Data Mining hw0 Report

姓名：蔣宜桓

學號：**0656092**

Project: https://github.com/thumbe12856/Taipower-and-Weather-Data-Mining

Task 1.

Step 1. 先建立 2 張資料表，test 用來儲存台電的資料，weather 用來儲存天氣的資料

/create-table.sql

```sql
create table test(
    `id` int NOT NULL AUTO_INCREMENT,
    `date`DATE,
    `time` char(128),
    `northSupply` DOUBLE,
    `northUsage` DOUBLE,
    `centerSupply` DOUBLE,
    `centerUsage` DOUBLE,
    `southSupply` DOUBLE,
    `southUsage` DOUBLE,
    `eastSupply` DOUBLE,
    `eastUsage` DOUBLE,
    PRIMARY KEY (id)
)

create table weather(
    `date`DATE,
    `time` char(128),
    `city` char(128),
    `cityIndex` int,
    `temperature` double
)
```

Step 2. 從 json、csv 檔中讀取台電資料、xml 檔中讀取天氣資料，將資料存進資料庫中。

/loadTaipowerData.py

因為台電的資料中，9/2 的資料在 json 和 csv 中重複，而 csv 中的資料比較詳細，所以以 csv 的資料為準，忽略 json 中 9/2 的資料。另外，9/10 的資料並不存在，所以忽略這份 csv 檔案。

讀取 json、csv 檔案，取得各地區的電力供給、使用資料後，存入資料庫中相對應的欄位。

```python
import mysql.connector
import json
import csv

# connet to mysql
connet = mysql.connector.connect(
    user="dm",
    password="dm",
    host="127.0.0.1",
    database="DM_hw0"
)
cursor = connet.cursor()
insertData = list()

# load Taipower data, json file, from 2016/09/27 - 2017/09/02.
with open("./data/Taipower/power.json") as data_file:
    loadData = json.load(data_file)

# tempDate, consider to load data from a day to day.
tempDate = '2016-09-27'
northSupply = northUsage = centerSupply = centerUsage = southSupply = southUsage = eastSupply = eastUsage = 0.0

for i in range(len(loadData)):

    # cuase there is 2017-09-02 data in csv file, and the csv file is more accurate.
    if loadData[i]["create_date"]["$date"][:10] == "2017-09-02": break;

    tempDate = loadData[i]["create_date"]["$date"][:10]
    tempTime = loadData[i]["create_date"]["$date"][10:]
    northSupply = float(loadData[i]["regionData"]["northSupply"])
    northUsage = float(loadData[i]["regionData"]["northUsage"])
    centerSupply = float(loadData[i]["regionData"]["centerSupply"])
    centerUsage = float(loadData[i]["regionData"]["centerUsage"])
    southSupply = float(loadData[i]["regionData"]["southSupply"])
    southUsage = float(loadData[i]["regionData"]["southUsage"])
    eastSupply = float(loadData[i]["regionData"]["eastSupply"])
    eastUsage = float(loadData[i]["regionData"]["eastUsage"])

    insertData.append(tuple((
        tempDate, tempTime, northSupply, northUsage, centerSupply, centerUsage, southSupply, southUsage, eastSupply, eastUsage
    )))

# load Taipower data, csv file, from 2017/09/02 - 2017/09/23.
for i in range(2, 24):
```

```python
    # cause the file does not exist.
    if i == 10: continue

    date = "2017-09-" + str(i)
    file = "./data/Taipower/" + date + ".csv"
    f = open(file, 'r')

    for row in csv.reader(f):
        northSupply = float(row[1])
        northUsage = float(row[2])
        centerSupply = float(row[3])
        centerUsage = float(row[4])
        southSupply = float(row[5])
        southUsage = float(row[6])
        eastSupply = float(row[7])
        eastUsage = float(row[8])

        insertData.append(tuple((
            date, row[0], northSupply, northUsage, centerSupply, centerUsage, southSupply, southUsage, eastSupply, eastUsage
        )))
    f.close()

# insert data to database.
insert = ("INSERT INTO `test` (date, time, northSupply, northUsage, centerSupply, centerUsage, southSupply, southUsage, eastSupply, east
cursor.executemany(insert, insertData);
connet.commit()
cursor.close()

# close mysql connetion.
connet.close()

