Engineering Analytics and Machine Learning Lab 4

for Specialist Diploma in Internet of Things

Author’s Name: Teo Kok Keong

1 Normalization

Lets try range scaling on the DAILYDATA_S24_201801.csv weather data. We are trying to find whether DailyRainfallTotal have any relationship with MeanTemperature. Before we even do anything, lets see how is the raw data like.

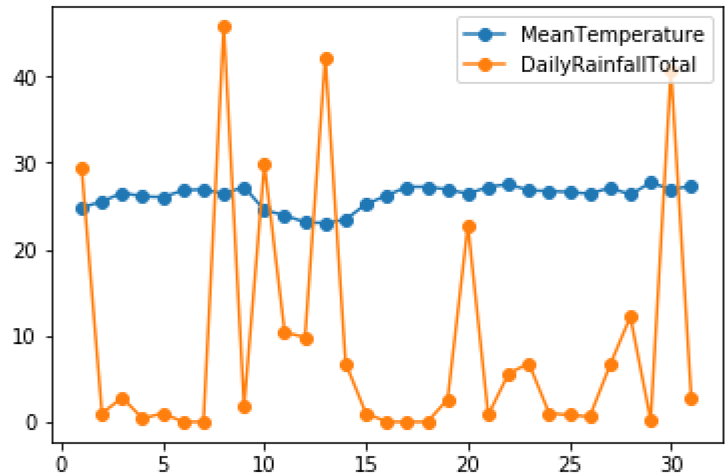
In [1]:
```
#import library required
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv('DAILYDATA_S24_201801.csv')  #load the data
df.head()   #have a preview of the data
```

Out[2]:

	Station	Year	Month	Day	DailyRainfallTotal	Highest30MinRainfall	Highest60MinRain
0	Changi	2018	1	1	29.4	6.0	11.6
1	Changi	2018	1	2	1.0	0.4	0.4
2	Changi	2018	1	3	2.8	1.8	2.0
3	Changi	2018	1	4	0.4	0.2	0.2
4	Changi	2018	1	5	1.0	1.0	1.0

In [3]:
```
#Let's plot scatter plot of both DailyRainfallTotal with MeanTemperature vs Da
y on the same graph
plt.plot(df.Day, df.MeanTemperature,'-o')
plt.plot(df.Day, df.DailyRainfallTotal,'-o')
plt.legend()
plt.show()
```



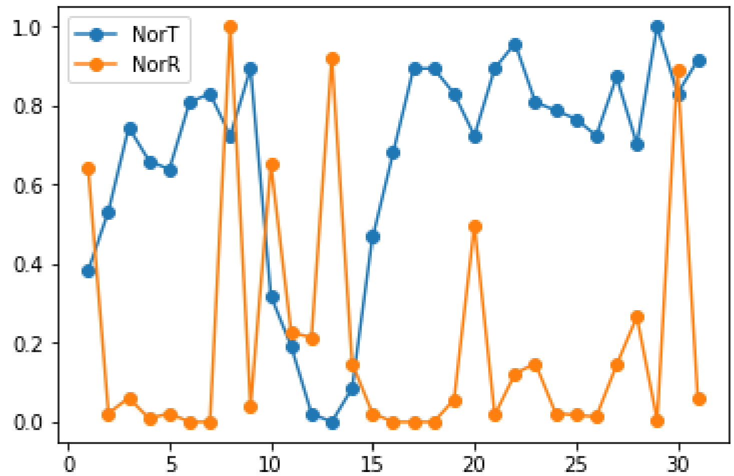
From the graph we can see that there is not much change in the mean temperature, it is nearly a straight line. We are unablet o observe any obvious relationship between the two parameters. This is because we are comparing two parameters of different scale, which is comparing apple with orange. Daily Rainfall total value varied from 0 to as high as more than 40. However, temperature only varied between the narrow band of maybe 25 to 31 or 32.

