**資料探勘DATA MINING** Project 1

Q36071059 蔡榮漾

1. **IBM Quest Data Generator.exe：**

此程式主要之參數有三個：ntrans、tlen與nitems，分別意義如下

ntrans transactions數量

tlen 每筆transactions平均items數

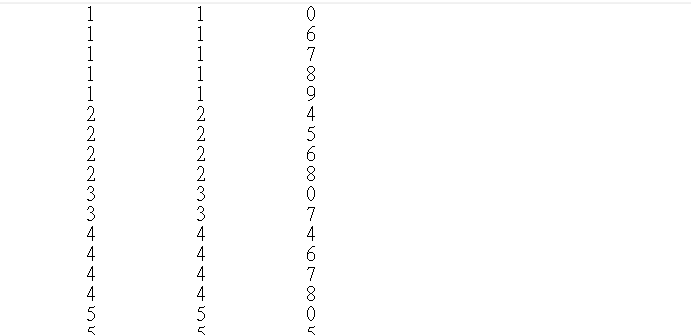
nitems items種類數

另IBM Quest Data Generator.exe程式中輸入的ntrans與nitems參數會再乘上1000才是實際數字，使用時須注意，此外參數可為小數(ex.0.1)；也可利用fname參數自訂檔案名稱，否則使用默認檔案名稱。

本報告使用參數為：

lit -ntrans 0.1 -tlen 5 -nitems 0.01

產生之資料格式如下(只取前面小部分)：



根據個人理解，其意義應為：

TID ITEMS

1. 0,6,7,8,9
2. 4,5,6,8
3. 0,7
4. 4,5,7,8

……

由於IBM Quest Data Generator.exe 產生之資料格式難以直接利用，

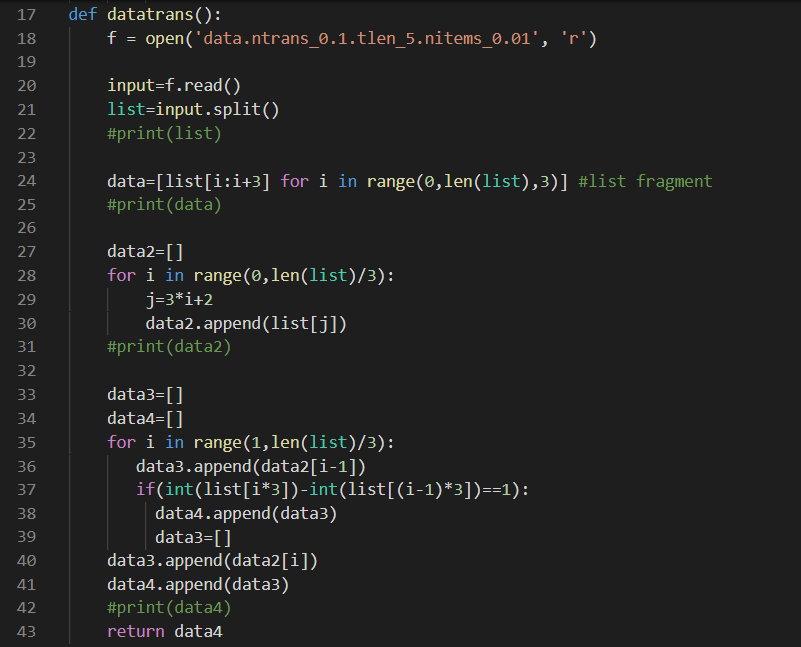
先將其整理，使用以下程式碼(Python2.7)：

Figure 1

先讀取資料檔案，將內容放至一個串列list，再將list轉為巢狀串列把每筆資料分開放至data，去掉TID放至data2，最後以巢狀串列區隔每筆不一樣的transactions。