```

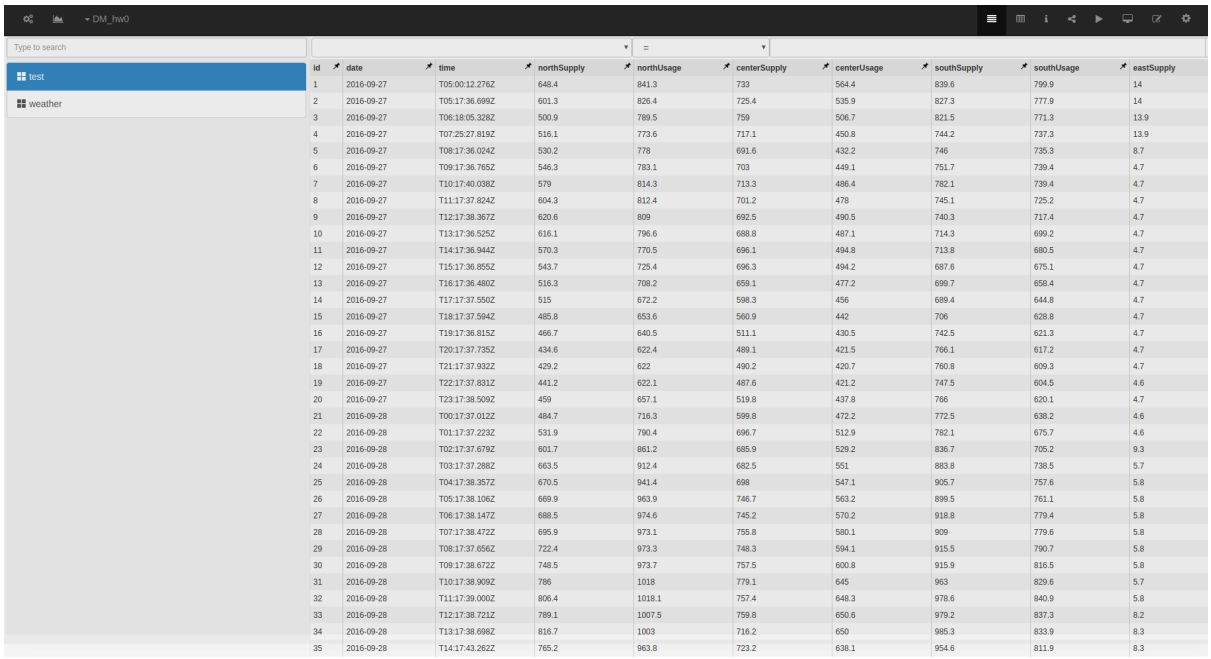
/loadWeatherData.py

先讀取 xml 找到根節點 root，再選取四座城市（北：台北，中：台中，南：高雄：台東）相對應的索引（index），再將時間、温度的資料寫入資料庫中。

```python
import mysql.connector
import xml.etree.ElementTree

def insertDataToDB(city, cityIndex):
    # connet to mysql
    connet = mysql.connector.connect(
        user="dm",
        password="dm",
        host="127.0.0.1",
        database="DM_hw0"
    )
    cursor = connet.cursor()
    insertData = list()
    root = xml.etree.ElementTree.parse('./data/weather/C-B0024-002.xml').getroot()

    # root -> dataset(root[7]) -> locationName(location[0])
    for timeElement in root[7][cityIndex][2].findall('{urn:cwb:gov:tw:cwbcommon:0.1}time'):

        # tempNowDate = time.strptime(timeElement[0].text[:10], "%Y-%m-%d")
        date = timeElement[0].text[:10]
        time = timeElement[0].text[10:]
        temperature = float(timeElement[2][1][0].text)
        insertData.append(tuple((
            date, time, city, cityIndex, temperature
        )))
    insert = ("INSERT INTO `weather` (date, time, city, cityIndex, temperature) VALUES (%s, %s, %s, %s, %s)")
    cursor.executemany(insert, insertData);
    connet.commit()
    cursor.close()

    # close mysql connetion.
    connet.close()


insertDataToDB('Taipei', 3)
insertDataToDB('Taitung', 26)
insertDataToDB('Kaohsiung', 15)
insertDataToDB('Taichung', 17)

```

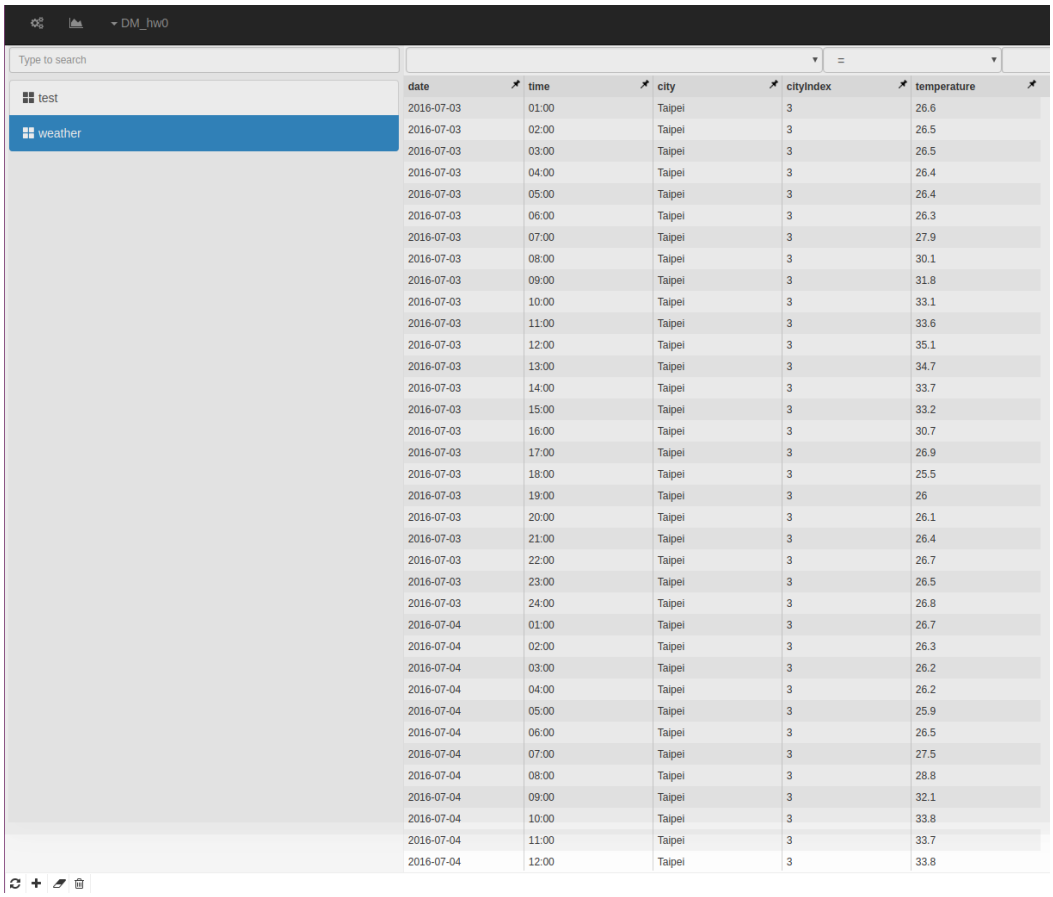
Task 1. Result.