In [4]:
```
minT=min(df.MeanTemperature)    #find the min of MeanTemperature
maxT=max(df.MeanTemperature)    #find the max of MeanTemperature
df['NorT']=(df.MeanTemperature-minT)/(maxT-minT)    #Scale to 0 t 1 of Mean Tem
perature
#Scale to 0 to 1 for Daily Rainfall Total
df['NorR']=(df.DailyRainfallTotal-min(df.DailyRainfallTotal))/(max(df.DailyRai
nfallTotal)-min(df.DailyRainfallTotal))
df.head() #Preview the data again
```

Out[4]:

	Station	Year	Month	Day	DailyRainfallTotal	Highest30MinRainfall	Highest60MinRain
0	Changi	2018	1	1	29.4	6.0	11.6
1	Changi	2018	1	2	1.0	0.4	0.4
2	Changi	2018	1	3	2.8	1.8	2.0
3	Changi	2018	1	4	0.4	0.2	0.2
4	Changi	2018	1	5	1.0	1.0	1.0

In [5]:
```
#let's plot scatter plot of both rescaled DailyRainfallTotal with MeanTemperat
ure vs Day on the same graph
plt.plot(df.Day, df.NorT,'-o')
plt.plot(df.Day, df.NorR,'-o')
plt.legend()
plt.show()
```



The relationship become obvious for observation after rescale both data to 0 to 1.

Exercise 1

1a Plot MeanWindSpeed vs Day and DailyRainfallTotal vs Day in one graph

In [6]:
```
#Exercise 1a
#let's plot scatter plot of both DailyRainfallTotal with MeanTemperature vs Da
y on the same graph
```

1b Scaled MeanWindSpeed to scale of 0 to 1. Plot scaled MeanWindSpeed vs Day and scaled DailyRainfallTotal vs Day in one graph

In [7]:
```
#Scale to 0 to 1 for MeanWindSpeed




#let's plot line plot of both scaled MeanWindSpeed vs day and Scaled DailyRain
fallTotal vs Day on the same graph
```

1c Any comment?

Exercise 2

Try with DAILYDATA_S24_201801.csv with standardization transformation to find the relationship btween

- Total Daily Rainfall and MeanTemperature
- Total Daily Rainfall and MeanWindSpeed

Hint: Use np.mean() and np.std() to find mean and standard deviation

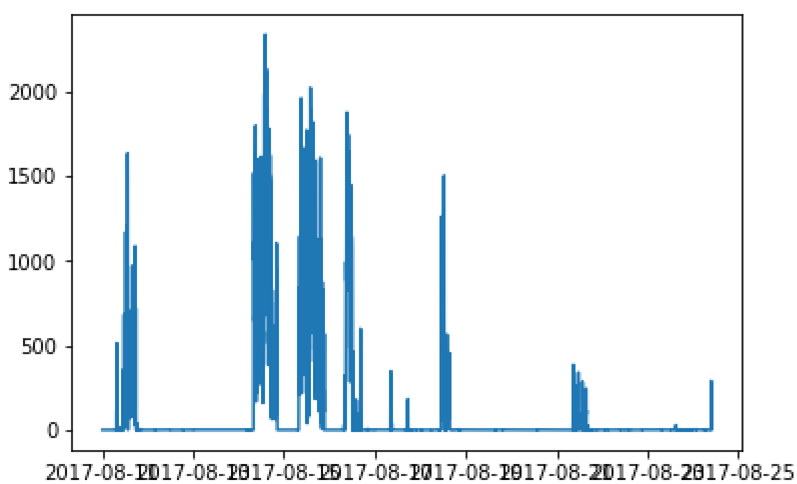
In [8]:
```
#your code start here
```

Is it as useful in this as re-scaling to 0 to 1?

2 Data Aggregration

Data aggregation is a type of data and information mining process where data is searched, gathered and presented in a report-based, summarized format to achieve specific business objectives or processes and/or conduct human analysis.

In [9]:
```
dd=pd.read_csv('remote_v3.csv')  #load the data
dd['time']=pd.to_datetime(dd['time'],format='%Y-%m-%d %H:%M:%S+08:00',utc=True
) #convert string to datetime
dd['onlytime']=pd.to_datetime(dd.time.dt.floor('h')).dt.time
dd.set_index('time', inplace=True)
plt.plot(dd.index,dd['value'])
plt.show()
```



In [10]:
```
dd.describe() #we could always use the describe method to understand the data
```

Out[10]:

	value
count	6441.000000
mean	89.950474
std	294.443643
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	2337.000000

We load the data, convert the datetime and change to index. We also plot a graph but this tell us nothing. Since this is in a class room, we ould expect the occupancy of the venue to change according to day of week (monday, tuesday...) and timing. We do have the time but no information on the day of week. Howeverr, we have the date so we could generate a new column that indicate the day of week.

