資料探勘導論

Final Project

Clustering

資訊三甲 10427101 馬若芸

一、 資料集介紹

使用的資料檔名:Repos500。

	A	B	C	D	E	F	G	H	I	J
1		login	name	fork	stargazers_count	watchers_count	forks_count	language	forks	watchers
2	1	sindresorhus	acosh	FALSE	5	5	0	JavaScript	0	5
3	2	sindresorhus	active-win	FALSE	54	54	9	JavaScript	9	54
4	3	sindresorhus	active-win-cli	FALSE	17	17	1	JavaScript	1	17
5	4	sindresorhus	add-asset-webpack-plugin	FALSE	36	36	0	JavaScript	0	36
6	5	sindresorhus	add-module-exports-webpack-plugin	FALSE	19	19	0	JavaScript	0	19
7	6	sindresorhus	aggregate-error	FALSE	81	81	4	JavaScript	4	81
8	7	sindresorhus	alfred-dark-mode	FALSE	63	63	1	NA	1	63
9	8	sindresorhus	alfred-emoj	FALSE	134	134	5	JavaScript	5	134
10	9	sindresorhus	alfred-lock	FALSE	39	39	1	NA	1	39
11	10	sindresorhus	alfred-npms	FALSE	195	195	9	JavaScript	9	195

欄位說明:

