- [1]：import packages

In [1]:
```
import pandas as pd
import numpy as np
import datetime
import warnings
warnings.filterwarnings('ignore')
```

- [2]：讀取資料，將資料欄位重新命名(column name)(date 為 timestamp，type 為汙染物種類)，並去除資料中的多餘空格。

In [2]:
```
# 讀取資料，將沒有用到的col(測站-region)刪掉，重新命名date和type
data = pd.read_csv('./data/新竹_2020.csv',skipinitialspace = True)[1:].reset_index(drop=True)
data.columns.values[0], data.columns.values[1],data. columns.values[2] = 'region','date','type'
data = data.drop('region', axis=1)

# 將多餘空格移除
data.columns = data.columns.str.replace(' ', '')
tmp = [e for e in list(data.columns) if e is not 'date']
for i in tmp:
    data[i] = data[i].str.replace(' ','')
del tmp

data
```

Out[2]:

	date	type	00	01	02	03	04	05	06	07	...	14	15	16	17	18	19	20	21	22	23
0	2020/01/01 00:00:00	AMB_TEMP	15.2	15.2	15.3	15.3	15.3	15.4	15.5	15.8	...	18.1	18.2	17.9	17.3	16.7	16.4	16.2	16.1	16	15.8
1	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	1.78	1.77	1.8	1.81	1.82	1.85	1.83	1.92	1.94
2	2020/01/01 00:00:00	CO	0.28	0.25	0.24	0.22	0.2	0.19	0.2	0.23	...	0.28	0.29	0.28	0.34	0.39	0.41	0.46	0.49	0.58	0.52
3	2020/01/01 00:00:00	NMHC	0.06	0.07	0.05	0.05	0.05	0.05	0.07	0.07	...	0.09	0.09	0.07	0.08	0.12	0.12	0.16	0.14	0.17	0.2
4	2020/01/01 00:00:00	NO	0.3	0.6	0.6	0.6	0.3	0.3	0.5	0.6	...	1.6	1.6	1.2	0.7	0.9	1.1	1.1	1.7	1.8	1.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6583	2020/12/31 00:00:00	THC	2.01	2.02	2	2	1.99	2	1.98	2	...	2.03	2.07	2.07	2.1	2.1	2.07	2.07	2.05	2.04	2.07
6584	2020/12/31 00:00:00	WD_HR	54	55	54	53	58	52	52	35	...	54	50	52	45	47	42	42	47	45	44
6585	2020/12/31 00:00:00	WIND_DIREC	53	52	57	58	49	54	36	33	...	48	43	44	33	50	40	46	46	51	38
6586	2020/12/31 00:00:00	WIND_SPEED	4.7	4.6	4.7	4.9	4.1	5.3	5.5	5.6	...	4.5	4.4	4.2	3.8	3.7	4.7	4.5	4.4	3.9	3.9
6587	2020/12/31 00:00:00	WS_HR	3.7	3.6	3.6	3.5	3.5	3.3	3.8	3.8	...	3.7	3.1	3.3	3.1	2.9	3.3	3.1	2.9	2.8	2.6

6588 rows × 26 columns

- [3]：使用 datetime 把 train/test 資料取出，用 groupby 將不同汙染物分組

In [3]:
```
data['date'] = pd.to_datetime(data['date'])

# 取出10, 11, 12月的資料
data = data[data['date']>=pd.datetime(2020,10,1)].reset_index(drop=True)

# 根據不同汙染物分組
group = data.groupby('type')
# 汙染物種類
polution_type = list(group.groups.keys())
# col_list = ['00','01',...,'23']
col_list = [e for e in list(data.columns) if e not in ('date', 'type')]
# process_data存填完值的dataframe
process_data = []
```

