團隊測驗報告

報名序號:111011 (報名序號(格式:111XXX)已寄至隊長email)

團隊名稱:_____Urban2.0_____

註1:請用本PowerPoint 文件撰寫團隊程式說明, **請轉成PDF檔案繳交**。
註2:依據競賽須知第七條, 第 4項規定:

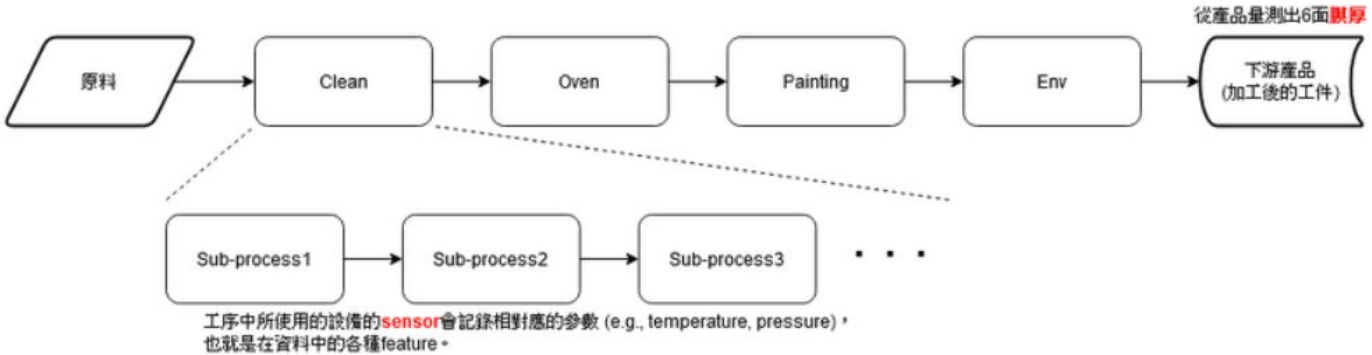
測試報告之簡報資料不得出現企業、學校系所標誌、提及企業名稱、學校系所、教授姓名及任何可供辨識參賽團隊組織或個人身分的資料或資訊, 違者取消參賽資格或由評審會議決議處理方式。

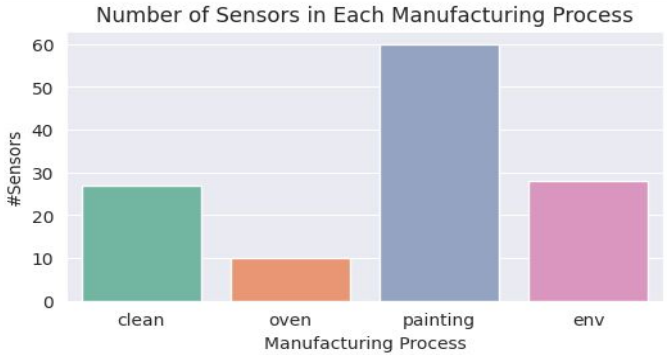
一、資料前處理(說明資料前處理過程)

- Data Overview
- NaN Analysis
- Duplicated Samples
- Outliers
- Feature Exploration
  - Sensor Value Trend
  - Correlation

Data Overview

本次競賽提供四種製程的感測器數 值作為特徵, 期望參賽者可以對加工後的工件 **膜厚**進行預測。而每個製程所擁有的感測器數 值數量不同 (如下圖所示)。





Process	Number of sensors
clean	27
oven	10
paintin	60
env	28

圖1. 挑戰目標及資料概覽

Data Overview (cont.)

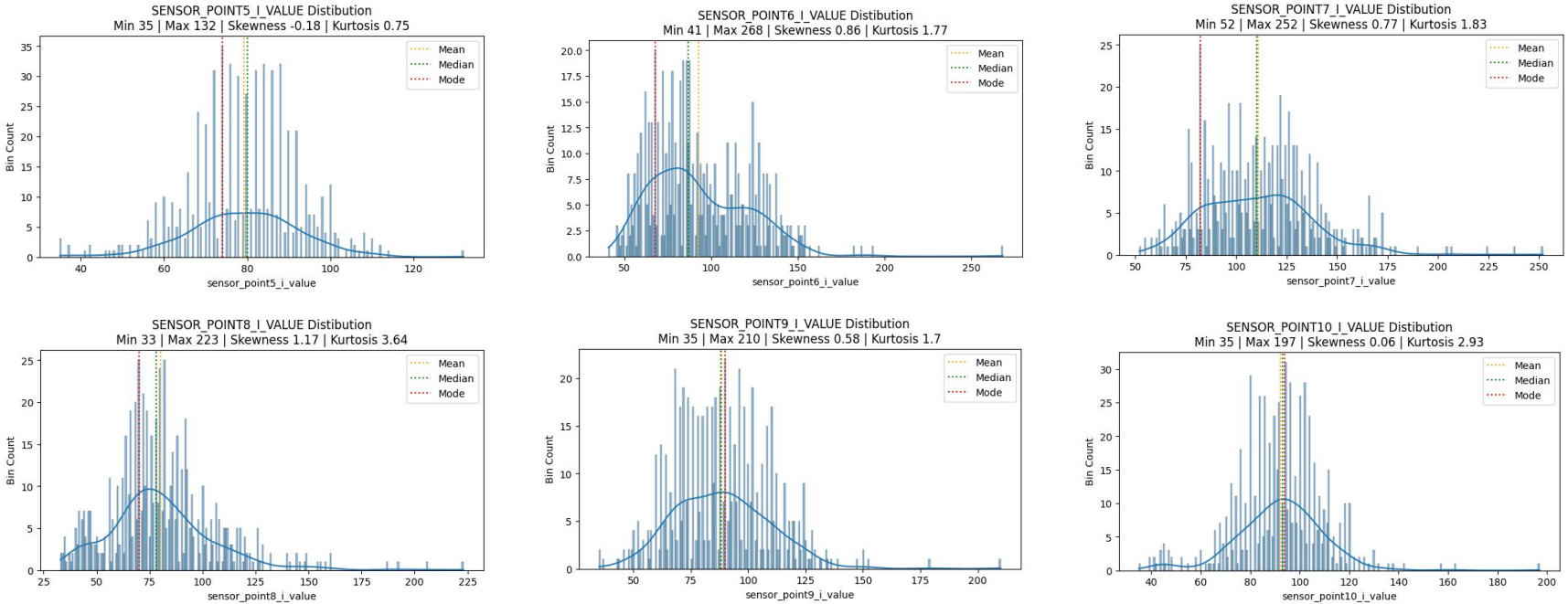


圖2. 六面膜厚數值分佈圖

NaN Analysis

NaN或0在此資料集中表示對應工序沒有被執行，也就是感測器不會有數 值量測與記錄。我們用 NaN 取代所有0以表示感測器未進行偵測，也確保在資料分析過程不會扭曲缺 值的意涵。

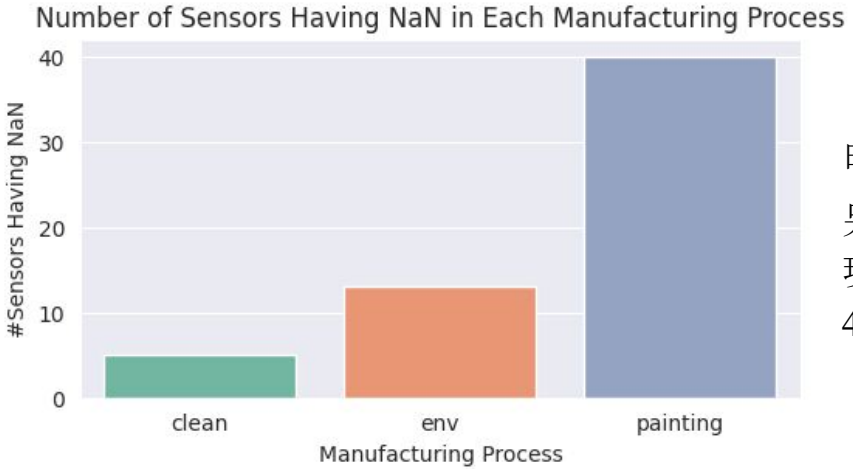


圖3. 各製程中包含缺失 值的感測器數量

由圖可看出在oven製程中並沒有缺失 值。

另外，在painting製程中約66.67%的感測器有出現至少一個缺失 值，env以及clean製程則分別為46.43%和18.52% 。

NaN Analysis (cont.)