Login: 使用者名稱。

Name: 建立的倉庫名稱。

Fork: 該倉庫是否 fork 別人的。

Stargazers_count = watchers_count = watchers: 關注該倉庫的人數。

Forks_count = forks: 該倉庫被 fork 的人數。

全部的資料共 124308 筆。

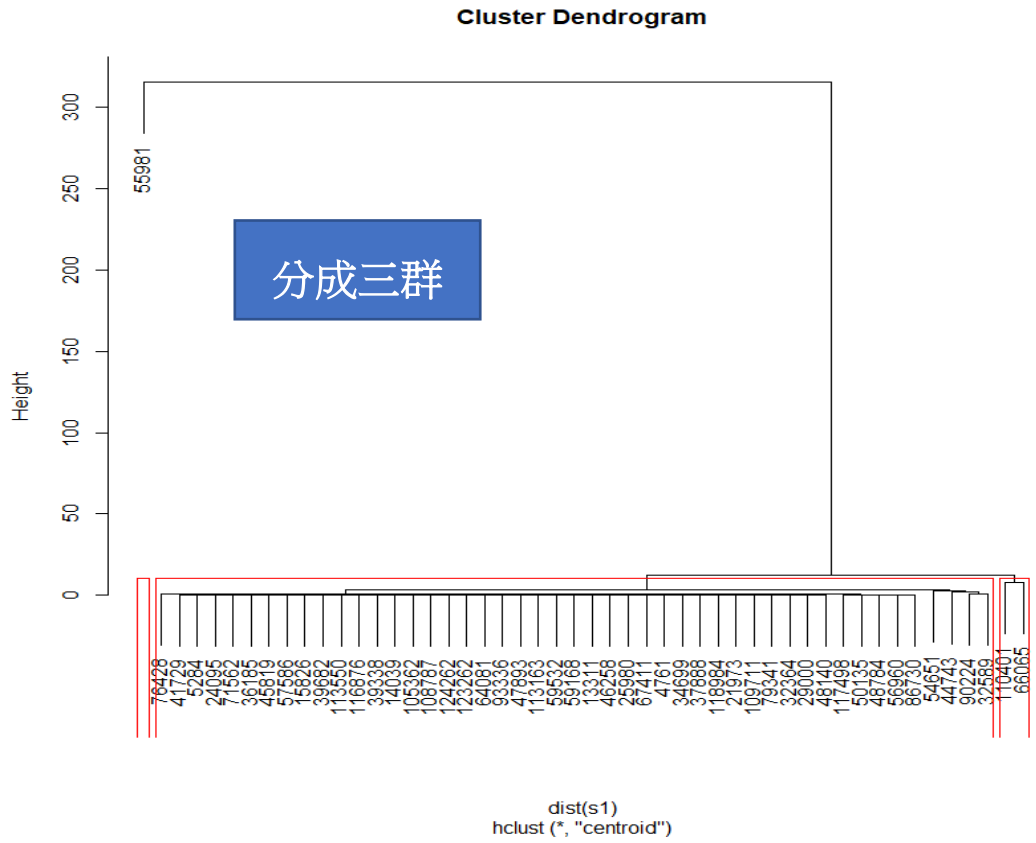
二、 資料分群方法

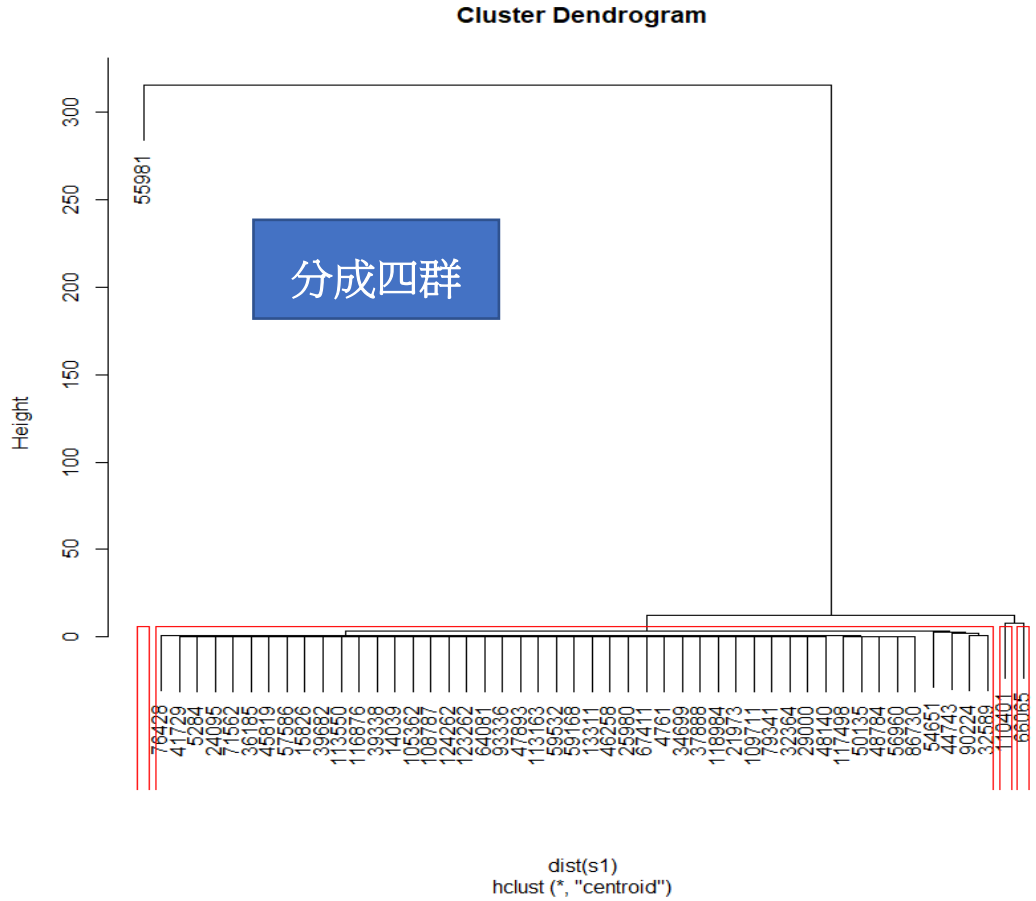
因為資料的筆數過多，R 的資料分群沒辦法全部處理完，所以是取樣不重複的方式來接著進行分群，取樣的數量分別有:50、1000、5000、10000 筆，進行這些取樣先用來觀察階層式分群資料的分布來決定要分幾個群，因為取樣的數量越多，所呈現的是圖示會糊成一團，才用由小到大數量取樣來確認小數量的取樣和大數量取樣的資料的分布會不會相差太多。

以 forks、watchers 兩個欄位作分群，以 centroid 方法進行階層式分群:

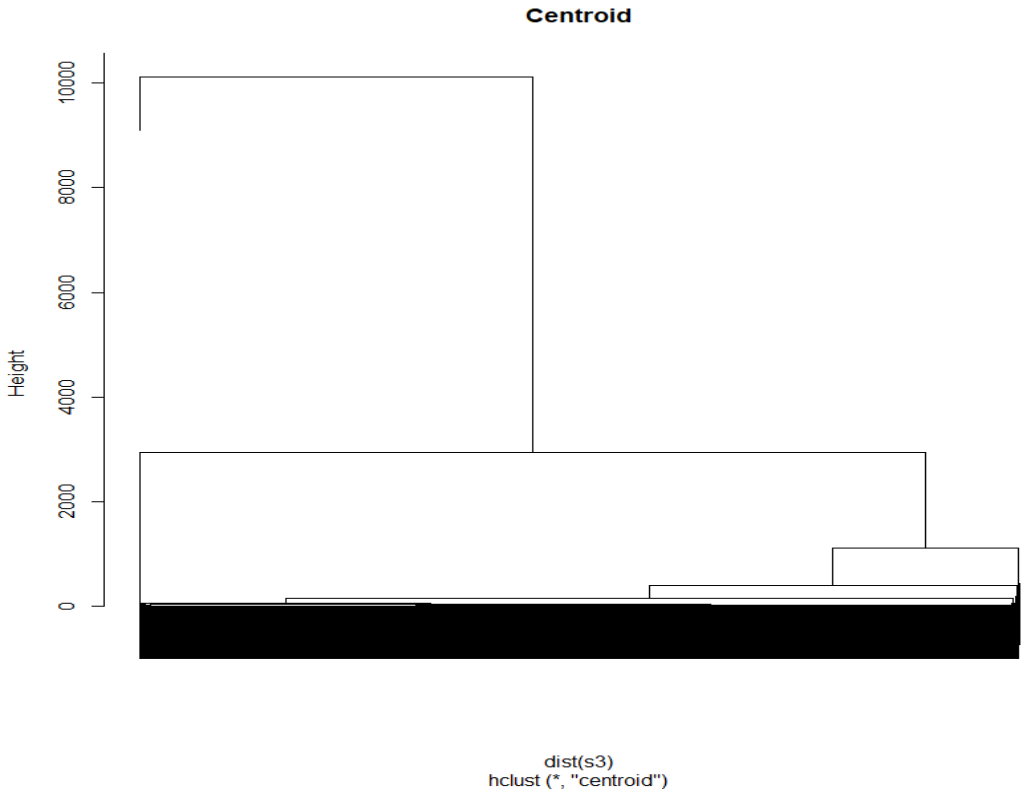
```
dataCluster1 <- hclust(dist(s1), method = "centroid")       #--進行階層式分群
plot(dataCluster1, labels = FALSE, main = "Centroid")       #--以圖像顯示
```

Sample size = 50，觀察出分群可以分成 3、4 群(因為隨機取樣，所以有時會明顯地分出 3 或 4 群):


Cluster Dendrogram
分成三群
Height
0
50
100
150
200
250
300
55981
dist(s1)
hclust (*, "centroid")


Cluster Dendrogram
分成四群
Height
0
50
100
150
200
250
300
55981
dist(s1)
hclust (*, "centroid")

1000 筆以上基本都會糊成一團，比較不方便觀察:



