In [1]:

```
import numpy as np
import seaborn as sb
import pandas as pd
```

In [2]:

```
import matplotlib.pyplot as mn
%matplotlib inline
```

bending1

In [13]:

```
df1=pd.read_csv('bending1/dataset1.csv',skiprows=4)
df2=pd.read_csv('bending1/dataset2.csv',skiprows=4)
df3=pd.read_csv('bending1/dataset3.csv',skiprows=4)
df4=pd.read_csv('bending1/dataset4.csv',skiprows=4)
df5=pd.read_csv('bending1/dataset5.csv',skiprows=4)
df6=pd.read_csv('bending1/dataset6.csv',skiprows=4)
df7=pd.read_csv('bending1/dataset7.csv',skiprows=4)
```

bending2

In [12]:

```
df8=pd.read_csv('bending2/dataset1.csv',skiprows=4)
df9=pd.read_csv('bending2/dataset2.csv',skiprows=4)
df10=pd.read_csv('bending2/dataset3.csv',skiprows=4)
df11=pd.read_csv('bending2/dataset4.csv',skiprows=4)
df12=pd.read_csv('bending2/dataset5.csv',skiprows=4)
df13=pd.read_csv('bending2/dataset6.csv',skiprows=4)
```

cycling

In [11]:

```
df14=pd.read_csv('cycling/dataset1.csv',skiprows=4)
df15=pd.read_csv('cycling/dataset2.csv',skiprows=4)
df16=pd.read_csv('cycling/dataset3.csv',skiprows=4)
df17=pd.read_csv('cycling/dataset4.csv',skiprows=4)
df18=pd.read_csv('cycling/dataset5.csv',skiprows=4)
df19=pd.read_csv('cycling/dataset6.csv',skiprows=4)
df20=pd.read_csv('cycling/dataset7.csv',skiprows=4)
df21=pd.read_csv('cycling/dataset8.csv',skiprows=4)
df22=pd.read_csv('cycling/dataset9.csv',skiprows=4)
df23=pd.read_csv('cycling/dataset10.csv',skiprows=4)
df24=pd.read_csv('cycling/dataset11.csv',skiprows=4)
df25=pd.read_csv('cycling/dataset12.csv',skiprows=4)
df26=pd.read_csv('cycling/dataset13.csv',skiprows=4)
df27=pd.read_csv('cycling/dataset14.csv',skiprows=4)
df28=pd.read_csv('cycling/dataset15.csv',skiprows=4)
```

Lying

In [10]:

```
df29=pd.read_csv('lying/dataset1.csv',skiprows=4)
df30=pd.read_csv('lying/dataset2.csv',skiprows=4)
df31=pd.read_csv('lying/dataset3.csv',skiprows=4)
df32=pd.read_csv('lying/dataset4.csv',skiprows=4)
df33=pd.read_csv('lying/dataset5.csv',skiprows=4)
df34=pd.read_csv('lying/dataset6.csv',skiprows=4)
df35=pd.read_csv('lying/dataset7.csv',skiprows=4)
df36=pd.read_csv('lying/dataset8.csv',skiprows=4)
```

```
df37=pd.read_csv('lying/dataset9.csv',skiprows=4)
df38=pd.read_csv('lying/dataset10.csv',skiprows=4)
df39=pd.read_csv('lying/dataset11.csv',skiprows=4)
df40=pd.read_csv('lying/dataset12.csv',skiprows=4)
df41=pd.read_csv('lying/dataset13.csv',skiprows=4)
df42=pd.read_csv('lying/dataset14.csv',skiprows=4)
df43=pd.read_csv('lying/dataset15.csv',skiprows=4)
```

sitting

In [9]:

```
df44=pd.read_csv('sitting/dataset1.csv',skiprows=4)
df45=pd.read_csv('sitting/dataset2.csv',skiprows=4)
df46=pd.read_csv('sitting/dataset3.csv',skiprows=4)
df47=pd.read_csv('sitting/dataset4.csv',skiprows=4)
df48=pd.read_csv('sitting/dataset5.csv',skiprows=4)
df49=pd.read_csv('sitting/dataset6.csv',skiprows=4)
df50=pd.read_csv('sitting/dataset7.csv',skiprows=4)
df51=pd.read_csv('sitting/dataset8.csv',skiprows=4)
df52=pd.read_csv('sitting/dataset9.csv',skiprows=4)
df53=pd.read_csv('sitting/dataset10.csv',skiprows=4)
df54=pd.read_csv('sitting/dataset11.csv',skiprows=4)
df55=pd.read_csv('sitting/dataset12.csv',skiprows=4)
df56=pd.read_csv('sitting/dataset13.csv',skiprows=4)
df57=pd.read_csv('sitting/dataset14.csv',skiprows=4)
df58=pd.read_csv('sitting/dataset15.csv',skiprows=4)
```

standing

In [8]:

```
df59=pd.read_csv('standing/dataset1.csv',skiprows=4)
df60=pd.read_csv('standing/dataset2.csv',skiprows=4)
df61=pd.read_csv('standing/dataset3.csv',skiprows=4)
df62=pd.read_csv('standing/dataset4.csv',skiprows=4)
df63=pd.read_csv('standing/dataset5.csv',skiprows=4)
df64=pd.read_csv('standing/dataset6.csv',skiprows=4)
df65=pd.read_csv('standing/dataset7.csv',skiprows=4)
df66=pd.read_csv('standing/dataset8.csv',skiprows=4)
df67=pd.read_csv('standing/dataset9.csv',skiprows=4)
df68=pd.read_csv('standing/dataset10.csv',skiprows=4)
df69=pd.read_csv('standing/dataset11.csv',skiprows=4)
df70=pd.read_csv('standing/dataset12.csv',skiprows=4)
df71=pd.read_csv('standing/dataset13.csv',skiprows=4)
df72=pd.read_csv('standing/dataset14.csv',skiprows=4)
df73=pd.read_csv('standing/dataset15.csv',skiprows=4)
```

walking

In [7]:

```
df74=pd.read_csv('walking/dataset1.csv',skiprows=4)
df75=pd.read_csv('walking/dataset2.csv',skiprows=4)
df76=pd.read_csv('walking/dataset3.csv',skiprows=4)
df77=pd.read_csv('walking/dataset4.csv',skiprows=4)
df78=pd.read_csv('walking/dataset5.csv',skiprows=4)
df79=pd.read_csv('walking/dataset6.csv',skiprows=4)
df80=pd.read_csv('walking/dataset7.csv',skiprows=4)
df81=pd.read_csv('walking/dataset8.csv',skiprows=4)
df82=pd.read_csv('walking/dataset9.csv',skiprows=4)
df83=pd.read_csv('walking/dataset10.csv',skiprows=4)
df84=pd.read_csv('walking/dataset11.csv',skiprows=4)
df85=pd.read_csv('walking/dataset12.csv',skiprows=4)
df86=pd.read_csv('walking/dataset13.csv',skiprows=4)
df87=pd.read_csv('walking/dataset14.csv',skiprows=4)
df88=pd.read_csv('walking/dataset15.csv',skiprows=4)
```

Keep datasets 1 and 2 in folders bending1 and bending 2, as well

as datasets 1,2, and 3 in other folders as test data and other datasets as train data

In [ ]:

```
training_list=[df3,df4,df5,df6,df7,df10,df12,df13,df17,df18,df19,df20,df21,df22,df23,df24,df25,df26
,df27,df28,df32,df33,df34,df35,df36,df37,df38,df39,df40,df41,df42,df43,df47,df48,df49,df50,df51,df
52,df53,df54,df55,df56,df57,df58,df62,df63,df64,df65,df66,df67,df68,df69,df70,df71,df72,df73,df77,
df78,df79,df80,df81,df82,df83,df84,df85,df86,df87,df88]
test_list=[df1,df2,df8,df9,df14,df15,df16,df29,df30,df31,df44,df45,df46,df59,df60,df61,df74,df75,d
f76]
```

FEATURE EXTRACTION

1 (C)(i) The time-domain features minimum, maximum, mean, median, standard deviation, first quartile, and third quartile for all of the 6 time series in each instance are extracted

In [15]:

```
import statistics
df_final1=pd.DataFrame()
list_of_dataframes=[df1,df2,df3,df4,df5,df6,df7,df8,df9,df10,df11,df12,df13,df14,df15,df16,df17,df1
8,df19,df20,df21,df22,df23,df24,df25,df26,df27,df28,df29,df30,df31,df32,df33,df34,df35,df36,df37,d
f38,df39,df40,df41,df42,df43,df44,df45,df46,df47,df48,df49,df50,df51,df52,df53,df54,df55,df56,df57
,df58,df59,df60,df61,df62,df63,df64,df65,df66,df67,df68,df69,df70,df71,df72,df73,df74,df75,df76,df
77,df78,df79,df80,df81,df82,df83,df84,df85,df86,df87,df88]
```

In [ ]:

In [32]:

```
df_final_features=pd.DataFrame(columns=['min1','max1','mean1','median1','std1','1stquart1','3rdquar
t1','min2','max2','mean2','median2','std2','1stquart2','3rdquart2','min3','max3','mean3','median3',
'std3','1stquart3','3rdquart3','min4','max4','mean4','median4','std4','1stquart4','3rdquart4','min5
','max5','mean5','median5','std5','1stquart5','3rdquart5','min6','max6','mean6','median6','std6','1
stquart6','3rdquart6'])
```

In [17]:

```
i=0
for df in list_of_dataframes:
    list_m=[]
    for cols in df.columns[1:7]:
        min1=df[cols].min()
        max1=df[cols].max()
        mean1=statistics.mean(df[cols])
        median1=statistics.median(df[cols])
        std1=df[cols].std()
        Firstquart=np.percentile(df[cols],25)
        Thirdquart=np.percentile(df[cols],75)
        list_m.append(min1)
        list_m.append(max1)
        list_m.append(round(mean1,2))
        list_m.append(median1)
        list_m.append(std1)
        list_m.append(round(Firstquart,2))
        list_m.append(round(Thirdquart,2))
    i=i+1
    df_final_features.loc[i]=list_m
```

In [18]:

```
df_final_features.columns
```

Out[18]:

```
Index(['min1', 'max1', 'mean1', 'median1', 'std1', '1stquart1', '3rdquart1',
       'min2', 'max2', 'mean2', 'median2', 'std2', '1stquart2', '3rdquart2',
       'min3', 'max3', 'mean3', 'median3', 'std3', '1stquart3', '3rdquart3',
       'min4', 'max4', 'mean4', 'median4', 'std4', '1stquart4', '3rdquart4',
       'min5', 'max5', 'mean5', 'median5', 'std5', '1stquart5', '3rdquart5',
       'min6', 'max6', 'mean6', 'median6', 'std6', '1stquart6', '3rdquart6'],
      dtype='object')
```

In [19]:

```
df_final_features
```

Out[19]:

	min1	max1	mean1	median1	std1	1stquart1	3rdquart1	min2	max2	mean2	...	std5	1stquart5	3rdquart5	min6	max
1	37.25	45.00	40.62	40.500	1.476967	39.25	42.00	0.0	1.30	0.36	...	2.188449	33.00	36.00	0.00	1.9
2	38.00	45.67	42.81	42.500	1.435550	42.00	43.67	0.0	1.22	0.37	...	1.995255	32.00	34.50	0.00	3.1
3	35.00	47.40	43.95	44.330	1.558835	43.00	45.00	0.0	1.70	0.43	...	1.999604	35.36	36.50	0.00	1.7
4	33.00	47.75	42.18	43.500	3.670666	39.15	45.00	0.0	3.00	0.70	...	3.849448	30.46	36.33	0.00	2.1
5	33.00	45.75	41.68	41.750	2.243490	41.33	42.75	0.0	2.83	0.54	...	2.411026	28.46	31.25	0.00	1.7
6	37.00	48.00	43.45	43.250	1.386098	42.50	45.00	0.0	1.58	0.38	...	2.488862	22.25	24.00	0.00	5.2
7	36.25	48.00	43.97	44.500	1.618364	43.31	44.67	0.0	1.50	0.41	...	3.318301	20.50	23.75	0.00	2.9
8	12.75	51.00	24.56	24.250	3.737514	23.19	26.50	0.0	6.87	0.59	...	3.693786	20.50	27.00	0.00	4.9
9	0.00	42.75	27.46	28.000	3.583582	25.50	30.00	0.0	7.76	0.45	...	5.053642	15.00	20.75	0.00	6.7
10	21.00	50.00	32.59	33.000	6.238143	26.19	34.50	0.0	9.90	0.52	...	5.032424	17.67	23.50	0.00	13.6
11	27.50	33.00	29.88	30.000	1.153837	29.00	30.27	0.0	1.00	0.26	...	1.745970	17.00	19.00	0.00	6.4
12	19.00	45.50	30.94	29.000	7.684146	26.75	38.00	0.0	6.40	0.47	...	5.845911	15.00	20.81	0.00	6.7
13	25.00	47.50	31.06	29.710	4.829794	27.50	31.81	0.0	6.38	0.41	...	7.853427	9.00	18.31	0.00	4.9
14	24.25	45.00	37.18	36.250	3.581301	34.50	40.25	0.0	8.58	2.37	...	2.890347	17.95	21.75	0.00	9.3
15	28.75	44.75	37.56	36.875	3.226507	35.25	40.25	0.0	9.91	2.08	...	2.727377	18.00	21.50	0.00	9.6
16	22.00	44.67	37.06	36.000	3.710180	34.50	40.06	0.0	14.17	2.44	...	3.537144	16.00	21.00	0.00	8.5
17	19.00	44.00	36.23	36.000	3.528617	34.00	39.00	0.0	12.28	2.83	...	3.166655	14.00	18.06	0.00	9.9
18	26.50	44.33	36.69	36.000	3.529404	34.25	39.37	0.0	12.89	2.97	...	2.978238	14.67	18.50	0.00	8.1
19	25.33	45.00	37.11	36.250	3.710385	34.50	40.25	0.0	10.84	2.73	...	2.847876	14.75	18.50	0.00	9.5
20	26.75	44.75	36.86	36.330	3.555787	34.50	39.75	0.0	11.68	2.76	...	2.655906	15.00	18.67	0.00	8.8
21	26.25	44.25	36.96	36.290	3.434863	34.50	40.25	0.0	8.64	2.42	...	2.851673	14.00	18.25	0.00	8.3
22	27.75	44.67	37.14	36.330	3.758904	34.00	40.50	0.0	10.76	2.42	...	2.689291	15.00	18.75	0.00	8.7
23	27.00	45.00	36.82	36.000	3.900459	33.75	40.25	0.0	10.47	2.60	...	2.781030	15.50	19.27	0.00	8.9
24	27.00	44.33	36.54	36.000	4.018922	33.25	39.81	0.0	10.43	2.85	...	3.088141	15.00	19.50	0.00	9.1
25	18.50	44.25	35.75	36.000	4.614802	33.00	39.33	0.0	12.60	3.33	...	3.120057	14.00	18.06	0.00	9.3
26	19.00	43.75	35.88	36.000	4.614878	33.00	39.50	0.0	11.20	3.41	...	3.537635	14.75	19.69	0.00	8.5
27	23.33	43.50	36.24	36.750	3.822016	33.46	39.25	0.0	9.71	2.74	...	3.617702	15.75	21.00	0.00	11.1
28	24.25	45.00	37.18	36.250	3.581301	34.50	40.25	0.0	8.58	2.37	...	2.890347	17.95	21.75	0.00	9.3
29	23.50	30.00	27.72	27.500	1.442253	27.00	29.00	0.0	1.79	0.36	...	4.074511	5.50	10.75	0.00	4.5
30	24.75	48.33	44.18	48.000	7.495615	48.00	48.00	0.0	3.11	0.10	...	3.274539	2.00	5.54	0.00	3.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
59	33.33	48.00	44.33	45.000	2.476940	42.25	46.50	0.0	3.90	0.43	...	5.401794	9.33	17.75	0.00	5.0
60	35.50	46.25	43.17	43.670	1.989052	42.50	44.50	0.0	2.12	0.51	...	2.983976	12.75	16.50	0.00	5.7
61	32.75	47.00	42.76	44.500	3.398919	41.33	45.37	0.0	3.34	0.49	...	4.296574	13.00	18.57	0.00	5.7
62	30.00	46.67	42.65	42.750	2.395338	41.50	45.00	0.0	2.95	0.40	...	3.141679	10.63	14.25	0.00	4.6
63	36.00	47.50	43.72	45.000	2.384105	43.00	45.00	0.0	1.92	0.37	...	3.289138	11.31	15.54	0.00	6.1
64	34.50	47.75	44.47	45.000	1.772553	45.00	45.25	0.0	2.18	0.29	...	2.612390	12.00	14.81	0.00	4.3