台電資料資料表:



id	date	time	northSupply	northUsage	centerSupply	centerUsage	southSupply	southUsage	eastSupply
1	2016-09-27	T05:00:12.276Z	648.4	841.3	733	564.4	839.6	799.9	14
2	2016-09-27	T05:17:36.699Z	601.3	826.4	725.4	535.9	827.3	777.9	14
3	2016-09-27	T06:18:05.328Z	500.9	789.5	759	506.7	821.5	771.3	13.9
4	2016-09-27	T07:25:27.819Z	516.1	773.6	717.1	450.8	744.2	737.3	13.9
5	2016-09-27	T08:17:36.024Z	530.2	778	691.6	432.2	746	735.3	8.7
6	2016-09-27	T09:17:36.765Z	546.3	783.1	703	449.1	751.7	739.4	4.7
7	2016-09-27	T10:17:40.038Z	579	814.3	713.3	486.4	782.1	739.4	4.7
8	2016-09-27	T11:17:37.824Z	604.3	812.4	701.2	478	745.1	725.2	4.7
9	2016-09-27	T12:17:38.367Z	620.6	809	692.5	490.5	740.3	717.4	4.7
10	2016-09-27	T13:17:36.525Z	616.1	796.6	688.8	487.1	714.3	699.2	4.7
11	2016-09-27	T14:17:36.944Z	570.3	770.5	696.1	494.8	713.8	680.5	4.7
12	2016-09-27	T15:17:36.855Z	543.7	725.4	696.3	494.2	687.6	675.1	4.7
13	2016-09-27	T16:17:36.480Z	516.3	708.2	659.1	477.2	699.7	658.4	4.7
14	2016-09-27	T17:17:37.550Z	515	672.2	598.3	456	689.4	644.8	4.7
15	2016-09-27	T18:17:37.594Z	485.8	653.6	560.9	442	706	628.8	4.7
16	2016-09-27	T19:17:36.815Z	466.7	640.5	511.1	430.5	742.5	621.3	4.7
17	2016-09-27	T20:17:37.735Z	434.6	622.4	489.1	421.5	766.1	617.2	4.7
18	2016-09-27	T21:17:37.932Z	429.2	622	490.2	420.7	760.8	609.3	4.7
19	2016-09-27	T22:17:37.831Z	441.2	622.1	487.6	421.2	747.5	604.5	4.6
20	2016-09-27	T23:17:38.509Z	459	657.1	519.8	437.8	766	620.1	4.7
21	2016-09-28	T00:17:37.012Z	484.7	716.3	599.8	472.2	772.5	638.2	4.6
22	2016-09-28	T01:17:37.223Z	531.9	790.4	696.7	512.9	782.1	675.7	4.6
23	2016-09-28	T02:17:37.679Z	601.7	861.2	685.9	529.2	836.7	705.2	9.3
24	2016-09-28	T03:17:37.288Z	663.5	912.4	682.5	551	883.8	738.5	5.7
25	2016-09-28	T04:17:38.357Z	670.5	941.4	698	547.1	905.7	757.6	5.8
26	2016-09-28	T05:17:38.106Z	669.9	963.9	746.7	563.2	899.5	761.1	5.8
27	2016-09-28	T06:17:38.147Z	688.5	974.6	745.2	570.2	918.8	779.4	5.8
28	2016-09-28	T07:17:38.472Z	695.9	973.1	755.8	580.1	909	779.6	5.8
29	2016-09-28	T08:17:37.656Z	722.4	973.3	748.3	594.1	915.5	790.7	5.8
30	2016-09-28	T09:17:38.672Z	748.5	973.7	757.5	600.8	915.9	816.5	5.8
31	2016-09-28	T10:17:38.909Z	786	1018	779.1	645	963	829.6	5.7
32	2016-09-28	T11:17:39.000Z	806.4	1018.1	757.4	648.3	978.6	840.9	5.8
33	2016-09-28	T12:17:38.721Z	789.1	1007.5	759.8	650.6	979.2	837.3	8.2
34	2016-09-28	T13:17:38.698Z	816.7	1003	716.2	650	985.3	833.9	8.3
35	2016-09-28	T14:17:43.262Z	765.2	963.8	723.2	638.1	954.6	811.9	8.3

天氣資料資料表:



date	time	city	cityIndex	temperature
2016-07-03	01:00	Taipei	3	26.6
2016-07-03	02:00	Taipei	3	26.5
2016-07-03	03:00	Taipei	3	26.5
2016-07-03	04:00	Taipei	3	26.4
2016-07-03	05:00	Taipei	3	26.4
2016-07-03	06:00	Taipei	3	26.3
2016-07-03	07:00	Taipei	3	27.9
2016-07-03	08:00	Taipei	3	30.1
2016-07-03	09:00	Taipei	3	31.8
2016-07-03	10:00	Taipei	3	33.1
2016-07-03	11:00	Taipei	3	33.6
2016-07-03	12:00	Taipei	3	35.1
2016-07-03	13:00	Taipei	3	34.7
2016-07-03	14:00	Taipei	3	33.7
2016-07-03	15:00	Taipei	3	33.2
2016-07-03	16:00	Taipei	3	30.7
2016-07-03	17:00	Taipei	3	26.9
2016-07-03	18:00	Taipei	3	25.5
2016-07-03	19:00	Taipei	3	26
2016-07-03	20:00	Taipei	3	26.1
2016-07-03	21:00	Taipei	3	26.4
2016-07-03	22:00	Taipei	3	26.7
2016-07-03	23:00	Taipei	3	26.5
2016-07-03	24:00	Taipei	3	26.8
2016-07-04	01:00	Taipei	3	26.7
2016-07-04	02:00	Taipei	3	26.3
2016-07-04	03:00	Taipei	3	26.2
2016-07-04	04:00	Taipei	3	26.2
2016-07-04	05:00	Taipei	3	25.9
2016-07-04	06:00	Taipei	3	26.5
2016-07-04	07:00	Taipei	3	27.5
2016-07-04	08:00	Taipei	3	28.8
2016-07-04	09:00	Taipei	3	32.1
2016-07-04	10:00	Taipei	3	33.8
2016-07-04	11:00	Taipei	3	33.7
2016-07-04	12:00	Taipei	3	33.8

Task 2.

