新北市112學年度中小學科學展覽會

作品說明書

科　　別：電腦與資訊學科

組　　別：高級中等學校組

作品名稱：設計新型演算法並應用於增強迴歸直線的穩健性之探究

關 鍵 詞：演算法、穩健性、迴歸直線

編　號：

目錄

摘要

本研究在深入探討迴歸直線模型中異常值對模型的影響，並提出有效的處理方法。研究發現一般的最小平方法在面對數據中的異常值時容易受到影響，進而降低迴歸直線的穩健性。因此我們結合了分治法、Dijkstra 演算法、K-近鄰演算法與模糊理論等，自創了三種新穎的演算法，透過這些演算法算出新的迴歸直線，並進行對比分析其在模擬數據與真實數據上的穩健性。研究結果顯示我們的模型較不容易受到異常值的影響，相較於現有的迴歸直線模型，其穩健性更好。未來的研究方向可能包括進一步評估其他演算法的效能，深入了解不同演算法在不同應用領域下的適用性，以及進一步優化演算法以提高其效能和實用性。

壹、 前言

一、研究動機

隨著資訊科技不斷地推陳出新，資訊分析與處理已成為各學科領域中不可或缺的核心元素。在這個發展趨勢下，資料視覺化和迴歸直線分析成為處理大量數據的重要工具。儘管現有演算法（例如：最小平方法、Lasso 與 Ridge 等）能夠迅速處理數據，卻面臨受到異常值的干擾或需手動調整參數（懲罰項的係數）的挑戰。因此我們想設計一種演算法，希望能在兼顧高效性和精確性的前提下完成一條具有很好的穩健性之迴歸直線，這個演算法的獨特之處在於其能夠抵抗異常值的影響，同時無需進行人工調整任何參數，從而極大地降低了解決這個複雜問題的難度。本研究的目的在於提供一個更加穩健且高效的迴歸直線建模方法，有望對未來資訊分析領域帶來深遠的影響。

二、研究目的

我們為了深入了解迴歸直線模型在面對數據挑戰時的表現，並致力於設計更為穩健且自動化的演算方法，以提高模型的實用性和廣泛適用性，將研究目的細分成下列幾項:

（一）評估現有迴歸直線模型的穩健性

分析傳統迴歸模型（如線性迴歸）在面對異常值時的表現，探討其對異常值（或稱離群點，Outliers）的敏感性。

（二）開發新穎演算法以提高穩健性

探究並設計新的演算法，須在兼顧效率和準確性的情況下，提高迴歸直線模型對異常值的穩健性。

（三）消除手動調整參數的需求

探討能夠自動調整模型參數以提高穩健性的方法，從而在使用迴歸直線模型時須進行手動調整的問題。

三、 文獻回顧

迴歸直線的穩健性一直是統計學和數據分析領域中的重要議題 [1]，而線性迴歸模型是迴歸分析的基石；但在真實的數據（Real data）中，往往會存在著雜訊和異常值 [2]，這些因素可能對線性迴歸模型的估計結果產生不良影響。為了增強迴歸模型對數據變異的穩健性，很多研究學者們提出了各種穩健性迴歸方法 [3]，在訓練資料中，異常值往往造成不佳的迴歸模型之參數值，進而影響迴歸模型的穩健性，因此在迴歸分析的研究中，我們應該特別關注其干擾模型的穩健之計算方法。文獻 [4] 中明確強調了處理訓練資料中異常值的三個方法，首先，第一種方法是透過預先處理數據（Data Preprocessing），排除異常值，但目前缺乏明確的標準來區分或過濾。第二種方法是樣本改進技術（Sample Polishing Techniques），其目的是在構建模型之前校正受損的數據，但這伴隨著時間大量的消耗，且僅適用於少量數據。第三種方法則是試圖增強訓練模型的穩健性。故在這次的作品中，我們嘗試挑戰要利用「自創的演算法」結合以上三種方法之優點，設計出一套增強迴歸模型的穩健性之演算法。本研究所需要自學探究與文獻蒐集的方面大致分為以下幾個方向：

（一）迴歸分析

迴歸分析是統計學中一種探討變數間關係的方法 [5]，其主要目的在於預測一個變數（被稱為因變數）如何受到一個或多個其他變數（被稱為自變數）的影響。透過建立一個數學模型，迴歸分析致力於描述和解釋因變數與自變數之間的互動關係，進而進行預測或推論。而依據自變數和因變數之間的關係，迴歸分析可區分為線性迴歸分析和非線性迴歸分析兩大類別。

（二）線性迴歸模型

在假設自變數和因變數之間存在線性關係的情況下，我們可以用線性方程表示自變數和因變數之間關係為$Y = \beta^T X + \varepsilon$，其中$Y$ 表示因變數，β 為一行矩陣$[\beta_0, \beta_1 \cdots, \beta_n]$，$\beta_0$ 為線性模型的截距，其餘則是各自變數的係數；X 是一行矩陣 $[1, x_1 \cdots, x_n]$，其中第一項是截距的係數，其餘是影響預測的自變數；ε 為實際數據與模擬數據之間的誤差（error）或殘差。當只有一個自變數 x 的模型時，常稱為簡單線性迴歸模型（Simple linear regression），其式子為 $y = \beta_0 + \beta_1 x + \varepsilon$，一般而言，當我們取得一組新的數據時，無法確定其 β_0 與 β_1，因此簡單線性迴歸的目標是找到最佳的 β_0 與 β_1 值，進而建立最符合資料的預測模型 [5]。

（三）最小平方法 （Least Squares Method，簡稱LSM）

最小平方法，或稱最小二乘法，是一種數學和統計方法 [5]，用於最小化觀測值與一個理論模型的殘差平方和，從而找到模型參數的最佳估計。我們需要先定義一個誤差函數，即殘差平方和（*RSS*），其公式如下:

$$RSS = \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)$$

將其上式之預測值展開成以下誤差函數:

$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i \right)^2$$

為了求此函數的最小值，會分別對 β_0 與 β_1 偏微分，並將其一階偏微分設為 0，可得

$$\begin{cases} \dfrac{\partial RSS}{\partial \beta_0} = -2 \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i \right) = 0 \\ \dfrac{\partial RSS}{\partial \beta_1} = -2 \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i \right) \cdot x_i = 0 \end{cases}$$