- [3]：用迴圈進行填空值的資料處理。在迴圈 i 中，df 為汙染物種類為 polution_type[i]的 dataframe，將此 dataframe 轉為一維 list，使用迴圈 j 找到內部空值(非 float 的值)，並將其填為'nan'。後將 list 轉回大小為(n*1)的 dataframe df_tmp，利用 ffill 和 bfill 將缺失值'nan'填為前一個非'nan'和後一個非'nan'值的平均。將 dataframe 進行 reshape 轉回 df 的 shape，並將每個 df concat 回原本 data 的 shape (new_df)。

```
for i in range(len(polution_type)):
    # 根據汙染物type分組的dataframe
    df = group.get_group(polution_type[i]).reset_index()
    display(df)
    # 將df的element value壓縮成list檢查有沒有空值
    list_tmp = df.drop(['index','date','type'],axis=1).values.reshape(-1,).tolist()
    for j in range(len(list_tmp)):
        element = list_tmp[j]
        # 轉成float
        try:
            list_tmp[j] = float(element)
        # 有缺失值
        except ValueError:
            if element == 'NR':
                print('no rain')
                list_tmp[j] = 0
            else:
#                 print('{},{}, {}'.format(polution_type[i],j%24,element))
                # 將缺失值設成nan
                list_tmp[j] = float('nan')
    # 將list_tmp轉成dataframe with one column
    df_tmp = pd.DataFrame (list_tmp, columns = ['tmp'])
    # 缺失值(nan)設定成前一個非nan與後一個非nan的平均
    df_tmp = (df_tmp.ffill()+df_tmp.bfill())/2
    df_tmp = df_tmp.bfill().ffill()
#     display(df_tmp)
    # dataframe轉回list_tmp
    list_tmp = df_tmp['tmp'].tolist()
    new_data = pd.DataFrame(np.array(list_tmp).reshape(len(df),len(col_list)), columns = col_list)
#     display(new_data)
    for col in col_list:
        df[col] = new_data[col]
#     display(df.head(10))
    process_data.append(df)

new_df = pd.concat(process_data)
```

Out[3]:

	index	date	type	00	01	02	03	04	05	06	...	14	15	16	17	18	19	20	21	22	23
0	0	2020-10-01	AMB_TEMP	23.7	23.8	23.8	23.9	23.9	23.8	24.1	...	29.9	29.6	28.7	27.5	26.4	25.7	25.5	25.3	24.9	24.5
1	18	2020-10-02	AMB_TEMP	24.5	24.5	24.1	24.0	23.9	23.8	23.9	...	31.2	31.1	30.2	28.6	27.6	27.4	27.1	26.6	26.4	26.0
2	36	2020-10-03	AMB_TEMP	25.7	25.3	25.0	24.7	24.2	24.0	24.0	...	31.8	31.6	30.4	30.1	29.8	29.7	28.8	27.9	27.6	27.2
3	54	2020-10-04	AMB_TEMP	26.4	26.0	25.7	25.8	25.4	25.2	25.2	...	31.2	31.0	30.7	30.0	29.5	29.0	28.3	28.1	27.9	27.7
4	72	2020-10-05	AMB_TEMP	27.4	27.1	26.8	25.8	25.4	25.3	25.2	...	27.5	26.9	26.2	25.5	25.3	25.1	24.6	24.2	24.0	23.7
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
87	1583	2020-12-27	WS_HR	1.4	1.0	1.0	0.7	0.8	1.0	0.8	...	1.2	1.4	1.8	1.4	1.5	1.0	1.2	1.4	1.3	1.3
88	1601	2020-12-28	WS_HR	1.8	1.7	2.0	2.5	3.0	2.8	3.0	...	2.6	2.5	1.9	2.1	1.5	1.3	1.1	0.7	0.4	0.7
89	1619	2020-12-29	WS_HR	0.6	0.6	0.1	0.6	0.4	0.7	0.5	...	0.8	1.2	1.1	1.4	1.4	1.3	1.4	1.7	1.7	1.9
90	1637	2020-12-30	WS_HR	3.1	4.2	4.5	3.8	4.8	4.7	4.7	...	3.9	3.8	4.1	3.8	2.8	2.8	3.1	2.6	3.1	3.7
91	1655	2020-12-31	WS_HR	3.7	3.6	3.6	3.5	3.5	3.3	3.8	...	3.7	3.1	3.3	3.1	2.9	3.3	3.1	2.9	2.8	2.6

1656 rows × 27 columns

- [4]：用 datetime 根據 timestamp (new_df['date'])，將 data 切分成 training set 和 test set。分別有 1098 和 558 筆資料。

