民生公共物聯網資料應用

時間維度資料分析

中央研究院 資訊科學所 陳伶志

民生公共物聯網
Civil IoT Taiwan

大綱

1. 時間序列資料
2. 時間序列資料處理
3. 時間序列資料預測
4. 時間序列資料分群

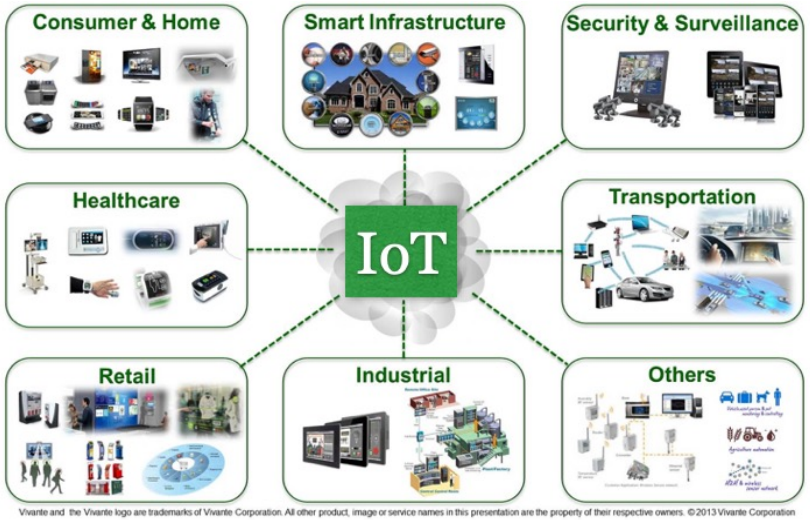
1. 時間序列資料

物聯網

物聯網

4.0 Bluetooth
Wi Fi
LoRa
SIGFOX
GSM
3G
4G


Artificial Intelligence
AI
Reasoning
Computer
Knowledge
Technology


Consumer & Home
Smart Infrastructure
Security & Surveillance
Healthcare
IoT
Transportation
Retail
Industrial
Others
Vivante and the Vivante logo are trademarks of Vivante Corporation. All other product, image or service names in this presentation are the property of their respective owners. ©2013 Vivante Corporation

民生公共物聯網資料

水資源

空氣品質

地震

防救災

民生物聯網資料集

- 空氣品質
- 水資源
- 地震活動
- 氣象
- CCTV
- 災害示警與災情通報

https://ci.taiwan.gov.tw/dsp/dataset_air.aspx

套件安裝與引用

```
!pip install --upgrade pip
# Kats
!pip install kats==0.1 ax-platform==0.2.3 statsmodels==0.12.2
# calplot
!pip install calplot

import warnings
import calplot
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import statsmodels.api as sm
import os, zipfile
```

```
from pyCIOT.data import *
from datetime import datetime, timedelta
from dateutil import parser as datetime_parser
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.tsa.seasonal import seasonal_decompose
from kats.detectors.outlier import OutlierDetector
from kats.detectors.cusum_detection import CUSUMDetector
from kats.consts import TimeSeriesData, TimeSeriesIterator
from IPython.core.pylabtools import figsize
```

讀取空氣品質歷史資料

- 下載「中研院校園空品微型感測器」歷史資料
- 因資料量過大，建議使用下載功能，不使用 pyCIOT

```
!mkdir Air CSV_Air
!wget -O Air/2018.zip -q
"https://history.colife.org.tw/?r=/download&path=L%2Bepuuawo%2BWTgeizqi%2FkuK3noJTpmaJf5qCh5ZyS56m65ZOB5b6u5Z6L5oSf5ris5ZmoLzIwMTguemlw"
!wget -O Air/2019.zip -q
"https://history.colife.org.tw/?r=/download&path=L%2Bepuuawo%2BWTgeizqi%2FkuK3noJTpmaJf5qCh5ZyS56m65ZOB5b6u5Z6L5oSf5ris5ZmoLzIwMTkuemlw"
!wget -O Air/2020.zip -q
"https://history.colife.org.tw/?r=/download&path=L%2Bepuuawo%2BWTgeizqi%2FkuK3noJTpmaJf5qCh5ZyS56m65ZOB5b6u5Z6L5oSf5ris5ZmoLzIwMjAuemlw"
!wget -O Air/2021.zip -q
"https://history.colife.org.tw/?r=/download&path=L%2Bepuuawo%2BWTgeizqi%2FkuK3noJTpmaJf5qCh5ZyS56m65ZOB5b6u5Z6L5oSf5ris5ZmoLzIwMjEuemlw"
```

讀取空氣品質歷史資料

- 由於所下載的資料是 zip 壓縮檔案的格式，我們需要先逐一將其解壓縮，產生每日資料的壓縮檔案。

```
#開始進行解壓縮
folder = 'Air'
extension_zip = '.zip'
extension_csv = '.csv'

for subfolder in os.listdir(folder):
    path = f'{folder}/{subfolder}'
    if path.endswith(extension_zip):
      print(path)
      zip_ref = zipfile.ZipFile(path)
      zip_ref.extractall(folder)
      zip_ref.close()
```

讀取空氣品質歷史資料

- 接著再將每日資料的壓縮檔案解壓縮，存入 CSV_Air 資料夾中。

```
for subfolder in os.listdir(folder):
    path = f'{folder}/{subfolder}'
    if os.path.isdir(path):
        for item in os.listdir(path):
            if item.endswith(extension_zip):
                file_name = f'{path}/{item}'
                zip_ref = zipfile.ZipFile(file_name)
                zip_ref.extractall(path)
                zip_ref.close()

        for item in os.listdir(path):
          path2 = f'{path}/{item}'
          if os.path.isdir(path2):
            for it in os.listdir(path2):
              if it.endswith(extension_zip):
                file_name = f'{path2}/{it}'
                zip_ref = zipfile.ZipFile(file_name)
                zip_ref.extractall('CSV_Air') # decide path
                zip_ref.close()
          elif item.endswith(extension_csv):
            os.rename(path2, f'CSV_Air/{item}')
```

讀取空氣品質歷史資料

- 讀取每個 csv 檔案，將 74DA38C7D2AC 測站的資料過濾出來存入名叫 air 的 dataframe 中。
- 將所有下載的資料與解壓縮後產生的資料移除，以節省雲端的儲存空間。

```
folder = 'CSV_Air'
extension_csv = '.csv'
id = '74DA38C7D2AC'

air = pd.DataFrame()
for item in os.listdir(folder):
  file_name = f'{folder}/{item}'
  df = pd.read_csv(file_name)
  if 'pm25' in list(df.columns):
    df.rename({'pm25':'PM25'}, axis=1, inplace=True)
  filtered = df.query(f'device_id==@id')
  air = pd.concat([air, filtered], ignore_index=True)
air.dropna(subset=['timestamp'], inplace=True)

for i, row in air.iterrows():
  aware = datetime_parser.parse(str(row['timestamp']))
  naive = aware.replace(tzinfo=None)
  air.at[i, 'timestamp'] = naive
air.set_index('timestamp', inplace=True)

!rm -rf Air CSV_Air
```

讀取空氣品質歷史資料

- 重新整理該測站的資料，將不需要用到的欄位資訊刪除，並且依照時間進行排序

```
air.drop(columns=['device_id', 'SiteName'], inplace=True)
air.sort_values(by='timestamp', inplace=True)
air.info()
print(air.head())
```

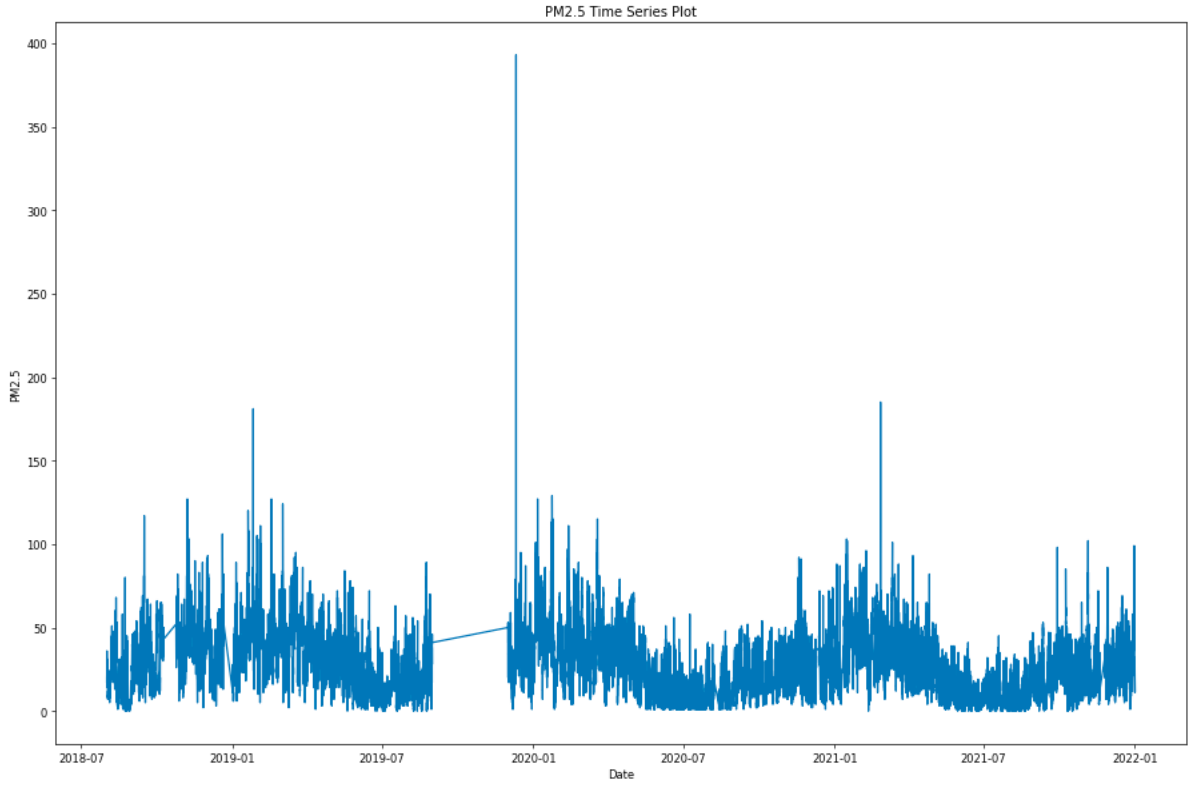
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 195305 entries, 2018-08-01 00:00:05 to 2021-12-31 23:54:46
Data columns (total 1 columns):
 \#   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   PM25    195305 non-null  object
dtypes: object(1)
memory usage: 3.0+ MB
                     PM25
timestamp
2018-08-01 00:00:05  20.0
2018-08-01 00:30:18  17.0
2018-08-01 01:12:34  18.0
2018-08-01 01:18:36  21.0
2018-08-01 01:30:44  22.0
```

2. 時間序列資料處理

- 使用作圖工具觀察時序資料
- 檢測與處理時序資料
- 分解時序資料得到趨勢與週期性

資料視覺化

```
plt.figure(figsize=(15, 10), dpi=60)
plt.plot(air[:]["PM25"])

plt.xlabel("Date")
plt.ylabel("PM2.5")
plt.title("PM2.5 Time Series Plot")

plt.tight_layout()

plt.show()
```



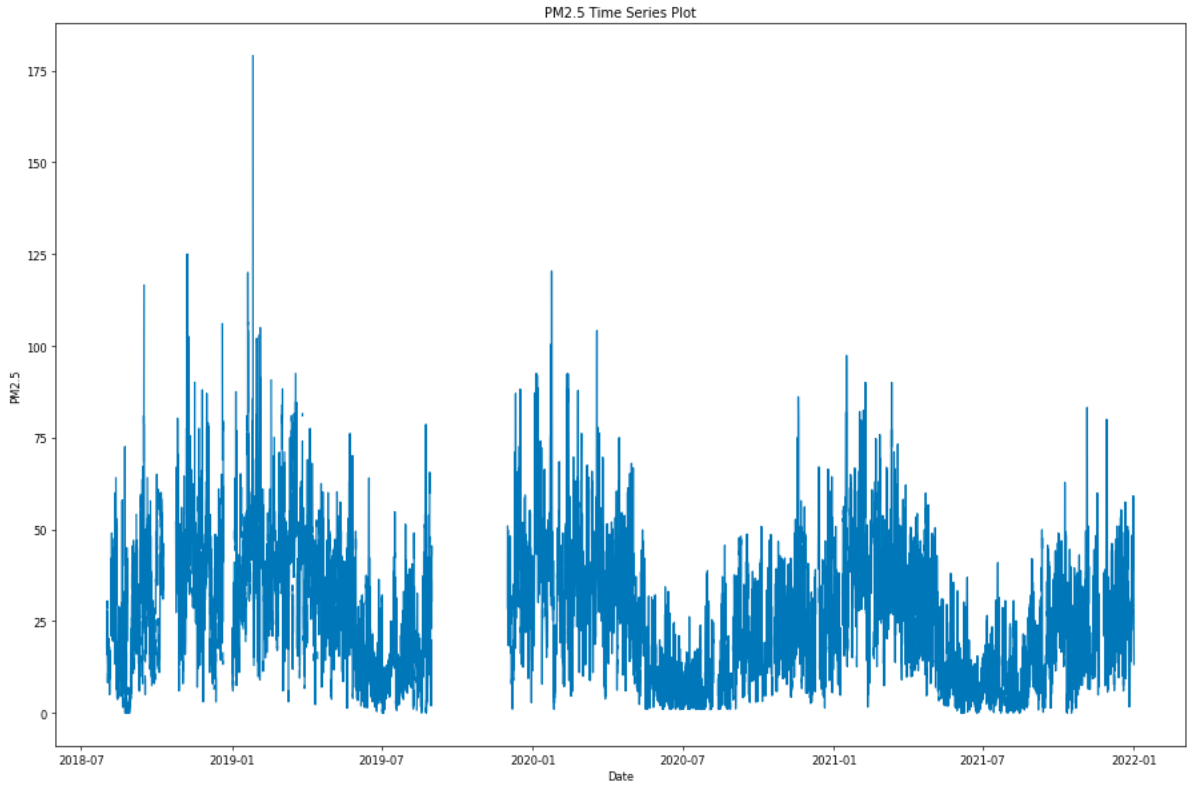
重新採樣

- 原始資料的採樣頻率約略是每五分鐘一筆
- 微環境採樣過於頻繁，反而容易產生劇烈震盪，不易呈現環境空品的整體變化趨勢
- 以較大尺度 (小時、日、月) 的採樣率重新取樣

