生成式人工智慧 HW1

分群、分類、關聯規則

工科所一丙
姓名|洪向霖
學號|n96131281

關聯規則

Apriori

Weka-Supermarket

Weka這邊操作上則比較好上手，也有調不同的metricType，會依據指定不同的參數作為評估關聯規則的指標，給出不一樣的規則

metricType:Confidence

Associator output

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     supermarket
Instances:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

條件項目集(左) 結果項目集(右)

metricType:Lift

Associator output

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 1 -C 1.1 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     supermarket
Instances:    4627
Attributes:   217
              [list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.35 (1619 instances)
Minimum metric <lift>: 1.1
Number of cycles performed: 13

Generated sets of large itemsets:

Size of set of large itemsets L(1): 22

Size of set of large itemsets L(2): 36

Size of set of large itemsets L(3): 3

Best rules found:

 1. fruit=t 2962 ==> bread and cake=t vegetables=t 1791    conf:(0.6) < lift:(1.22)> lev:(0.07) [319] conv:(1.27)
 2. bread and cake=t vegetables=t 2298 ==> fruit=t 1791    conf:(0.78) < lift:(1.22)> lev:(0.07) [319] conv:(1.63)
 3. vegetables=t 2961 ==> bread and cake=t fruit=t 1791    conf:(0.6) < lift:(1.2)> lev:(0.07) [303] conv:(1.26)
 4. bread and cake=t fruit=t 2325 ==> vegetables=t 1791    conf:(0.77) < lift:(1.2)> lev:(0.07) [303] conv:(1.56)
 5. baking needs=t 2795 ==> margarine=t 1645    conf:(0.59) < lift:(1.19)> lev:(0.06) [262] conv:(1.23)
 6. margarine=t 2288 ==> baking needs=t 1645    conf:(0.72) < lift:(1.19)> lev:(0.06) [262] conv:(1.41)
 7. frozen foods=t 2717 ==> biscuits=t 1810    conf:(0.67) < lift:(1.18)> lev:(0.06) [280] conv:(1.31)
 8. biscuits=t 2605 ==> frozen foods=t 1810    conf:(0.69) < lift:(1.18)> lev:(0.06) [280] conv:(1.35)
 9. vegetables=t 2961 ==> fruit=t 2207    conf:(0.75) < lift:(1.16)> lev:(0.07) [311] conv:(1.41)
10. fruit=t 2962 ==> vegetables=t 2207    conf:(0.75) < lift:(1.16)> lev:(0.07) [311] conv:(1.41)
```

Weka-Supermarket

分析(各舉一個例子)

metricType:Confidence

<u>購買餅乾、冷凍食品、水果且總金額高的顧客中，有很大比例會購買麵包和蛋糕</u>

Support：條件項目集的交易總數為 788，同時滿足條件和結果的交易數為 723。

Confidence：有 **92%** 的交易同時滿足結果（購買麵包和蛋糕）

Lift：值為 1.27，與隨機購買相比，這條規則增加了 27% 的關聯性。

metricType:Lift

<u>購買水果的顧客，60% 會同時購買麵包、蛋糕和蔬菜</u>

Support：2962 筆交易包含條件；1791 筆同時滿足條件和結果。

Confidence：60% 的交易同時包含條件和結果。

Lift：值為 1.22，表示該規則的相關性比隨機情況高 22%。

Python-Red Wine

與iris不同的是，uci可以藉由模組dataset(id=53)直接抓數據但分析wine的數據要手動分出feature與target以及type的轉換因此在資料前處理需要花點時間。但一樣是將各feature分成3個區間來找關聯規則。

Data Preparing

```python
# from ucimlrepo import fetch_ucirepo
import pandas as pd

# fetch dataset
# iris = fetch_ucirepo(id=53)

# data (as pandas dataframes)
row_data=pd.read_csv('winequality-red.csv')
row_data=pd.DataFrame(row_data)
X = row_data.iloc[:, :11+1]
y= row_data[['quality category']]
```

3] ✓ 0.0s

```python
# 格式化印出
i = 1
for item in association_results:
    base_ls = list(item[2][0][0])
    add_ls = list(item[2][0][1])
    print("Rule {}: ".format(i) + str(base_ls) + " -> " + str(add_ls))
    print("Support: " + str(item[1]))
    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("==================================")

    if len(item[2]) == 2:
        base_ls = list(item[2][1][0])
        add_ls = list(item[2][1][1])
        print("Rule {}: ".format(i) + str(base_ls) + " -> " + str(add_ls))
        print("Support: " + str(item[1]))
        print("Confidence: " + str(item[2][1][2]))
        print("Lift: " + str(item[2][1][3]))
        print("==================================")
    i+=1
```

✓ 0.0s

