Project 1_Report (edited ver.)

P76074389 蔡雨芝

(Coding with Python 3.6.5)

1. Dataset:

(a) (Kaggle)Titanic: Machine Learning from Disaster

- 為鐵達尼號乘客名單，多被用來預測生存與否，作業一探討關聯法則，因此想說可以利用這個dataset，看看怎麼樣特性的乘客較容易生存/死亡。
- Original Data Columns: PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked
- PassengerId: 乘客編號
  Survived: 生存(1)/死亡(0)
  Pclass: 1/2/3 (數字越小，艙等級越高)
  Name: 乘客姓名(稱謂+名字)
  Sex: Male/Female/Unknown
  Age:乘客年紀(數值，有整數也有浮點數)
  SibSp:兄弟姊妹+老婆丈夫之數量
  Parch:父母小孩之數量
  Ticket:票號
  Fare:票價
  Cabin:船艙編號（多數為 N/A）
  Embarked: 登船港口(Q/S/C)
- Delete the data columns that aren't important for analyzing:

  PassengerID: 僅代表編號，與特性無關，因此刪除。

  Name: 僅代表名字，除了稱謂以外沒有太具特徵性的資料，且稱謂的種類太多，因此刪除。

  Sibsp & Parch: 家人數量，認為與生存死亡關係較不明顯，因此刪除。

  Ticket: 票號，因為票號內容多樣性太高，且沒有辦法確定數值代表什麼意義，因此刪除。

  Fare: 票價，因認為倉等(Pclass)與Fare的Feature意義類似，且Pclass分類清楚，因此刪除。

  Cabin: 船艙編號，因多數是N/A，因此刪除。
- **Final Data Columns: Survived, Pclass, Sex, Age, Embarked**
  - ✧ **16 item(s), 891 transaction(s)**
  - ✧ 資料前處理：
    - ◆ Survived:‘Survived’(1), ‘Not_Survived’(0), ‘Not_sure_if_survived’(N/A)
    - ◆ Pclass: ‘Pclass1’(1), ‘Pclass2’(2), ‘Pclass3’(3)
    - ◆ Sex: ‘Male’, ‘Female’, ‘Not_Sure_Sex’
    - ◆ ‘Age’: (split into 5 range class)
      ‘Kids(0~8)’, ‘Teenager(9~18)’, ‘Adult(19~40)’, ‘Middle(41~65)’, ‘Elder(66~)’
    - ◆ Embarked: 'Embarked_at_C'(C), 'Embarked_at_S'(S), 'Embarked_at_Q'(Q), 'Not_Sure_Where_the_Passenger_Embarked_at'(N/A)

(b) IBM Quest Data Generator dataset

- 利用 Generator 產生之 Dataset，為了比較不同的資料量、item 數、transaction 數，因此分為以下三種 dataset:
  - **[nitems_0.1, ntrans_0.1] 93 item(s), 99 transaction(s)**
    - Number of items = 100
    - Number of transactions in database = 100
    - Avg transaction len = 10
    - 100 個 items、100 筆 transaction、平均 transaction 長度為 10
  - **[nitems_0.1, ntrans_1] 99 item(s), 984 transaction(s)**
    - Number of items = 100
    - Number of transactions in database = 1000
    - Avg transaction len = 10
    - 100 個 items、1000 筆 transaction、平均 transaction 長度為 10
  - **[nitems_1, ntrans_0.1] 499 item(s), 984 transaction(s)**
    - Number of items = 1000
    - Number of transactions in database = 100
    - Avg transaction len = 10
    - 1000 個 items、100 筆 transaction、平均 transaction 長度為 10

---

2. **Inplement Apriori/Hash-Tree/FP-Growth algorithm**

- **執行方式：**python3 HW1-new.py (must in "code" folder)
- 在 "results" 資料夾中有儲存執行python檔的output結果，分別以dataset_(No.)_(Algorithm).txt 存檔。
- **Apriori:**
  - 實作方法：

    (1) 建立一個 Apriori 的 Class，儲存 min_sup、min_conf（預設為 0）。

    (2) 將 transactions 丟入 fit()函式，經過排序後，尋找 frequent patterns（若沒有經過排序，會造成 support 次數一樣的 item 可能因順序不同被視為 pattern 不同）。

    (3) 在 fit()中，先以 get_freq()函式，找出 frequent-1-term itemset，再以 frequent-1-term itemset 作為 candidate itemset，丟入 get_freq()後，找出 frequent-2-term itemset，再逐次擴展至 frequent-3-term itemset、frequent-4-term itemset、...(term 長度逐次增加)，直到 frequent-N-term itemset 為空集合時停止，此時所有符合 support >= min_sup 的 frequent patterns 都已被召回。

    (4) 最後 return 紀錄 frequent patterns 的 set 以及對應每個 frequent pattern support 的 dictionary，即為最後得到的 frequent patterns。