```
#每小時的平均
air_hour = air.resample('H').mean()

#每日的平均
air_day = air.resample('D').mean()

#每月的平均
air_month = air.resample('M').mean()
```

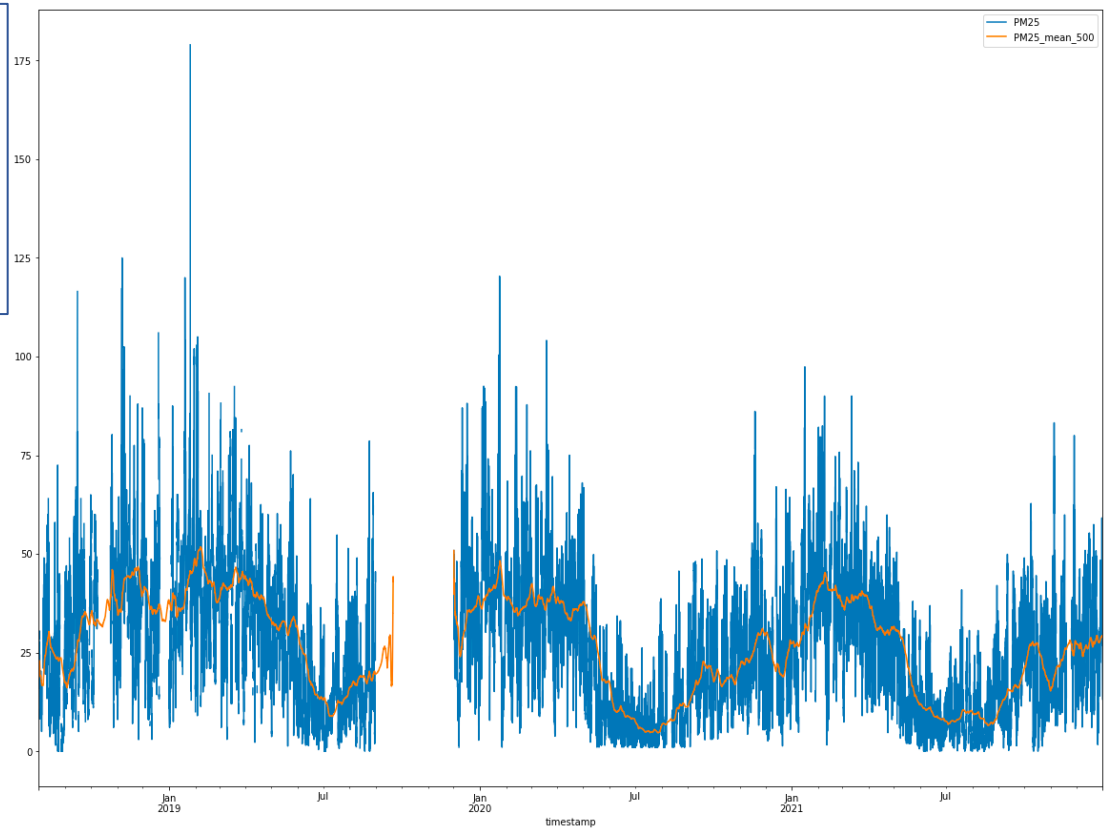
每小時重新採樣的資料視覺化

```
plt.figure(figsize=(15, 10), dpi=60)
plt.plot(air_hour[:]["PM25"])

plt.xlabel("Date")
plt.ylabel("PM2.5")
plt.title("PM2.5 Time Series Plot")

plt.tight_layout()

plt.show()
```



移動平均

```
# plt.figure(figsize=(15, 10), dpi=60)
MA = air_hour
MA10 = MA.rolling(window=500, min_periods=1).mean()

MA.join(MA10.add_suffix('_mean_500')).plot(figsize=(20, 15))
```

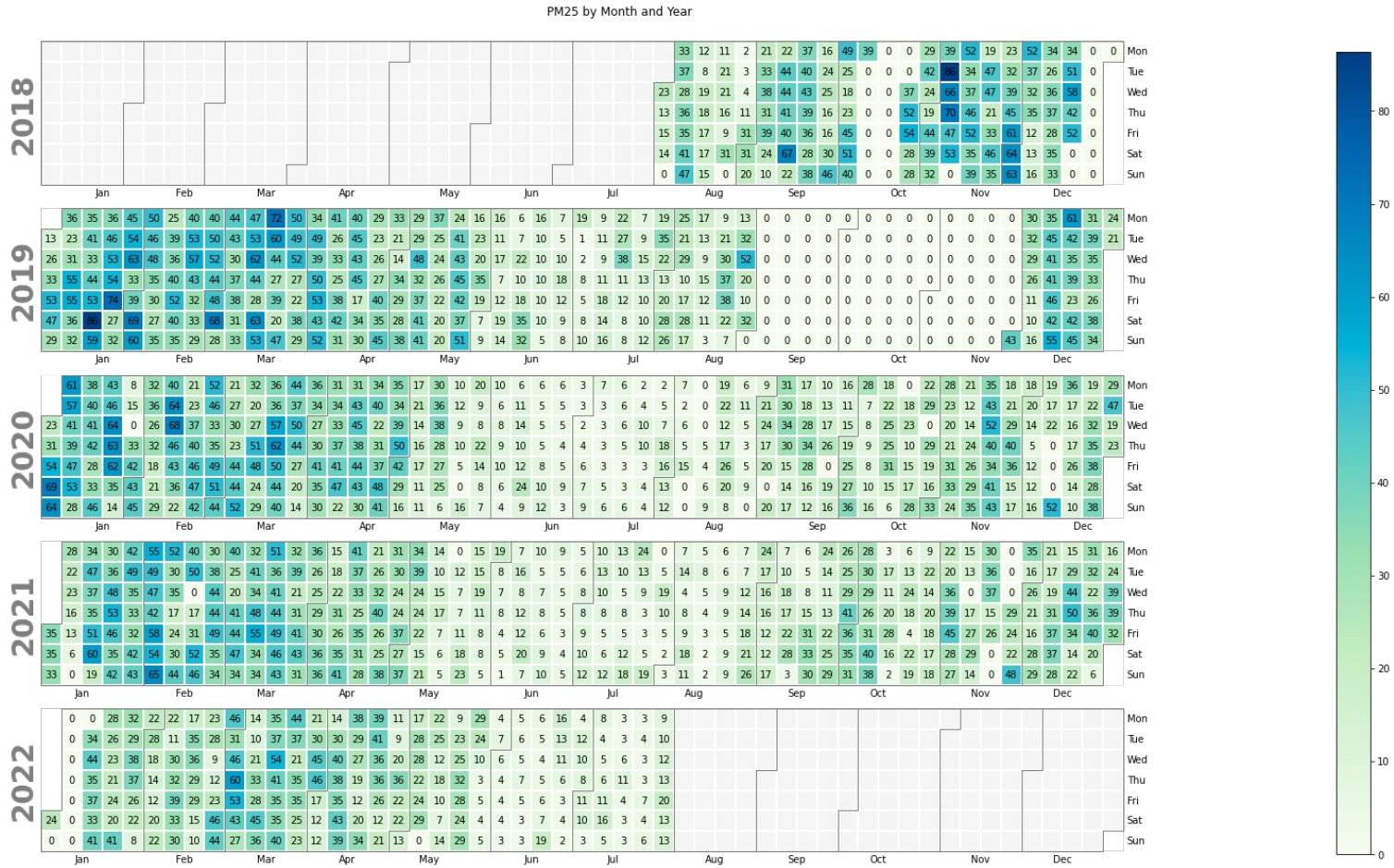


多曲線圖

```
air_month.reset_index(inplace=True)
air_month['year'] = [d.year for d in air_month.timestamp]
air_month['month'] = [d.strftime('%b') for d in air_month.timestamp]
years = air_month['year'].unique()

np.random.seed(100)
mycolors = np.random.choice(list(mpl.colors.XKCD_COLORS.keys()),
len(years), replace=False)

plt.figure(figsize=(15, 10), dpi=60)
for i, y in enumerate(years):
  if i > 0:
    plt.plot('month', 'PM25', data=air_month.loc[air_month.year==y, :],
color=mycolors[i], label=y)
    plt.text(air_month.loc[air_month.year==y, :].shape[0]-.9,
air_month.loc[air_month.year==y, 'PM25'][-1:].values[0], y,
fontsize=12, color=mycolors[i])

plt.show()
```

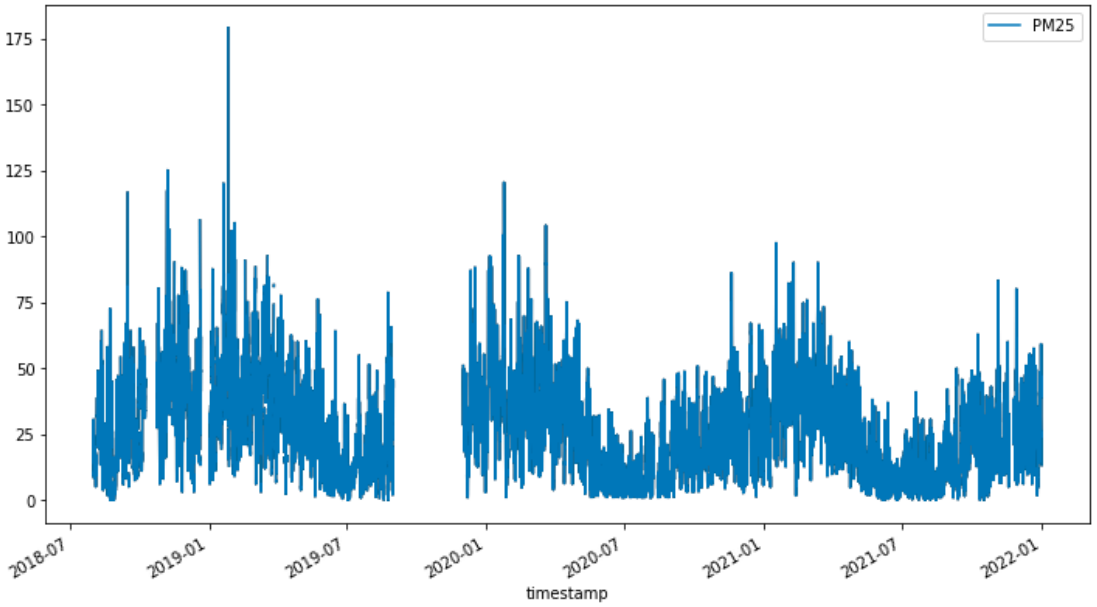
日曆熱力圖

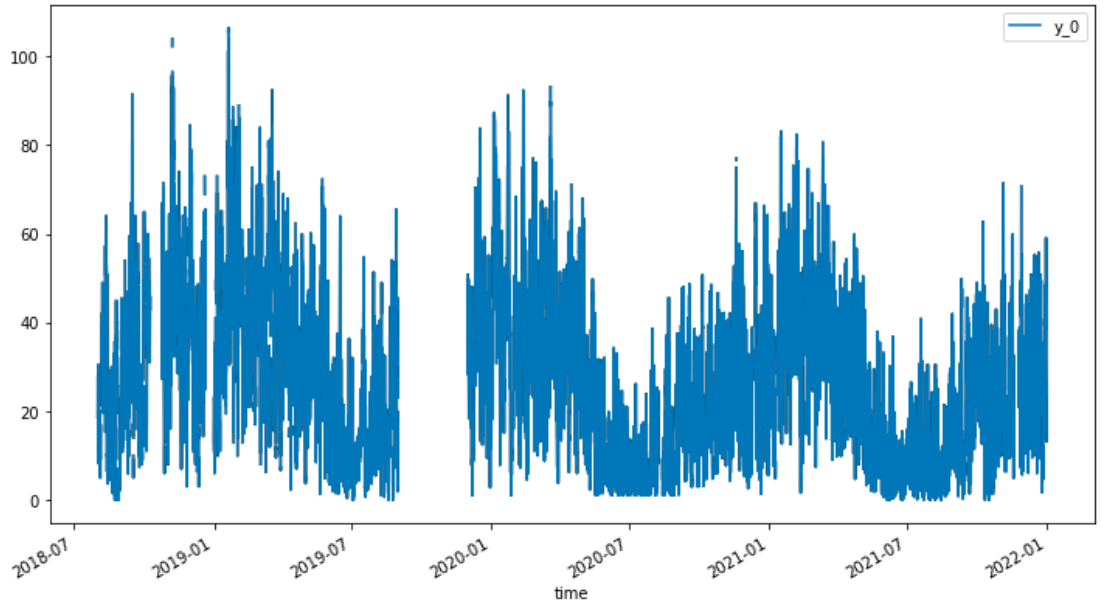
```
# cmap: 設定呈現的顏色色盤
(https://matplotlib.org/stable/gallery/color/colormap_reference.html)
# textformat: 設定圖中數字呈現的樣式
pl1 = calplot.calplot(data = air_day['PM25'], cmap = 'GnBu', textformat = '{:.0f}',
                      figsize = (24, 12), suptitle = "PM25 by Month and Year")
```



離群值 (Outliers) 偵測與移除