```
Rule 1: ['(0.607, 1.093]_va'] -> ['(-0.001, 0.333]_ca']
Support: 0.2782989368355222
Confidence: 0.9026369168356998
Lift: 1.4304424479883886
==================================
Rule 2: ['(-0.001, 0.333]_ca'] -> ['(4.589, 8.367]_fa']
Support: 0.5128205128205128
Confidence: 0.8126858275520317
Lift: 1.341057418220535 3
==================================
Rule 2: ['(4.589, 8.367]_fa'] -> ['(-0.001, 0.333]_ca']
Support: 0.5128205128205128
Confidence: 0.8462332301341589
Lift: 1.3410574182205353
==================================
Rule 3: ['(0.333, 0.667]_ca'] -> ['(0.119, 0.607]_va']
Support: 0.31957473420888055
Confidence: 0.9207207207207206
Lift: 1.3408309949293553
==================================
Rule 4: ['(6.333, 8.0]_qual'] -> ['(0.119, 0.607]_va']
Support: 0.12195121951219512
Confidence: 0.8986175115207373
Lift: 1.308642441640855
==================================
...
Support: 0.06879299562226392
Confidence: 0.990990990990991
Lift: 7.474502804691483
==================================
```

Python-Red Wine

分析:

就Confidence最高的兩條規則來解讀:

1. 在數據中，當揮發性酸度較高時（0.607-1.093），檸檬酸通常處於較低的區間（-0.001-0.333）。這可能是製酒過程中某種特徵的表現，表明揮發性酸度和檸檬酸之間存在一定的聯繫。

2. 在數據中，當檸檬酸含量較低時（-0.001-0.333），固定酸度通常處於中高區間（4.589-8.367）。這可能反映了檸檬酸和固定酸度之間的潛在化學關聯，可能與葡萄品質或發酵過程有關。

(Lift值為 1.34，表示條件和結果之間的相關性比隨機出現的情況高出 34%，是一條有價值的規則。)

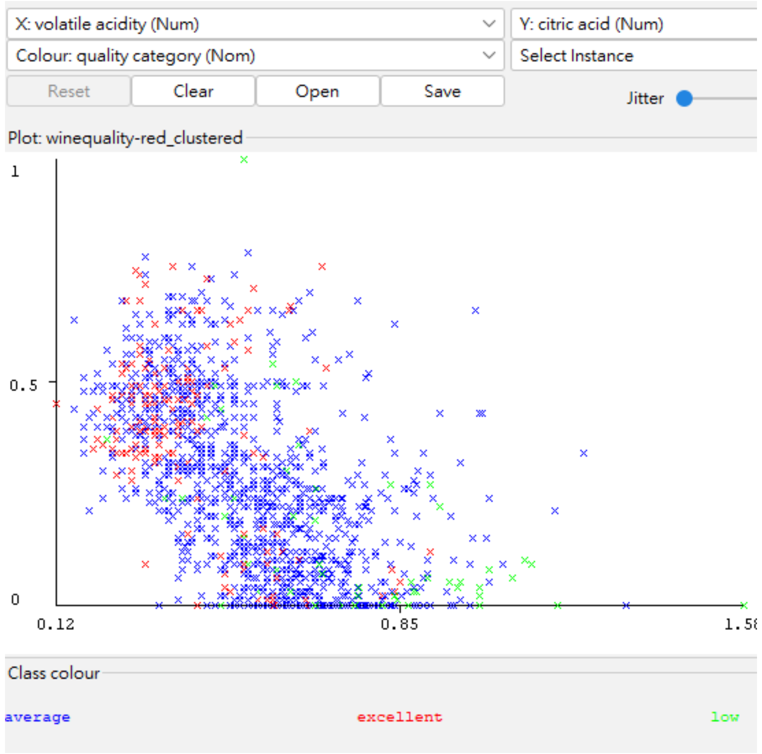
分群法

Kmeans & Hierarchical Clustering

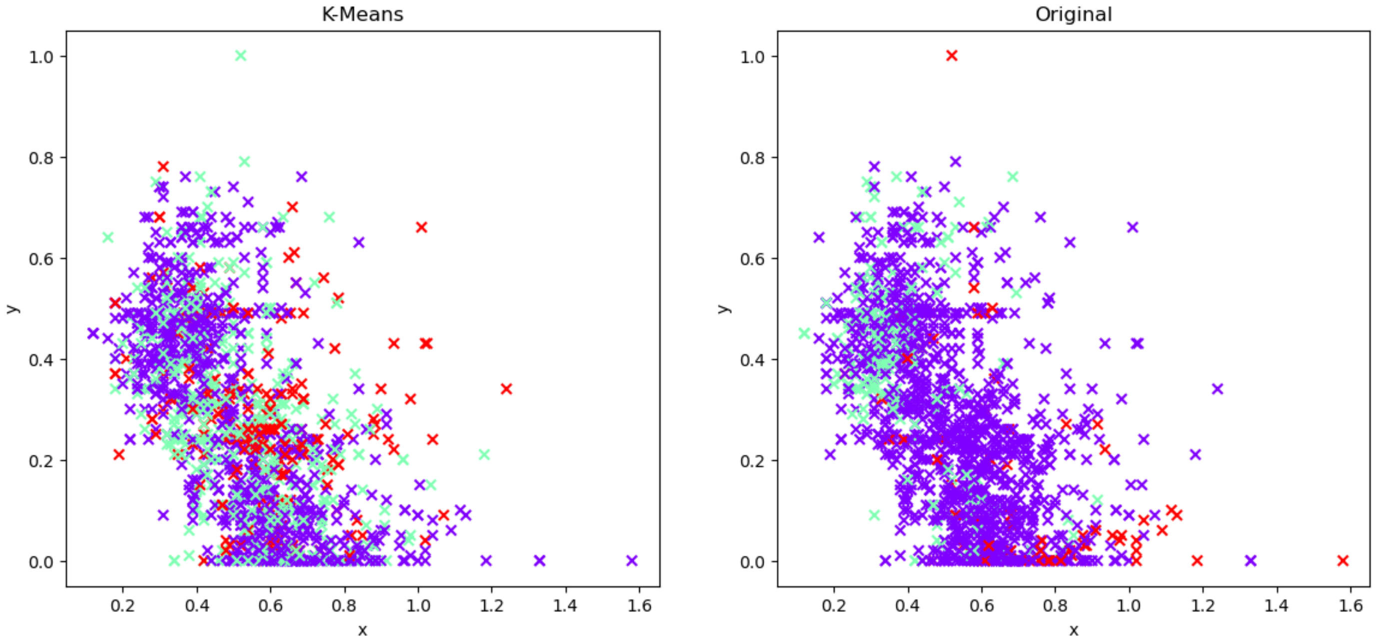
Kmeans

在資料的選擇上，是根據關聯規則給出的第一項(Confidence最高的Rule1)作為XY軸的選擇
就演算法而言，明顯可見Weka幾乎都分對 (Python的紅綠跟Weka剛好相反)

Weka



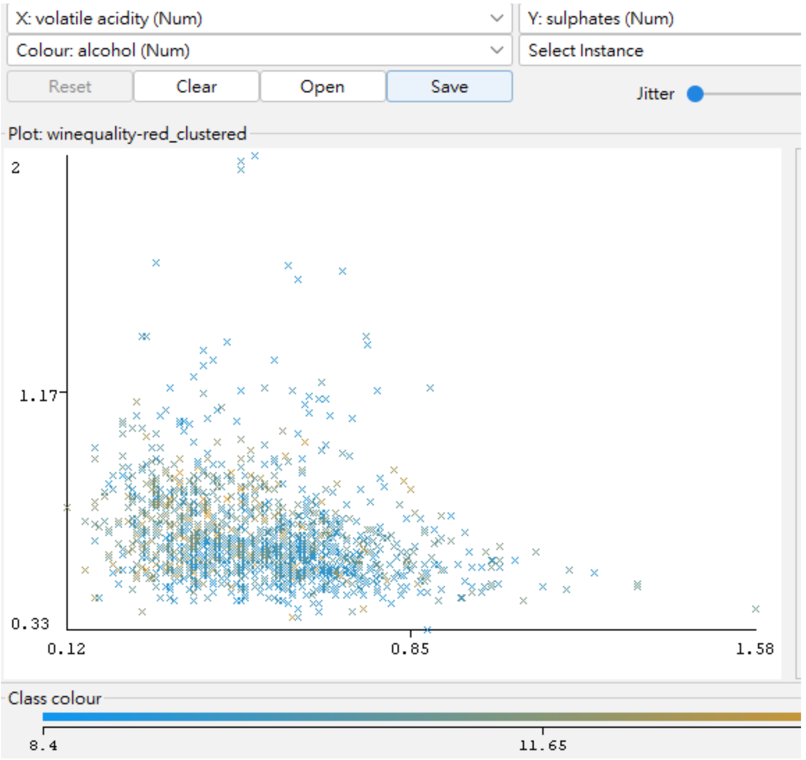
Python



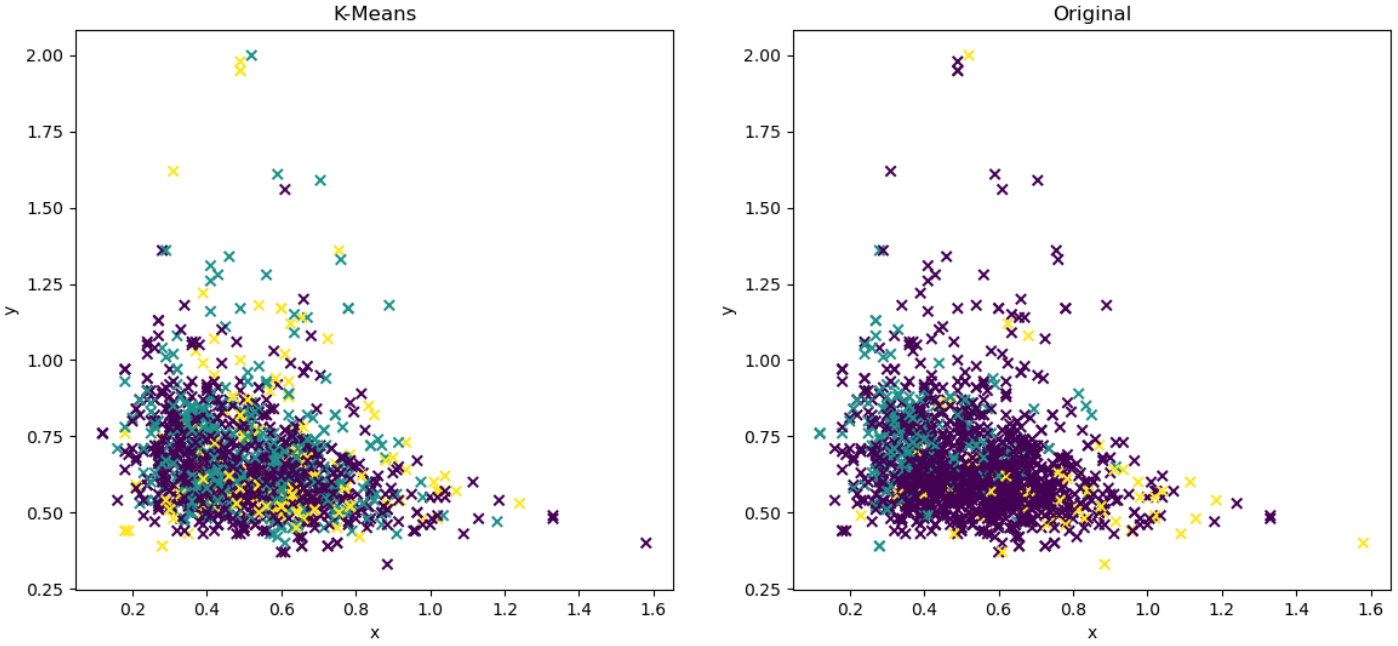
Kmeans

這邊是選用 X:volatile acidity ; Y:sulphates ; Colour : alcohol
可以看出一樣是Weka在分群表現上比Python更精準，更貼近Original的結果

Weka



Python



Kmeans (Code)

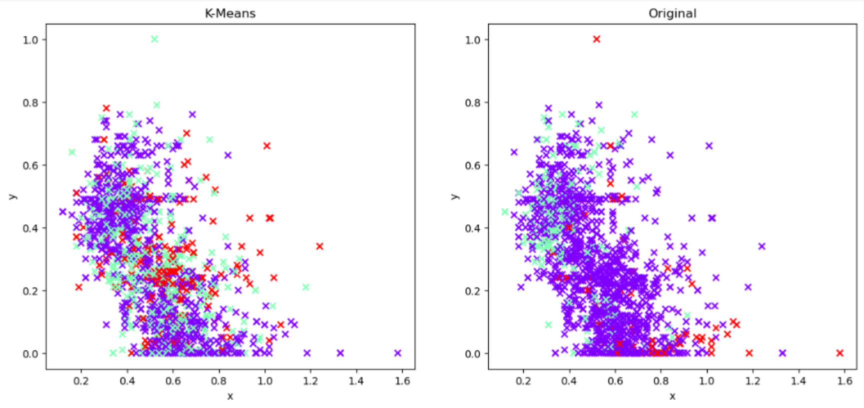
```
y9=np_wine_X[:, 9]-np.mean(np_wine_X[:, 9])/np.std(np_wine_X[:, 9])
y9
```

✓ 0.0s 開啟 Data Wrangler 中的 'y9'

