# 資料探勘 Homework1

學生：吳汶峻

學號：Q36071229

系級：電通甲

程式環境：Python3/Windows10

## 一、用 Sample Data 跑程式

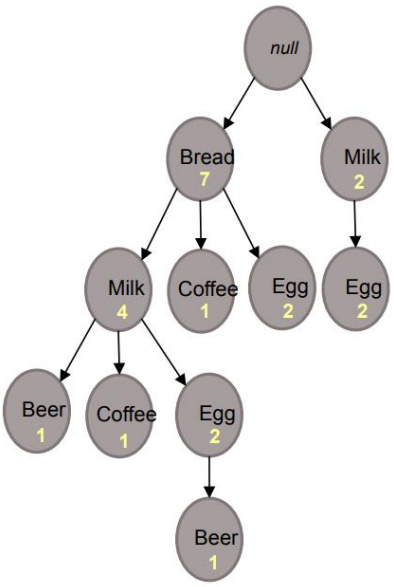
首先，在我 Github 上 FP_Growth.py 的 code 中，我設計與老師範例(FP Growth Example.pdf)相同的 Item Table 內容，如下：

| TID | Items |
|---|---|
| 1 | Bread, Milk, Beer |
| 2 | Bread, Coffee |
| 3 | Bread, Egg |
| 4 | Bread, Milk, Coffee |
| 5 | Milk, Egg |
| 6 | Bread, Egg |
| 7 | Milk, Egg |
| 8 | Bread, Milk, Egg, Beer |
| 9 | Bread, Milk, Egg |

Sample data 為：

```
def loadSimpDat():    #生成itemsets
    simpDat = [['Bread', 'Milk', 'Beer'],
               ['Bread', 'Coffee'],
               ['Bread', 'Egg'],
               ['Bread', 'Milk', 'Coffee'],
               ['Milk', 'Egg'],
               ['Bread', 'Egg'],
               ['Milk', 'Egg'],
               ['Bread', 'Milk', 'Egg', 'Beer'],
               ['Bread', 'Milk', 'Egg']]
    return simpDat
```

將 Min-support 設定為 2、產生 FP-tree 後，可以得知範例的 Tree-node 與 Count 跟程式執行結果相同：




minsup = 2

```
Null Set    1
  Bread     7
    Milk    4
      Beer      1
      Coffee    1
      Egg       2
        Beer    1
    Coffee      1
    Egg         2
  Egg       2
    Milk    2
```

(此顆 Tree 是由左到右生成，最左邊(Null Set)為 Root，左右距離相同的位於同一層，依序往右長出。)

可以利用 FP-Growth.py 中的 associate()函數將每個 Frequent item sets 與 Confidence 列舉出來，Frequent item sets 與其 Confidence 執行結果如下：

```
[{'Beer'}, {'Bread', 'Beer'}, {'Beer', 'Milk'}, {'Bread', 'Beer', 'Milk'}, {'Coffee'}, {'Coffee', 'Bread'}, {'Milk'}, {'Bread', 'Milk'},
{'Egg', 'Milk'}, {'Bread', 'Egg', 'Milk'}, {'Egg'}, {'Bread', 'Egg'}, {'Bread'}]
```

(Min-support=2 的 Frequent item sets)

```
{'Bread', 'Milk'}  >>>  {'Beer'}
item_i_count : 4
item_j_count : 2
conf : 0.5
{'Bread', 'Milk'}  >>>  {'Egg'}
item_i_count : 4
item_j_count : 2
conf : 0.5
{'Egg', 'Milk'}  >>>  {'Bread'}
item_i_count : 4
item_j_count : 2
conf : 0.5
{'Egg'}  >>>  {'Milk'}
item_i_count : 6
item_j_count : 4
conf : 0.6666666666666666
```