```
air_ts = TimeSeriesData(air_hour.reset_index(), time_col_name='timestamp’ )
outlierDetection = OutlierDetector(air_ts, 'additive')
outlierDetection.detector()

outliers_removed = outlierDetection.remover(interpolate=False)
outliersremoved.plot(cols=['y_0’])
```





改變點 (Change Point) 偵測

- 改變點是資料中突然發生重大改變的時間點，代表的是事件的發生、資料狀態的轉變或資料分布的轉變。

```
air_ts = TimeSeriesData(air_day.reset_index(), time_col_name='timestamp')
detector = CUSUMDetector(air_ts)

change_points = detector.detector(change_directions=["increase", "decrease"])
# print("The change point is on", change_points[0][0].start_time)

# plot the results
plt.xticks(rotation=45)
detector.plot(change_points)
plt.show()
```

缺失值 (Missing Data) 處理

1. 標示為 NaN
2. Forward fill 法：用前一個數值來填補當下的缺失值
3. K-Nearest Neighbor (KNN) 法：尋找距離缺失值最近的 k 個數值進行平均，並用來填補這個缺失值

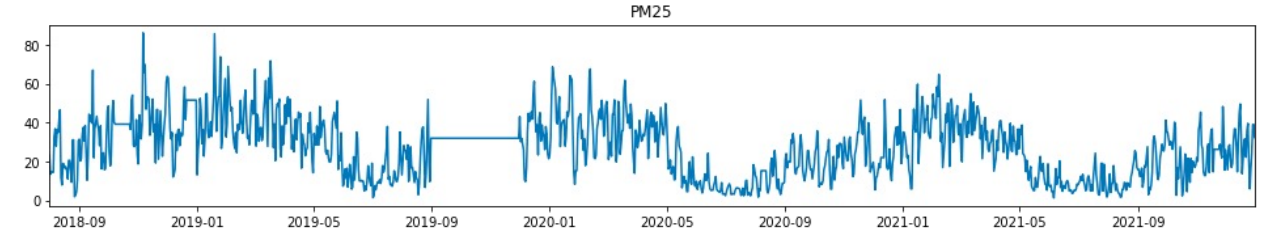
```
def knn_mean(ts, n):
    out = np.copy(ts)
    for i, val in enumerate(ts):
        if np.isnan(val):
            n_by_2 = np.ceil(n/2)
            lower = np.max([0, int(i-n_by_2)])
            upper = np.min([len(ts)+1, int(i+n_by_2)])
            ts_near = np.concatenate([ts[lower:i], ts[i:upper]])
            out[i] = np.nanmean(ts_near)
    return out
```

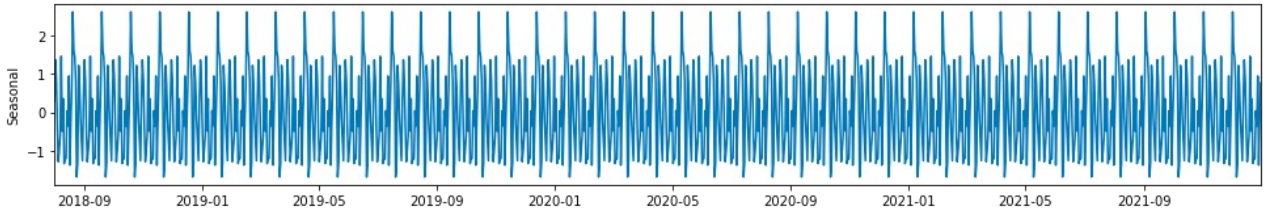
```
# forward fill
df_ffill = air.ffill(inplace=False)
df_ffill.plot()

# KNN
df_knn = df.copy()
df_knn['PM25'] = knn_mean(air.PM25.to_numpy(), 5000)
df_knn.plot()
```

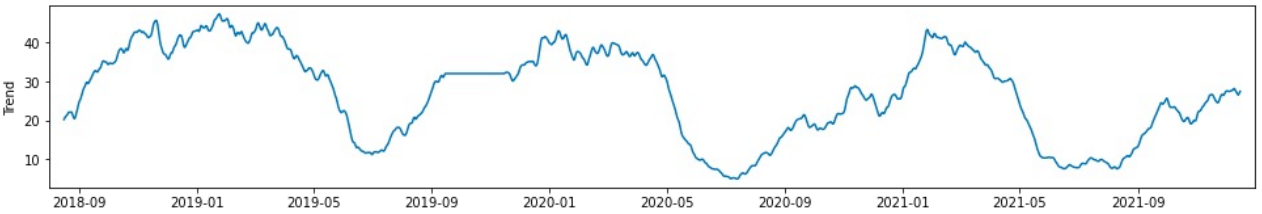
資料分解 (Decomposition)

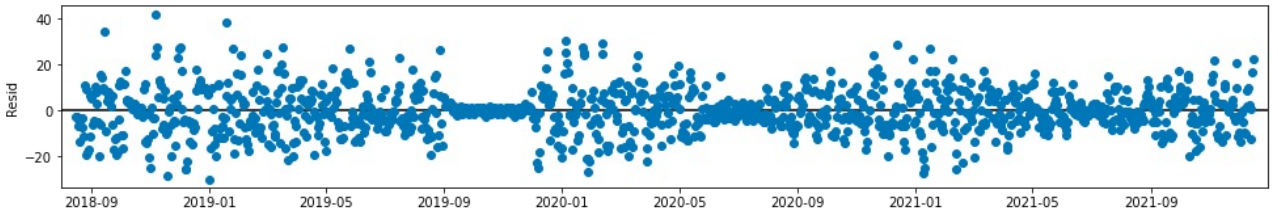
- 拆解的週期設定為 30 天
- 執行後便會依序產出四張圖：原始資料、趨勢圖、週期性圖與殘差圖

```
air_process = air_day.copy()
# new.round(1).head(12)
air_process.ffill(inplace=True)

decompose = seasonal_decompose(air_process['PM25’ ],
                               model='additive', period=30)
decompose.plot().set_size_inches((15, 15))
plt.show()
```





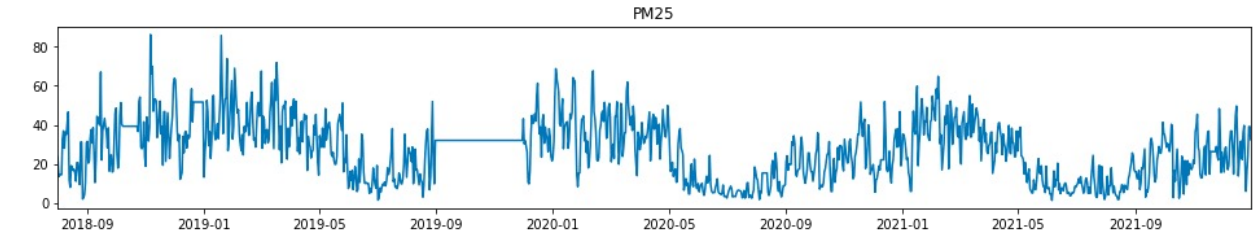


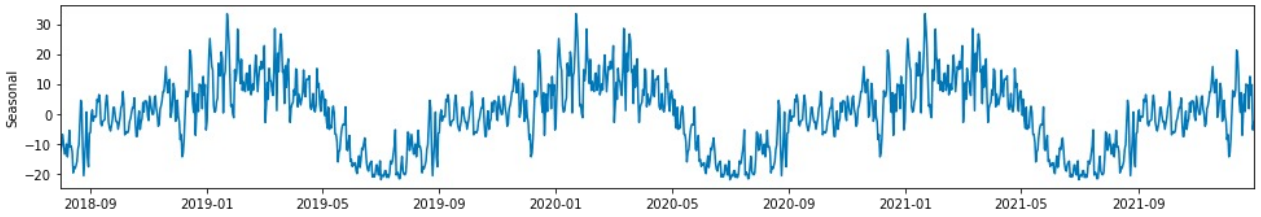


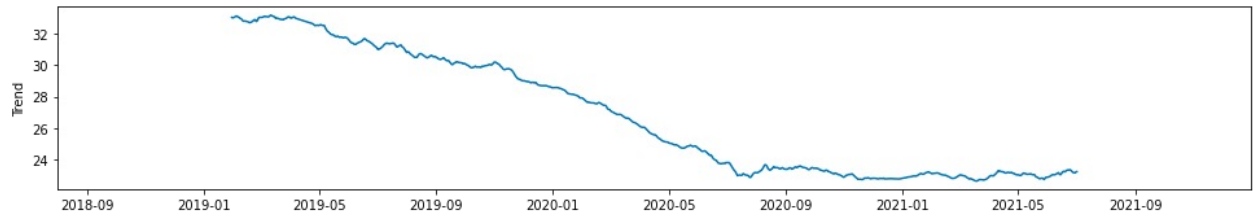
資料分解 (Decomposition)

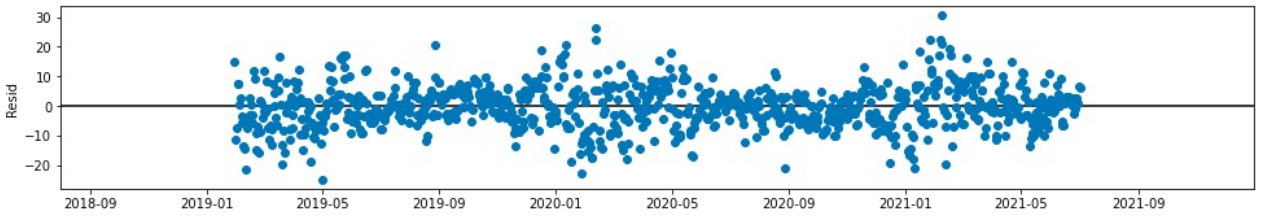
- 將 period 變數改為 365，以較大的時間尺度 (一年) 來進行資料分解

```
decompose = seasonal_decompose(air_process['PM25'],model='additive', period=365)
decompose.plot().set_size_inches((15, 15))
plt.show()
```









3. 時間序列資料預測

- 時序資料特性的判斷：平穩性
- 學習各種預測模型並進行比較
- 使用時序資料進行訓練與預測

資料準備

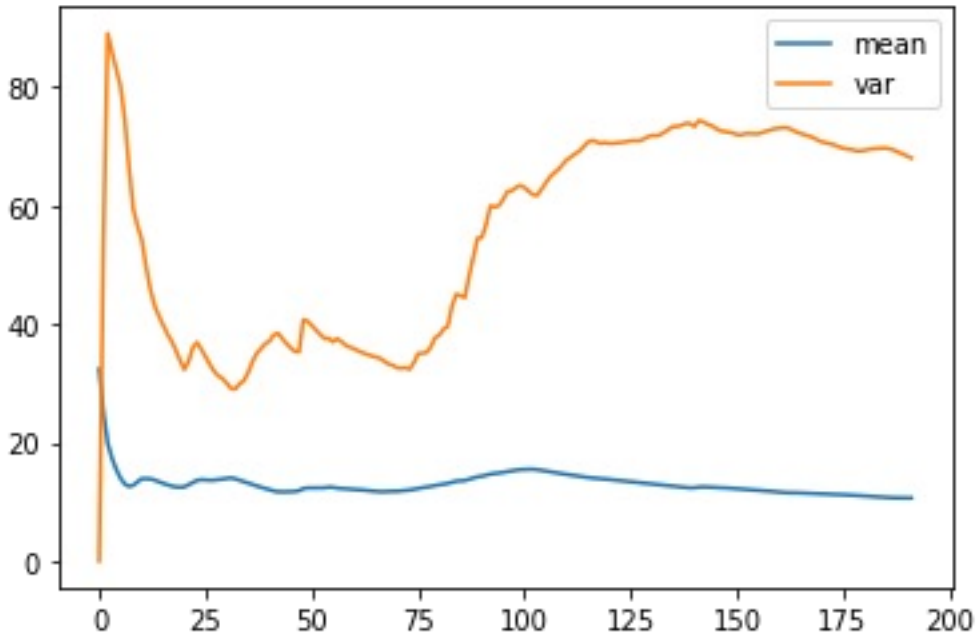
- 重新準備接下來要使用的空品資料
- 首先將原始資料的離群值刪除
- 接著僅留下需要的欄位資料
- 重新取樣成為每小時一筆資料
- 用 Forward Fill 方法進行缺失值填值

```
air_hour = air.resample('H').mean()
air_day = air.resample('D').mean()
air_month = air.resample('M').mean()

air_ts = TimeSeriesData(air_hour.reset_index(),
time_col_name='timestamp')

# 移除離群值
outlierDetection = OutlierDetector(air_ts, 'additive')
outlierDetection.detector()
outliers_removed = outlierDetection.remover(interpolate=False)

air_hour_df = outliers_removed.to_dataframe()
air_hour_df.rename(columns={'time': 'timestamp', 'y_0':
'PM25'}, inplace=True)
air_hour_df.set_index('timestamp', inplace=True)
air_hour = air_hour_df
air_hour = air_hour.resample('H').mean()

# 用 Forward 方法填回缺失值
air_hour.ffill(inplace=True)
```

平穩性 (Stationary) 檢查

```
data = air_hour.loc['2020-06-10':'2020-06-17']
nmp = data.PM25.to_numpy()
size = np.size(nmp)

nmp_mean = np.zeros(size)
nmp_var = np.zeros(size)
for i in range(size):
  nmp_mean[i] = nmp[:i+1].mean()
  nmp_var[i] = nmp[:i+1].var()

y1 = nmp_mean[:]
y2 = nmp_var[:]
y3 = nmp
x = np.arange(size)
plt.plot(x, y1, label='mean')
plt.plot(x, y2, label='var')
plt.legend()
plt.show()
```

- 資料的平均數變化不大，但變異數的變化很大 -> 平穩性差



平穩性 (Stationary) 檢查

- Augmented Dickey Fuller (ADF) test
  - 使用 unit root test
  - 如果 p-value < 0.05，則平穩性高。