In [4]:
```
# 10~11月資料 (1098 entries)
train = new_df[(new_df['date']>=pd.datetime(2020,10,1)) & (new_df['date']<=pd.datetime(2020,11,30))].reset_index()
# 12月資料 (558 entries)
test = new_df[(new_df['date']>=pd.datetime(2020,12,1)) & (new_df['date']<=pd.datetime(2020,12,31))].reset_index()
```

- [5]：使用迴圈將 train/test 兩個 dataframe 進行 reshape，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。train_tran 為(18,61*24)的 dataframe (每個屬性都有 61 天*24 小時共 1464 筆資料)。

In [5]:
```
group_train = train.groupby('type')
group_test = test.groupby('type')

train_trans = pd.DataFrame()
train_trans['type'] = ''
test_trans = pd.DataFrame()
test_trans['type'] = ''

for i in range(len(polution_type)):
    # polution type為polution_type[i]的dataframe
    a = group_train.get_group(polution_type[i])
    b = group_test.get_group(polution_type[i])
    # 轉成一維向量
    a_list = a.drop(['level_0','index','date','type'],axis=1).values.reshape(-1,).tolist()
    b_list = b.drop(['level_0','index','date','type'],axis=1).values.reshape(-1,).tolist()
    # 一維向量轉成dataframe row
    df_a = pd.DataFrame([a_list])
    df_b = pd.DataFrame([b_list])
    # 集合成train/test set
    train_trans = pd.concat((train_trans,df_a))
    train_trans['type'].iloc[i] = polution_type[i]
    test_trans = pd.concat((test_trans,df_b))
    test_trans['type'].iloc[i] = polution_type[i]

train_trans
# test_trans
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	1455	1456	1457	1458	1459	1460	1461	1462	1463	type
0	23.70	23.80	23.80	23.90	23.90	23.80	24.10	24.70	26.00	27.20	...	21.50	20.40	20.00	20.10	19.90	19.40	18.90	18.90	18.70	AMB_TEMP
0	1.97	1.95	1.96	1.96	1.95	1.96	1.97	1.97	1.96	1.98	...	1.94	1.93	1.94	1.94	1.95	1.95	1.95	1.95	1.95	CH4
0	0.23	0.22	0.21	0.20	0.20	0.22	0.24	0.29	0.27	0.33	...	0.27	0.27	0.29	0.29	0.31	0.25	0.22	0.20	0.18	CO
0	0.06	0.05	0.03	0.03	0.03	0.04	0.04	0.05	0.06	0.07	...	0.06	0.06	0.09	0.07	0.09	0.07	0.07	0.07	0.06	NMHC
0	1.20	0.70	0.50	0.70	0.50	0.30	0.70	0.90	1.00	1.80	...	2.40	2.00	1.80	1.60	1.60	1.80	1.70	1.60	1.60	NO
0	8.00	6.00	5.50	5.20	5.30	5.80	8.00	7.60	6.60	8.00	...	5.40	6.60	9.00	7.50	8.60	6.90	6.00	4.80	4.10	NO2
0	9.20	6.70	6.10	5.80	5.80	6.30	8.60	8.50	7.60	9.80	...	7.70	8.50	10.80	9.10	10.30	8.70	7.80	6.30	5.70	NOx
0	48.00	50.60	53.10	53.00	50.50	47.80	44.80	46.60	51.90	55.80	...	39.70	35.90	32.40	34.50	33.50	35.20	34.90	36.30	37.80	O3
0	21.00	24.00	28.00	26.00	28.00	22.00	26.00	27.00	29.00	23.00	...	30.00	15.00	14.00	14.00	16.00	11.00	18.00	14.00	18.00	PM10
0	16.00	9.00	11.00	10.00	9.00	15.00	10.00	10.00	10.00	9.00	...	9.00	5.00	3.00	4.00	6.00	7.00	9.00	9.00	5.00	PM2.5
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.20	0.40	0.40	0.40	0.60	0.40	0.60	0.80	0.60	RAINFALL
0	72.00	71.00	72.00	72.00	72.00	72.00	72.00	68.00	61.00	55.00	...	60.00	69.00	72.00	72.00	74.00	78.00	82.00	82.00	82.00	RH
0	2.00	2.20	2.30	2.60	2.80	3.00	3.40	2.90	2.50	2.40	...	2.80	3.40	3.30	2.80	2.50	2.00	2.40	2.10	2.10	SO2