```
array([-3.32393883, -3.20393883, -3.23393883, ..., -3.13393883,
       -3.17393883, -3.22393883])
```

```
# plt.subplots(橫列數量, 直行數量)
# sharey=True 共享y軸
f, axes = plt.subplots(1, 2, figsize=(14,6))
axes[0].set_title('K-Means')
# 選擇petal length/petal width兩個維度來畫點狀分布圖
axes[0].scatter(np_wine_X[:, 1], np_wine_X[:, 2], c=km.labels_, cmap='rainbow',marker='x')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')

axes[1].set_title('Original')
axes[1].scatter(np_wine_X[:, 1], np_wine_X[:, 2], c=new_y['quality category'].to_numpy(), cmap='rainbow',marker='x')
axes[1].set_xlabel('x')
axes[1].set_ylabel('y')

plt.savefig('kmeans-1.9.10.png')
```

✓ 0.3s



評估模型分群成效

```
# 此數值越接近1，表示群內差異小，且不同分群之間的差異大 => 好的分群
# 此數值越接近0，表示群內差異大，且不同分群之間的差異小 => 壞的分群
metrics.silhouette_score(X, km.labels_)
```

[13] ✓ 0.0s

```
0.52048520454398
```

```
def get_kscore(k):
    km = cluster.KMeans(n_clusters=k)
    km.fit(X)
    return metrics.silhouette_score(X, km.labels_)
```

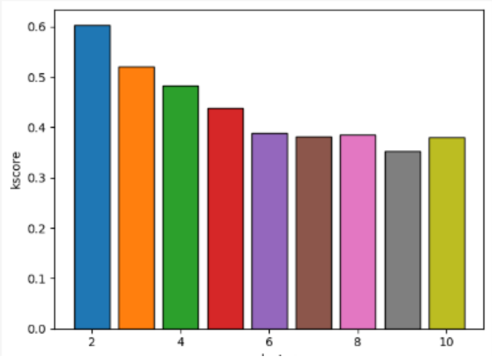
[14] ✓ 0.0s