	min1	max1	mean1	median1	std1	1stquart1	3rdquart1	min2	max2	mean2	...	std5	1stquart5	3rdquart5	min6	max6
65	35.50	48.00	46.22	46.000	1.748315	45.25	48.00	0.0	4.50	0.32	...	2.931585	12.00	15.25	0.00	6.9
66	29.75	48.00	46.93	47.500	1.832665	47.24	47.75	0.0	4.60	0.43	...	3.134822	11.67	15.50	0.00	6.5
67	36.33	47.67	45.40	45.500	1.328121	45.00	46.33	0.0	1.66	0.46	...	3.374095	11.25	14.50	0.00	4.5
68	36.00	45.80	42.42	42.670	2.520129	41.33	44.62	0.0	2.12	0.46	...	3.722074	7.63	12.00	0.00	6.6
69	37.00	48.25	42.52	42.500	2.195751	41.00	44.50	0.0	2.12	0.44	...	3.623557	12.63	17.50	0.00	6.8
70	36.25	45.50	42.96	42.670	1.500878	42.00	44.33	0.0	2.60	0.35	...	2.702605	14.00	16.69	0.00	4.0
71	36.00	47.33	42.67	43.670	2.384170	40.00	44.75	0.0	2.17	0.42	...	3.261617	12.75	16.50	0.00	3.7
72	36.25	45.75	43.19	44.750	2.491162	39.75	45.00	0.0	2.83	0.27	...	3.566038	16.50	21.00	0.00	3.8
73	36.00	47.33	44.44	45.000	2.417797	44.63	45.75	0.0	4.50	0.35	...	3.414454	11.00	14.67	0.00	5.9
74	19.33	43.50	34.23	35.500	4.889576	30.50	37.75	0.0	14.50	4.00	...	3.092094	14.75	18.67	0.00	9.7
75	12.50	45.00	33.51	34.125	4.850923	30.50	36.75	0.0	13.05	4.45	...	3.133564	14.63	18.75	0.00	8.9
76	15.00	46.75	34.66	35.000	5.315110	31.00	38.25	0.0	13.44	4.20	...	3.155015	14.25	18.50	0.00	8.9
77	18.00	46.00	35.19	36.000	4.751868	32.00	38.75	0.0	16.20	4.32	...	3.207642	14.25	18.50	0.00	8.5
78	20.75	46.25	34.76	35.290	4.742208	31.67	38.25	0.0	12.68	4.22	...	3.174681	14.25	18.33	0.00	9.3
79	21.50	51.00	34.94	35.500	4.645944	32.00	38.06	0.0	12.21	4.12	...	3.192058	14.24	18.25	0.00	10.2
80	18.33	47.67	34.33	34.750	4.948770	31.25	38.00	0.0	12.48	4.40	...	3.000493	13.75	18.00	0.00	8.0
81	18.33	45.75	34.60	35.125	4.731790	31.50	38.00	0.0	15.37	4.40	...	2.905688	14.00	18.25	0.00	8.8
82	15.50	43.67	34.23	34.750	4.441798	31.25	37.25	0.0	17.24	4.35	...	2.992920	14.33	18.25	0.00	9.4
83	21.50	51.25	34.25	35.000	4.940741	30.94	37.75	0.0	13.55	4.46	...	3.116627	13.75	18.00	0.00	8.3
84	19.50	45.33	33.59	34.250	4.650935	30.25	37.00	0.0	14.67	4.58	...	3.283983	13.73	18.25	0.00	8.3
85	19.75	45.50	34.32	35.250	4.752477	31.00	38.00	0.0	13.47	4.46	...	3.119856	13.50	17.75	0.00	9.6
86	19.50	46.00	34.55	35.250	4.842294	31.25	37.81	0.0	12.47	4.37	...	2.823124	14.00	17.75	0.00	10.0
87	23.50	46.25	34.87	35.250	4.531720	31.75	38.25	0.0	14.82	4.38	...	3.131076	13.75	18.00	0.00	9.5
88	19.25	44.00	34.47	35.000	4.796705	31.25	38.00	0.0	13.86	4.36	...	3.156320	13.73	17.75	0.43	9.0

88 rows × 42 columns

In [20]:

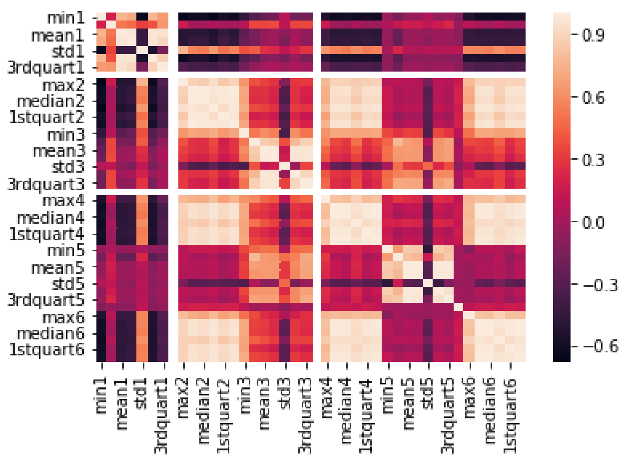
```
corr=df_final_features.corr()
```

In [21]:

```
sb.heatmap(corr)
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x23ff1f2be10>
```



1(c)(iii) Three important features max,mean and median of each time series is extracted

time series is extracted

In [22]:

```
df_feature_extraction=df_final_features.filter(items=['max1','mean1','median1','max2','mean2','medi
an2','max3','mean3','median3','max4','mean4','median4','max5','mean5','median5','max6','mean6','med
ian6'])
```

In [23]:

```
df_feature_extraction
```

Out[23]:

	max1	mean1	median1	max2	mean2	median2	max3	mean3	median3	max4	mean4	median4	max5	mean5	median5	max6
1	45.00	40.62	40.500	1.30	0.36	0.430	29.50	19.04	19.250	7.23	0.83	0.500	38.25	34.31	35.000	1.92
2	45.67	42.81	42.500	1.22	0.37	0.470	29.50	20.10	21.000	5.76	0.88	0.500	38.50	33.02	33.000	3.11
3	47.40	43.95	44.330	1.70	0.43	0.470	29.75	22.12	23.000	4.44	0.50	0.430	38.50	35.59	36.000	1.79
4	47.75	42.18	43.500	3.00	0.70	0.500	30.00	22.18	23.000	5.15	0.99	0.830	38.67	33.49	35.000	2.18
5	45.75	41.68	41.750	2.83	0.54	0.500	28.25	19.01	19.125	6.42	0.84	0.500	37.50	29.86	30.000	1.79
6	48.00	43.45	43.250	1.58	0.38	0.470	27.00	15.79	15.000	10.03	0.85	0.500	33.50	23.03	23.500	5.26
7	48.00	43.97	44.500	1.50	0.41	0.470	26.33	15.87	16.250	5.17	0.67	0.470	30.75	22.10	21.670	2.96
8	51.00	24.56	24.250	6.87	0.59	0.430	25.33	19.12	20.250	6.76	0.74	0.470	30.00	23.49	23.750	4.97
9	42.75	27.46	28.000	7.76	0.45	0.430	35.00	20.84	20.750	5.76	0.78	0.500	33.00	17.62	18.000	6.76
10	50.00	32.59	33.000	9.90	0.52	0.430	28.25	13.94	14.250	7.40	0.87	0.500	33.75	20.35	19.585	13.61
11	33.00	29.88	30.000	1.00	0.26	0.000	14.50	8.17	8.750	4.44	0.54	0.470	23.25	18.12	18.000	6.40
12	45.50	30.94	29.000	6.40	0.47	0.430	32.75	14.59	15.750	11.42	0.78	0.470	36.00	18.39	17.500	6.73
13	47.50	31.06	29.710	6.38	0.41	0.430	28.33	15.30	15.000	5.32	0.82	0.500	40.33	14.41	13.000	4.92
14	45.00	37.18	36.250	8.58	2.37	1.920	26.75	16.53	16.670	8.05	2.91	2.620	25.50	19.61	20.000	9.34
15	44.75	37.56	36.875	9.91	2.08	1.700	24.67	16.57	17.000	8.32	3.03	2.950	24.33	19.52	20.000	9.62
16	44.67	37.06	36.000	14.17	2.44	1.920	24.00	16.39	16.500	9.74	2.98	2.860	24.25	18.13	18.875	8.55
17	44.00	36.23	36.000	12.28	2.83	2.285	25.25	15.42	15.250	9.50	3.12	2.940	24.50	15.87	16.000	9.98
18	44.33	36.69	36.000	12.89	2.97	2.360	28.25	18.40	18.000	9.63	2.98	2.685	24.67	16.48	16.750	8.19
19	45.00	37.11	36.250	10.84	2.73	2.240	27.25	16.66	16.670	10.57	3.14	2.870	23.33	16.49	16.670	9.50
20	44.75	36.86	36.330	11.68	2.76	2.230	27.00	16.49	16.000	9.01	3.01	2.860	23.00	16.58	16.750	8.81
21	44.25	36.96	36.290	8.64	2.42	2.050	26.50	15.31	15.250	8.06	2.78	2.565	22.25	15.99	16.330	8.34
22	44.67	37.14	36.330	10.76	2.42	1.880	24.75	15.00	15.000	9.00	2.85	2.500	23.00	16.77	17.000	8.75
23	45.00	36.82	36.000	10.47	2.60	2.120	25.00	15.30	15.000	10.61	2.94	2.620	24.67	17.30	17.415	8.99
24	44.33	36.54	36.000	10.43	2.85	2.450	27.67	16.16	16.000	9.63	3.06	2.870	24.50	17.06	16.750	9.18
25	44.25	35.75	36.000	12.60	3.33	2.830	27.00	16.06	16.000	9.46	2.87	2.650	24.33	16.00	16.250	9.39
26	43.75	35.88	36.000	11.20	3.41	2.920	26.50	16.69	17.000	8.87	3.13	2.870	26.50	17.08	17.000	8.50
27	43.50	36.24	36.750	9.71	2.74	2.170	28.50	18.44	18.330	9.78	3.13	2.895	27.00	18.50	18.500	11.15
28	45.00	37.18	36.250	8.58	2.37	1.920	26.75	16.53	16.670	8.05	2.91	2.620	25.50	19.61	20.000	9.34
29	30.00	27.72	27.500	1.79	0.36	0.430	13.25	6.08	6.250	5.02	0.87	0.820	21.00	8.34	8.750	4.50
30	48.33	44.18	48.000	3.11	0.10	0.000	16.50	6.68	6.250	5.91	0.58	0.430	12.75	4.38	3.330	3.91
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59	48.00	44.33	45.000	3.90	0.43	0.470	18.75	11.65	12.250	5.79	0.84	0.500	23.00	13.44	14.750	5.02
60	46.25	43.17	43.670	2.12	0.51	0.500	20.67	12.77	13.000	6.56	0.69	0.485	21.25	14.29	14.670	5.72
61	47.00	42.76	44.500	3.34	0.49	0.470	21.00	15.04	15.250	5.85	0.59	0.430	21.33	15.55	16.585	5.73
62	46.67	42.65	42.750	2.95	0.40	0.430	21.25	18.13	18.500	7.50	0.47	0.430	20.75	12.06	12.290	4.64
63	47.50	43.72	45.000	1.92	0.37	0.430	21.00	17.01	17.750	6.02	0.54	0.430	20.25	13.20	13.750	6.18
64	47.75	44.47	45.000	2.18	0.29	0.000	21.33	17.95	18.500	5.54	0.57	0.430	19.67	13.21	13.000	4.32
65	48.00	46.22	46.000	4.50	0.31	0.000	21.00	15.03	15.585	5.12	0.60	0.470	21.00	13.39	13.500	6.00
66	48.00	46.93	47.500	4.60	0.43	0.500	21.00	16.85	18.000	6.52	0.54	0.430	21.25	13.28	13.670	6.58

	max1	mean1	median1	max2	mean2	median2	max3	mean3	median3	max4	mean4	median4	max5	mean5	median5	max6
67	47.67	43.40	43.500	1.66	0.46	0.500	21.25	17.67	18.000	4.11	0.44	0.430	19.75	12.43	13.125	4.50
68	45.80	42.42	42.670	2.12	0.46	0.470	24.00	16.32	16.750	5.59	0.75	0.500	22.00	10.07	9.750	6.65
69	48.25	42.52	42.500	2.12	0.44	0.470	21.75	13.22	13.500	5.61	0.80	0.500	21.00	14.64	15.000	6.85
70	45.50	42.96	42.670	2.60	0.35	0.470	22.00	11.78	12.000	4.72	0.56	0.470	20.25	14.95	15.250	4.00
71	47.33	42.67	43.670	2.17	0.42	0.470	21.00	12.11	12.670	5.56	0.57	0.430	19.67	14.25	14.500	3.77
72	45.75	43.19	44.750	2.83	0.27	0.000	22.75	12.73	12.710	3.74	0.64	0.470	24.00	18.20	18.250	3.83
73	47.33	44.44	45.000	4.50	0.35	0.430	21.00	13.36	13.500	5.54	0.65	0.470	21.00	12.61	12.750	5.91
74	43.50	34.23	35.500	14.50	4.00	3.630	23.50	15.71	15.750	8.86	3.30	3.200	26.00	16.62	16.670	9.74
75	45.00	33.51	34.125	13.05	4.45	4.085	23.75	15.56	15.635	9.10	3.35	3.110	25.00	16.54	16.750	8.96
76	46.75	34.66	35.000	13.44	4.20	3.900	25.25	15.22	15.250	8.58	3.11	2.870	24.50	16.25	16.330	8.99
77	46.00	35.19	36.000	16.20	4.32	3.880	24.50	15.46	15.670	8.76	3.07	2.860	23.50	16.10	16.330	8.50
78	46.25	34.76	35.290	12.68	4.22	3.900	23.75	15.24	15.330	9.20	3.21	3.000	25.50	16.30	16.250	9.39
79	51.00	34.94	35.500	12.21	4.12	3.845	23.33	15.52	15.500	9.09	3.09	2.870	25.00	16.00	16.250	10.21
80	47.67	34.33	34.750	12.48	4.40	3.900	23.33	15.56	15.500	9.01	3.20	2.930	24.00	15.86	16.000	8.01
81	45.75	34.60	35.125	15.37	4.40	4.025	24.00	15.17	15.000	9.18	3.15	3.015	23.25	16.06	16.000	8.86
82	43.67	34.23	34.750	17.24	4.35	3.900	23.00	15.61	15.500	9.20	3.37	3.030	24.00	16.15	16.250	9.42
83	51.25	34.25	35.000	13.55	4.46	4.150	24.00	15.25	15.250	9.50	3.28	3.100	24.25	15.72	15.750	8.32
84	45.33	33.59	34.250	14.67	4.58	4.260	23.25	15.32	15.330	9.00	3.23	3.100	25.00	15.89	16.000	8.32
85	45.50	34.32	35.250	13.47	4.46	3.900	22.25	15.21	15.250	9.00	3.28	3.110	23.25	15.55	15.750	9.67
86	46.00	34.55	35.250	12.47	4.37	4.135	22.67	15.19	15.250	8.34	3.03	2.860	22.75	15.76	15.750	10.00
87	46.25	34.87	35.250	14.82	4.38	3.925	24.25	15.47	15.500	9.90	3.21	3.030	23.50	15.87	16.000	9.51
88	44.00	34.47	35.000	13.86	4.36	3.960	22.75	15.42	15.500	9.10	3.19	3.030	23.25	15.70	16.000	9.00

