HW3 Time Series Regression

0813458 資財 簡辰穎

一. Import Library

Import Library

```
In [1]: import pandas as pd
        import numpy as np
        import datetime
        import matplotlib.pyplot as plt
        #model
        from sklearn.linear_model import LinearRegression
        from sklearn import linear_model
        from sklearn.metrics import mean_squared_error, r2_score
        from sklearn.metrics import mean_absolute_error
        import xgboost
        from xgboost import XGBRegressor
```

二. Load Data

Load Data

```
In [2]: df = pd.read_excel('新竹_2020.xlsx')

In [3]: df.head()
```

Out[3]:

	測站	日期	測項	0	1	2	3	4	5	6	...	14	15	16	17	18	19	20	21	22	23
0	----------	----------	----	-------	-------	-------	-------	-------	-------	-------	...	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------
1	新竹	2020-01-01 00:00:00	AMB_TEMP	15.20	15.20	15.30	15.30	15.30	15.40	15.50	...	18.10	18.20	17.90	17.30	16.70	16.40	16.20	16.10	16	15.80
2	新竹	2020-01-01 00:00:00	CH4	1.74	1.74	1.77	1.78	1.77	1.77	1.77	...	1.78	1.78	1.77	1.80	1.81	1.82	1.85	1.83	1.92	1.94
3	新竹	2020-01-01 00:00:00	CO	0.28	0.25	0.24	0.22	0.20	0.19	0.20	...	0.28	0.29	0.28	0.34	0.39	0.41	0.46	0.49	0.58	0.52
4	新竹	2020-01-01 00:00:00	NMHC	0.06	0.07	0.05	0.05	0.05	0.05	0.07	...	0.09	0.09	0.07	0.08	0.12	0.12	0.16	0.14	0.17	0.20

5 rows × 27 columns

並將第一列中奇怪的---------符號刪除

```
In [4]: #將第一行奇怪的東西刪除
        df = df.drop([0])
```

```
In [5]: df.head()
```

Out[5]:

	測站	日期	測項	0	1	2	3	4	5	6	...	14	15	16	17	18	19	20	21	22	23
1	新竹	2020-01-01 00:00:00	AMB_TEMP	15.20	15.20	15.30	15.30	15.30	15.40	15.50	...	18.10	18.20	17.90	17.30	16.70	16.40	16.20	16.10	16	15.80
2	新竹	2020-01-01 00:00:00	CH4	1.74	1.74	1.77	1.78	1.77	1.77	1.77	...	1.78	1.78	1.77	1.80	1.81	1.82	1.85	1.83	1.92	1.94
3	新竹	2020-01-01 00:00:00	CO	0.28	0.25	0.24	0.22	0.20	0.19	0.20	...	0.28	0.29	0.28	0.34	0.39	0.41	0.46	0.49	0.58	0.52
4	新竹	2020-01-01 00:00:00	NMHC	0.06	0.07	0.05	0.05	0.05	0.05	0.07	...	0.09	0.09	0.07	0.08	0.12	0.12	0.16	0.14	0.17	0.20
5	新竹	2020-01-01 00:00:00	NO	0.30	0.60	0.60	0.60	0.30	0.30	0.50	...	1.60	1.60	1.20	0.70	0.90	1.10	1.10	1.70	1.80	1.40

5 rows × 27 columns

三. 資料前處理

1. **取出 10, 11, 12 月資料**

   ⇨ 原本日期顯示為 object 的形態，先將日期轉為 datetime 的形式

```
In [8]: #object轉成datetime形式
        df['日期          '] = pd.to_datetime(df['日期          '],format='%m/%d/%Y %H:%M:%S')

In [9]: df['日期          ']
```

Out[9]:

```
1       2020-01-01
2       2020-01-01
3       2020-01-01
4       2020-01-01
5       2020-01-01
           ...
6584    2020-12-31
6585    2020-12-31
6586    2020-12-31
6587    2020-12-31
6588    2020-12-31
Name: 日期          , Length: 6588, dtype: datetime64[ns]
```

⇨ 取出 10-12 月資料，並重設旁邊的 index 值

```
In [11]: #取出10, 11, 12月資料
         data = df[df['日期                ']>= '2020-10-01']

In [12]: data.head()
```

Out[12]:

	測站	日期	測項	0	1	2	3	4	5	6	...	14	15	16	17	18	19	20	21	22	23
4933	新竹	2020-10-01	AMB_TEMP	23.70	23.80	23.80	23.90	23.90	23.80	24.10	...	29.90	29.60	28.70	27.50	26.40	25.70	25.50	25.30	24.90	24.50
4934	新竹	2020-10-01	CH4	1.97	1.95	1.96	1.96	1.95	1.96	1.97	...	1.97	1.98	1.97	2	2.03	2.04	2.05	2.02	2.10	2.14
4935	新竹	2020-10-01	CO	0.23	0.22	0.21	0.20	0.20	0.22	0.24	...	0.29	0.30	0.33	0.38	0.46	0.50	0.45	0.39	0.46	0.45
4936	新竹	2020-10-01	NMHC	0.06	0.05	0.03	0.03	0.03	0.04	0.04	...	0.06	0.07	0.09	0.11	0.13	0.15	0.10	0.07	0.12	0.18
4937	新竹	2020-10-01	NO	1.20	0.70	0.50	0.70	0.50	0.30	0.70	...	1.30	1	0.90	0.80	0.50	0.90	0.90	0.30	0.70	0.90

5 rows × 27 columns

```
In [13]: #reset index
         data = data.reset_index(drop = True)
         data
```

Out[13]:

	測站	日期	測項	0	1	2	3	4	5	6	...	14	15	16	17	18	19	20	21	22	23
0	新竹	2020-10-01	AMB_TEMP	23.70	23.80	23.80	23.90	23.90	23.80	24.10	...	29.90	29.60	28.70	27.50	26.40	25.70	25.50	25.30	24.90	24.50
1	新竹	2020-10-01	CH4	1.97	1.95	1.96	1.96	1.95	1.96	1.97	...	1.97	1.98	1.97	2	2.03	2.04	2.05	2.02	2.10	2.14
2	新竹	2020-10-01	CO	0.23	0.22	0.21	0.20	0.20	0.22	0.24	...	0.29	0.30	0.33	0.38	0.46	0.50	0.45	0.39	0.46	0.45
3	新竹	2020-10-01	NMHC	0.06	0.05	0.03	0.03	0.03	0.04	0.04	...	0.06	0.07	0.09	0.11	0.13	0.15	0.10	0.07	0.12	0.18
4	新竹	2020-10-01	NO	1.20	0.70	0.50	0.70	0.50	0.30	0.70	...	1.30	1	0.90	0.80	0.50	0.90	0.90	0.30	0.70	0.90

2. 缺失值以及無效值以前後一小時平均值取代

⇨ 先判斷是否為數值

⇨ 如果不是數值，進行轉換，以前後一小時的平均來取代，如果後一小時沒有值就往後再取一小時

```
In [15]: #判斷是不是數值
def is_number(s):
    try: # 如果能运行float(s)语句，返回True(字符串s是浮点数)
        float(s)
        return True
    except ValueError: # ValueError为Python的一种标准异常，表示"传入无效的参数"
        pass
    try:
        import unicodedata # 处理ASCii码的包
        unicodedata.numeric(s) # 把一个表示数字的字符串转换为浮点数返回的函数
        return True
    except (TypeError, ValueError):
        pass
    return False
```

先將一開始的文字與數值分開，只進行數值的處理

```
In [16]: data_text = data.iloc[:, 0:3]
         #print(data_text)
         data_tmp = data.iloc[:,3:]
         #print(data_tmp)
         for i in range(1656):
             for j in range(24):
                 if(is_number(data_tmp.iloc[i, j]) == False): #如果並非數值
                     b = j
                     a = i
                     y = j
                     x = i
                     while(is_number(data_tmp.iloc[a,b]) == False):
                         b = (b+23)%24
                         if b == 23:
                             a = a-18
                     while(is_number(data_tmp.iloc[x,y]) == False):
                         y = (y+25)%24
                         if y == 0:
                             x = x+18
                             if x>= 1655:
                                 x = 1655
                     data_tmp.iloc[i, j] = (data_tmp.iloc[a,b]+data_tmp.iloc[x,y])/2
         data_tmp
```

處理前一小時資料

處理後一小時資料

取前一小時與後一小時的平均

Out[16]:

	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	2
0	23.70	23.80	23.80	23.90	23.90	23.80	24.10	24.70	26	27.20	...	29.90	29.60	28.70	27.50	26.40	25.70	25.50	25.30	24.90	24.5
1	1.97	1.95	1.96	1.96	1.95	1.96	1.97	1.97	1.96	1.98	...	1.97	1.98	1.97	2	2.03	2.04	2.05	2.02	2.10	2.1
2	0.23	0.22	0.21	0.20	0.20	0.22	0.24	0.29	0.27	0.33	...	0.29	0.30	0.33	0.38	0.46	0.50	0.45	0.39	0.46	0.4
3	0.06	0.05	0.03	0.03	0.03	0.04	0.04	0.05	0.06	0.07	...	0.06	0.07	0.09	0.11	0.13	0.15	0.10	0.07	0.12	0.1
4	1.20	0.70	0.50	0.70	0.50	0.30	0.70	0.90	1	1.80	...	1.30	1	0.90	0.80	0.50	0.90	0.90	0.30	0.70	0.5

```
In [44]: data_df = pd.DataFrame()
         data_df = pd.concat([data_text, data_tmp], axis = 1)
         data_df
```

Out[44]:

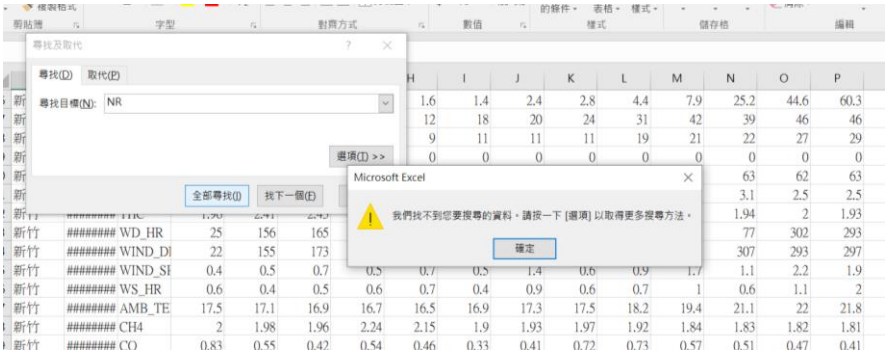
	測站	日期	測項	0	1	2	3	4	5	6	...	14	15	16	17	18	19	20	21	22	23
0	新竹	2020-10-01	AMB_TEMP	23.70	23.80	23.80	23.90	23.90	23.80	24.10	...	29.90	29.60	28.70	27.50	26.40	25.70	25.50	25.30	24.90	24.50
1	新竹	2020-10-01	CH4	1.97	1.95	1.96	1.96	1.95	1.96	1.97	...	1.97	1.98	1.97	2	2.03	2.04	2.05	2.02	2.10	2.14
2	新竹	2020-10-01	CO	0.23	0.22	0.21	0.20	0.20	0.22	0.24	...	0.29	0.30	0.33	0.38	0.46	0.50	0.45	0.39	0.46	0.45
3	新竹	2020-10-01	NMHC	0.06	0.05	0.03	0.03	0.03	0.04	0.04	...	0.06	0.07	0.09	0.11	0.13	0.15	0.10	0.07	0.12	0.18
4	新竹	2020-10-01	NO	1.20	0.70	0.50	0.70	0.50	0.30	0.70	...	1.30	1	0.90	0.80	0.50	0.90	0.90	0.30	0.70	0.90
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1651	新竹	2020-12-31	THC	2.01	2.02	2	2	1.99	2	1.98	...	2.03	2.07	2.07	2.10	2.10	2.07	2.07	2.05	2.04	2.07
1652	新竹	2020-12-31	WD_HR	54	55	54	53	58	52	52	...	54	50	52	45	47	42	42	47	45	44
1653	新竹	2020-12-31	WIND_DIREC	53	52	57	58	49	54	36	...	48	43	44	33	50	40	46	46	51	38
1654	新竹	2020-12-31	WIND_SPEED	4.70	4.60	4.70	4.90	4.10	5.30	5.50	...	4.50	4.40	4.20	3.80	3.70	4.70	4.50	4.40	3.90	3.90
1655	新竹	2020-12-31	WS_HR	3.70	3.60	3.60	3.50	3.50	3.30	3.80	...	3.70	3.10	3.30	3.10	2.90	3.30	3.10	2.90	2.80	2.60

1656 rows × 27 columns

最終處理完缺失值及無效值後所呈現的 Dataframe

3. **NR 表示無降雨，以 0 取代**

   資料集中並沒有 NR 值，因此不須處理



4. **將資料切割成訓練集(10.11 月)以及測試集(12 月)**

   ⇨ 訓練集為 10-11 月資料，測試集為 12 月資料

   ⇨ 將測站、日期 drop 掉

```
In [18]: #將資料切割成訓練集(10.11月)以及測試集(12月)
         train = data_df[data_df['日期                ']<'2020-12-01']
         test = data_df[data_df['日期                ] >= '2020-12-01']

In [19]: print(train.shape)
         print(test.shape)

         (1098, 27)
         (558, 27)

In [20]: train = train.drop(['測站                ','日期                ], axis = 1)
         train
```