In [11]:
```
dd['dayofweek']=dd.index.dayofweek
dd.head()
```

Out[11]:

	value	onlytime	dayofweek
time			
2017-08-11 00:00:00+00:00	0.0	00:00:00	4
2017-08-11 00:03:00+00:00	0.0	00:00:00	4
2017-08-11 00:06:00+00:00	0.0	00:00:00	4
2017-08-11 00:09:00+00:00	0.0	00:00:00	4
2017-08-11 00:12:00+00:00	0.0	00:00:00	4

In [12]:
```
#this section only to illustration some aggregation method available
print("result of sum:")
print(dd.sum()) #sum  accordingly to column
print("result of mean:")
print(dd.mean())  #mean  accordingly to colum

#by default aggregation return results within each column
#by specifying the axis argument, we can instead aggregate within each row:

print(dd.mean(axis='columns').head()) #in this case does not make sense but we
 are only illustrating the feature
```

```
result of sum:
value        579371.0
dayofweek     19323.0
dtype: float64
result of mean:
value        89.950474
dayofweek     3.000000
dtype: float64
time
2017-08-11 00:00:00+00:00    2.0
2017-08-11 00:03:00+00:00    2.0
2017-08-11 00:06:00+00:00    2.0
2017-08-11 00:09:00+00:00    2.0
2017-08-11 00:12:00+00:00    2.0
dtype: float64
```

So far we have summary the data by rows or columns, most of the time we need to aggregate with GroupBy to find insight into the data.

Back to analysising the remote eye data. We probably want to groupby dayofweek and apply mean operation.

In [13]:
```
#lets group by day of  week and view it statistical properties
dd.groupby('dayofweek').describe()
```

Out[13]:

	value							
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
dayofweek								
0	960.0	238.586458	478.290875	0.0	0.0	0.0	161.00	2337.0
1	960.0	211.870833	414.311167	0.0	0.0	0.0	188.25	2022.0
2	960.0	81.553125	300.554526	0.0	0.0	0.0	1.00	1876.0
3	681.0	2.819383	22.476418	0.0	0.0	0.0	1.00	348.0
4	960.0	69.116667	179.080594	0.0	0.0	0.0	2.00	1634.0
5	960.0	0.198958	0.455529	0.0	0.0	0.0	0.00	3.0
6	960.0	0.185417	0.436829	0.0	0.0	0.0	0.00	3.0