透過解此聯立方程，推出最佳的 β_0 與 β_1 值。

（四）最佳參數的演算法：Lasso 與 Ridge

在模型預測中，我們常常會發現模型與訓練數據產生過度擬合（Overfitting）的情況，為了避免此情況發生，會在模型擬合過程中加入額外的正則項（Regularization

term），以限制模型參數的大小，避免模型過度擬合於訓練數據，而偏離原模型預測目標的情況發生 [5]，其主要想法是在原本的最小平方法的誤差函數中加入正則項以達成正則化之目的，依據加入的正則項的不同，可列出對應的誤差函數如下：

$$RSS_{Lasso} = \sum_{i=1}^{n}\left(y_i - \beta_0 - \beta_1 x_i\right)^2 + \lambda\left|\beta_1\right|$$

$$RSS_{Ridge} = \sum_{i=1}^{n}\left(y_i - \beta_0 - \beta_1 x_i\right)^2 + \lambda\beta_1^{\,2}$$

其中 λ 是正則項的係數，可透過超參數調參方法 （Grid Search）並配合梯度下降法（Gradient descent）的計算來找出最佳的 λ 值。

（五）分治法（Divide and conquer）

在電腦科學計算中，分治法是一種用於解決排序、搜索和最優化等問題的重要方法 [6]，從其名稱可知就是把一個較難且複雜的問題分解成更小或更容易管理編輯的相似之子問題，接著分別處理各個子問題，最終將各個子問題的結果合併起來得到原本問題的解 [7]。本研究就是利用了「由繁化簡」的特性來推廣並結合機率的概念，設計出可以在大量的數據中判斷哪些為異常值並篩選出較正確的資料，以便幫助提高評估迴歸模型的穩健性之計算。

（六）Dijkstra演算法

Dijkstra演算法是一種用來找尋圖中最短路徑的方法 [8]，這個演算法的主要目標是找到一個點到圖中其他點的最短路徑，其演算法的基本步驟依序分別為初始化、選擇節點、鬆弛操作與標記節點等，並重複操作以上動作，即可找到每個節點的最短路徑 [9]，本研究就是利用了「最短路徑」的思維來延伸設計一套可以區分並排除異常值，進而提高迴歸模型的穩健性。

（七）K-近鄰演算法（K-NN Algorithm）

KNN是一種監督式學習方法，通常被用來做分類和迴歸。它的原理是基於相似的事物在特徵空間中通常靠近的概念 [9]。當有一個新的沒有標籤的數據點時，KNN會找到

最接近它的K個已經標記的數據點，然後根據這些鄰近點的標籤進行預測。使用KNN的步驟依序為選擇K值、選擇距離度量（例如歐氏距離、曼哈頓距離、切比雪夫距離與明可夫斯基距離等）、計算點與點的距離、識別最近的鄰近點與進行預測等。這裡我們就是利用「識別最近的鄰近點之距離」的特色來計算數據的密集程度，藉此設計出「自動過濾異常值」之演算法。

（八）模糊理論（Fuzzy Theory）

Fuzzy 一詞即指模糊的事物，模糊理論是一門處理不確定性和模糊性的數學工具和理論，根據 [10]，有提及其主要目標是處理那些難以用傳統的精確邏輯或集合理論準確描述的問題，此理論在各個領域都有廣泛的應用，包括人工智慧、控制系統、模式識別與信息檢索等，我們利用了此領域所教的知識「模糊集合」與「隸屬函數」設計一個「數據資料本身自動給予權重」的方法，此方法可弱化異常值對於迴歸模型之影響，大大地提高其模型的穩健性。

（九）評估模型的穩健性之指標

根據 [11] 與 [12] 之模型穩健性的相關討論中，有提及到均方根誤差（Root Mean Squared Error, RMSE）是一種用來評估線性迴歸模型預測能力的指標，而決定係數（R-squared）通常用來評估線性迴歸模型的擬合程度，如果單獨地使用RMSE值或是R-squared值並不能直接衡量模型的穩健性，必須要透過對比不同模型的RMSE值或是R-squared值，因此我們將會呈現不同模型的研究結果，以凸顯我們對於提高穩健性之演算法的優勢。

貳、 研究設備與器材

一、硬體部分： 筆記型電腦、桌上型電腦。配備：Intel Core i5。

二、軟體部分：

（一）Google Colab

Google Colab 是由 Google 免費提供的、基於 Jupyter Notebook 技術的線上開源交互性編成環境。Google Colab 允許使用者使用其 Google 帳號登入使用，因此他可以輕易地使用雲端硬碟保存 Colab 筆記本，以方便使用者訪問筆記本或從雲端硬碟載入資料，也允許其他使用者共同編輯 Colab 筆記本。Colab 提供免費的 GPU（圖形處理器）資源，這對於需要大量計算的模型的任務來說非常有用，且 Colab 支援眾多流行的 Python 庫和框架，如 Matplotlib、Numpy、Keras 等，使用者可以方便地進行機器學習和深度學習的開發。

（二）Python 3.10.12 函數、計算與繪圖等套件庫

1. Numpy：此為進行機器學習的基礎函式庫，支援了大量的函式以便於進行矩陣等複雜的數學運算。

2. Pandas：此為一個開源且強大的數據分析庫，其提供了易於使用的數據結構和數據分析工具，常用於處理和分析表格型數據，如 CSV 檔案（Comma-Separated Values，逗號分隔值）、Excel 表格、SQL 數據庫等。

3. Matplotlib：這是 Python 當中用來視覺化數據、進行表格整理和繪圖最重要的程式庫之一。它提供了多種繪圖選項，能夠生成各種不同類型的圖表，包括折線圖、散點圖、直方圖、餅圖等。Matplotlib 的靈活性和豐富的功能使得它常用於科學計算、數據分析和機器學習等領域中。