Task 2.(a). 利用 DB 的 sql 達成找出特定時間內各個區域每天最大用電、供電量。

已每天為單位，選出在日期 2017/6/31 之前、2016/10/1 之後，最大的該單位數值。

/getMaxPowerData.sql

```sql
select test.date, MAX(test.northSupply) from `test` where date >= '2016/10/1' and date < '2017/6/31' group by test.date;
select test.date, MAX(test.northUsage) from `test` where  date >= '2016/10/1' and date < '2017/6/31' group by test.date;
select test.date, MAX(test.centerSupply) from `test` where date >= '2016/10/1' and date < '2017/6/31' group by test.date;
select test.date, MAX(test.centerUsage) from `test` where date >= '2016/10/1' and date < '2017/6/31' group by test.date;
select test.date, MAX(test.southSupply) from `test` where date >= '2016/10/1' and  date < '2017/6/31' group by test.date;
select test.date, MAX(test.southUsage) from `test` where date >= '2016/10/1' and  date < '2017/6/31' group by test.date;
select test.date, MAX(test.eastSupply) from `test` where date >= '2016/10/1' and  date < '2017/6/31' group by test.date;
select test.date, MAX(test.eastUsage) from `test` where date >= '2016/10/1' and  date < '2017/6/31' group by test.date;

```

Task 2.(a). Result.

/result/task2/2.a/max_centerSupply.csv
/result/task2/2.a/max_centerUsage.csv
/result/task2/2.a/max_eastSupply.csv
/result/task2/2.a/max_eastUsage.csv
/result/task2/2.a/max_northSupply.csv
/result/task2/2.a/max_northUsage.csv
/result/task2/2.a/max_southSupply.csv
/result/task2/2.a/max_southUsage.csv

/result/task2/2.a/max_northSupply.csv:

	A	B	C
1	date	MAX(test.northSupply)	
2	2016-10-01	943	
3	2016-10-02	872.9	
4	2016-10-03	1150.5	
5	2016-10-04	1139.6	
6	2016-10-05	1150.8	
7	2016-10-06	1099.9	
8	2016-10-07	1108.1	
9	2016-10-08	890.7	
10	2016-10-09	739.8	
11	2016-10-10	837.1	
12	2016-10-11	1012.7	
13	2016-10-12	1011.5	
14	2016-10-13	1044.8	
15	2016-10-14	1049.5	

Task 2.(b). 先讀取 xml 找到根節點 root，再用 for 迴圈將 root->dataset->location->locationName 的內容印出來。

/weather.py

```
79  # main
80  # get root of the xml file
81  root = xml.etree.ElementTree.parse('./data/weather/C-B0024-002.xml').getroot()
82
83  # Task 2.b
84  # find all locationName
85  # root -> dataset(root[7]) -> locationName(location[0])
86  for location in root[7]:
87      print location[0].text
```

Task 2.(b). Result.

/result/task2/2.b/2.b.txt

```
1  BANQIAO,板橋
2  TAMSUI,淡水
3  ANBU,鞍部
4  TAIPEI,臺北
5  ZHUZIHU,竹子湖
6  KEELUNG,基隆
7  PENGJIAYU,彭佳嶼
8  HUALIEN,花蓮
9  XINWU,新屋
10 SU-AO,蘇澳
11 YILAN,宜蘭
12 KINMEN,金門
13 DONGJIDAO,東吉島
14 PENGHU,澎湖
15 TAINAN,臺南
16 KAOHSIUNG,高雄
17 CHIAYI,嘉義
18 TAICHUNG,臺中
19 ALISHAN,阿里山
20 DAWU,大武
21 YUSHAN,玉山
22 HSINCHU,新竹
23 HENGCHUN,恆春
24 CHENGGONG,成功
25 LANYU,蘭嶼
26 SUN MOON LAKE,日月潭
27 TAITUNG,臺東
28 WUQI,梧棲
29 MATSU,馬祖
30
```

Task 2.(c).

Step 1. 先決定好選取四座城市（北：台北，中：台中，南：高雄：台東），其各自在 xml 中 location tag 的索引值為 3、26、15、17。

/weather.py

```
 89  # Task 2.c
 90  # Find the maximum temperature value for each day and each area,
 91  #   (North , Center , South , East) from 2016/10/01 to 2017/06/30.
 92  # North: Taipei, 466920, root[7][3]
 93  # Center: Taitung, 467490, root[7][26]
 94  # South: Kaohsiung, 467440, root[7][15]
 95  # East: Taichung, 467490, root[7][17]
 96
 97  findAreaMaxTemperature('North area', 'Taipei', 3)
 98  findAreaMaxTemperature('Center area', 'Taitung', 26)
 99  findAreaMaxTemperature('South area', 'Kaohsiung', 15)
100  findAreaMaxTemperature('East area', 'Taichung', 17)
```

Step 2. 設定開始和結束日期（2016/10/01，2017/07/01），再用 for 迴圈尋找該城市下，所有 tag 為 time 中所含的溫度資料，利用 python 的 list 將每日最高溫紀錄下來，最後再寫入至 csv 檔中。

/weather.py

```
 5  def findAreaMaxTemperature(area, city, cityIndex):
 6      fromDate = time.strptime('2016/10/01', "%Y/%m/%d")
 7      toDate = time.strptime('2017/07/01', "%Y/%m/%d")
 8      tempDate = '2016-10-01'
 9      MaxTemperature = -10000.0
10
11      f = open('./result/task2/2.c/' + area + '.csv', 'w')
12      w = csv.writer(f)
13      writeData = [
14          [area, 'Date', 'Max Temperature']
15      ]
16
17      # root -> dataset (root[7]) -> location Taipei (root[7][3])
18      #   -> weatherElement (root[7][3][2]) -> find all time
19      for timeElement in root[7][cityIndex][2].findall('{urn:cwb:gov:tw:cwbcommon:0.1}time'):
20          tempNowDate = time.strptime(timeElement[0].text[:10], "%Y-%m-%d")
21          if(tempNowDate >= fromDate and tempNowDate <= toDate):
22              if(tempDate != timeElement[0].text[:10]):
23                  writeData.append([city, tempDate, MaxTemperature])
24                  tempDate = timeElement[0].text[:10]
25                  MaxTemperature = -10000
26              else:
27                  tempTemperature = float(timeElement[2][1][0].text)
28                  if(tempTemperature > MaxTemperature):
29                      MaxTemperature = tempTemperature
30
31      # write result to ./result/task2/2.c/{area}.csv.
32      w.writerows(writeData)
33      f.close()
```

Task 2.(c). Result.