88 rows × 18 columns

In [ ]:

```
pip ins
```

1(d)(i)

SCATTER PLOTS ARE DEPICTED FOR THE FEATURES EXTRACTED FROM TIME SERIES 1,2 AND 6

In [24]:

```
df_feature_scatter=df_feature_extraction.filter(items=['max1','mean1','median1','max2','mean2','med
ian2','max6','mean6','median6'])
```

In [25]:

```
df_feature_scatter
```

Out[25]:

	max1	mean1	median1	max2	mean2	median2	max6	mean6	median6
1	45.00	40.62	40.500	1.30	0.36	0.430	1.92	0.57	0.430
2	45.67	42.81	42.500	1.22	0.37	0.470	3.11	0.57	0.430
3	47.40	43.95	44.330	1.70	0.43	0.470	1.79	0.49	0.430
4	47.75	42.18	43.500	3.00	0.70	0.500	2.18	0.61	0.500
5	45.75	41.68	41.750	2.83	0.54	0.500	1.79	0.38	0.430
6	48.00	43.45	43.250	1.58	0.38	0.470	5.26	0.68	0.500
7	48.00	43.97	44.500	1.50	0.41	0.470	2.96	0.56	0.490

	max1	mean1	median1	max2	mean2	median2	max6	mean6	median6
8	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
9	42.75	27.46	28.000	7.76	0.45	0.430	6.76	1.12	0.830
10	50.00	32.59	33.000	9.90	0.52	0.430	13.61	1.16	0.830
11	33.00	29.88	30.000	1.00	0.26	0.000	6.40	0.70	0.710
12	45.50	30.94	29.000	6.40	0.47	0.430	6.73	1.11	0.830
13	47.50	31.06	29.710	6.38	0.41	0.430	4.92	1.10	0.940
14	45.00	37.18	36.250	8.58	2.37	1.920	9.34	2.92	2.500
15	44.75	37.56	36.875	9.91	2.08	1.700	9.62	2.77	2.450
16	44.67	37.06	36.000	14.17	2.44	1.920	8.55	2.98	2.570
17	44.00	36.23	36.000	12.28	2.83	2.285	9.98	3.48	3.340
18	44.33	36.69	36.000	12.89	2.97	2.360	8.19	3.07	2.690
19	45.00	37.11	36.250	10.84	2.73	2.240	9.50	3.08	2.770
20	44.75	36.86	36.330	11.68	2.76	2.230	8.81	2.77	2.590
21	44.25	36.96	36.290	8.64	2.42	2.050	8.34	2.93	2.525
22	44.67	37.14	36.330	10.76	2.42	1.880	8.75	2.82	2.590
23	45.00	36.82	36.000	10.47	2.60	2.120	8.99	2.89	2.525
24	44.33	36.54	36.000	10.43	2.85	2.450	9.18	3.23	2.870
25	44.25	35.75	36.000	12.60	3.33	2.830	9.39	3.07	2.770
26	43.75	35.88	36.000	11.20	3.41	2.920	8.50	3.09	2.930
27	43.50	36.24	36.750	9.71	2.74	2.170	11.15	3.53	3.110
28	45.00	37.18	36.250	8.58	2.37	1.920	9.34	2.92	2.500
29	30.00	27.72	27.500	1.79	0.36	0.430	4.50	0.73	0.710
30	48.33	44.18	48.000	3.11	0.10	0.000	3.91	0.69	0.500
...	...	...	...	...	...	...	...	...	...
59	48.00	44.33	45.000	3.90	0.43	0.470	5.02	0.93	0.830
60	46.25	43.17	43.670	2.12	0.51	0.500	5.72	0.91	0.830
61	47.00	42.76	44.500	3.34	0.49	0.470	5.73	0.84	0.710
62	46.67	42.65	42.750	2.95	0.40	0.430	4.64	0.92	0.830
63	47.50	43.72	45.000	1.92	0.37	0.430	6.18	1.04	0.830
64	47.75	44.47	45.000	2.18	0.29	0.000	4.32	0.93	0.830
65	48.00	46.22	46.000	4.50	0.31	0.000	6.00	0.88	0.830
66	48.00	46.93	47.500	4.60	0.43	0.500	6.58	0.99	0.830
67	47.67	45.40	45.500	1.66	0.46	0.500	4.50	0.80	0.820
68	45.80	42.42	42.670	2.12	0.46	0.470	6.65	1.23	1.090
69	48.25	42.52	42.500	2.12	0.44	0.470	6.85	0.98	0.830
70	45.50	42.96	42.670	2.60	0.35	0.470	4.00	0.75	0.820
71	47.33	42.67	43.670	2.17	0.42	0.470	3.77	0.70	0.500
72	45.75	43.19	44.750	2.83	0.27	0.000	3.83	0.65	0.500
73	47.33	44.44	45.000	4.50	0.35	0.430	5.91	1.16	0.940
74	43.50	34.23	35.500	14.50	4.00	3.630	9.74	3.39	3.100
75	45.00	33.51	34.125	13.05	4.45	4.085	8.96	3.38	3.085
76	46.75	34.66	35.000	13.44	4.20	3.900	8.99	3.24	3.000
77	46.00	35.19	36.000	16.20	4.32	3.880	8.50	3.24	3.015
78	46.25	34.76	35.290	12.68	4.22	3.900	9.39	3.29	3.270
79	51.00	34.94	35.500	12.21	4.12	3.845	10.21	3.28	3.015
80	47.67	34.33	34.750	12.48	4.40	3.900	8.01	3.26	2.980
81	45.75	34.60	35.125	15.37	4.40	4.025	8.86	3.29	3.015
82	43.67	34.23	34.750	17.24	4.35	3.900	9.42	3.48	3.270
83	51.25	34.25	35.000	13.55	4.46	4.150	8.32	3.50	3.285
84	45.33	33.59	34.250	14.67	4.58	4.260	8.32	3.26	3.110

	max1	mean1	median1	max2	mean2	median2	max6	mean6	median6
85	45.50	34.32	35.250	13.47	4.46	3.900	9.67	3.43	3.200
86	46.00	34.55	35.250	12.47	4.37	4.135	10.00	3.34	3.080
87	46.25	34.87	35.250	14.82	4.38	3.925	9.51	3.42	3.270
88	44.00	34.47	35.000	13.86	4.36	3.960	9.00	3.34	3.090

88 rows × 9 columns

In [205]:

```
df_scatterplot=df_scatterplot.reset_index(drop=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-205-4b514ad014b6> in <module>
----> 1 df_scatterplot=df_scatterplot.reset_index(drop=True)

NameError: name 'df_scatterplot' is not defined
```

In [202]:

```
list_bending=[1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

In [204]:

```
df_scatterplot['Label']=list_bending
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-204-0edf9f49cfe6> in <module>
----> 1 df_scatterplot['Label']=list_bending

NameError: name 'df_scatterplot' is not defined
```

In [395]:

```
df_scatterplot
```

Out[395]:

	max1	mean1	median1	max2	mean2	median2	max6	mean6	median6	Label
0	47.40	43.95	44.330	1.70	0.43	0.470	1.79	0.49	0.430	1
1	47.75	42.18	43.500	3.00	0.70	0.500	2.18	0.61	0.500	1
2	45.75	41.68	41.750	2.83	0.54	0.500	1.79	0.38	0.430	1
3	48.00	43.45	43.250	1.58	0.38	0.470	5.26	0.68	0.500	1
4	48.00	43.97	44.500	1.50	0.41	0.470	2.96	0.56	0.490	1
5	50.00	32.59	33.000	9.90	0.52	0.430	13.61	1.16	0.830	1
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1
7	45.50	30.94	29.000	6.40	0.47	0.430	6.73	1.11	0.830	1
8	47.50	31.06	29.710	6.38	0.41	0.430	4.92	1.10	0.940	1
9	44.00	36.23	36.000	12.28	2.83	2.285	9.98	3.48	3.340	0
10	44.33	36.69	36.000	12.89	2.97	2.360	8.19	3.07	2.690	0
11	45.00	37.11	36.250	10.84	2.73	2.240	9.50	3.08	2.770	0
12	44.75	36.86	36.330	11.68	2.76	2.230	8.81	2.77	2.590	0
13	44.25	36.96	36.290	8.64	2.42	2.050	8.34	2.93	2.525	0
14	44.67	37.14	36.330	10.76	2.42	1.880	8.75	2.82	2.590	0
15	45.00	36.82	36.000	10.47	2.60	2.120	8.99	2.89	2.525	0
16	44.33	36.54	36.000	10.43	2.85	2.450	9.18	3.23	2.870	0
17	44.25	35.75	36.000	12.60	3.33	2.830	9.39	3.07	2.770	0

	max1	mean1	median1	max2	mean2	median2	max6	mean6	median6	Label
18	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0
19	43.50	36.24	36.750	9.71	2.74	2.170	11.15	3.53	3.110	0
20	45.00	37.18	36.250	8.58	2.37	1.920	9.34	2.92	2.500	0
21	51.00	42.71	40.500	4.85	0.52	0.500	4.97	0.55	0.470	0
22	41.00	39.67	39.500	1.00	0.58	0.500	3.49	0.64	0.500	0
23	40.67	39.51	39.500	1.00	0.50	0.500	3.19	0.62	0.500	0
24	40.00	39.43	39.500	1.00	0.42	0.470	4.06	0.58	0.500	0
25	40.00	39.35	39.330	0.50	0.37	0.470	3.50	0.59	0.500	0
26	56.25	47.33	42.670	8.49	0.27	0.000	5.72	0.77	0.500	0
27	30.00	27.72	27.500	1.79	0.36	0.430	4.50	0.74	0.710	0
28	48.25	48.00	48.000	0.43	0.01	0.000	2.50	0.64	0.500	0
29	41.00	39.67	39.500	1.00	0.58	0.500	3.49	0.64	0.500	0
...	...	...	...	...	...	...	...	...	...	...
39	48.50	40.22	39.250	3.28	0.62	0.500	6.36	1.06	0.830	0
40	48.25	43.88	45.250	3.28	0.52	0.500	7.00	1.35	1.090	0
41	45.00	42.11	42.000	1.09	0.34	0.470	6.36	0.96	0.820	0
42	44.75	42.28	41.500	1.00	0.50	0.500	7.85	0.87	0.820	0
43	44.67	42.36	42.000	1.00	0.48	0.500	4.64	0.72	0.500	0
44	46.00	42.73	43.250	4.72	0.56	0.500	5.10	0.89	0.710	0
45	46.67	42.65	42.750	2.95	0.40	0.430	4.64	0.92	0.830	0
46	47.50	43.72	45.000	1.92	0.37	0.430	6.18	1.04	0.830	0
47	47.75	44.47	45.000	2.18	0.29	0.000	4.32	0.93	0.830	0
48	48.00	46.22	46.000	4.50	0.31	0.000	6.00	0.88	0.830	0
49	48.00	46.93	47.500	4.60	0.43	0.500	6.58	0.99	0.830	0
50	47.67	45.40	45.500	1.66	0.46	0.500	4.50	0.80	0.820	0
51	45.80	42.42	42.670	2.12	0.46	0.470	6.65	1.23	1.090	0
52	48.25	42.52	42.500	2.12	0.44	0.470	6.85	0.98	0.830	0
53	45.50	42.96	42.670	2.60	0.35	0.470	4.00	0.75	0.820	0
54	47.33	42.67	43.670	2.17	0.42	0.470	3.77	0.70	0.500	0
55	45.75	43.19	44.750	2.83	0.27	0.000	3.83	0.65	0.500	0
56	47.33	44.44	45.000	4.50	0.35	0.430	5.91	1.16	0.940	0
57	46.00	35.19	36.000	16.20	4.32	3.880	8.50	3.24	3.015	0
58	46.25	34.76	35.290	12.68	4.22	3.900	9.39	3.29	3.270	0
59	51.00	34.94	35.500	12.21	4.12	3.845	10.21	3.28	3.015	0
60	47.67	34.33	34.750	12.48	4.40	3.900	8.01	3.26	2.980	0
61	45.75	34.60	35.125	15.37	4.40	4.025	8.86	3.29	3.015	0
62	43.67	34.23	34.750	17.24	4.35	3.900	9.42	3.48	3.270	0
63	51.25	34.25	35.000	13.55	4.46	4.150	8.32	3.50	3.285	0
64	45.33	33.59	34.250	14.67	4.58	4.260	8.32	3.26	3.110	0
65	45.50	34.32	35.250	13.47	4.46	3.900	9.67	3.43	3.200	0
66	46.00	34.55	35.250	12.47	4.37	4.135	10.00	3.34	3.080	0
67	46.25	34.87	35.250	14.82	4.38	3.925	9.51	3.42	3.270	0
68	44.00	34.47	35.000	13.86	4.36	3.960	9.00	3.34	3.090	0

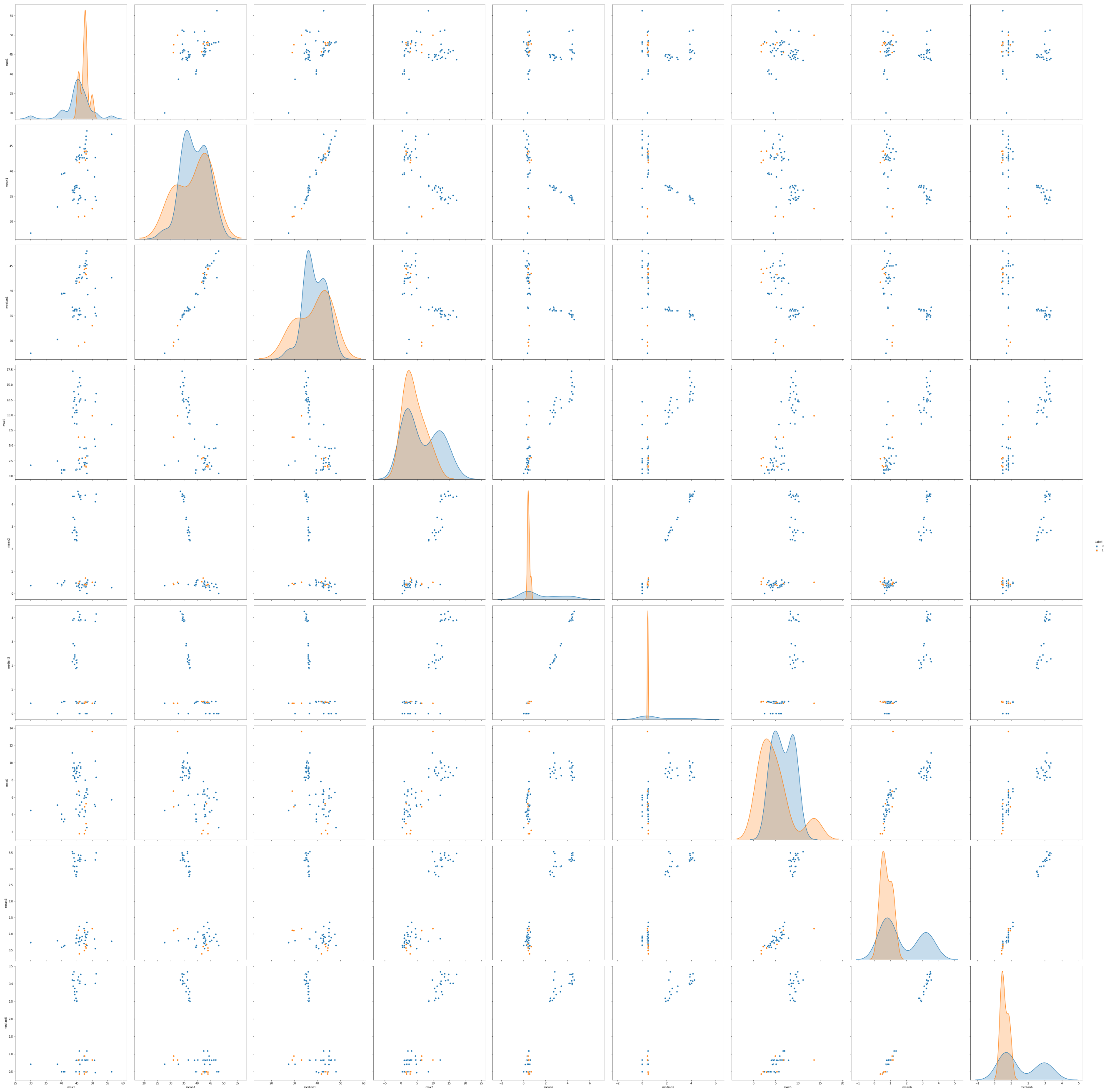
69 rows × 10 columns

In [420]:

```
sb.pairplot(df_scatterplot,hue='Label',dropna=True,height=6,vars=['max1','mean1','median1','max2','mean2','median2','max6','mean6','median6'])
```

Out[420]:

```
<seaborn.axisgrid.PairGrid at 0x196813cf860>
```



1(d)(ii)

Breaking each time series in your training set into two (approximately) equal length time series and the scatter plots are drawn for the features obtained

In [128]:

```
df_scatterplot_more=pd.DataFrame(columns=['max1','mean1','median1','max2','mean2','median2','max12'
,'mean12','median12'])
```

In [609]:

```
training_list=[df3,df4,df5,df6,df7,df10,df12,df13,df17,df18,df19,df20,df21,df22,df23,df24,df25,df26
,df27,df28,df32,df33,df34,df35,df36,df37,df38,df39,df40,df41,df42,df43,df47,df48,df49,df50,df51,df
52,df53,df54,df55,df56,df57,df58,df62,df63,df64,df65,df66,df67,df68,df69,df70,df71,df72,df73,df77,
df78,df79,df80,df81,df82,df83,df84,df85,df86,df87,df88]
```

In [230]:

```
i=0
for df in training_list:
    check_list=[]
    dfz=[]
    df=df.drop('# Columns: time',axis=1)
    dfz=df[240:480]
    dfz=dfz.reset_index(drop=True)
    dfz=pd.concat([df,dfz],axis=1)
    dfz=dfz.drop(dfz.index[240:480])
    columns_list=[dfz.columns[0],dfz.columns[1],dfz.columns[11]]
    for cols in columns_list:
        maximum=df[cols].max()
        mean=statistics.mean(df[cols])
        median=statistics.median(df[cols])
        check_list.append(maximum)
        check_list.append(mean)
        check_list.append(median)
    i=i+1
    df_scatterplot_more.loc[i]=check_list
```

In [231]:

```
df_scatterplot_more['Label']=list_bending
```

In [232]:

```
df_scatterplot_more
```

Out[232]:

	max1	mean1	median1	max2	mean2	median2	max12	mean12	median12	Label
1	47.40	43.954500	44.330	1.70	0.426250	0.470	1.79	0.493292	0.430	1
2	47.75	42.179812	43.500	3.00	0.696042	0.500	2.18	0.613521	0.500	1
3	45.75	41.678063	41.750	2.83	0.535979	0.500	1.79	0.383292	0.430	1
4	48.00	43.454958	43.250	1.58	0.378083	0.470	5.26	0.679646	0.500	1
5	48.00	43.969125	44.500	1.50	0.413125	0.470	2.96	0.555312	0.490	1
6	50.00	32.586208	33.000	9.90	0.516125	0.430	13.61	1.162042	0.830	1
7	45.50	30.938104	29.000	6.40	0.467167	0.430	6.73	1.107354	0.830	1
8	47.50	31.058250	29.710	6.38	0.405458	0.430	4.92	1.098104	0.940	1
9	44.00	36.228396	36.000	12.28	2.831688	2.285	9.98	3.480688	3.340	1
10	44.33	36.687292	36.000	12.89	2.973042	2.360	8.19	3.073313	2.690	0
11	45.00	37.114312	36.250	10.84	2.730000	2.240	9.50	3.076354	2.770	0
12	44.75	36.863375	36.330	11.68	2.757312	2.230	8.81	2.773312	2.590	0
13	44.25	36.957458	36.290	8.64	2.420083	2.050	8.34	2.934625	2.525	0
14	44.67	37.144833	36.330	10.76	2.419062	1.880	8.75	2.822437	2.590	0
15	45.00	36.819521	36.000	10.47	2.600146	2.120	8.99	2.887563	2.525	0
16	44.33	36.541667	36.000	10.43	2.847958	2.450	9.18	3.225458	2.870	0
17	44.25	35.752354	36.000	12.60	3.328104	2.830	9.39	3.069667	2.770	0
18	43.75	35.879875	36.000	11.20	3.414312	2.920	8.50	3.093021	2.930	0
19	43.50	36.244083	36.750	9.71	2.736021	2.170	11.15	3.530500	3.110	0
20	45.00	37.177042	36.250	8.58	2.374208	1.920	9.34	2.921729	2.500	0
21	51.00	42.706063	40.500	4.85	0.519813	0.500	4.97	0.549312	0.470	0
22	41.00	39.667833	39.500	1.00	0.583604	0.500	3.49	0.635937	0.500	0
23	40.67	39.506188	39.500	1.00	0.496479	0.500	3.19	0.622917	0.500	0
24	40.00	39.433792	39.500	1.00	0.422104	0.470	4.06	0.582708	0.500	0
25	40.00	39.347104	39.330	0.50	0.366396	0.470	3.50	0.588458	0.500	0
26	56.25	47.325125	42.670	8.49	0.274313	0.000	5.72	0.766167	0.500	0

	max1	mean1	median1	max2	mean2	median2	max12	mean12	median12	Label
27	30.00	27.716375	27.500	1.79	0.363687	0.430	4.50	0.735396	0.710	0
28	48.25	48.004167	48.000	0.43	0.007167	0.000	2.50	0.641229	0.500	0
29	41.00	39.667833	39.500	1.00	0.583604	0.500	3.49	0.635937	0.500	0
30	40.00	39.433792	39.500	1.00	0.422104	0.470	4.06	0.582708	0.500	0
...	...	...	...	...	...	...	...	...	...	...
40	48.25	43.884833	45.250	3.28	0.517354	0.500	7.00	1.354917	1.090	0
41	45.00	42.111583	42.000	1.09	0.341938	0.470	6.36	0.961167	0.820	0
42	44.75	42.282667	41.500	1.00	0.498354	0.500	7.85	0.869000	0.820	0
43	44.67	42.360188	42.000	1.00	0.482500	0.500	4.64	0.719812	0.500	0
44	46.00	42.728854	43.250	4.72	0.555333	0.500	5.10	0.892083	0.710	0
45	46.67	42.648521	42.750	2.95	0.402833	0.430	4.64	0.917354	0.830	0
46	47.50	43.720021	45.000	1.92	0.366708	0.430	6.18	1.039688	0.830	0
47	47.75	44.471146	45.000	2.18	0.290479	0.000	4.32	0.927375	0.830	0
48	48.00	46.224938	46.000	4.50	0.312354	0.000	6.00	0.882583	0.830	0
49	48.00	46.932208	47.500	4.60	0.429667	0.500	6.58	0.991125	0.830	0
50	47.67	45.399625	45.500	1.66	0.460146	0.500	4.50	0.795104	0.820	0
51	45.80	42.419917	42.670	2.12	0.460562	0.470	6.65	1.226271	1.090	0
52	48.25	42.516958	42.500	2.12	0.440688	0.470	6.85	0.977417	0.830	0
53	45.50	42.959354	42.670	2.60	0.352875	0.470	4.00	0.748479	0.820	0
54	47.33	42.674583	43.670	2.17	0.419167	0.470	3.77	0.702042	0.500	0
55	45.75	43.187521	44.750	2.83	0.271271	0.000	3.83	0.645458	0.500	0
56	47.33	44.441187	45.000	4.50	0.346604	0.430	5.91	1.155083	0.940	0
57	46.00	35.193333	36.000	16.20	4.321021	3.880	8.50	3.241958	3.015	0
58	46.25	34.763333	35.290	12.68	4.223792	3.900	9.39	3.288271	3.270	0
59	51.00	34.935812	35.500	12.21	4.115750	3.845	10.21	3.280021	3.015	0
60	47.67	34.333042	34.750	12.48	4.396958	3.900	8.01	3.261583	2.980	0
61	45.75	34.599875	35.125	15.37	4.398833	4.025	8.86	3.289542	3.015	0
62	43.67	34.225875	34.750	17.24	4.354500	3.900	9.42	3.479542	3.270	0
63	51.25	34.253521	35.000	13.55	4.457896	4.150	8.32	3.500750	3.285	0
64	45.33	33.586875	34.250	14.67	4.576562	4.260	8.32	3.259729	3.110	0
65	45.50	34.322750	35.250	13.47	4.456333	3.900	9.67	3.432562	3.200	0
66	46.00	34.546229	35.250	12.47	4.371958	4.135	10.00	3.338125	3.080	0
67	46.25	34.873229	35.250	14.82	4.380583	3.925	9.51	3.424646	3.270	0
68	44.00	34.473188	35.000	13.86	4.359312	3.960	9.00	3.340458	3.090	0
69	44.00	34.473188	35.000	13.86	4.359312	3.960	9.00	3.340458	3.090	0

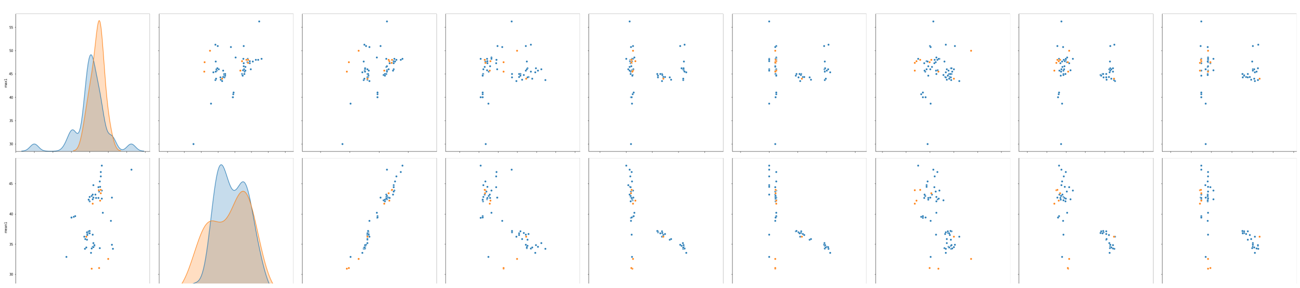
69 rows × 10 columns

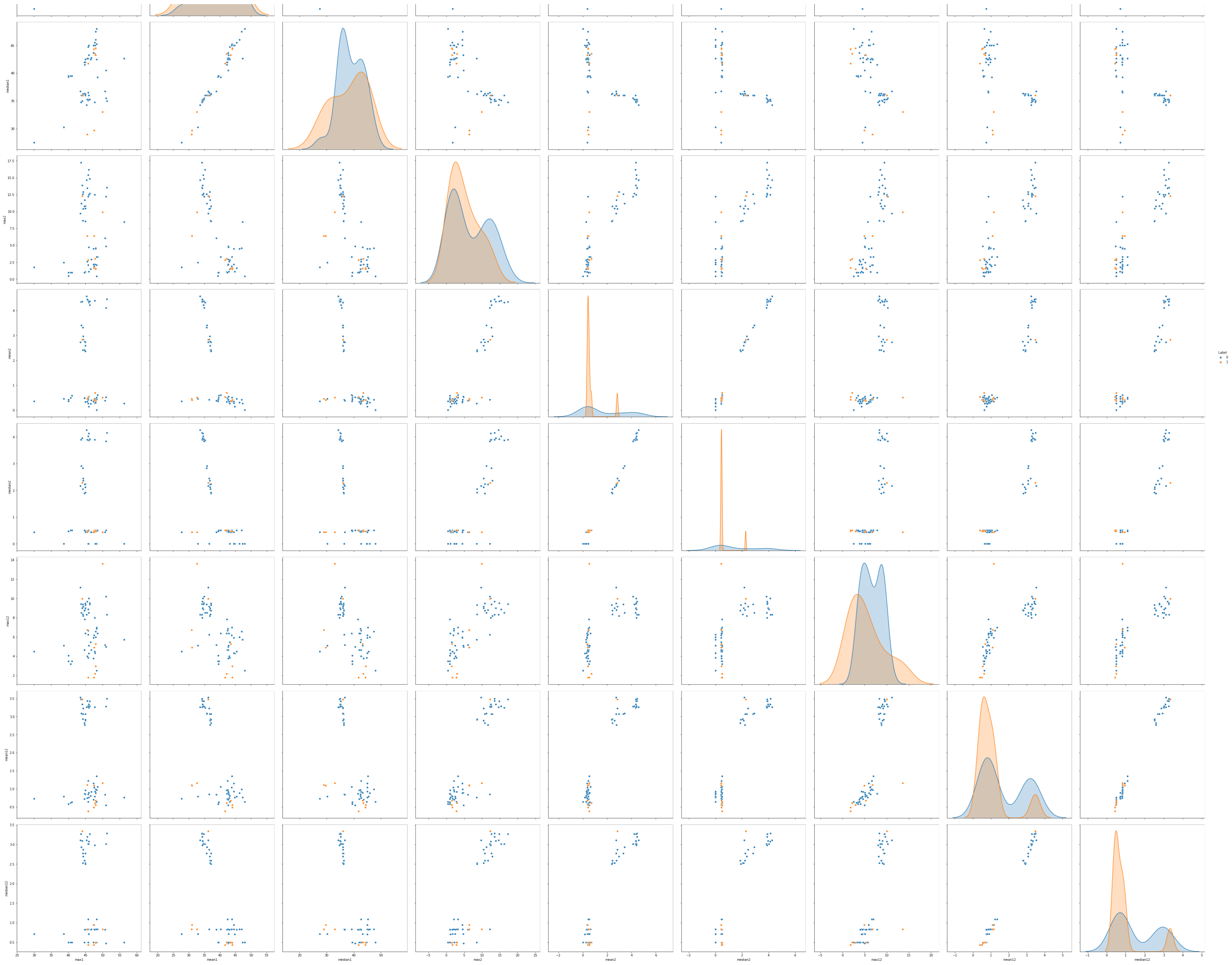
In [233]:

```
sb.pairplot(df_scatterplot_more,hue='Label',dropna=True,height=6,vars=['max1','mean1','median1','ma
x2','mean2','median2','max12','mean12','median12'])
```

Out[233]:

```
<seaborn.axisgrid.PairGrid at 0x15c595507f0>
```





we could see that there are more data points in each subplot, and some patterns are more clearn compared to that of the previous one in 1(d)(i)

1(d)(iii)

Break each time series in your training set into l ∈ {1, 2, . . . , 20} time series of approximately equal length and use logistic regression5 to solve the binary classification problem, using time-domain features. Calculate the p-values for your logistic regression parameters and refit a logistic regression model using your pruned set of features.6 Alternatively, you can use backward selection using sklearn.feature selection or glm in R. Use 5-fold cross-validation to de- termine the best value of l. Explain what the right way and the wrong way are to perform cross-validation in this problem.7 Obviously, use the right way! Also, you may encounter the problem of class imbalance, which may make some of your folds not having any instances of the rare class. In such a case, you can use stratified cross validation. Research what it means and use it if needed

In [186]:

```
df_checking=pd.DataFrame()
```

In [50]:

```
import statsmodels.api as sm
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.feature_selection import f_regression
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
```

In [234]:

```
i=0
for df in training_list:
    dataframes_final=[]
    dataframes_final=pd.DataFrame()
    for item in np.split(df,3):
        dataframes=pd.DataFrame(item)
        dataframes=dataframes.drop('# Columns: time',axis=1)
        dataframes=dataframes.reset_index(drop=True)
        dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
        dataframes=[]
    dataframes_final.columns=range(1,19,1)
    columns_list=dataframes_final.columns
    list_of_features=[]
    for cols in columns_list:
        min1=dataframes_final[cols].min()
        max1=dataframes_final[cols].max()
        mean1=statistics.mean(dataframes_final[cols])
        median1=statistics.median(dataframes_final[cols])
        std1=dataframes_final[cols].std()
        Firstquart=np.percentile(dataframes_final[cols],25)
        Thirdquart=np.percentile(dataframes_final[cols],75)
        list_of_features.append(min1)
        list_of_features.append(max1)
        list_of_features.append(round(mean1,2))
        list_of_features.append(median1)
        list_of_features.append(std1)
        list_of_features.append(round(Firstquart,2))
        list_of_features.append(round(Thirdquart,2))
    array_features=np.array([list_of_features])
    if i==0:
        print(1)
        i=i+1
        df_checking=pd.DataFrame(array_features)
    else:
        df_checking.loc[i]=list_of_features
        i=i+1
```

1

Usage of all the features for the Cross Validation(the wrong way)

In [238]:

```
df_checking['label']=list_bending[1:69]
```

In [244]:

```
df_train=df_checking.drop(['label'],axis=1)
df_test=df_checking['label']
LR_classifier=LogisticRegression()
training_result=LR_classifier.fit(df_train,df_test)
cv_score=cross_val_score(LR_classifier,df_train,df_test,cv=5)
accuracy=np.mean(cv_score)
print(accuracy)
```

```
0.9703296703296704
```

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

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

In [345]:

```
i=0
for l in range(1,21,1):
    i=0
    df_checking=pd.DataFrame()
    for df in training_list:
        dataframes_final=pd.DataFrame()
        for item in np.array_split(df,l):
            dataframes=pd.DataFrame(item)
            dataframes=dataframes.drop('# Columns: time',axis=1)
            dataframes=dataframes.reset_index(drop=True)
            dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
            dataframes=[]
        dataframes_final=dataframes_final.fillna(method='ffill')
        dataframes_final.columns=range(1,(6*l)+1,1)
        columns_list=dataframes_final.columns
        list_of_features=[]
        for cols in columns_list:
            min1=dataframes_final[cols].min()
            max1=dataframes_final[cols].max()
            mean1=statistics.mean(dataframes_final[cols])
            median1=statistics.median(dataframes_final[cols])
            std1=dataframes_final[cols].std()
            Firstquart=np.percentile(dataframes_final[cols],25)
            Thirdquart=np.percentile(dataframes_final[cols],75)
            list_of_features.append(min1)
            list_of_features.append(max1)
            list_of_features.append(round(mean1,2))
            list_of_features.append(median1)
            list_of_features.append(std1)
            list_of_features.append(round(Firstquart,2))
            list_of_features.append(round(Thirdquart,2))
        array_features=np.array([list_of_features])
        if i==0:
            i=i+1
            df_checking=pd.DataFrame(array_features)
        else:
            df_checking.loc[i]=list_of_features
            i=i+1
    df_checking['label']=list_bending[1:69]
    df_train=df_checking.drop(['label'],axis=1)
    df_test=df_checking['label']
    LR_classifier=LogisticRegression(solver='liblinear')
    training_result=LR_classifier.fit(df_train,df_test)
    cv_score=cross_val_score(LR_classifier,df_train,df_test,cv=5)
    accuracy=np.mean(cv_score)
    print('The predict accrucy of l= '+str(l)+' is '+accuracy.astype('str'))
    columns_list=[]
```

```
The predict accrucy of l= 1 is 0.9857142857142858
The predict accrucy of l= 2 is 0.956043956043956
The predict accrucy of l= 3 is 0.9703296703296704
The predict accrucy of l= 4 is 0.9549450549450551
The predict accrucy of l= 5 is 0.9549450549450551
The predict accrucy of l= 6 is 0.9703296703296704
The predict accrucy of l= 7 is 0.9549450549450551
The predict accrucy of l= 8 is 0.9549450549450551
The predict accrucy of l= 9 is 0.9549450549450551
```

```
The predict accrucy of l= 9 is 0.9549450549450551
The predict accrucy of l= 10 is 0.9406593406593406
The predict accrucy of l= 11 is 0.9549450549450551
The predict accrucy of l= 12 is 0.9549450549450551
The predict accrucy of l= 13 is 0.9549450549450551
The predict accrucy of l= 14 is 0.9549450549450551
The predict accrucy of l= 15 is 0.9549450549450551
The predict accrucy of l= 16 is 0.9549450549450551
The predict accrucy of l= 17 is 0.9549450549450551
The predict accrucy of l= 18 is 0.9549450549450551
The predict accrucy of l= 19 is 0.9549450549450551
The predict accrucy of l= 20 is 0.9549450549450551
```

Here the unimportant features are first eliminated through the Recursive feature elimination and then the Cross Validation technique is applied(the right way)

In [492]:

```python
i=0
for l in range(1,21,1):
    i=0
    df_checking=pd.DataFrame()
    for df in training_list:
        dataframes_final=pd.DataFrame()
        for item in np.array_split(df,l):
            dataframes=pd.DataFrame(item)
            dataframes=dataframes.drop('# Columns: time',axis=1)
            dataframes=dataframes.reset_index(drop=True)
            dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
            dataframes=[]
        dataframes_final=dataframes_final.fillna(method='ffill')
        dataframes_final.columns=range(1,(6*l)+1,1)
        columns_list=dataframes_final.columns
        list_of_features=[]
        for cols in columns_list:
            min1=dataframes_final[cols].min()
            max1=dataframes_final[cols].max()
            mean1=statistics.mean(dataframes_final[cols])
            median1=statistics.median(dataframes_final[cols])
            std1=dataframes_final[cols].std()
            Firstquart=np.percentile(dataframes_final[cols],25)
            Thirdquart=np.percentile(dataframes_final[cols],75)
            list_of_features.append(min1)
            list_of_features.append(max1)
            list_of_features.append(round(mean1,2))
            list_of_features.append(median1)
            list_of_features.append(std1)
            list_of_features.append(round(Firstquart,2))
            list_of_features.append(round(Thirdquart,2))
        array_features=np.array([list_of_features])
        if i==0:
            i=i+1
            df_checking=pd.DataFrame(array_features)
        else:
            df_checking.loc[i]=list_of_features
            i=i+1
    df_checking['label']=list_bending[1:69]
    df_train=df_checking.drop(['label'],axis=1)
    df_test=df_checking['label']
    from sklearn.feature_selection import RFE
    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression(solver='liblinear')
    rfe = RFE(model)
    rfe = rfe.fit(df_train, df_test)
    #print(rfe.support_)
    #print(rfe.ranking_)
    f = rfe.get_support(1) #the most important features
    X = df_checking[df_checking.columns[f]]
    LR_classifier=LogisticRegression(solver='liblinear')
    training_result=LR_classifier.fit(X,df_test)
    cv_score=cross_val_score(LR_classifier,X,df_test,cv=5)
    accuracy=np.mean(cv_score)
    pruned_features=np.count_nonzero(rfe.ranking_==1)
```

```
    print('The predict accrucy of l= '+str(l)+' when used pruned features= '+str(pruned_features)+'
is '+accuracy.astype('str'))
    columns_list=[]
    pruned_features=[]
```

```
The predict accrucy of l= 1 when used pruned features= 21 is 0.9857142857142858
The predict accrucy of l= 2 when used pruned features= 42 is 0.956043956043956
The predict accrucy of l= 3 when used pruned features= 63 is 0.9703296703296704
The predict accrucy of l= 4 when used pruned features= 84 is 0.9703296703296704
The predict accrucy of l= 5 when used pruned features= 105 is 0.9857142857142858
The predict accrucy of l= 6 when used pruned features= 126 is 0.9703296703296704
The predict accrucy of l= 7 when used pruned features= 147 is 0.9703296703296704
The predict accrucy of l= 8 when used pruned features= 168 is 0.9549450549450551
The predict accrucy of l= 9 when used pruned features= 189 is 0.9857142857142858
The predict accrucy of l= 10 when used pruned features= 210 is 0.9703296703296704
The predict accrucy of l= 11 when used pruned features= 231 is 0.9549450549450551
The predict accrucy of l= 12 when used pruned features= 252 is 0.9703296703296704
The predict accrucy of l= 13 when used pruned features= 273 is 0.9703296703296704
The predict accrucy of l= 14 when used pruned features= 294 is 0.9549450549450551
The predict accrucy of l= 15 when used pruned features= 315 is 0.9549450549450551
The predict accrucy of l= 16 when used pruned features= 336 is 0.9703296703296704
The predict accrucy of l= 17 when used pruned features= 357 is 0.9549450549450551
The predict accrucy of l= 18 when used pruned features= 378 is 0.9703296703296704
The predict accrucy of l= 19 when used pruned features= 399 is 0.9703296703296704
The predict accrucy of l= 20 when used pruned features= 420 is 0.9703296703296704
```

1(d)(iv) Report the confusion matrix and show the ROC and AUC for your classifier on train data. Report the parameters of your logistic regression βi's as well as the p-values associated with them.

In [487]:

```
i=0
for df in training_list:
    dataframes_final=[]
    dataframes_final=pd.DataFrame()
    for item in np.split(df,3):
        dataframes=pd.DataFrame(item)
        dataframes=dataframes.drop('# Columns: time',axis=1)
        dataframes=dataframes.reset_index(drop=True)
        dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
        dataframes=[]
    dataframes_final.columns=range(1,19,1)
    columns_list=dataframes_final.columns
    list_of_features=[]
    for cols in columns_list:
        min1=dataframes_final[cols].min()
        max1=dataframes_final[cols].max()
        mean1=statistics.mean(dataframes_final[cols])
        median1=statistics.median(dataframes_final[cols])
        std1=dataframes_final[cols].std()
        Firstquart=np.percentile(dataframes_final[cols],25)
        Thirdquart=np.percentile(dataframes_final[cols],75)
        list_of_features.append(min1)
        list_of_features.append(max1)
        list_of_features.append(round(mean1,2))
        list_of_features.append(median1)
        list_of_features.append(std1)
        list_of_features.append(round(Firstquart,2))
        list_of_features.append(round(Thirdquart,2))
    array_features=np.array([list_of_features])
    if i==0:
        print(1)
        i=i+1
        df_checking=pd.DataFrame(array_features)
    else:
        df_checking.loc[i]=list_of_features
        i=i+1
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='liblinear')
```

```
model = LogisticRegression(solver='liblinear')
rfe = RFE(model)
rfe = rfe.fit(df_finalcheck.drop(['label'],axis=1), df_finalcheck['label'])
print(rfe.support_)
print(rfe.ranking_)
f = rfe.get_support(1) #the most important features
X = df_finalcheck[df_finalcheck.columns[f]]
```

```
[ True False  True  True False False  True False  True False False False
 False False  True  True False  True  True  True False False False False
 False False False False  True  True  True  True  True  True  True False
  True False  True False False  True  True False  True  True  True  True
 False False  True False False False False False False  True False  True
 False  True  True False False False False False False False  True  True
  True  True False  True  True False  True  True False False False  True
  True  True  True  True  True  True  True False  True  True False False
 False  True  True  True False False False  True  True False  True False
 False False False False  True False False False False False False False
  True False False False False  True  True  True  True  True False  True
  True False  True  True  True False False  True  True False  True  True
  True  True False False  True  True  True False  True  True  True  True
 False False  True False  True False  True  True False False False  True
 False False False False False  True  True False  True False False False
 False  True  True False  True  True  True False  True False False False
  True False False  True  True False  True  True  True  True  True False
  True  True  True False False  True]
[  1  79   1   1  87  32   1 104   1  40  64  69  71   3   1   1  52   1
   1   1  49 100   6  66  59  83  72  48   1   1   1   1   1   1   1  90
   1   5   1  75  17   1   1  60   1   1   1   1   2  95   1  23  56  28
  80  14  11   1  70   1  21   1   1 101  45  67  77  86  89  44   1   1
   1   1  25   1   1  97   1   1   7  51   9   1   1   1   1   1   1   1
   1 103   1   1  65  41  36   1   1   1  91  24  15   1   1 102   1  42
  33  88  57  31   1  27  53  68  22  26  13  99   1  12   8  47  81   1
   1   1   1   1  55   1   1  74   1   1   1  35  18   1   1  43   1   1
   1   1  54 106   1   1   1  50   1   1   1   1  73  61   1  20   1  92
   1   1   4  34  38   1  78  62  63  30  16   1   1 105   1  37  39  46
  98   1   1  94   1   1   1  85   1  93  82  10   1  76  29   1   1  84
   1   1   1   1   1  96   1   1   1  58  19   1]
```

In [488]:

```
np.count_nonzero(rfe.ranking_==1)
```

Out[488]:

```
105
```

In [562]:

```
l=9
i=0
df_checking=pd.DataFrame()
for df in training_list:
    dataframes_final=pd.DataFrame()
    for item in np.array_split(df,l):
        dataframes=pd.DataFrame(item)
        dataframes=dataframes.drop('# Columns: time',axis=1)
        dataframes=dataframes.reset_index(drop=True)
        dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
        dataframes=[]
    dataframes_final=dataframes_final.fillna(method='ffill')
    dataframes_final.columns=range(1,(6*l)+1,1)
    columns_list=dataframes_final.columns
    list_of_features=[]
    for cols in columns_list:
        min1=dataframes_final[cols].min()
        max1=dataframes_final[cols].max()
        mean1=statistics.mean(dataframes_final[cols])
        median1=statistics.median(dataframes_final[cols])
        std1=dataframes_final[cols].std()
        Firstquart=np.percentile(dataframes_final[cols],25)
        Thirdquart=np.percentile(dataframes_final[cols],75)
        list_of_features.append(min1)
        list_of_features.append(max1)
        list_of_features.append(round(mean1,2))
```

```
        list_of_features.append(round(mean1,2))
        list_of_features.append(median1)
        list_of_features.append(std1)
        list_of_features.append(round(Firstquart,2))
        list_of_features.append(round(Thirdquart,2))
    array_features=np.array([list_of_features])
    if i==0:
        i=i+1
        df_checking=pd.DataFrame(array_features)
    else:
        df_checking.loc[i]=list_of_features
        i=i+1
df_checking['label']=list_bending[1:69]
df_train=df_checking.drop(['label'],axis=1)
df_test=df_checking['label']
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='liblinear')
rfe = RFE(model)
rfe = rfe.fit(df_train, df_test)
f = rfe.get_support(1) #the most important features
X = df_checking[df_checking.columns[f]]
LR_classifier=LogisticRegression(solver='liblinear')
training_result=LR_classifier.fit(X,df_test)
cv_score=cross_val_score(LR_classifier,X,df_test,cv=5)
accuracy=np.mean(cv_score)
pruned_features=np.count_nonzero(rfe.ranking_==1)
print('The predict accrucy of l= '+str(l)+' when used pruned features= '+str(pruned_features)+' is '+accuracy.astype('str'))
columns_list=[]
pruned_features=[]
```

```
The predict accrucy of l= 9 when used pruned features= 189 is 0.9857142857142858
```

In [563]:

```
from sklearn.metrics import confusion_matrix
training_result=LR_classifier.fit(X,df_test)
cv_score=cross_val_score(LR_classifier, X, df_test,cv=5)
result_predicted=training_result.predict(X)
tn, fp, fn, tp = confusion_matrix(result_predicted,df_test).ravel()
print ('tn:'+str(tn))
print ('fp:'+str(fp))
print ('fn:'+str(fn))
print ('tp:'+str(tp))
#confusion matrix
confusion_matrix(predict_result,df_test)
```

```
tn:60
fp:0
fn:0
tp:8
```

Out[563]:

```
array([[60,  0],
       [ 0,  8]], dtype=int64)
```

In [523]:

```
j=0
for item in X.columns:
    coefficients=str(training_result.coef_[0][item])
    name_of_feature=str(X.columns.values[item])
    print('The coefficient of ' +name_of_feature+' is ' +coefficients)
```

```
The coefficient of 0 is -0.017085613467908517
The coefficient of 18 is 0.012225894185339453
The coefficient of 29 is 0.10598863788902797
The coefficient of 34 is 0.05872166873274884
The coefficient of 36 is -0.014822668603613007
The coefficient of 41 is -0.01370728805795311
The coefficient of 47 is -0.01090906397143429
```