觀察缺失值在不同感測器中出現的比率, 我們發現同製程中、多個感測器之間的缺失 值比率相同。
因此我們進一步視覺化觀察 NaN出現的位置。

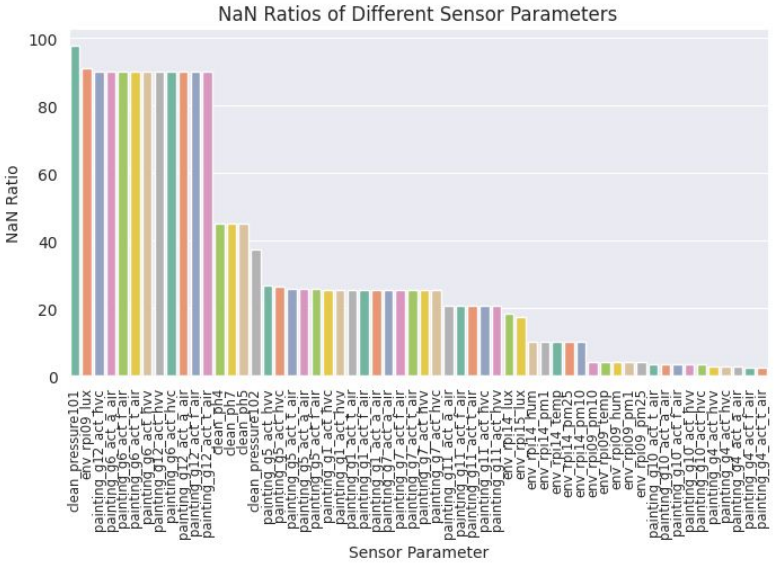


圖4. 缺失值在各個感測器中出現的比率

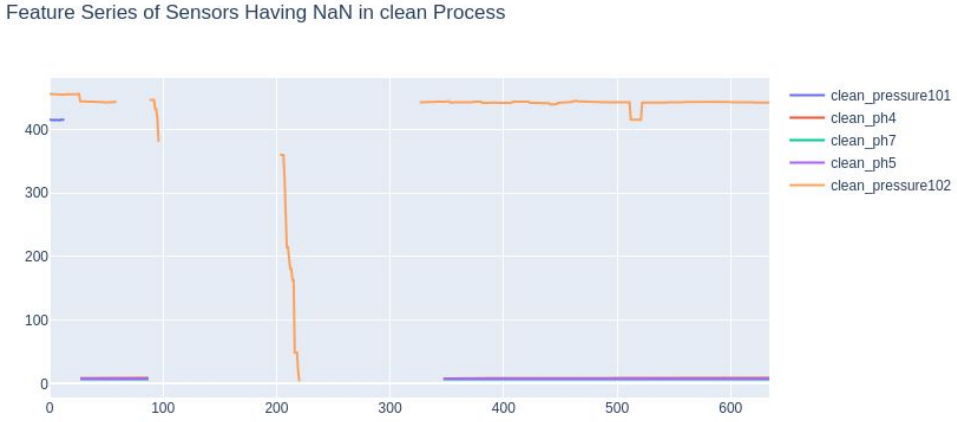


圖5. 橫軸為工件編號, 縱軸為感測器數 值。
clean_ph4, clean_ph5 和 clean_ph7感測器會同步出現Nan。

NaN Analysis (cont.)

painting_g1 ~ g12共12個subprocess中，8個subprocess在hvv、hvc、a_air、f_air和t_air五種類別的感測器皆有出現缺失值。另外，我們發現subproccess 之間的缺失值有**同步**情形，如：g1 & g7、g4 & g10、g6 & g12以及g5 & g11，兩兩一組的subprocess同時出現感測器未進行偵測的行為。

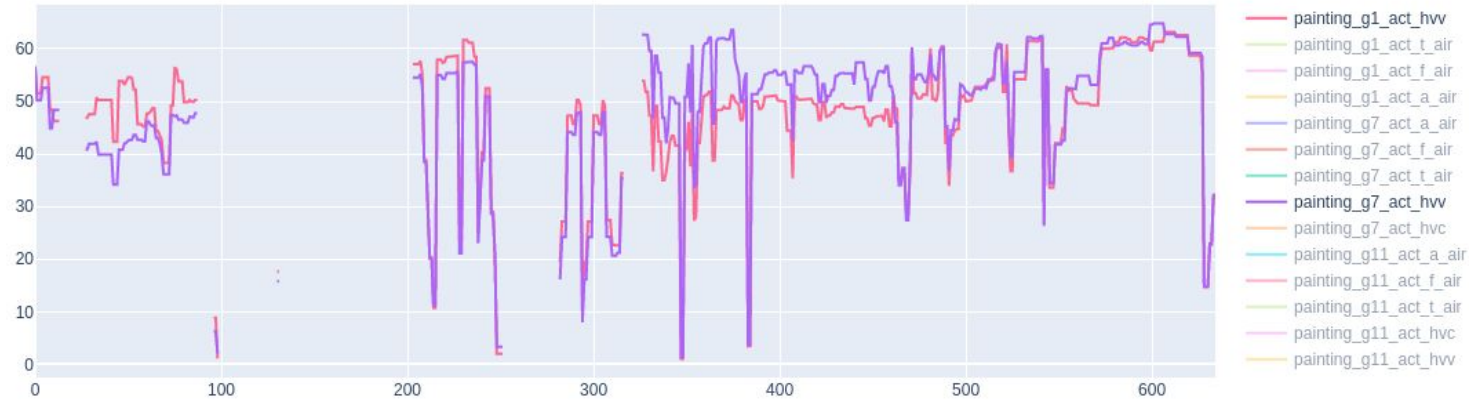


圖6. 橫軸為工件編號，縱軸為該工件的感測器數 值。
以painting_g1&g7為例，hvv感測器停止動作的行為會同時發生。

NaN Analysis (cont.)

在前面的分析中我們發現 subprocess間的關係, 並在此進一步觀察製程之間的 NaN。對每個工件中缺失值的數量統計, 並拆分為四個製程以及所有製程之缺失 值加總的圖表進行觀察。

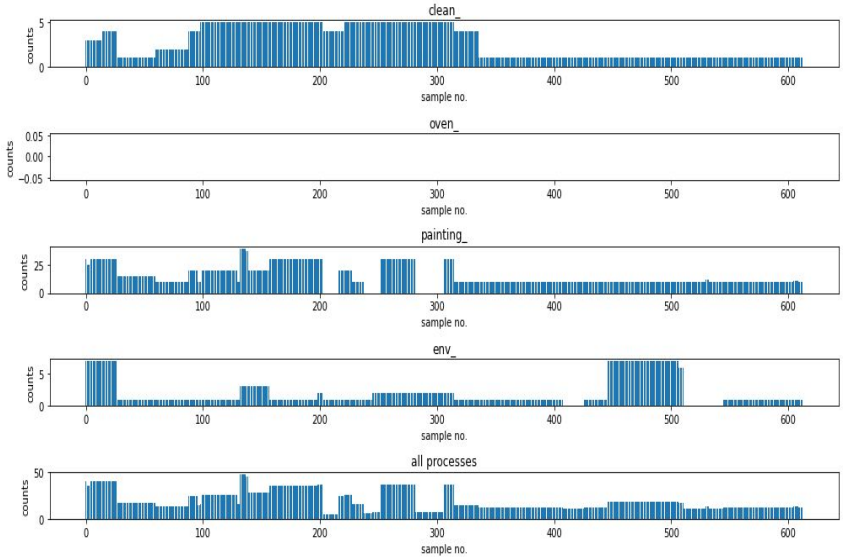


圖7. 四種製程下, 每一筆資料中NaN出現的次數。橫軸為第n筆資料, 縱軸為NaN在該筆資料中出現的個數。

藉由四個製程間NaN數量的比較, 觀察到同個工件不同製程之間 NaN出現次數並沒有相同的趨勢。

另外, 由最下方的圖中可以看出同個工件不同製程之間NaN的數量並沒有互補的情形。從停止偵測的感測器數量上, 我們並沒有發現製程間的相互關係。

Duplicated Samples

在training set中存在22組**感測器數值與六面膜厚皆相同**的數據，共44筆。

另外，我們發現在training set中有188組感測器數值相同的工件但最終的膜厚並不相同。由於 188組重複的樣本總計為514筆training samples，佔所有資料的80%，因此無法直接將重複資料捨棄。

考量到缺乏相關背景知識，我們認為直接將重複工件的膜厚做簡單的平均或是取中位數來當訓練資料並非明智之舉。然而，這樣的觀察卻幫助我們激盪出重建資料集的想法，詳細作法請參考第二章節的Complete Dataset Generation。

