# 統計學習與資料探勘 期末報告

### 順序：15

### 組員：黃偉柏 温宏岳

### 題目：Taiwan's Air Quality Data by Hours

### (台灣各小時空氣指數狀態資料分析--分類分群)

參考資料：[https://www.kaggle.com/datasets/yenruchen/taiwans-air-quality-data-by-hours (https://www.kaggle.com/datasets/yenruchen/taiwans-air-quality-data-by-hours)](https://www.kaggle.com/datasets/yenruchen/taiwans-air-quality-data-by-hours)

## Data Pre-Processing

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

import seaborn as sns
import sklearn
import graphviz

from sklearn import datasets, cluster, datasets, metrics, tree, neighbors
from sklearn.model_selection import train_test_split, learning_curve, cross_val_score, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler, scale

from imblearn.over_sampling import SMOTE
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, plot_tree, export_graphviz
from xgboost import XGBClassifier
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

from tabulate import tabulate

from mlxtend.plotting import plot_decision_regions

from sklearn.metrics import mean_squared_error, accuracy_score, confusion_matrix, ConfusionMatrixDisplay,\
classification_report,roc_curve, auc, RocCurveDisplay, precision_score, recall_score, f1_score

%matplotlib inline

import statsmodels.api as sm
import statsmodels.formula.api as smf
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df = pd.read_csv("file.csv")
df2 = df.drop(columns=["Pollutant","Unit","Longitude","Latitude","SiteId"])
df3=df2.dropna()
df4=df3[df3['County']=='高雄市']
df5=df4[(df4['DataCreationDate']=='2021-12-26 23:00')|(df4['DataCreationDate']=='2021-12-26 21:00')|
(df4['DataCreationDate']=='2021-12-26 19:00')|(df4['DataCreationDate']=='2021-12-26 17:00')|
(df4['DataCreationDate']=='2021-12-26 15:00')|(df4['DataCreationDate']=='2021-12-26 13:00')|
(df4['DataCreationDate']=='2021-12-26 11:00')|(df4['DataCreationDate']=='2021-12-26 09:00')|
(df4['DataCreationDate']=='2021-12-26 07:00')|(df4['DataCreationDate']=='2021-12-26 05:00')|
(df4['DataCreationDate']=='2021-12-26 03:00')|(df4['DataCreationDate']=='2021-12-26 01:00')]
df5 = df5.drop(columns=["County"])

labelencoder = LabelEncoder()
df_le = pd.DataFrame(df5)
df_le['SiteName'] = labelencoder.fit_transform(df5['SiteName'])
df_le['Status'] = labelencoder.fit_transform(df5['Status'])
df_le['DataCreationDate'] = labelencoder.fit_transform(df5['DataCreationDate'])

df_le.rename(columns={'O3_8hr': 'O38hr'}, inplace=True)
df_le.rename(columns={'PM2.5': 'PM25'}, inplace=True)
df_le.rename(columns={'PM2.5_AVG': 'PM25AVG'}, inplace=True)
df_le.rename(columns={'SO2_AVG': 'SO2AVG'}, inplace=True)
df_le = pd.DataFrame(df_le,dtype=np.float)
```

In [3]:

```
df_le['SiteName'] = df_le['SiteName'].astype(int)
df_le['Status'] = df_le['Status'].astype(int)
df_le['DataCreationDate'] = df_le['DataCreationDate'].astype(int)
df_le
```

Out[3]:

| | SiteName | AQI | Status | SO2 | CO | O3 | O38hr | PM10 | PM25 | NO2 | NOx | NO | WindSpeed | WindDirec | DataCreationDate | CO_8hr | PM25AVG | PI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **3506558** | 11 | 48.0 | 2 | 1.4 | 0.36 | 22.0 | 20.0 | 18.0 | 10.0 | 13.9 | 14.3 | 0.4 | 1.2 | 311.0 | 0 | 0.3 | 15.0 | |
| **3506602** | 10 | 61.0 | 1 | 1.9 | 0.33 | 13.6 | 19.0 | 19.0 | 10.0 | 7.0 | 8.2 | 1.2 | 1.4 | 260.0 | 0 | 0.3 | 20.0 | |
| **3506603** | 9 | 67.0 | 1 | 1.2 | 0.37 | 18.3 | 18.0 | 29.0 | 16.0 | 13.9 | 14.3 | 0.4 | 2.8 | 7.0 | 0 | 0.4 | 22.0 | |
| **3506604** | 0 | 75.0 | 1 | 0.1 | 0.36 | 18.4 | 18.0 | 16.0 | 18.0 | 12.6 | 14.3 | 1.6 | 3.4 | 345.0 | 0 | 0.4 | 25.0 | |
| **3506605** | 12 | 87.0 | 1 | 2.0 | 0.46 | 10.4 | 11.0 | 32.0 | 25.0 | 18.4 | 20.9 | 2.4 | 0.9 | 264.0 | 0 | 0.6 | 30.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **3508449** | 1 | 61.0 | 1 | 2.3 | 0.40 | 10.0 | 20.0 | 33.0 | 16.0 | 21.1 | 22.3 | 1.1 | 2.3 | 9.0 | 11 | 0.4 | 20.0 | |
| **3508450** | 2 | 61.0 | 1 | 2.3 | 0.51 | 10.3 | 18.0 | 39.0 | 19.0 | 20.2 | 23.0 | 2.8 | 2.1 | 9.0 | 11 | 0.5 | 19.0 | |
| **3508451** | 4 | 63.0 | 1 | 1.1 | 0.50 | 8.6 | 17.0 | 32.0 | 17.0 | 23.4 | 24.2 | 0.7 | 0.9 | 27.0 | 11 | 0.5 | 20.0 | |
| **3508464** | 6 | 63.0 | 1 | 1.5 | 0.55 | 9.0 | 17.0 | 33.0 | 20.0 | 21.5 | 24.8 | 3.3 | 1.5 | 107.0 | 11 | 0.5 | 20.0 | |
| **3508473** | 11 | 41.0 | 2 | 1.0 | 0.34 | 15.4 | 20.0 | 18.0 | 10.0 | 14.6 | 15.8 | 1.2 | 0.9 | 100.0 | 11 | 0.3 | 13.0 | |

155 rows × 19 columns

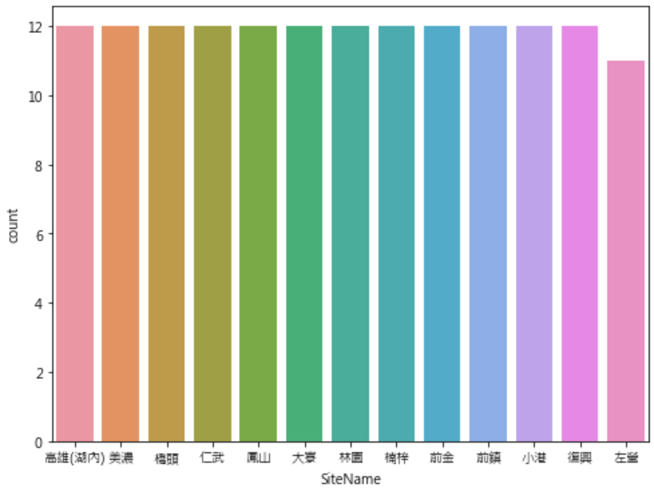
# EDA

## Correspondence of Label

## Site Name

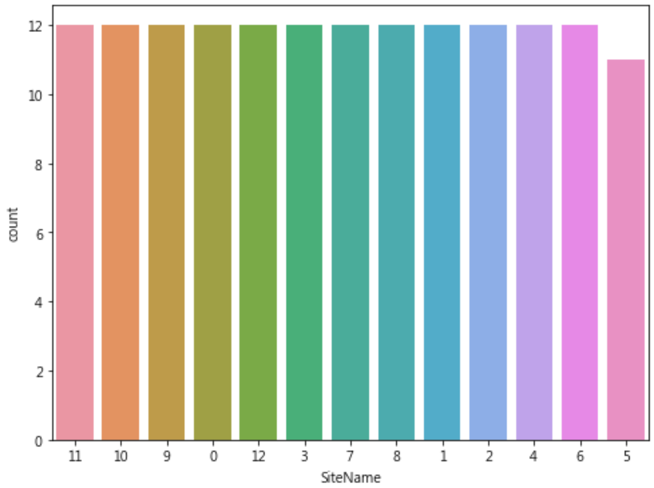
In [4]:

```
plt.figure(figsize = (8,6))
sns.countplot(x = 'SiteName',data = df5,order = df5['SiteName'].value_counts().index)
plt.rcParams['font.sans-serif'] = ['Microsoft JhengHei']
plt.rcParams['axes.unicode_minus'] = False
```



In [5]:

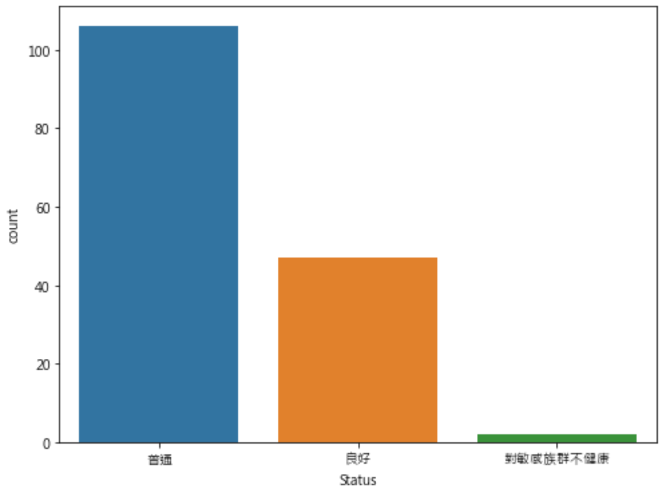
```
plt.figure(figsize = (8,6))
sns.countplot(x = 'SiteName',data = df_le,order = df_le['SiteName'].value_counts().index);
```




## Status

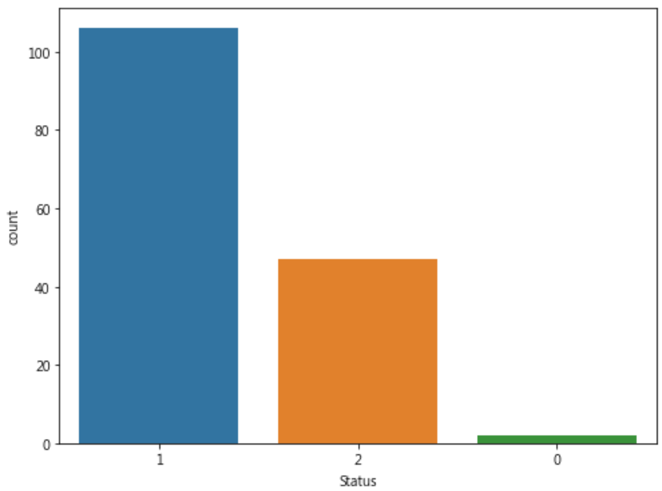
In [6]:

```
plt.figure(figsize = (8,6))
sns.countplot(x = 'Status',data = df5,order = df5['Status'].value_counts().index)
plt.rcParams['font.sans-serif'] = ['Microsoft JhengHei']
plt.rcParams['axes.unicode_minus'] = False
```




In [7]:

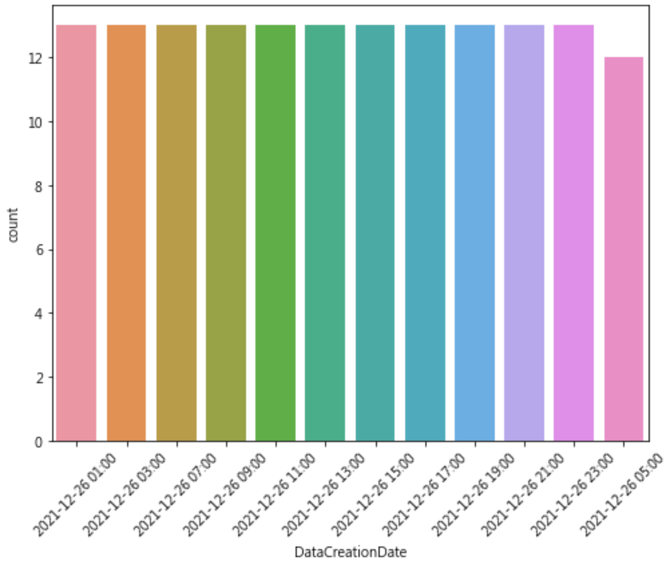
```
plt.figure(figsize = (8,6))
sns.countplot(x = 'Status',data = df_le,order = df_le['Status'].value_counts().index);
```



## Data Creation Date

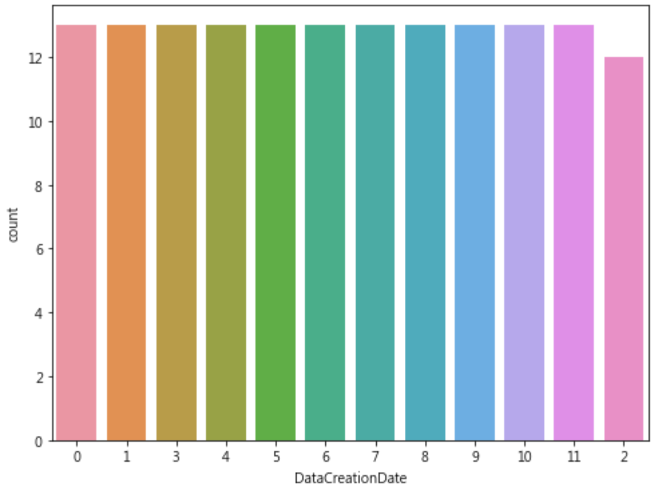
In [8]:

```
plt.figure(figsize = (8,6))
sns.countplot(x = 'DataCreationDate',data = df5,order = df5['DataCreationDate'].value_counts().index)
plt.xticks(rotation=45)
plt.rcParams['font.sans-serif'] = ['Microsoft JhengHei']
plt.rcParams['axes.unicode_minus'] = False
```




In [9]:

```
plt.figure(figsize = (8,6))
sns.countplot(x = 'DataCreationDate',data = df_le,order = df_le['DataCreationDate'].value_counts().index);
```



In [10]:

```
df_le.dtypes
```

Out[10]:

```
SiteName              int32
AQI                 float64
Status                int32
SO2                 float64
CO                  float64
O3                  float64
O38hr               float64
PM10                float64
PM25                float64
NO2                 float64
NOx                 float64
NO                  float64
WindSpeed           float64
WindDirec           float64
DataCreationDate      int32
CO_8hr              float64
PM25AVG             float64
PM10_AVG            float64
SO2AVG              float64
dtype: object
```

In [11]:

```
df_le.describe()
```

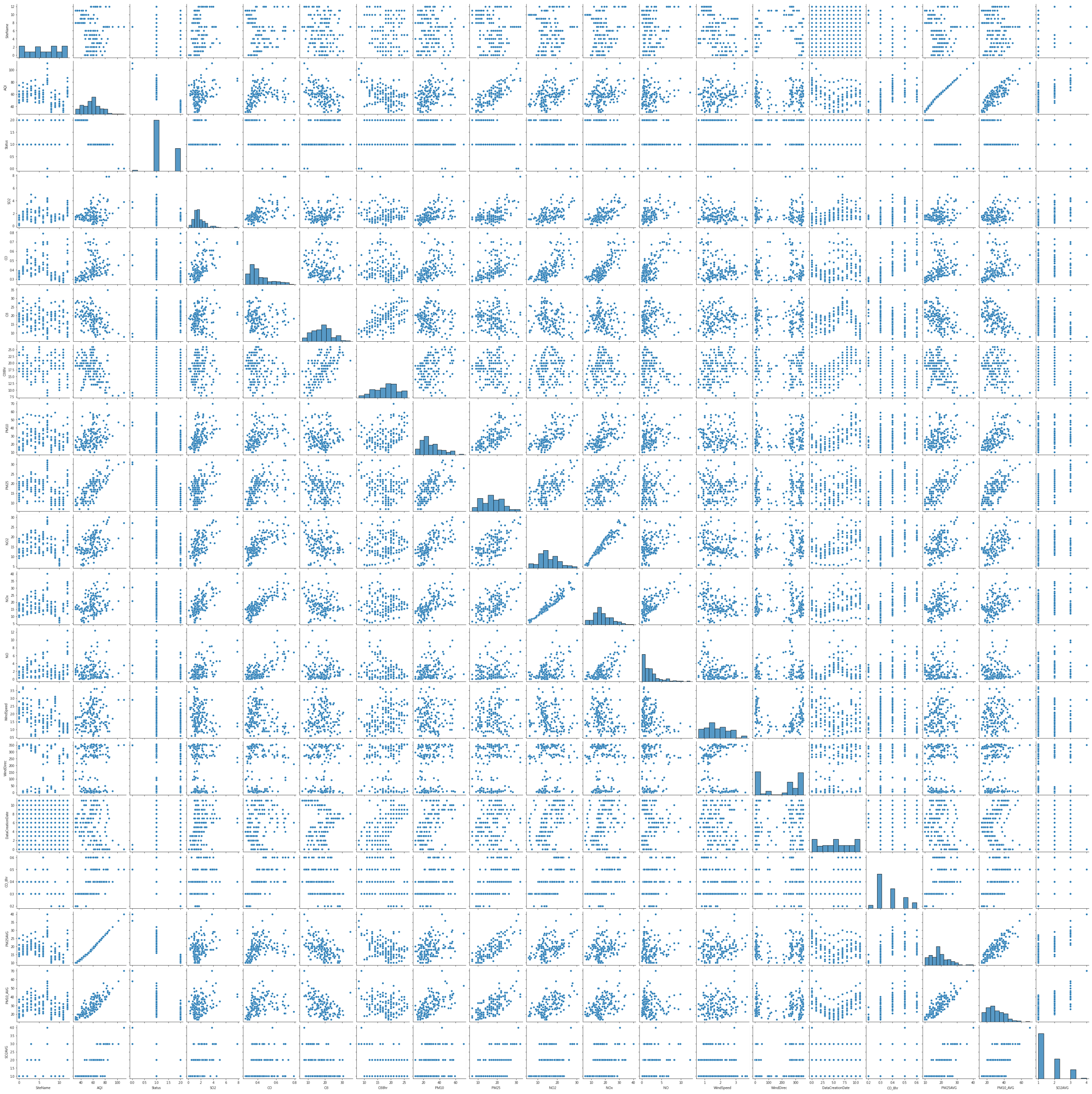
Out[11]:

| | SiteName | AQI | Status | SO2 | CO | O3 | O38hr | PM10 | PM25 | NO2 | NOx | NO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 155.000000 | 155.000000 | 155.000000 | 155.000000 | 155.000000 | 155.000000 | 155.000000 | 155.000000 | 155.000000 | 155.000000 | 155.000000 | 155.000000 |
| **mean** | 6.006452 | 58.264516 | 1.290323 | 1.826452 | 0.416387 | 18.198710 | 17.703226 | 28.851613 | 17.496774 | 15.769032 | 17.943226 | 2.135484 |
| **std** | 3.765008 | 14.529896 | 0.483060 | 1.160501 | 0.113085 | 5.548909 | 4.268845 | 12.041483 | 5.372337 | 5.137418 | 6.389714 | 2.206270 |
| **min** | 0.000000 | 31.000000 | 0.000000 | 0.100000 | 0.270000 | 6.600000 | 8.000000 | 10.000000 | 7.000000 | 5.400000 | 5.800000 | 0.000000 |
| **25%** | 3.000000 | 47.500000 | 1.000000 | 1.100000 | 0.335000 | 13.850000 | 14.500000 | 20.000000 | 13.000000 | 12.300000 | 14.100000 | 0.500000 |
| **50%** | 6.000000 | 58.000000 | 1.000000 | 1.500000 | 0.380000 | 18.300000 | 18.000000 | 26.000000 | 18.000000 | 14.600000 | 16.800000 | 1.300000 |
| **75%** | 9.000000 | 66.000000 | 2.000000 | 2.250000 | 0.465000 | 21.500000 | 21.000000 | 35.000000 | 21.000000 | 19.250000 | 21.900000 | 3.050000 |
| **max** | 12.000000 | 111.000000 | 2.000000 | 7.800000 | 0.790000 | 34.800000 | 26.000000 | 70.000000 | 32.000000 | 30.100000 | 40.100000 | 12.400000 |

# Pairplot

In [12]:

```
sns.pairplot(df_le);
```



## Heatmap

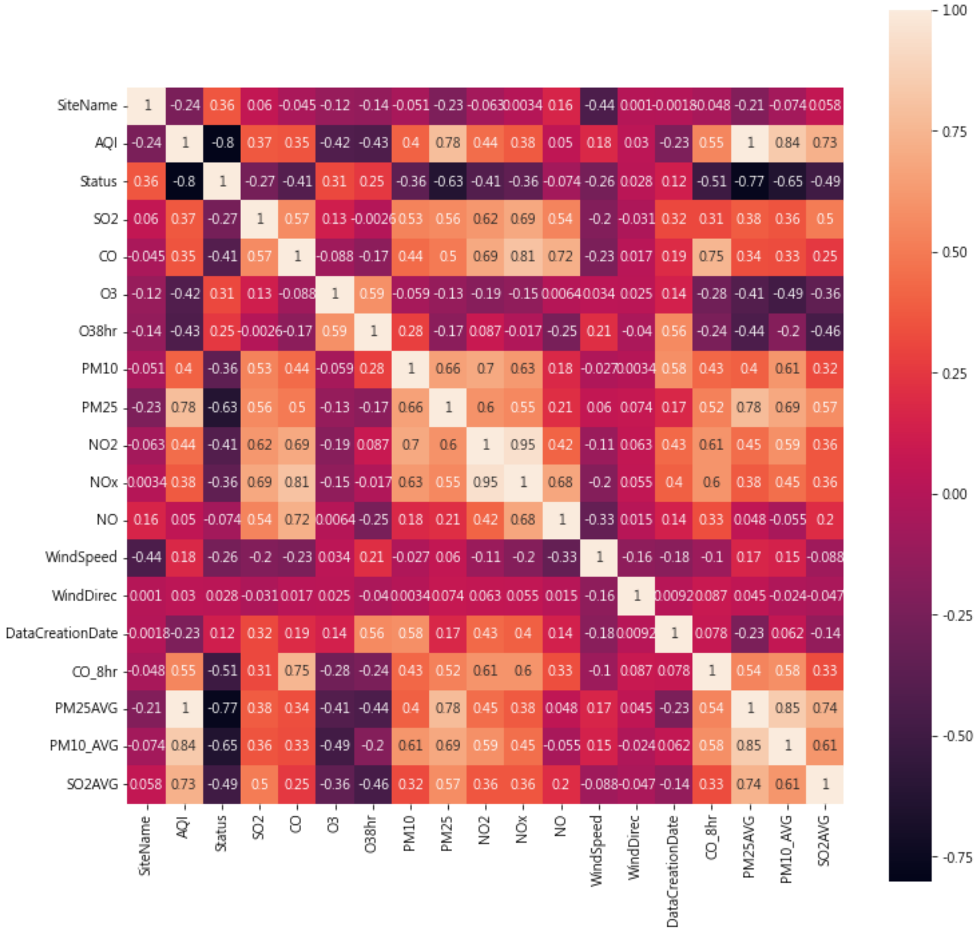
判斷相關係數

In [13]:

```
plt.figure(figsize = (12,12))
corr = df_le.corr()
sns.heatmap(corr,square = True, annot = True)
```

Out[13]:

<AxesSubplot:>



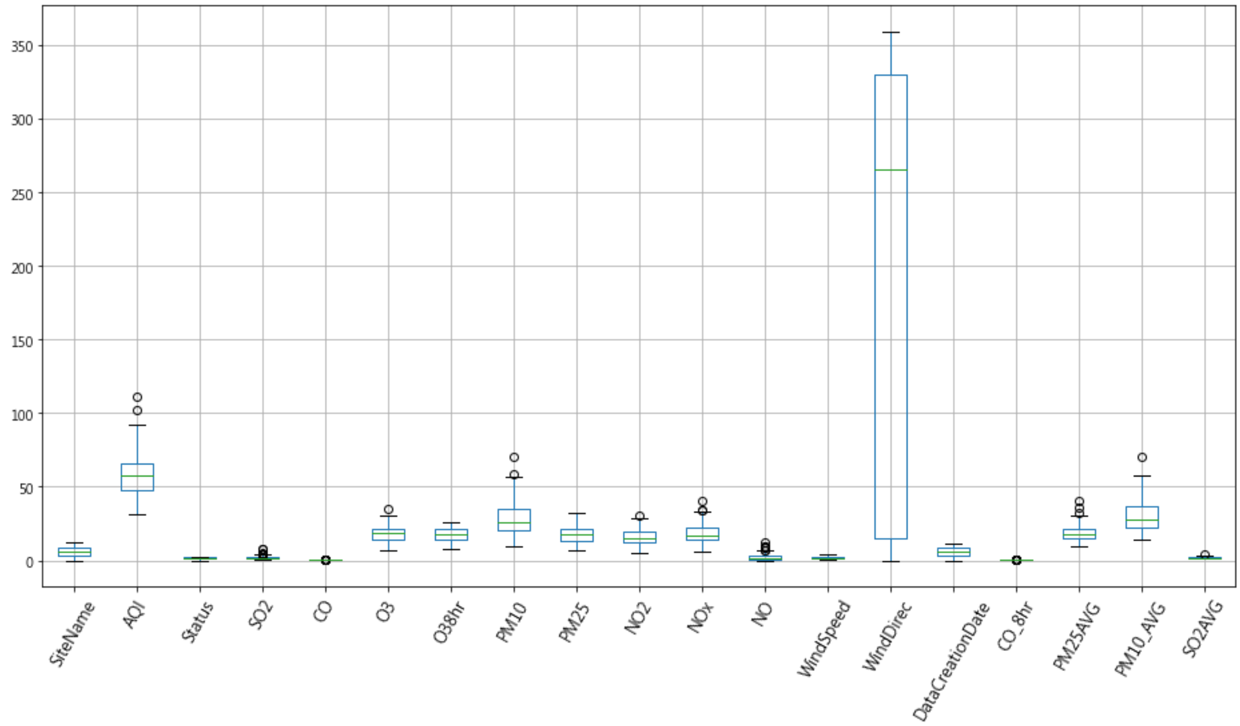
|相關係數| ≥ 0.5
X = AQI PM25 CO_8hr PM25AVG PM10_AVG SO2AVG
y = Status

# Boxplot

判斷outliers

In [14]:

```
df_le.boxplot(figsize=(16,8))
plt.xticks(rotation=60, fontsize=12);
```




In [15]:

```
X_train_drop, X_test_drop, y_train_drop, y_test_drop = train_test_split(df_le.drop("Status",axis= 1), \
                                                                        df_le["Status"], test_size = 0.2, random_state = 4)
X_train_drop.shape, y_train_drop.shape
```

Out[15]:

```
((124, 18), (124,))
```

In [16]:

```
X_test_drop.shape, y_test_drop.shape
```

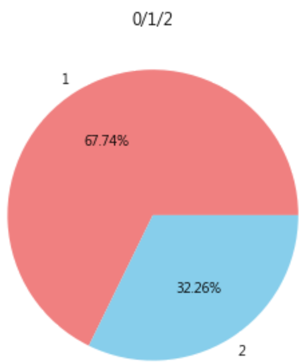
Out[16]:

```
((31, 18), (31,))
```

## Observation information

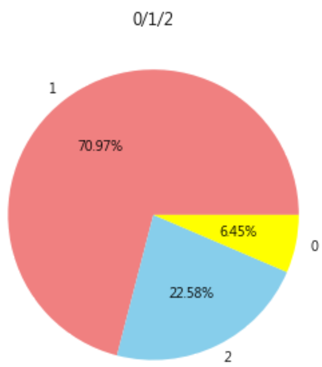
In [17]:

```
plt.figure( figsize=(10,5) )
y_train_drop.value_counts().plot(kind='pie', colors=['lightcoral',

plt.title( '0/1/2' )  # 圖標題
plt.ylabel( '' )
plt.show()
```




In [18]:

```
plt.figure( figsize=(10,5) )
y_test_drop.value_counts().plot(kind='pie', colors=['lightcoral','

plt.title( '0/1/2' )  # 圖標題
plt.ylabel( '' )
plt.show()
```



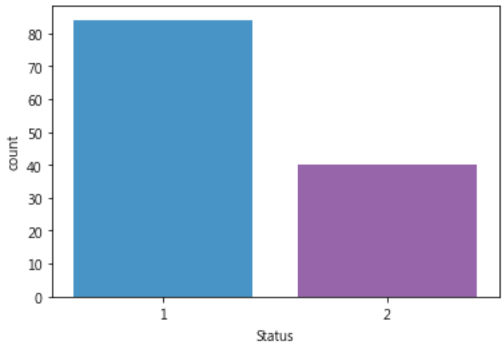

## Model

In [19]:

```
sns.countplot(y_train_drop, palette=["#3498db", "#9b59b6"])
```

Out[19]:

```
<AxesSubplot:xlabel='Status', ylabel='count'>
```

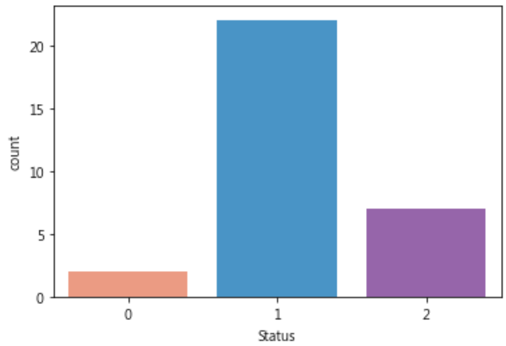



In [20]:

```
ns.countplot(y_test_drop, palette=["#fc9272", "#3498db", "#9b59b6"])
```

Out[20]:

```
<AxesSubplot:xlabel='Status', ylabel='count'>
```




機器學習可以分成「監督」與「非監督」式學習。兩者的差異在於所收集到的資料是否有被標籤（Labeled）。換言之，資料是否有被定義。

# Supervised learning

## Linear Classification

#### LDA

使用區分 AQI, PM25, CO_8hr, PM25AVG, PM10_AVG, SO2AVG 的資料

=> LDA希望投影後的資料，組內分散量(within-class scatter)越小越好，組間分散量(between-class scatter)越大越好。

In [21]:

```
LDA_model = LDA()
numerical_features = ["AQI", "PM25", "CO_8hr", "PM25AVG", "PM10_AVG", "SO2AVG"]

LDA_model.fit(X_train_drop[numerical_features],y_train_drop)
```

Out[21]:

```
LinearDiscriminantAnalysis()
```

In [22]:

```
predicted = LDA_model.predict(X_train_drop[numerical_features])
TA_LDA = LDA_model.score(X_train_drop[numerical_features],y_train_drop)

print('Train Accuracy of LDA: ', TA_LDA)
print('Train Error of LDA:    ', 1 - TA_LDA)
```

```
Train Accuracy of LDA:  0.967741935483871
Train Error of LDA:     0.032258064516129004
```

In [23]:

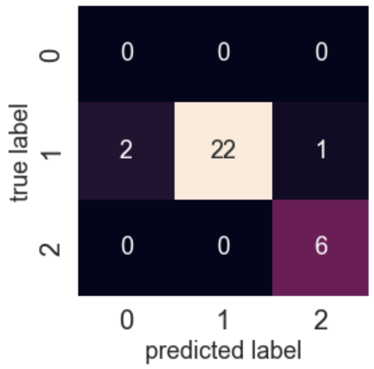
```
y_pred = LDA_model.predict(X_test_drop[numerical_features])
clf_LDA = accuracy_score(y_test_drop, y_pred)

print("Test Accuracy of LDA: ",clf_LDA)
print("Test Error of LDA:    ", 1 - clf_LDA)
```