接著要使用 kmeans 方式以得到的數值來決定要分成 3 群還是 4 群，主要以 sample size = 5000 來進行分群觀察結果。

以 sample size = 5000 為例，用 k-means 方式隨機 25 次找初始化中心點進行分群。

```
> set.seed(280411)
> resultCluster3v1 <- kmeans(x = s3, centers = 3, nstart = 25)  # 分成三群
> resultCluster3v1
K-means clustering with 3 clusters of sizes 5, 4966, 1

Cluster means:
        forks     watchers
1  94.4000000  1500.400000
2   0.6512284     3.193919
3 765.0000000 16624.000000

Within cluster sum of squares by cluster:
[1] 1162060 2682012       0
 (between_SS / total_SS =  98.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

分成三群

```
> set.seed(280411)
> resultCluster3v2 <- kmeans(x = s3, centers = 4, nstart = 25)  # 分成四群
> resultCluster3v2
K-means clustering with 4 clusters of sizes 1, 20, 4, 4947

Cluster means:
        forks     watchers
1 765.0000000 16624.000000
2  40.3000000   340.950000
3 104.2500000  1653.250000
4   0.5019204     2.007479

Within cluster sum of squares by cluster:
[1]      0.0 604299.1 692857.5 545221.5
 (between_SS / total_SS =  99.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
>
```

分成四群

從結果來看:三群(98.7%) < 四群(99.4%)，分成四群的話，群和群之間的距離比較遠，能夠分群分得明顯。雖然兩組數值得很高，但接著就以分四群來進行分析。

```
> resultCluster3v2$totss           #--各個點的離均差平方和
[1] 291814383
> resultCluster3v2$withinss        #--每個單一群內部的離均差平方和 (群內點跟點之間)
[1]      0.0 604299.1 692857.5 545221.5
> resultCluster3v2$tot.withinss    #--每個單一群內部的離均差平方和的總和
[1] 1842378
> resultCluster3v2$betweenss       #--各個群之間的離均差平方和 (群跟群之間)
[1] 289972004
> resultCluster3v2$size            #--每個群的點個數
[1]    1   20    4 4947
```

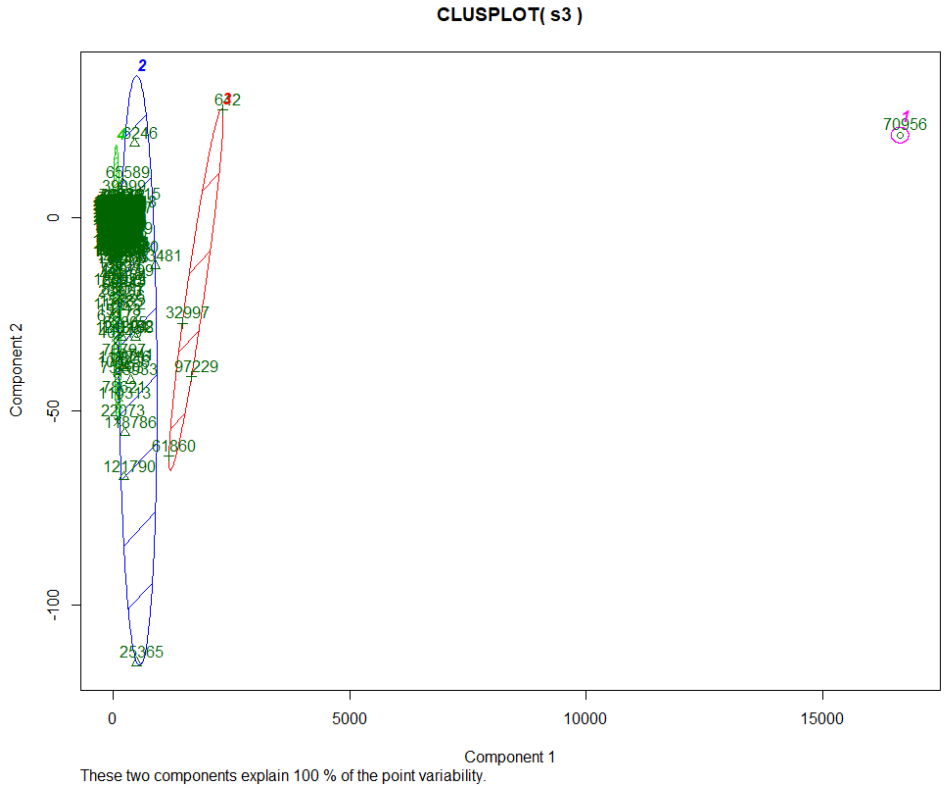
其中有一群的點個數只有一個，推測此點可能是離異點。