/result/task2/2.c/Center\ area.csv
/result/task2/2.c/East\ area.csv
/result/task2/2.c/South\ area.csv
/result/task2/2.c/East\ area.csv

/result/task2/2.c/Center\ area.csv

	A	B	C
1	Center area	Date	Max Temperature
2	Taitung	2016-10-01	30.4
3	Taitung	2016-10-02	30.6
4	Taitung	2016-10-03	31.5
5	Taitung	2016-10-04	31.6
6	Taitung	2016-10-05	30.8
7	Taitung	2016-10-06	28.6
8	Taitung	2016-10-07	26
9	Taitung	2016-10-08	24.7
10	Taitung	2016-10-09	24.9
11	Taitung	2016-10-10	28.3
12	Taitung	2016-10-11	29.9
13	Taitung	2016-10-12	30.6
14	Taitung	2016-10-13	31.1
15	Taitung	2016-10-14	30.3

Task 2.(d).

Step 1. 找出台灣各區域城市的最高及最低溫度

```
102  # Task 2.d
103  findAllAreaMaxAndMinTemperature()
104
```

Step 2. 設定開始和結束日期（2016/10/01，2017/07/01），再用 for 迴圈尋找所有城市底下 tag 為 time 中所含的溫度資料，再紀錄最高、最低溫度、第 2 低溫度、城市名稱及時間，並寫入 result.csv

```
35  def findAllAreaMaxAndMinTemperature():
36      fromDate = time.strptime('2016/10/01', "%Y/%m/%d")
37      toDate = time.strptime('2017/07/01', "%Y/%m/%d")
38      MaxTemperature = -10000.0
39      SecondMinTemperature = MinTemperature = 10000.0
40      SecondMinTempCityName = SecondMinTempDate = MinTempCityName = MinTempDate = MaxTempCityName = MaxTempDate = ''
41
42      for city in root[7]:
43          for timeElement in city[2].findall('{urn:cwb:gov:tw:cwbcommon:0.1}time'):
44              tempNowDate = time.strptime(timeElement[0].text[:10], "%Y-%m-%d")
45              if(tempNowDate >= fromDate and tempNowDate <= toDate):
46                  tempTemperature = float(timeElement[2][1][0].text)
47                  if(tempTemperature > MaxTemperature):
48                      MaxTempCityName = city[0].text
49                      MaxTempDate = timeElement[0].text
50                      MaxTemperature = tempTemperature
51
52                  if(tempTemperature < MinTemperature):
53                      MinTempCityName = city[0].text
54                      MinTempDate = timeElement[0].text
55                      MinTemperature = tempTemperature
56
57                  # because the minimal value of temperature is unusual.
58                  if(tempTemperature < SecondMinTemperature and tempTemperature > -99.5):
59                      SecondMinTempCityName = city[0].text
60                      SecondMinTempDate = timeElement[0].text
61                      SecondMinTemperature = tempTemperature
62
63      # write result to ./result/weather/task2/2.d/result.csv.
64      f = open('./result/task2/2.d/result.csv', 'w')
65      w = csv.writer(f)
66      writeData = [
67          ['City', 'Date', 'Max Temperature'],
68          [MaxTempCityName.encode('utf-8').strip(), MaxTempDate, MaxTemperature],
69          ['', ''],
70          ['City', 'Date', 'Min Temperature'],
71          [MinTempCityName.encode('utf-8').strip(), MinTempDate, MinTemperature],
72          ['', ''],
73          ['City', 'Date', 'Second Min Temperature'],
74          [SecondMinTempCityName.encode('utf-8').strip(), SecondMinTempDate, SecondMinTemperature]
75      ]
76      w.writerows(writeData)
77      f.close()
```

Task 2.(d). Result.

依據 xml 檔案中的資料，最高溫度為 37.5，在台灣南部的大武，時間為 2017/06/24。時間上接近夏天，地理上靠近赤道，數值很合理。

但最低溫度為-99.5，時間為 2017/1/26，地點在淡水。即使單位是華氏，轉換成攝氏溫度也有-73.05555562 °C。依常理來判斷，淡水不可能出現如此低溫，很明顯資料錯誤。

因此，我紀錄了第 2 低溫的資料，2017/02/13 在玉山上出現-10.2 度。時間為冬天，又是台灣第 1 高山，數值很合理。

/result/task2/2.d/result.csv

	A	B	C
1	City	Date	Max Temperature
2	DAWU,大武	2017-06-24	37.5
3			
4	City	Date	Min Temperature
5	TAMSUI,淡水	2017-01-26	-99.5
6			
7	City	Date	Second Min Temperature
8	YUSHAN,玉山	2017-02-13	-10.2

Task 3.