```
# ADF Test
result = adfuller(data.PM25.values, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: -2.70261940885417O4
p-value: 0.07358609270498144
Critial Values:   1%, -3.4654311561944873
Critial Values:   5%, -2.8769570530458792
Critial Values:   10%, -2.574988319755886
```

- Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test
  - 如果 p-value <0.05，則平穩性差。

```
# KPSS Test
result = kpss(data.PM25.values, regression='c')
print('\nKPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[3].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

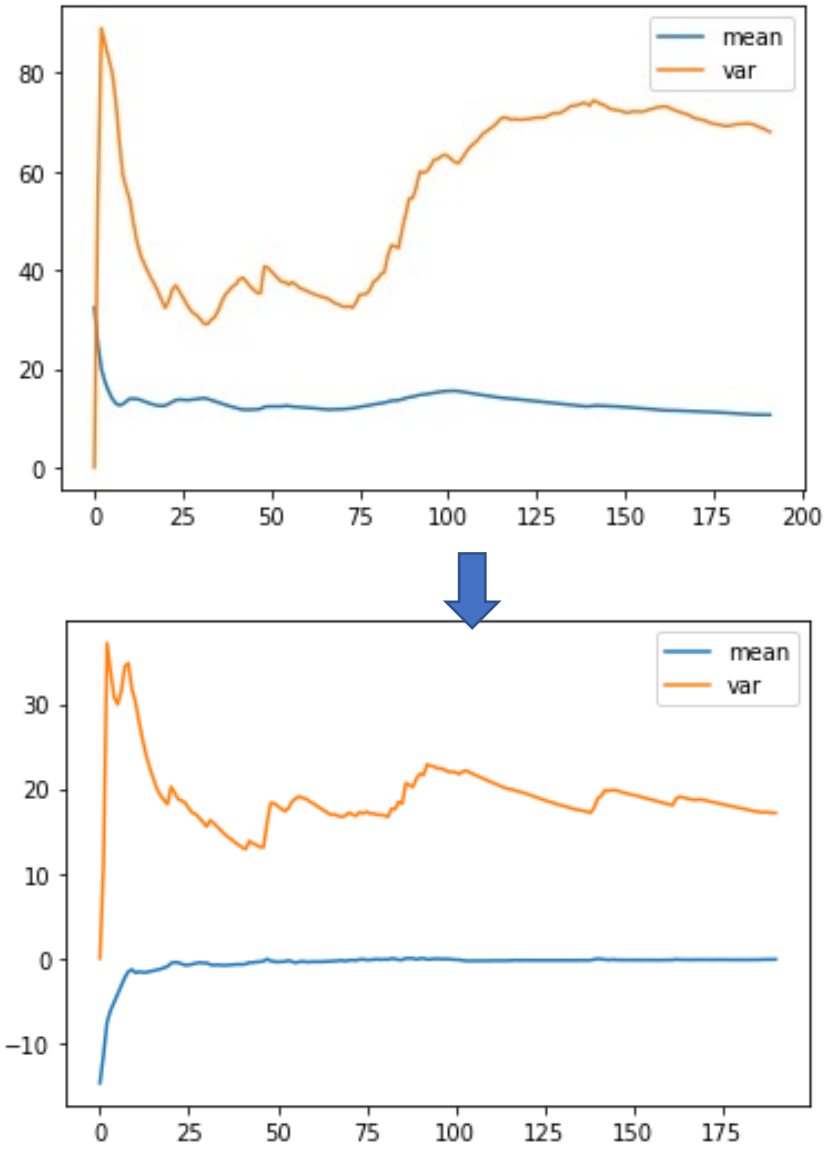
```
KPSS Statistic: 0.620177
p-value: 0.020802
Critial Values:   10%, 0.347
Critial Values:   5%, 0.463
Critial Values:   2.5%, 0.574
Critial Values:   1%, 0.739
```

平穩性 (Stationary) 檢查

- 針對平穩性差的資料，我們使用差分來改進
- 若第一筆資料為NaN，需直接捨棄

```
data_diff = data.diff()
data_diff = data_diff[1:]

nmp = data_diff.PM25.to_numpy()
size = np.size(nmp)

nmp_mean = np.zeros(size)
nmp_var = np.zeros(size)
for i in range(size):
  nmp_mean[i] = nmp[:i+1].mean()
  nmp_var[i] = nmp[:i+1].var()
```

```
y1 = nmp_mean[:]
y2 = nmp_var[:]
y3 = nmp
x = np.arange(size)
plt.plot(x, y1, label='mean')
plt.plot(x, y2, label='var')
plt.legend()
plt.show()
```



平穩性 (Stationary) 檢查

- Augmented Dickey Fuller (ADF) test

```
# ADF Test
result = adfuller(data_diff.PM25.values, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: -13.350457196046884
p-value: 5.682260865619701e-25
Critial Values:   1%, -3.4654311561944873
Critial Values:   5%, -2.8769570530458792
Critial Values:   10%, -2.574988319755886
```

- Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test

```
# KPSS Test
result = kpss(data_diff.PM25.values, regression='c')
print('\nKPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[3].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
KPSS Statistic: 0.114105
p-value: 0.100000
Critial Values:   10%, 0.347
Critial Values:   5%, 0.463
Critial Values:   2.5%, 0.574
Critial Values:   1%, 0.739
```

由此可知，我們所處理的資料，在經過一次差分後便具有平穩性！

資料預測

- 我們依序示範下列模型
  - ARIMA
  - SARIMAX
  - auto_arima
  - Prophet
  - LSTM
  - Holt-Winter

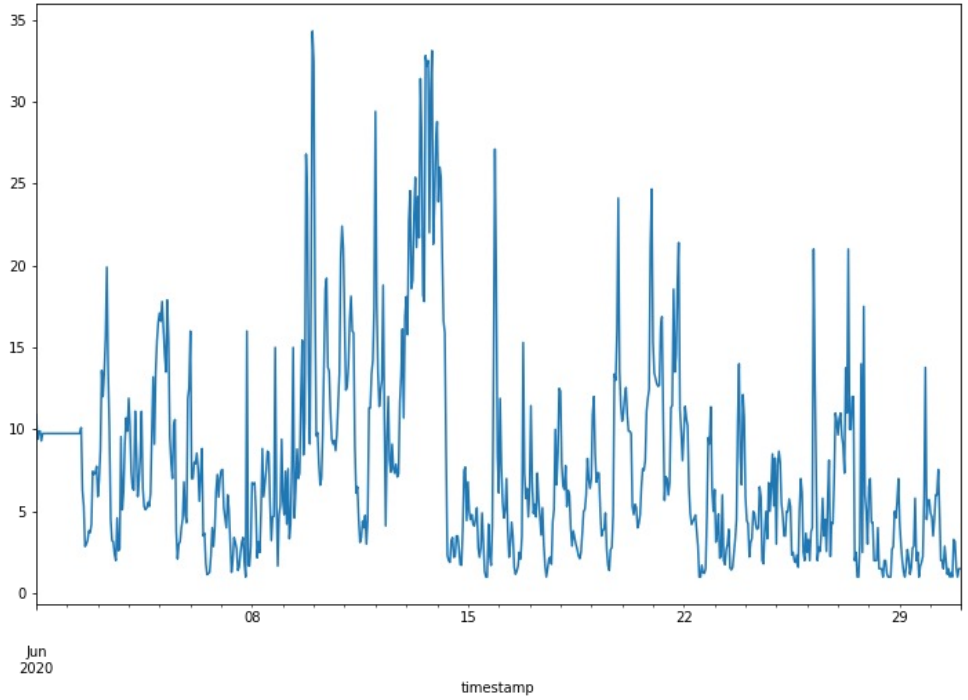
套件安裝與引用

```
!pip install --upgrade pip
!pip install kats==0.1 ax-platform==0.2.3 statsmodels==0.12.2
!pip install pmdarima

import numpy as np
import pandas as pd
import pmdarima as pm
import statsmodels.api as sm
import matplotlib.pyplot as plt
import os, zipfile
from pyCIOT.data import *
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller, kpss
from kats.consts import TimeSeriesData, TimeSeriesIterator
from kats.detectors.outlier import OutlierDetector
from kats.models.prophet import ProphetModel, ProphetParams
from kats.models.lstm import LSTMModel, LSTMParams
from kats.models.holtwinters import HoltWintersParams, HoltWintersModel
```

資料預測

- 我們選用 2020-06-01 到 2020-06-30 的資料
- 我們將資料分為
  - 訓練資料：用於建構預測模型
  - 測試資料：用於測試模型效能
- 我們將倒數 48 小時的資料做為測試資料，其餘為訓練資料

```
air_hour.loc['2020-06-01':'2020-06-30']['PM25'].plot(figsize=(12, 8))
```



資料預測：ARIMA

- ARIMA 模型其實是 ARMA 模型的擴展版本：
  - 自迴歸模型 (AR, autogressive model)：使用一個參數 p，並以前 p 個歷史值做線性組合來預測當下的數值。
  - 移動平均模型 (MA, moving average model)：使用一個參數 q，並以前 q 個使用 AR 模型的預測誤差進行線性組合，以預測當下的數值。
  - 而 ARIMA 模型比 ARMA 模型還多使用一個參數 d，代表資料需要進行 d 次差分才能擁有高平穩性。

```
data_arima = air_hour.loc['2020-06-17':'2020-06-21']
train_len = -48
train = data_arima.iloc[:train_len]
test = data_arima.iloc[train_len:]

# Run Dicky-Fuller test
result = adfuller(train)

# Print test statistic
print('The test stastics:', result[0])
# Print p-value
print("The p-value:",result[1])
```

```
The test stastics: -3.1129543556288826
The p-value: 0.02560924361534107 4
```

由此可知，我們所處理的資料，無需差分便具有平穩性！
因此 d = 0。

資料預測：ARIMA

我們可以使用 AIC 或 BIC 方法，先將 p 和 q 的範圍限制在 0~2 之間，這樣總共有 9 種可能的組合，再分別查看其 AIC 與 BIC 的數值，並以數值最小的 p 和 q 組合，作為這兩個參數的決定值。

```
order_aic_bic =[]

for p in range(3):
    for q in range(3):
        try:
            # create and fit ARMA(p,q) model
            model = sm.tsa.statespace.SARIMAX(train['PM25'], order=(p, 0, q))
            results = model.fit()

            order_aic_bic.append((p, q, results.aic, results.bic))
        except:
            print(p, q, None, None)

# Make DataFrame of model order and AIC/BIC scores
order_df = pd.DataFrame(order_aic_bic, columns=['p', 'q', 'aic','bic'])

# Sort by AIC
print("Sorted by AIC ")
print(order_df.sort_values('aic').reset_index(drop=True))

# Sort by BIC
print("Sorted by BIC ")
print(order_df.sort_values('bic').reset_index(drop=True))
```

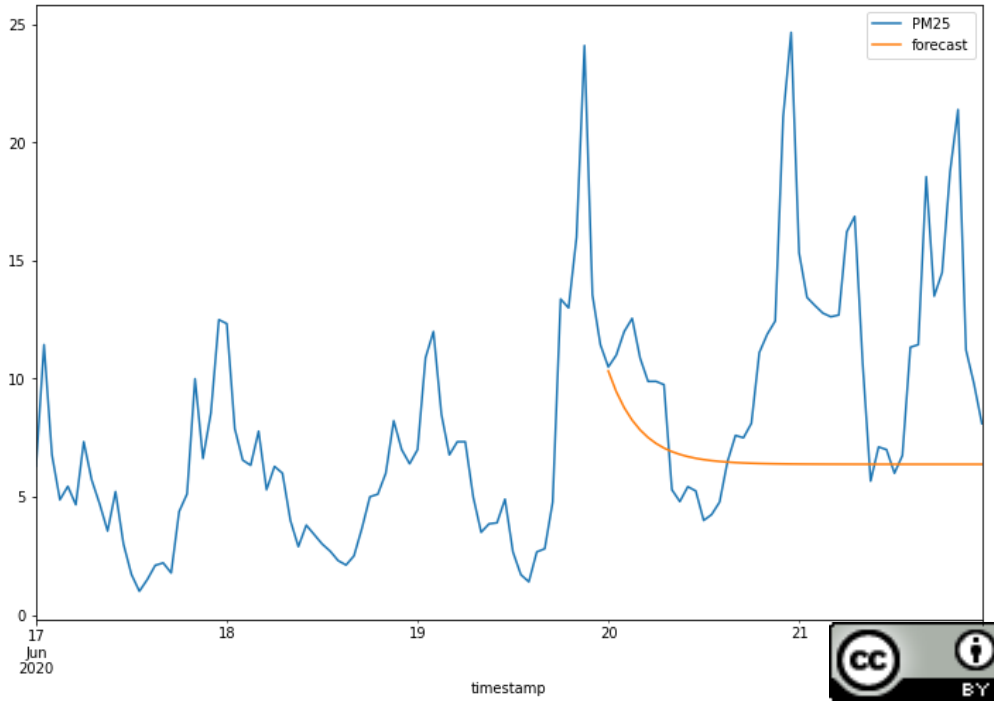
```
Sorted by AIC
   p  q       aic       bic
0  1  0 349.493661  354.046993
1  1  1 351.245734  358.075732
2  2  0 351.299268  358.129267
3  1  2 352.357930  361.464594
4  2  1 353.015921  362.122586
5  2  2 353.063243  364.446574
6  0  2 402.213407  409.043405
7  0  1 427.433962  431.987294
8  0  0 493.148188  495.424854
Sorted by BIC
   p  q       aic       bic
0  1  0 349.493661  354.046993
1  1  1 351.245734  358.075732
2  2  0 351.299268  358.129267
3  1  2 352.357930  361.464594
4  2  1 353.015921  362.122586
5  2  2 353.063243  364.446574
6  0  2 402.213407  409.043405
7  0  1 427.433962  431.987294
8  0  0 493.148188  495.424854
```