Out[20]:

	測項	0	1	2	3	4	5	6	7	8	...	14	15	16	17	18	19	20	21	22	23
0	AMB_TEMP	23.70	23.80	23.80	23.90	23.90	23.80	24.10	24.70	26	...	29.90	29.60	28.70	27.50	26.40	25.70	25.50	25.30	24.90	24.50
1	CH4	1.97	1.95	1.96	1.96	1.95	1.96	1.97	1.97	1.96	...	1.97	1.98	1.97	2	2.03	2.04	2.05	2.02	2.10	2.14
2	CO	0.23	0.22	0.21	0.20	0.20	0.22	0.24	0.29	0.27	...	0.29	0.30	0.33	0.38	0.46	0.50	0.45	0.39	0.46	0.45
3	NMHC	0.06	0.05	0.03	0.03	0.03	0.04	0.04	0.05	0.06	...	0.06	0.07	0.09	0.11	0.13	0.15	0.10	0.07	0.12	0.18
4	NO	1.20	0.70	0.50	0.70	0.50	0.30	0.70	0.90	1	...	1.30	1	0.90	0.80	0.50	0.90	0.90	0.30	0.70	0.90
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1093	THC	1.95	1.94	1.95	1.96	1.95	1.95	1.95	1.96	1.99	...	1.98	2	1.99	2.03	2.01	2.04	2.02	2.02	2.02	2.01
1094	WD_HR	52	53	49	50	58	55	56	57	52	...	53	52	55	51	60	45	36	47	46	39
1095	WIND_DIREC	53	50	49	47	63	54	65	59	45	...	59	43	60	56	57	41	30	55	38	41
1096	WIND_SPEED	4.40	4.20	5.20	4.80	6.10	5.40	5.80	6.60	5.20	...	5.50	5.40	4.70	5.10	5.60	5.50	5.80	5.20	4.60	4.80
1097	WS_HR	3.50	3.60	3.60	3.80	3.90	4.10	4	4.50	4	...	4.60	4.20	3.80	3.80	4.50	4.10	5.30	3.80	3.40	3.90

5. **製作時序資料: 將資料形式轉換為行(row)代表 18 種屬性，欄(column)代表逐時數據資料**

In [21]:
```
df_train = pd.DataFrame()
df_train = train[0:18]
for i in range(18, train.shape[0], 18): #1098
    df_train = pd.merge(df_train, train[i:i+18], how = 'right',left_on='測項                    ', right_on = '測項
```

In [22]:
```
print(df_train.shape)
print(type(df_train))
```

```
(18, 1465)
<class 'pandas.core.frame.DataFrame'>
```

In [23]:
```
df_train = df_train.transpose()
df_train = df_train.reset_index(drop = True)
df_train = df_train.transpose()
```

In [24]:
```
df_train
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	...	1455	1456	1457	1458	1459	1460	1461	1462	1463	1464
0	AMB_TEMP	23.70	23.80	23.80	23.90	23.90	23.80	24.10	24.70	26	...	21.60	21.50	20.40	20	20.10	19.90	19.40	18.90	18.90	18.70
1	CH4	1.97	1.95	1.96	1.96	1.95	1.96	1.97	1.97	1.96	...	1.93	1.94	1.93	1.94	1.94	1.95	1.95	1.95	1.95	1.95
2	CO	0.23	0.22	0.21	0.20	0.20	0.22	0.24	0.29	0.27	...	0.26	0.27	0.27	0.29	0.29	0.31	0.25	0.22	0.20	0.18
3	NMHC	0.06	0.05	0.03	0.03	0.03	0.04	0.04	0.05	0.06	...	0.05	0.06	0.06	0.09	0.07	0.09	0.07	0.07	0.07	0.06
4	NO	1.20	0.70	0.50	0.70	0.50	0.30	0.70	0.90	1	...	2.50	2.40	2	1.80	1.60	1.60	1.80	1.70	1.60	1.60
5	NO2	8	6	5.50	5.20	5.30	5.80	8	7.60	6.60	...	4.50	5.40	6.60	9	7.50	8.60	6.90	6	4.80	4.10
6	NOx	9.20	6.70	6.10	5.80	5.80	6.30	8.60	8.50	7.60	...	6.90	7.70	8.50	10.80	9.10	10.30	8.70	7.80	6.30	5.70
7	O3	48	50.60	53.10	53	50.50	47.80	44.80	46.60	51.90	...	42.40	39.70	35.90	32.40	34.50	33.50	35.20	34.90	36.30	37.80

四. 時間序列

a. 將未來第一個小時當預測目標

1. X 只取 PM 2.5

切割 x_train, x_test, y_train, y_test

In [31]:
```
x_train = np.zeros((df_train_drop.shape[1]-6,6))
x_test= np.zeros((df_test_drop.shape[1]-6,6))