```
The coefficient of 64 is 0.019478348567572103
The coefficient of 70 is 0.02081666435978205
The coefficient of 71 is 0.10031301793234683
The coefficient of 72 is 0.07795290484204946
The coefficient of 73 is 0.09178027270026405
The coefficient of 74 is 0.017634731567039364
The coefficient of 75 is 0.07653543812866215
The coefficient of 80 is -0.010285936598863439
The coefficient of 92 is -0.04534598308017135
The coefficient of 98 is 0.024215084911547155
The coefficient of 99 is 0.013528815161232421
The coefficient of 111 is -0.012036661233253441
The coefficient of 112 is 0.10890167013363558
The coefficient of 114 is 0.054121733338916574
The coefficient of 120 is -0.02975071439000935
The coefficient of 121 is -0.01519828282227704
The coefficient of 122 is -0.013231918274569278
The coefficient of 126 is 0.01096207600068065
The coefficient of 134 is -0.0185343598623168
The coefficient of 139 is -0.010713817504250803
The coefficient of 140 is -0.023810616949571085
The coefficient of 142 is -0.01295838468966903
The coefficient of 155 is -0.012252768129712963
The coefficient of 162 is -0.026235899353021088
The coefficient of 163 is -0.01141855780962303 5
The coefficient of 164 is -0.01235286716594232
The coefficient of 167 is -0.013658863547635458
The coefficient of 168 is -0.04031549352868608
The coefficient of 169 is -0.06341343428200215
The coefficient of 174 is -0.01632801253907425
The coefficient of 181 is -0.011204748752381874
The coefficient of 183 is 0.03342112061640901
The coefficient of 190 is -0.0174734356111273
The coefficient of 196 is -0.011676840654105958
The coefficient of 209 is -0.011408745190334421
The coefficient of 210 is -0.019698394983182637
The coefficient of 215 is -0.05346125558152205
The coefficient of 216 is -0.06105804090933828
The coefficient of 218 is -0.028575283849769186
The coefficient of 225 is -0.012778225138412862
The coefficient of 237 is -0.015812231894798332
The coefficient of 238 is 0.05836167446988663
The coefficient of 239 is -0.011038323179066807
The coefficient of 240 is 0.010776096522726899
The coefficient of 242 is -0.012804400884673998
The coefficient of 243 is 0.015104971032699687
The coefficient of 246 is -0.03184440388941957
The coefficient of 247 is -0.010764689005203814
The coefficient of 252 is -0.01933422374295727
The coefficient of 253 is -0.0448474051335247
The coefficient of 254 is -0.040288091154335376
The coefficient of 258 is -0.05536164072708398
The coefficient of 260 is -0.036293001842780385
The coefficient of 270 is -0.012083579735610793
The coefficient of 278 is -0.011553102061639956
The coefficient of 279 is -0.020388333377874514
The coefficient of 281 is -0.0312746836856415
The coefficient of 282 is -0.013061762098077887
The coefficient of 283 is -0.011198775687382721
The coefficient of 286 is -0.02162133699680962
The coefficient of 288 is -0.03931116788068742
The coefficient of 309 is -0.026204116012565268
The coefficient of 310 is -0.0330784265001531
The coefficient of 311 is -0.04691029131320892
The coefficient of 326 is -0.018602722799664975
The coefficient of 327 is 0.037352992587061984
The coefficient of 330 is -0.03484060351005244
The coefficient of 335 is -0.013192958495113089
The coefficient of 337 is -0.029051010675695228
The coefficient of 342 is -0.013691888662382707
The coefficient of 344 is -0.02606031251841739
The coefficient of 349 is -0.011139490146661015
The coefficient of 350 is -0.026862891504531007
The coefficient of 358 is -0.03068304421675269
The coefficient of 360 is -0.013198781973441624
The coefficient of 362 is -0.01156196385763681
The coefficient of 366 is 0.03140311914777113
```

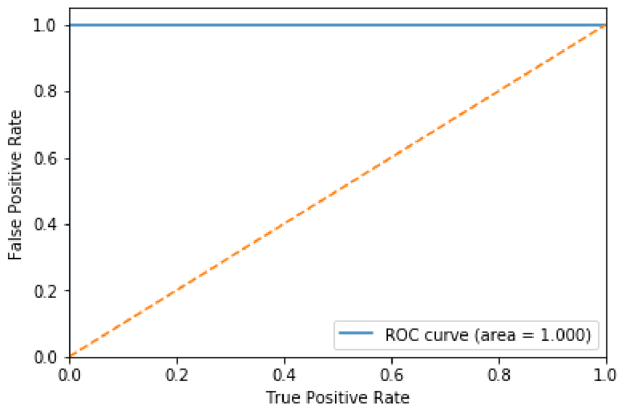
```
The coefficient of 369 is 0.041627798785992175
The coefficient of 370 is 0.02538109419442307
The coefficient of 374 is -0.011954123493335593
The coefficient of 377 is -0.01623235753720035


---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-523-3ff43c0d0953> in <module>
      1 j=0
      2 for item in X.columns:
----> 3     coefficients=str(training_result.coef_[0][item])
      4     name_of_feature=str(X.columns.values[item])
      5     print('The coefficient of ' +name_of_feature+' is ' +coefficients)

IndexError: index 190 is out of bounds for axis 0 with size 189
```

In [501]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
```

In [514]:

```
logit=roc_auc_score(df_test,predict_result)
y_scores=LR_classifier.decision_function(X)
fp_rate,tp_rate,Th=roc_curve(df_test,y_scores)
plt.figure()
plt.plot(fp_rate,tp_rate,label='ROC curve (area = %0.3f)'%logit)
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.legend(loc="lower right")
plt.plot([0, 1], [0, 1], '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.show()
```



1(d)(V) Test the classifier on the test set. Remember to break the time series in your test set into the same number of time series into which you broke your training set. Remember that the classifier has to be tested using the features extracted from the test set. Compare the accuracy on the test set with the cross-validation accuracy you obtained previously

In [528]:

```
test_list=[df1,df2,df8,df9,df14,df15,df16,df29,df30,df31,df44,df45,df46,df59,df60,df61,df74,df75,d
f76]
test_label=[1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

In [567]:

```
l=9
i=0
df_checking=pd.DataFrame()
for df in test_list:
    dataframes_final=pd.DataFrame()
    for item in np.array_split(df,l):
        dataframes=pd.DataFrame(item)
        dataframes=dataframes.drop('# Columns: time',axis=1)
        dataframes=dataframes.reset_index(drop=True)
        dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
        dataframes=[]
    dataframes_final=dataframes_final.fillna(method='ffill')
    dataframes_final.columns=range(1,(6*l)+1,1)
    columns_list=dataframes_final.columns
    list_of_features=[]
    for cols in columns_list:
        min1=dataframes_final[cols].min()
        max1=dataframes_final[cols].max()
        mean1=statistics.mean(dataframes_final[cols])
        median1=statistics.median(dataframes_final[cols])
        std1=dataframes_final[cols].std()
        Firstquart=np.percentile(dataframes_final[cols],25)
        Thirdquart=np.percentile(dataframes_final[cols],75)
        list_of_features.append(min1)
        list_of_features.append(max1)
        list_of_features.append(round(mean1,2))
        list_of_features.append(median1)
        list_of_features.append(std1)
        list_of_features.append(round(Firstquart,2))
        list_of_features.append(round(Thirdquart,2))
    array_features=np.array([list_of_features])
    if i==0:
        i=i+1
        df_checking=pd.DataFrame(array_features)
    else:
        df_checking.loc[i]=list_of_features
        i=i+1
df_checking['label']=test_label
df_train=df_checking.drop(['label'],axis=1)
df_test1=df_checking['label']
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='liblinear')
rfe = RFE(model)
rfe = rfe.fit(df_train, df_test1)
    #print(rfe.support_)
    #print(rfe.ranking_)
f = rfe.get_support(1) #the most important features
X1 = df_checking[df_checking.columns[f]]
LR_classifier=LogisticRegression(solver='liblinear')
training_result=LR_classifier.fit(X1,df_test1)
cv_score=cross_val_score(LR_classifier,X1,test_label,cv=5)
accuracy=np.mean(cv_score)
pruned_features=np.count_nonzero(rfe.ranking_==1)
print('The predict accrucy of l= '+str(l)+' when used pruned features= '+str(pruned_features)+' is
'+accuracy.astype('str'))
columns_list=[]
pruned_features=[]
```

```
The predict accrucy of l= 9 when used pruned features= 189 is 1.0
```

```
C:\Users\mohan\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:652: Warning: The lea
st populated class in y has only 4 members, which is too few. The minimum number of members in any
class cannot be less than n_splits=5.
  % (min_groups, self.n_splits)), Warning)
```

In [569]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
training_result=LR_classifier.fit(X,df_test)
cv_score=cross_val_score(LR_classifier, X,df_test,cv=5)
accuracy=np.mean(cv_score)
result_predicted=training_result.predict(X1)
tn, fp, fn, tp = confusion_matrix(result_predicted,test_label).ravel()
```

```
print ('tn:'+str(tn))
print ('fp:'+str(fp))
print ('fn:'+str(fn))
print ('tp:'+str(tp))
print(accuracy)
#confusion matrix
confusion_matrix(result_predicted,test_label)
print(classification_report(test_label,result_predicted))
```

```
tn:15
fp:4
fn:0
tp:0
0.9857142857142858
              precision    recall  f1-score   support

           0       0.79      1.00      0.88        15
           1       0.00      0.00      0.00         4

   micro avg       0.79      0.79      0.79        19
   macro avg       0.39      0.50      0.44        19
weighted avg       0.62      0.79      0.70        19
```

```
C:\Users\mohan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\mohan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\mohan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Here the accuracy on the test set is 79% when compared to 98.5% of that on the training set

1(d)(vi) Do your classes seem to be well-separated to cause instability in calculating logistic regression parameters?

No the classes seem to be well separated as the bending classes are very few when compared to the other class causing the class imbalance. Therefore there is instability in calculating the logistic regression parameters. So the over sampling technique is employed to overcome this problem

In [59]:

```
from sklearn.metrics import recall_score
from imblearn.over_sampling import SMOTE
from sklearn.metrics import confusion_matrix
```

In [33]:

```
df_final_features_smote=pd.DataFrame(columns=['min1','max1','mean1','median1','std1','1stquart1','3rdquart1','min2','max2','mean2','median2','std2','1stquart2','3rdquart2','min3','max3','mean3','median3','std3','1stquart3','3rdquart3','min4','max4','mean4','median4','std4','1stquart4','3rdquart4','min5','max5','mean5','median5','std5','1stquart5','3rdquart5','min6','max6','mean6','median6','std6','1stquart6','3rdquart6'])
```

In [34]:

```
training_list=[df3,df4,df5,df6,df7,df10,df11,df12,df13,df17,df18,df19,df20,df21,df22,df23,df24,df25,df26,df27,df28,df32,df33,df34,df35,df36,df37,df38,df39,df40,df41,df42,df43,df47,df48,df49,df50,df51,df52,df53,df54,df55,df56,df57,df58,df62,df63,df64,df65,df66,df67,df68,df69,df70,df71,df72,df73,df77,df78,df79,df80,df81,df82,df83,df84,df85,df86,df87,df88]
```

In [36]:

```
i=0
for df in training_list:
    list_m=[]
    for cols in df.columns[1:7]:
        min1=df[cols].min()
        max1=df[cols].max()
        mean1=statistics.mean(df[cols])
        median1=statistics.median(df[cols])
        std1=df[cols].std()
        Firstquart=np.percentile(df[cols],25)
        Thirdquart=np.percentile(df[cols],75)
        list_m.append(min1)
        list_m.append(max1)
        list_m.append(round(mean1,2))
        list_m.append(median1)
        list_m.append(std1)
        list_m.append(round(Firstquart,2))
        list_m.append(round(Thirdquart,2))
    i=i+1
    df_final_features_smote.loc[i]=list_m
```

In [43]:

```
label_list=[1]*9+[0]*60
df_final_features_smote['label']=label_list
df_final_features_smote
```

Out[43]:

	min1	max1	mean1	median1	std1	1stquart1	3rdquart1	min2	max2	mean2	...	1stquart5	3rdquart5	min6	max6	mean6
1	35.00	47.40	43.95	44.330	1.558835	43.00	45.00	0.0	1.70	0.43	...	35.36	36.50	0.00	1.79	0.49
2	33.00	47.75	42.18	43.500	3.670666	39.15	45.00	0.0	3.00	0.70	...	30.46	36.33	0.00	2.18	0.61
3	33.00	45.75	41.68	41.750	2.243490	41.33	42.75	0.0	2.83	0.54	...	28.46	31.25	0.00	1.79	0.38
4	37.00	48.00	43.45	43.250	1.386098	42.50	45.00	0.0	1.58	0.38	...	22.25	24.00	0.00	5.26	0.68
5	36.25	48.00	43.97	44.500	1.618364	43.31	44.67	0.0	1.50	0.41	...	20.50	23.75	0.00	2.96	0.56
6	21.00	50.00	32.59	33.000	6.238143	26.19	34.50	0.0	9.90	0.52	...	17.67	23.50	0.00	13.61	1.16
7	27.50	33.00	29.88	30.000	1.153837	29.00	30.27	0.0	1.00	0.26	...	17.00	19.00	0.00	6.40	0.70
8	19.00	45.50	30.94	29.000	7.684146	26.75	38.00	0.0	6.40	0.47	...	15.00	20.81	0.00	6.73	1.11
9	25.00	47.50	31.06	29.710	4.829794	27.50	31.81	0.0	6.38	0.41	...	9.00	18.31	0.00	4.92	1.10
10	19.00	44.00	36.23	36.000	3.528617	34.00	39.00	0.0	12.28	2.83	...	14.00	18.06	0.00	9.98	3.48
11	26.50	44.33	36.69	36.000	3.529404	34.25	39.37	0.0	12.89	2.97	...	14.67	18.50	0.00	8.19	3.07
12	25.33	45.00	37.11	36.250	3.710385	34.50	40.25	0.0	10.84	2.73	...	14.75	18.50	0.00	9.50	3.08
13	26.75	44.75	36.86	36.330	3.555787	34.50	39.75	0.0	11.68	2.76	...	15.00	18.67	0.00	8.81	2.77
14	26.25	44.25	36.96	36.290	3.434863	34.50	40.25	0.0	8.64	2.42	...	14.00	18.25	0.00	8.34	2.93
15	27.75	44.67	37.14	36.330	3.758904	34.00	40.50	0.0	10.76	2.42	...	15.00	18.75	0.00	8.75	2.82
16	27.00	45.00	36.82	36.000	3.900459	33.75	40.25	0.0	10.47	2.60	...	15.50	19.27	0.00	8.99	2.89
17	27.00	44.33	36.54	36.000	4.018922	33.25	39.81	0.0	10.43	2.85	...	15.00	19.50	0.00	9.18	3.23
18	18.50	44.25	35.75	36.000	4.614802	33.00	39.33	0.0	12.60	3.33	...	14.00	18.06	0.00	9.39	3.07
19	19.00	43.75	35.88	36.000	4.614878	33.00	39.50	0.0	11.20	3.41	...	14.75	19.69	0.00	8.50	3.09
20	23.33	43.50	36.24	36.750	3.822016	33.46	39.25	0.0	9.71	2.74	...	15.75	21.00	0.00	11.15	3.53
21	24.25	45.00	37.18	36.250	3.581301	34.50	40.25	0.0	8.58	2.37	...	17.95	21.75	0.00	9.34	2.92
22	34.00	51.00	42.71	40.500	3.537476	40.25	48.00	0.0	4.85	0.52	...	1.00	8.00	0.00	4.97	0.55
23	39.00	41.00	39.67	39.500	0.280158	39.50	39.75	0.0	1.00	0.58	...	1.63	9.33	0.00	3.49	0.64
24	0.00	40.67	39.51	39.500	1.817498	39.50	39.67	0.0	1.00	0.50	...	11.33	13.00	0.00	3.19	0.62
25	39.00	40.00	39.43	39.500	0.208558	39.33	39.50	0.0	1.00	0.42	...	9.00	12.33	0.00	4.06	0.58
26	39.00	40.00	39.35	39.330	0.231405	39.25	39.50	0.0	0.50	0.37	...	15.75	17.67	0.00	3.50	0.59
27	39.00	56.25	47.33	42.670	5.961280	42.00	54.00	0.0	8.49	0.27	...	11.75	18.00	0.00	5.72	0.77
28	23.50	30.00	27.72	27.500	1.442253	27.00	29.00	0.0	1.79	0.36	...	5.50	10.75	0.00	4.50	0.74

	min1	max1	mean1	median1	std1	1stquart1	3rdquart1	min2	max2	mean2	...	1stquart5	3rdquart5	min6	max6	mean6
29	48.00	48.25	48.00	48.000	0.032038	48.00	48.00	0.0	0.43	0.04	...	4.67	10.00	0.00	2.50	0.64
30	39.00	41.00	39.67	39.500	0.280158	39.50	39.75	0.0	1.00	0.58	...	1.63	9.33	0.00	3.49	0.64
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
40	35.25	48.50	40.22	39.250	2.741217	37.75	42.50	0.0	3.28	0.62	...	11.67	19.75	0.00	6.36	1.06
41	28.50	48.25	43.88	45.250	3.198894	42.00	46.50	0.0	3.28	0.52	...	10.50	19.25	0.00	7.00	1.35
42	39.50	45.00	42.11	42.000	1.122245	41.50	42.00	0.0	1.09	0.34	...	9.00	17.25	0.00	6.36	0.96
43	39.67	44.75	42.28	41.500	1.356149	41.50	44.33	0.0	1.00	0.50	...	8.50	18.25	0.00	7.85	0.87
44	40.00	44.67	42.36	42.000	1.017372	41.50	43.25	0.0	1.00	0.48	...	9.75	22.00	0.00	4.64	0.72
45	29.25	46.00	42.73	43.250	2.046362	41.33	44.50	0.0	4.72	0.56	...	13.73	19.00	0.00	5.10	0.89
46	30.00	46.67	42.65	42.750	2.395338	41.50	45.00	0.0	2.95	0.40	...	10.63	14.25	0.00	4.64	0.92
47	36.00	47.50	43.72	45.000	2.384105	43.00	45.00	0.0	1.92	0.37	...	11.31	15.54	0.00	6.18	1.04
48	34.50	47.75	44.47	45.000	1.772553	45.00	45.25	0.0	2.18	0.29	...	12.00	14.81	0.00	4.32	0.93
49	35.50	48.00	46.22	46.000	1.748315	45.25	48.00	0.0	4.50	0.31	...	12.00	15.25	0.00	6.00	0.88
50	29.75	48.00	46.93	47.500	1.832665	47.24	47.75	0.0	4.60	0.43	...	11.67	15.50	0.00	6.58	0.99
51	36.33	47.67	45.40	45.500	1.328121	45.00	46.33	0.0	1.66	0.46	...	11.25	14.50	0.00	4.50	0.80
52	36.00	45.80	42.42	42.670	2.520129	41.33	44.62	0.0	2.12	0.46	...	7.63	12.00	0.00	6.65	1.23
53	37.00	48.25	42.52	42.500	2.195751	41.00	44.50	0.0	2.12	0.44	...	12.63	17.50	0.00	6.85	0.98
54	36.25	45.50	42.96	42.670	1.500878	42.00	44.33	0.0	2.60	0.35	...	14.00	16.69	0.00	4.00	0.75
55	36.00	47.33	42.67	43.670	2.384170	40.00	44.75	0.0	2.17	0.42	...	12.75	16.50	0.00	3.77	0.70
56	36.25	45.75	43.19	44.750	2.491162	39.75	45.00	0.0	2.83	0.27	...	16.50	21.00	0.00	3.83	0.65
57	36.00	47.33	44.44	45.000	2.417797	44.63	45.75	0.0	4.50	0.35	...	11.00	14.67	0.00	5.91	1.16
58	18.00	46.00	35.19	36.000	4.751868	32.00	38.75	0.0	16.20	4.32	...	14.25	18.50	0.00	8.50	3.24
59	20.75	46.25	34.76	35.290	4.742208	31.67	38.25	0.0	12.68	4.22	...	14.25	18.33	0.00	9.39	3.29
60	21.50	51.00	34.94	35.500	4.645944	32.00	38.06	0.0	12.21	4.12	...	14.24	18.25	0.00	10.21	3.28
61	18.33	47.67	34.33	34.750	4.948770	31.25	38.00	0.0	12.48	4.40	...	13.75	18.00	0.00	8.01	3.26
62	18.33	45.75	34.60	35.125	4.731790	31.50	38.00	0.0	15.37	4.40	...	14.00	18.25	0.00	8.86	3.29
63	15.50	43.67	34.23	34.750	4.441798	31.25	37.25	0.0	17.24	4.35	...	14.33	18.25	0.00	9.42	3.48
64	21.50	51.25	34.25	35.000	4.940741	30.94	37.75	0.0	13.55	4.46	...	13.75	18.00	0.00	8.32	3.50
65	19.50	45.33	33.59	34.250	4.650935	30.25	37.00	0.0	14.67	4.58	...	13.73	18.25	0.00	8.32	3.26
66	19.75	45.50	34.32	35.250	4.752477	31.00	38.00	0.0	13.47	4.46	...	13.50	17.75	0.00	9.67	3.43
67	19.50	46.00	34.55	35.250	4.842294	31.25	37.81	0.0	12.47	4.37	...	14.00	17.75	0.00	10.00	3.34
68	23.50	46.25	34.87	35.250	4.531720	31.75	38.25	0.0	14.82	4.38	...	13.75	18.00	0.00	9.51	3.42
69	19.25	44.00	34.47	35.000	4.796705	31.25	38.00	0.0	13.86	4.36	...	13.73	17.75	0.43	9.00	3.34

69 rows × 43 columns

In [46]:

```
from sklearn.model_selection import train_test_split
```

In [45]:

```
training_features=df_final_features_smote.drop(['label'],axis=1)
testing_features=label_list
```

In [47]:

```
X_train, X_test, y_train, y_test = train_test_split(training_features, testing_features, test_size=
0.33, random_state=42)
```

In [49]:

```
smote = SMOTE(random_state=12, ratio = 1.0)
result_X_train, result_y_train = sm.fit_sample(X_train, y_train)
```

In [63]:

```
LR_classifier=LogisticRegression(solver='liblinear')
LR_classifier.fit(result_X_train,result_y_train)
print('The accuracy is as follows')
print(LR_classifier.score(X_test,y_test))
print(recall_score(y_test, LR_classifier.predict(X_test)))
```

```
The accuracy is as follows
0.9565217391304348
1.0
```

In [61]:

```
print(LR_classifier.score(X_test,y_test))
print(recall_score(y_test, LR_classifier.predict(X_test)))
```

Out[61]:

```
array([[19,  0],
       [ 1,  3]], dtype=int64)
```

As we could see that there is an increase in the accuracy of the classifier after over sampling

1(e)(i)

Repeat 1(d)iii using L1-penalized logistic regression, i.e. instead of using pvalues for variable selection, use L1 regularization. Note that in this problem,you have to cross-validate for both l, the number of time series into which you break each of your instances, and λ, the weight of L1 penalty in your logistic regression objective function (or C, the budget). Packages usually perform cross-validation for λ automatically.

In [596]:

```
i=0
res=0
current_list=[]
for l in range(1,21,1):
    i=0
    df_checking=pd.DataFrame()
    for df in training_list:
        dataframes_final=pd.DataFrame()
        for item in np.array_split(df,l):
            dataframes=pd.DataFrame(item)
            dataframes=dataframes.drop('# Columns: time',axis=1)
            dataframes=dataframes.reset_index(drop=True)
            dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
            dataframes=[]
        dataframes_final=dataframes_final.fillna(method='ffill')
        dataframes_final.columns=range(1,(6*l)+1,1)
        columns_list=dataframes_final.columns
        list_of_features=[]
        for cols in columns_list:
            min1=dataframes_final[cols].min()
            max1=dataframes_final[cols].max()
            mean1=statistics.mean(dataframes_final[cols])
            median1=statistics.median(dataframes_final[cols])
            std1=dataframes_final[cols].std()
            Firstquart=np.percentile(dataframes_final[cols],25)
            Thirdquart=np.percentile(dataframes_final[cols],75)
            list_of_features.append(min1)
            list_of_features.append(max1)
            list_of_features.append(round(mean1,2))
            list_of_features.append(median1)
```

```
            list_of_features.append(std1)
            list_of_features.append(round(Firstquart,2))
            list_of_features.append(round(Thirdquart,2))
        array_features=np.array([list_of_features])
        if i==0:
            i=i+1
            df_checking=pd.DataFrame(array_features)
        else:
            df_checking.loc[i]=list_of_features
            i=i+1
    df_checking['label']=list_bending[1:69]
    df_train=df_checking.drop(['label'],axis=1)
    df_test=df_checking['label']
    for j in np.arange(1,100):
        LR_classifier=LogisticRegression(solver='liblinear',penalty='l1',C=1/j)
        training_result=LR_classifier.fit(df_train,df_test)
        cv_score=cross_val_score(LR_classifier,df_train,df_test,cv=5)
        accuracy=np.mean(cv_score)
        #print('The predict accrucy of l= '+str(l)+' is '+accuracy.astype('str'))
        columns_list=[]
        if accuracy>res:
                res=accuracy
                best_k_number=l
                best_lambda=j
```

In [603]:

```
print('The predict accrucy of l= '+str(res)+'')
print('The best l is '+str(best_k_number)+'')
print('The best lambda is '+str(best_lambda)+'')
```

```
The predict accrucy of l= 0.9857142857142858
The best l is 1
The best lambda is 2
```

1(e)(ii) Compare the L1-penalized with variable selection using p-values. Which one performs better? Which one is easier to implement?

From The above result, we can see that L1 Regualizer is much easier to implement because it can Automatically select the feature where as the latter requires feature selection. Also The best Accuracy score is 98.57% which is higher than the RTF best Accuracy is 95%

1(f)(i)

Find the best l in the same way as you found it in 1(e)i to build an L1-penalized multinomial regression model to classify all activities in your training set. Report your test error.

In [645]:

```
from warnings import filterwarnings
filterwarnings('ignore')
accuracy=[]
list_accuracy=[]
i=0
res=0
current_list=[]
for l in range(1,21,1):
    i=0
    c=0.01
    df_checking=pd.DataFrame()
    for df in training_list:
        dataframes_final=pd.DataFrame()
        for item in np.array_split(df,l):
            dataframes=pd.DataFrame(item)
```

```
            dataframes=pd.DataFrame(item)
            dataframes=dataframes.drop('# Columns: time',axis=1)
            dataframes=dataframes.reset_index(drop=True)
            dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
            dataframes=[]
        dataframes_final=dataframes_final.fillna(method='ffill')
        dataframes_final.columns=range(1,(6*l)+1,1)
        columns_list=dataframes_final.columns
        list_of_features=[]
        for cols in columns_list:
            min1=dataframes_final[cols].min()
            max1=dataframes_final[cols].max()
            mean1=statistics.mean(dataframes_final[cols])
            median1=statistics.median(dataframes_final[cols])
            std1=dataframes_final[cols].std()
            Firstquart=np.percentile(dataframes_final[cols],25)
            Thirdquart=np.percentile(dataframes_final[cols],75)
            list_of_features.append(min1)
            list_of_features.append(max1)
            list_of_features.append(round(mean1,2))
            list_of_features.append(median1)
            list_of_features.append(std1)
            list_of_features.append(round(Firstquart,2))
            list_of_features.append(round(Thirdquart,2))
        array_features=np.array([list_of_features])
        if i==0:
            i=i+1
            df_checking=pd.DataFrame(array_features)
        else:
            df_checking.loc[i]=list_of_features
            i=i+1
    df_checking['label']=list_multinomial
    df_train=df_checking.drop(['label'],axis=1)
    df_test=df_checking['label']
    while c<101:
        LR_classifier=LogisticRegression(C=c,multi_class='multinomial',solver='saga',penalty='l1')
        training_result=LR_classifier.fit(df_train,df_test)
        cv_score=cross_val_score(LR_classifier,df_train,df_test,cv=5)
        accuracy=np.mean(cv_score)
        list_accuracy.append(accuracy)
        print('The predict accuracy of l= '+str(l)+' c= '+str(c)+' is '+accuracy.astype('str'))
        columns_list=[]
        c=c*10
        if accuracy>res:
                res=accuracy
                best_l_number=l
                best_c=c/10
```

```
The predict accuracy of l= 1 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 1 c= 0.1 is 0.7703208556149732
The predict accuracy of l= 1 c= 1.0 is 0.8650623885918003
The predict accuracy of l= 1 c= 10.0 is 0.8768270944741532
The predict accuracy of l= 1 c= 100.0 is 0.8768270944741532
The predict accuracy of l= 2 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 2 c= 0.1 is 0.7168449197860963
The predict accuracy of l= 2 c= 1.0 is 0.7972370766488414
The predict accuracy of l= 2 c= 10.0 is 0.8090017825311943
The predict accuracy of l= 2 c= 100.0 is 0.8090017825311943
The predict accuracy of l= 3 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 3 c= 0.1 is 0.710427807486631
The predict accuracy of l= 3 c= 1.0 is 0.7820855614973262
The predict accuracy of l= 3 c= 10.0 is 0.7820855614973262
The predict accuracy of l= 3 c= 100.0 is 0.7820855614973262
The predict accuracy of l= 4 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 4 c= 0.1 is 0.707397504456328
The predict accuracy of l= 4 c= 1.0 is 0.8256684491978609
The predict accuracy of l= 4 c= 10.0 is 0.8256684491978609
The predict accuracy of l= 4 c= 100.0 is 0.8256684491978609
The predict accuracy of l= 5 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 5 c= 0.1 is 0.7142602495543672
The predict accuracy of l= 5 c= 1.0 is 0.7972370766488414
The predict accuracy of l= 5 c= 10.0 is 0.8090017825311943
The predict accuracy of l= 5 c= 100.0 is 0.8207664884135472
The predict accuracy of l= 6 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 6 c= 0.1 is 0.7608734402852051
The predict accuracy of l= 6 c= 1.0 is 0.802584670231729
The predict accuracy of l= 6 c= 10.0 is 0.8207664884135472
```