4. Sklearn：這是建立在 Numpy 之上的函式庫，內建了許多有關於機器學習的演算法，例如：分類數據、線性迴歸和模型的檢查都屬 sklearn 的範疇之內。以下的演算法將會是作為我們研究的演算法之對比。

   （1）KNN：一種選擇最近的 k 個樣本以進行模型的訓練並預測數值之演算法。

   （2）Linear Regression：一種處理自變數、應變數之關係以用於預測模型之演算法。

（3）Lasso：和 Linear Regression 的方法大致相同，但其多了懲罰項（正則項以絕對值呈現），以用於避免過度擬合。

（4）Ridge：和 Lasso 類似，僅正則向以平方表示。

5. SciPy（Scientific Python）：此為一個建立在 NumPy 基礎上的開源科學計算庫，提供了許多用於解決科學和工程問題的高級功能。它擴展了 NumPy 的功能，包括優化的線性代數、積分、優化、訊號處理、統計等模塊。

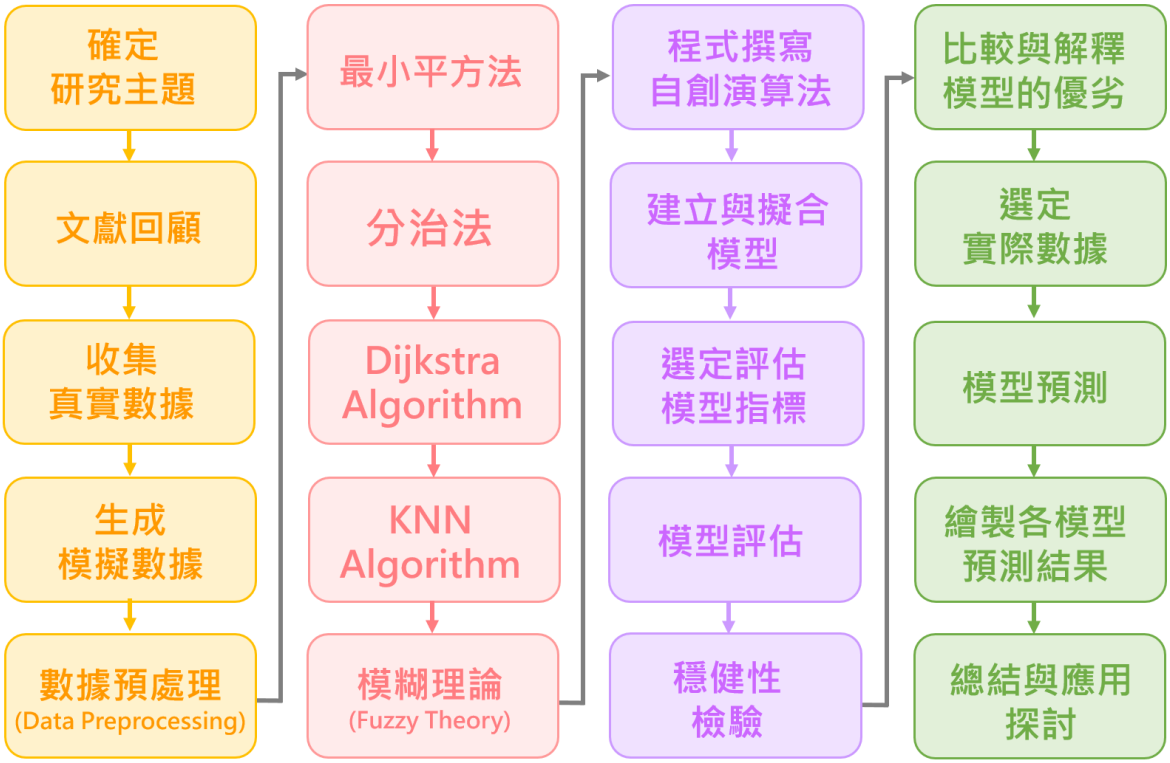
6. 其他標準函式庫：如 random、statistics 等 Python 的內建免安裝函式庫。

（三）Kaggle 數據資料：

Kaggle 是全球最大的數據科學和機器學習交流網站平台之一，該網站提供了豐富的數據集，其涵蓋各種領域，並提供用戶進行數據分析、模型訓練和預測用。

參、 研究過程或方法

一、研究架構圖

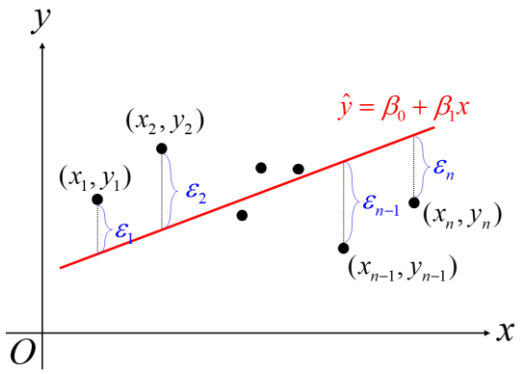


圖一：研究架構圖。

二、研究最小平方法（Least Squares Method）與探討異常值對其之影響

此方法是統計學和數學中一種廣泛應用的迴歸分析方法，特別是在線性迴歸模型中。這種方法的目的在於找到一條最佳擬合線，使模型預測值（Predicted Values）和實際觀測值（Observed Values）之間的平方誤差的總和能最小化，進而找到最佳擬合模型的參數。

假設在 n 組二維的數據 (x_i, y_i)，$i = 1, 2, \cdots, n$ 中，我們希望找到一條最佳擬合的直線 $y = \beta_0 + \beta_1 x$，其中 β_0 為直線的截距，β_1 為直線的斜率，如圖二所示。



圖二：數據資料與最適直線（迴歸直線）。

先將這 n 組數據代入此直線，可得模型預測值 $y_i = \beta_0 + \beta_1 x_i$，$i = 1, 2, \cdots, n$，接著考慮實際觀測值 y_i 與模型預測值 y_i 之間的殘差，記為 $\varepsilon_i = y_i - y_i$，$i = 1, 2, \cdots, n$，我們的目標為透過解 $\sum_{i=1}^{n} (\varepsilon_i)^2$ 之最小值時，進一步來估計未知模型之最佳的參數 β_0 與 β_1。然而最小平方法的一個重大缺點是對異常值非常敏感，即在數據中若存在極端值（或稱為離群點，Outliers）時，模型容易受到異常值的影響，進而影響參數的估計值，為了驗證其異常值對於模型的影響，我們開始撰寫程式來呈現其影響，首先隨機生成 45 組二維數據資料與 5 組離群點，如圖三所示。

```
#generate datas
m = 45
x = 2 * np.random.rand(m)
y = 10 + 2 * x + np.random.randn(m)
data = np.vstack((x, y)).T

#add outliers
m1 = 5
x1 = np.random.rand(m1)
y1 = np.random.rand(m1)
data_outliers = np.vstack((x1, y1)).T
```