接著再將 Data 放進 Weka 比較其關聯法則(Associate-Rule)及 Confidence：

```
 7. [Milk=T]: 6 ==> [Bread=T]: 4   <conf:(0.67)> lift:(0.86) lev:(-0.07) conv:(0.44)
 8. [Egg=T]: 6 ==> [Bread=T]: 4   <conf:(0.67)> lift:(0.86) lev:(-0.07) conv:(0.44)
 9. [Milk=T]: 6 ==> [Egg=T]: 4   <conf:(0.67)> lift:(1) lev:(0) conv:(0.67)
10. [Egg=T]: 6 ==> [Milk=T]: 4   <conf:(0.67)> lift:(1) lev:(0) conv:(0.67)
11. [Bread=T]: 7 ==> [Milk=T]: 4   <conf:(0.57)> lift:(0.86) lev:(-0.07) conv:(0.58)
12. [Bread=T]: 7 ==> [Egg=T]: 4   <conf:(0.57)> lift:(0.86) lev:(-0.07) conv:(0.58)
13. [Bread=T, Milk=T]: 4 ==> [Egg=T]: 2   <conf:(0.5)> lift:(0.75) lev:(-0.07) conv:(0.44)
14. [Bread=T, Egg=T]: 4 ==> [Milk=T]: 2   <conf:(0.5)> lift:(0.75) lev:(-0.07) conv:(0.44)
15. [Milk=T, Egg=T]: 4 ==> [Bread=T]: 2   <conf:(0.5)> lift:(0.64) lev:(-0.12) conv:(0.3)
16. [Bread=T, Milk=T]: 4 ==> [Beer=T]: 2   <conf:(0.5)> lift:(2.25) lev:(0.12) conv:(1.04)
```

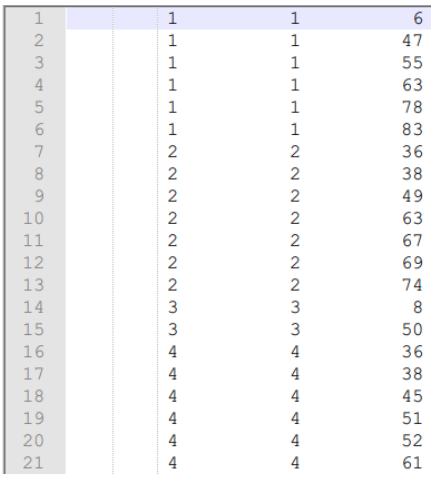
在圖中可以發現反白處的關聯法則與 Confidence 是一模一樣的，其他關聯法則及 Confidence 在做比較後發現也是一模一樣！

## 二、IBM Quest Synthetic Data Generator

接著，利用老師提供的 IBM Quest Synthetic Data Generator 產生 data 數據，這個產生器可以依據自己喜好來選擇想要的 data 長度、數量及平均每筆 item 數量，而在我的實作中，我使用以下指令：

「"IBM-Quest-Data-Generator.exe" lit -ntrans 0.1 -tlen 5 nitems 0.1」

目的是要產生 ID 總數 85、平均每個 ID 有 5 個 items，item 範圍是 0~100 的 Data。跑出來的部分 Data 如下：



| | | | |
|---|---|---|---|
| 1 | | 1 | 1 | 6 |
| 2 | | 1 | 1 | 47 |
| 3 | | 1 | 1 | 55 |
| 4 | | 1 | 1 | 63 |
| 5 | | 1 | 1 | 78 |
| 6 | | 1 | 1 | 83 |
| 7 | | 2 | 2 | 36 |
| 8 | | 2 | 2 | 38 |
| 9 | | 2 | 2 | 49 |
| 10 | | 2 | 2 | 63 |
| 11 | | 2 | 2 | 67 |
| 12 | | 2 | 2 | 69 |
| 13 | | 2 | 2 | 74 |
| 14 | | 3 | 3 | 8 |
| 15 | | 3 | 3 | 50 |
| 16 | | 4 | 4 | 36 |
| 17 | | 4 | 4 | 38 |
| 18 | | 4 | 4 | 45 |
| 19 | | 4 | 4 | 51 |
| 20 | | 4 | 4 | 52 |
| 21 | | 4 | 4 | 61 |

在思考上述 Data 想表達的涵義，並比較老師 PPT 的內容，可以推導出可能為下列情形：

| TID | Items |
|---|---|
| 1 | 6,47,55,63,78,83 |
| 2 | 36,38,49,63,67,69,74 |
| 3 | 8,50 |
| 4 | 36,38,45,51,52,61 |