- **FP-Growth:**
  - 實作方法：
    (1) 建立一個 FPTree 的 Class，儲存 root_node、routes(Head, Tail)。
    (2) 建立一個 FPNode 的 Class，儲存 tree、item、count、nodeLink、parent、children。
    (3) 將 transactions、min_sup 丟入 find_frequent_items()函式，先計算每個 1-term itemset 的 support'存放在 data_dict 中'並將 < min_sup 的 item 刪除。
    (4) 以 sorting 過的 frequent-1-term 建立 FPTree（若沒有經過 sorting 會造成 support 次數一樣的 item 可能因順序不同被視為 pattern 不同）。
    (5) 以 conditional_tree_from_paths()函式，將 FPTree 的每個 node 的 prefix_path 丟入，建立 conditional FPTree。
    (6) 建立完 conditional FPTree 後，mining recursively on such tree。
    (7) 以 suffix（初始化為[]）逐次增加的方式去尋找 frequent itemsets，將 FPTree 與 suffix 丟入 find_with_suffix()'依序建立 conditional tree 並 mining，得到 frequent itemsets。
    (8) 當整個 FPTree 都尋找完，最終得到的結果即為 frequent patterns。

- **Hash-Tree:**
  - 實作方法：
    (1) 建立一個 HashNode 的 Class，儲存 children、isLeaf、bucket。
    (2) 建立一個 HashTree 的 Class，儲存 root node、max_leaf_size、max_child_size、frequent_itemsets。建立一個 Apriori_HashTree 的 Class，儲存 min_sup、min_conf（預設為 0）。
    (3) Hash function: 因為可能有 node 值不是 int 而是 string 的狀況，因此以 string 中的每個 char 之 ascii code 值做加總，並將此值表達為 4 位數字後，將第 3 位數及第 4 位數（最後 2 位數）相加後平方，最後 mod(%) max_child_size 後即為 hash 值。
    (4) Apriori_HashTree 中的函式基本上與 Apriori 相同，差別在於 fit()部分，將 candidate itemsets 存放在 Hash Tree。
    (5) 將 transactions 丟入 fit()函式，與 Apriori 相同，先以 get_freq()函式'找出 frequent-1-term itemset'再以其作為 candidate itemset'並且以經過排序的 frequent-1-term itemset 建立 HashTree（若沒有經過排序，可能會造成 support 次數一樣的 item 可能因順序不同被視為 pattern 不同）。此外，利用 freq_term_set 紀錄 frequent patterns。

(6) 以 frequent-1-term itemset 建立 HashTree 後，擴增至 frequent-2-term itemset，更新所有 transactions 的 2-term-subsets 在 HashTree 之 support 次數。更新後，利用 get_freq()找出 frequent-2-term itemset，並更新 frequent-2-term itemset 至 freq_term_set。

(7) 再逐次擴展至 frequent-3-term itemset、frequent-4-term itemset、...(term 長度逐次增加)，直到 frequent-N-term itemset 為空集合時停止，此時所有符合 support >= min_sup 的 frequent patterns(freq_term_set) 都已被召回。

(a) (Kaggle)Titanic: Machine Learning from Disaster

◆ **[16 item(s), 891 transaction(s)], min support = 20%**

✧ **(i) Apriori (Spent Time:0.058988094329833984 s)**

Frequent 1-term set with min_support(178):

```
['Embarked_at_S'] :  644
['Male'] :  577
['Not_Survived'] :  549
['Pclass3'] :  491
['Adult(19~40)'] :  427
['Survived'] :  342
['Female'] :  314
['Kids(0~8)'] :  231
['Pclass1'] :  216
['Pclass2'] :  184
```

Frequent 2-term set with min_support(178):

```
['Male', 'Not_Survived'] :  468
['Embarked_at_S', 'Male'] :  441
['Embarked_at_S', 'Not_Survived'] :  427
['Pclass3', 'Not_Survived'] :  372
['Embarked_at_S', 'Pclass3'] :  353
['Male', 'Pclass3'] :  347
['Adult(19~40)', 'Embarked_at_S'] :  339
['Adult(19~40)', 'Male'] :  282
['Adult(19~40)', 'Not_Survived'] :  262
['Survived', 'Female'] :  233
['Adult(19~40)', 'Pclass3'] :  223
['Embarked_at_S', 'Survived'] :  217
['Embarked_at_S', 'Female'] :  203
```

Frequent 3-term set with min_support(178):

```
['Embarked_at_S', 'Male', 'Not_Survived'] :  364
['Male', 'Pclass3', 'Not_Survived'] :  300
['Embarked_at_S', 'Pclass3', 'Not_Survived'] :  286
['Embarked_at_S', 'Male', 'Pclass3'] :  265
['Adult(19~40)', 'Male', 'Not_Survived'] :  231
['Adult(19~40)', 'Embarked_at_S', 'Male'] :  229
['Adult(19~40)', 'Embarked_at_S', 'Not_Survived'] :  221
['Adult(19~40)', 'Embarked_at_S', 'Pclass3'] :  188
```