```
# 迭代找到最佳的分群數量
for k in range(2, 11):
    plt.bar(k, get_kscore(k),edgecolor="black")

plt.xlabel('n_cluster')
plt.ylabel('kscore')
```

[15] ✓ 0.6s

```
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
c:\Users\n9613\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is know
  warnings.warn(
```

```
Text(0, 0.5, 'kscore')
```



riodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-las

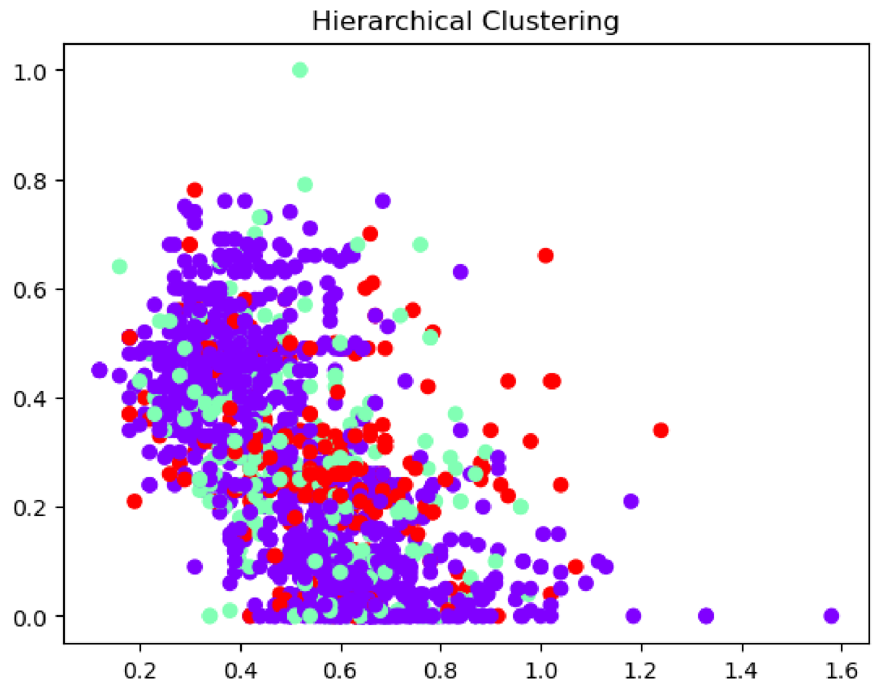
Clusterer output

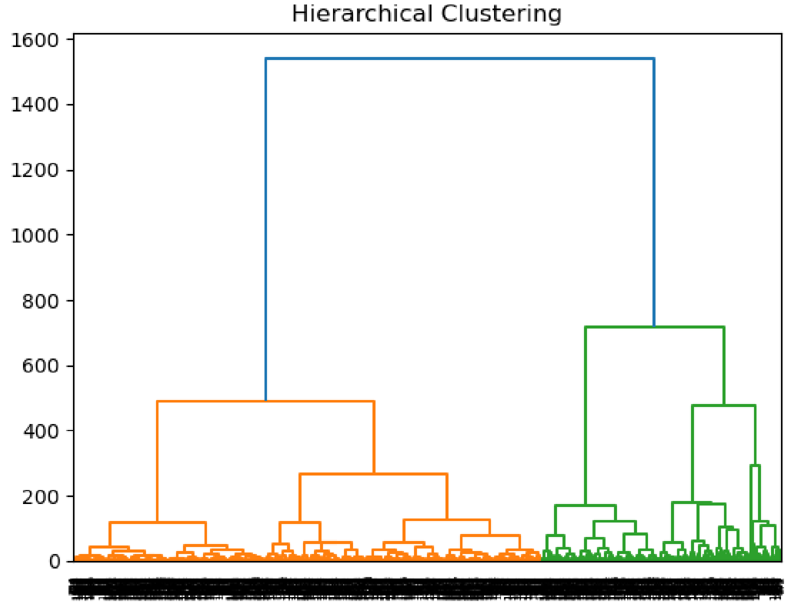
```
             quality
             quality category
Test mode:   evaluate on training data


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 16
Within cluster sum of squared errors: 556.9144996717787

Initial starting points (random):

Cluster 0: 7.7,0.49,0.26,1.9,0.062,9,31,0.9966,3.39,0.64,9.6,5,average
Cluster 1: 5.4,0.74,0,1.2,0.041,16,46,0.99258,4.01,0.59,12.5,6,average

Missing values globally replaced with mean/mode

Final cluster centroids:
                                       Cluster#
Attribute                 Full Data          0          1
                           (1599.0)    (632.0)    (967.0)
=========================================================
ï»¿fixed acidity             8.3196      9.741     7.3907
volatile acidity             0.5278     0.4034     0.6092
citric acid                   0.271     0.4654     0.1439
residual sugar               2.5388     2.7011     2.4327
chlorides                    0.0875     0.0955     0.0822
free sulfur dioxide         15.8749    14.2801    16.9173
total sulfur dioxide        46.4678    40.3449    50.4695
density                      0.9967     0.9974     0.9963
pH                           3.3111     3.2185     3.3716
sulphates                    0.6581     0.7339     0.6086
alcohol                      10.423    10.7098    10.2356
quality                       5.636     5.9747     5.4147
quality category            average    average    average




Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       632 ( 40%)
1       967 ( 60%)
```

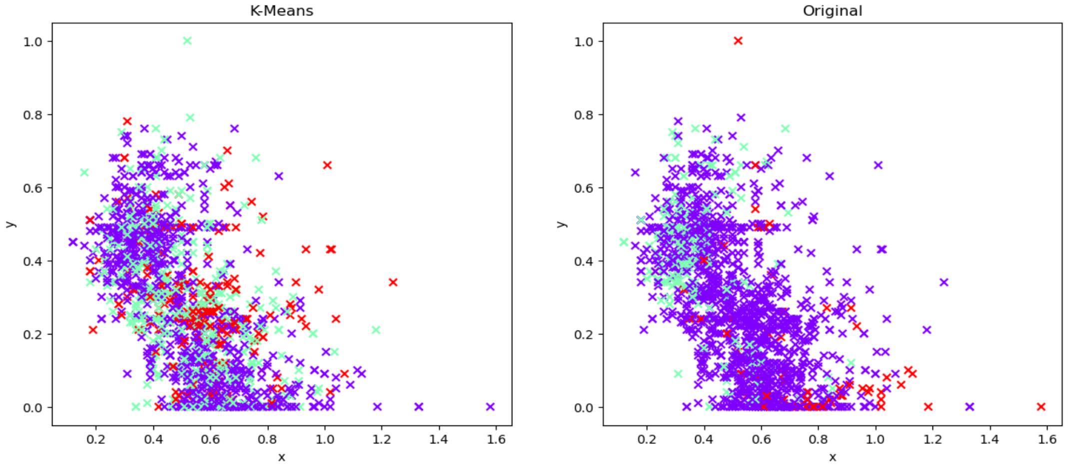
Hierarchical Clustering

Python的部分，可以跟kmeans做對比

可以看出跟kmeans分的非常相似(直接設定3群)
但離實際值(average,low,excellent)上有進步空間



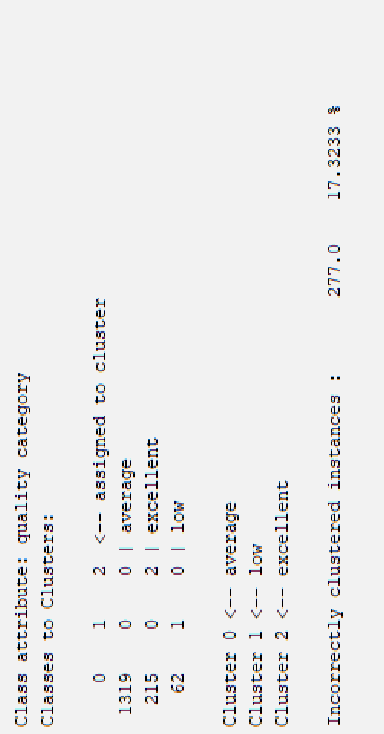




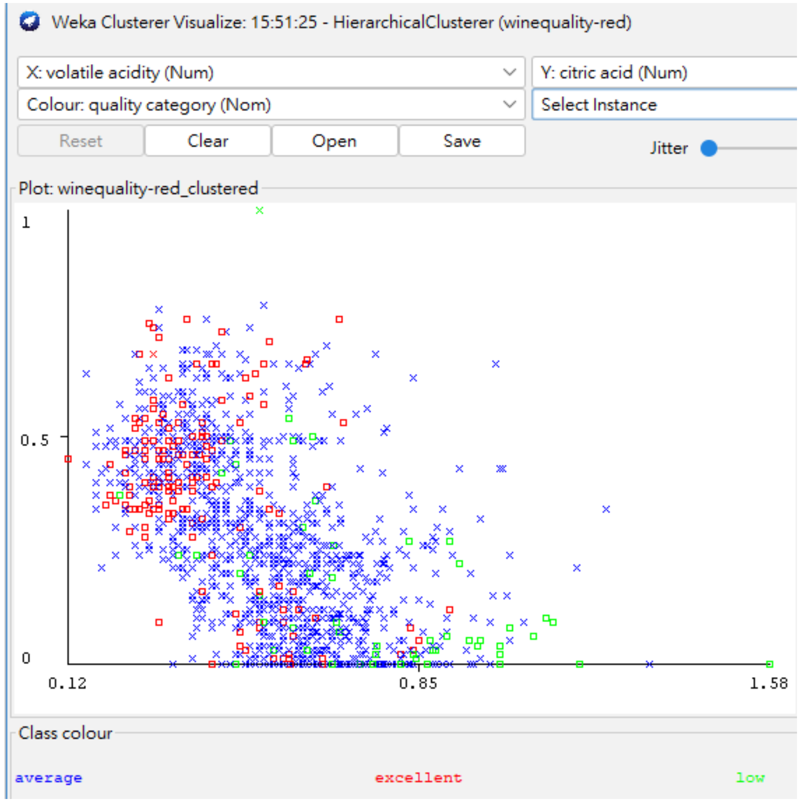
Hierarchical Clustering

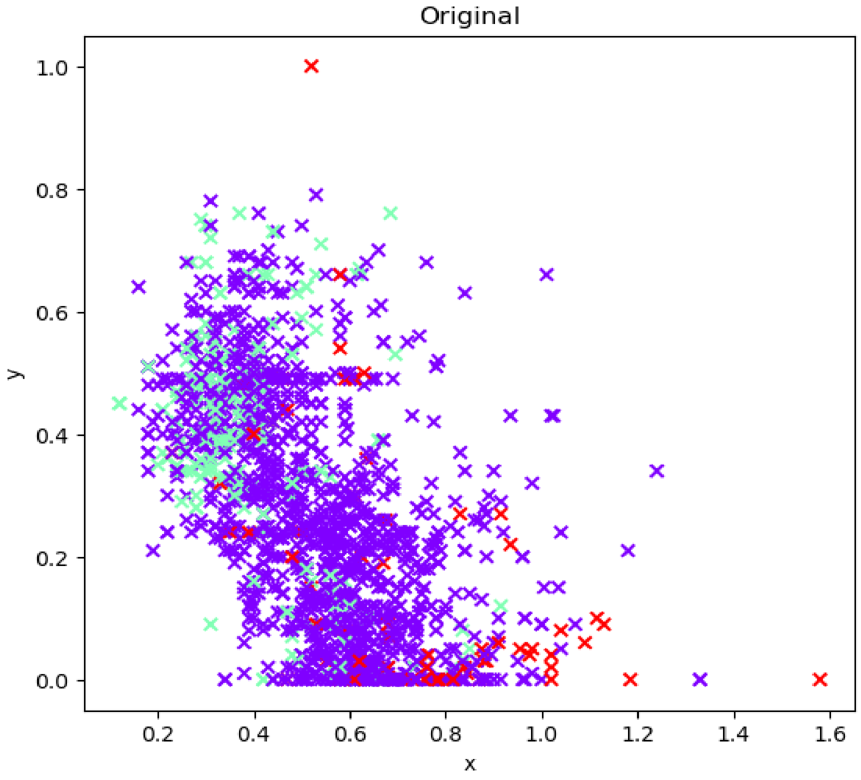
Weka的部分，一樣是表現的比較優異（Python的紅綠跟Weka剛好相反）

也有嘗試調不同的距離計算方式，結果大同小異

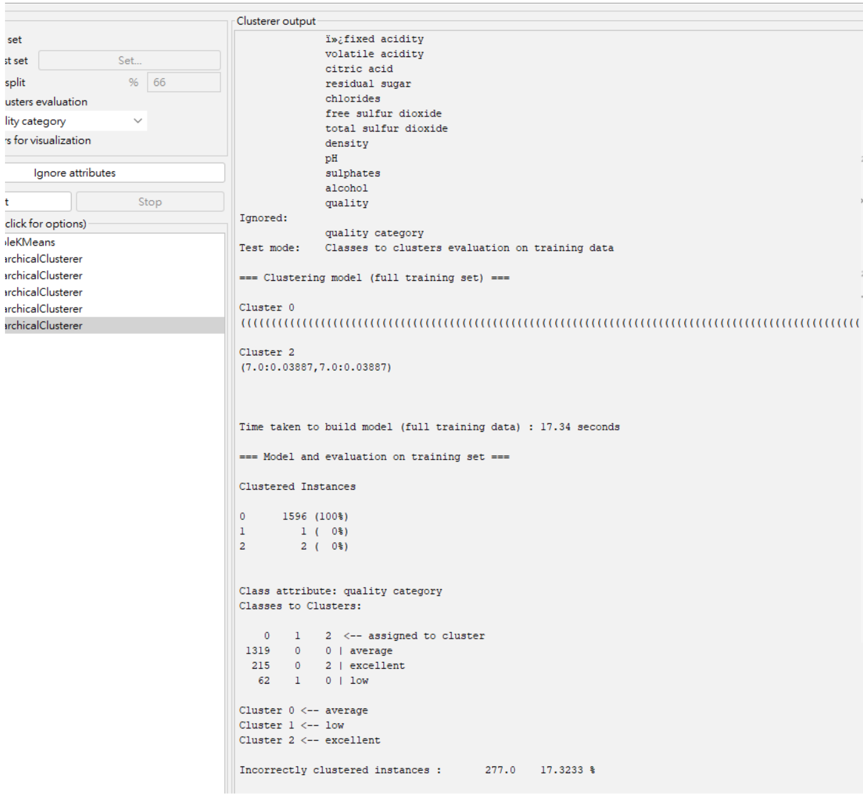