Outliers

分別畫出每個工件的六項膜厚，明顯可以看出六項膜厚皆有 outlier的存在，膜厚度與大部分工件的數值差距極大。

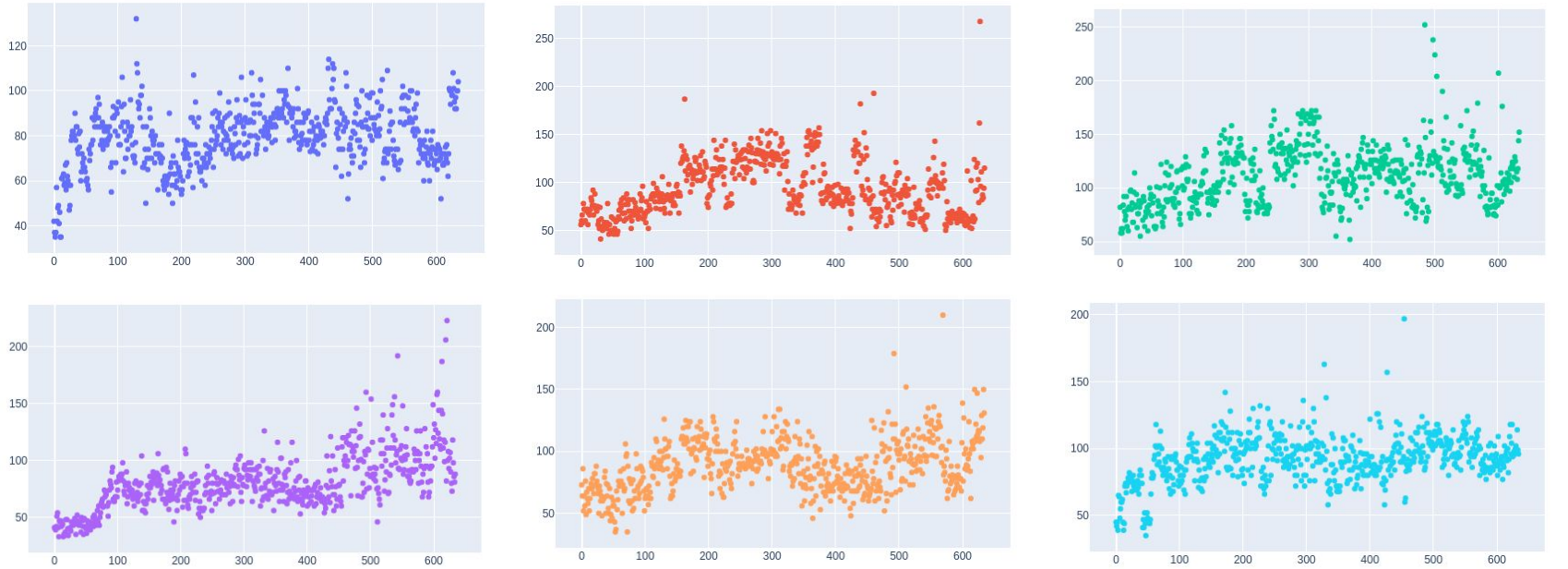


圖8. 橫軸為工件編號，縱軸為該工件的膜厚。六張圖由左到右由上到下依序為 sensor point 5 ~ 10。

Feature Exploration

Sensor Value Trend – Clean

先前的NaN數值分析中，我們觀察到製程中特定 subprocess間會有**同步缺失**的情形。再者，我們也發現subprocess間的數值會有類似**同步變動**的行為。以下圖中第 520個工件為例，製程 clean的大部分感測數值皆有下降。

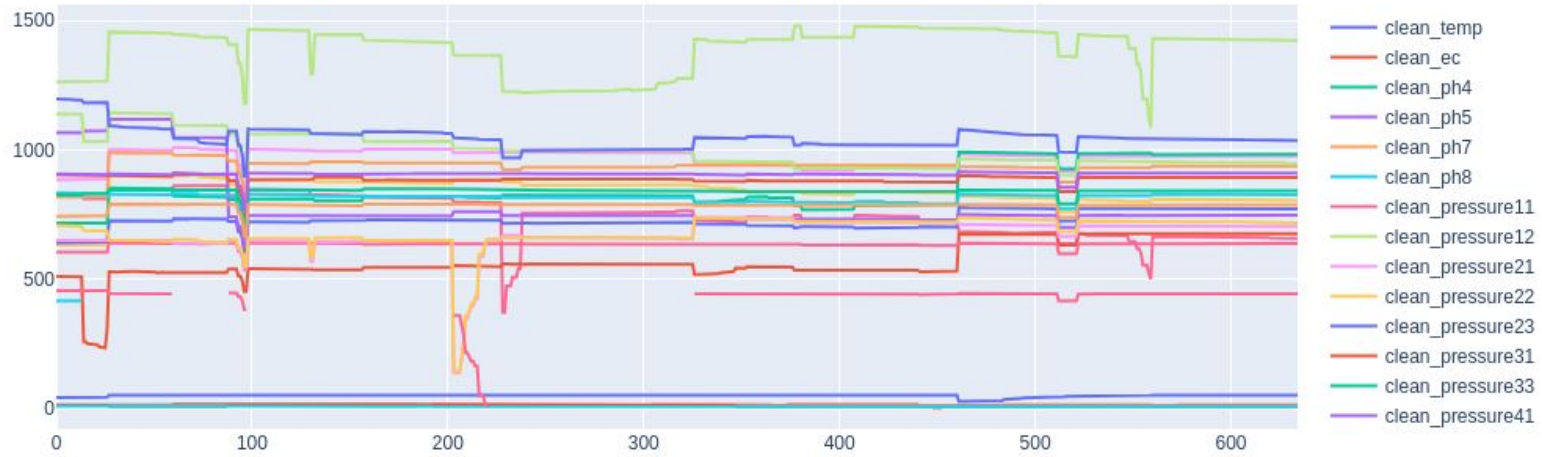


圖9. clean製程感測器數值趨勢觀察。橫軸為工件編號，縱軸為該工件的感測器數 值。由圖可以觀察同製程不同感測器之間數 值的變動及關係。

Sensor Value Trend – Painting

製程painting中, 數值變動的情形可以歸類為幾個組別 : g1 & g7、g2 & g8、g3 & g9、g4 & g10、g5 & g11以及g6 & g12, 兩兩一組相差六的subprocess數值變動方式尤為相似。