Frequent 4-term set with min_support(178):

```
['Embarked_at_S', 'Not_Survived', 'Male', 'Pclass3'] :  231
['Embarked_at_S', 'Not_Survived', 'Adult(19~40)', 'Male'] :  194
```

✧ **(ii) Apriori with Hash Tree (Spent Time:0.3384120464324951 s)**

Frequent Patterns whose appearance is >= min_support(178):

```
['Embarked_at_S'] :  644
['Male'] :  577
['Not_Survived'] :  549
['Pclass3'] :  491
['Not_Survived', 'Male'] :  468
['Male', 'Embarked_at_S'] :  441
['Not_Survived', 'Embarked_at_S'] :  427
['Adult(19~40)'] :  427
['Pclass3', 'Not_Survived'] :  372
['Not_Survived', 'Male',
```

```
'Embarked_at_S'] :  364                    'Adult(19~40)'] :  231
  ['Pclass3', 'Embarked_at_S'] :  353        ['Kids(0~8)'] :  231
  ['Pclass3', 'Male'] :  347                 ['Male', 'Embarked_at_S',
  ['Survived'] :  342                      'Adult(19~40)'] :  229
  ['Embarked_at_S', 'Adult(19~40)'] : 339    ['Pclass3', 'Adult(19~40)'] :  223
  ['Female'] :  314                          ['Not_Survived', 'Embarked_at_S',
  ['Pclass3', 'Not_Survived', 'Male']: 300 'Adult(19~40)'] :  221
  ['Pclass3', 'Not_Survived',                ['Survived', 'Embarked_at_S'] :  217
'Embarked_at_S'] :  286                      ['Pclass1'] :  216
  ['Male', 'Adult(19~40)'] :  282            ['Female', 'Embarked_at_S'] :  203
  ['Pclass3', 'Male', 'Embarked_at_S']:265   ['Not_Survived', 'Male',
  ['Not_Survived', 'Adult(19~40)'] :  262  'Embarked_at_S', 'Adult(19~40)'] :  194
  ['Survived', 'Female'] :  233              ['Pclass3', 'Embarked_at_S',
  ['Pclass3', 'Not_Survived', 'Male',      'Adult(19~40)'] : 188
'Embarked_at_S'] :  231                      ['Pclass2'] :  184
  ['Not_Survived', 'Male',
```

- ✧ **(iii) FP-Growth (Spent Time:0.011029958724975586 s)**

Frequent Patterns whose appearance is >= min_support(178):

```
  ['Embarked_at_S'] :  644                   ['Not_Survived', 'Adult(19~40)'] :  262
  ['Male'] :  577                            ['Survived', 'Female'] :  233
  ['Not_Survived'] :  549                    ['Male', 'Not_Survived',
  ['Pclass3'] :  491                       'Adult(19~40)'] :  231
  ['Male', 'Not_Survived'] :  468            ['Kids(0~8)'] :  231
  ['Embarked_at_S', 'Male'] :  441           ['Embarked_at_S', 'Male',
  ['Embarked_at_S', 'Not_Survived'] :  427 'Not_Survived', 'Pclass3'] :  231
  ['Adult(19~40)'] :  427                    ['Embarked_at_S', 'Male',
  ['Not_Survived', 'Pclass3'] :  372       'Adult(19~40)'] :  229
  ['Embarked_at_S', 'Male',                  ['Pclass3', 'Adult(19~40)'] :  223
'Not_Survived'] :  364                       ['Embarked_at_S', 'Not_Survived',
  ['Embarked_at_S', 'Pclass3'] :  353      'Adult(19~40)'] :  221
  ['Male', 'Pclass3'] :  347                 ['Embarked_at_S', 'Survived'] :  217
  ['Survived'] :  342                        ['Pclass1'] :  216
  ['Embarked_at_S', 'Adult(19~40)'] : 339    ['Embarked_at_S', 'Female'] :  203
  ['Female'] :  314                          ['Embarked_at_S', 'Male',
  ['Male', 'Not_Survived', 'Pclass3']: 300 'Not_Survived', 'Adult(19~40)'] :  194
  ['Embarked_at_S', 'Not_Survived',          ['Embarked_at_S', 'Pclass3',
'Pclass3'] :  286                          'Adult(19~40)'] :  188
  ['Male', 'Adult(19~40)'] :  282            ['Pclass2'] :  184
  ['Embarked_at_S', 'Male', 'Pclass3']:265
```

- **討論：（針對生存與否）（黃底標示部分）**
  - 男性死亡率極高（約81%），女性生存率約74%。[1]
  - 在S港口上船的人數最多，但其中約66%的人死亡(為登船港口中最多)，其中男性又佔約85%。[2]
  - (承上)可能的原因為在S港口上船的人有71%都是PClass3(最低等級船艙)，其中81%死亡。[3]
  - 在PClass3(最低等級船艙)中死亡的人中有76%是從S港口上船的。[4]
  - Apriori、Hash-Tree、FP-Growth三種演算法，產生出來的Frequent Patterns是相同的，然而從Time cost中可以看出，FP-Growth的運算時間是最少的，其次是Apriori，而Hash-Tree