```
Class attribute: quality category
Classes to Clusters:

    0    1    2  <-- assigned to cluster
 1319    0    0 | average
  215    0    2 | excellent
   62    1    0 | low

Cluster 0 <-- average
Cluster 1 <-- low
Cluster 2 <-- excellent

Incorrectly clustered instances :     277.0     17.3233 %
```



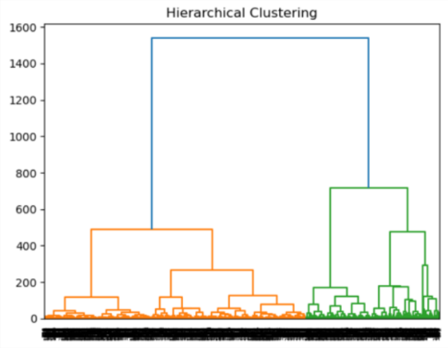


Hierarchical Clustering (Code)




```python
dis=sch.linkage(X,metric='euclidean',method='ward')
```
✓ 0.0s

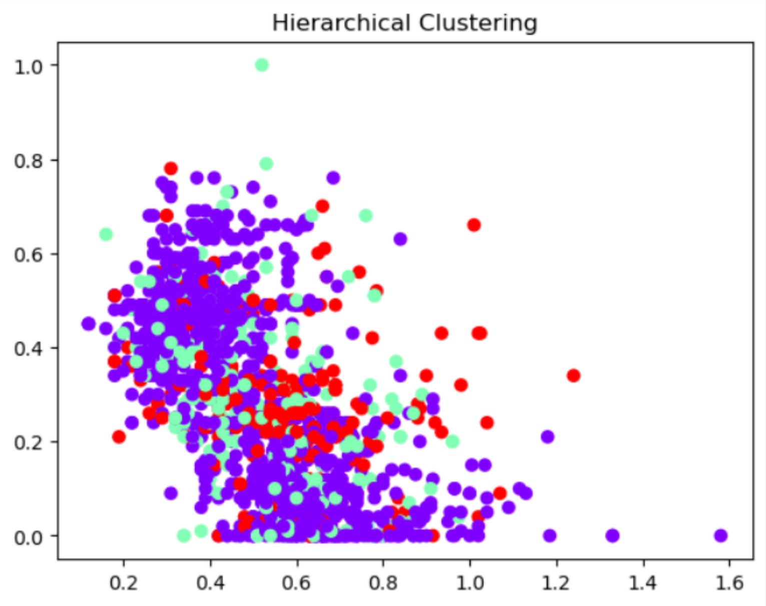
```python
sch.dendrogram(dis)
plt.title('Hierarchical Clustering')
plt.show()
# x_axis_labels.txt
```
✓ 7.1s




```python
k=3
clusters=sch.fcluster(dis,k,criterion='maxclust')
```
✓ 0.0s

```python
plt.scatter(X[:,1],X[:,2],c=clusters,cmap='rainbow',marker='o')
plt.title('Hierarchical Clustering')
```
✓ 0.1s

```
Text(0.5, 1.0, 'Hierarchical Clustering')
```




```python
ml=AgglomerativeClustering(n_clusters=3,metric='euclidean',linkage='ward')
```
✓ 0.0s

分類法

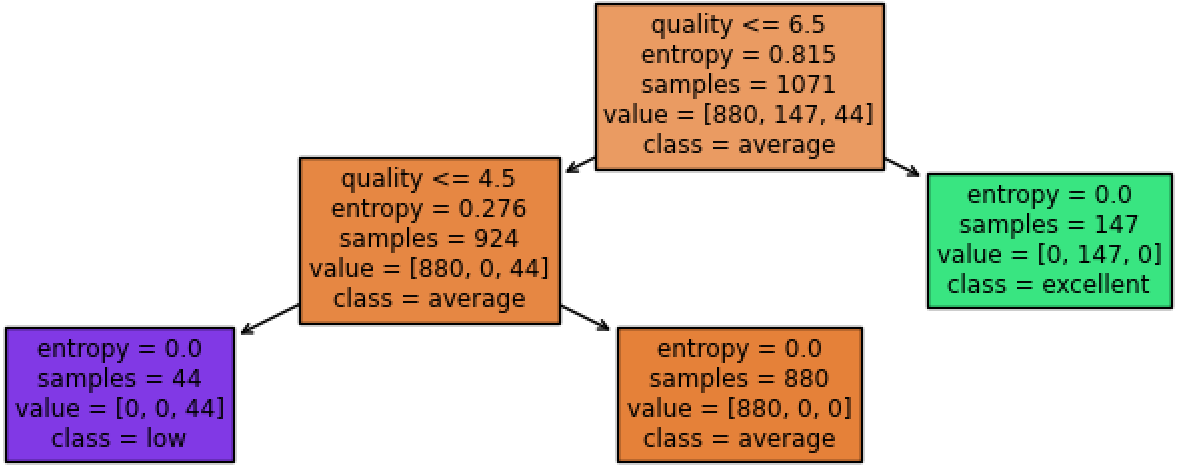
ID3 J48 Randomforest

Decision Tree

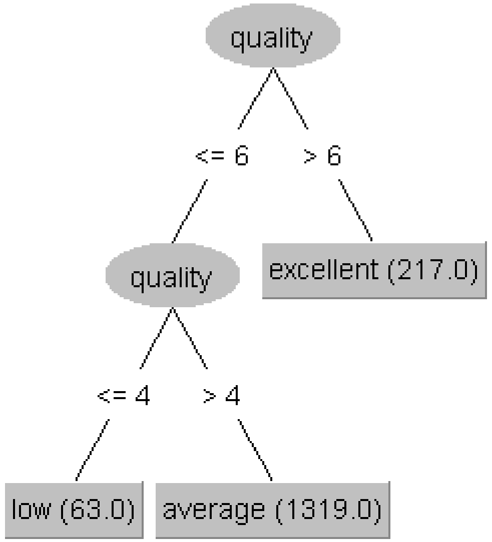
```
J48 pruned tree
------------------

quality <= 6
|   quality <= 4: low (63.0)
|   quality > 4: average (1319.0)
quality > 6: excellent (217.0)