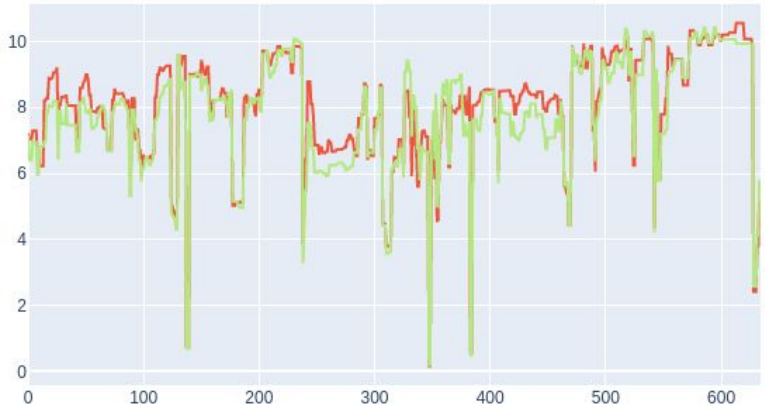


圖10. 紅線為painting_g2_act_a_air
綠線為painting_g8_act_a_air

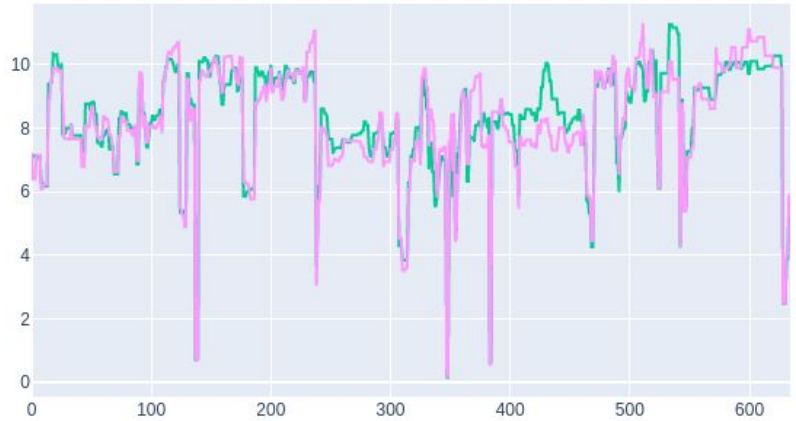


圖11. 綠線為painting_g3_act_a_air
粉線為painting_g9_act_a_air

Sensor Value Trend – Oven

oven製程下，p開頭的subprocess表現出類似的數值變動。a1、a2、a3間也能夠看出相同趨勢，當a1數值大幅降低時，a2、a3也同樣會大幅下降。

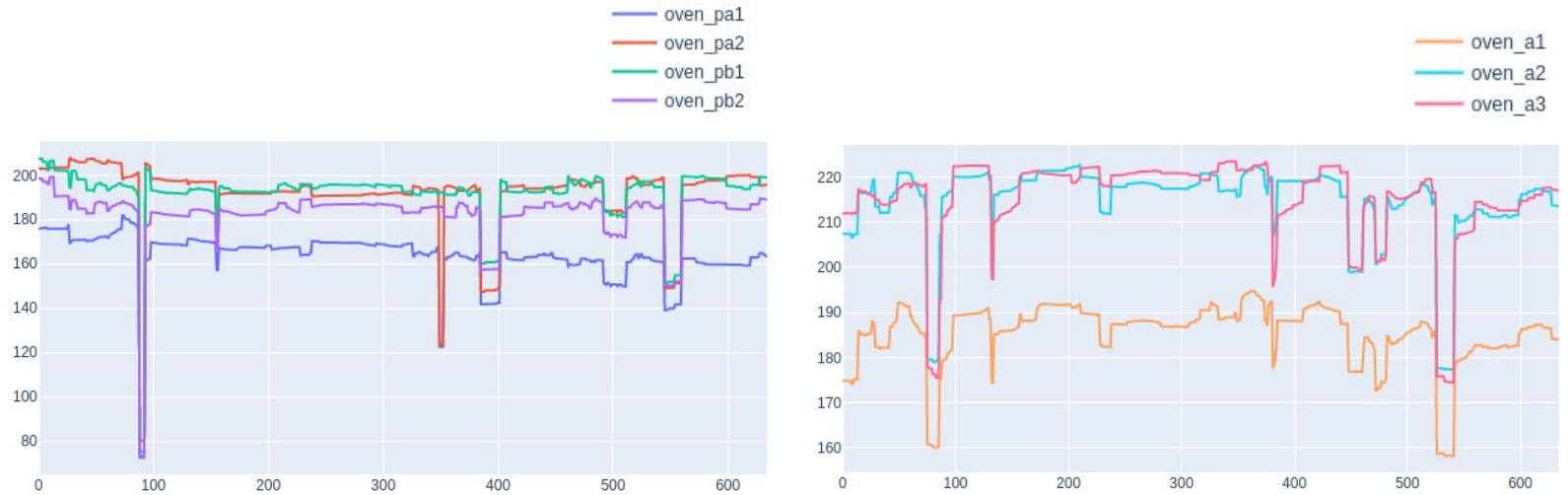


圖12. oven製程感測器數值趨勢觀察。橫軸為工件編號，縱軸為該工件的感測器數 值。

Sensor Value Trend – Env

env製程下, 同樣也能觀察到不同感測器之間數 值的同向趨勢。與前三個製程不同的是我們在此製程中觀察到反向趨勢, 左圖在 pm1、pm10、pm25數值上升的地方可以看到 hum感測數值的下降, rpi05_pm1、pm10、pm25與rpi07_hum的反向趨勢極為顯著。右圖為 rpi05下hum與pm25的比較, 同樣能發現hum與pm25大致上表現出相反方向的變動, 但趨勢卻沒有左圖強烈。

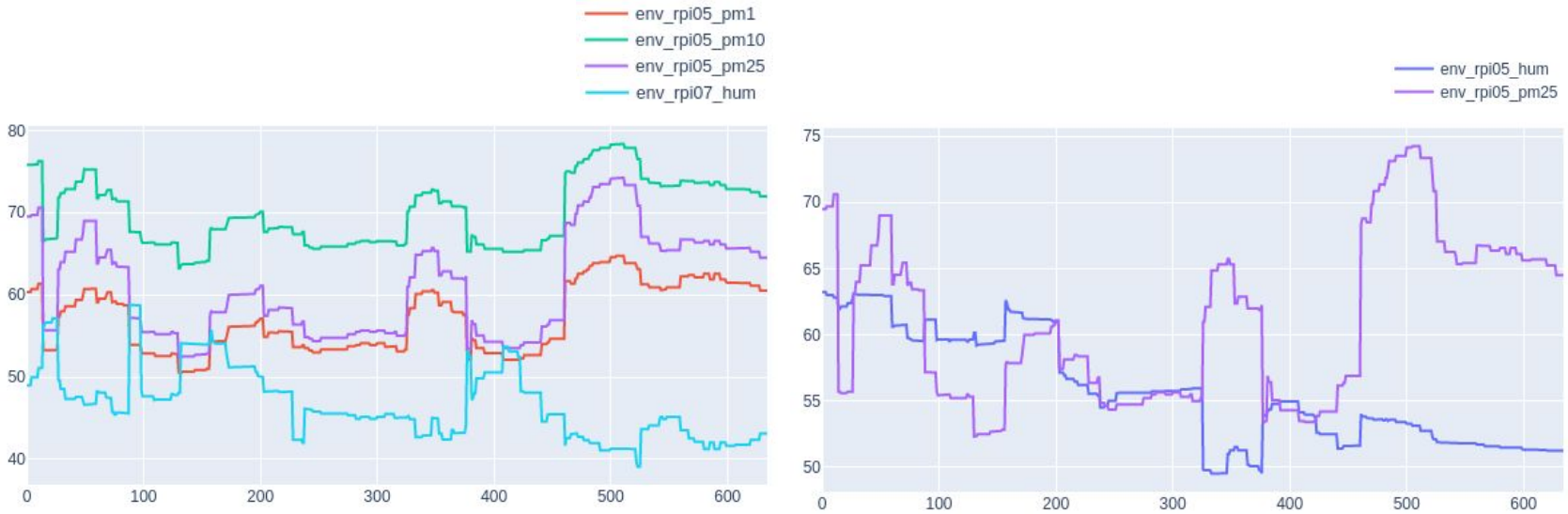


圖13. env製程感測器數值趨勢觀察。橫軸為工件編號, 縱軸為該工件的感測數 值。

Correlation

基於前面對於數值趨勢的觀察，進一步做correlation的分析。以painting製程為例，視覺化兩個感測器間的數值，可以看出強烈的正相關。

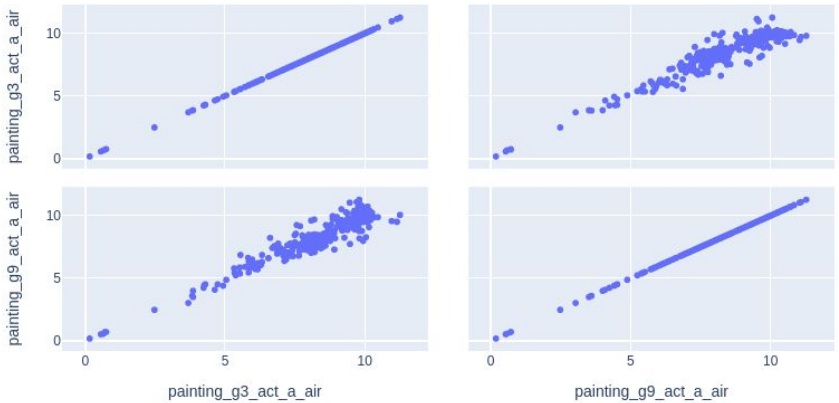


圖14. g3_act_a_air與g9_act_a_air雙變數分析結果

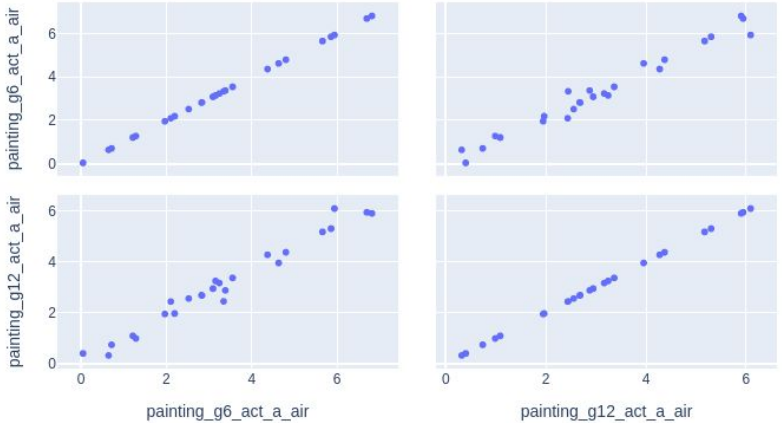


圖15. g6_act_a_air與g12_act_a_air雙變數分析結果

Correlation (cont.)