Step 1. 先決定好選取四座城市（北：台北，中：台中，南：高雄：台東），其各自在 xml 中 location tag 的索引值為 3、26、15、17。

/powerAndWeatherRelation.py

```
111  # main
112  # North: Taipei, 466920, [3]
113  drawCurve(3, 'north')
114
115  # Center: Taitung, 467490, [26]
116  drawCurve(26, 'center')
117
118  # South: Kaohsiung, 467440, [15]
119  drawCurve(15, 'south')
120
121  # East: Taichung, 467490, [17]
122  drawCurve(17, 'east')
123
```

Step 2. 透過資料庫，取出該城市的供電、用電資料及氣溫資料。

/powerAndWeatherRelation.py

```
 94  def drawCurve(cityIndex, area):
 95      # get power data
 96      powerDate, powerSupply = getPowerData(area + 'Supply')
 97      powerDate, powerUsage = getPowerData(area + 'Usage')
 98
 99      # get temperature data
100      weatherDate, temperature = getWeatherData(cityIndex)
101
```

Step 3. 因為電力及天氣的資料長度不同（電力缺少 2017 年 1 月到 4 月份的資料），因此需要將兩者的的資料長度補成一樣長，我的做法是，缺少的資料補 0（例如：缺少 2017/02/01 的電力資料，則當天的供給或使用量設 0；缺少 2017/02/01 的氣溫資料，則當天的氣溫設 0）

/powerAndWeatherRelation.py

```
102     # normalize the data
103     weatherDate, temperature, powerDate, powerSupply, powerUsage = \
104         setData(weatherDate, temperature, powerDate, powerSupply, powerUsage)
```

/powerAndWeatherRelation.py:setData()

```
70  # because there are some date that have no power data or weather data,
71  # so, this is to normalize.
72  def setData(weatherDate, temperature, powerDate, powerSupply, powerUsage):
73      i = 0
74      for singleDate in daterange(startDate, endDate):
75          tempDate = singleDate.strftime("%Y-%m-%d")
76          temperature[i] = temperature[i] * 50
77
78          # if weather has no data at the date, append 0 to data set
79          if(weatherDate[i] != tempDate):
80              print tempDate, ', ', i
81              weatherDate = np.insert(weatherDate, i, tempDate)
82              temperature = np.insert(temperature, i, 0)
83
84          # if power has no data at the date, append 0 to data set
85          if(powerDate[i] != tempDate):
86              powerDate = np.insert(powerDate, i, tempDate)
87              powerSupply = np.insert(powerSupply, i, 0)
88              powerUsage = np.insert(powerUsage, i, 0)
89
90          i = i + 1;
91
92      return weatherDate, temperature, powerDate, powerSupply, powerUsage
```

Step 4. 將電力資料及氣溫資料用 plot 的方式畫出。

/powerAndWeatherRelation.py:

```
106     # draw curve
107     plt.plot(weatherDate, temperature, 'r-', weatherDate, powerSupply, 'b-', weatherDate, powerUsage, 'g-')
108     plt.show()
```

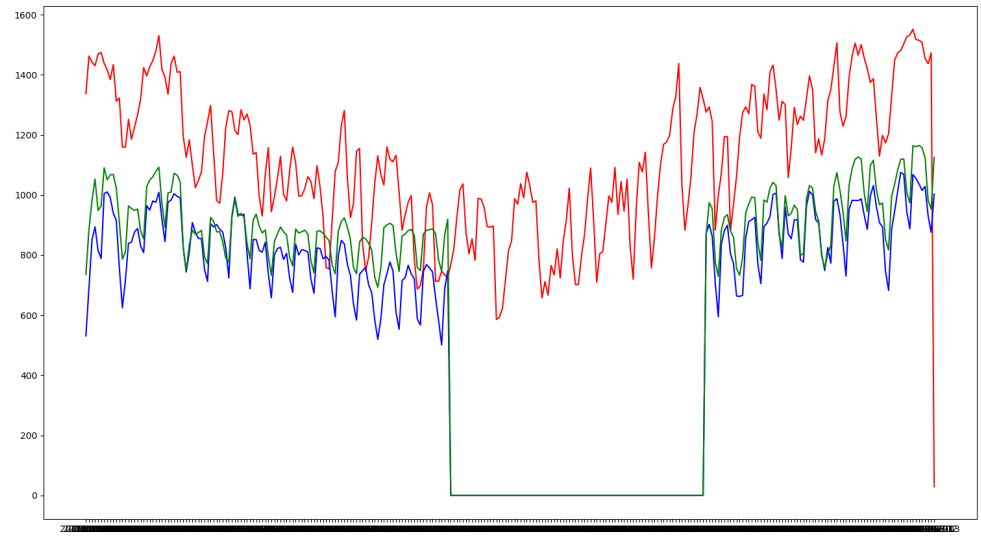
Task 3. Result.

紅色的線為氣溫，藍色的線為電力供給量，綠色的線為電力使用量。

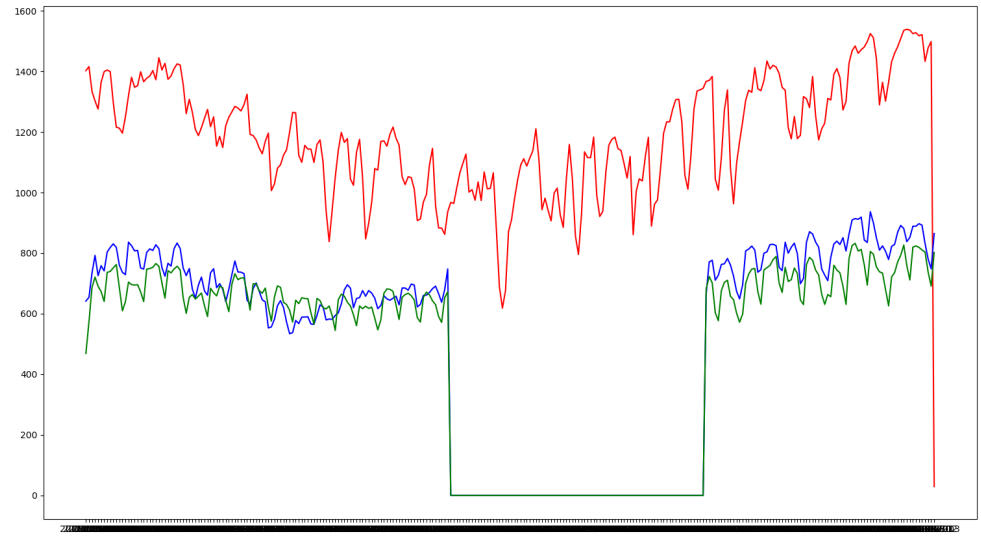
從各圖中可以發現，氣溫和電力有著大概的趨勢，當氣溫上升時，用電量和供電量上升；反之亦然，氣溫下降上升時，用電量和供電量下降。