[1] ['Male'] 577, ['Male', 'Not_Survived'] 468 ; ['Female'] 314, ['Survived', 'Female'] 233

[2] ['Embarked_at_S'] 644, ['Embarked_at_S', 'Not_Survived'] 427, ['Embarked_at_S', 'Male', 'Not_Survived'] 364

[3] ['Embarked_at_S'] 644, ['Embarked_at_S', 'Pclass3'] 353, ['Embarked_at_S', 'Not_Survived', 'Pclass3'] 286

[4] ['Not_Survived', 'Pclass3'] 372, ['Embarked_at_S', 'Not_Survived', 'Pclass3'] 286

的時間花費最多。老師上課有提過，FP-Growth是基於Apriori優化過的演算法，因此運算速度較快、時間成本較低；另外Hash-Tree不見得比較節省時間，因為如果itemset太多，tree會很深，這樣一來其實跟線性比對的結果差不了多少，可能因為Titanic這個dataset的itemset較多，所以在time cost上，Hash-Tree並沒有達到好的效果。

(b) IBM Quest Data Generator dataset

◆ [nitems_0.1, ntrans_0.1], min support = 15%

✧ (i) Apriori (Spent Time:0.05232405662536621 s)

```
Frequent 1-term set with min_support(14):
['item'] count
------------------------------
['38'] :  43          ['81'] :  21          ['80'] :  17          ['89'] :  15
['63'] :  32          ['8'] :  21           ['35'] :  17          ['74'] :  15
['87'] :  27          ['17'] :  20          ['3'] :  17           ['39'] :  15
['36'] :  27          ['11'] :  20          ['72'] :  16          ['86'] :  14
['69'] :  26          ['43'] :  19          ['61'] :  16          ['62'] :  14
['48'] :  24          ['28'] :  19          ['40'] :  16          ['14'] :  14
['85'] :  22          ['83'] :  18          ['93'] :  15          ['12'] :  14

Frequent 2-term set with min_support(14):
['item'] count
------------------------------
['63', '38'] :  16          ['48', '38'] :  16          ['38', '36'] :  15
```

✧ (ii) Apriori with Hash Tree (Spent Time:0.07640194892883301 s)

```
Frequent Patterns whose appearance is >= min_support(14):
  ['38'] :  43                ['43'] :  19                ['93'] :  15
  ['63'] :  32                ['28'] :  19                ['89'] :  15
  ['87'] :  27                ['83'] :  18                ['74'] :  15
  ['36'] :  27                ['80'] :  17                ['39'] :  15
  ['69'] :  26                ['35'] :  17                ['38', '36'] :  15
  ['48'] :  24                ['3'] :  17                 ['86'] :  14
  ['85'] :  22                ['72'] :  16                ['62'] :  14
  ['81'] :  21                ['63', '38'] :  16          ['14'] :  14
  ['8'] :  21                 ['61'] :  16                ['12'] :  14
  ['17'] :  20                ['48', '38'] :  16
  ['11'] :  20                ['40'] :  16
```

✧ (iii) FP-Growth (Spent Time:0.02343893051147461 s)

```
Frequent Patterns whose appearance is >= min_support(14):
  ['38'] :  43                ['43'] :  19                ['93'] :  15
  ['63'] :  32                ['28'] :  19                ['89'] :  15
  ['87'] :  27                ['83'] :  18                ['74'] :  15
  ['36'] :  27                ['80'] :  17                ['39'] :  15
  ['69'] :  26                ['35'] :  17                ['38', '36'] :  15
  ['48'] :  24                ['3'] :  17                 ['86'] :  14
  ['85'] :  22                ['72'] :  16                ['62'] :  14
  ['81'] :  21                ['61'] :  16                ['14'] :  14
  ['8'] :  21                 ['40'] :  16                ['12'] :  14
  ['17'] :  20                ['38', '63'] :  16
  ['11'] :  20                ['38', '48'] :  16
```

◆ [nitems_0.1, ntrans_1], min support = 20%

✧ (i) Apriori (Spent Time:0.9091930389404297 s)

```
Frequent 1-term set with min_support(98):
------------------------------
['38'] :  373          ['17'] :  194          ['14'] :  154          ['71'] :  133
['63'] :  358          ['47'] :  176          ['61'] :  153          ['35'] :  132
['87'] :  348          ['11'] :  176          ['86'] :  152          ['93'] :  129
['69'] :  298          ['83'] :  167          ['62'] :  150          ['89'] :  128
['36'] :  268          ['80'] :  166          ['67'] :  147          ['73'] :  128
['8'] :  227           ['43'] :  162          ['78'] :  142          ['7'] :  113
['48'] :  226          ['39'] :  157          ['29'] :  142          ['52'] :  109
['28'] :  207          ['9'] :  155           ['23'] :  138          ['95'] :  106
['85'] :  198          ['81'] :  155          ['72'] :  136          ['21'] :  104
['3'] :  196           ['40'] :  155          ['51'] :  136

Frequent 2-term set with min_support(98):
------------------------------
['38', '87'] :  145          ['63', '69'] :  116          ['38', '36'] :  107
['63', '38'] :  140          ['38', '69'] :  111          ['36', '87'] :  102
['63', '87'] :  134          ['63', '36'] :  107          ['87', '69'] :  101
```