對製程下所有感測器計算相關性, 以更精確的進行數據分析。在此章節我們將 correlation大於0.75以及correlation小於-0.75定義為strong relationship。同時我們也在每個製程下加入 6項膜厚計算correlation, 四項製程皆未出現與 6項膜厚有stron relationship的感測器數值。

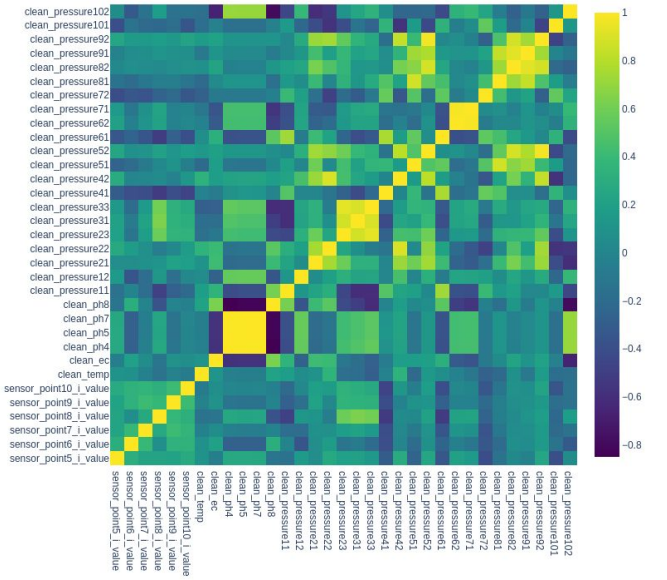


圖16. clean製程與六項膜厚correlation heatmap。clean製程下的strong relationship共有317組。

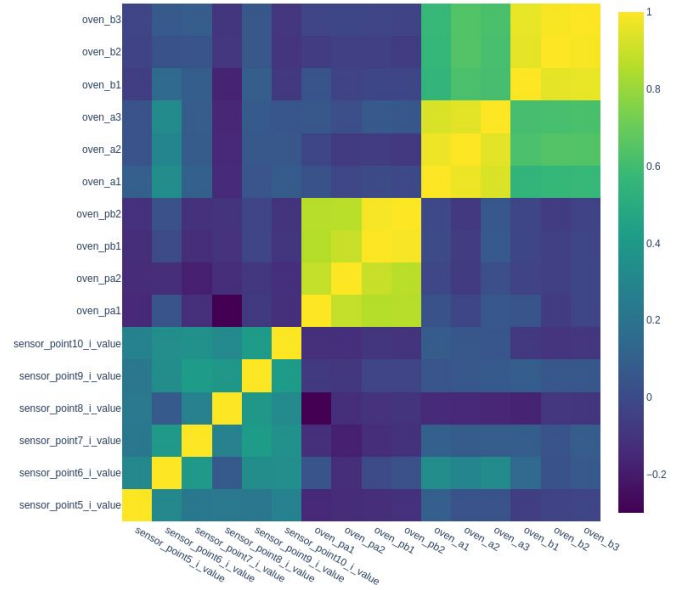


圖17. oven製程與六項膜厚correlation heatmap。oven製程下的strong relationship共有55組。

Correlation (cont.)

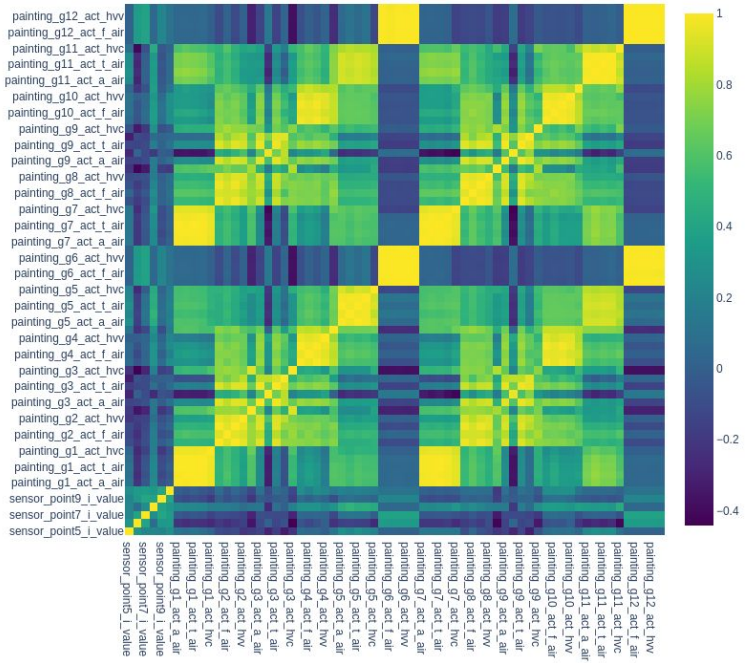


圖18. painting製程與六項膜厚 correlation heatmap。clean製程下的strong relationship共有23組。

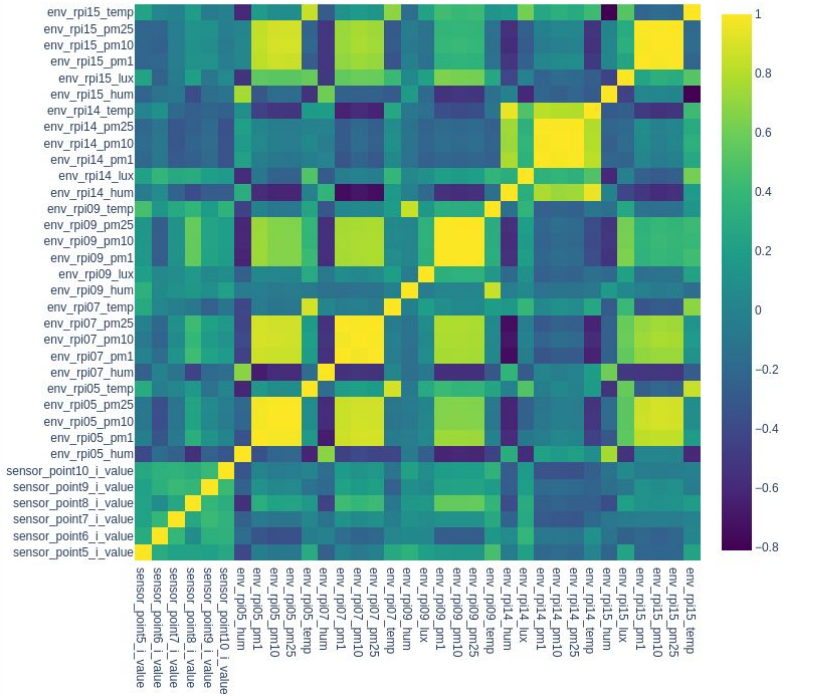


圖19. env製程與六項膜厚 correlation heatmap。clean製程下的strong relationship共有23組。

二、演算法和模型介紹(介紹方法細節)

- Overview
- Complete Dataset Regeneration
- Final Prediction Generation
  - Scenario Study
  - Heuristics - Mean and Direct Filling
  - Chunk-based Modeling