另外，北部和東部的用電量通常大於供電量；中部和南部的用電量通常小於供電量。

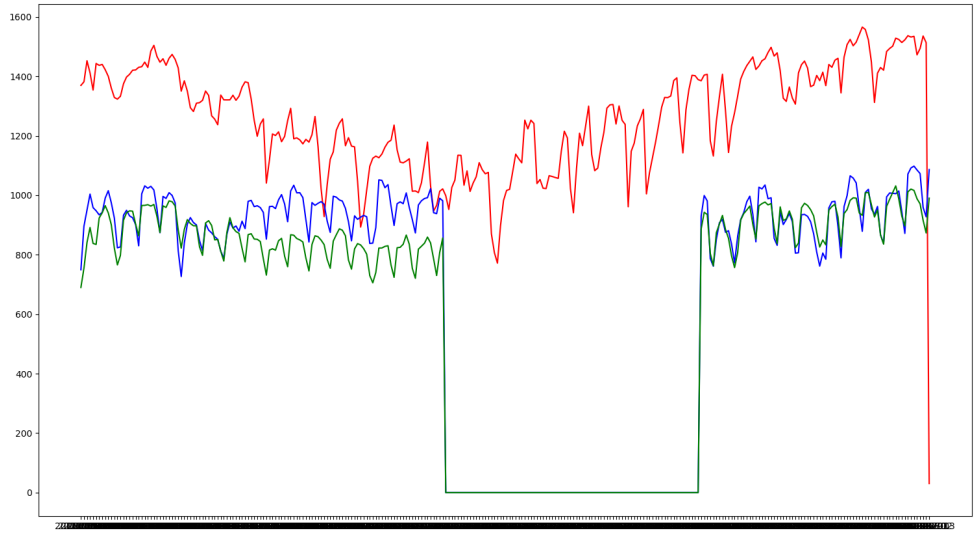
/result/task3/north.png



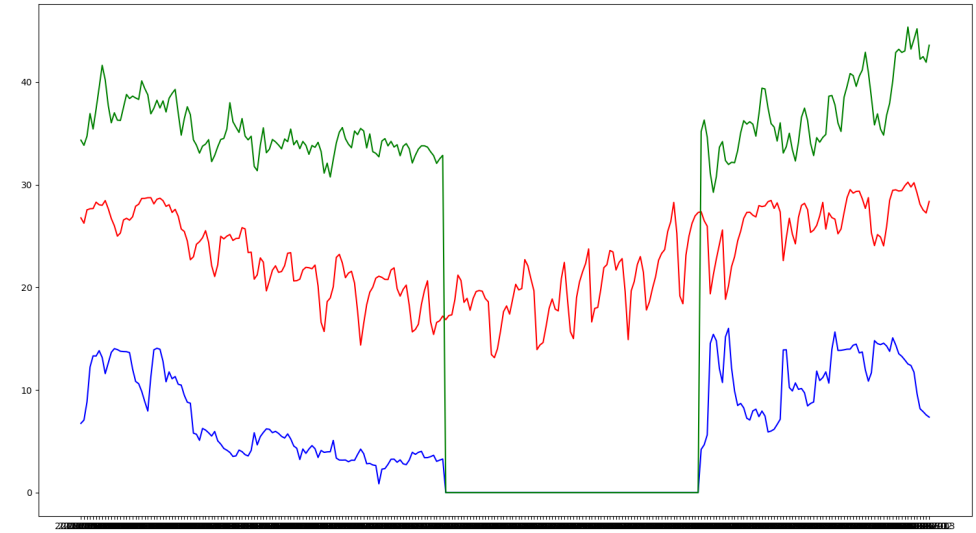
/result/task3/center.png



/result/task3/south.png



/result/task3/east.png



Task 4.

Step 1. 先決定好選取兩個區域（北 north，南 south），再從資料庫中取得北部及南部的電力資料。計算兩者的供電及用電的 Squared Euclidean distance，並將其印出。

/taipowerDatWithTransformationRule.py

```
78  # main
79  # get north power data
80  powerDate, northSupply = getPowerData('northSupply')
81  powerDate, northUsage = getPowerData('northUsage')
82
83  # get south power data
84  powerDate, southSupply = getPowerData('southSupply')
85  powerDate, southUsage = getPowerData('southUsage')
86
87  # original power supply and usage data comparation
88  powerSupplyDistance, powerUsageDistance = printDistance(northSupply, northUsage, southSupply, southUsage)
89  print 'original data:'
90  print 'Distance between north and south power supply: ', powerSupplyDistance
91  print 'Distance between north and south power usage: ', powerUsageDistance, '\n'
```

將兩者資料相減，再平方。

/taipowerDatWithTransformationRule.py:printDsitance()

```
32  # print original power data distance
33  def printDistance(northSupply, northUsage, southSupply, southUsage):
34      powerSupplyDistance = 0
35      for i in range(northSupply.size):
36          powerSupplyDistance += (northSupply[i] - southSupply[i]) * (northSupply[i] - southSupply[i])
37
38      powerUsageDistance = 0
39      # power usage comparation
40      for i in range(northUsage.size):
41          powerUsageDistance += (northUsage[i] - southUsage[i]) * (northUsage[i] - southUsage[i])
42
43      return powerSupplyDistance, powerUsageDistance
```

Step 2. 利用 Offset transformation 處理北部及南部的電力資料。再計算兩者的供電及用電的 Squared Euclidean distance，並將其印出。