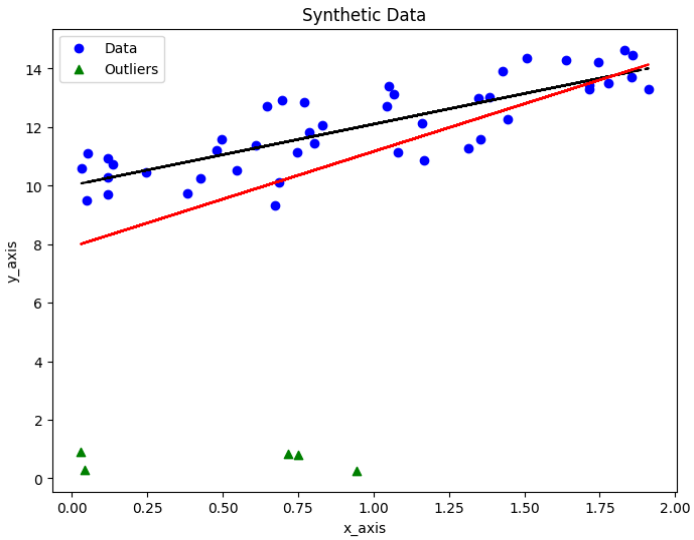
圖三：隨機生成 45 組數據與 5 組異常值之程式碼。

再利用最小平方法分別計算無包含異常值的數據與有包含異常的數據之迴歸直線，程式碼如圖四所示。

```
#Without outliers
ls_result = optimization.leastsq(func, thelda, args=(X,y))
intercept = ls_result[0][0]
slope = ls_result[0][1]
print(f"[Without outliers] intercept: {intercept:.3f}, slope: {slope:.3f}")
plt.plot(x, slope*x + intercept,'k--',label = 'Without outliers')

#With outliers
ls_result_ = optimization.leastsq(func, thelda_, args=(X_,y_))
intercept_ = ls_result_[0][0]
slope_ = ls_result_[0][1]
print(f"[With outliers] intercept: {intercept_:.3f}, slope: {slope_:.3f}")
plt.plot(x_, slope_*x_ + intercept_,'red',label = 'With outliers')
```

圖四：利用統計套件的最小平方法函數分別計算有、無包含異常值的迴歸直線之程式碼。

並在二維平面上分別繪製兩條直線，其中黑色的直線是透過原 45 組的數據資料所生成的迴歸直線，另外一條紅色的直線為考慮 45 組的數據與 5 組離群的數據所生成的迴歸直線，可以清楚地觀察到紅色的迴歸直線受到 5 個極端值的影響，出現被往下偏移的行為，即解出表現較不佳的直線截距 β_0 與斜率 β_1，如圖五所示。



圖五：分別繪製有、無包含異常值之兩條迴歸直線。

因此，在實際應用中，我們需要特別留意異常值對模型的潛在影響，並考慮使用其他方法，如穩健迴歸，以提高模型對異常值的穩健性。不過最小平方法畢竟是一個非常重要、經典而有效的迴歸分析方法，在理論和實踐中都具有廣泛的應用價值。

三、探究分治法（Divide and conquer）並計算點出現的頻率，進而篩選正常的數據資料

分治法常見於解決許多計算機科學和數學中的問題，例如:排序算法（包括快速排序、合併排序）、搜索算法（包括二分搜索）、圖論算法（包括最接近點對問題）等。透過將問題分解成更小的部分，此法可以有效地減少問題的複雜性，提高解決計算過程的效率，常見的三個步驟為分割、解決和合併，對於這次的研究，我們要處理大量的數據時，開始撰寫程式進行資料切割，處理過程就是在每次的測試中，先隨機選取樣，並紀錄資料點的標籤，最後去計算數據點出現的頻率，一般來說，異常值的數量偏少，因此在取樣的時候，出現的機率較低，當我們操作最後的統計時，可以達到篩選出正常的數據之效果，我們結合分治法的核心思維與統計隨機的機率來延伸設計一個新的演算法。

四、探究 Dijkstra 演算法並應用於區分異常值的存在

此法的命名來自荷蘭的計算機科學家艾茲赫爾·迪克斯特拉（Edsger Dijkstra），是一種用於解決最短路徑問題的經典演算法，而最短路徑問題的目標是找到從一個起點到所有其他節點的最短路徑，或是找到到特定目標節點的最短路徑，該演算法使用了一個稱為「距離」或「權重」的概念，用來記錄從起點到每個節點的目前最短路徑的長度，常見的步驟分解如下：

（一）初始化：將起點的距離一開始設定為零，其他節點的距離設定為為無窮大，起點的前一節點為空值（NULL），程式碼如圖六所示。

```
#  取得節點的數量
num_nodes  =  len(graph)

#  初始化距離陣列，起點到每個節點的距離初始值為無窮大
distances  =  [float('inf')]  *  num_nodes

#  起點到自己的距離為0
distances[start]  =  0

#  初始化訪問標誌陣列
visited  =  [False]  *  num_nodes
```

圖六：初始化所有點之程式碼。

（二）選擇最近節點：將尚未處理的節點中選擇距離最短的節點，並將其標記為已處理過，程式碼如圖七所示。

```
# 找到目前未訪問節點中距離最短的節點
min_distance = float('inf')
min_index = -1
for i in range(num_nodes):
        if not visited[i] and distances[i] < min_distance:
                min_distance = distances[i]
                min_index = i

# 標記該節點為已訪問
visited[min_index] = True
```

圖七：利用迴圈找找尋並標記尚未處理與已處理的點之程式碼。

（三）更新距離：對於與當前節點相鄰的未處理過之節點，更新它們的距離，如果從起點經過當前節點到達它們的距離比目前已知的距離短，則更新其距離，程式碼如圖八所示。

```
# 更新與該節點相鄰的節點的距離
for i in range(num_nodes):
        if not visited[i] and graph[min_index][i] > 0:
                new_distance = distances[min_index] + graph[min_index][i]
                if new_distance < distances[i]:
                        distances[i] = new_distance
```

圖八：更新已標記處理過的點與點的距離之程式碼。

（四）重複動作：重複步驟（二）和步驟（三），直到所有節點都被處理過，即可計算出所有節點的最短路徑。

Dijkstra 演算法確保每個節點的最短路徑在選擇最近節點時都是最適當的，其前提是它要求所有邊的權重必須為非負值，此法在求解最短路徑問題上是非常有效的；然而在處理大型圖時，可能會面臨存儲和計算效率的挑戰；不過對於此次的研究而言，我們就是要利用其可算出「最短路徑」的思維來延伸設計一套可以區分並排除異常值，因為在一般的二維數據資料中，往往只會存在少量的異常值，當我們考慮資料中點與點的距離時，異常值的點會造成「較長的路徑」之影響，因此再將其距離偏大的資料點捨去，就可以達到擷取到真正較有價值且可用的資料，進而提高評估模型的穩健性。