Number of Leaves  :     3

Size of the tree :      5
```

Python使用的ID3是依據資訊理論作為分割規則，而Weka的J48則是其改良版

畫出來的的決策樹具有一樣的維度，但在中間節點(測試的條件)的分類上有些許的不同(ex.>6 & >6.5)



Python



Weka

Decision Tree (Code)

Visualization

決策樹視覺化

```
!pip install pydotplus
```

[18] ✓ 4.1s

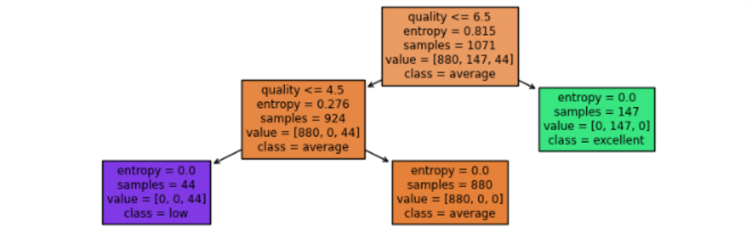
```
Requirement already satisfied: pydotplus in c:\users\n9613\anaconda3\lib\site-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\n9613\anaconda3\lib\site-packages (from pydotplus) (3.0.9)
```

```
from IPython.core.display import Image
import pydotplus
from sklearn.tree import plot_tree
```

[19] ✓ 0.0s

```
# dot_data = tree.export_graphviz(dtree,
#                 feature_names=['fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dioxide','total sulfur dioxide','density','pH','sulphates','alcohol','quality'],
#                 class_names=class_name_ls)

# graph = pydotplus.graph_from_dot_data(dot_data)
# Image(graph.create_png())

plt.figure(figsize=(10, 3))
plot_tree(dtree,
          feature_names=['fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dioxide','total sulfur dioxide','density','pH','sulphates','alcohol','quality'],
          class_names=class_name_ls,
          filled=True)
plt.show()

plt.savefig('ID3.png')
```

[20] ✓ 0.1s




```
<Figure size 640x480 with 0 Axes>
```

Cross validation

這是一種用於評估機器學習模型性能的技術，透過多次劃分資料集，以更全面的評估模型的泛用性與穩定性

```
dtree_cv = tree.DecisionTreeClassifier(criterion="entropy")
```

[21] ✓ 0.0s

```
scores = model_selection.cross_val_score(dtree_cv, X, new_y, cv=10, scoring='accuracy')
scores
```

[22] ✓ 0.0s 開啟 Data Wrangler 中的 'scores'

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
mean_scores = []
var_scores = []
for n in range(10):
    # cross_validate(dtree, X, y, cv=分成多少等份) 隨機抽取資料進行cv次驗證
    scores = model_selection.cross_val_score(dtree_cv, X, new_y, cv=10, scoring='accuracy')
    mean_scores.append(scores.mean())#取這10次的平均值

    var_scores.append(scores.var())#取這10次的方差
```

[23] ✓ 0.4s

```
mean_scores
scores.mean()
```

[27] ✓ 0.0s

```
1.0
```

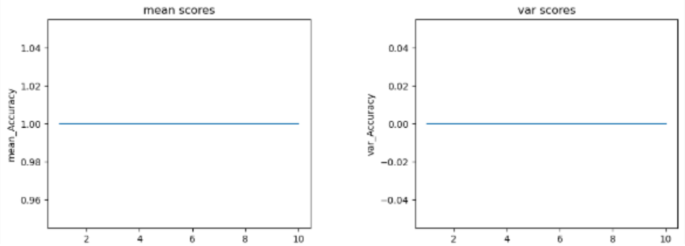
```
var_scores
scores.var()
```

[25] ✓ 0.0s

```
0.0
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

# 平均值
# 平均值越接近1，表示模型在平均情況下表現得越好
# 也就是說，在取到平衡數據的情況下，模型的表現良好
axes[0].plot(range(1,11), mean_scores)
axes[0].set_xlabel('K')
axes[0].set_ylabel('mean_Accuracy')
axes[0].set_title('mean scores')

# 變異數=>(X-mean)^2的總和取平均
# 變異數(方差)越小，代表模型性能越穩定
# 也就是說，即使取到不平衡的訓練集，對整體模型的影響也較小，模型相對穩定
axes[1].plot(range(1,11), var_scores)
axes[1].set_xlabel('K')
axes[1].set_ylabel('var_Accuracy')
axes[1].set_title('var scores')

plt.show()
```

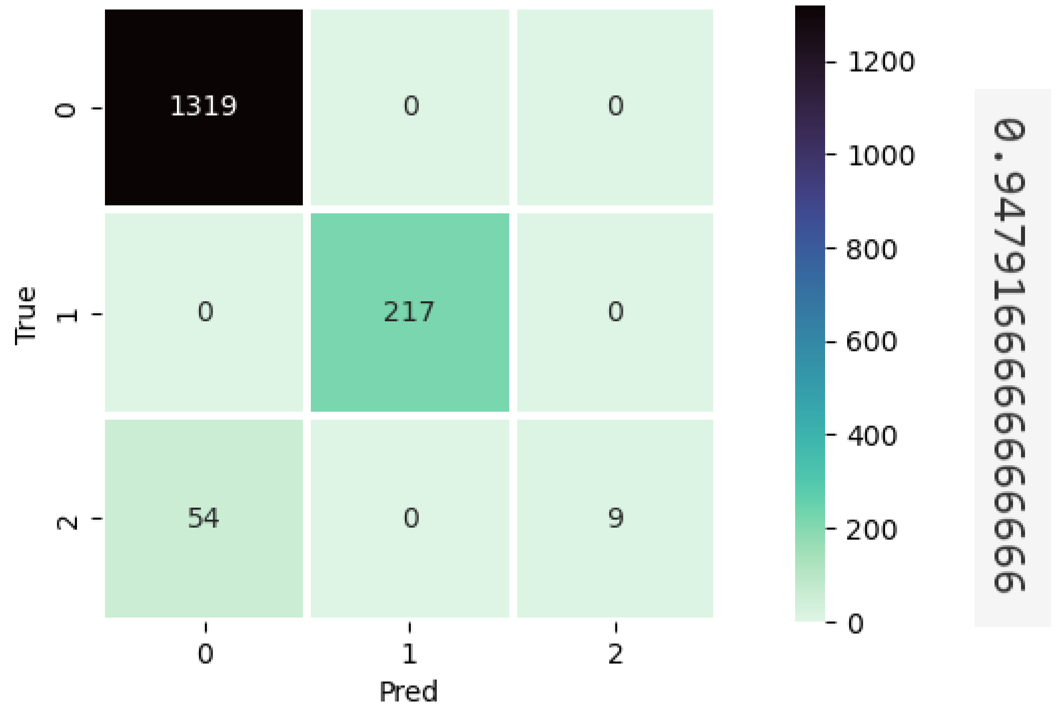
[26] ✓ 0.2s



Random Forest

相對ID3，隨機森林沒有視覺化的決策樹，但可以以混淆矩陣看出模型分類的情況。經過幾次實驗，可以看出Weka幾乎都是全對，Python則分類上的錯誤率比較高(非對角線有值)