Overview

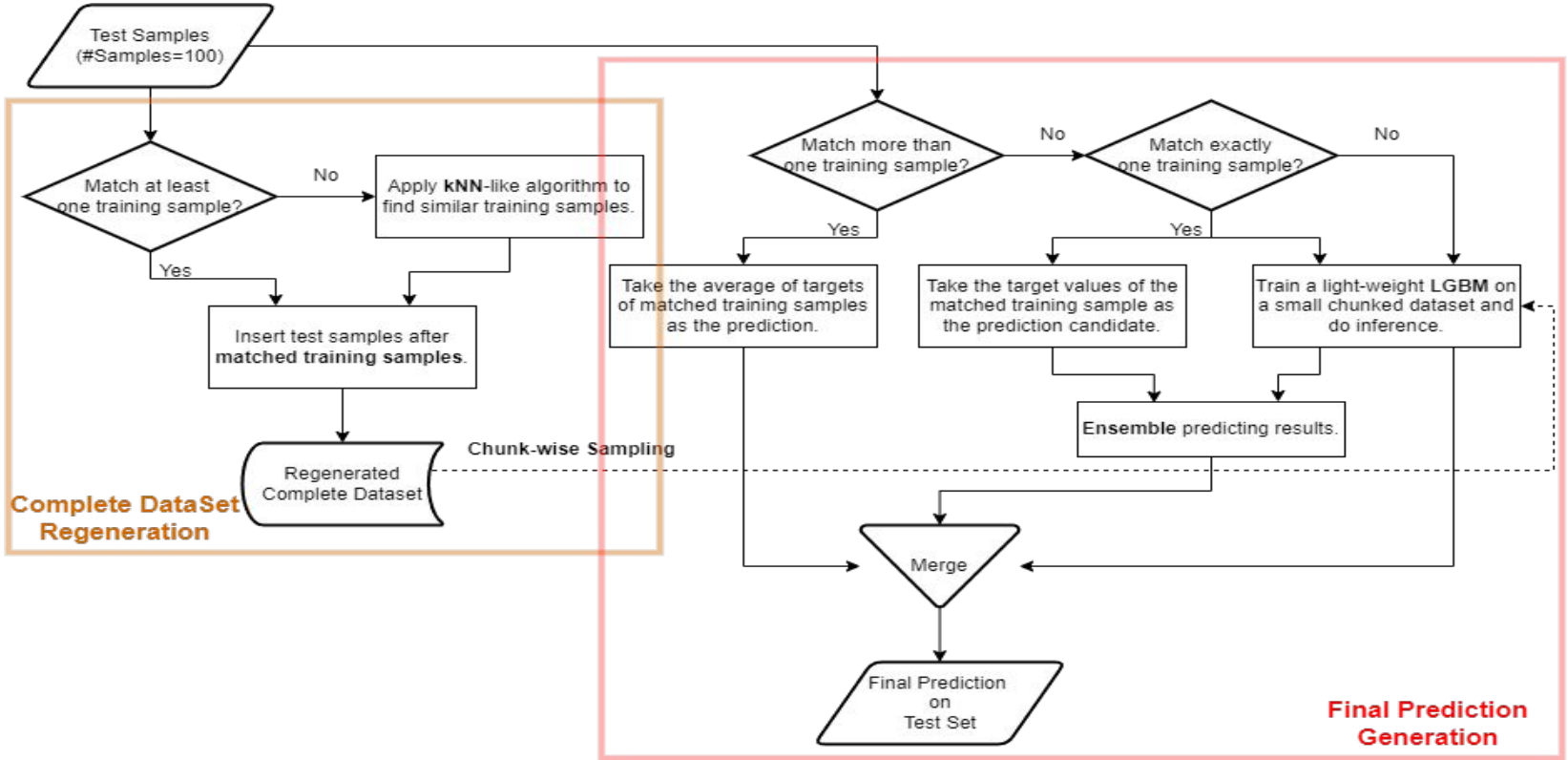


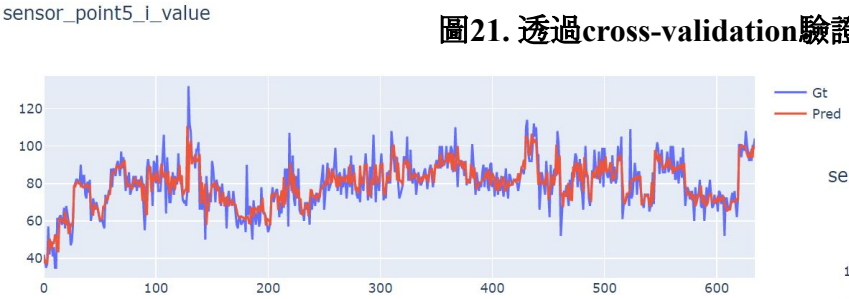
圖20. 核心演算法流程圖概覽

Complete Dataset Regeneration

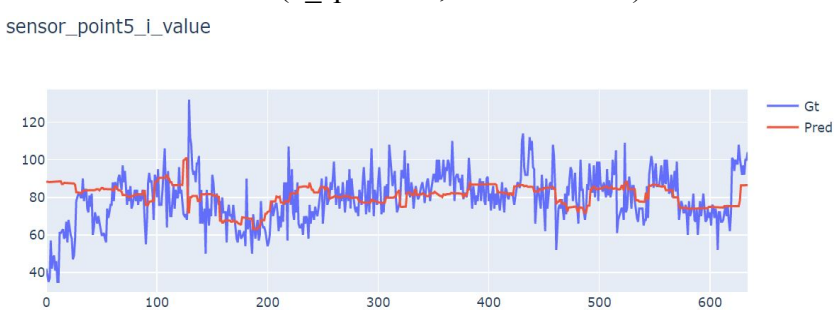
由於training set中存在許多感測器數值完全一致的樣本 (詳細說明請見章節Duplicated Samples)，而且感測器數值相近的樣本又被排列在一起，故我們猜想樣本是按照 **製程pipeline所具備的特性** 來作規則性的擺放。為驗證這樣的想法，我們比較兩種 cross-validation的方式，分別是KFold(n_splits=10, shuffle=**True**, random_state=168)以及KFold(n_splits=10, shuffle=**False**)。

從下圖預測結果可以看出在 **不shuffle**的情形下，validation連趨勢估計都做不好，故推論樣本擺放方式是依照**製程pipeline所具備的特性**做群聚。也就是**製程pipeline的特性**對於最終膜厚有決定性的影響，以至於當模型只用擁有特定製程特性的樣本來訓練時，會導致其 **不易泛化**到其他製程情境。

圖21. 透過cross-validation驗證training sample擺放方式



KFold(n_splits=10, shuffle=**True**, random_state=168)



KFold(n_splits=10, shuffle=**False**)

*** 另外五個target也可以觀察到一樣的現象**

Complete Dataset Regeneration (cont.)

為了驗證test set的抽樣方式, 我們同時觀察 training和test sets的感測器數值, 從圖中 (以clean_temp為例)可發現sample擺放方式有**同步的**規律, 因此將test samples 映射回training set的特定樣本區間中 (e.g., 類**kNN**方法)將有助於後續建模預測的發想。

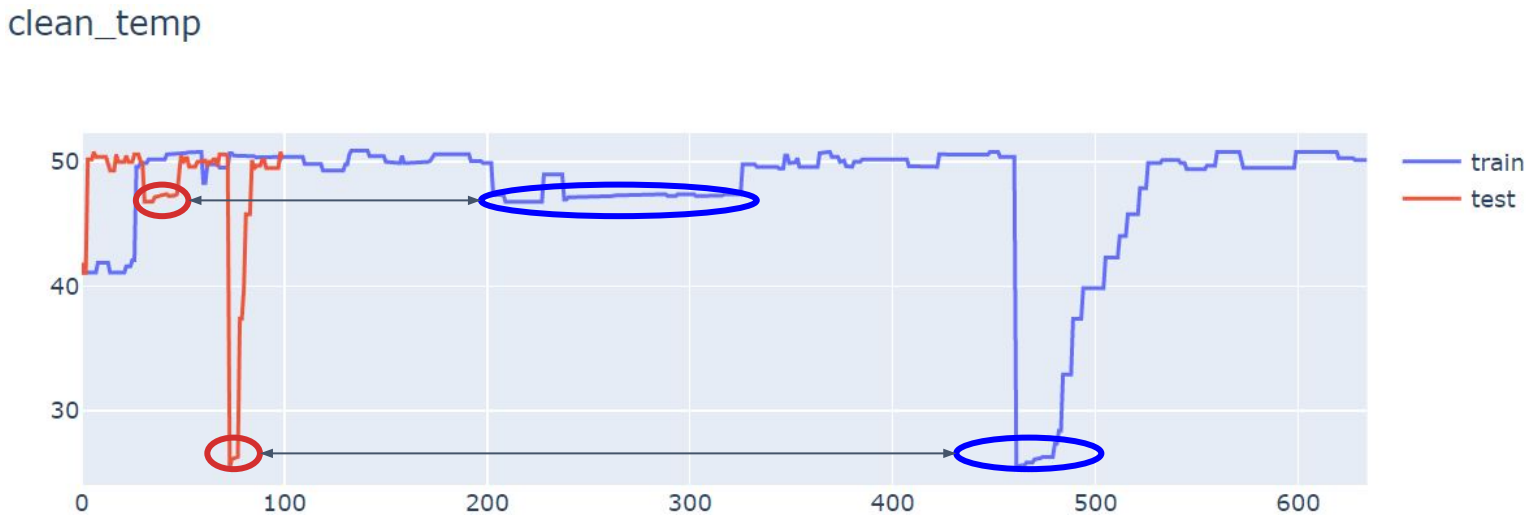


圖22. Training和test sample擺放方式有同步規律

Complete Dataset Regeneration (cont.)