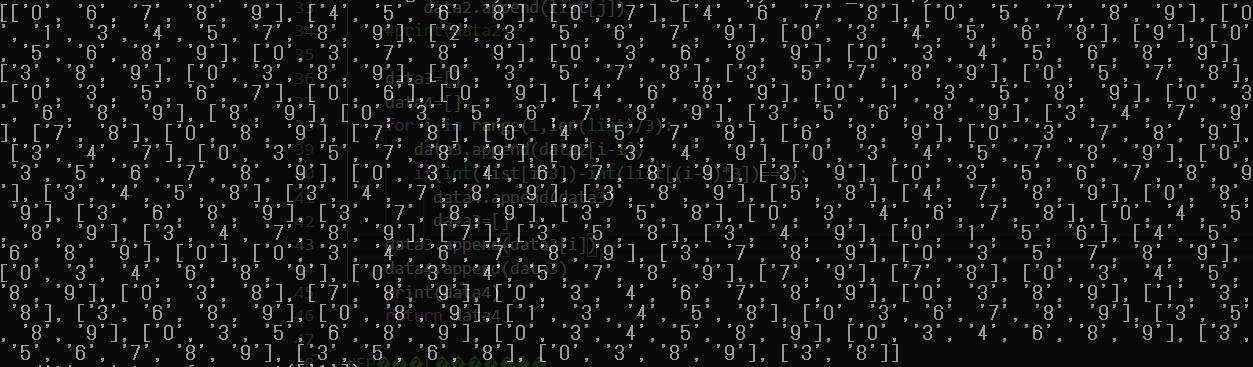
得到之data4如下：

Figure 2

至此資料之處理暫告一段落，後續將提到此datatrans()函式如何使用。

1. **WEKA：**

由於WEKA似乎只吃 .arff檔案，

且有特定格式，經查詢後，撰寫程式trans\_arff.py，

將先前IBM Quest Data Generator.exe所產生之原始資料data.ntrans\_0.1.tlen\_5.nitems\_0.01轉為稀疏型資料weka\_data.arff。