由此可知，(p,q) = (1,0) 時，AIC 和 BIC 的值最小，代表這是最好的模型組態，因此 (p,d,q) = (1,0,0)

資料預測：ARIMA

- ARIMA 模型其實是 ARMA 模型的擴展版本：
  - 自迴歸模型 (AR, autogressive model)：使用一個參數 p，並以前 p 個歷史值做線性組合來預測當下的數值。
  - 移動平均模型 (MA, moving average model)：使用一個參數 q，並以前 q 個使用 AR 模型的預測誤差進行線性組合，以預測當下的數值。
  - 而 ARIMA 模型比 ARMA 模型還多使用一個參數 d，代表資料需要進行 d 次差分才能擁有高平穩性。

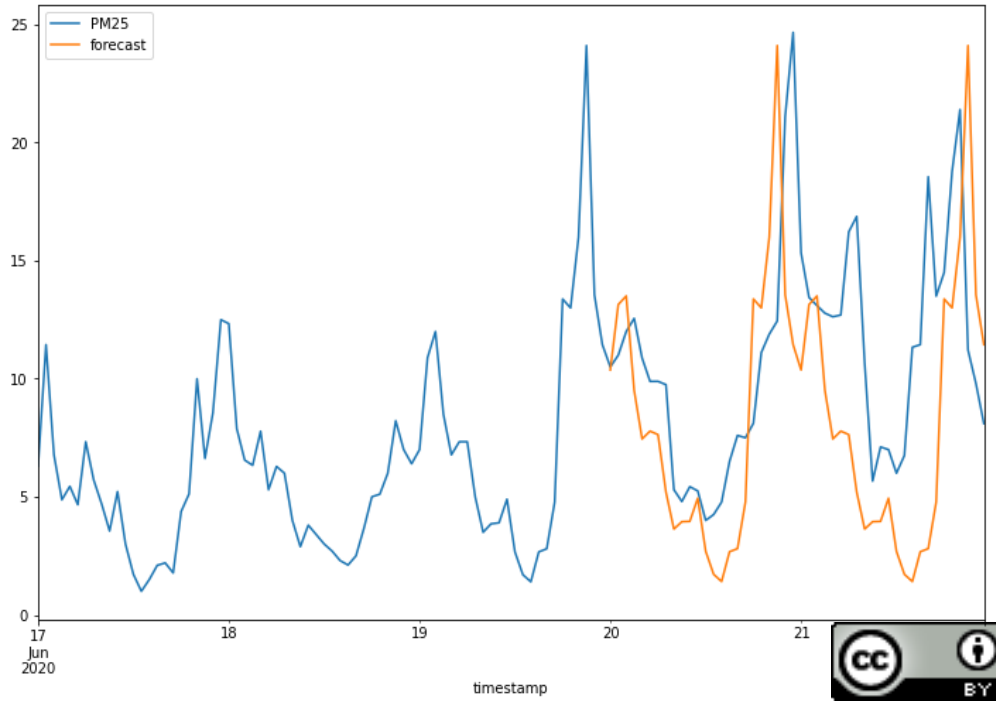
```
# Instantiate model object
model = ARIMA(train, order=(1,0,0))

# Fit model
results = model.fit()

data_arima['forecast'] = results.predict(start=24*5-48, end=24*5)
data_arima[['PM25', 'forecast']].plot(figsize=(12, 8))
```



資料預測：SARIMAX

- SARIMAX 模型共有七個參數，分別是 p, d, q, P, D, Q, s
- 這些參數可以分為兩組：第一組為 order=(p, d, q) 這三個參數跟 ARIMA 模型的參數一樣；另一組是 seasonal_order=(P, D, Q, s)，也就是週期性的 AR 模型參數、週期性的差分次數、週期性的 MA 模型參數，最後再加上一個週期性長度的參數（我們在範例中設為 24）。

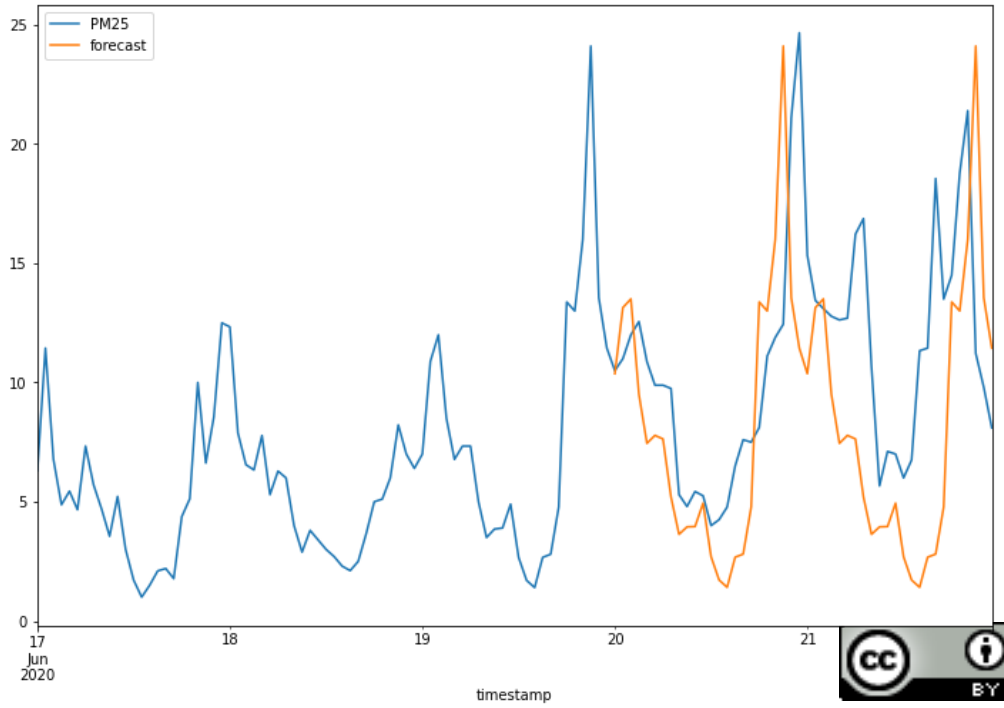
```
data_sarimax = air_hour.loc['2020-06-17':'2020-06-21']
train_len = -48
train = data_sarimax.iloc[:train_len]
test = data_sarimax.iloc[train_len:]

# Instantiate model object
model = SARIMAX(train, order=(1,0,0), seasonal_order=(0, 1, 0, 24))

# Fit model
results = model.fit()

data_sarimax['forecast'] = results.predict(start=24*5-48, end=24*5)
data_sarimax[['PM25', 'forecast']].plot(figsize=(12, 8))
```



資料預測：auto_arima

- 使用 pmdarima 這個 Python 套件，自動尋找最合適的 ARIMA 模型參數

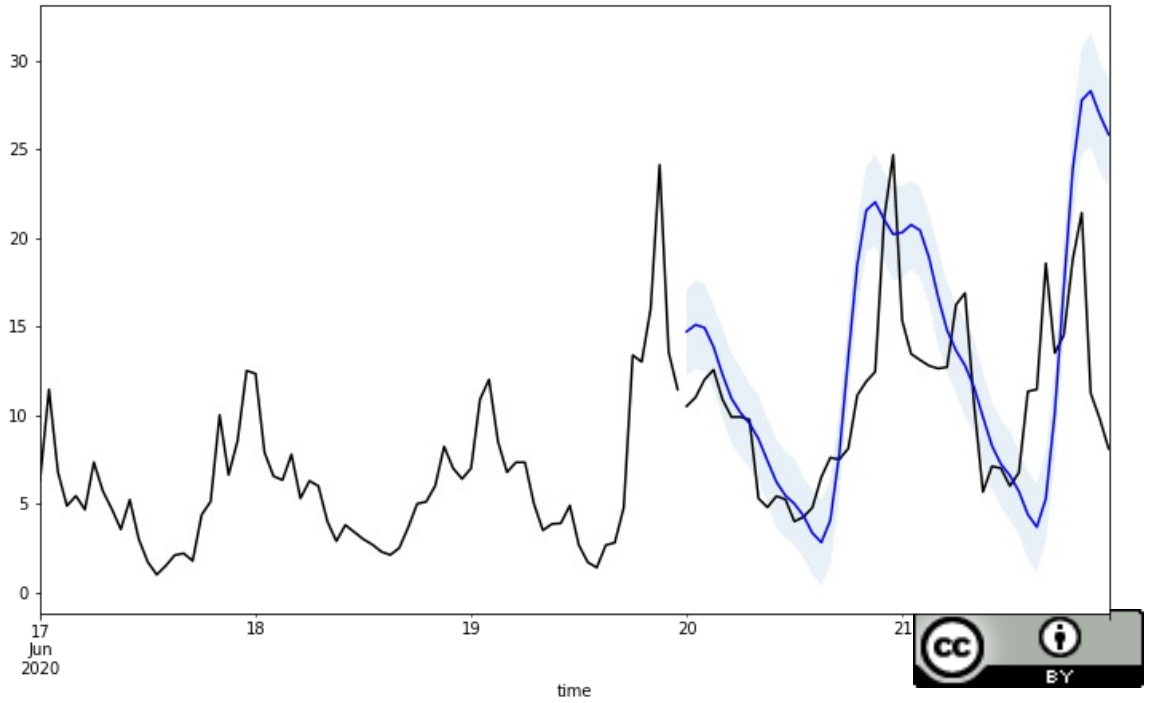
```
data_autoarima = air_hour.loc['2020-06-17':'2020-06-21']
train_len = -48
train = data_autoarima.iloc[:train_len]
test = data_autoarima.iloc[train_len:]

results = pm.auto_arima(train,start_p=0, d=0, start_q=0, max_p=5,
max_d=5, max_q=5, start_P=0, D=1, start_Q=0, max_P=5, max_D=5,
max_Q=5, m=24, seasonal=True, error_action='warn', trace = True,
supress_warnings=True, stepwise = True, random_state=20, n_fits = 20)

data_autoarima['forecast']= pd.DataFrame(results.predict(n_periods=48),
index=test.index)
data_autoarima[['PM25', 'forecast']].plot(figsize=(12, 8))
```



資料預測：Prophet

```
data_prophet = air_hour.loc['2020-06-17':'2020-06-21']
train_len = -48
train = data_prophet.iloc[:train_len]
test = data_prophet.iloc[train_len:]

trainData = TimeSeriesData(train.reset_index(), time_col_name='timestamp’)

# Specify parameters
params = ProphetParams(seasonality_mode="multiplicative")

# Create a model instance and fit model
m = ProphetModel(trainData, params)
m.fit()

# Forecast
fcst = m.predict(steps=48, freq="H")
data_prophet['forecast'] = fcst[['time','fcst']].set_index('time')

fig, ax = plt.subplots(figsize=(12, 7))
train.plot(ax=ax, label='train', color='black')
test.plot(ax=ax, color='black')
fcst.plot(x='time', y='fcst', ax=ax, color='blue')

ax.fill_between(test.index, fcst['fcst_lower'], fcst['fcst_upper'], alpha=0.1)
ax.get_legend().remove()
```

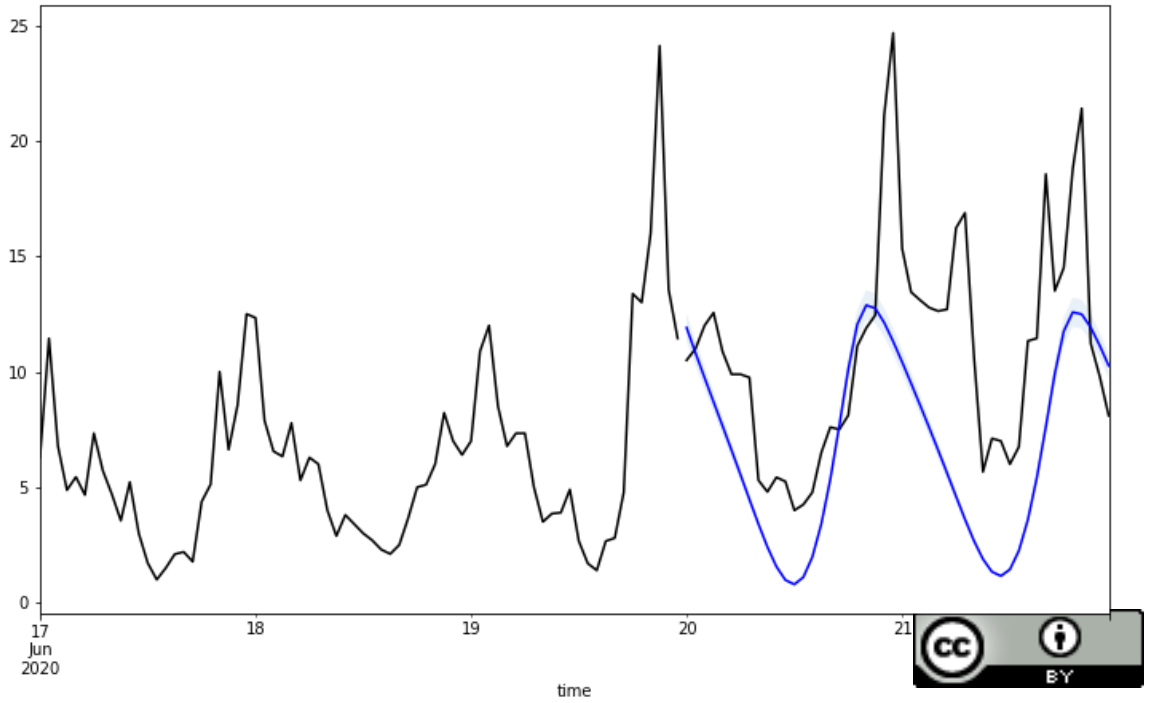
這個模型是由 Facebook 的資料科學團隊提出，擅長針對週期性強的時間序列資料進行預測，並且可以容忍缺失資料 (missing data)、資料偏移 (shift) 以及偏離值 (outlier)。