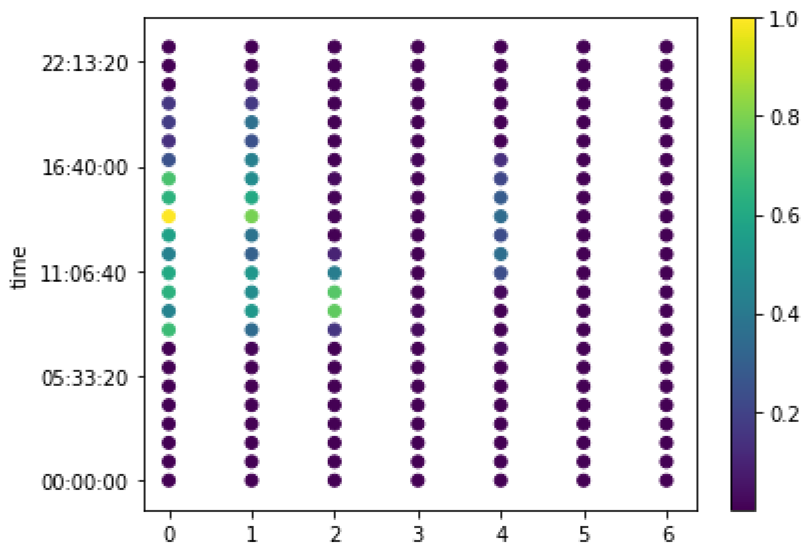
In [14]:
```
#we would perform aggregate on the data
gdd=dd.groupby(['dayofweek','onlytime']).aggregate([np.mean, min])
print(gdd)
print(gdd.loc[0])
```

```
                        value     
                         mean  min
dayofweek onlytime                
0         00:00:00      0.250  0.0
          01:00:00      0.275  0.0
          02:00:00      0.200  0.0
          03:00:00      0.125  0.0
          04:00:00      0.275  0.0
          05:00:00      0.175  0.0
          06:00:00      0.350  0.0
          07:00:00      0.075  0.0
          08:00:00    602.750  0.0
          09:00:00    396.775  0.0
          10:00:00    567.500  0.0
          11:00:00    536.375  0.0
          12:00:00    385.825  0.0
          13:00:00    505.800  0.0
          14:00:00    885.925  0.0
          15:00:00    569.700  0.0
          16:00:00    626.900  0.0
          17:00:00    224.875  0.0
          18:00:00    123.200  0.0
          19:00:00    158.975  0.0
          20:00:00    138.975  0.0
          21:00:00      0.500  0.0
          22:00:00      0.050  0.0
          23:00:00      0.225  0.0
1         00:00:00      0.025  0.0
          01:00:00      0.400  0.0
          02:00:00      0.075  0.0
          03:00:00      0.250  0.0
          04:00:00      0.025  0.0
          05:00:00      0.325  0.0
...                       ...  ...
5         18:00:00      0.200  0.0
          19:00:00      0.425  0.0
          20:00:00      0.125  0.0
          21:00:00      0.125  0.0
          22:00:00      0.050  0.0
          23:00:00      0.325  0.0
6         00:00:00      0.175  0.0
          01:00:00      0.225  0.0
          02:00:00      0.125  0.0
          03:00:00      0.300  0.0
          04:00:00      0.100  0.0
          05:00:00      0.175  0.0
          06:00:00      0.050  0.0
          07:00:00      0.200  0.0
          08:00:00      0.175  0.0
          09:00:00      0.150  0.0
          10:00:00      0.075  0.0
          11:00:00      0.275  0.0
          12:00:00      0.200  0.0
          13:00:00      0.350  0.0
          14:00:00      0.075  0.0
          15:00:00      0.375  0.0
          16:00:00      0.125  0.0
```

```
          17:00:00    0.175  0.0
          18:00:00    0.150  0.0
          19:00:00    0.250  0.0
          20:00:00    0.225  0.0
          21:00:00    0.200  0.0
          22:00:00    0.100  0.0
          23:00:00    0.200  0.0

[168 rows x 2 columns]
            value
             mean  min
onlytime
00:00:00    0.250  0.0
01:00:00    0.275  0.0
02:00:00    0.200  0.0
03:00:00    0.125  0.0
04:00:00    0.275  0.0
05:00:00    0.175  0.0
06:00:00    0.350  0.0
07:00:00    0.075  0.0
08:00:00  602.750  0.0
09:00:00  396.775  0.0
10:00:00  567.500  0.0
11:00:00  536.375  0.0
12:00:00  385.825  0.0
13:00:00  505.800  0.0
14:00:00  885.925  0.0
15:00:00  569.700  0.0
16:00:00  626.900  0.0
17:00:00  224.875  0.0
18:00:00  123.200  0.0
19:00:00  158.975  0.0
20:00:00  138.975  0.0
21:00:00    0.500  0.0
22:00:00    0.050  0.0
23:00:00    0.225  0.0
```

In [15]:

```
#we would perform aggregate on the data
gdd=dd.groupby(['dayofweek','onlytime']).aggregate([np.mean])

#find the max mean value to normalize the data for comparision
bb=[]

for i in range(7):
    b=max(gdd.loc[i,'value']['mean'])
    bb.append(b)

bb_max=max(bb)

fig, ax = plt.subplots()
ll=[0]*len(gdd.xs(0))

for i in range(7):
    ll=[i]*len(gdd.xs(i))
    ax=plt.scatter(ll,gdd.xs(i).index,c=gdd.loc[i,'value']['mean']/bb_max,vmax
=1)

plt.colorbar()
fig.tight_layout()
plt.show()
```