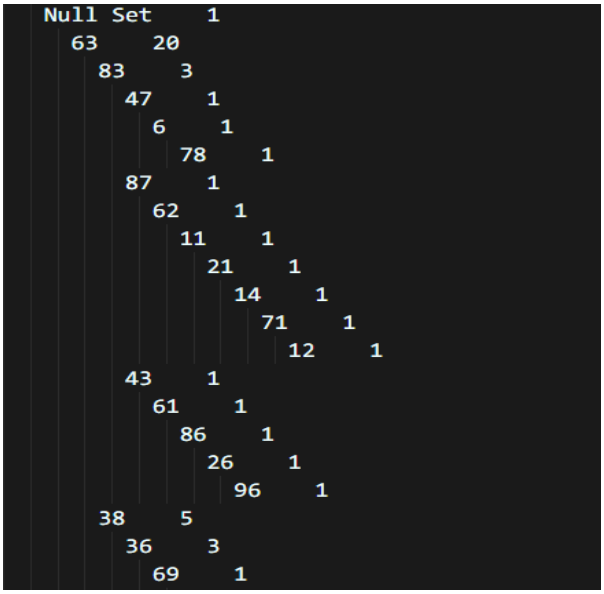
但這個 Data 格式不是我想要的，所以寫了一個 Data_conversion.py 的檔案來讓資料變得更簡潔：

```
6 47 55 63 78 83
36 38 49 63 67 69 74
8 50
36 38 45 51 52 61 62 63
0 7 42 52 81 94
4 5 6 17 36 39 41 53 55 73 78
35 38 85 87 93
11 93 97
9 11 18 35 63
8 14 42 69 87
39 40 43 59 80
7 14 21 25 40 43 83 85
8 10 38 60 69 93
36 38 63 83
8 9 38 45
```

在上面 data 中，左邊深色背景為 TID，右邊為其 Items，這樣就可以進行 FP-Growth 的動作了。

# 三、執行 FP-Growth 程式與比較結果

接著，我用 Data_read.py 開始讀檔並利用 FP-Growth.py 進行資料剖析，我將 min-support 設定為 3，找出 FP-tree 還有其 Frequent Item sets：





```
[{'62'}, {'38', '62'}, {'36', '62'}, {'63', '36', '62'}, {'63', '62'}, {'87', '63', '62'}, {'87', '62'}, {'61'}, {'38', '61'}, {'36',
'61'}, {'61', '62'}, {'43', '61'}, {'63', '61'}, {'85'}, {'85', '83'}, {'11'}, {'43'}, {'43', '83'}, {'43', '69'}, {'43', '63'}, {'83'},
{'38', '83'}, {'36', '83'}, {'63', '83'}, {'69'}, {'38', '69'}, {'87'}, {'87', '69'}, {'87', '63'}, {'87', '36'}, {'36'}, {'38', '36'},
{'38', '36', '63'}, {'63', '36'}, {'38'}, {'38', '63'}, {'63'}, {'4'}, {'18'}, {'9'}, {'25'}, {'20'}, {'20', '63'}, {'96'}, {'0'}, {'39'}
, {'97'}, {'59'}, {'13'}, {'31'}, {'66'}, {'6'}, {'78'}, {'67'}, {'38', '67'}, {'50'}, {'50', '63'}, {'45'}, {'52'}, {'36', '52'}, {'61',
'52'}, {'73'}, {'57'}, {'85', '57'}, {'12'}, {'26'}, {'63', '26'}, {'23'}, {'72'}, {'70'}, {'47'}, {'74', '47'}, {'51'}, {'36', '51'},
{'63', '51'}, {'7'}, {'42'}, {'5'}, {'93'}, {'38', '93'}, {'14'}, {'80'}, {'29'}, {'95'}, {'33'}, {'48'}, {'71'}, {'74'}, {'38', '74'},
{'36', '74'}, {'3', '74'}, {'86'}, {'61', '86'}, {'89'}, {'89', '21'}, {'17'}, {'38', '17'}, {'83', '17'}, {'35'}, {'35', '63'}, {'38',
'35'}, {'40'}, {'43', '40'}, {'61', '40'}, {'3', '40'}, {'8'}, {'8', '38'}, {'81'}, {'81', '83'}, {'21'}, {'43', '21'}, {'21', '83'},
{'21', '62'}, {'81', '21'}, {'3'}, {'3', '36'}, {'3', '61'}, {'28'}]
```

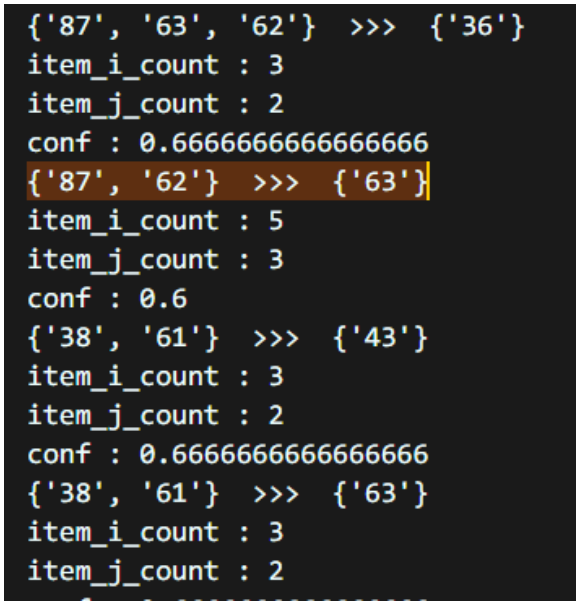

(上圖為 min-support 設為 3 的 Frequent Item sets)

