
    best_score_mult = 0
    best_score_g = 0
    best_score_m = 0
    for l in range(1,21):
        matrix = split_inst(Data_test_train, l,1)
        ls = tetr_split(matrix, l)
        X_train = ls[0]
        y_train = ls[1]
        if(flag==0):
         stdmul = StandardScaler()
#Standardization of data
         X_train_l1 = stdmul.fit_transform(X_train)
#Standardization of data
         model = LogisticRegression(penalty='l1', solver='saga', multi_class='multinomial', max_ite
r=10)
         logreg_mult = model.fit(X_train, y_train)
         accuracy_mult = logreg_mult.score(X_train, y_train)
         array = np.array(X_train_mult.columns.values)
         if accuracy_mult > best_score_mult:
            best_score_mult = accuracy_mult
            best_l_mult = l
            logreg_mult1 = logreg_mult
            y_train_optimal_mult = y_train
            X_train_optimal_mult = X_train
            y_test_optimal_mult = ls[3]
            X_test_optimal_mult = ls[2]
        else:
            gauss = GaussianNB()
            multinom = MultinomialNB()
            logreg_g = gauss.fit(X_train, y_train)
            logreg_m = multinom.fit(X_train, y_train)
            accuracy_g = logreg_g.score(X_train, y_train)
            accuracy_m = logreg_m.score(X_train, y_train)
            if accuracy_g > best_score_g:
                best_score_g = accuracy_g
                best_l_g = l
```

```
                logreg_g1 = logreg_g
                y_train_optimal_g = y_train
                X_train_optimal_g = X_train
                y_test_optimal_g = ls[3]
                X_test_optimal_g = ls[2]
            if accuracy_m > best_score_m:
                best_score_m = accuracy_m
                best_l_m = l
                logreg_m1 = logreg_m
                y_train_optimal_m = y_train
                X_train_optimal_m = X_train
                y_test_optimal_m = ls[3]
                X_test_optimal_m = ls[2]
multicalss(0)
```

In [94]:

```
test_error_mult = 1-logreg_mult1.score(X_test_optimal_mult, y_test_optimal_mult)
print('The test error is', test_error_mult)
print('Confusion matrix for multinomial logistic regression',
      metrics.confusion_matrix(y_test_optimal_mult, logreg_mult1.predict(X_test_optimal_mult)))
```

```
The test error is 0.631578947368421
Confusion matrix for multinomial logistic regression [[0 0 2 0 0 0 0]
 [1 0 1 0 0 0 0]
```

```
 [0  0  2  0  0  0  1]
 [0  0  0  0  2  1  0]
 [0  0  0  0  1  2  0]
 [0  0  0  0  2  1  0]
 [0  0  0  0  0  0  3]]
```

F)ii)Naive Bayes Classication:

In [96]:

```
multicalss(1)
```

In [97]:

```
#For Gausssian
test_error_g = 1-logreg_g1.score(X_test_optimal_g, y_test_optimal_g)
print('The test error for naive bayes with gaussian priors is', test_error_g)
print('Confusion matrix with Gaussian priors for Naive bayes',
      metrics.confusion_matrix(y_test_optimal_g, logreg_g1.predict(X_test_optimal_g)))

#For Multinomial
test_error_m = 1-logreg_m1.score(X_test_optimal_m, y_test_optimal_m)
print('The test error for naive bayes with multinomial priors is', test_error_g)
print('Confusion matrix with Multinomial priors for Naive bayes',
      metrics.confusion_matrix(y_test_optimal_m, logreg_m1.predict(X_test_optimal_m)))
```

```
The test error for naive bayes with gaussian priors is 0.6842105263157895
Confusion matrix with Gaussian priors for Naive bayes [[0 0 2 0 0 0 0]
 [1 0 1 0 0 0 0]
 [2 0 0 1 0 0 0]
 [0 1 0 0 2 0 0]
 [0 2 0 0 0 1 0]
 [0 0 0 0 0 3 0]
 [0 0 0 0 0 0 3]]
The test error for naive bayes with multinomial priors is 0.6842105263157895
Confusion matrix with Multinomial priors for Naive bayes [[0 0 2 0 0 0 0]
 [1 0 1 0 0 0 0]
 [1 0 0 1 0 0 1]
 [0 1 0 0 2 0 0]
 [0 1 0 0 1 1 0]
 [0 0 0 0 2 1 0]
 [0 0 0 0 0 0 3]]
```

F(iii) - L1-penalized Multinomial logistic regression is better than Naive bayes classifier as can be observed from the from the test errors values