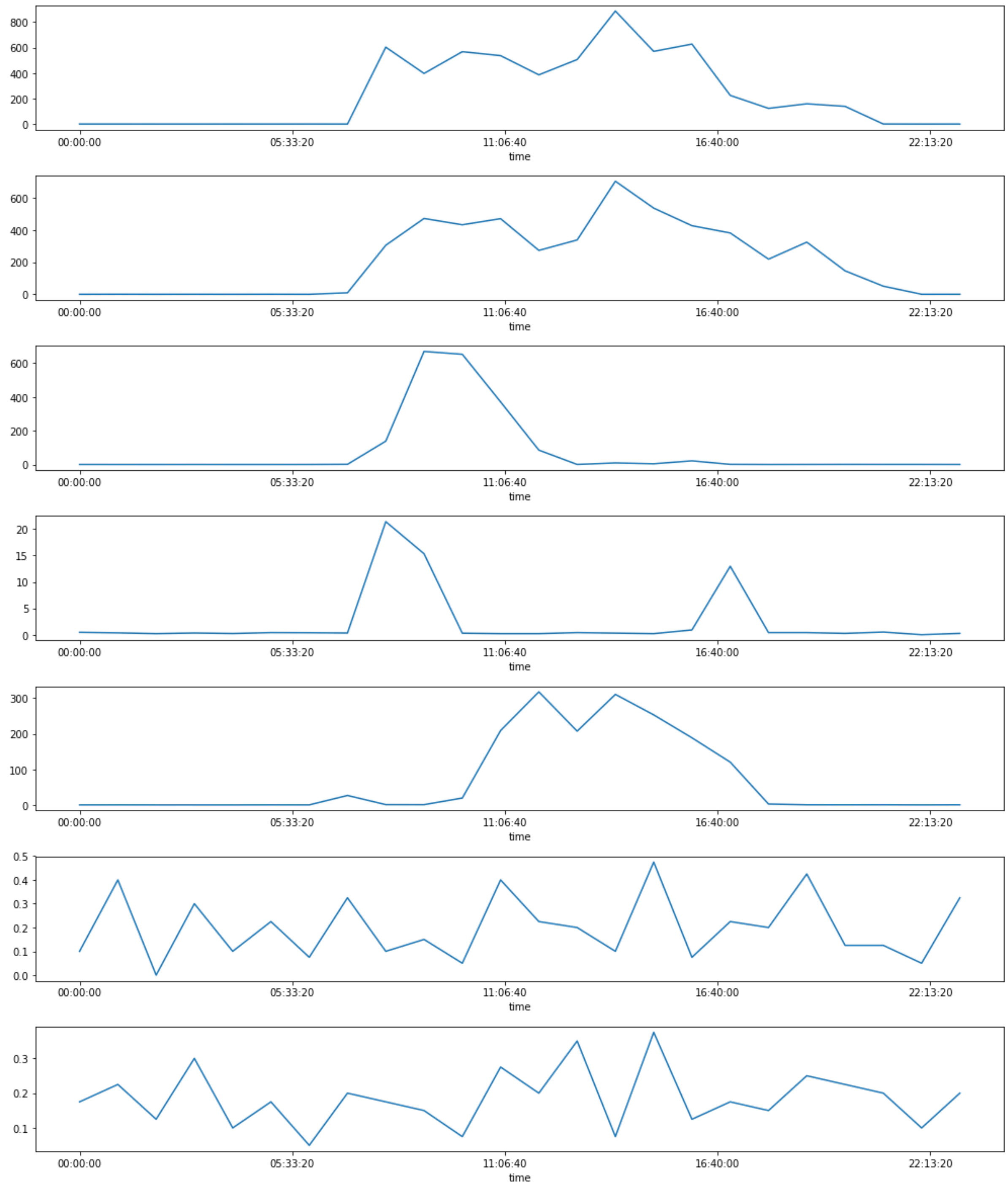


List of aggregate built-in statistic functions

Function	Description
count	Number of non-null observations
sum	Sum of values
mean	Mean of values
mad	Mean absolute deviation
median	Arithmetic median of values
min	Minimum
max	Maximum
mode	Mode
abs	Absolute Value
prod	Product of values
std	Unbiased standard deviation
var	Unbiased variance
sem	Unbiased standard error of the mean
skew	Unbiased skewness (3rd moment)
kurt	Unbiased kurtosis (4th moment)
quantile	Sample quantile (value at %)
cumsum	Cumulative sum
cumprod	Cumulative product
cummax	Cumulative maximum
cummin	Cumulative minimum