✧ (ii) Apriori with Hash Tree (Spent Time:1.367096185684204 s)

```
Frequent Patterns whose appearance is >= min_support(98):
  ['38'] :  373                ['39'] :  157                ['87', '63'] :  134
  ['63'] :  358                ['9'] :  155                 ['71'] :  133
  ['87'] :  348                ['81'] :  155                ['35'] :  132
  ['69'] :  298                ['40'] :  155                ['93'] :  129
  ['36'] :  268                ['14'] :  154                ['89'] :  128
  ['8'] :  227                 ['61'] :  153                ['73'] :  128
  ['48'] :  226                ['86'] :  152                ['69', '63'] :  116
  ['28'] :  207                ['62'] :  150                ['7'] :  113
  ['85'] :  198                ['67'] :  147                ['69', '38'] :  111
  ['3'] :  196                 ['87', '38'] :  145          ['52'] :  109
  ['17'] :  194                ['78'] :  142                ['63', '36'] :  107
  ['47'] :  176                ['29'] :  142                ['38', '36'] :  107
  ['11'] :  176                ['63', '38'] :  140          ['95'] :  106
  ['83'] :  167                ['23'] :  138                ['21'] :  104
  ['80'] :  166                ['72'] :  136                ['87', '36'] :  102
  ['43'] :  162                ['51'] :  136                ['87', '69'] :  101
```

✧ (iii) FP-Growth (Spent Time:0.19161605834960938 s)

```
Frequent Patterns whose appearance is >= min_support(98):
  ['38'] :  373                ['80'] :  166                ['38', '63'] :  140
  ['63'] :  358                ['43'] :  162                ['23'] :  138
  ['87'] :  348                ['39'] :  157                ['72'] :  136
  ['69'] :  298                ['9'] :  155                 ['51'] :  136
  ['36'] :  268                ['81'] :  155                ['63', '87'] :  134
  ['8'] :  227                 ['40'] :  155                ['71'] :  133
  ['48'] :  226                ['14'] :  154                ['35'] :  132
  ['28'] :  207                ['61'] :  153                ['93'] :  129
  ['85'] :  198                ['86'] :  152                ['89'] :  128
  ['3'] :  196                 ['62'] :  150                ['73'] :  128
  ['17'] :  194                ['67'] :  147                ['63', '69'] :  116
  ['47'] :  176                ['38', '87'] :  145          ['7'] :  113
  ['11'] :  176                ['78'] :  142                ['38', '69'] :  111
  ['83'] :  167                ['29'] :  142                ['52'] :  109
```

['63', '36'] : 107 ['95'] : 106 ['87', '36'] : 102
['38', '36'] : 107 ['21'] : 104 ['87', '69'] : 101

◆ [nitems_1, ntrans_0.1], min support = 5%

✧ (i) Apriori (Spent Time:0.29866695404052734 s)

```
Frequent 1-term set with min_support(4):
------------------------------
['124'] : 9          ['790'] : 5          ['909'] : 4          ['553'] : 4
['416'] : 8          ['772'] : 5          ['868'] : 4          ['486'] : 4
['806'] : 7          ['673'] : 5          ['827'] : 4          ['459'] : 4
['778'] : 7          ['60'] : 5           ['815'] : 4          ['444'] : 4
['682'] : 7          ['578'] : 5          ['744'] : 4          ['439'] : 4
['592'] : 7          ['38'] : 5           ['737'] : 4          ['432'] : 4
['988'] : 6          ['36'] : 5           ['733'] : 4          ['425'] : 4
['874'] : 6          ['316'] : 5          ['709'] : 4          ['395'] : 4
['707'] : 6          ['306'] : 5          ['7'] : 4            ['371'] : 4
['456'] : 6          ['238'] : 5          ['69'] : 4           ['368'] : 4
['214'] : 6          ['132'] : 5          ['629'] : 4          ['360'] : 4
['994'] : 5          ['127'] : 5          ['628'] : 4          ['325'] : 4
['981'] : 5          ['123'] : 5          ['62'] : 4           ['246'] : 4
['934'] : 5          ['119'] : 5          ['613'] : 4          ['221'] : 4
['858'] : 5          ['117'] : 5          ['607'] : 4          ['209'] : 4
['837'] : 5          ['970'] : 4          ['601'] : 4          ['173'] : 4
['820'] : 5          ['959'] : 4          ['599'] : 4          ['154'] : 4
['813'] : 5          ['946'] : 4          ['559'] : 4
```