資料預測：LSTM

```
data_lstm = air_hour.loc['2020-06-17':'2020-06-21']
train_len = -48
train = data_lstm.iloc[:train_len]
test = data_lstm.iloc[train_len:]

trainData = TimeSeriesData(train.reset_index(), time_col_name='timestamp’)

params = LSTMParams(hidden_size=10, time_window=24, num_epochs=30)
m = LSTMModel(trainData, params)
m.fit()

fcst = m.predict(steps=48, freq="H")
data_lstm['forecast'] = fcst[['time', 'fcst']].set_index('time’)

fig, ax = plt.subplots(figsize=(12, 7))

train.plot(ax=ax, label='train', color='black')
test.plot(ax=ax, color='black')
fcst.plot(x='time', y='fcst', ax=ax, color='blue')

ax.fill_between(test.index, fcst['fcst_lower'], fcst['fcst_upper'], alpha=0.1)
ax.get_legend().remove()
```

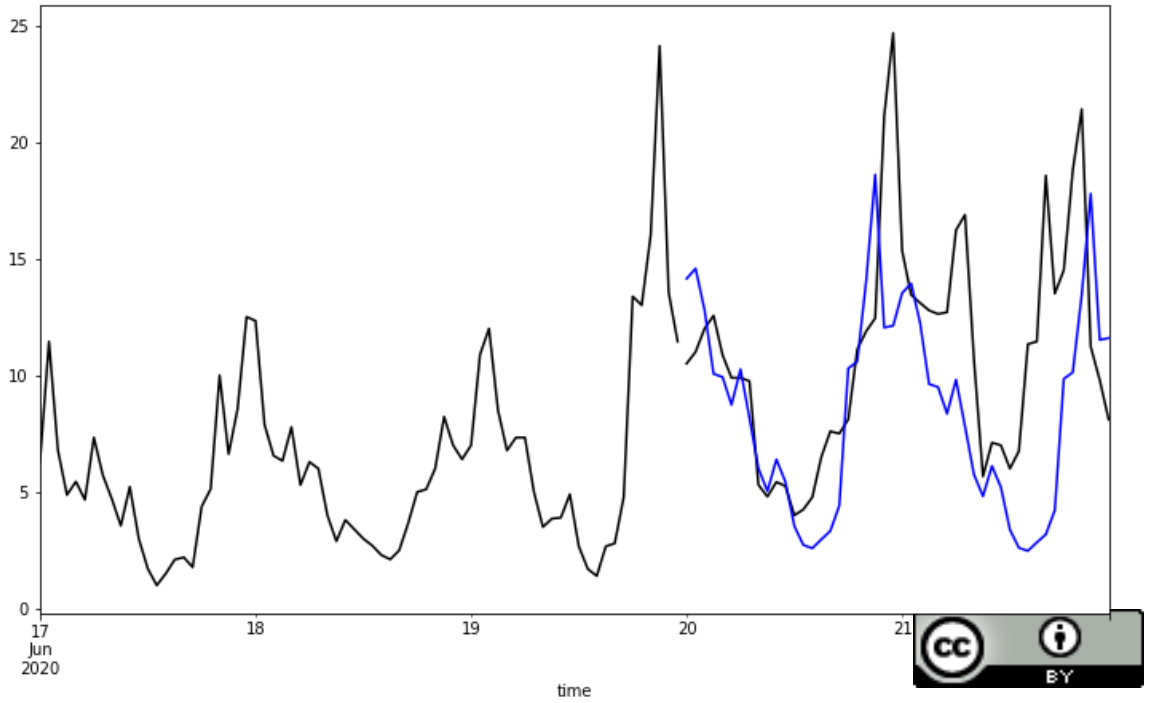
Long Short-Term Memory (LSTM) 模型是一種適合使用在連續資料的預測模型，因為它會對不同時間的資料產生不同的長短期記憶，並藉此預測出最後的結果。



資料預測：Holt-Winter

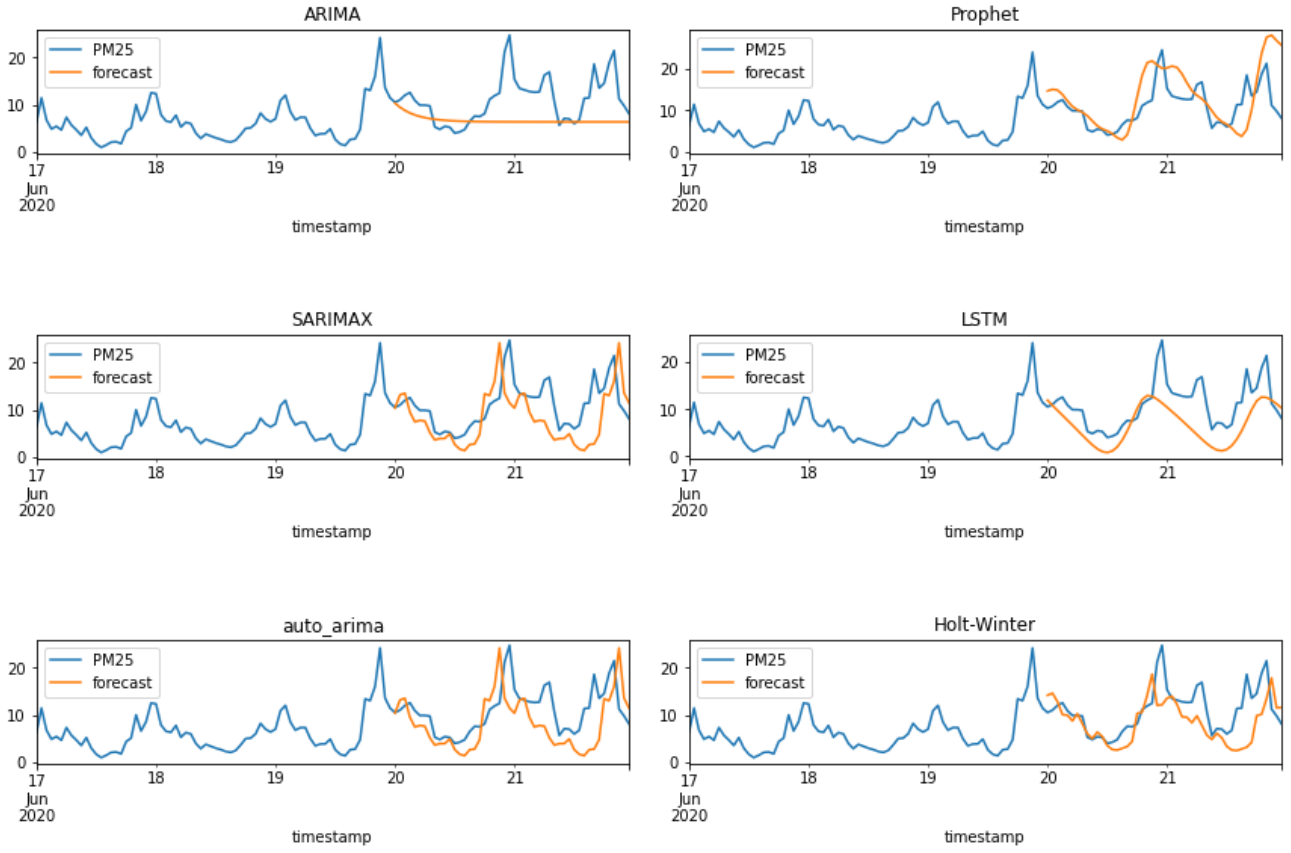
```
data_hw = air_hour.loc['2020-06-17':'2020-06-21']
train_len = -48
train = data_hw.iloc[:train_len]
test = data_hw.iloc[train_len:]
trainData = TimeSeriesData(train.reset_index(), time_col_name='timestamp’)

# Specify parameters
params = HoltWintersParams(trend="mul",
        seasonal="mul", seasonal_periods=24,)

# Create a model instance
m = HoltWintersModel(data=trainData, params=params)
m.fit()

# Forecast
fcst = m.predict(steps=48, freq='H')
data_hw['forecast'] = fcst[['time', 'fcst']].set_index('time’)

fig, ax = plt.subplots(figsize=(12, 7))
train.plot(ax=ax, label='train', color='black')
test.plot(ax=ax, color='black')
fcst.plot(x='time', y='fcst', ax=ax, color='blue')
ax.get_legend().remove()
```

Holt-Winter 模型是一種利用移動平均的概念，分配歷史資料的權重，以進行資料預測的方法。



資料預測：綜合比較

```
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(12, 8))

data_arima[['PM25', 'forecast']].plot(ax=axes[0, 0], title='ARIMA')
data_sarimax[['PM25', 'forecast']].plot(ax=axes[1, 0], title='SARIMAX')
data_autoarima[['PM25', 'forecast']].plot(ax=axes[2, 0], title='auto_arima')
data_prophet[['PM25', 'forecast']].plot(ax=axes[0, 1], title='Prophet')
data_lstm[['PM25', 'forecast']].plot(ax=axes[1, 1], title='LSTM')
data_hw[['PM25', 'forecast']].plot(ax=axes[2, 1], title='Holt-Winter')

fig.tight_layout(pad=1, w_pad=2, h_pad=5)
```



4. 時間序列資料分群

- 學習使用快速傅立葉轉換 (FFT) 與小波轉換 (Wavelet) 擷取時序資料的特徵
- 採用非監督式學習的方法，針對時序資料進行分群

資料下載

只使用 2021/08 的資料

```
!mkdir Air CSV_Air
!wget -O Air/2021.zip -q
"https://history.colife.org.tw/?r=/download&path=L%2Bepuuawo%2BWTgeizqi%2FkuK3noJTpmaJf5qCh5ZyS56m65ZOB5b6u5Z6L5oSf5ris5ZmoLzIwMjEuemlw"
!unzip Air/2021.zip -d Air

folder = 'Air/2021/202108'
extension_zip = 'zip'
extension_csv = 'csv'
for itemin os.listdir(folder):
  if item.endswith(extension_zip):
    file_name = f'{folder}/{item}'
    zip_ref = zipfile.ZipFile(file_name)
    zip_ref.extractall(folder)
    zip_ref.close()

air_month = pd.DataFrame()
for itemin os.listdir(folder):
  if item.endswith(extension_csv):
       file_name = f'{folder}/{item}'
       df = pd.read_csv(file_name, parse_dates=['timestamp'])
       air_month = air_month.append(df)
air_month.set_index('timestamp', inplace=True)
air_month.sort_values(by='timestamp', inplace=True)
```

套件安裝與引用

```
!pip install --upgrade pip
!pip install tslearn

import numpy as np
import pandas as pd
import pywt
import os, zipfile
import matplotlib.pyplot as plt

from pyCIOT.data import *
from datetime import datetime, timedelta
from numpy.fft import fft, ifft
from pywt import cwt
from tslearn.clustering import TimeSeriesKMeans
```

資料前處理

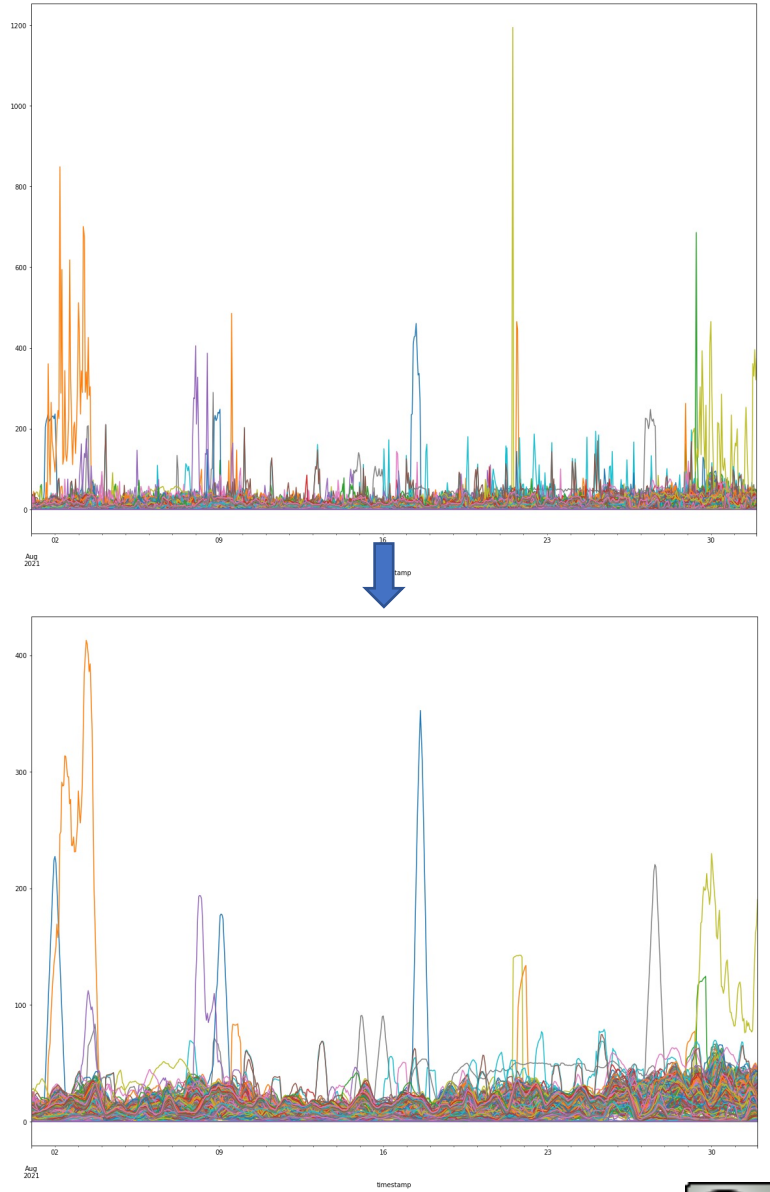
- 先找出資料中共有多少站點，並將這些站點資訊存在一個序列中
- 接著將每個站點的資料存成一個欄，並放入 air 這個 dataframe 中
- 最後將所有下載的資料與解壓縮後產生的資料移除，以節省雲端的儲存空間

```
id_list = air_month['device_id'].to_numpy()
id_uniques = np.unique(id_list)

air = pd.DataFrame()
for iin range(len(id_uniques)):
    # print('device_id==''' + id_uniques[i] + '''')
    query = air_month.query('device_id==''' + id_uniques[i] + '''')
    query.sort_values(by='timestamp', inplace=True)
    query_mean = query.resample('H').mean()
    air = pd.concat([air, query_mean], axis=1)

!rm -rf Air
```