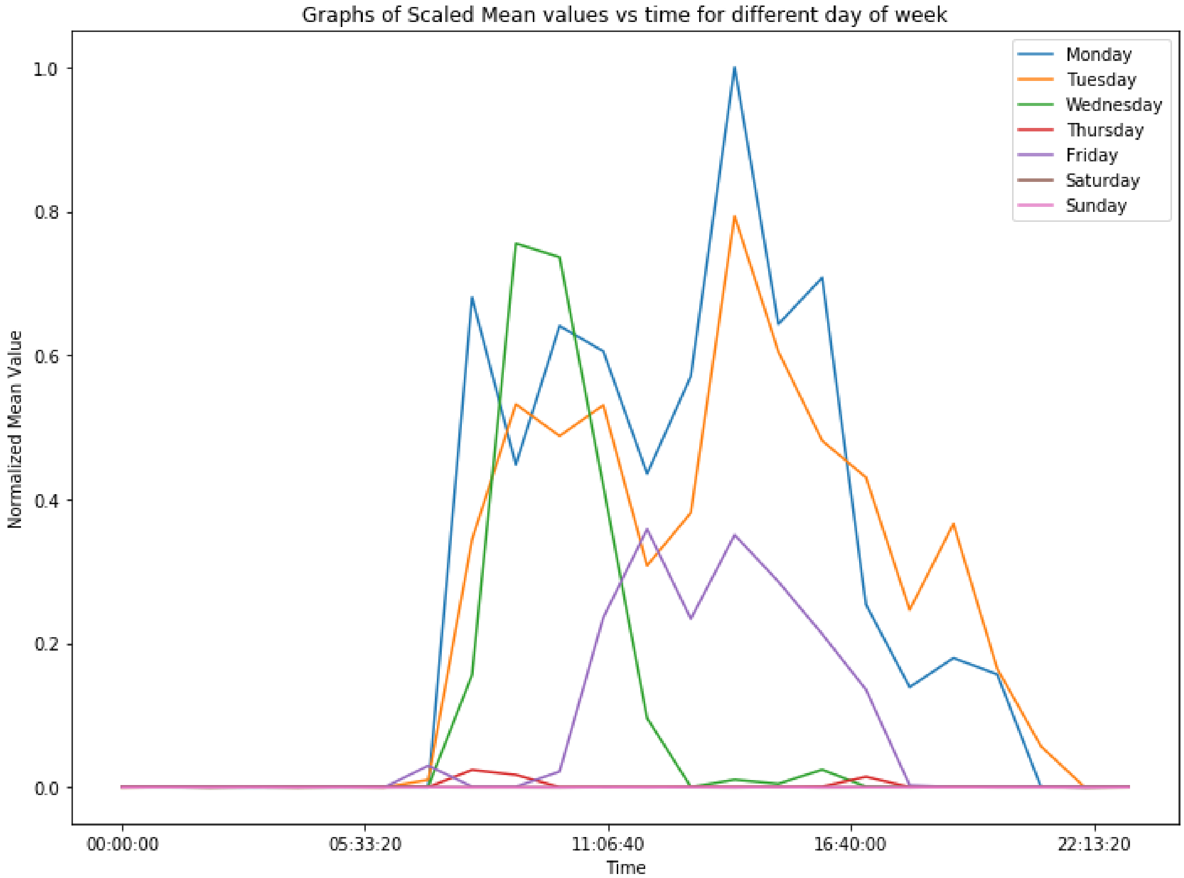
In [16]:
```
plt.subplot(711)
fig = plt.gcf()
fig.set_size_inches(14, 16.5)

# equivalent but more general
axx=plt.subplot(7, 1, 1)


axx.plot(gdd.loc[0].index,gdd.loc[0,'value']['mean'])
axx2=plt.subplot(7,1, 2)
axx2.plot(gdd.loc[1].index,gdd.loc[1,'value']['mean'])
axx3=plt.subplot(7,1, 3)
axx3.plot(gdd.loc[2].index,gdd.loc[2,'value']['mean'])
axx4=plt.subplot(7,1, 4)
axx4.plot(gdd.loc[3].index,gdd.loc[3,'value']['mean'])
axx5=plt.subplot( 7,1, 5)
axx5.plot(gdd.loc[4].index,gdd.loc[4,'value']['mean'])
axx6=plt.subplot(7,1, 6)
axx6.plot(gdd.loc[5].index,gdd.loc[5,'value']['mean'])
axx7=plt.subplot(7,1, 7)
axx7.plot(gdd.loc[6].index,gdd.loc[6,'value']['mean'])
plt.tight_layout()
plt.show()
```

```
c:\users\teokk\appdata\local\programs\python\python35\lib\site-packages\matpl
otlib\cbook\deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes
using the same arguments as a previous axes currently reuses the earlier inst
ance.  In a future version, a new instance will always be created and returne
d.  Meanwhile, this warning can be suppressed, and the future behavior ensure
d, by passing a unique label to each axes instance.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



In [17]:

```python
bb=[]
for i in range(7):
    b=max(gdd.loc[i,'value']['mean'])
    bb.append(b)

bb_max=max(bb)

#plt.subplot(711)
fig = plt.gcf()
fig.set_size_inches(10, 7.5)

# equivalent but more general
#axx=plt.subplot(7, 1, 1)


plt.plot(gdd.loc[0].index,gdd.loc[0,'value']['mean']/bb_max,label='Monday')
#axx2=plt.subplot(7,1, 2)
plt.plot(gdd.loc[1].index,gdd.loc[1,'value']['mean']/bb_max,label='Tuesday')
#axx3=plt.subplot(7,1, 3)
plt.plot(gdd.loc[2].index,gdd.loc[2,'value']['mean']/bb_max,label='Wednesday')
#axx4=plt.subplot(7,1, 4)
plt.plot(gdd.loc[3].index,gdd.loc[3,'value']['mean']/bb_max,label='Thursday')
#axx5=plt.subplot( 7,1, 5)
plt.plot(gdd.loc[4].index,gdd.loc[4,'value']['mean']/bb_max,label='Friday')
#axx6=plt.subplot(7,1, 6)
plt.plot(gdd.loc[5].index,gdd.loc[5,'value']['mean']/bb_max,label='Saturday')
#axx7=plt.subplot(7,1, 7)
plt.plot(gdd.loc[6].index,gdd.loc[6,'value']['mean']/bb_max,label='Sunday')
plt.xlabel('Time')
plt.ylabel('Normalized Mean Value')
plt.title('Graphs of Scaled Mean values vs time for different day of week')
plt.legend()
plt.tight_layout()
plt.show()
```



Exercise 3

Load the iris dataset from sklearn (code would be provided below), the data set would be saved in variable data. Thare are two sets of data, data.data (data itself) and data.target (the bred code).

- use data.data and data.target to found one dataframe
- Compute mean, max, min of each parameter for each bred type (0,1,2,). Which aggregated value would be more suitable to classified the data according to the target.

In [18]:
```
#your code start here
```

Exercise 4

oad the amazonaws credit data from this site https://rodeo-tutorials.s3.amazonaws.com/data/credit-data-non-null.csv (https://rodeo-tutorials.s3.amazonaws.com/data/credit-data-non-null.csv)

- Slice a subset of data that consist of 'serious_dlqin2yrs', 'age', 'monthly_income'
- Compute the mean parameters for each serious_dlqin2yrs category of the subset

In [19]:
```
#your code start here
```

3 Filter data

A data frames columns can be queried with a boolean expression. Every frame has the module query() as one of its objects members.

We start by importing pandas, numpy and creating a dataframe:

In [20]:
```
import pandas as pd
import numpy as np

data = {'name': ['Alice', 'Bob', 'Charles', 'David', 'Eric'],
        'year': ['01-01-2017', '02-12-2017', '03-03-2017', '23-04-2017', '30-0
3-2017'],
        'salary': [40000, 24000, 31000, 20000, 30000]}

df = pd.DataFrame(data)
df.year=pd.to_datetime(df['year'],format='%d-%m-%Y',utc=True)
print(df)
print(df.info())
```