```
Test Accuracy of LDA:  0.9032258064516129
Test Error of LDA:     0.09677419354838712
```

In [24]:

```
mat_LDA = confusion_matrix(y_test_drop, y_pred)

sns.set(font_scale = 1.5)
sns.heatmap(mat_LDA.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xticks(fontsize = 20)
plt.yticks(fontsize = 20)
plt.ylabel('true label')
plt.xlabel('predicted label')
target_names = ['0','1','2']
print(classification_report(y_test_drop, y_pred, target_names = target_names))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.88      1.00      0.94        22
           2       1.00      0.86      0.92         7

    accuracy                           0.90        31
   macro avg       0.63      0.62      0.62        31
weighted avg       0.85      0.90      0.87        31
```




### Nonlinear Classification

#### DecisionTree

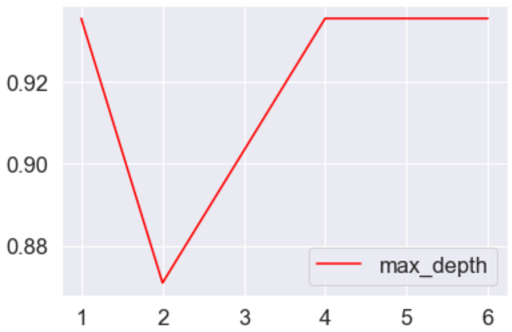
使用區分 AQI, PM25, CO_8hr, PM25AVG, PM10_AVG, SO2AVG 的資料

=>可以同時處理連續型與類別型變數，不需要進行太多的資料預處理（Preprocessing）。

In [25]:

```
depth_list = list(range(2,7))
depth_tuning = np.zeros((len(depth_list), 2))
depth_tuning[:,0] = depth_list
y_eff = []
col_names = ['Max_Depth','Accuracy']
for i in range(6):    # 測試的條件數
    tree_clf = tree.DecisionTreeClassifier(criterion="entropy"
                                          ,random_state = 4
                                          ,splitter = "random"
                                          ,max_depth = i+1    #測試條件
                                          )
    tree_clf = tree_clf.fit(X_train_drop[numerical_features], y_train_drop)
    score = tree_clf.score(X_test_drop[numerical_features], y_test_drop)
    y_eff.append(score)
    depth_tuning[i-1,1] = score

print(pd.DataFrame(depth_tuning, columns=col_names))
plt.plot(range(1,7),y_eff,color="red",label="max_depth")
plt.legend()
plt.show()
```

```
   Max_Depth  Accuracy
0        2.0  0.870968
1        3.0  0.903226
2        4.0  0.935484
3        5.0  0.935484
4        6.0  0.935484
```




In [26]:

```
numerical_features = ["AQI", "PM25", "CO_8hr", "PM25AVG", "PM10_AVG", "SO2AVG"]

decisionTreeModel = DecisionTreeClassifier(criterion="entropy", max_depth=4, random_state=4)
decisionTreeModel.fit(X_train_drop[numerical_features], y_train_drop.values.astype(int))
```

Out[26]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=4)
```

In [27]:

```
predicted = decisionTreeModel.predict(X_train_drop[numerical_features])
TA_DT = decisionTreeModel.score(X_train_drop[numerical_features],y_train_drop)

print('Train Accuracy of LDA: ', TA_DT)
print('Train Error of LDA:    ', 1 - TA_DT)
```

```
Train Accuracy of LDA:  1.0
Train Error of LDA:     0.0
```

In [28]:

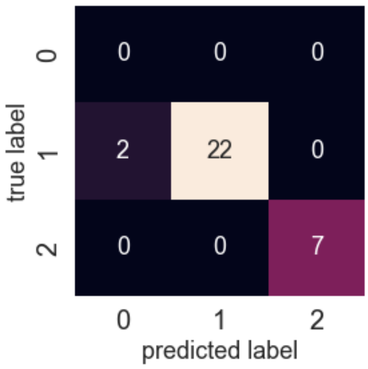
```
y_pred = decisionTreeModel.predict(X_test_drop[numerical_features])
clf_DT = accuracy_score(y_test_drop, y_pred)

print("Test Accuracy of LDA: ",clf_DT)
print("Test Error of LDA:    ", 1 - clf_DT)
```

```
Test Accuracy of LDA:  0.9354838709677419
Test Error of LDA:     0.06451612903225812
```

In [29]:

```
mat_LDA = confusion_matrix(y_test_drop, y_pred)

sns.set(font_scale = 1.5)
sns.heatmap(mat_LDA.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xticks(fontsize = 20)
plt.yticks(fontsize = 20)
plt.ylabel('true label')
plt.xlabel('predicted label')
target_names = ['0','1','2']
print(classification_report(y_test_drop, y_pred, target_names = target_names))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.92      1.00      0.96        22
           2       1.00      1.00      1.00         7

    accuracy                           0.94        31
   macro avg       0.64      0.67      0.65        31
weighted avg       0.88      0.94      0.90        31
```




**Xgboost**

使用區分 AQI, PM25, CO_8hr, PM25AVG, PM10_AVG, SO2AVG 的資料

=> XGBoost 除了可以做分類也能進行迴歸連續性數值的預測，而且效果通常都不差。並透過 Boosting 技巧將許多弱決策樹集成在一起形成一個強的預測模型。

利用了二階梯度來對節點進行劃分 利用局部近似算法對分裂節點進行優化 在損失函數中加入了 L1/L2 項，控制模型的複雜度 提供 GPU 平行化運算

In [30]:

```
# df_test=pd.DataFrame(X_test_drop[numerical_features])
# df_test['Status'] = y_test_drop

# pred = xgboostModel.predict(X_test_drop[numerical_features])
# df_test['Predict'] = pred
```

In [31]:

```
# sns.lmplot("AQI", "PM25", hue='Status', data = df_test, fit_reg=False, legend=False)
# plt.legend(title='target', loc='upper left', labels=['0', '1', '2'])
# plt.show()
```

In [32]:

```
# sns.lmplot("AQI", "PM25", hue="Predict", data = df_test, fit_reg=False, legend=False)
# plt.legend(title='target', loc='upper left', labels=['0', '1', '2'])
# plt.show()
```

In [33]:

```
params = { 'max_depth': [3,6,10],
           'learning_rate': [0.01, 0.05, 0.1],
           'n_estimators': [100, 500, 1000],
           'colsample_bytree': [0.3, 0.7]}

xg2 = XGBClassifier(random_state=4)
clf = GridSearchCV(estimator=xg2,
                   param_grid=params,
                   scoring='neg_mean_squared_error',
                   verbose=1)
clf.fit(X_train_drop[numerical_features],y_train_drop)

print("Best parameters:", clf.best_params_)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Best parameters: {'colsample_bytree': 0.3, 'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}
```

In [34]:

```
xgboostModel = XGBClassifier(criterion="entropy", colsample_bytree=0.3, learning_rate=0.01, max_depth=3, n_estimators=100)
numerical_features = ["AQI", "PM25", "CO_8hr", "PM25AVG", "PM10_AVG", "SO2AVG"]

xgboostModel.fit(X_train_drop[numerical_features], y_train_drop)
```

Out[34]:

```
XGBClassifier(colsample_bytree=0.3, criterion='entropy', learning_rate=0.01)
```

In [35]:

```
predicted = xgboostModel.predict(X_train_drop[numerical_features])
TA_XGB = xgboostModel.score(X_train_drop[numerical_features],y_train_drop)

print('Train Accuracy of xgboost: ', TA_XGB)
print('Train Error of xgboost::   ', 1 - TA_XGB)
```

```
Train Accuracy of xgboost:  1.0
Train Error of xgboost::    0.0
```

In [36]:

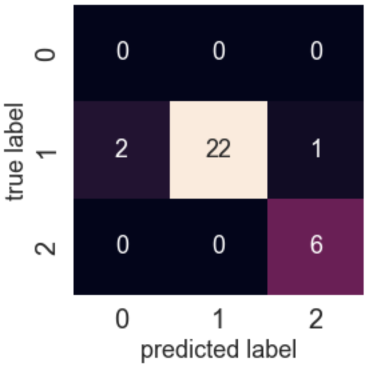
```
clf_XGB = xgboostModel.score(X_test_drop[numerical_features],y_test_drop)

print("Test Accuracy of xgboost: ",clf_XGB)
print("Test Error of xgboost:    ", 1 - clf_XGB)
```

```
Test Accuracy of xgboost:  0.9032258064516129
Test Error of xgboost:     0.09677419354838712
```

In [37]:

```
y_pred = xgboostModel.predict(X_test_drop[numerical_features])
mat_LDA = confusion_matrix(y_test_drop.values, y_pred)

sns.set(font_scale = 1.5)
sns.heatmap(mat_LDA.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xticks(fontsize = 20)
plt.yticks(fontsize = 20)
plt.ylabel('true label')
plt.xlabel('predicted label')
target_names = ['0','1','2']
print(classification_report(y_test_drop.values, y_pred, target_names = target_names))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.88      1.00      0.94        22
           2       1.00      0.86      0.92         7

    accuracy                           0.90        31
   macro avg       0.63      0.62      0.62        31
weighted avg       0.85      0.90      0.87        31
```



### Comparison

In [38]:

```
dict = {'Model Training' : ['LDA', 'DecisionTree', 'Xgboost'],
        'Accuracy' : [TA_LDA, TA_DT, TA_XGB],
        'Error' : [1 - TA_LDA, 1 - TA_DT, 1 - TA_XGB]}

dataframe = pd.DataFrame(dict)
dataframe.style

print(tabulate(dataframe, headers = 'keys', tablefmt = 'pretty'))
```

```
+---+----------------+--------------------+----------------------+
|   | Model Training |      Accuracy      |        Error         |
+---+----------------+--------------------+----------------------+
| 0 |      LDA       | 0.967741935483871  | 0.032258064516129004 |
| 1 |  DecisionTree  |        1.0         |         0.0          |
| 2 |    Xgboost     |        1.0         |         0.0          |
+---+----------------+--------------------+----------------------+
```

In [39]:

```
dict = {'Model Testing' : ['LDA', 'DecisionTree', 'Xgboost'],
        'Accuracy' : [clf_LDA, clf_DT, clf_XGB],
        'Error' : [1 - clf_LDA, 1 - clf_DT, 1 - clf_XGB]}

dataframe = pd.DataFrame(dict)
dataframe.style

print(tabulate(dataframe, headers = 'keys', tablefmt = 'pretty'))
```

```
+---+---------------+--------------------+---------------------+
|   | Model Testing |      Accuracy      |        Error        |
+---+---------------+--------------------+---------------------+
| 0 |      LDA      | 0.9032258064516129 | 0.09677419354838712 |
| 1 | DecisionTree  | 0.9354838709677419 | 0.06451612903225812 |
| 2 |    Xgboost    | 0.9032258064516129 | 0.09677419354838712 |
+---+---------------+--------------------+---------------------+
```

Type *Markdown* and LaTeX: $\alpha^2$

## Unsupervised learning

### Clustering

#### KMeans 演算法

使用區分 AQI, PM25, CO_8hr, PM25AVG, PM10_AVG, SO2AVG 的資料

=> K-Means 演算法可以非常快速地完成分群任務，但是如果觀測值具有雜訊（Noise）或者極端值，其分群結果容易被這些雜訊與極端值影響，適合處理分布集中的大型樣本資料。

In [40]:

```
numerical_features = ["AQI", "PM25", "CO_8hr", "PM25AVG", "PM10_AVG", "SO2AVG"]
df_le_X = df_le.drop("Status",axis= 1)[numerical_features].values

# KMeans 演算法
kmeans_fit = cluster.KMeans(n_clusters = 3, random_state = 4).fit(df_le_X)

# 印出分群結果
cluster_labels = kmeans_fit.labels_
print("分群結果：")
print(cluster_labels)
print("---")
```

```
分群結果：
[1 2 2 2 0 0 0 2 2 0 0 0 0 1 2 2 0 2 0 1 2 2 2 2 2 1 1 2 2 2 2 0 1 2 2 2 2
 1 1 1 1 1 2 2 0 1 2 2 2 2 2 1 1 1 2 2 0 1 1 1 1 1 1 1 1 1 1 2 2 0 1 1 1 1
 1 1 1 1 1 1 2 2 0 1 1 1 1 1 1 1 1 1 1 2 2 0 1 2 1 2 2 1 1 1 1 1 1 2 2 0 1
 2 2 2 2 2 1 1 2 2 2 2 0 1 2 2 2 2 2 2 1 2 2 2 2 0 1 2 2 2 2 2 2 1 1 2 2 2 2 2 0
 1 2 2 2 2 2 1]
---
```

In [41]:

```
col_y = ['Status']
df_le_y = df_le[col_y]
print("真實狀態：")
print(df_le_y)
```

```
真實狀態：
         Status
3506558       2
3506602       1
3506603       1
3506604       1
3506605       1
...         ...
3508449       1
3508450       1
3508451       1
3508464       1
3508473       2

[155 rows x 1 columns]
```

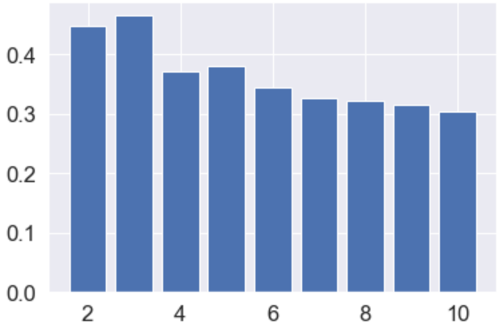
In [42]:

```
# 印出績效，我們使用 sklearn.metrics 的 silhouette_score() 方法，這個數值愈接近1 表示績效愈好，反之愈接近-1 表示績效愈差
silhouette_avg = metrics.silhouette_score(df_le_X, cluster_labels)
print('silhouette_avg: ',silhouette_avg)
```

```
silhouette_avg:  0.46526360052382965
```

In [43]:

```
silhouette_avgs = []
ks = range(2, 11)
for k in ks:
    kmeans_fit = cluster.KMeans(n_clusters = k).fit(df_le_X)
    cluster_labels = kmeans_fit.labels_
    silhouette_avg = metrics.silhouette_score(df_le_X, cluster_labels)
    silhouette_avgs.append(silhouette_avg)

# 作圖並印出 k = 2 到 10
plt.bar(ks, silhouette_avgs)
plt.show()
print(silhouette_avgs)
```




```
[0.4475491003555782, 0.46526360052382965, 0.3711313328717566, 0.3802094019099495, 0.344898736777075 83, 0.32731707562400353, 0.3211367137632005, 0.3153241197902054, 0.30449617592363 49]
```

In [44]:

```
X = df_le.drop(labels=['Status'],axis=1)[numerical_features]
y = df_le['Status']

X_train_drop, X_test_drop, y_train_drop, y_test_drop = train_test_split(X, \
                                                                        y, test_size = 0.2,\
                                                                        random_state = 4)

kmeansModel = KMeans(n_clusters=3, random_state = 4)
clusters_pred = kmeansModel.fit_predict(X_train_drop)
y_pred = kmeansModel.fit_predict(X_test_drop)
```

In [45]:

```
X_train_drop.shape, y_train_drop.shape
```

Out[45]:

```
((124, 6), (124,))
```

In [46]:

```
df_le2 = pd.DataFrame(X_train_drop)
df_le3 = df_le2.assign(Status=y_train_drop)
df_le3
```