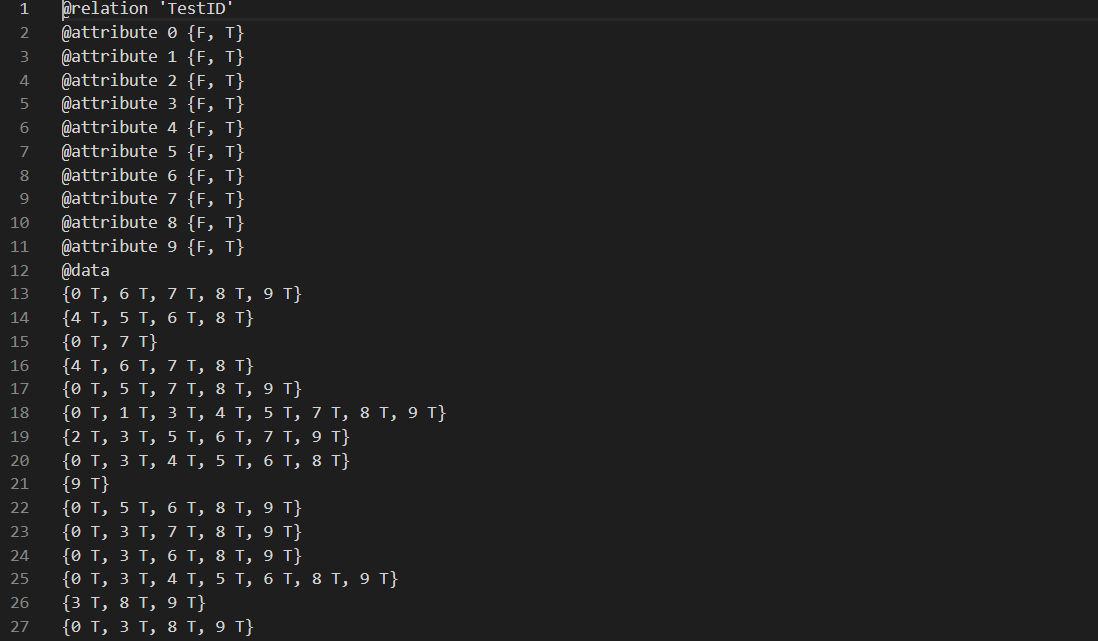
weka\_data.arff之格式如下：

Figure 3

簡單舉例，此data(data.ntrans\_0.1.tlen\_5.nitems\_0.01)共有0~9 10種items，

第一行必須先為此筆資料(@relation)命名，

再來則要列出所有items(@attribute 0 {F, T}~@attribute 9 {F, T})，

然後以一行@data隔開，此後一行放置一個transaction。

先使用WEKA對此資料進行測試，進行FP\_Growth，使用參數為：



得到之結果：

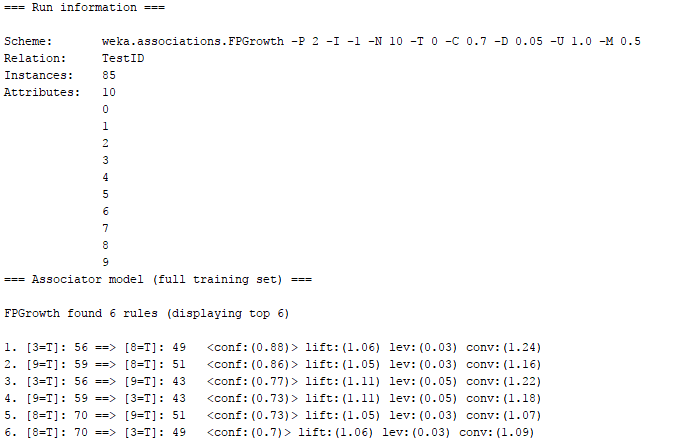
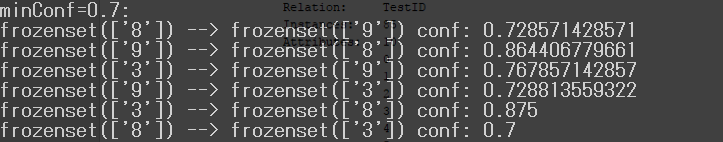