✧ (ii) Apriori with Hash Tree (Spent Time:0.07738399505615234 s)

```
Frequent Patterns whose appearance is >= min_support(4):
['124'] : 9          ['790'] : 5          ['909'] : 4          ['553'] : 4
['416'] : 8          ['772'] : 5          ['868'] : 4          ['486'] : 4
['806'] : 7          ['673'] : 5          ['827'] : 4          ['459'] : 4
['778'] : 7          ['60'] : 5           ['815'] : 4          ['444'] : 4
['682'] : 7          ['578'] : 5          ['744'] : 4          ['439'] : 4
['592'] : 7          ['38'] : 5           ['737'] : 4          ['432'] : 4
['988'] : 6          ['36'] : 5           ['733'] : 4          ['425'] : 4
['874'] : 6          ['316'] : 5          ['709'] : 4          ['395'] : 4
['707'] : 6          ['306'] : 5          ['7'] : 4            ['371'] : 4
['456'] : 6          ['238'] : 5          ['69'] : 4           ['368'] : 4
['214'] : 6          ['132'] : 5          ['629'] : 4          ['360'] : 4
['994'] : 5          ['127'] : 5          ['628'] : 4          ['325'] : 4
['981'] : 5          ['123'] : 5          ['62'] : 4           ['246'] : 4
['934'] : 5          ['119'] : 5          ['613'] : 4          ['221'] : 4
['858'] : 5          ['117'] : 5          ['607'] : 4          ['209'] : 4
['837'] : 5          ['970'] : 4          ['601'] : 4          ['173'] : 4
['820'] : 5          ['959'] : 4          ['599'] : 4          ['154'] : 4
['813'] : 5          ['946'] : 4          ['559'] : 4
```

✧ (iii) FP-Growth (Spent Time:0.009912252426147461 s)

```
Frequent Patterns whose appearance is >= min_support(4):
```

['124'] : 9
['416'] : 8
['806'] : 7
['778'] : 7
['682'] : 7
['592'] : 7
['988'] : 6
['874'] : 6
['707'] : 6
['456'] : 6
['214'] : 6
['994'] : 5
['981'] : 5
['934'] : 5
['858'] : 5
['837'] : 5
['820'] : 5
['813'] : 5
['790'] : 5
['772'] : 5
['673'] : 5
['60'] : 5
['578'] : 5
['38'] : 5
['36'] : 5
['316'] : 5
['306'] : 5
['238'] : 5
['132'] : 5
['127'] : 5
['123'] : 5
['119'] : 5
['117'] : 5
['970'] : 4
['959'] : 4
['946'] : 4
['909'] : 4
['868'] : 4
['827'] : 4
['815'] : 4
['744'] : 4
['737'] : 4
['733'] : 4
['709'] : 4
['7'] : 4
['69'] : 4
['629'] : 4
['628'] : 4
['62'] : 4
['613'] : 4
['607'] : 4
['601'] : 4
['599'] : 4
['559'] : 4
['553'] : 4
['486'] : 4
['459'] : 4
['444'] : 4
['439'] : 4
['432'] : 4
['425'] : 4
['395'] : 4
['371'] : 4
['368'] : 4
['360'] : 4
['325'] : 4
['246'] : 4
['221'] : 4
['209'] : 4
['173'] : 4
['154'] : 4

- **討論:**
  - 第三筆(nitems_1, ntrans_0.1)最後得到的frequent patterns都只有一個element，推測原因為總item數有1000個，但transaction只有100個（平均每筆含10items），因此重複性太低，造成support總體而言偏低，min_support value也需要降低。
  - 第二筆(nitems_0.1, ntrans_1)和第一筆(nitems_0.1, ntrans_0.1)比較，可以看出因為transaction數上升(100->1000)，因此重複性提升，frequent patterns中出現大於1個element的數量變多了（以黃底標示），min_support也提高許多。

2.Use association analysis tools to generate association rules

使用 Christian Borgelt 之 Apriori Tools https://www.filecluster.com/Apriori.html

(a) (Kaggle)Titanic: Machine Learning from Disaster

✧ Frequent patterns with same min_support(20%) as above (item (Sup%))