然而，由於我使用的是 Weka 這個資料分析工具，它所需要的 CSV 檔格式中第一列必須為"屬性名稱"，因此我將 Data 利用 Data_convert_Weka.py 的程式轉成 Weka 承認的格式：

| | 6 | 47 | 55 | 63 | 78 | 83 | 36 | 38 |
|---|---|---|---|---|---|---|---|---|
| 1 | T | T | T | T | T | T | | |
| 2 | | | | T | | | T | T |
| 3 | | | | | | | | |
| 4 | | | | T | | | T | T |
| 5 | | | | | | | | |
| 6 | T | | T | | T | | T | |
| 7 | | | | | | | | T |
| 8 | | | | | | | | |
| 9 | | | | T | | | | |
| 10 | | | | | | | | |
| 11 | | | | | | | | |
| 12 | | | | | | T | | |
| 13 | | | | | | | | T |
| 14 | | | | T | | T | T | T |
| 15 | | | | | | | | T |
| 16 | | | | | | | | |

其中空白的部分為該 TID 沒有此數值(Null)，T 代表此 TID 有此數值。

而在 Data 丟進 Weka 跑時產生的是關聯法則及其 Confidence，所以我再利用先前產生的 Frequent Item sets 生成關聯法則及每個 Confidence，其部分執行結果為：




而將 Weka 的設定值設為 min-support=3.0、Confidence=0.5，跑出來結果為：

```
11. [50=T]: 5 ==> [63=T]: 3   <conf:(0.6)> lift:(2.55) lev:(0.02) conv:(1.27)
12. [26=T]: 5 ==> [63=T]: 3   <conf:(0.6)> lift:(2.55) lev:(0.02) conv:(1.27)
13. [67=T]: 5 ==> [38=T]: 3   <conf:(0.6)> lift:(2.83) lev:(0.02) conv:(1.31)
14. [52=T]: 5 ==> [36=T]: 3   <conf:(0.6)> lift:(3.92) lev:(0.03) conv:(1.41)
15. [57=T]: 5 ==> [85=T]: 3   <conf:(0.6)> lift:(5.1) lev:(0.03) conv:(1.47)
16. [52=T]: 5 ==> [61=T]: 3   <conf:(0.6)> lift:(5.1) lev:(0.03) conv:(1.47)
17. [63=T, 38=T]: 5 ==> [36=T]: 3   <conf:(0.6)> lift:(3.92) lev:(0.03) conv:(1.41)
18. [87=T, 62=T]: 5 ==> [63=T]: 3   <conf:(0.6)> lift:(2.55) lev:(0.02) conv:(1.27)
19. [36=T]: 13 ==> [63=T]: 7   <conf:(0.54)> lift:(2.29) lev:(0.05) conv:(1.42)
20. [35=T]: 8 ==> [38=T]: 4   <conf:(0.5)> lift:(2.36) lev:(0.03) conv:(1.26)
21. [93=T]: 6 ==> [38=T]: 3   <conf:(0.5)> lift:(2.36) lev:(0.02) conv:(1.18)
22. [62=T]: 10 ==> [87=T]: 5   <conf:(0.5)> lift:(3.54) lev:(0.04) conv:(1.43)
23. [47=T]: 6 ==> [74=T]: 3   <conf:(0.5)> lift:(6.07) lev:(0.03) conv:(1.38)
24. [51=T]: 6 ==> [63=T, 36=T]: 3   <conf:(0.5)> lift:(6.07) lev:(0.03) conv:(1.38)
25. [3=T]: 9 ==> [61=T]: 4   <conf:(0.44)> lift:(3.78) lev:(0.03) conv:(1.32)
26. [74=T]: 7 ==> [38=T]: 3   <conf:(0.43)> lift:(2.02) lev:(0.02) conv:(1.1)
27. [74=T]: 7 ==> [36=T]: 3   <conf:(0.43)> lift:(2.8) lev:(0.02) conv:(1.19)
28. [86=T]: 7 ==> [61=T]: 3   <conf:(0.43)> lift:(3.64) lev:(0.03) conv:(1.24)
29. [74=T]: 7 ==> [3=T]: 3   <conf:(0.43)> lift:(4.05) lev:(0.03) conv:(1.25)
30. [89=T]: 7 ==> [21=T]: 3   <conf:(0.43)> lift:(4.05) lev:(0.03) conv:(1.25)
```