為了將test samples塞回training set中，我們用了兩階段的判斷準則，假設 test sample與至少一筆 training sample有一樣的感測器數 值，則將此筆test sample插入相同的training sample後方，而符合這個條件的test samples共有83筆。

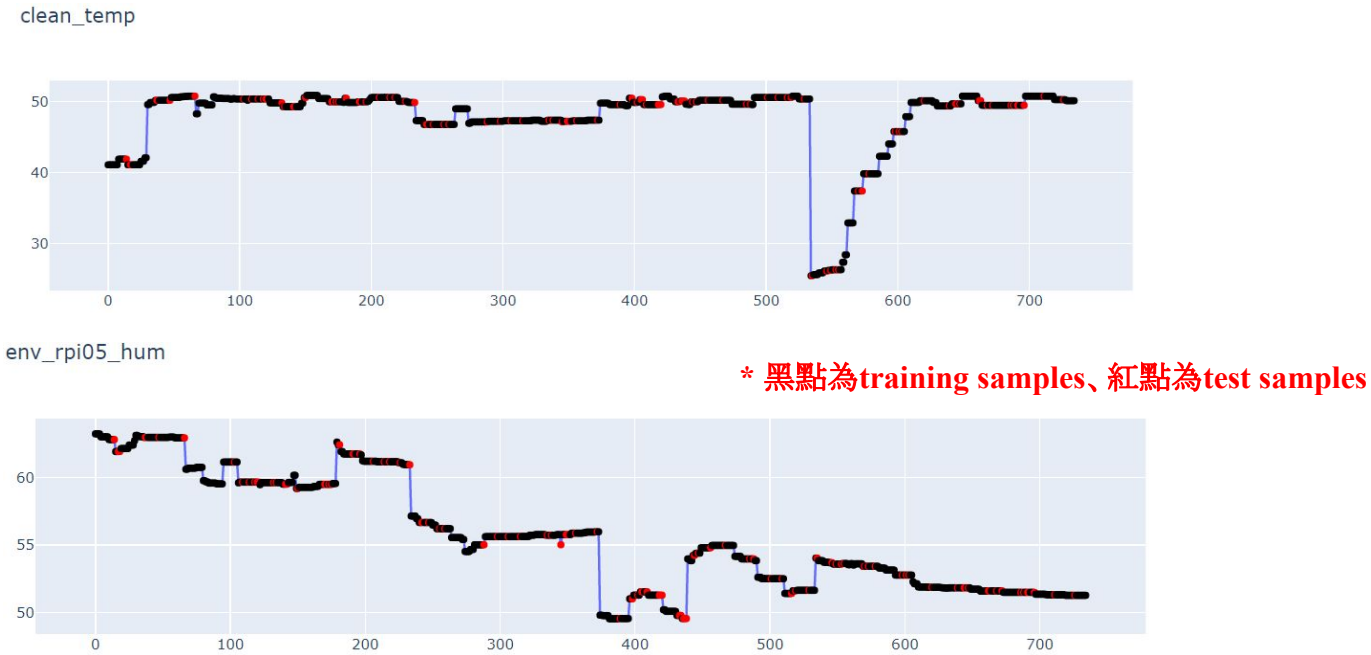
反過來說，若test sample完全沒有對應到任何一筆 training sample，則透過類**kNN**的方法，來找出**極度相似**的training sampls，再將此筆test sample插入相似的training sample後方。

```python
def get_train_nn(dataset: str, idx: List[int], rtol: float) -> Dict[int, List[int]]:
    """Find and return nearest training samples of each given sample
    (either in training or test set).

    Parameters:
        dataset: either `train` or `test`
        idx: sample indices
        rtol: relative tolerance

    Return:
        close_rows: nearest rows of each given sample
    """
    df_base = df_tr.copy() if dataset == "train" else df_test.copy()
    close_rows = defaultdict(list)

    for samp_i in tqdm(idx):
        samp_feats = df_base.iloc[samp_i][FEAT_COLS]
        for tr_i, r in df_tr.iterrows():
            # Iterate all training samples
            if np.allclose(samp_feats.astype(np.float), r[FEAT_COLS].astype(np.float), atol=0, rtol=rtol, equal_nan=True):
                close_rows[samp_i].append(tr_i)

    return close_rows
```

圖23. 以類kNN法來找出與test sample極度相似的training samples

Complete Dataset Regeneration (cont.)

確認為每個test sample可以插入的位子後，我們將新的資料集 complete.csv生成出來，觀察下圖 (以clean_temp與env_rpi05_hum為例)可以看出test samples插入的位子是符合原先的假設與期待的。



*** 黑點為training samples、紅點為test samples**

圖24. Complete Dataset生成的結果

Final Prediction Generation

Scenario Study

在探討生成最終預測結果的實作方法前, 我們先將 test samples分為三種情境:

1. Test sample與大於一筆training sample擁有一樣的感測器數 值 (共41筆)
2. Test sample只與一筆training sample擁有一樣的感測器數 值 (共43筆)
3. Test sample沒有對應任何一筆 training sample (共17筆)

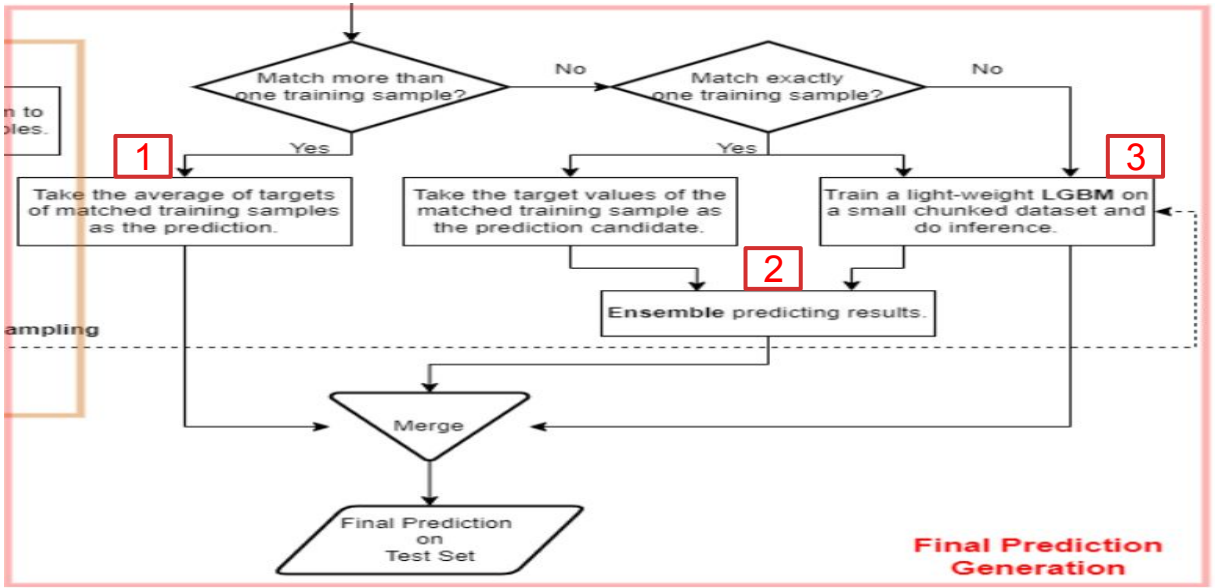


圖25. 三種預測情境在演算法流程中的位置

Heuristics - Mean and Direct Filling

在情境1中，由於一筆test sample與多於一筆training sample擁有一樣的感測器數 值，故我們選擇直接將這些training samples的膜厚取平均，作為預測答案。

而在情境2中，一筆test sample僅與一筆training sample擁有一樣的感測器數 值，我們將這筆training sample的膜厚作為**候選預測答案**，記做y_pred_hard。考量到資料集中有 outlier存在，我們認為直接以單筆training sample的膜厚當答案可能過於極端;因此，這個候選答案會與另外一個透過模型預測出的答案 (y_pred_soft，後面會說明)進行簡單平均，以達到一定的平滑效果。