```
Cluster means:
        forks     watchers
1 765.0000000 16624.000000
2  40.3000000   340.950000
3 104.2500000  1653.250000
4   0.5019204     2.007479
```

watchers 和 forks 之間成正比，可見關注度越高，被 fork 的次數也越高。第一群的點個數只有一個，之前推測此點可能屬於離異點，接下來將細部分析每一群的點是屬於哪一種的程式語言，並找出潛在的用途。

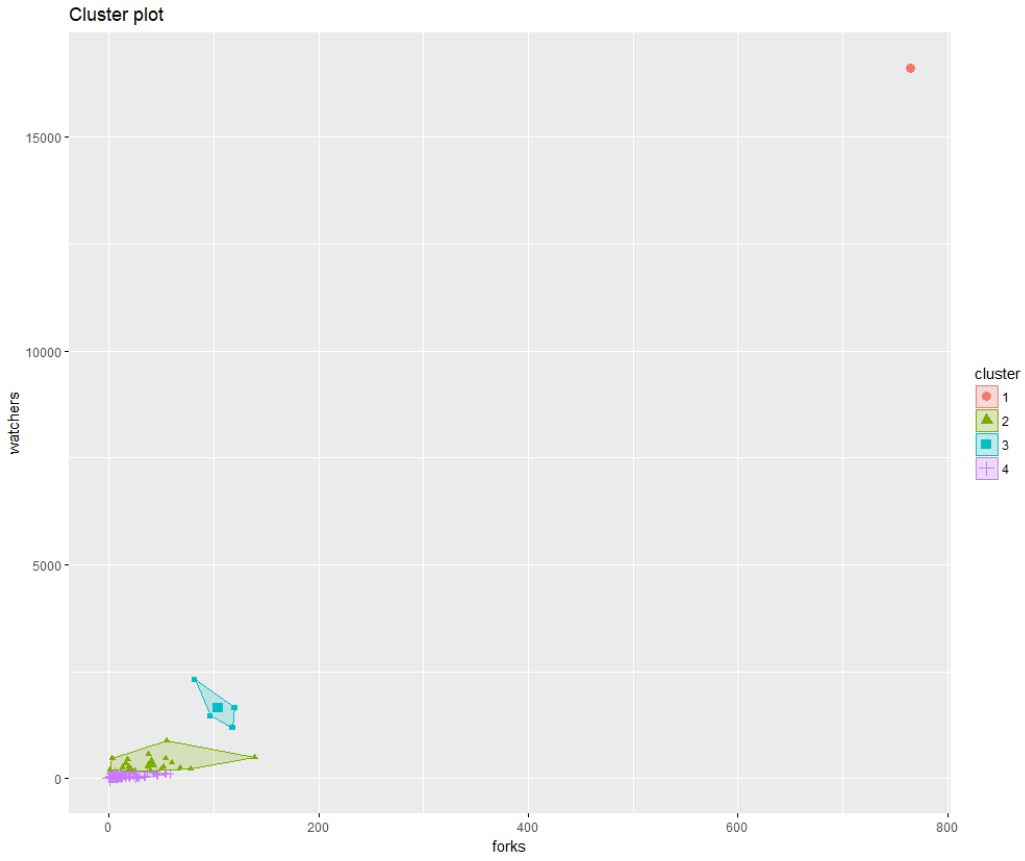
三、 分群結果

```
#library(cluster)
clusplot(s3, resultCluster3v2$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



第一群:粉色；第二群:藍色；第三群:紅色；第四群:綠色。

```
#library(factoextra)
fviz_cluster(resultCluster3v2, data = s3, geom = "point", stand = FALSE)
```



從上面兩張圖來看因為第一群的緣故，其他三群不能從圖示上清楚觀察，所以試著將第一群的值移除，因為第一群的值移除後只剩下三群，故以 kmeans 方式分成三群。
(因為是隨機抽樣來分析的，每次不一定會出現離異點，所以以下的程式碼沒有寫在檔案中。)

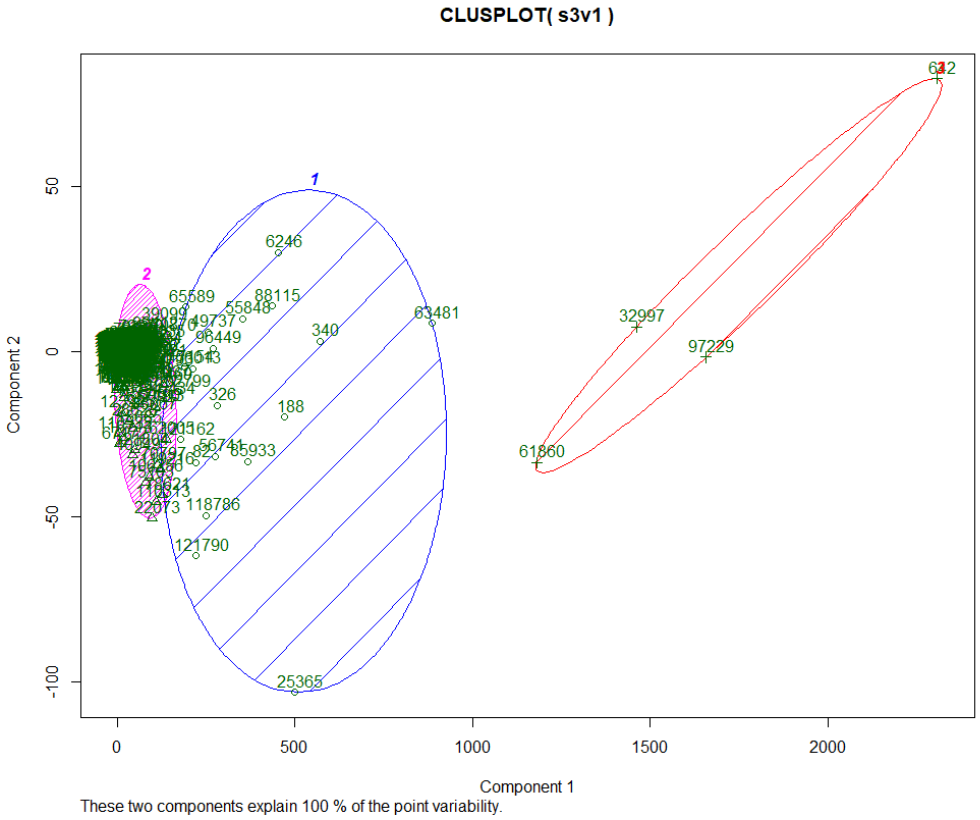
```
> which(s3$watchers >= 16624)
[1] 2052
> s3[2052,]
       forks watchers
70956    765    16624
```