0	2.03	2.00	1.99	1.99	1.98	2.00	2.01	2.02	2.02	2.05	...	2.00	1.99	2.03	2.01	2.04	2.02	2.02	2.02	2.01	THC
0	49.00	49.00	52.00	55.00	54.00	54.00	55.00	46.00	48.00	47.00	...	52.00	55.00	51.00	60.00	45.00	36.00	47.00	46.00	39.00	WD_HR
0	57.00	43.00	49.00	60.00	58.00	47.00	54.00	46.00	57.00	38.00	...	43.00	60.00	56.00	57.00	41.00	30.00	55.00	38.00	41.00	WIND_DIREC
0	3.70	2.90	3.30	3.00	3.20	2.50	2.90	3.10	4.20	4.30	...	5.40	4.70	5.10	5.60	5.50	5.80	5.20	4.60	4.80	WIND_SPEED
0	2.50	2.20	2.50	2.50	2.40	2.30	2.10	2.40	3.10	3.40	...	4.20	3.80	3.80	4.50	4.10	5.30	3.80	3.40	3.90	WS_HR

18 rows × 1465 columns

- [6]：test_tran 為(18,31*24)的 dataframe (每個屬性都有 31 天*24 小時共 744 筆資料)。

In [6]:
```
test_trans
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	...	735	736	737	738	739	740	741	742	743	type
0	18.50	18.40	18.30	18.20	18.20	18.30	18.40	18.60	19.30	20.30	...	11.00	10.80	10.50	10.50	10.60	10.80	10.90	11.00	11.00	AMB_TEMP
0	1.95	1.95	1.95	1.95	1.95	1.95	1.94	1.95	1.95	1.95	...	1.98	1.98	1.99	1.99	1.99	1.98	1.98	1.97	1.99	CH4
0	0.17	0.16	0.16	0.16	0.16	0.17	0.17	0.25	0.28	0.27	...	0.30	0.31	0.34	0.34	0.31	0.29	0.28	0.26	0.30	CO
0	0.06	0.05	0.06	0.06	0.04	0.06	0.03	0.08	0.09	0.11	...	0.09	0.09	0.11	0.11	0.08	0.09	0.07	0.07	0.08	NMHC
0	1.30	1.50	1.20	1.30	1.20	1.30	1.30	1.90	3.40	4.00	...	3.00	2.80	2.30	2.10	1.60	1.50	1.50	1.40	1.40	NO
0	3.00	2.90	3.80	3.60	3.60	4.20	4.70	7.40	9.00	7.90	...	10.30	11.90	13.70	13.10	10.80	9.30	8.60	7.70	9.70	NO2
0	4.40	4.50	5.00	4.90	4.80	5.50	6.00	9.30	12.40	11.80	...	13.30	14.80	16.00	15.20	12.40	10.80	10.20	9.10	11.10	NOx
0	38.70	37.80	35.90	35.60	35.40	33.70	33.50	30.90	30.80	32.30	...	28.30	27.10	25.10	26.20	28.40	28.90	29.50	29.70	25.80	O3
0	17.00	13.00	12.00	10.00	5.00	13.00	15.00	12.00	12.00	17.00	...	36.00	25.00	34.00	30.00	31.00	27.00	26.00	23.00	27.00	PM10
0	5.00	9.00	7.00	4.00	5.00	11.00	7.00	5.00	5.00	5.00	...	16.00	20.00	12.00	17.00	13.00	12.00	15.00	18.00	15.00	PM2.5
0	0.80	0.60	1.00	1.00	0.80	0.80	0.80	1.00	0.80	0.40	...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	RAINFALL
0	85.00	86.00	87.00	87.00	86.00	86.00	87.00	86.00	85.00	80.00	...	60.00	61.00	62.00	62.00	61.00	61.00	60.00	60.00	62.00	RH
0	0.50	1.90	1.80	1.80	1.80	2.00	1.80	2.50	3.10	3.30	...	6.90	6.90	6.90	6.90	6.90	6.90	6.90	6.90	6.90	SO2
0	2.01	2.00	2.01	2.01	1.99	2.01	1.97	2.03	2.04	2.06	...	2.07	2.07	2.10	2.10	2.07	2.07	2.05	2.04	2.07	THC
0	43.00	45.00	49.00	43.00	50.00	35.00	36.00	34.00	34.00	37.00	...	50.00	52.00	45.00	47.00	42.00	42.00	47.00	45.00	44.00	WD_HR
0	31.00	38.00	40.00	47.00	55.00	37.00	30.00	41.00	35.00	35.00	...	43.00	44.00	33.00	50.00	40.00	46.00	46.00	51.00	38.00	WIND_DIREC
0	4.30	4.20	3.80	3.90	4.70	5.30	4.30	4.00	4.60	5.30	...	4.40	4.20	3.80	3.70	4.70	4.50	4.40	3.90	3.90	WIND_SPEED
0	3.50	3.00	3.00	3.10	3.00	4.00	3.60	3.30	3.70	4.00	...	3.10	3.30	3.10	2.90	3.30	3.10	2.90	2.80	2.60	WS_HR