Out[46]:

| | AQI | PM25 | CO_8hr | PM25AVG | PM10_AVG | SO2AVG | Status |
|---|---|---|---|---|---|---|---|
| **3507302** | 31.0 | 12.0 | 0.3 | 10.0 | 15.0 | 1.0 | 2 |
| **3507780** | 41.0 | 15.0 | 0.3 | 13.0 | 22.0 | 1.0 | 2 |
| **3508282** | 70.0 | 21.0 | 0.4 | 23.0 | 42.0 | 2.0 | 1 |
| **3506783** | 81.0 | 20.0 | 0.5 | 28.0 | 37.0 | 3.0 | 1 |
| **3506787** | 65.0 | 13.0 | 0.4 | 21.0 | 34.0 | 2.0 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **3507126** | 60.0 | 19.0 | 0.4 | 19.0 | 26.0 | 1.0 | 1 |
| **3507632** | 53.0 | 19.0 | 0.4 | 16.0 | 25.0 | 1.0 | 1 |
| **3507944** | 43.0 | 16.0 | 0.3 | 13.0 | 21.0 | 1.0 | 2 |
| **3508278** | 45.0 | 16.0 | 0.3 | 14.0 | 26.0 | 1.0 | 2 |
| **3508117** | 84.0 | 29.0 | 0.5 | 29.0 | 47.0 | 3.0 | 1 |

124 rows × 7 columns

In [47]:

```
df_le4 = pd.DataFrame(X_test_drop)
df_le5 = df_le4.assign(Status=y_test_drop)
df_le5
```

Out[47]:

| | AQI | PM25 | CO_8hr | PM25AVG | PM10_AVG | SO2AVG | Status |
|---|---|---|---|---|---|---|---|
| **3508120** | 60.0 | 21.0 | 0.4 | 19.0 | 29.0 | 1.0 | 1 |
| **3507786** | 56.0 | 20.0 | 0.3 | 18.0 | 24.0 | 2.0 | 1 |
| **3506954** | 65.0 | 22.0 | 0.4 | 21.0 | 32.0 | 2.0 | 1 |
| **3506811** | 42.0 | 11.0 | 0.3 | 13.0 | 24.0 | 1.0 | 2 |
| **3506786** | 55.0 | 11.0 | 0.3 | 17.0 | 25.0 | 1.0 | 1 |
| **3507454** | 80.0 | 24.0 | 0.3 | 27.0 | 37.0 | 3.0 | 1 |
| **3507945** | 46.0 | 15.0 | 0.3 | 14.0 | 25.0 | 1.0 | 2 |
| **3506625** | 76.0 | 21.0 | 0.6 | 26.0 | 42.0 | 1.0 | 1 |
| **3507109** | 40.0 | 10.0 | 0.3 | 12.0 | 19.0 | 1.0 | 2 |
| **3507781** | 50.0 | 18.0 | 0.3 | 15.0 | 18.0 | 1.0 | 2 |
| **3506802** | 67.0 | 17.0 | 0.5 | 22.0 | 35.0 | 1.0 | 1 |
| **3508115** | 65.0 | 27.0 | 0.6 | 21.0 | 35.0 | 2.0 | 1 |
| **3506781** | 60.0 | 11.0 | 0.3 | 19.0 | 33.0 | 1.0 | 1 |
| **3508284** | 46.0 | 13.0 | 0.3 | 14.0 | 36.0 | 1.0 | 2 |
| **3507279** | 55.0 | 10.0 | 0.4 | 17.0 | 25.0 | 2.0 | 1 |
| **3507105** | 58.0 | 12.0 | 0.3 | 18.0 | 21.0 | 1.0 | 1 |
| **3506785** | 102.0 | 30.0 | 0.5 | 36.0 | 58.0 | 3.0 | 0 |
| **3506606** | 82.0 | 21.0 | 0.4 | 28.0 | 50.0 | 3.0 | 1 |
| **3507110** | 61.0 | 17.0 | 0.3 | 19.0 | 25.0 | 2.0 | 1 |
| **3506612** | 84.0 | 22.0 | 0.5 | 29.0 | 47.0 | 2.0 | 1 |
| **3506945** | 44.0 | 9.0 | 0.3 | 14.0 | 23.0 | 1.0 | 2 |
| **3508113** | 55.0 | 21.0 | 0.3 | 17.0 | 33.0 | 1.0 | 1 |
| **3506607** | 111.0 | 31.0 | 0.5 | 40.0 | 70.0 | 4.0 | 0 |
| **3506603** | 67.0 | 16.0 | 0.4 | 22.0 | 40.0 | 1.0 | 1 |
| **3508135** | 56.0 | 17.0 | 0.6 | 18.0 | 33.0 | 1.0 | 1 |
| **3508448** | 62.0 | 16.0 | 0.4 | 20.0 | 39.0 | 2.0 | 1 |
| **3507111** | 62.0 | 20.0 | 0.3 | 20.0 | 23.0 | 1.0 | 1 |
| **3507950** | 36.0 | 18.0 | 0.3 | 11.0 | 20.0 | 1.0 | 2 |
| **3506948** | 72.0 | 19.0 | 0.4 | 24.0 | 32.0 | 3.0 | 1 |
| **3508464** | 63.0 | 20.0 | 0.5 | 20.0 | 41.0 | 1.0 | 1 |
| **3507625** | 57.0 | 14.0 | 0.5 | 18.0 | 27.0 | 2.0 | 1 |

In [48]:

```
kmeansModel.labels_
```

Out[48]:

```
array([0, 0, 2, 0, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0, 1, 2, 0, 2, 0, 0,
       1, 2, 0, 2, 0, 0, 2, 2, 0])
```

In [49]:

```
kmeansModel.inertia_
```

Out[49]:

```
3283.0484343434346
```

In [50]:

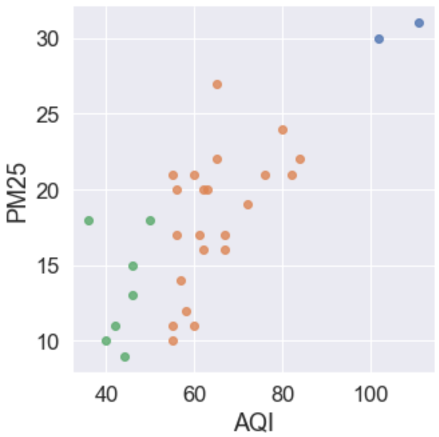
```
kmeansModel.cluster_centers_
```

Out[50]:

```
array([[ 52.16666667,  14.88888889,   0.33888889,  16.27777778,
         25.72222222,   1.22222222],
       [106.5       ,  30.5       ,   0.5       ,  38.        ,
         64.        ,   3.5       ],
       [ 71.18181818,  20.45454545,   0.45454545,  23.63636364,
         39.09090909,   1.90909091]])
```

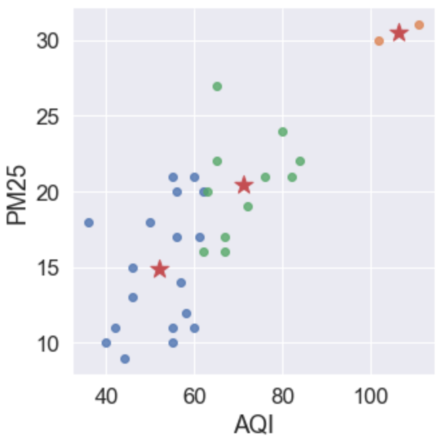
In [51]:

```
sns.lmplot("AQI", "PM25", hue='Status', data=df_le5, fit_reg=False, legend=False)
plt.show()
```




In [52]:

```
df_le5['Predict']=y_pred
sns.lmplot("AQI", "PM25", hue="Predict", data=df_le5 , fit_reg=False, legend=False)
plt.scatter(kmeansModel.cluster_centers_[:, 0], kmeansModel.cluster_centers_[:, 1], s=200,c="r",marker='*')
plt.show()
```




### Dimension Reduction

#### PCA vs LDA

使用區分 AQI, PM25, CO_8hr, PM25AVG, PM10_AVG, SO2AVG 的資料

=> PCA:
為一種統計分析、簡化數據集的方法，利用正交變換來對一系列可能相關的變數的觀測值進行線性變換，從而投影為一系列線性不相關變數的值，這些不相關變數稱為主成分（Principal Components）。
將座標軸中心移至數據集的中心，利用旋轉座標軸，使數據在C1軸的變異數最大，以保留更多信息，C1即為第一主成分。 找一個與C1主成分的共變異數為0的C2主成分，以避免信息重疊。
主成分分析經常用於減少數據集的維數，同時保留數據集當中對變異數貢獻最大的特徵。