Python

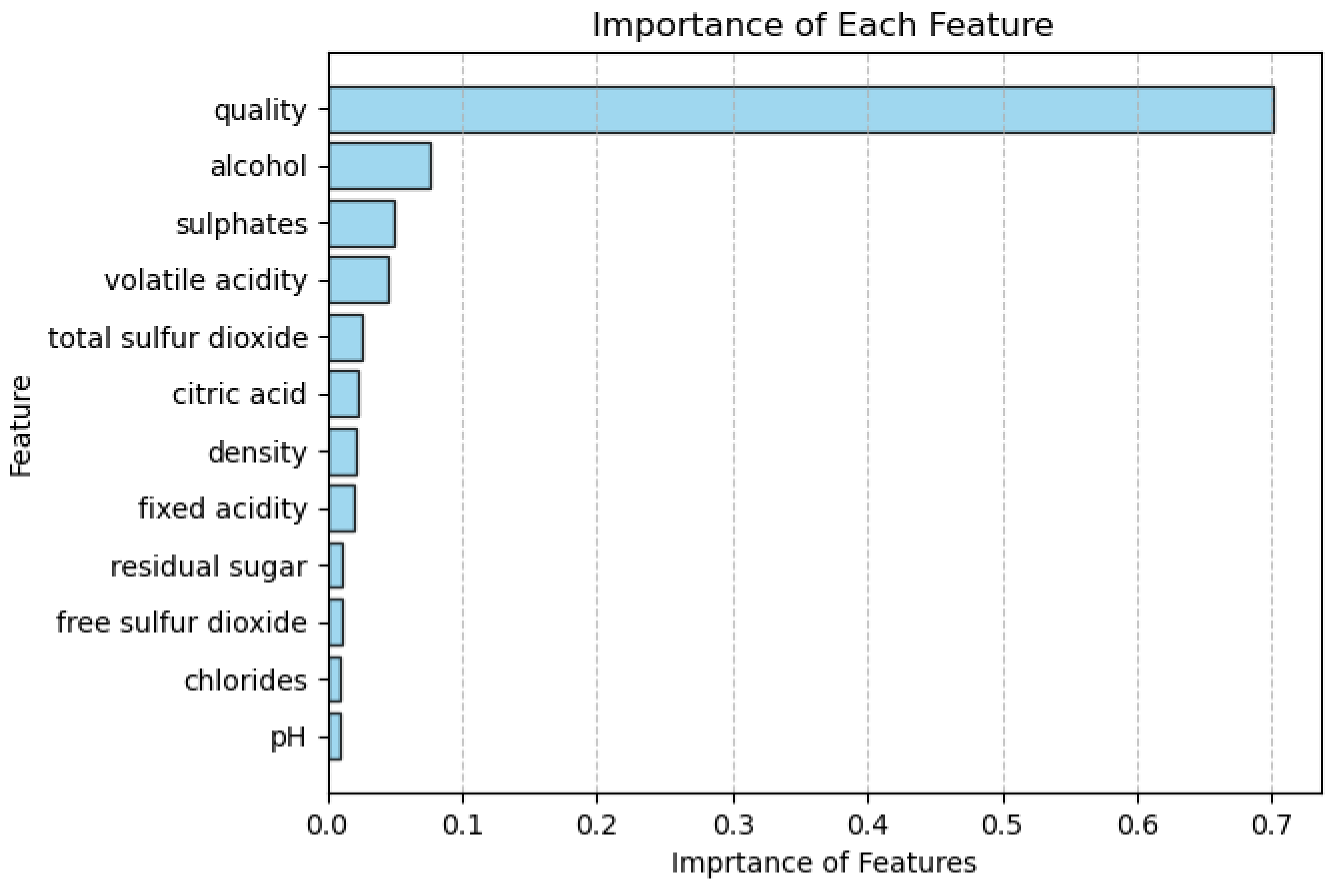


Weka

```
=== Confusion Matrix ===

    a    b    c   <-- classified as
 1319    0    0 |    a = average
    0  217    0 |    b = excellent
    0    0   63 |    c = low
```

Random Forest

與Weka不同的是，Python能看出各feature量化的重要性

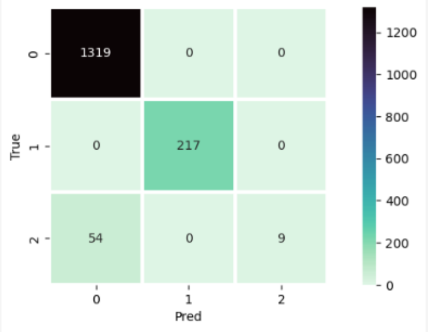


Random Forest (Code)

```
Classifier output
              density
              pH
              sulphates
              alcohol
              quality
              quality category
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

quality <= 6
|   quality <= 4: low (63.0)
|   quality > 4: average (1319.0)
quality > 6: excellent (217.0)

Number of Leaves  :     3

Size of the tree :      5


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1599              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances             1599

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     average
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     excellent
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     low
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

    a    b    c   <-- classified as
 1319    0    0 |    a = average
    0  217    0 |    b = excellent
    0    0   63 |    c = low
```

```python
plt.figure(figsize=(10, 4))
sns.heatmap(cm, square=True, annot=True, fmt='d', cmap='mako_r', linecolor='white', linewidths=1.5, cbar=True)
plt.xlabel('Pred')
plt.ylabel('True')
plt.show()

plt.savefig('RandomForest.png')
```

✓ 0.1s




```python
zip(imp,names)
imp, names = zip(*sorted(zip(imp,names)))
# plt.barh(range(len(names)),imp,align="center")
plt.barh(range(len(names)), imp, align="center", color="skyblue", edgecolor="black", alpha=0.8)
plt.yticks(range(len(names)),names)
plt.xlabel("Imprtance of Features")
plt.ylabel("Feature")
plt.title("Importance of Each Feature")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.show()
```

✓ 0.1s