五、探究 K-近鄰演算（K-NN Algorithm）法並應用其計算結果於異常值的權重之設計

K-近鄰演算法（K-Nearest Neighbors，簡稱 KNN）是一種廣泛應用於分類與迴歸問題之常見的監督式機器學習算法。其基本思想是利用周圍鄰近的資料點的信息來預測新數據點的標籤或值。在 KNN 中，K 表示鄰近的鄰居數量。在解決分類的問題上，算法尋找與新數據點最相似的 K 個訓練數據點，然後基於這 K 個鄰居的標籤進行多數投票決定新數據點的分類。然而應用於迴歸的問題中，則是計算 K 個最相似鄰居的平均值作為新數據點的預測值。，此演算法相對好理解與操作，主要包括以下步驟：

（一）先決定鄰近數量 K 值：K 值的選擇對 KNN 的性能至關重要。較小的 K 值使模型對異常值更敏感，而較大的 K 值可能使模型過於平滑。選擇一個合適的 K 值通常需要通過交叉驗證（Cross-Validation）等方法進行調參，而此次模擬數據實驗時，我們選擇先設定 K 值為 5，在程式碼中，如圖九所示，其值為 6 的原因是為了要計算一個點到其他鄰近的 5 個資料點的距離，其值包含本身的點到本身的距離為 0，也必須要存取在各點之距離的陣列中。

（二）計算距離：對於每一個新數據點，計算它與訓練數據集中每個樣本點的距離。距離可以使用歐氏距離、曼哈頓距離或其他度量方式來衡量，這裡是選用我們熟悉的歐氏距離來計算點與點之間的距離。

（三）鄰近資料點的選擇：選擇距離最近的 K 個樣本點，這些樣本點即為新數據點的鄰近鄰居，這裡是利用 sklearn.neighbors 套件下的函數 NearestNeighbors（）來幫我們完成計算，再將其結果存放在變數 result 中，如圖九所示。

```
k = 6 #include self
result = NearestNeighbors(n_neighbors=k)
result.fit(data_comb)
```

圖九：決定鄰近數量 K 值之程式碼。

（四）進行分類或迴歸：對於分類的問題時，根據 K 個鄰居的多數決定新數據點的分類；對於迴歸的問題時，計算 K 個鄰居的平均值作為新數據點的預測值。

KNN 演算法的優點之一就是簡單易懂，無需進行模型的訓練，並且適用於多類別問題。不過在這次研究中，我們不會用來解決分類或迴歸等問題，而是要利用此演算法計算出來的各點到其他點的距離之結果，要將其推廣於設計異常值的權重，我們希望在整個數據資料中，能大幅降低異常值的影響，即會給予其「較低的權重」，簡單來說就是加權平均的概念，越不重要的資料點，越是要讓它在整個評估迴歸模型時的參與度盡可能減少或甚至消失，進而達到提高評估模型的穩健性。

首先我們繼續利用原先隨機生成 45 組數據與 5 組異常值之數據資料來操作 KNN 演算法，為了視覺化每個資料點到其他點的距離，我們利用迴圈隨機選色，讓點與點的距離呈現不同的顏色，程式碼如圖十所示。

```
#make connect line different colors
number_of_colors = m+ml
color = ["#"+''.join([random.choice('0123456789ABCDEF') for j in range(6)])
                        for i in range(number_of_colors)]
```

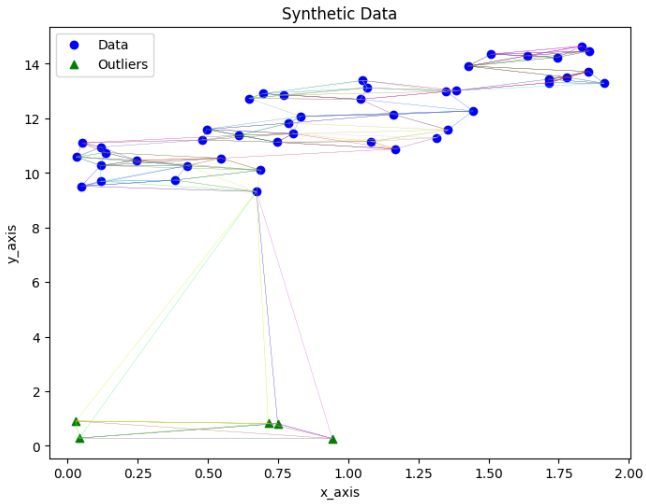
圖十：撰寫迴圈隨機將點與點之間的距離配色之程式碼。

接著再設計涵數 connectpoints（），將 KNN 演算法所算出來的點與點之間的距離之結果代入此函數中，程式碼如圖十一所示。

```
def connectpoints(x,y,p1,p2):
        x1, x2 = x[p1], x[p2]
        y1, y2 = y[p1], y[p2]
        plt.plot([x1,x2],[y1,y2],color=color[p1],linewidth=0.2)
i=0
j=1

while i < m+ml:
        while j < k:
                connectpoints(x_,y_,i,result.kneighbors([data_comb[i]])[1][0][j])
                j=j+1
        i=i+1
        j=1
```

圖十一：使用涵數 connectpoints（）描繪距離之程式碼。

最後再將其連線後的結果繪製出來，如圖十二所示。因為是代入 KNN 演算法所算出的結果並存入多維變數 result 中，且當初選定的 K 值為 5，所以可以清楚地觀察到每個點所連線出去於鄰近的距離都皆有五條線，同時也將 KNN 演算法所算出的每五條線之歐氏距離值存入在變數 result[0][0]中，以便於我們之後計算每個資料點的權重用。



圖十二：利用 KNN 演算法視覺化數據資料點與點的距離。

六、探究模糊理論並設計出「模糊集合」與「隸屬函數」

模糊理論提供一個強大且靈活的數學框架，用於處理現實世界中的不確定性和模糊性。其理論的基本思維是模擬人類認知中的模糊性，將不確定性引入數學模型，得以更好地處理模糊和模糊信息，例如: 評價一家餐廳的服務品質，傳統的評價方式可能是使用具體的數字或詞語，例如「服務很好」、「食物很美味」等，但這樣的描述可能仍然無法完全涵蓋人們對整體餐廳體驗的感受，因為這些描述通常是非常主觀的，可能認為服務品質在「好」和「非常好」之間，但不確定具體屬於哪一個，這時就可以應用模糊理論提出的「模糊集合（Fuzzy Set）」與「隸屬函數（Membership Function）」來進行「較彈性的空間」來討論人類的感受。我們研讀與探究了比較重要且實用的觀念，將其探究的內容整理如下：