- Take the average of targets of matched training samples

```
df_complete.loc[i, TARGET] = matched_tr_rows[TARGET].mean().values
```

- Directly use target values of the matched training sample

```
y_pred_hard = matched_tr_row[TARGET].values
```

Chunk-Based Modeling

呈上所述，情境2的y_pred_hard會和模型所預測出的y_pred_soft進行**ensemble**；此外，情境3中的17筆test sample沒有對應任何一筆training sample，故我們必須訓練模型來進行預測，而如何挑選合適的樣本來作為訓練集顯得至關重要，regenerated complete dataset在此時便派上用場。

我們以每個test sample作為中心點，往前與往後各抓取 10個樣本來作為小訓練集 (若被涵蓋的樣本剛好是test sample則捨棄不用)，故將之命名為**chunk-based** modeling，以此確保小型資料集 內的**製程流水線特性**相似。下圖演示chunk-based sampling的執行過程 (以clean_temp為例協助說明)。

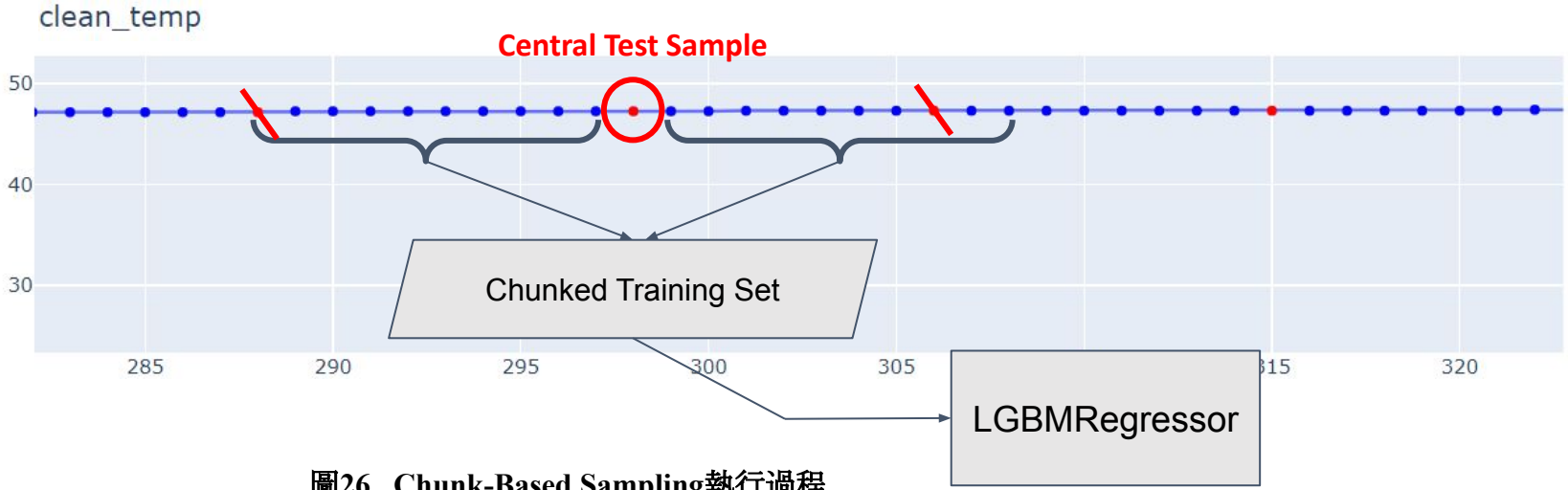


圖26. Chunk-Based Sampling執行過程

三、預測結果

在不同情境運用不同的方法進行預測，分別得到 41 (情境1)、43 (情境2)以及17筆 (情境3)test samples 的預測結果，最後將這些預測結果合併，以生成最終的 TestResult.xlsx。

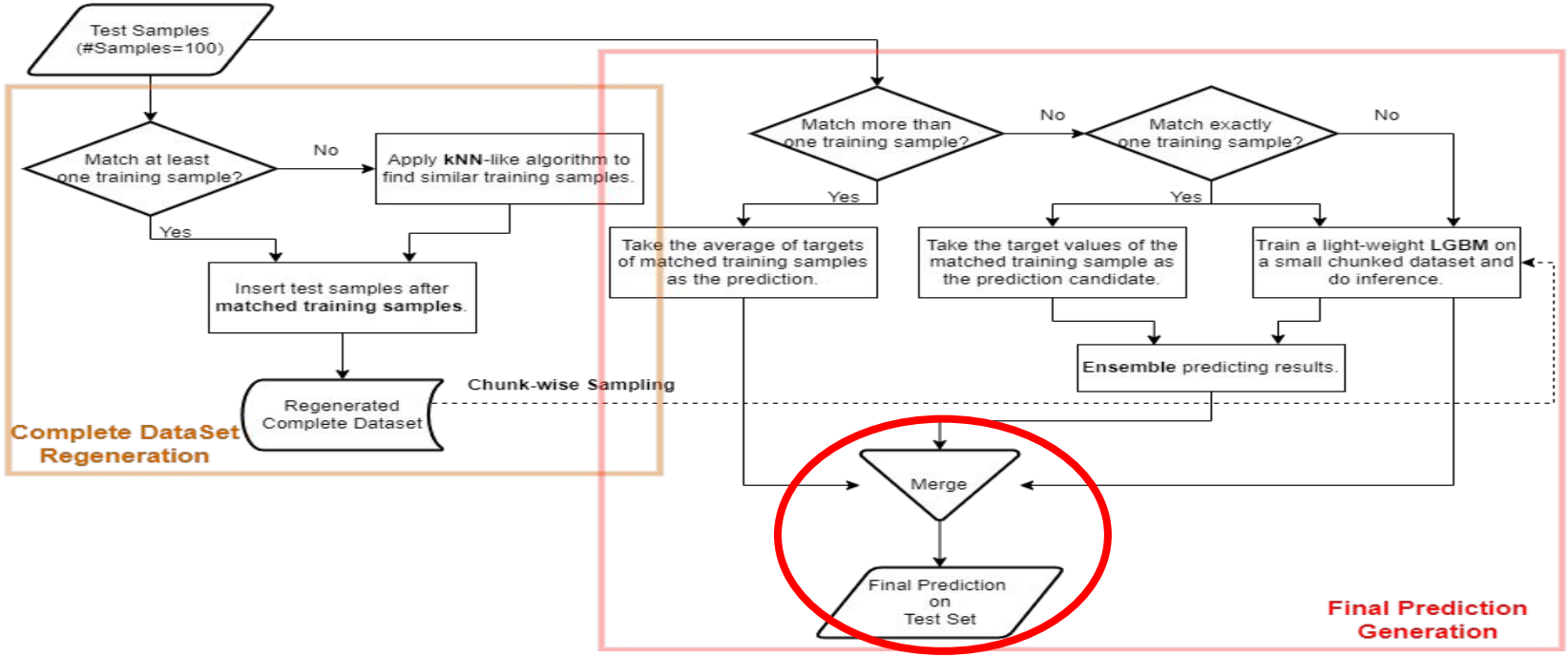


圖27. 最終預測結果生成示意圖

四、補充說明(其他或自行定義項目)

- PCA with Agglomerative Clustering

PCA with Agglomerative Clustering

除了在章節2的核心演算法外，我們也嘗試過利用 PCA先對感測器數值進行降維，再利用噪聲較小的component來進行Agglomerative Clustering。由於我們假設每個cluster保有自己的製程流水線特性，故我們對每個cluster分別進行建模，類似章節 2中的chunk-based modeling，這邊所採用的便是cluster-based modeling。

圖28. 利用elbow method選取最佳cluster數

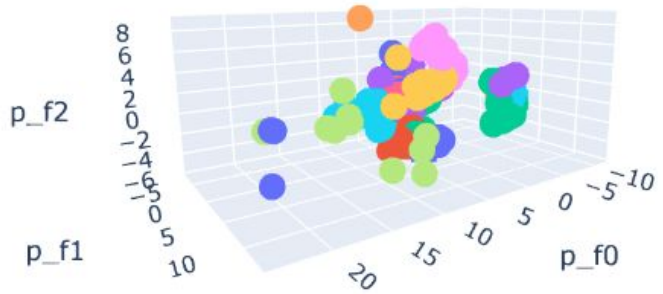


圖29. 3維空間中的clustering結果

* 最終因為cross-validation成效不佳，故捨棄此做法