資料前處理

- 將資料缺漏 (Nan) 部分刪除並繪圖
- 將資料做移動平均 (w=10) 後再繪圖

```
air = air[:-1]
# 將具有Nan值的欄位捨棄
air_clean = air.dropna(1, how='any')
air_clean.plot(figsize=(20, 15), legend=None)

# 使用移動平均使曲線變化更為平順
air_clean = air_clean.rolling(window=10, min_periods=1).mean()
air_clean.plot(figsize=(20, 15), legend=None)
```



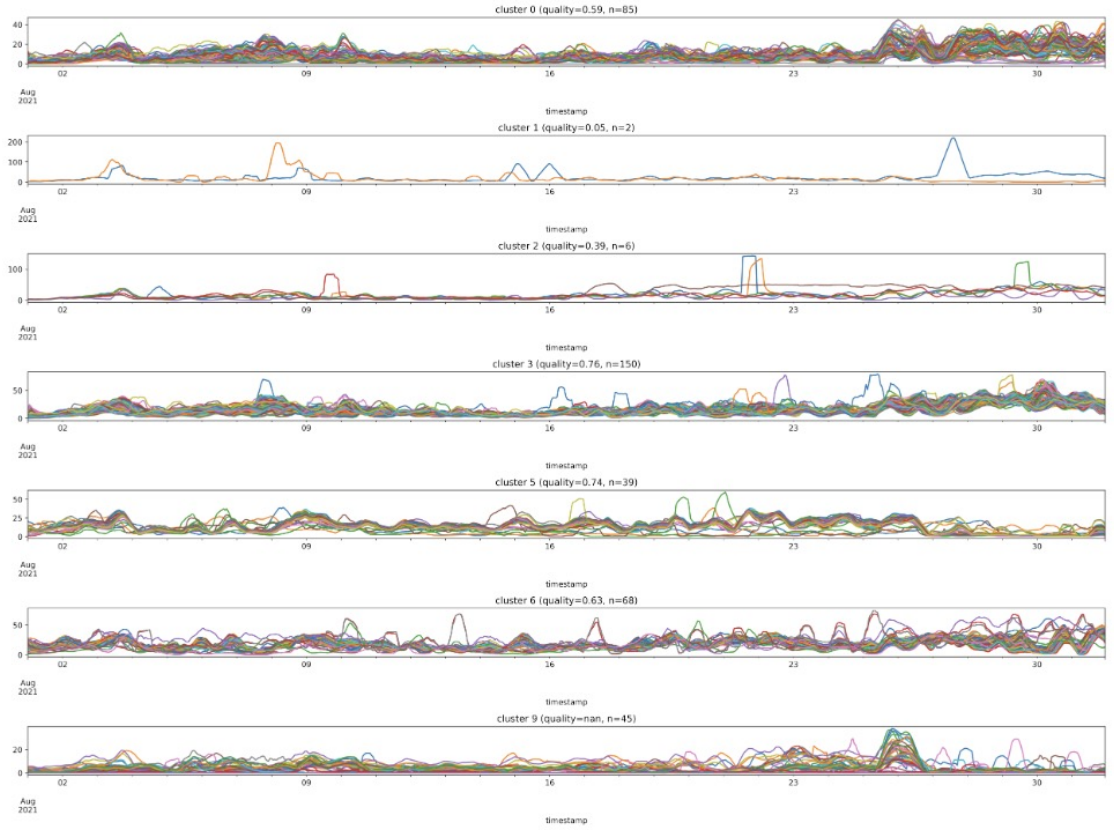
資料分群 (Clustering)

- 使用 KMeans 方法：
  1. 先決定 *k* 的值，亦即最後要分成幾個群 (k=10)；
  2. 隨機 *k* 筆資料，當為起始的中心點；
  3. 計算每筆資料到每個群心的距離，並選擇最近的群心，將該筆資料歸屬到該群；
  4. 針對每個群，重新計算其新的群心，並重複上述步驟，直到 *k* 的群的群心不再有變動為止。

```
air_transpose = air_clean.transpose()
# n_cluster:分群數量, max_iter: 分群的步驟最多重複幾次
model = TimeSeriesKMeans(n_clusters=10, metric="dtw", max_iter=5)
pre = model.fit(air_transpose)

# build helper df to map metrics to their cluster labels
df_cluster = pd.DataFrame(list(zip(air_clean.columns, pre.labels_)), columns=['metric', 'cluster'])

# 將只有一筆資料的群刪除 and make some helper dictionaries and lists
cluster_metrics_dict = df_cluster.groupby(['cluster'])['metric'].apply(lambda x: [x for x in x]).to_dict()
cluster_len_dict = df_cluster['cluster'].value_counts().to_dict()
clusters_dropped = [cluster for cluster in cluster_len_dict if cluster_len_dict[cluster]==1]
clusters_final = [cluster for cluster in cluster_len_dict if cluster_len_dict[cluster]>1]
clusters_final.sort()
```

資料分群 (Clustering)

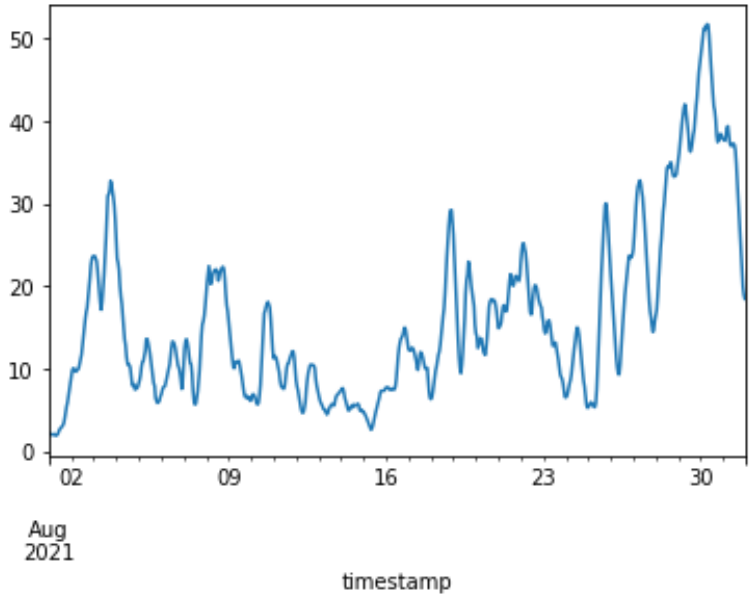
```
fig, axes = plt.subplots(nrows=len(clusters_final), ncols=1, figsize=(20, 15), dpi=500)
# legend = axes.legend(loc="upper left", bbox_to_anchor=(1.02, 0, 0.07, 1))
for idx, cluster_number in enumerate(clusters_final):
  x_corr = air_clean[cluster_metrics_dict[cluster_number]].corr().abs().values
  x_corr_mean = round(x_corr[np.triu_indices(x_corr.shape[0],1)].mean(),2)
  plot_title = f'cluster {cluster_number} (quality={x_corr_mean}, n={cluster_len_dict[cluster_number]})'
  air_clean[cluster_metrics_dict[cluster_number]].plot(ax=axes[idx], title=plot_title)
  axes[idx].get_legend().remove()

fig.tight_layout()
plt.show()
```

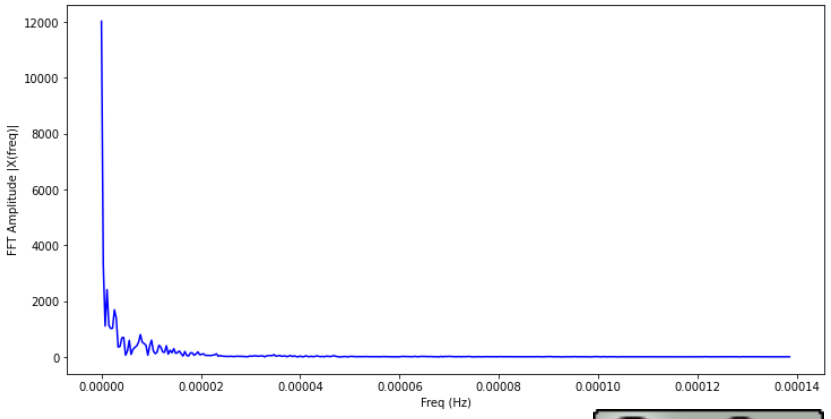


快速傅立葉轉換 (Fast Fourier Transform)

- 能將原始資料由時域 (time domain) 轉換至頻域 (frequency domain)，以方便更進一步的特徵擷取與資料分析

```
air_fft = pd.DataFrame()
col_names = list(air_clean.columns)
for name in col_names:
  X = fft(air_clean[name])
  N = len(X)
  n = np.arange(N)
  # get the sampling rate
  sr = 1 / (60*60)
  T = N/sr
  freq = n/T

  # Get the one-sided specturm
  n_oneside = N//2
  # get the one side frequency
  f_oneside = freq[:n_oneside]
  air_fft[name] = np.abs(X[:n_oneside])
```



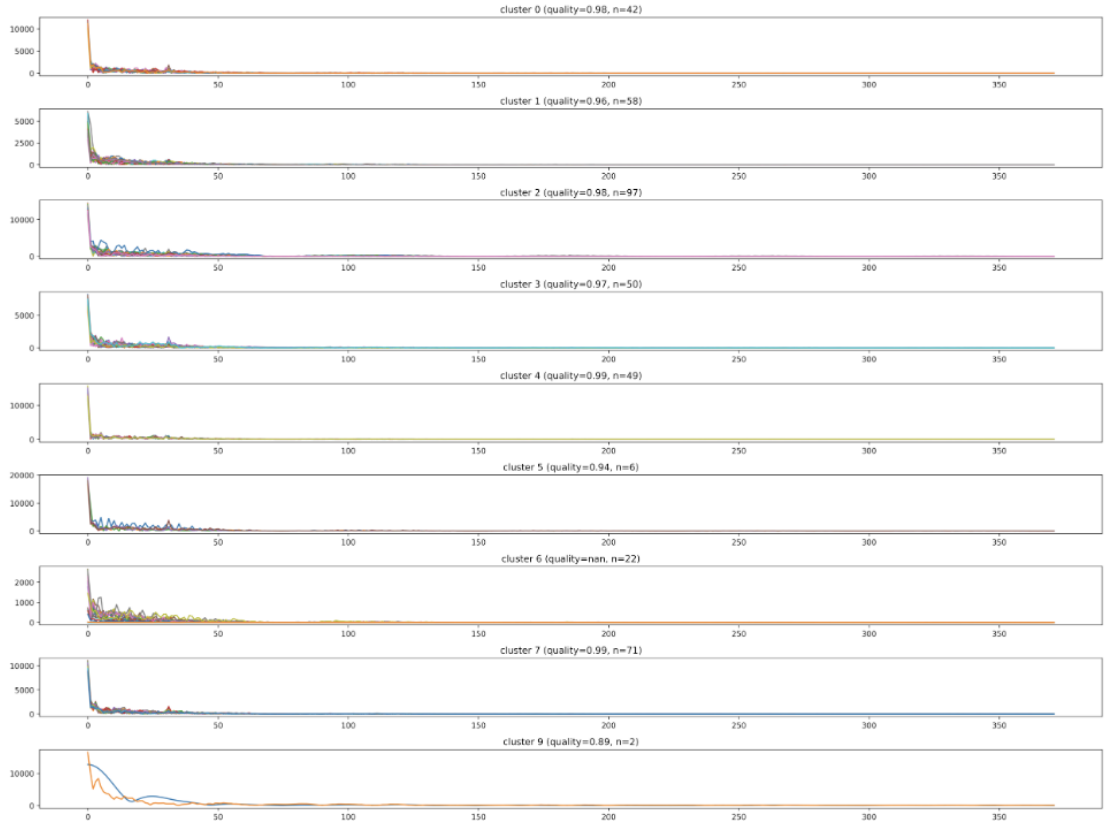


快速傅立葉轉換 (Fast Fourier Transform)

```
# 將資料行列交換以符合分群模型輸入的需求
fft_transpose = air_fft.transpose()
model = TimeSeriesKMeans(n_clusters=10, metric="dtw", max_iter=5)
pre = model.fit(fft_transpose)

# build helper df to map metrics to their cluster labels
df_cluster = pd.DataFrame(list(zip(air_fft.columns, pre.labels_)), columns=['metric', 'cluster'])

# make some helper dictionaries and lists
cluster_metrics_dict = df_cluster.groupby(['cluster'])['metric'].apply(lambda x: [x for x in
x]).to_dict()
cluster_len_dict = df_cluster['cluster'].value_counts().to_dict()
clusters_dropped = [cluster for cluster in cluster_len_dict if cluster_len_dict[cluster]==1]
clusters_final = [cluster for cluster in cluster_len_dict if cluster_len_dict[cluster]>1]
clusters_final.sort()
```