移除後分成三群(resultCluster3v2_r1)。

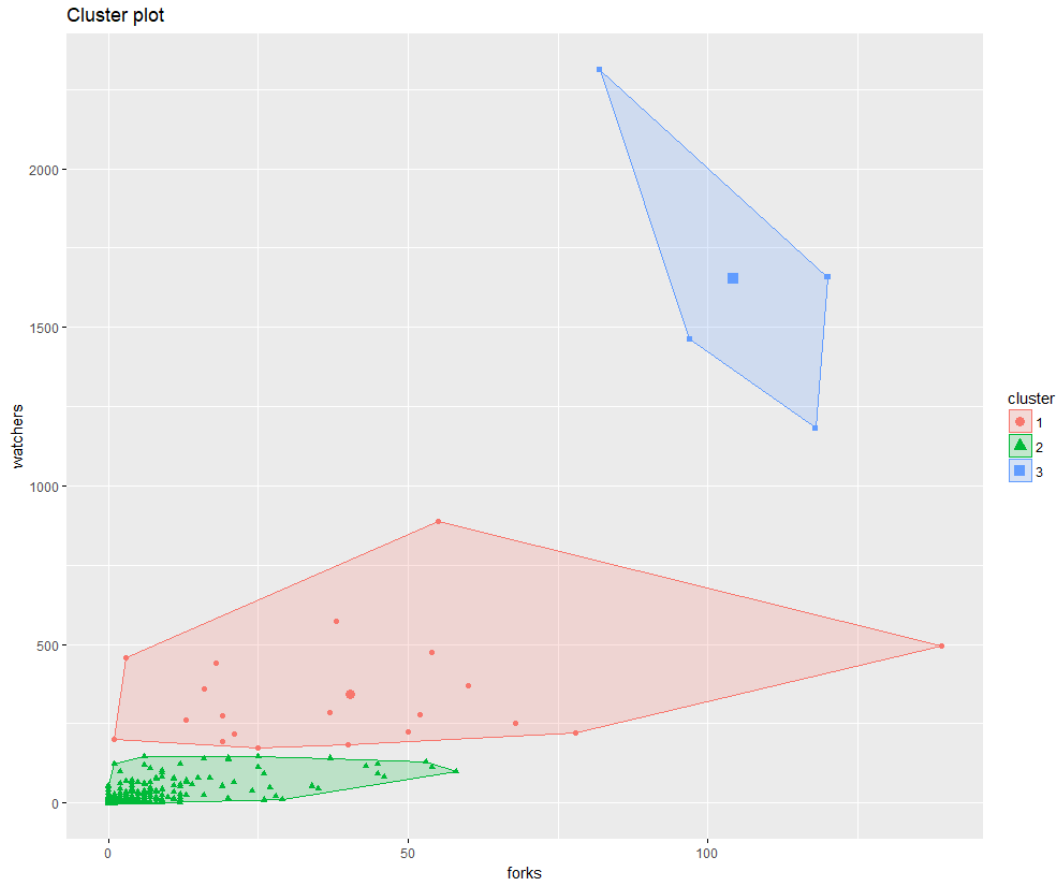
```
> s3v1 <- s3[-2052,]
> set.seed(280411)
> resultCluster3v2_r1 <- kmeans(x = s3v1, centers = 3, nstart = 25)
> resultCluster3v2_r1
K-means clustering with 3 clusters of sizes 20, 4947, 4

Cluster means:
         forks    watchers
1  40.3000000  340.950000
2   0.5019204    2.007479
3 104.2500000 1653.250000

Within cluster sum of squares by cluster:
[1] 604299.1 545221.5 692857.5
 (between_SS / total_SS =  87.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

觀察到移除離異點後，群之間的距離平方占總體距離平方差的比率的值下降不少，可見此離異點的影響之大。

```
> clusplot(s3v1, resultCluster3v2_r1$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```




```
> fviz_cluster(resultCluster3v2_r1, data = s3v1, geom = "point", stand = FALSE)
```



感覺上第一、二群距離相近，分群的狀況比較不明顯。
回到原本還有離異點的分四群結果，現在來觀察分群的結果與程式語言配對之間的關係。

```
Cluster means:
        forks      watchers
1 765.0000000 16624.000000
2  40.3000000   340.950000
3 104.2500000  1653.250000
4   0.5019204     2.007479
```