Embarked_at_S (72.2783)
Embarked_at_S Male (49.4949)
Embarked_at_S Male Not_Survived (40.853)
Embarked_at_S Male Not_Survived Pclass3 (25.9259)
Embarked_at_S Male Not_Survived Adult(19~40) (21.7733)
Embarked_at_S Male Pclass3 (29.7419)
Embarked_at_S Male Adult(19~40) (25.7015)
Embarked_at_S Not_Survived (47.9237)
Embarked_at_S Not_Survived Pclass3 (32.0988)
Embarked_at_S Not_Survived Adult(19~40) (24.8036)
Embarked_at_S Pclass3 (39.6184)
Embarked_at_S Pclass3 Adult(19~40) (21.0999)
Embarked_at_S Adult(19~40) (38.0471)
Embarked_at_S Survived (24.3547)
Embarked_at_S Female (22.7834)
Male (64.7587)
Male Not_Survived (52.5253)
Male Not_Survived Pclass3 (33.67)
Male Not_Survived Adult(19~40) (25.9259)
Male Pclass3 (38.945)
Male Adult(19~40) (31.6498)
Not_Survived (61.6162)
Not_Survived Pclass3 (41.7508)
Not_Survived Adult(19~40) (29.4052)
Pclass3 (55.1066)
Pclass3 Adult(19~40) (25.0281)
Adult(19~40) (47.9237)
Survived (38.3838)
Survived Female (26.1504)
Female (35.2413)
Kids(0~8) (25.9259)
Pclass1 (24.2424)
Pclass2 (20.651)

✧ **Association Rules,** min_support ≥ 40%, min_confidence ≥ 60% (Sup%, Conf%)

Not_Survived <- Embarked_at_S Male(40.853,82.5397)
Male <- Embarked_at_S Not_Survived(40.853,85.2459)

```
Embarked_at_S <- Male
Not_Survived(40.853,77.7778)
Male <- Embarked_at_S(49.4949,68.4783)
Embarked_at_S <- Male(49.4949,76.4298)
Not_Survived <- Embarked_at_S(47.9237,66.3043)
Embarked_at_S <- Not_Survived(47.9237,77.7778)
Embarked_at_S <- (72.2783,72.2783)
```

```
Not_Survived <- Male(52.5253,81.1092)
Male <- Not_Survived(52.5253,85.2459)
Male <- (64.7587,64.7587)
Pclass3 <- Not_Survived(41.7508,67.7596)
Not_Survived <- Pclass3(41.7508,75.7637)
Not_Survived <- (61.6162,61.6162)
```

- **討論:**
  - 觀察到男性佔64.8%，事件死亡率為61.6%，S港口上船比例72.3%。
  - [男性-S港口上船]、[S港口上船-死亡]、[男性-死亡]、[S港口上船-男性-死亡]相互關聯緊密。
  - 若乘客死亡，很可能是位於最低等級船艙(PClass3)，也很可能是男性。

(b) IBM Quest Data Generator dataset

◆ **[nitems_0.1, ntrans_0.1], min support = 15%,** Frequent patterns (item (Sup%))

38 (43.4343)	36 (27.2727)	11 (20.202)	3 (17.1717)	93 (15.1515)
38 63 (16.1616)	69 (26.2626)	17 (20.202)	80 (17.1717)	39 (15.1515)
38 36 (15.1515)	48 (24.2424)	28 (19.1919)	61 (16.1616)	74 (15.1515)
38 48 (16.1616)	85 (22.2222)	43 (19.1919)	40 (16.1616)	
63 (32.3232)	81 (21.2121)	83 (18.1818)	72 (16.1616)	
87 (27.2727)	8 (21.2121)	35 (17.1717)	89 (15.1515)	

◆ **[nitems_0.1, ntrans_1], min support = 10%,** Frequent patterns (item (Sup%))

38 (37.9065)	87 69 (10.2642)	47 (17.8862)	61 (15.5488)	35 (13.4146)
38 63 (14.2276)	87 36 (10.3659)	11 (17.8862)	86 (15.4472)	93 (13.1098)
38 87 (14.7358)	69 (30.2846)	83 (16.9715)	62 (15.2439)	89 (13.0081)
38 69 (11.2805)	36 (27.2358)	80 (16.8699)	67 (14.939)	73 (13.0081)
38 36 (10.874)	8 (23.0691)	43 (16.4634)	29 (14.4309)	7 (11.4837)
63 (36.3821)	48 (22.9675)	39 (15.9553)	78 (14.4309)	52 (11.0772)
63 87 (13.6179)	28 (21.0366)	81 (15.752)	23 (14.0244)	95 (10.7724)
63 69 (11.7886)	85 (20.122)	9 (15.752)	72 (13.8211)	21 (10.5691)
63 36 (10.874)	3 (19.9187)	40 (15.752)	51 (13.8211)	
87 (35.3659)	17 (19.7154)	14 (15.6504)	71 (13.5163)	

◆ **[nitems_1, ntrans_0.1], min support = 5%,** Frequent patterns (item (Sup%))

124 (9.09091)	707 (6.06061)	117 (5.05051)	813 (5.05051)	790 (5.05051)
416 (8.08081)	214 (6.06061)	316 (5.05051)	36 (5.05051)	60 (5.05051)
806 (7.07071)	988 (6.06061)	981 (5.05051)	934 (5.05051)	837 (5.05051)
778 (7.07071)	456 (6.06061)	119 (5.05051)	820 (5.05051)	858 (5.05051)
592 (7.07071)	238 (5.05051)	127 (5.05051)	673 (5.05051)	123 (5.05051)
682 (7.07071)	132 (5.05051)	994 (5.05051)	38 (5.05051)	
874 (6.06061)	306 (5.05051)	578 (5.05051)	772 (5.05051)	