y_train = np.zeros((df_train_drop.shape[1]-6,1))
y_test = np.zeros((df_test_drop.shape[1]-6,1))

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

for i in range(0,x_train.shape[0]):
    x_train[i] = df_train_drop.iloc[9,i:i+6]
    y_train[i] = df_train_drop.iloc[9,i+6]

for i in range(0,x_test.shape[0]):
    x_test[i] = df_test_drop.iloc[9,i:i+6]
    y_test[i] = df_test_drop.iloc[9,i+6]
```

```
(1458, 6)
(738, 6)
(1458, 1)
(738, 1)
```

⇨ Linear Regression

- Linear Regression with X: PM (將未來第一個小時當預測目標)

In [32]:
```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(x_test)


print("Linear Regression predict one hour (X: PM 2.5)", end = "\n")
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test, y_pred))
```

```
Linear Regression predict one hour (X: PM 2.5)
Mean squared error: 11.84
Mean absolute error: 2.52
```

⇨ XGBoost

- XGBoost with X: PM (將未來第一個小時當預測目標)

In [46]:
```
XGB = XGBRegressor(n_estimators=10, gamma = 35) #調整XGBoost參數
XGB.fit(x_train, y_train)
y_pred = XGB.predict(x_test)


print("XGBoost predict one hour (X: PM 2.5)", end = "\n")
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test, y_pred))
```

```
XGBoost predict one hour (X: PM 2.5)
Mean squared error: 16.62
Mean absolute error: 2.70
```

2. X 取所有 18 種屬性

切割 x_train, x_test, y_train, y_test

2. X 取所有18種屬性

In [34]:
```
x_train_all = np.array(df_train_drop.iloc[:, 0:6]).flatten()
for i in range(1,df_train_drop.shape[1]-6):
    x_train_all = np.vstack((x_train_all, np.array(df_train_drop.iloc[:,i:i+6]).flatten()))
y_train_all = np.array(df_train_drop.iloc[9, 6:df_train_drop.shape[1]])

x_test_all= np.array(df_test_drop.iloc[:,0:6]).flatten()
for i in range(1,df_test_drop.shape[1]-6):
    x_test_all = np.vstack((x_test_all, np.array(df_test_drop.iloc[:,i:i+6]).flatten()))
y_test_all = np.array(df_test_drop.iloc[9, 6:df_test_drop.shape[1]])


print(x_train_all.shape)
print(x_test_all.shape)
print(y_train_all.shape)
print(y_test_all.shape)
```

```
(1458, 108)
(738, 108)
(1458,)
(738,)
```

⇨ Linear Regression

- Linear Regression with X: 18種屬性 (將未來第一個小時當預測目標)

In [35]:
```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(x_train_all, y_train_all)

# Make predictions using the testing set
y_pred_all = regr.predict(x_test_all)

print("Linear Regression predict one hour (X: 18種屬性)", end = "\n")
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test_all, y_pred_all))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test_all, y_pred_all))
```

```
Linear Regression predict one hour (X: 18種屬性)
Mean squared error: 13.88
Mean absolute error: 2.69
```

⇨ XGBoost

- XGBoost with X: 18種屬性 (將未來第一個小時當預測目標)

In [36]:
```
XGB = XGBRegressor(n_estimators=20, gamma = 40) #調整XGBoost參數
XGB.fit(x_train_all, y_train_all)
y_pred_all = XGB.predict(x_test_all)


print("XGBoost predict one hour (X: 18種屬性)", end = "\n")
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test_all, y_pred_all))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test_all, y_pred_all))
```

```
XGBoost predict one hour (X: 18種屬性)
Mean squared error: 15.02
Mean absolute error: 2.69
```

b. 將未來第六個小時當預測目標

1. X 只取 PM 2.5

切割 x_train, x_test, y_train, y_test

將未來第六個小時當預測目標

1. X 只取 PM2.5

In [37]:
```
x_train = np.zeros((df_train_drop.shape[1]-11,6))
x_test= np.zeros((df_test_drop.shape[1]-11,6))

y_train = np.zeros((df_train_drop.shape[1]-11,1))
y_test = np.zeros((df_test_drop.shape[1]-11,1))