```
      name  salary                      year
0    Alice   40000 2017-01-01 00:00:00+00:00
1      Bob   24000 2017-12-02 00:00:00+00:00
2  Charles   31000 2017-03-03 00:00:00+00:00
3    David   20000 2017-04-23 00:00:00+00:00
4     Eric   30000 2017-03-30 00:00:00+00:00
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
name      5 non-null object
salary    5 non-null int64
year      5 non-null datetime64[ns, UTC]
dtypes: datetime64[ns, UTC](1), int64(1), object(1)
memory usage: 200.0+ bytes
None
```

Filter by Query

A data frames columns can be queried with a boolean expression. Every frame has the module query() as one of its objects members.

We start by importing pandas, numpy and creating a dataframe:

In [21]:
```
df_filtered = df.query('salary>30000')
print(df_filtered)
```

```
      name  salary                      year
0    Alice   40000 2017-01-01 00:00:00+00:00
2  Charles   31000 2017-03-03 00:00:00+00:00
```

In [22]:
```
df_filtered = df.query('salary<=30000')
print(df_filtered)
```

```
    name  salary                      year
1    Bob   24000 2017-12-02 00:00:00+00:00
3  David   20000 2017-04-23 00:00:00+00:00
4   Eric   30000 2017-03-30 00:00:00+00:00
```

In [23]:
```
df_filtered = df.query('salary==30000')
print(df_filtered)
```

```
   name  salary                      year
4  Eric   30000 2017-03-30 00:00:00+00:00
```

In [24]:
```
df_filtered = df.query('salary>=2000 and salary<=30000')
print(df_filtered)
```

```
    name  salary                      year
1    Bob   24000 2017-12-02 00:00:00+00:00
3  David   20000 2017-04-23 00:00:00+00:00
4   Eric   30000 2017-03-30 00:00:00+00:00
```

Filter by indexing

In [25]:
```
df_filtered = df[(df.salary >= 30000) & (df.year >'01-01-2017')]
print(df_filtered)
```

```
      name  salary                      year
2  Charles   31000 2017-03-03 00:00:00+00:00
4     Eric   30000 2017-03-30 00:00:00+00:00
```

In [26]:
```
df_filtered = df[(df.year >= '01-01-2007') & (df.year > '01-06-2017')]
print(df_filtered)
```

```
      name  salary                      year
1      Bob   24000 2017-12-02 00:00:00+00:00
2  Charles   31000 2017-03-03 00:00:00+00:00
3    David   20000 2017-04-23 00:00:00+00:00
4     Eric   30000 2017-03-30 00:00:00+00:00
```

Filter by Pandas Groupby

In [27]:
```
df1 = pd.DataFrame( {
    "Name" : ["Alice", "Ada", "Mallory", "Mallory", "Billy" , "Mallory"] ,
    "City" : ["Sydney", "Sydney", "Paris", "Sydney", "Sydney", "Paris"]} )
print(df1)
```

```
     City     Name
0  Sydney    Alice
1  Sydney      Ada
2   Paris  Mallory
3  Sydney  Mallory
4  Sydney    Billy
5   Paris  Mallory
```

In [28]:
```
print(df1.groupby(["City"])[['Name']].count())
```

```
        Name
City        
Paris      2
Sydney     4
```

Exercise 5

Load the bike sharing hourly data file bike_sharing_hourly.xlsx file. Please filter to obtain data frame that:

- contain only on holiday
- contain only season 1 and 3
- contain only data from 2011-01-01 to 2011-010-05

Note: Below contact code to load excel file to Pandas dataframe Note: Install xlrd package for excel support, on anaconda use command "command conda install xlrd" for installation withhout virtual environment use pip with command "pip install xlrd

In [29]:
```
#your code start here
```