Figure 4

以Apriori演算法進行驗證(minConf:0.7 minSup 0.5)，得知WEKA操作上沒有問題，產生之關聯法則與各信賴度大致相同。

1. **FP-Growth之實作(fpg.py)：**

建立FP-tree：

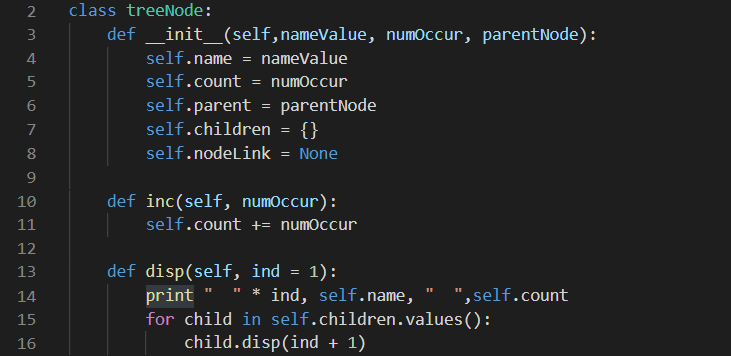
在開始建樹之前，需先決定節點構造：

Figure 5

name 元素名稱

count 元素出現次數

parent 指向子點

children 指向父點

nodeLink 指向下一相同名稱節點

inc() 計算元素出現次數

disp() 輸出節點&子點的FP結構

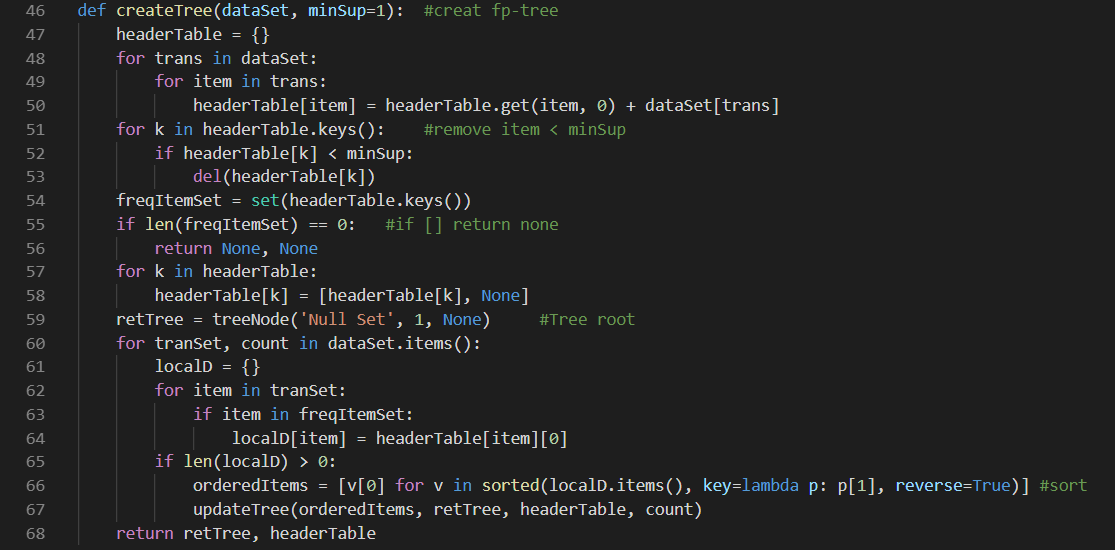
建立FP-tree：

Figure 6

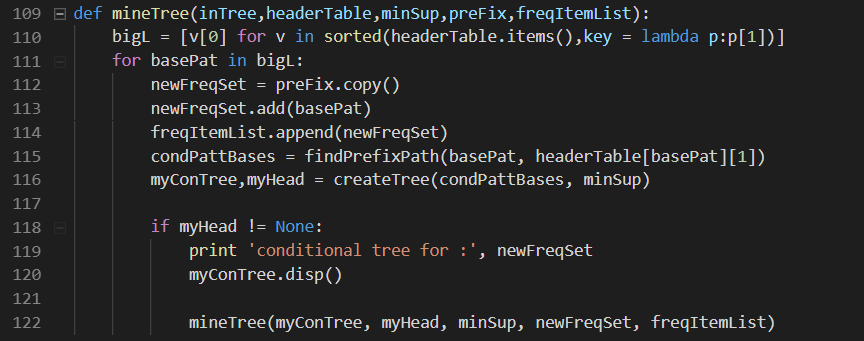
建立Conditional FP-tree：

Figure 7

此處流程大致上是，藉由前處creatTree() 函式所建立之FP-tree得到CPB(Conditional Pattern Base)，並利用CPB建立Conditional FP-tree，然後持續遞迴，直到樹只包含一個item。

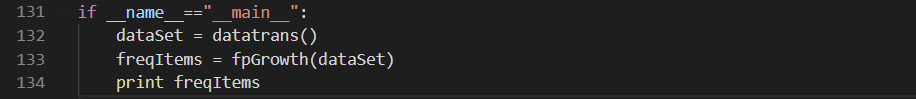
主程式：

Figure 8

此處利用datatrans()將IBM Quest Data Generator.exe產生之數據轉換格式並放到變數dataSet供其他函式使用，之後運行fpGrowth()函數。