(四群的內容)
上排欄位是資料中出現的各種程式語言，左側欄位是群編號，內部的值是在該群出現的次數。

```
> table(resultCluster3v2$cluster, orgS3$language)

  1C Enterprise ActionScript  Ada Agda AGS Script Alloy AMPL ANTLR ApacheConf Apex
1             0            0    0    0          0     0    0     0          0    0
2             0            0    0    0          0     0    0     0          0    0
3             0            0    0    0          0     0    0     0          0    0
4             0            2    0    1          0     1    0     1          4    0

  API Blueprint AppleScript  Arc Arduino  ASP Assembly Augeas AutoHotkey AutoIt  Awk
1             0           0    0       0    0        0      0          0      0    0
2             0           0    0       0    0        0      0          0      0    0
3             0           0    0       0    0        0      0          0      0    0
4             0           0    0       4    0        3      0          0      0    1

  Batchfile Bison Brainfuck  Bro    C   C#  C++ CartoCSS Cirru Clojure CMake COBOL
1         0     0         0    0    0    0    0        0     0       0     0     0
2         0     0         0    0    0    0    0        0     0       0     0     0
3         0     0         0    0    0    0    0        0     0       0     0     0
4         0     0         0    1  139   79  102        2     0      13     1     0

  CoffeeScript ColdFusion Common Lisp  Coq Crystal  CSS Cucumber Cuda    D Dart Delphi
1            0          0           0    0       0    0        0    0    0    0      0
2            0          0           0    0       0    1        0    0    0    0      0
3            0          0           0    0       0    0        0    0    0    0      0
4           51          0           0    1       1  125        1    1    2   11      0

  Diff DIGITAL Command Language  DOT Dylan Eagle Elixir  Elm Emacs Lisp Erlang   F#
1    0                        0    0     0     0      0    0          0      0    0
2    0                        0    0     0     0      0    0          0      0    0
3    0                        0    0     0     0      0    0          0      0    0
4    0                        0    0     0     0     12   11         14     13    4

  Factor Forth Fortran FORTRAN FreeMarker Game Maker Language GAMS  GAP
1      0     0       0       0          0                    0    0    0
2      0     0       0       0          0                    0    0    0
3      0     0       0       0          0                    0    0    0
4      0     1       0       2          0                    0    0    0

  GCC Machine Description GDScript Gherkin GLSL Gnuplot   Go Grammatical Framework Groff
1                       0        0       0    0       0    0                       0     0
2                       0        0       0    0       0    1                       0     0
3                       0        0       0    0       0    0                       0     0
4                       1        1       0    1       0  142                       0     1

  Groovy Hack Handlebars Haskell Haxe HaXe  HCL HTML   Hy  IDL Idris Inno Setup   Io Java
1      0    0          0       0    0    0    0    0    0    0     0          0    0    0
2      0    0          0       0    0    0    0    3    0    0     0          0    0    0
3      0    0          0       0    0    0    0    0    0    0     0          0    0    0
4      5    0          0      48    1    0    2  156    0    0     1          0    0  182

  JavaScript Julia Jupyter Notebook KiCad Kotlin LabVIEW Lean  Lex Limbo LiveScript LLVM
1          1     0                0     0      0       0    0    0     0          0    0
2         11     0                0     0      0       0    0    0     0          0    0
3          2     0                0     0      0       0    0    0     0          0    0
4       1499     6               32     0      6       0    0    1     0          0    0

  Logos  LSL  Lua   M4 Makefile Mako Mask Mathematica Matlab  Max MAXScript Mercury Mirah
1     0    0    0    0        0    0    0           0      0    0         0       0     0
2     0    0    0    0        0    0    0           0      0    0         0       0     0
3     0    0    0    0        0    0    0           0      0    0         0       0     0
4     4    0   13    0       21    0    0           0      4    0         0       0     0

  MoonScript mupad Nemerle NetLogo NewLisp Nginx  Nim Nimrod  Nix NSIS Objective-C
1          0     0       0       0       0     0    0      0    0    0          0
2          0     0       0       0       0     0    0      0    0    0          0
3          0     0       0       0       0     0    0      0    0    0          0
4          0     0       0       0       0     0    0      0    0    1         48

  Objective-C++ Objective-J OCaml  ooc  Opa OpenEdge ABL OpenSCAD Pascal Perl Perl 6
1             0           0     0    0    0           0        0      0    0      0
2             0           0     0    0    0           0        0      0    0      0
3             0           0     0    0    0           0        0      0    0      0
4             1           0    19    0    2           1        0      4   83      1
```

	Perl6	PHP	PicoLisp	PigLatin	PLpgSQL	PLSQL	PogoScript	PostScript	POV-Ray	SDL	PowerShell
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	175	0	0	0	0	0	1	0	0	10

	Processing	Prolog	Protocol	Buffer	Puppet	Pure Data	PureBasic	PureScript	Python	QML
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0
4	1	2	0	9	0	0	6	492	0	

	R	Racket	Ragel in Ruby Host	RAML	Rascal	Red	RobotFramework	Roff	Rouge	Ruby	Rust
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	1	0
4	33	10	0	0	0	0	0	0	1	362	34

	SaltStack	SAS	Scala	Scheme	ShaderLab	Shell	Slash	Smali	Smalltalk	Smarty	SourcePawn
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	0
4	1	0	51	1	0	128	0	0	0	4	1

	SQL	SQLPL	Stan	Standard ML	Stata	SuperCollider	Swift	SystemVerilog	Tcl	TeX	Thrift
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	33	1	0	14	0

	TLA	Turing	TypeScript	Uno	UnrealScript	Vala	Verilog	VHDL	Vim script	VimL	Visual Basic
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	34	0	0	0	0	0	7	30	1

	Vue	Web Ontology Language	WebAssembly	wisp	XML	XSLT	Xtend	Yacc	Zephir
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	4	1	0	1	0	5	0	0	0

四、 討論

因為以隨機抽樣進行分群，雖然結果都會有點不太一樣，但可以看出大概的趨勢。從 table 中:

	GCC	Machine Description	GDScript	Gherkin	GLSL	Gnuplot	Go	Grammatical Framework	Groff
1		0	0	0	0	0	0	0	0
2		0	0	0	0	0	1	0	0
3		0	0	0	0	0	0	0	0
4		1	1	0	1	0	142	0	1

	Groovy	Hack	Handlebars	Haskell	Haxe	HaXe	HCL	HTML	Hy	IDL	Idris	Inno Setup	Io	Java
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	3	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	5	0	0	48	1	0	2	156	0	0	1	0	0	182

	JavaScript	Julia	Jupyter Notebook	KiCad	Kotlin	LabVIEW	Lean	Lex	Limbo	LiveScript	LLVM
1	1	0	0	0	0	0	0	0	0	0	0
2	11	0	0	0	0	0	0	0	0	0	0
3	2	0	0	0	0	0	0	0	0	0	0
4	1499	6	32	0	6	0	0	1	0	0	0

	Logos	LSL	Lua	M4	Makefile	Mako	Mask	Mathematica	Matlab	Max	MAXScript	Mercury	Mirah
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	0	13	0	21	0	0	0	4	0	0	0	0

	MoonScript	mupad	Nemerle	NetLogo	NewLisp	Nginx	Nim	Nimrod	Nix	NSIS	Objective-C
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1	0	48

	Objective-C++	Objective-J	OCaml	ooc	Opa	OpenEdge ABL	OpenSCAD	Pascal	Perl	Perl 6
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	1	0	19	0	2	1	0	4	83	1

其中 HTML、JavaScript 是有部分被分至第二、三群的，第二、三群內容是

有被 fork、watcher 值較多的。所以從 table 中我想到可能的用途如下:

假設有一個人，心裡也沒有特別指定要學哪一種程式語言，而程式語言百百種不知從何選起，所以想要藉著從 GitHub 觀摩別人的程式碼來學習一些新的程式語言。

從 table 上觀察出，用該語言出現的次數總和，可以了解那些語言較冷門、熱門，例如 JavaScript 總數:1513、MaxScript:0 (雖然總數會根據隨機抽樣而改變，不過每次抽樣得到的總數不會相差太多)，那麼此人可以此選擇學習 JavaScript，或許學會之後的用途比較廣泛。還有可以比較在 GitHub 上觀注、fork 的價值，以上圖紅框的 JavaScript、HTML 為例，由於 JavaScript 佔第一~三群的數量較多，表示有不少的 JavaScript 程式碼是被較多人關注、fork 的，這或許是值得拿來觀摩學習或利用的，那麼從 GitHub 上學習 JavaScript 或許比 HTML 來的有用。

現在把每一群出現哪種程式語言的比例歸納如下:

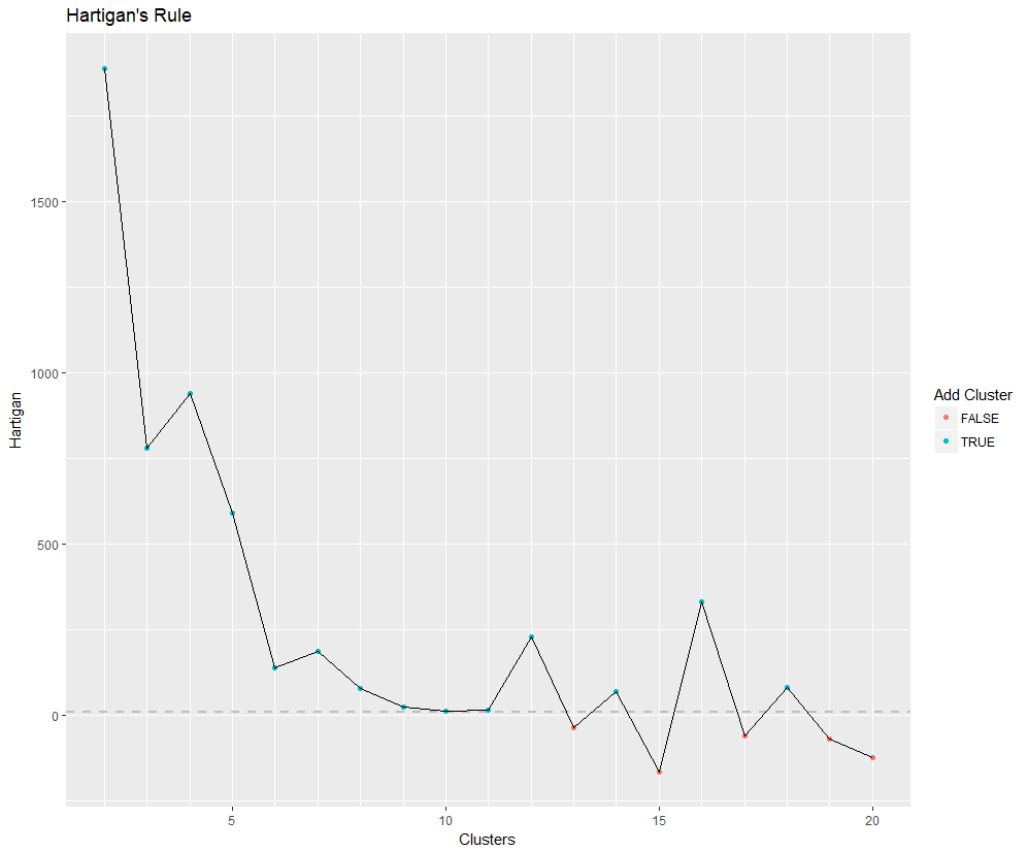
依序由 fork、watcher 值高到低:

1. 第一群: JavaScript(100%)。
2. 第三群: JavaScript(50%)、Ruby(25%)、Shell(25%)。
3. 第二群: JavaScript(58%)、HTML(16%)、GO = CSS = Ruby = Python = Shell(各 5.2%)。
4. 第四群: 因為 fork、watcher 值太低且程式語言數量太多就不考慮。

可見 fork、watcher 值較高的前三群裡 JavaScript 佔的比例最多，可以認為從 GitHub 觀摩學習 JavaScript 比起其他語言是比較有價值的。

在一開始決定分群數是用階層式分群以及自己的判斷來決定的，但實際上到底是要分成幾群才是較佳的結果呢?

```
s3Best1 <- FitKMeans(s3, max.clusters=20, nstart=50, seed=280411)
PlotHartigan(s3Best1)
```




```
> s3Best1
   Clusters    Hartigan AddCluster
1         2 1888.84759        TRUE
2         3  779.51355        TRUE
3         4  940.46869        TRUE
4         5  588.76107        TRUE
5         6  137.14690        TRUE
6         7  187.48481        TRUE
7         8   77.32520        TRUE
8         9   25.19559        TRUE
9        10   12.44441        TRUE
10       11   15.43438        TRUE
11       12  228.21085        TRUE
12       13  -37.44238       FALSE
13       14   68.01415        TRUE
14       15 -166.09201       FALSE
15       16  332.69880        TRUE
16       17  -59.67454       FALSE
17       18   81.21840        TRUE
18       19  -68.47913       FALSE
19       20 -122.91168       FALSE
```

接著試著分成 12 個群。

```
> set.seed(280411)
> resultCluster3EX <- kmeans(x = s3, centers = 12, nstart = 25)  # 分成12群
> resultCluster3EX
K-means clustering with 12 clusters of sizes 6, 168, 3, 5, 54, 22, 5, 3, 3, 31, 4671, 1

Cluster means:
          forks     watchers
1  165.5000000 1471.1666667
2    3.5000000   13.5357143
3  313.6666667 2605.6666667
4   32.6000000  276.0000000
5   10.9259259   34.9444444
6   20.7727273  174.5909091
7   59.6000000  709.2000000
8  236.3333333  214.3333333
9   43.3333333  428.3333333
10  10.5161290   89.2903226
11   0.1794048    0.4440163
12 638.0000000 8879.0000000

Within cluster sum of squares by cluster:
 [1] 212220.333   9981.786 507371.333   6077.200  15004.537  21411.182  26536.000  14013.333   9081.333  13720.129   8112.769      0.000
 (between_SS / total_SS =  99.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"         "ifault"
```