（一）模糊集合

模糊集合可說是此理論最核心的基本且重要的觀念，其集合允許元素「部分地」屬於集合，而不是侷限於傳統集合（Crisp Set）理論那樣以二進制方式，即所謂的 0 或 1，分別對應表示完全屬於或不屬於兩種，例如：假設有一集合 $A=\left\{x \mid x^2-3x+2=0\right\}$，則 $1 \in A$ 但 $100 \notin A$。然而在模糊集合中，每個元素都有一個屬於該集合的程度，稱為「隸屬度（Membership Degree）」，其值的範圍通常為 [0, 1]，此範圍的度量允許元素同時屬於多個集合，並且可以用連續的方式表示模糊性，這一概念在描述真實世界中的模糊和不確定性問題時非常有用。我們再次以評價一家餐廳的服務品質為例，使用模糊集合來描述不同層次的服務品質，例如「非常差」、「差」、「一般」、「好」、「非常好」。每個層

次都有一個相應的隸屬度，表示人們對該層次的評價程度，當評價服務品質時，就可以同時給予多個層次的歸屬度。

（二）隸屬函數

隸屬函數，常以數學符號α_A表示，是模糊集合中的一個重要元素，它描述了元素對於模糊集合的隸屬程度，其值通常在 [0, 1] 的範圍內，記為$\alpha_A(x): X \to [0,1]$，而隸屬函數的常見圖形為三角形或梯形型，這樣的函數能夠更靈活地建模不同元素的隸屬度，以下是我們簡單地設計隸屬函數，用以描述「服務品質」這個模糊集合，整理如下：

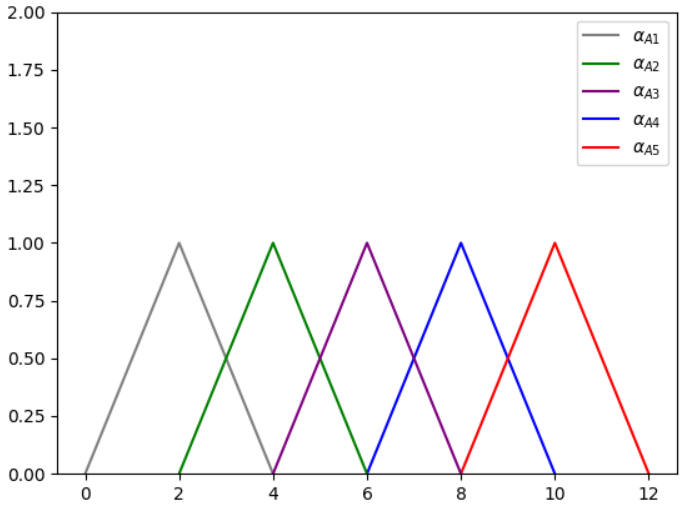
1. 非常差：隸屬函數圖形為三角形型，峰值位置（peak）為 2，左邊界（left base）為 0，右邊界（right base）為 4。
2. 差：隸屬函數圖形為三角形型，峰值位置為 4，左邊界為 2，右邊界為 6。
3. 一般：隸屬函數圖形為三角形型，峰值位置為 6，左邊界為 4，右邊界為 8。
4. 好：隸屬函數圖形為三角形型，峰值位置為 8，左邊界為 6，右邊界為 10。
5. 非常好：隸屬函數圖形為三角形型，峰值位置為 10，左邊界為 8，右邊界為 12。

我們統一把以上五個對應的模糊集合$A_1, A_2, \cdots, A_5$與隸屬函數$\alpha_{A_1}, \alpha_{A_2}, \cdots, \alpha_{A_5}$，以數學的形式整理如下:

$$\alpha_{A_1}(x)=\begin{cases}\frac{1}{2}x & ,0\le x<2\\ 1 & ,x=2\\ -\frac{1}{2}(x-4) & ,2<x\le 4\end{cases},\quad \alpha_{A_2}(x)=\begin{cases}\frac{1}{2}(x-2) & ,2\le x<4\\ 1 & ,x=4\\ -\frac{1}{2}(x-6) & ,4<x\le 6\end{cases},\quad \alpha_{A_3}(x)=\begin{cases}\frac{1}{2}(x-4) & ,4\le x<6\\ 1 & ,x=6\\ -\frac{1}{2}(x-8) & ,6<x\le 8\end{cases}$$

$$\alpha_{A_4}(x)=\begin{cases}\frac{1}{2}(x-6) & ,6\le x<8\\ 1 & ,x=8\\ -\frac{1}{2}(x-10) & ,8<x\le 10\end{cases},\quad \alpha_{A_5}(x)=\begin{cases}\frac{1}{2}(x-8) & ,8\le x<10\\ 1 & ,x=10\\ -\frac{1}{2}(x-12) & ,10<x\le 12\end{cases}$$

並繪製在圖一個二維平面上，可以觀察到，服務品質分為五個層次，但每個層次都可以用隸屬函數來描述其品質的層度，而不再只是 2、4、6、8 與 10 等五個評分值，如圖十三所示。



圖十三：用來描述「服務品質」的五個層次之隸屬函數圖形。

七、設計不同的評估線性迴歸模型之演算法

有了以上的學習與探究後，我們開始整合所有觀念與思維並推廣延伸應用於設計出三種提高模型的穩健性之演算法，將自創的演算法與其設計的原理分別整理如下：

（一） Randomized Divide and Conquer algorithm

我們將此演算法命名為「隨機分治演算法（Randomized Divide and Conquer algorithm）」，簡寫為 RDCA，一組真實數據可以分成異常值以及原始數據，前者通常是反常且在真實數據中不常見的觀測值，這些值極端偏離原始數據。因此若整理異常值與其他數據的距離，擁有較大距離的數值往往是數據中的異常值，且因其極端的性質，也必定在真實數據中佔絕對少數。因此利用分治法的概念，我們便可將有關辨識異常值之複雜問題簡化成有關分析數據與數據間距離之簡單問題。