樹的顯示方式：

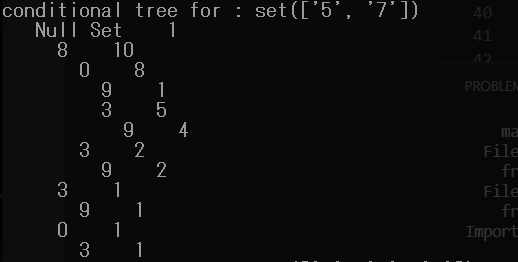


Figure 9

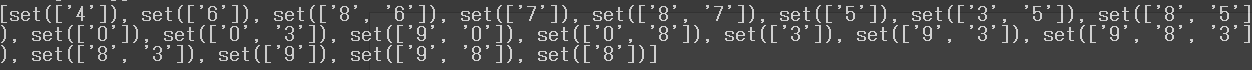
以此為例，可表示為以下樹：

得到之頻繁項集：

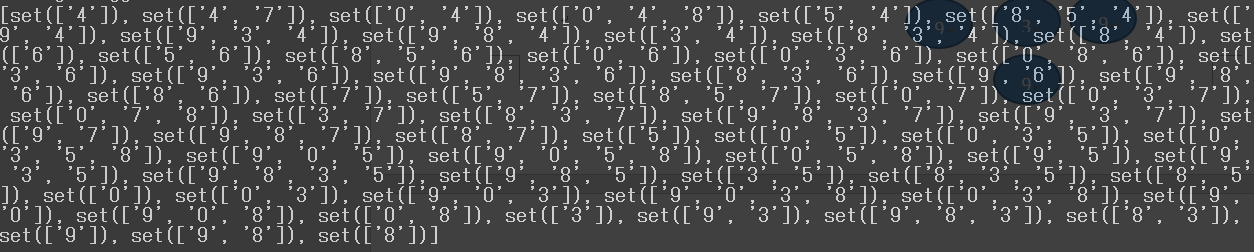
minSup為30時，Frequent Patterns為：



minSup為20時，Frequent Patterns為：



minSup為10時，Frequent Patterns為：



觀察得知，FP-tree建構時所設立的min support值越高，確實如預想的會讓頻繁項集數量減少，亦可作為驗證程式正確性的步驟之一。

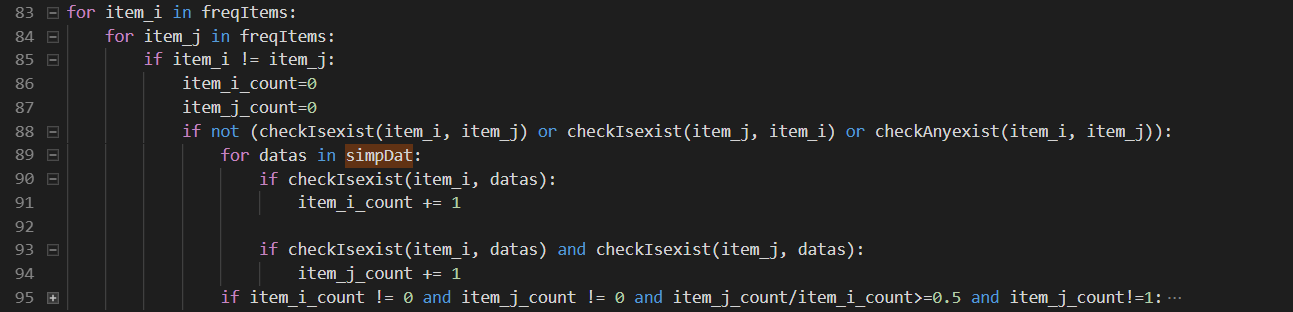
利用頻繁項集找尋關聯法則與計算confident：

Figure 10

得到之結果：

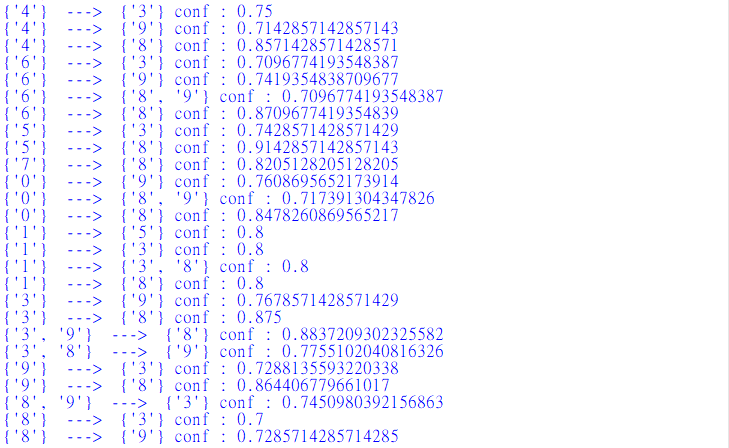


Figure 11

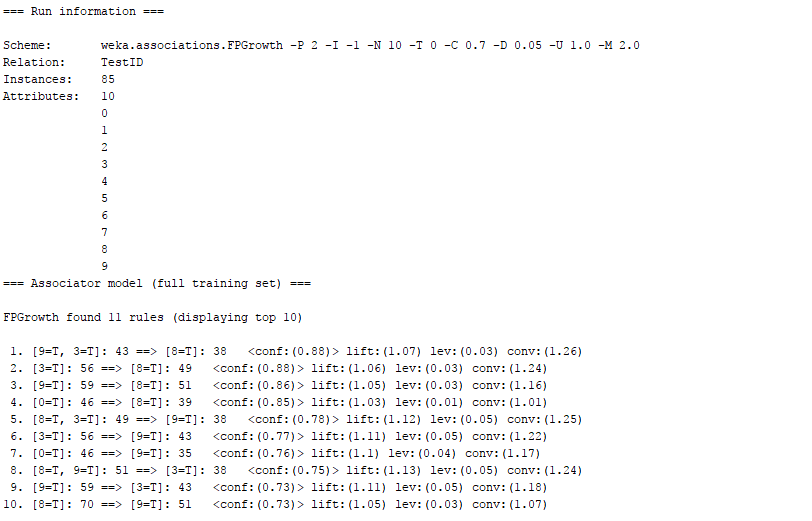
與WEKA所得之結果比較：

Figure 12

發現兩者產生之關聯法則與其conf大致相同。

列出前三者為例進行比較：

[9,3]-->[8] conf:0.88

[3]-->[8] conf:0.88