/taipowerDatWithTransformationRule.py

```
94  # Offset transformation
95  meanTransNorthSupply, meanTransNorthUsage = meanTransformation(northSupply, northUsage)
96  meanTransSouthSupply, meanTransSouthUsage = meanTransformation(southSupply, southUsage)
97
98  powerSupplyDistance, powerUsageDistance = printDistance(meanTransNorthSupply, meanTransNorthUsage, meanTransSouthSuppl
99  print 'After offset transformation:'
100 print 'Distance between north and south power supply: ', powerSupplyDistance
101 print 'Distance between north and south power usage: ', powerUsageDistance, '\n'
```

將資料減去平均值相減做為位移。

/taipowerDatWithTransformationRule.py:meanTransformation()

```
45  # transform data by mean
46  def meanTransformation(supply, usage):
47      supply = np.subtract(supply, supply.mean())
48      usage = np.subtract(usage, usage.mean())
49
50      return supply, usage
```

Step 3. 利用 Amplitude Scaling transformation 處理北部及南部的電力資料。再計算兩者的供電及用電的 Squared Euclidean distance，並將其印出。

/taipowerDatWithTransformationRule.py

```
104  # Amplitude Scaling transformation
105  asTransNorthSupply, asTransNorthUsage = amplitudeScalingTransformation(northSupply, northUsage)
106  asTransSouthSupply, asTransSouthUsage = amplitudeScalingTransformation(southSupply, southUsage)
107
108  powerSupplyDistance, powerUsageDistance = printDistance(asTransNorthSupply, asTransNorthUsage, asTransSouthSupply, asT
109  print 'After amplitude scaling transformation:'
110  print 'Distance between north and south power supply: ', powerSupplyDistance
111  print 'Distance between north and south power usage: ', powerUsageDistance, '\n'
```

將資料減去平均值相減再除以標準差。

/taipowerDatWithTransformationRule.py:amplitudeScalingTransformation()

```
52  # transform data by Amplitude Scaling
53  def amplitudeScalingTransformation(supply, usage):
54      supply = np.subtract(supply, supply.mean()) / np.std(supply)
55      usage = np.subtract(usage, usage.mean()) / np.std(usage)
56
57      return supply, usage
```

Step 4. 利用 Linear Trend Removal transformation 處理北部及南部的電力資料。再計算兩者的供電及用電的 Squared Euclidean distance，並將其印出。

/taipowerDatWithTransformationRule.py

```
114  # Linear Trend Removal transformation
115  detrendTransNorthSupply, detrendTransNorthUsage = detrendTransformation(northSupply, northUsage)
116  detrendTransSouthSupply, detrendTransSouthUsage = detrendTransformation(southSupply, southUsage)
117
118  powerSupplyDistance, powerUsageDistance = printDistance(detrendTransNorthSupply, detrendTransNorthUsage, detrendTransS
119  print 'After linear trend removal transformation:'
120  print 'Distance between north and south power supply: ', powerSupplyDistance
121  print 'Distance between north and south power usage: ', powerUsageDistance, '\n'
```

引入 pyhon library 中的 from scipy import signal，再用 signal 的 detrend 方法移除 linear trend。

/taipowerDatWithTransformationRule.py:detrendTransformation()

```
59  # transform data by Linear Trend Removal
60  def detrendTransformation(supply, usage):
61      supply = signal.detrend(supply)
62      usage = signal.detrend(usage)
63
64      return supply, usage
```

Step 5. 利用 Noise Reduction transformation 處理北部及南部的電力資料。再計算兩者的供電及用電的 Squared Euclidean distance，並將其印出。

/taipowerDatWithTransformationRule.py

```
124 # Noise Reduction transformation
125 nrTransNorthSupply, nrTransNorthUsage = noiseReductionTransformation(northSupply, northUsage)
126 nrTransSouthSupply, nrTransSouthUsage = noiseReductionTransformation(southSupply, southUsage)
127
128 powerSupplyDistance, powerUsageDistance = printDistance(nrTransNorthSupply, nrTransNorthUsage, nrTransSouthSupply, nrT
129 print 'After noise reduction transformation:'
130 print 'Distance between north and south power supply: ', powerSupplyDistance
131 print 'Distance between north and south power usage: ', powerUsageDistance, '\n'
132
```

去除 noise 的方法有很多，因為電力資料只有一維，所以我選用 pyhon library 中的 from scipy.signal import lfilter，a 為分母，b 為分子，再用 lfilter 去除 noise。

/taipowerDatWithTransformationRule.py:noiseReductionTransformation()

```
66  # tranform data by Noise Reduction
67  def noiseReductionTransformation(supply, usage):
68      b = [1.0 / supply.size] * supply.size
69      a = 1
70      supply = lfilter(b, a, supply)
71
72      b = [1.0 / usage.size] * usage.size
73      usage = lfilter(b, a, usage)
74
75      return supply, usage
```

Task 4. Result.

從結果看起來，經過 amplitude scaling transformation 後，兩者的資料十分相似！

original data:
Distance between north and south power supply: 5790431.9771
Distance between north and south power usage: 2949009.03081

After offset transformation:
Distance between north and south power supply: 2711284.53046
Distance between north and south power usage: 1643763.32962

After amplitude scaling transformation:
Distance between north and south power supply: 159.809964907
Distance between north and south power usage: 50.6820752656

After linear trend removal transformation:
Distance between north and south power supply: 2705740.4477
Distance between north and south power usage: 1200566.29161

After noise reduction transformation:
Distance between north and south power supply: 1112429.72224
Distance between north and south power usage: 246717.925397

/result/task4/4.png

```
original data:
Distance between north and south power supply:  5790431.9771
Distance between north and south power usage:  2949009.03081

After offset transformation:
Distance between north and south power supply:  2711284.53046
Distance between north and south power usage:  1643763.32962

After amplitude scaling transformation:
Distance between north and south power supply:  159.809964907
Distance between north and south power usage:  50.6820752656

After linear trend removal transformation:
Distance between north and south power supply:  2705740.4477
Distance between north and south power usage:  1200566.29161

After noise reduction transformation:
Distance between north and south power supply:  1112429.72224
Distance between north and south power usage:  246717.925397
```