18 rows × 745 columns

- [7]：

a. 預測目標分成兩種：

①將未來第一個小時當預測目標(取 6 小時為一單位 x_train_1hour 作為訓練資料，第 7 小時的 pm2.5 值 y_train_1hour_pm25 作為訓練預測值(切割後 X 的長度為 1464-6=1458)

②將未來第六個小時當預測目標(取 6 小時為一單位 x_train_6hour 作為訓練資料，第 11 小時的 pm2.5 值 y_train_6hour_pm25 作為訓練預測值(切割後 X 的長度為 1464-11=1453)

b. 訓練特徵分成兩種：

①只有 PM2.5 (x_train_pm25，6 個特徵，第 0~5 小時的 PM2.5 數值)

②所有 18 種屬性(x_train_all，18*6 個特徵，第 0~5 小時的 18 種屬性數值)

In [7]:
```
# 訓練集分成看全部汙染物和只看pm2.5，前六小時的資料
# 將未來第一個小時當預測目標 (training set)
x_train_1hour_all, x_train_1hour_pm25 = [], []
y_train_1hour_pm25 = []
# 將未來第六個小時當預測目標 (training set)
x_train_6hour_all, x_train_6hour_pm25 = [], []
y_train_6hour_pm25 = []

train_new = train_trans.copy()
test_new = test_trans.copy()

# hint: 切割後X的長度應為1464-6=1458
# range-1：不要append到type
for i in range(train_new.shape[1]-7-1):
    # 前六小時數值
    sixHourBefore_all = train_new.iloc[:,i:(i+6)].values.reshape(-1,).tolist()
    sixHourBefore_pm25 = train_new.loc[train_new['type']=='PM2.5'].iloc[:,i:(i+6)].values.reshape(-1,).tolist()
    x_train_1hour_all.append(sixHourBefore_all)
    x_train_1hour_pm25.append(sixHourBefore_pm25)
    # 未來第一小時PM2.5數值(y_1hour)
    oneHourAfter_pm25 = train_new.loc[train_new['type']=='PM2.5'].iloc[:,(i+6+1)][0]
    y_train_1hour_pm25.append(oneHourAfter_pm25)

# hint: 切割後X的長度應為1464-11=1453
# range-1：不要append到type
for i in range(train_new.shape[1]-12-1):
    # 前六小時數值
    sixHourBefore_all = train_new.iloc[:,i:(i+6)].values.reshape(-1,).tolist()
    sixHourBefore_pm25 = train_new.loc[train_new['type']=='PM2.5'].iloc[:,i:(i+6)].values.reshape(-1,).tolist()
    x_train_6hour_all.append(sixHourBefore_all)
    x_train_6hour_pm25.append(sixHourBefore_pm25)
    # 未來第六小時PM2.5數值(y_1hour)
    sixHourAfter_pm25 = train_new.loc[train_new['type']=='PM2.5'].iloc[:,(i+6+6)][0]
    y_train_6hour_pm25.append(sixHourAfter_pm25)
```

- [8]：以和[7]同樣的方法切割測試集(兩種訓練特徵與兩種預測目標)