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

for i in range(0,x_train.shape[0]):
    x_train[i] = df_train_drop.iloc[9,i:i+6]
    y_train[i] = df_train_drop.iloc[9,i+11]

for i in range(0,x_test.shape[0]):
    x_test[i] = df_test_drop.iloc[9,i:i+6]
    y_test[i] = df_test_drop.iloc[9,i+11]
```

```
(1453, 6)
(733, 6)
(1453, 1)
(733, 1)
```

⇨ Linear Regression

Create Model

- Linear Regression with X: PM 2.5 (將未來第六個小時當預測目標)

In [38]:
```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(x_test)

print("Linear Regression predict six hour (X: PM 2.5)", end = "\n")
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test, y_pred))
```

```
Linear Regression predict six hour (X: PM 2.5)
Mean squared error: 41.64
Mean absolute error: 4.58
```

⇨ XGBoost

- XGBoost with X: PM 2.5 (將未來第六個小時當預測目標)

In [39]:
```
XGB = XGBRegressor(n_estimators=10, gamma = 60) #調整XGBoost參數
XGB.fit(x_train, y_train)
y_pred = XGB.predict(x_test)

print("XGBoost predict six hour (X: PM 2.5)", end = "\n")
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test, y_pred))
```

```
XGBoost predict six hour (X: PM 2.5)
Mean squared error: 45.68
Mean absolute error: 4.77
```

2. X 取所有 18 種屬性

切割 x_train, x_test, y_train, y_test

2. X 取18種屬性

In [40]:
```
x_train_all = np.array(df_train_drop.iloc[:, 0:6]).flatten()
for i in range(1,df_train_drop.shape[1]-11):
    x_train_all = np.vstack((x_train_all, np.array(df_train_drop.iloc[:,i:i+6]).flatten()))
y_train_all = np.array(df_train_drop.iloc[9, 11:df_train_drop.shape[1]])

x_test_all= np.array(df_test_drop.iloc[:,0:6]).flatten()
for i in range(1,df_test_drop.shape[1]-11):
    x_test_all = np.vstack((x_test_all, np.array(df_test_drop.iloc[:,i:i+6]).flatten()))
y_test_all = np.array(df_test_drop.iloc[9, 11:df_test_drop.shape[1]])


print(x_train_all.shape)
print(x_test_all.shape)
print(y_train_all.shape)
print(y_test_all.shape)
```

```
(1453, 108)
(733, 108)
(1453,)
(733,)
```

⇨ Linear Regression

- Linear Regression with X: 所有18種屬性 (將未來第六個小時當預測目標)

In [41]:
```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(x_train_all, y_train_all)

# Make predictions using the testing set
y_pred_all = regr.predict(x_test_all)

print("Linear Regression predict six hour (X: 18種屬性)", end = "\n")
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test_all, y_pred_all))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test_all, y_pred_all))
```

```
Linear Regression predict six hour (X: 18種屬性)
Mean squared error: 58.03
Mean absolute error: 6.09
```

⇨ XGBoost

- XGBoost with X: 所有18種屬性 (將未來第六個小時當預測目標)

In [42]:
```
XGB = XGBRegressor(n_estimators=20, gamma = 60) #調整XGBoost參數
XGB.fit(x_train_all, y_train_all)
y_pred_all = XGB.predict(x_test_all)


print("XGBoost predict one hour (X: 18種屬性)", end = "\n")
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test_all, y_pred_all))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test_all, y_pred_all))
```

```
XGBoost predict one hour (X: 18種屬性)
Mean squared error: 44.53
Mean absolute error: 4.58
```

統整表格:

Mean absolute error	Linear Regression	XGBoost
將未來第一個小時當預測目標(X : PM2.5)	2.52	2.70
將未來第一個小時當預測目標(X : 所有 18 種屬性)	2.69	2.69
將未來第六個小時當預測目標(X : PM2.5)	4.58	4.77
將未來第六個小時當預測目標(X : 所有 18 種屬性)	6.09	4.58

由表格可以看出，將未來第一個小時當預測目標會比將未來第六個小時當預測目標更為準確; X 只取 PM2.5 相較於 X 取 18 種屬性而言，Linear Regression 會更為準確，但對於 XGBoost 來說，X 取 18 種屬性似乎會又較佳的預測成果。