我們首先從真實數據中隨機取樣並計算數據與數據之間的距離，之所以隨機取樣是為了減少對所有數據大量計算而消耗的時間成本，對比計算所有數據點到彼此之間的距離，只計算取樣之距離可以將時間複雜度從$O\left(n^2\right)$降低至$O\left(k(c-1)\right)$，其中 k 為取樣次數，c 為每次取樣之取樣數量，將每次取樣之取樣數量單獨提出是必要的，因為每次取樣必須取出不同的數據，才能體現每次取樣對不同元素之機率獨立，程式碼如圖十四。

```
c = int(1E+3)    # Test times
k = 25

def distance(x1, y1, x2, y2):
        return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
for i in range(c):
        #Find diffrent point
        a_point = [0 for i in range(k)]
        for j in range(k):
                a_point[k] = random.randint(0,len(data)-1)
        while len(set(a_point))!=len(a_point):
                for j in range(k):
                    a_point[k] = random.randint(0,len(data)-1)

        #calculate their total distance
        for j in range(len(a_point)-1):
                ds[i][0] += distance(data[a_point[j]][0], data[a_point[j]][1], data[a_point[j+1]][0], data[a_point[j+1]][1])

        ds[i][1] = a_point
```

圖十四：隨機取樣所需的 *k* 與 *c* 值並利用迴圈操作資料切割。

其中關於 *k* 與 *c* 值之討論尤為重要，這攸關模型的準確性。一般來說，當 *k* 和 *c* 值越大，模型的準確性越高，但花費時間也會增加，因此選擇一組效率高且不失準確性的 *k* 和 *c* 值是此演算法之重點，以下兩個表格分別是以不同的 *k* 和 *c* 值之組合計算出的 RMSE 值與 R-squared 值，如表一與表二所示，可清楚地觀察出當 *k* 為 25，*c* 值為 1000 時，RMSE 值最小，R-squared 值大於 0.5（其值越趨近於 1，表示評估模型的能力較佳），因此我們將選用兩值作為試驗之取樣係數的初始設定。

表一：依據不同的 *k* 與 *c* 值之組合進行 RMSE 值的比較。

	***c* = 100**	***c* = 1000**
***k* = 10**	0.99	1.01
***k* = 25**	1.05	0.87

表二：依據不同的 *k* 與 *c* 值之組合進行 R-squared 值的比較。

	***c* = 100**	***c* = 1000**
***k* = 10**	0.47	0.47
***k* = 25**	0.48	0.62

選定好 *k* 與 *c* 值之組合後，再進行資料隨機的切割，每次切割的過程中順便去計算取樣後所有資料點的距離總和與紀錄被選取到的點之標籤，以便後續計算其出現頻率，結果如圖十五所示。

	sum_distance	points
test_1	41.303128	[43, 37, 39, 12, 15, 16, 7, 2, 10, 23, 8, 40, ...
test_2	79.074690	[35, 8, 17, 2, 49, 10, 36, 16, 44, 15, 4, 24, ...
test_3	83.072915	[22, 26, 9, 14, 47, 27, 12, 5, 43, 41, 3, 33, ...
test_4	80.484983	[34, 13, 29, 14, 26, 18, 6, 42, 28, 30, 22, 45...
test_5	99.969250	[13, 25, 40, 47, 11, 49, 48, 29, 0, 33, 42, 27...
...	...	...
test_96	94.351534	[28, 25, 23, 6, 49, 13, 38, 0, 2, 18, 30, 35, ...
test_97	93.056615	[12, 29, 10, 38, 9, 33, 49, 14, 1, 16, 3, 20, ...
test_98	118.172832	[11, 20, 16, 25, 28, 27, 9, 48, 35, 10, 21, 24...
test_99	120.017669	[12, 30, 21, 33, 43, 18, 49, 35, 44, 29, 17, 0...
test_100	127.874137	[19, 23, 8, 30, 49, 29, 25, 10, 4, 43, 32, 0, ...

圖十五：隨機取樣進行資料分割的結果。

再將取樣後的加總距離，代入以下我們設計的公式，程式撰寫如圖十六所示，以作為該次取樣所得之權重。

```
def  Wg(a1,total):
    return  1  -  round(a1/total,6)
```

圖十六：隨機取樣資料點的加總距離之加權公式。

其中變數 al 存取該次取樣之加總的距離，變數 total 存取全部取樣之加總的距離。最後使用陣列變數 A_Q 儲存每次測試後取得之權重加總，再取其中較大權重者，即可獲得較正常的數據資料點，做為新的模型預測對象，便可以利用此一演算法排除部分異常值以及不符合原始數據之部分數據，以達到篩選出我們想要的原始數據之目的。

（二）Self-Spanned Connected Graph algorithm

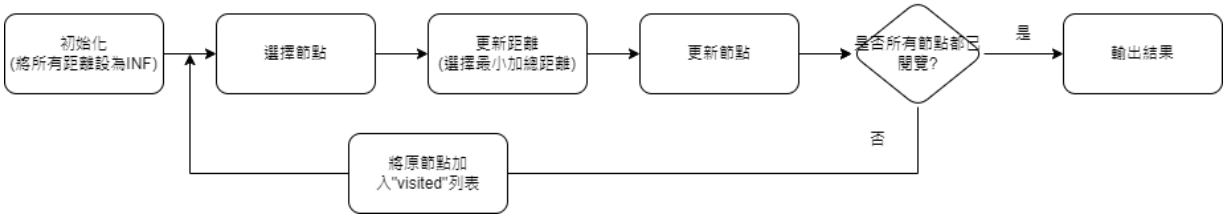
我們將此演算法命名為「自我生成連通圖演算法（Self-Spanned Connected Graph algorithm）」，簡寫為 SSCGA，在此演算法，先給定一組二維數據資料中，讓各點之間可自動產生一條邊（edge），進而生成一個資料的連通圖，而要實現與完成產生邊的過程，我們可以利用 KNN 演算法，其中 K 值設定為所有資料總數的一半，此設計是為了確保圖的連通性，否則會違反連通圖的假設，且在此演算法中，最佳的執行時間複雜度為 $O\left(\frac{n^2}{2}\right)$，也就是一次搜尋便得到結果；最差為 $O(n!)$，也就是最後一次搜尋才得到結果，無論如何，使用此演算法皆優於使用窮舉法的 $O(n!)$，進而加速之後 Dijkstra 演算法於此連通圖中算出所有點與點之間的距離總和之最小值，最後進行比較其他較大的距離總和所行徑的資料點，進一步即可區分哪些為異常值並給予排除。