In [8]:
```
# 測試集分成看全部汙染物和只看pm2.5，前六小時的資料
# 將未來第一個小時當預測目標 (test set)
x_test_1hour_all, x_test_1hour_pm25 = [], []
y_test_1hour_pm25 = []
# 將未來第六個小時當預測目標 (test set)
x_test_6hour_all, x_test_6hour_pm25 = [], []
y_test_6hour_pm25 = []
# 丟入x_test得到預測答案，在用y_test去對答案

# hint: 切割後X的長度應為744-6=738
# range-1：不要append到type
for i in range(test_new.shape[1]-7-1):
    # 前六小時數值
    sixHourBefore_all = test_new.iloc[:,i:(i+6)].values.reshape(-1,).tolist()
    sixHourBefore_pm25 = test_new.loc[test_new['type']=='PM2.5'].iloc[:,i:(i+6)].values.reshape(-1,).tolist()
    x_test_1hour_all.append(sixHourBefore_all)
    x_test_1hour_pm25.append(sixHourBefore_pm25)
    # 未來第一小時PM2.5數值(y_1hour)
    oneHourAfter_pm25 = test_new.loc[test_new['type']=='PM2.5'].iloc[:,(i+6+1)][0]
    y_test_1hour_pm25.append(oneHourAfter_pm25)

# hint: 切割後X的長度應為744-11=733
# range-1：不要append到type
for i in range(test_new.shape[1]-12-1):
    # 前六小時數值
    sixHourBefore_all = test_new.iloc[:,i:(i+6)].values.reshape(-1,).tolist()
    sixHourBefore_pm25 = test_new.loc[test_new['type']=='PM2.5'].iloc[:,i:(i+6)].values.reshape(-1,).tolist()
    x_test_6hour_all.append(sixHourBefore_all)
    x_test_6hour_pm25.append(sixHourBefore_pm25)
    # 未來第六小時PM2.5數值(y_1hour)
    sixHourAfter_pm25 = test_new.loc[test_new['type']=='PM2.5'].iloc[:,(i+6+6)][0]
    y_test_6hour_pm25.append(sixHourAfter_pm25)
```

- [9]：使用 Linear Regression 建四種模型。

  ① model_1 使用 6*18 種特徵，未來第一小時 PM2.5 數值當預測目標。

  ② model_2 使用 6 種特徵，未來第一小時 PM2.5 數值當預測目標。

  ③ model_3 使用 6*18 種特徵，未來第六小時 PM2.5 數值當預測目標。

  ④ model_4 使用 6 種特徵，未來第六小時 PM2.5 數值當預測目標。

  結果：MAE 分別為 3.58, 3.23, 6.40, 4.78。如同預期，預測越遠小時的數值會越不准，預測未來第六小時的 MAE 較高。而只看 PM2.5 特徵作為訓練和測試集，其預測會最準確(可能比較不受到其他種類汙染物的雜訊干擾)。

In [9]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

# model 1 ：使用linear Regression，x包含全部汙染物(所有18種屬性)，將未來第一個小時當預測目標
model_1 = LinearRegression()
model_1.fit(x_train_1hour_all, y_train_1hour_pm25)
model_1_y_pred = model_1.predict(x_test_1hour_all)
model_1_MAE = mean_absolute_error(y_test_1hour_pm25, model_1_y_pred)
score_1 = model_1.score(x_test_1hour_all, y_test_1hour_pm25)
print('model_1 MAE= {}'.format(model_1_MAE))

# model 2 ：使用linear Regression，x只有PM2.5，將未來第一個小時當預測目標
model_2 = LinearRegression()
model_2.fit(x_train_1hour_pm25, y_train_1hour_pm25)
model_2_y_pred = model_2.predict(x_test_1hour_pm25)
model_2_MAE = mean_absolute_error(y_test_1hour_pm25, model_2_y_pred)
score_2 = model_2.score(x_test_1hour_pm25, y_test_1hour_pm25)
print('model_2 MAE= {}'.format(model_2_MAE))
```

```
# model 3 ：使用linear Regression，x包含全部汙染物(所有18種屬性)，將未來第六個小時當預測目標
model_3 = LinearRegression()
model_3.fit(x_train_6hour_all, y_train_6hour_pm25)
model_3_y_pred = model_3.predict(x_test_6hour_all)
model_3_MAE = mean_absolute_error(y_test_6hour_pm25, model_3_y_pred)
score_3 = model_3.score(x_test_6hour_all, y_test_6hour_pm25)
print('model_3 MAE= {}'.format(model_3_MAE))