在比較 Code 與 Weka 出來的結果，可以發現在圖中反白的畫面其 Confidence 都是 0.6，而其他的數據也可以從程式執行結果找出。由此可知，程式跑出來與 Weka 執行結果是一模一樣的！

## 四、Kaggle

在 www.kaggle.com 的網站裡找到一個適合做 Weka 的 Data(Pokemon with stats)，這是一個 pokemon 各種屬性值(Ex:HP、Attack…)的 Data。下載下來後，經過 Data Parsing，將極端值及獨立的數據刪除後，有分析出與 Weka 符合的格式：

| | Total | HP | Attack | Defense | Sp. Atk | Sp. Def |
|---|---|---|---|---|---|---|
| 2 | | | | | | |
| 3 | | | | | T | T |
| 4 | T | T | T | T | T | T |
| 5 | T | T | T | T | T | T |
| 6 | | | | | | |
| 7 | | | | | T | |
| 8 | T | T | T | T | T | T |
| 9 | T | T | T | T | T | T |
| 10 | T | T | T | T | T | T |
| 11 | | | | | | |
| 12 | | | | T | | T |
| 13 | T | T | T | T | T | T |
| 14 | T | T | T | T | T | T |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | T | T |
| 18 | | | | | | |

Weka 設定將 min-support=2、confidence>0.5，執行結果如下：

```
1. [HP=T, Attack=T]: 269 ==> [Total=T]: 247   <conf:(0.92)> lift:(1.76) lev:(0.13) conv:(5.6)
2. [Sp. Def=T]: 363 ==> [Total=T]: 306   <conf:(0.84)> lift:(1.62) lev:(0.15) conv:(3)
3. [HP=T]: 380 ==> [Total=T]: 317   <conf:(0.83)> lift:(1.6) lev:(0.15) conv:(2.84)
4. [Sp. Atk=T]: 346 ==> [Total=T]: 285   <conf:(0.82)> lift:(1.58) lev:(0.13) conv:(2.67)
5. [Attack=T]: 379 ==> [Total=T]: 307   <conf:(0.81)> lift:(1.55) lev:(0.14) conv:(2.49)
6. [Defense=T]: 352 ==> [Total=T]: 284   <conf:(0.81)> lift:(1.55) lev:(0.13) conv:(2.44)
7. [Total=T, Attack=T]: 307 ==> [HP=T]: 247   <conf:(0.8)> lift:(1.69) lev:(0.13) conv:(2.64)
8. [Total=T, HP=T]: 317 ==> [Attack=T]: 247   <conf:(0.78)> lift:(1.64) lev:(0.12) conv:(2.35)
9. [Total=T]: 417 ==> [HP=T]: 317   <conf:(0.76)> lift:(1.6) lev:(0.15) conv:(2.17)
10. [Speed=T]: 382 ==> [Total=T]: 288   <conf:(0.75)> lift:(1.45) lev:(0.11) conv:(1.93)
11. [Total=T]: 417 ==> [Attack=T]: 307   <conf:(0.74)> lift:(1.55) lev:(0.14) conv:(1.98)
12. [Total=T]: 417 ==> [Sp. Def=T]: 306   <conf:(0.73)> lift:(1.62) lev:(0.15) conv:(2.03)
13. [Defense=T]: 352 ==> [Sp. Def=T]: 258   <conf:(0.73)> lift:(1.62) lev:(0.12) conv:(2.02)
```

有顯示出關聯法則及其 Confidence，但因為這份 Data 若要放在我的程式執行的話，需要分析更多的 Data parsing 才能夠建 FP-tree，目前 Kaggle Data 分析只做到 Weka 實際分析的部分。

# 五、實作心得

剛開始聽到老師說要做 FP-Growth 的實作時，當下是很緊張的，擔心自己做不出來，但在自己查找資料、了解 FP-Tree 涵義是什麼後，其實只要肯花心思與時間去學習跟寫 code，就能學到很多有關寫程式技巧與演算法。

而在 IBM 程式產生 Data、跑入我的程式與 Weka 所展示的結果相符合後，我在 Kaggle 上雖然有找到一個關於 pokemon 特性的比較，蠻適合當 Weka 分析的 Data，但由於它的 Data parsing 有點難去分析，至今尚未能在我的程式中順利執行，還沒想到較好的解決方法，不過我會在往後的時間盡可能將其分析出來，將 Kaggle 的 Data 做出 FP-Growth，因為這是實作有魅力又能學到很多技巧的地方！