[9]-->[8] conf:0.86

確實到小數點第三位為止並無太大差異，

可知撰寫之程式所產生的結果應該正確；

唯產生之關聯法則數目有差異，

此部分是否為參數設定上之差異，還可進一步測試。

1. **Kaggle**

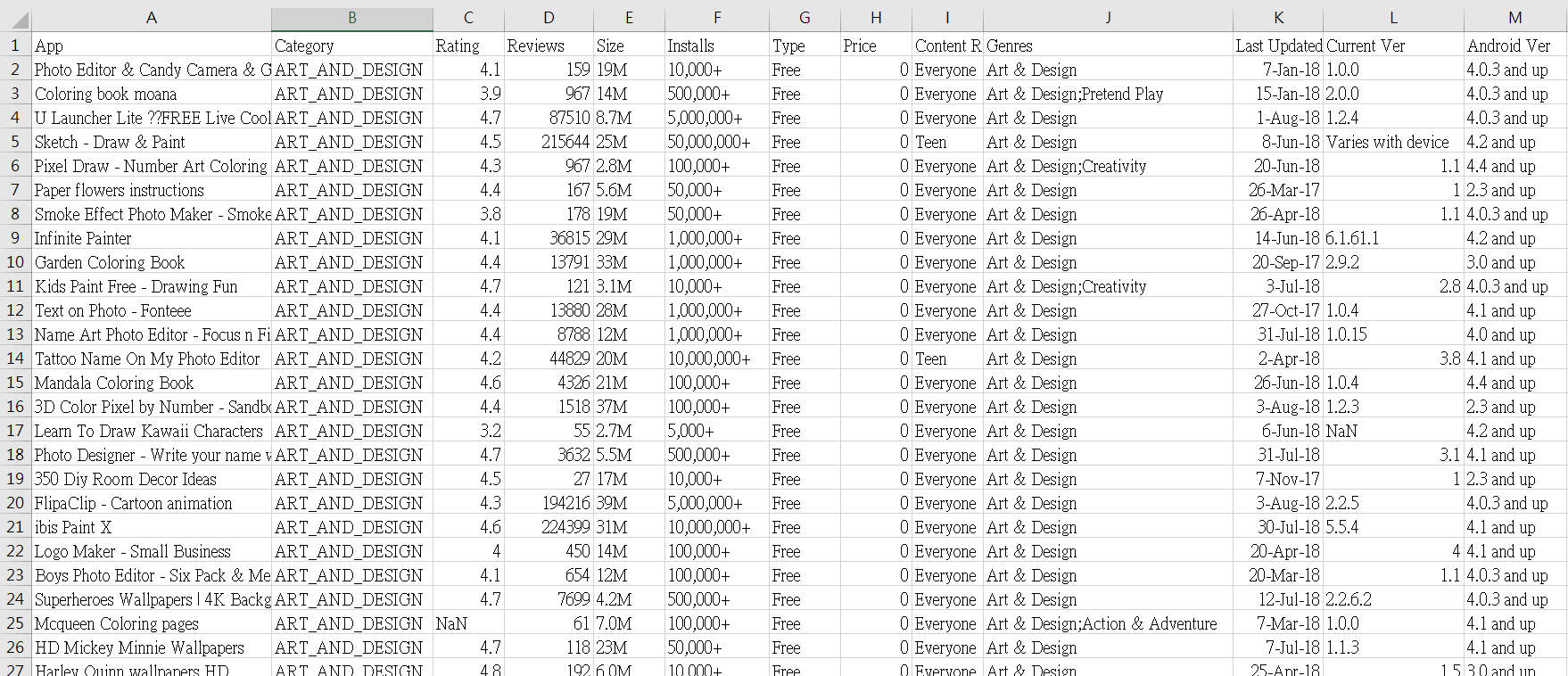
Dataset取自：

<https://www.kaggle.com/lava18/google-play-store-apps>

檔案名稱：Kaggle\_googleplaystore.csv

唯一個紀錄Google play store內各APP資料的Dataset。

初觀資料：



首先根據這份Data，如果我想知道APP之熱門程度(Install)與Category、Rating、Size、等等屬性的關係，應可去掉其他我較不關心之屬性(ex. Last Updated)，得到一份較小的Dataset，使程式處理之速度稍快一些。

此外，發現有部分APP之資料內容有些許瑕疵，比方說Rating最高為五，卻有部分APP之Rating高達5、6位數或無紀錄，或是否免費屬性除了是與否之外還有標示售價的狀況，以上這些資料予以剔除不計或改正。

某些屬性可用一threshold當作門檻，比方說Install，可將100萬下載數以上之APP視為熱門軟體，得到一個新的屬性「熱門與否」取代下載數，進行處理。