快速傅立葉轉換 (Fast Fourier Transform)

```
fig, axes = plt.subplots(nrows=len(clusters_final), ncols=1, figsize=(20, 15), dpi=500)
for idx, cluster_number in enumerate(clusters_final):
  x_corr = air_fft[cluster_metrics_dict[cluster_number]].corr().abs().values
  x_corr_mean = round(x_corr[np.triu_indices(x_corr.shape[0],1)].mean(),2)
  plot_title = f'cluster {cluster_number} (quality={x_corr_mean}, n={cluster_len_dict[cluster_number]})'
  air_fft[cluster_metrics_dict[cluster_number]].plot(ax=axes[idx], title=plot_title)
  axes[idx].get_legend().remove()

fig.tight_layout()
plt.show()
```

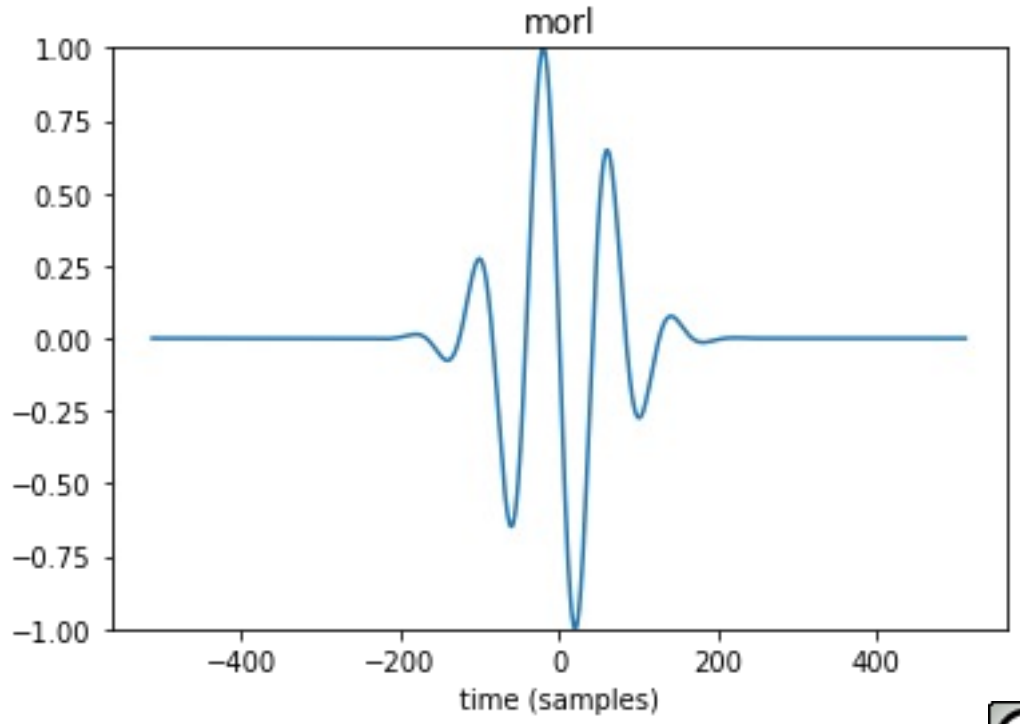


小波轉換 (Wavelet Transform)

- 小波轉換也是另一種將原始資料從時域轉換到頻域的常見手法，使用到母小波來擷取時序資料中的特徵，可以提供更多頻域資料的觀察角度

以 morl 母小波為例

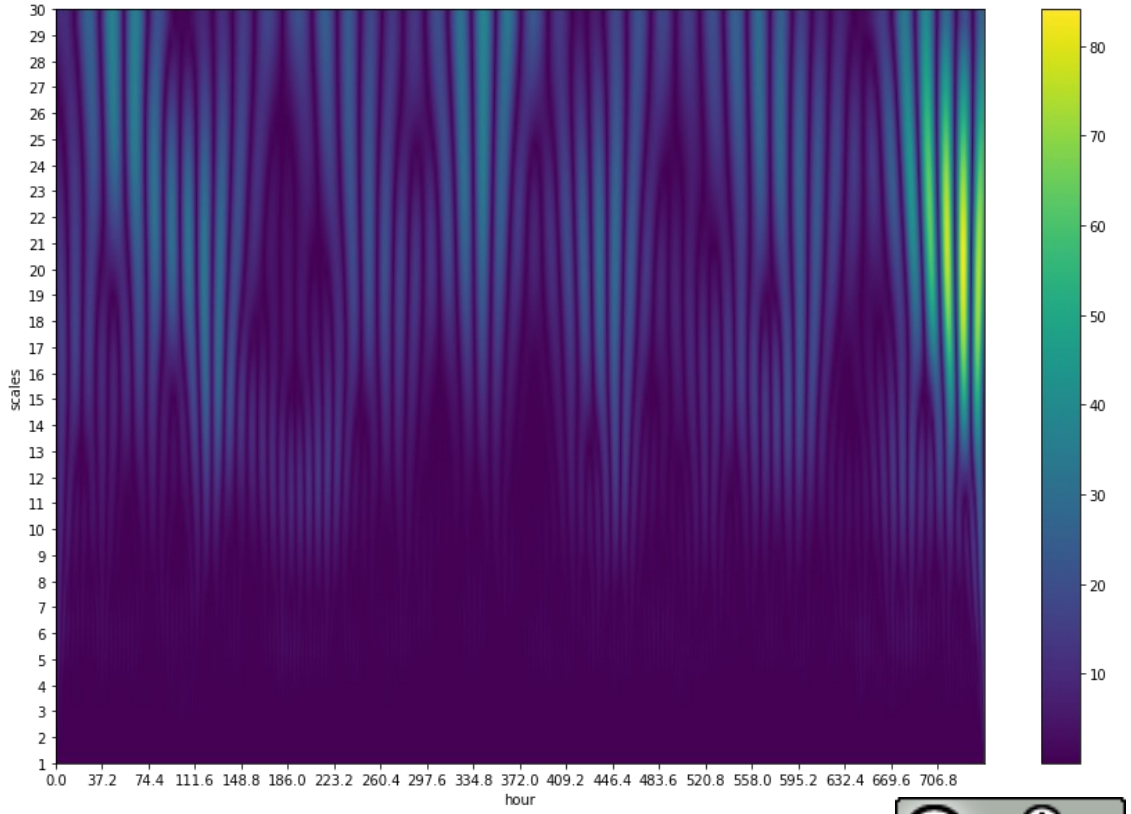
```
wav = pywt.ContinuousWavelet("morl")
scale = 1

int_psi, x = pywt.integrate_wavelet(wav, precision=10)
int_psi /= np.abs(int_psi).max()
wav_filter = int_psi[::-1]

nt = len(wav_filter)
t = np.linspace(-nt // 2, nt // 2, nt)
plt.plot(t, wav_filter.real)
plt.ylim([-1, 1])
plt.xlabel("time (samples)")
plt.title(wavelet_name)
```



小波轉換 (Wavelet Transform)

- 使用 morl 母小波，針對選定的感測器，將小波的縮放範圍設為 1~31 倍，進行小波轉換

```
F = 1  #Samples per hour
hours = 744
nos = np.int(F*hours)  #No of samples in 31 days

x = air_clean['08BEAC09FF2A']
scales = np.arange(1, 31, 1)
coef, freqs = cwt(x, scales, 'morl')

# scalogram
plt.figure(figsize=(15, 10))
plt.imshow(abs(coef), extent=[0, 744, 30, 1], interpolation='bilinear', cmap='viridis',
        aspect='auto', vmax=abs(coef).max(), vmin=abs(coef).min())
plt.gca().invert_yaxis()
plt.yticks(np.arange(1, 31, 1))
plt.xticks(np.arange(0, nos/F, nos/(20*F)))
plt.ylabel("scales")
plt.xlabel("hour")
plt.colorbar()
plt.show()
```



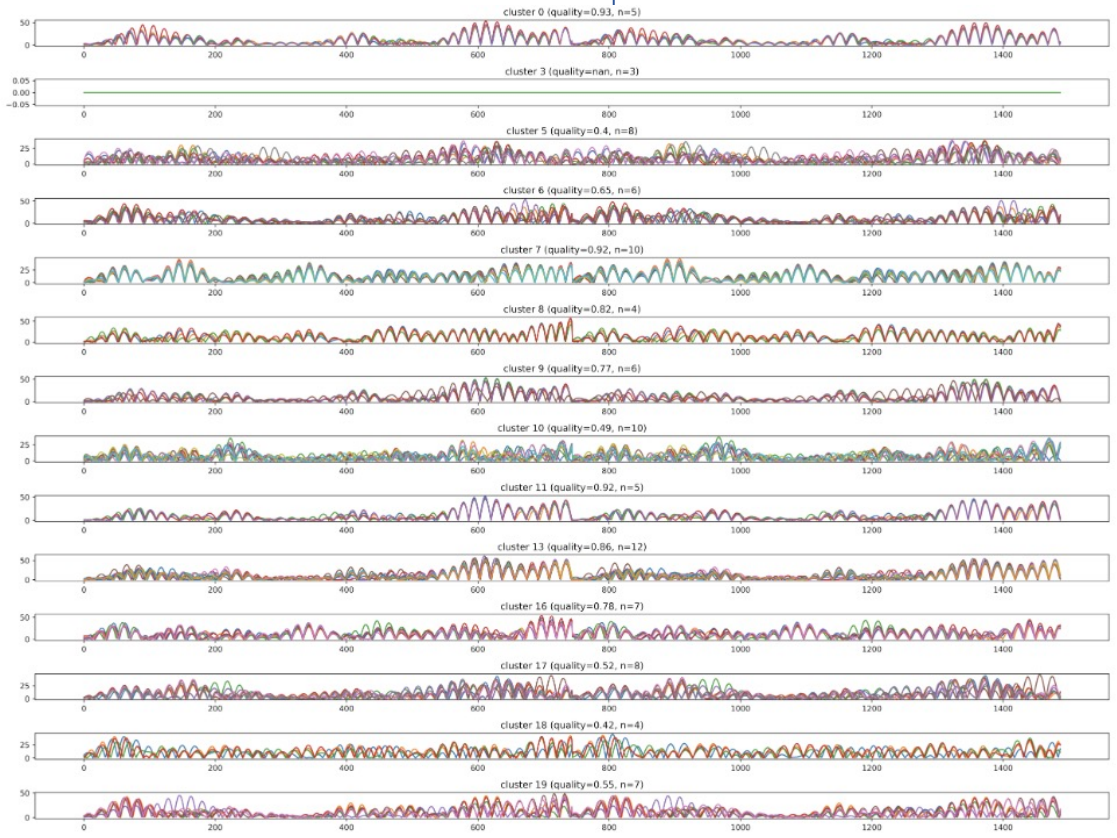
小波轉換 (Wavelet Transform)

- 預設分為 20 個群集
- 刪除單一資料的群集後，剩下 14 個群集

```
air_cwt = pd.DataFrame()
scales = np.arange(28, 31, 2)
col_names = list(air_clean.columns)
for namein col_names:
  coef, freqs = cwt(air_clean[name], scales, 'morl')
  air_cwt[name] = np.abs(coef.reshape(coef.shape[0]*coef.shape[1]))

air_cwt_less = air_cwt.iloc[:, 0:101]

cwt_transpose = air_cwt_less.transpose() # 將資料行列交換以符合分群模型輸入的需求
model = TimeSeriesKMeans(n_clusters=20, metric="dtw", max_iter=10, verbose=1, n_jobs=-1)
pre = model.fit(cwt_transpose)

# build helper df to map metrics to their cluster labels
df_cluster = pd.DataFrame(list(zip(air_cwt.columns, pre.labels_)), columns=['metric', 'cluster'])

# make some helper dictionaries and lists
cluster_metrics_dict = df_cluster.groupby(['cluster'])['metric'].apply(lambda x: [x for x in x]).to_dict()
cluster_len_dict = df_cluster['cluster'].value_counts().to_dict()
clusters_dropped = [cluster for cluster in cluster_len_dict if cluster_len_dict[cluster]==1]
clusters_final = [cluster for cluster in cluster_len_dict if cluster_len_dict[cluster]>1]
clusters_final.sort()
```

小波轉換 (Wavelet Transform)

```
fig, axes = plt.subplots(nrows=len(clusters_final), ncols=1, figsize=(20, 15), dpi=500)
for idx, cluster_number in enumerate(clusters_final):
  x_corr = air_cwt[cluster_metrics_dict[cluster_number]].corr().abs().values
  x_corr_mean = round(x_corr[np.triu_indices(x_corr.shape[0],1)].mean(),2)
  plot_title = f'cluster {cluster_number} (quality={x_corr_mean}, n={cluster_len_dict[cluster_number]})'
  air_cwt[cluster_metrics_dict[cluster_number]].plot(ax=axes[idx], title=plot_title)
  axes[idx].get_legend().remove()

fig.tight_layout()
plt.show()
```



參考資料

- 民生公共物聯網 (https://ci.taiwan.gov.tw)
- 民生公共物聯網資料服務平台 (https://ci.taiwan.gov.tw/dsp/)
- 民生公共物聯網歷史資料 (https://history.colife.org.tw/)
- Rob J Hyndman and George Athanasopoulos, Forecasting: Principles and Practice, 3rd edition (https://otexts.com/fpp3/)
- NIST Engineering Statistics Handbook (https://www.itl.nist.gov/div898/handbook/index.htm)