# model 4 ：使用linear Regression，x只有PM2.5，將未來第六個小時當預測目標
model_4 = LinearRegression()
model_4.fit(x_train_6hour_pm25, y_train_6hour_pm25)
model_4_y_pred = model_4.predict(x_test_6hour_pm25)
model_4_MAE = mean_absolute_error(y_test_6hour_pm25, model_4_y_pred)
score_4 = model_4.score(x_test_6hour_pm25, y_test_6hour_pm25)
print('model_4 MAE= {}'.format(model_4_MAE))
```

```
model_1 MAE= 3.5830487927081482
model_2 MAE= 3.229456981096107
model_3 MAE= 6.398858606335599
model_4 MAE= 4.7689672932060345
```

- [10]：使用 XGBoost 建四種模型。
  ① model_5 使用 6*18 種特徵，未來第一小時 PM2.5 數值當預測目標。
  ② model_6 使用 6 種特徵，未來第一小時 PM2.5 數值當預測目標。
  ③ model_7 使用 6*18 種特徵，未來第六小時 PM2.5 數值當預測目標。
  ④ model_8 使用 6 種特徵，未來第六小時 PM2.5 數值當預測目標。
  結果：MAE 分別為 3.96, 4.15, 5.23, 5.87。同樣如預期，預測越遠小時的數值會越不准，預測未來第六小時的 MAE 較高。而在使用此模型中，看全部汙染物特徵作為訓練和測試集，其預測會比只看 PM2.5 更準確。(資訊多)

In [10]:
```
from xgboost import XGBClassifier

# model 5 :使用XGBoost，x包含全部汙染物(所有18種屬性)，將未來第一個小時當預測目標
model_5 = XGBClassifier(learning_rate=0.1,
                        max_depth=10,
                        eval_metric='mlogloss')
model_5.fit(np.array(x_train_1hour_all), np.array(y_train_1hour_pm25))
model_5_y_pred = model_5.predict(np.array(x_test_1hour_all))
model_5_MAE = mean_absolute_error(y_test_1hour_pm25, model_5_y_pred)
print('model_5 MAE= {}'.format(model_5_MAE))

# model 6 :使用XGBoost，x只有PM2.5，將未來第一個小時當預測目標
model_6 = XGBClassifier(learning_rate=0.1,
                        max_depth=10,
                        eval_metric='mlogloss')
model_6.fit(np.array(x_train_1hour_pm25), np.array(y_train_1hour_pm25))
model_6_y_pred = model_6.predict(np.array(x_test_1hour_pm25))
model_6_MAE = mean_absolute_error(y_test_1hour_pm25, model_6_y_pred)
print('model_6 MAE= {}'.format(model_6_MAE))
```

```
# model 7 :使用XGBoost，x包含全部汙染物(所有18種屬性)，將未來第六個小時當預測目標
model_7 = XGBClassifier(learning_rate=0.1,
                        max_depth=10,
                        eval_metric='mlogloss')
model_7.fit(np.array(x_train_6hour_all), np.array(y_train_6hour_pm25))
model_7_y_pred = model_7.predict(np.array(x_test_6hour_all))
model_7_MAE = mean_absolute_error(y_test_6hour_pm25, model_7_y_pred)
print('model_7 MAE= {}'.format(model_7_MAE))

# model 8 :使用XGBoost，x只有PM2.5，將未來第六個小時當預測目標
model_8 = XGBClassifier(learning_rate=0.1,
                        max_depth=10,
                        eval_metric='mlogloss')
model_8.fit(np.array(x_train_6hour_pm25), np.array(y_train_6hour_pm25))
model_8_y_pred = model_8.predict(np.array(x_test_6hour_pm25))
model_8_MAE = mean_absolute_error(y_test_6hour_pm25, model_8_y_pred)
print('model_8 MAE= {}'.format(model_8_MAE))

# score = model_5.score(np.array(x_test_1hour_all), np.array(y_test_1hour_pm25).reshape((-1,1)))
```

```
model_5 MAE= 3.960651289009498
model_6 MAE= 4.146540027137042
model_7 MAE= 5.229508196721311
model_8 MAE= 5.870901639344262
```