優點：
以變異數為衡量信息量的指標，不受數據集以外的因素影響。 用正交轉換方式，可消除數據成分間相互影響的因素。

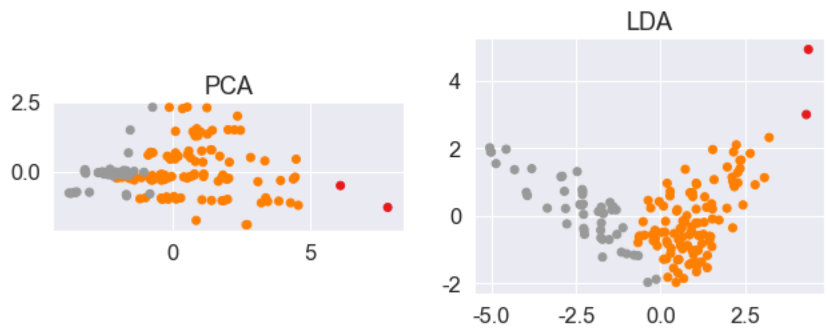
缺點：
主成分間的特徵維度較難解釋。 容易捨棄一些也帶有信息量的特徵，分析結果可能會受影響。

In [53]:

```
col_X = ['AQI', 'PM25', 'CO_8hr', 'PM25AVG', 'PM10_AVG', 'SO2AVG']
df_le_X = df_le[col_X].astype(float)
df_le_X = df_le_X.values

col_y = ['Status']
df_le_y = df_le[col_y].astype(float)
df_le_y = df_le_y.values

X_scale = StandardScaler().fit_transform(df_le_X)
```

In [54]:

```
pca = PCA(n_components=2)
lda = LDA(n_components=2)

plt.figure(figsize=(16,8))

plt.subplot(1, 3, 1, aspect=1)
pca.fit(X_scale, df_le_y)
X_embedded = pca.transform(X_scale)
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=df_le_y, cmap='Set1')
plt.title(f'PCA')

plt.subplot(1, 3, 2, aspect=1)
lda.fit(X_scale, df_le_y)
X_embedded = lda.transform(X_scale)
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=df_le_y, cmap='Set1')
plt.title(f'LDA')
plt.show()
```




**PCA後對LDA、Decision Tree、Xgboost做分類並畫 decision regions**

In [55]:

```
# Initializing Classifiers
clf1 = LDA()
clf2 = DecisionTreeClassifier(max_depth=4, random_state=4)
clf3 = XGBClassifier(colsample_bytree=0.3, learning_rate=0.01, max_depth=3, n_estimators=100)
```

In [56]:

```
X_train_drop[numerical_features]
```

Out[56]:

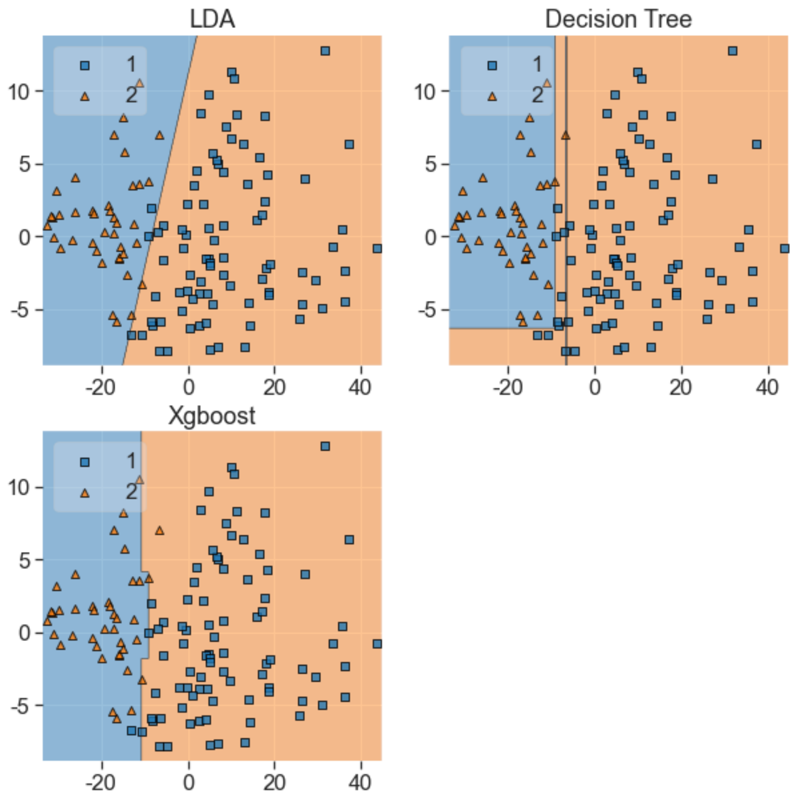
| | AQI | PM25 | CO_8hr | PM25AVG | PM10_AVG | SO2AVG |
|---|---|---|---|---|---|---|
| **3507302** | 31.0 | 12.0 | 0.3 | 10.0 | 15.0 | 1.0 |
| **3507780** | 41.0 | 15.0 | 0.3 | 13.0 | 22.0 | 1.0 |
| **3508282** | 70.0 | 21.0 | 0.4 | 23.0 | 42.0 | 2.0 |
| **3506783** | 81.0 | 20.0 | 0.5 | 28.0 | 37.0 | 3.0 |
| **3506787** | 65.0 | 13.0 | 0.4 | 21.0 | 34.0 | 2.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **3507126** | 60.0 | 19.0 | 0.4 | 19.0 | 26.0 | 1.0 |
| **3507632** | 53.0 | 19.0 | 0.4 | 16.0 | 25.0 | 1.0 |
| **3507944** | 43.0 | 16.0 | 0.3 | 13.0 | 21.0 | 1.0 |
| **3508278** | 45.0 | 16.0 | 0.3 | 14.0 | 26.0 | 1.0 |
| **3508117** | 84.0 | 29.0 | 0.5 | 29.0 | 47.0 | 3.0 |

124 rows × 6 columns

In [57]:

```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
import matplotlib.gridspec as gridspec
import itertools

pca = PCA(n_components=2, iterated_power=1)
train_reduced = pca.fit_transform(X_train_drop[numerical_features])
test_reduced = pca.transform(X_test_drop[numerical_features])

gs = gridspec.GridSpec(2, 2)

fig = plt.figure(figsize=(10,10))

labels = ['LDA', 'Decision Tree', 'Xgboost']
for clf, lab, grd in zip([clf1, clf2, clf3],
                         labels,
                         itertools.product([0, 1], repeat=2)):

    clf.fit(train_reduced, y_train_drop)
    predicted = clf.predict(train_reduced)
    TA_clf = clf.score(train_reduced,y_train_drop)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=train_reduced, y=y_train_drop.values, clf=clf, legend=2)
    plt.title(lab)
    print(clf,'Train Accuracy:', TA_clf)

plt.show()
```

```
LinearDiscriminantAnalysis() Train Accuracy: 0.9758064516129032
DecisionTreeClassifier(max_depth=4, random_state=4) Train Accuracy: 1.0
XGBClassifier(colsample_bytree=0.3, learning_rate=0.01) Train Accuracy: 0.9838709677419355
```



In [58]:

```
gs = gridspec.GridSpec(2, 2)

fig = plt.figure(figsize=(10,8))

labels = ['LDA', 'Decision Tree', 'Xgboost']
for clf, lab, grd in zip([clf1, clf2, clf3],
                         labels,
                         itertools.product([0, 1], repeat=2)):
    clf.fit(test_reduced, y_test_drop)
    predicted = clf.predict(test_reduced)
    TeA_clf = clf.score(test_reduced,y_test_drop)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=test_reduced, y=y_test_drop.values, clf=clf, legend=2)
    plt.title(lab)
    print(clf,'Test Accuracy:', TeA_clf)

plt.show()
```

```
LinearDiscriminantAnalysis() Test Accuracy: 0.967741935483871
DecisionTreeClassifier(max_depth=4, random_state=4) Test Accuracy: 1.0
XGBClassifier(colsample_bytree=0.3, learning_rate=0.01,
              objective='multi:softprob') Test Accuracy: 0.967741935483871
```