得到的群之間的距離平方占總體距離平方差的比率的值不錯，但也可能是因為只包含兩三個點的群導致的，那些點也可能屬於離異點。

發現第十二群的各數只有一個，可以判斷為離異點，暫時撇開不考慮，並且特別 fork、watcher 值比較高的第一和第三群觀察是由哪幾個語言占的比例較高。

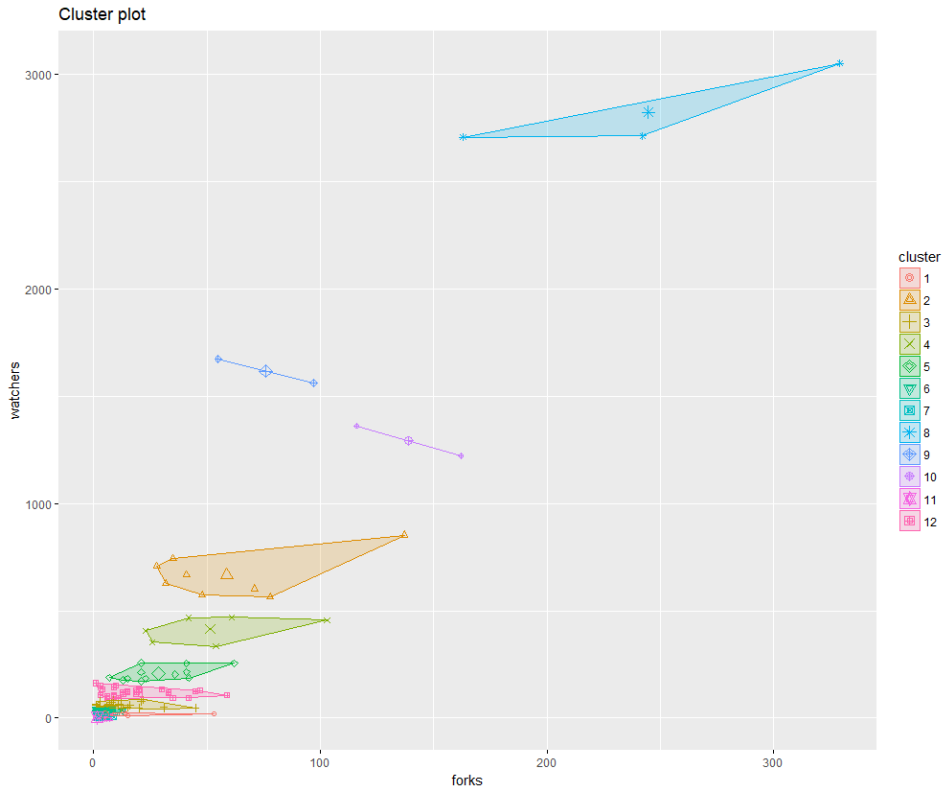
第一群: JavaScript(100%)。

第三群: JavaScript(67%)、C(33%)

觀察 table 後，不管是在哪一群，JavaScript 個數占的比例是最高的。可見這個資料集裡 JavaScript 是占大多數的。

```
> table(resultCluster3EX$cluster, orgS3$language)

   Gnuplot  Go Grammatical Framework Groff Groovy Hack Handlebars Haskell Haxe HaXe  HCL HTML   Hy  IDL Idris Inno Setup   Io Java
1        0   0                     0     0      0    0          0       0    0    0    0    0    0    0    0          0    0    0
2        0   5                     0     0      0    0          0       2    0    0    0    5    0    0    0          0    0    1
3        0   0                     0     0      0    0          0       0    0    0    0    0    0    0    0          0    0    0
4        0   0                     0     0      0    0          0       0    0    0    0    0    0    0    0          0    0    0
5        0   1                     0     0      0    0          0       0    0    0    0    1    0    0    0          0    0    1
6        0   3                     0     0      0    0          0       0    0    0    0    1    0    0    0          0    0    0
7        0   0                     0     0      0    0          0       1    0    0    0    0    0    0    0          0    0    0
8        0   0                     0     0      0    0          0       0    0    0    0    0    0    0    0          0    0    0
9        0   0                     0     0      0    0          0       0    0    0    0    0    0    0    0          0    0    0
10       0   1                     0     0      0    0          0       0    0    0    0    0    0    0    0          0    0    0
11       0 177                     0     2      4    0          0      46    1    1    6  146    1    0    1          0    1  175
12       0   0                     0     0      0    0          0       0    0    0    0    0    0    0    0          0    0    0

   JavaScript Julia Jupyter Notebook KiCad Kotlin LabVIEW Lean  Lex Limbo LiveScript LLVM Logos  LSL  Lua   M4 Makefile Mako Mask
1           6     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
2          85     0                0     0      1       0    0    0     0          0    0     1    0    0    0        1    0    0
3           2     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
4           5     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
5          26     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
6          10     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
7           1     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
8           1     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
9           3     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
10         20     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
11       1376     9               22     1      6       0    0    0     0          1    0     3    0   10    1       20    0    0
12          1     0                0     0      0       0    0    0     0          0    0     0    0    0    0        0    0    0
```

```
> fviz_cluster(resultCluster3EX, data = s3, geom = "point", stand = FALSE)
```



粉色、青綠色以上的區塊分群較明顯。

五、 心得

不同於前一次作業一分類，在分群之前無法知道資料之間會產生何種關係，必須做出結果並觀察來找出之間的關聯。雖然一開始有點擔心作出的結果會不會不知所云，不過製作的過程算是蠻有趣的，而且不管是從數值還是圖示上都可以發現一些資料間的些微關係。