```
The predict accuracy of l= 6 c= 100.0 is 0.8207664884135472
The predict accuracy of l= 7 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 7 c= 0.1 is 0.7275401069518718
The predict accuracy of l= 7 c= 1.0 is 0.7393048128342247
The predict accuracy of l= 7 c= 10.0 is 0.7692513368983958
The predict accuracy of l= 7 c= 100.0 is 0.8056149732620319
The predict accuracy of l= 8 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 8 c= 0.1 is 0.7142602495543672
The predict accuracy of l= 8 c= 1.0 is 0.7854723707664883
The predict accuracy of l= 8 c= 10.0 is 0.8271836007130124
The predict accuracy of l= 8 c= 100.0 is 0.8271836007130124
The predict accuracy of l= 9 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 9 c= 0.1 is 0.7377896613190732
The predict accuracy of l= 9 c= 1.0 is 0.7923351158645277
The predict accuracy of l= 9 c= 10.0 is 0.8105169340463458
The predict accuracy of l= 9 c= 100.0 is 0.8105169340463458
The predict accuracy of l= 10 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 10 c= 0.1 is 0.7275401069518718
The predict accuracy of l= 10 c= 1.0 is 0.7623885918003566
The predict accuracy of l= 10 c= 10.0 is 0.8040998217468805
The predict accuracy of l= 10 c= 100.0 is 0.8222816399286987
The predict accuracy of l= 11 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 11 c= 0.1 is 0.7260249554367201
The predict accuracy of l= 11 c= 1.0 is 0.7805704099821746
The predict accuracy of l= 11 c= 10.0 is 0.7923351158645277
The predict accuracy of l= 11 c= 100.0 is 0.8090017825311943
The predict accuracy of l= 12 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 12 c= 0.1 is 0.7211229946524064
The predict accuracy of l= 12 c= 1.0 is 0.7790552584670232
The predict accuracy of l= 12 c= 10.0 is 0.7972370766488414
The predict accuracy of l= 12 c= 100.0 is 0.7972370766488414
The predict accuracy of l= 13 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 13 c= 0.1 is 0.7442067736185383
The predict accuracy of l= 13 c= 1.0 is 0.74572192513369
The predict accuracy of l= 13 c= 10.0 is 0.7756684491978609
The predict accuracy of l= 13 c= 100.0 is 0.7756684491978609
The predict accuracy of l= 14 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 14 c= 0.1 is 0.7142602495543672
The predict accuracy of l= 14 c= 1.0 is 0.7506238859180037
The predict accuracy of l= 14 c= 10.0 is 0.7623885918003566
The predict accuracy of l= 14 c= 100.0 is 0.7623885918003566
The predict accuracy of l= 15 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 15 c= 0.1 is 0.7260249554367201
The predict accuracy of l= 15 c= 1.0 is 0.7805704099821746
The predict accuracy of l= 15 c= 10.0 is 0.7805704099821746
The predict accuracy of l= 15 c= 100.0 is 0.7923351158645277
The predict accuracy of l= 16 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 16 c= 0.1 is 0.6975935828877006
The predict accuracy of l= 16 c= 1.0 is 0.7623885918003566
The predict accuracy of l= 16 c= 10.0 is 0.7923351158645277
The predict accuracy of l= 16 c= 100.0 is 0.7923351158645277
The predict accuracy of l= 17 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 17 c= 0.1 is 0.7260249554367201
The predict accuracy of l= 17 c= 1.0 is 0.7324420677361855
The predict accuracy of l= 17 c= 10.0 is 0.7805704099821746
The predict accuracy of l= 17 c= 100.0 is 0.7805704099821746
The predict accuracy of l= 18 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 18 c= 0.1 is 0.6858288770053476
The predict accuracy of l= 18 c= 1.0 is 0.749108734402852
The predict accuracy of l= 18 c= 10.0 is 0.7672905525846702
The predict accuracy of l= 18 c= 100.0 is 0.7790552584670232
The predict accuracy of l= 19 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 19 c= 0.1 is 0.6794117647058824
The predict accuracy of l= 19 c= 1.0 is 0.7142602495543672
The predict accuracy of l= 19 c= 10.0 is 0.7506238859180036
The predict accuracy of l= 19 c= 100.0 is 0.7623885918003565
The predict accuracy of l= 20 c= 0.01 is 0.17664884135472372
The predict accuracy of l= 20 c= 0.1 is 0.7093582887700535
The predict accuracy of l= 20 c= 1.0 is 0.7672905525846702
The predict accuracy of l= 20 c= 10.0 is 0.7790552584670232
The predict accuracy of l= 20 c= 100.0 is 0.7790552584670232
```

In [646]:

```
print('The predict accrucy of l= '+str(res)+'')
print('The best l is '+str(best_l_number)+'')
```

```
print('The best c is '+str(best_c)+'')
```

```
The predict accrucy of l= 0.8768270944741532
The best l is 1
The best c is 10.0
```

In [641]:

```
list_multinomial=[1]*8+[2]*12+[3]*12+[4]*12+[5]*12+[6]*12
```

In [659]:

```
l=1
i=0
df_checking=pd.DataFrame()
for df in training_list:
    dataframes_final=pd.DataFrame()
    for item in np.array_split(df,l):
        dataframes=pd.DataFrame(item)
        dataframes=dataframes.drop('# Columns: time',axis=1)
        dataframes=dataframes.reset_index(drop=True)
        dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
        dataframes=[]
    dataframes_final=dataframes_final.fillna(method='ffill')
    dataframes_final.columns=range(1,(6*l)+1,1)
    columns_list=dataframes_final.columns
    list_of_features=[]
    for cols in columns_list:
        min1=dataframes_final[cols].min()
        max1=dataframes_final[cols].max()
        mean1=statistics.mean(dataframes_final[cols])
        median1=statistics.median(dataframes_final[cols])
        std1=dataframes_final[cols].std()
        Firstquart=np.percentile(dataframes_final[cols],25)
        Thirdquart=np.percentile(dataframes_final[cols],75)
        list_of_features.append(min1)
        list_of_features.append(max1)
        list_of_features.append(round(mean1,2))
        list_of_features.append(median1)
        list_of_features.append(std1)
        list_of_features.append(round(Firstquart,2))
        list_of_features.append(round(Thirdquart,2))
    array_features=np.array([list_of_features])
    if i==0:
        i=i+1
        df_checking=pd.DataFrame(array_features)
    else:
        df_checking.loc[i]=list_of_features
        i=i+1
df_checking['label']=list_multinomial
df_train1=df_checking.drop(['label'],axis=1)
df_test1=df_checking['label']
```

In [649]:

```
l=1
c=10.0
LR_multiclassifier=LogisticRegression(C=c,multi_class='multinomial',solver='saga',penalty='l1')
training_result=LR_classifier.fit(df_train1,df_test1)
cv_score=cross_val_score(LR_classifier,df_train1,df_test1,cv=5)
accuracy=np.mean(cv_score)
prediction_result=training_result.predict(df_train1)
print('The predict accuracy of l= '+str(l)+' c= '+str(c)+' is '+accuracy.astype('str'))
```

```
The predict accuracy of l= 1 c= 10.0 is 0.8768270944741532
```

In [650]:

```
confusion_matrix(prediction_result,df_test1)
```

Out[650]:

```
array([[ 8,  0,  0,  0,  0,  0],
       [ 0, 12,  0,  0,  0,  0],
       [ 0,  0, 12,  0,  0,  0],
       [ 0,  0,  0, 12,  2,  0],
       [ 0,  0,  0,  0, 10,  0],
       [ 0,  0,  0,  0,  0, 12]], dtype=int64)
```

Repeat 1(f)i using a Naıve Bayes' classifier. Use both Gaussian and Multinomial priors and compare the results

1(F)(ii)Gaussian Naive Bayes

In [651]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
```

In [657]:

```
from warnings import filterwarnings
filterwarnings('ignore')
accuracy=[]
list_accuracy=[]
i=0
res=0
current_list=[]
for l in range(1,21,1):
    i=0
    c=0.01
    df_checking=pd.DataFrame()
    for df in training_list:
        dataframes_final=pd.DataFrame()
        for item in np.array_split(df,l):
            dataframes=pd.DataFrame(item)
            dataframes=dataframes.drop('# Columns: time',axis=1)
            dataframes=dataframes.reset_index(drop=True)
            dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
            dataframes=[]
        dataframes_final=dataframes_final.fillna(method='ffill')
        dataframes_final.columns=range(1,(6*l)+1,1)
        columns_list=dataframes_final.columns
        list_of_features=[]
        for cols in columns_list:
            min1=dataframes_final[cols].min()
            max1=dataframes_final[cols].max()
            mean1=statistics.mean(dataframes_final[cols])
            median1=statistics.median(dataframes_final[cols])
            std1=dataframes_final[cols].std()
            Firstquart=np.percentile(dataframes_final[cols],25)
            Thirdquart=np.percentile(dataframes_final[cols],75)
            list_of_features.append(min1)
            list_of_features.append(max1)
            list_of_features.append(round(mean1,2))
            list_of_features.append(median1)
            list_of_features.append(std1)
            list_of_features.append(round(Firstquart,2))
            list_of_features.append(round(Thirdquart,2))
        array_features=np.array([list_of_features])
        if i==0:
            i=i+1
            df_checking=pd.DataFrame(array_features)
        else:
            df_checking.loc[i]=list_of_features
            i=i+1
    df_checking['label']=list_multinomial
    df_train1=df_checking.drop(['label'],axis=1)
    df_test1=df_checking['label']
    GNB_classifier=GaussianNB()
    GNB_classifier.fit(df_train1,df_test1)
    accuracy=GNB_classifier.score(df_train1,df_test1)
```

```
    list_accuracy.append(accuracy)
    print('The predict accuracy of l= '+str(l)+' is '+accuracy.astype('str'))
    columns_list=[]
    if accuracy>res:
        res=accuracy
        best_l_number=l
```

```
The predict accuracy of l= 1 is 1.0
The predict accuracy of l= 2 is 1.0
The predict accuracy of l= 3 is 0.9852941176470589
The predict accuracy of l= 4 is 1.0
The predict accuracy of l= 5 is 0.9852941176470589
The predict accuracy of l= 6 is 0.9705882352941176
The predict accuracy of l= 7 is 1.0
The predict accuracy of l= 8 is 0.9705882352941176
The predict accuracy of l= 9 is 1.0
The predict accuracy of l= 10 is 1.0
The predict accuracy of l= 11 is 1.0
The predict accuracy of l= 12 is 1.0
The predict accuracy of l= 13 is 1.0
The predict accuracy of l= 14 is 1.0
The predict accuracy of l= 15 is 1.0
The predict accuracy of l= 16 is 1.0
The predict accuracy of l= 17 is 1.0
The predict accuracy of l= 18 is 1.0
The predict accuracy of l= 19 is 1.0
The predict accuracy of l= 20 is 1.0
```

In [658]:

```
print('The predict accrucy of l= '+str(res)+'')
print('The best l is '+str(best_l_number)+'')
```

```
The predict accrucy of l= 1.0
The best l is 1
```

In [662]:

```
GNB_classifier=GaussianNB()
GNB_classifier.fit(df_train1,df_test1)
prediction_result=GNB_classifier.predict(df_train1)
confusion_matrix(prediction_result, df_test1)
```

Out[662]:

```
array([[ 8,  0,  0,  0,  0,  0],
       [ 0, 12,  0,  0,  0,  0],
       [ 0,  0, 12,  0,  0,  0],
       [ 0,  0,  0, 12,  0,  0],
       [ 0,  0,  0,  0, 12,  0],
       [ 0,  0,  0,  0,  0, 12]], dtype=int64)
```

Multinomial Naive Bayes

In [665]:

```
from warnings import filterwarnings
filterwarnings('ignore')
accuracy=[]
list_accuracy=[]
i=0
res=0
current_list=[]
for l in range(1,21,1):
    i=0
    c=0.01
    df_checking=pd.DataFrame()
    for df in training_list:
        dataframes_final=pd.DataFrame()
        for item in np.array_split(df,l):
            dataframes=pd.DataFrame(item)
```

```python
            dataframes=pd.DataFrame(item)
            dataframes=dataframes.drop('# Columns: time',axis=1)
            dataframes=dataframes.reset_index(drop=True)
            dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
            dataframes=[]
        dataframes_final=dataframes_final.fillna(method='ffill')
        dataframes_final.columns=range(1,(6*l)+1,1)
        columns_list=dataframes_final.columns
        list_of_features=[]
        for cols in columns_list:
            min1=dataframes_final[cols].min()
            max1=dataframes_final[cols].max()
            mean1=statistics.mean(dataframes_final[cols])
            median1=statistics.median(dataframes_final[cols])
            std1=dataframes_final[cols].std()
            Firstquart=np.percentile(dataframes_final[cols],25)
            Thirdquart=np.percentile(dataframes_final[cols],75)
            list_of_features.append(min1)
            list_of_features.append(max1)
            list_of_features.append(round(mean1,2))
            list_of_features.append(median1)
            list_of_features.append(std1)
            list_of_features.append(round(Firstquart,2))
            list_of_features.append(round(Thirdquart,2))
        array_features=np.array([list_of_features])
        if i==0:
            i=i+1
            df_checking=pd.DataFrame(array_features)
        else:
            df_checking.loc[i]=list_of_features
            i=i+1
    df_checking['label']=list_multinomial
    df_train1=df_checking.drop(['label'],axis=1)
    df_test1=df_checking['label']
    MNB_classifier=MultinomialNB()
    MNB_classifier.fit(df_train1,df_test1)
    accuracy=MNB_classifier.score(df_train1,df_test1)
    list_accuracy.append(accuracy)
    print('The predict accuracy of l= '+str(l)+' is '+accuracy.astype('str'))
    columns_list=[]
    if accuracy>res:
        res=accuracy
        best_l_number=l
```

```
The predict accuracy of l= 1 is 0.9264705882352942
The predict accuracy of l= 2 is 0.8823529411764706
The predict accuracy of l= 3 is 0.8970588235294118
The predict accuracy of l= 4 is 0.8970588235294118
The predict accuracy of l= 5 is 0.9264705882352942
The predict accuracy of l= 6 is 0.9558823529411765
The predict accuracy of l= 7 is 0.9558823529411765
The predict accuracy of l= 8 is 0.9558823529411765
The predict accuracy of l= 9 is 0.9558823529411765
The predict accuracy of l= 10 is 0.9558823529411765
The predict accuracy of l= 11 is 0.9558823529411765
The predict accuracy of l= 12 is 0.9558823529411765
The predict accuracy of l= 13 is 0.9558823529411765
The predict accuracy of l= 14 is 0.9558823529411765
The predict accuracy of l= 15 is 0.9558823529411765
The predict accuracy of l= 16 is 0.9558823529411765
The predict accuracy of l= 17 is 0.9558823529411765
The predict accuracy of l= 18 is 0.9558823529411765
The predict accuracy of l= 19 is 0.9558823529411765
The predict accuracy of l= 20 is 0.9558823529411765
```

In [666]:

```python
print('The predict accrucy of l= '+str(res)+'')
print('The best l is '+str(best_l_number)+'')
```

```
The predict accrucy of l= 0.9558823529411765
The best l is 6
```

In [667]:

```
l=6
i=0
df_checking=pd.DataFrame()
for df in training_list:
    dataframes_final=pd.DataFrame()
    for item in np.array_split(df,l):
        dataframes=pd.DataFrame(item)
        dataframes=dataframes.drop('# Columns: time',axis=1)
        dataframes=dataframes.reset_index(drop=True)
        dataframes_final=pd.concat([dataframes_final,dataframes],axis=1)
        dataframes=[]
    dataframes_final=dataframes_final.fillna(method='ffill')
    dataframes_final.columns=range(1,(6*l)+1,1)
    columns_list=dataframes_final.columns
    list_of_features=[]
    for cols in columns_list:
        min1=dataframes_final[cols].min()
        max1=dataframes_final[cols].max()
        mean1=statistics.mean(dataframes_final[cols])
        median1=statistics.median(dataframes_final[cols])
        std1=dataframes_final[cols].std()
        Firstquart=np.percentile(dataframes_final[cols],25)
        Thirdquart=np.percentile(dataframes_final[cols],75)
        list_of_features.append(min1)
        list_of_features.append(max1)
        list_of_features.append(round(mean1,2))
        list_of_features.append(median1)
        list_of_features.append(std1)
        list_of_features.append(round(Firstquart,2))
        list_of_features.append(round(Thirdquart,2))
    array_features=np.array([list_of_features])
    if i==0:
        i=i+1
        df_checking=pd.DataFrame(array_features)
    else:
        df_checking.loc[i]=list_of_features
        i=i+1
df_checking['label']=list_multinomial
df_train1=df_checking.drop(['label'],axis=1)
df_test1=df_checking['label']
```

In [669]:

```
MNB_classifier=MultinomialNB()
MNB_classifier.fit(df_train1,df_test1)
prediction_result=MNB_classifier.predict(df_train1)
confusion_matrix(prediction_result, df_test1)
```

Out[669]:

```
array([[ 8,  0,  0,  1,  0,  0],
       [ 0, 12,  0,  0,  0,  0],
       [ 0,  0, 12,  1,  0,  0],
       [ 0,  0,  0, 10,  1,  0],
       [ 0,  0,  0,  0, 11,  0],
       [ 0,  0,  0,  0,  0, 12]], dtype=int64)
```

Which method is better for multi-class classification

In [675]:

```
print('While using L1-penalized Multinomial regression,We have best predict accrucy of 87.68%')
print('While using Gaussian Naive Bayes,We have best predict accrucy of 100%')
print('While using Multinomial Naive Bayes,We have best predict accrucy of 95.58%')
```

```
While using L1-penalized Multinomial regression,We have best predict accrucy of 87.68%
While using Gaussian Naive Bayes,We have best predict accrucy of 100%
While using Multinomial Naive Bayes,We have best predict accrucy of 95.58%
```

From the above it is clearly understood that Gaussian Naive Bayes is the better for Multi-class Classification