➔ **Association Rules** (Sup%, Conf%)

◆ **[nitems_0.1, ntrans_0.1],** min_support = 7%, min_confidence = 60%

63 <- 38 43 (10.101, 80)	48 <- 38 43 (10.101, 70)	48 <- 63 43 (11.1111, 63.6364)
38 <- 63 43 (11.1111, 72.7273)	38 <- 48 43 (9.09091, 77.7778)	63 <- 48 43 (9.09091, 77.7778)
63 <- 38 35 (10.101, 70)	38 <- 48 (24.2424, 66.6667)	63 <- 62 (14.1414, 64.2857)
38 <- 63 35 (7.07071, 100)	38 <- 14 (14.1414, 71.4286)	87 <- 33 (11.1111, 63.6364)
36 <- 38 43 (10.101, 70)	38 <- 25 (13.1313, 61.5385)	
38 <- 36 43 (9.09091, 77.7778)	43 <- 63 48 (8.08081, 87.5)	

◆ **[nitems_0.1, ntrans_1],** min_support = 3%, min_confidence = 50%

63 <- 38 35(3.15041,53.4483)	38 <- 87 11(3.04878,54.5455)	87 <- 38 62(3.76016,54.4118)
38 <- 87 28(4.36992,56.5789)	87 <- 38 43(3.25203,53.3333)	38 <- 87 62(3.76016,54.4118)

87 <- 38 29(3.15041,54.386)
38 <- 87 29(3.15041,56.3636)
87 <- 38 71(3.04878,52.6316)

38 <- 87 71(3.04878,53.5714)
87 <- 63 72(3.04878,54.5455)
63 <- 87 72(3.04878,52.6316)

87 <- 49(3.65854,50)
87 <- 55(3.15041,56.3636)

- **[nitems_1, ntrans_0.1],** min_support = 3%, min_confidence = **50%**

416 <- 38 (5.05051, 60)

778 <- 214 (6.06061, 50)

- **討論:**

  (i) Christian Borgelt之Apriori Tools執行速度比自己撰寫的code還快，可能有做過優化。

  (ii)Association Rules

  - 一開始使用的參數為Strong Rules(min_sup=3%, min_conf=60%)。
  - 上述所列出之結果皆為調整後的參數產生之關聯法則，實驗過程受限篇幅，因此不一一列出。
  - 第一筆(nitems_0.1, ntrans_0.1)因為重複性比較高，因此使用預設參數會有815筆，因此將min_sup提高至7%，結果精簡為16筆。從結果可以看出，其中有11筆都含有[38]元素，6筆含有[63]，可以猜測這兩個item是最常出現的item之二，參照上面frequent patterns的結果，果然得到38為出現頻率最高、63為頻率第二高的item。
  - 第二筆(nitems_0.1, ntrans_1)使用預設參數結果為0筆，因此將min_conf降至50%，推測原因為item數較少，從黃底標示部分可以看出：以[87<-38 29]、[38<-87 29]為例，即使三個element相同、支持度相同，但信心度仍不同，在[87、29]出現的transaction中，亦出現[38]的次數，比含有[38、29]的transaction，亦出現[87]的次數還多。
  - 第三筆(nitems_1, ntrans_0.1)使用預設參數結果為 1 筆，調整 min_conf 為 50%後亦只有 2 筆。可能原因為 item 總數較多(1000)，Transaction 數量又較少(100)，可能造成 item 並未全部出現於 transaction 中的狀況，即使出現可能重複性也不高，因此 support 與 confidence 總體來說會較低。

- **綜合討論:**
  - Apriori:
    - 實作簡單
    - 在找完frequent patterns前，需要對database進行多次的scan（計算一次candidate itemset就需要重新scan一次database）、產生大量的 candidate itemsets，因此時間、空間複雜度較高。
  - Hash-Tree:
    - 結構簡單，不需要預留空間，為動態結構。
    - 隨著資料量增加而增大。
    - 刪除node時不需改變結構。
    - 並沒有「排序」的特性，因此在需要排序時較沒有效率。
    - 如果資料量太多，tree會很深，而造成跟線性比對的time cost差不多，不見得比較節省時間。

- FP-Growth:
  - 利用divide-and-conquer的方式，focus在很小的database的scan。
  - 以suffix的方式recursively搜尋較短的patterns，搜尋完後再concatenate suffix，使suffix逐次增加。
  - 整個過程僅需對database進行一次scan，且不像Apriori有candidate itemsets的產生，節省許多時間、空間。
- 此次datasets的 Time cost:FP-Growth最少、Apriori居中、Hash-Tree最多。