首先 KNN 演算法會選取距離目標數據最近的點，我們便利用此特性來創建連通圖，程式撰寫如圖十七所示，此舉意味著每個數據在連通圖中只會和與他最近的一半數據點相連線，而異常值擁有與原始距離相對長的距離，此步驟可以切斷與其他數據點與異常值之連線，並初步地減少其對模型的影響。

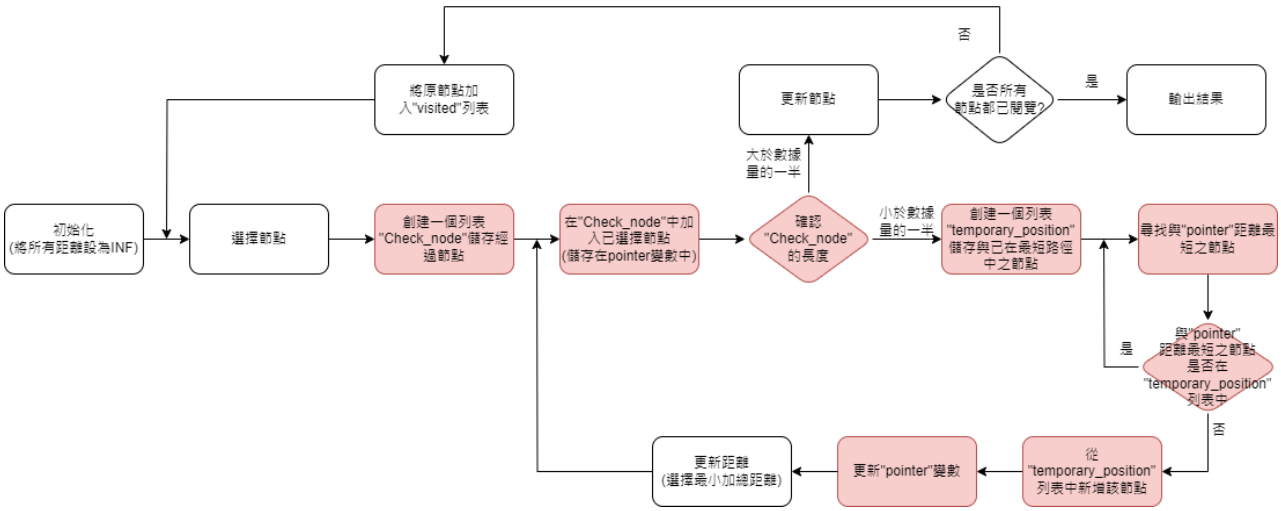
```
k = int(len(data)/2)
result = NearestNeighbors(n_neighbors=k)
result.fit(data)
graph = np.zeros((len(data),2,k))

for i in range(len(data)):
    a = np.array(result.kneighbors(data[i].reshape((1,-1)))[0])
    b = np.array(result.kneighbors(data[i].reshape((1,-1)))[1])
    graph[i][0],graph[i][1] = a , b
for i in range(len(data)):
    graph[i][0][0] = np.inf
for i in range(len(data)):
    for j in range(len(graph[i][1])):
        graph[i][1][j] = int(graph[i][1][j])
print(graph)
```

圖十七：建立連通圖之程式碼。

有了連通圖以後，便可將後續的計算交給「我們改良的 Dijkstra 演算法」，由於原本的 Dijkstra 演算法，其流程如圖十八所示，只會專注於最短路徑，當計算最短路徑時，並不顧及所經過之節點數量，在處理真實數據時會因經過數據過少而使模型失真，所以我們在 Dijkstra 演算法中加入一個條件，即改良的 Dijkstra 演算法必須經過一半的數據點才得以輸出結果，如此一來便可解決因參考數據過少而導致模型失真的情況發生，其流程如圖十九所示。



圖十八：原本的 Dijkstra 演算法之計算流程圖。



圖十九：SSCGA 中的改良 Dijkstra 演算法之計算流程圖。

其中圖十九中的紅色區塊的對應程式碼如圖二十所示。

```
    for  i  in  graph[index][1]:
        Check_node.append(index)
        pointer  =  i
        new_cost  =  0
        temporary_position  =  []
        temporary_position.append(pointer)
        while  len(Check_node)  !=  k:
            bi  =  0
            for  ci  in  graph[int(pointer)][1]:

                if  ci  in  temporary_position:
                    bi  +=  1
                    continue
                else:
                    temporary_position.append(ci)
                    Check_node.append(ci)
                    new_cost  +=  graph[int(pointer)][0][int(bi)]        #新路徑距離
                    pointer  =  ci
                    break

                bi  +=  1
            if    new_cost  <  nodes[i]:                          #  新路徑如果比較短
                  nodes[i]  =  new_cost                          #  採用新路徑
                  path[i]  =  Check_node    #  更新節點的路徑

        a  +=  1
        Check_node  =  []
    next  =  INF
    for  n  in  nodes:                                             #  從串列中找出下一個節點
        if  n  in  visited:                                       #  如果已拜訪回到for選下一個
                continue
        if  nodes[n]  <  next:                                   #  找出新的最小權重節點
                next  =  nodes[n]
                index  =  n
return  nodes  ,  path
```

圖二十：改良 Dijkstra 演算法之所需的條件確認程式碼。

如此便可以使用改良 Dijkstra 演算法計算該連通圖中的最短距離，並輸出數據量的一半節點，當起始的節點選擇為原始數據時，進而以最短距離找出其中最佳的一半數據，又可以避免模型失真的情況發生。

（三）Self-Generated Weighting algorithm

我們將此演算法命名為「自我生成權重演算法（Self-Generated Weighting algorithm）」，簡寫為 SGWA，顧名思義就是在給定一組二維數據資料中，各點本身可自動計算自己的權重，我們都知道異常值對於評估線性迴歸模型會造成不良的影響，因此我們盡可能要減低或甚至消弭其對於評估過程的影響，最核心的思維就是針對每個資料點，給予一個參與計算的程度，即為權重值，因為異常值（或稱離群點）為一群少數與其他數據點相比時具有顯著不同或偏離常態分佈的觀測值，所以要給予它們較低的權重，讓一般正常且常態分佈的資料得以凸顯其重要性，進一步地再計算迴歸模型時能大幅地提高其穩健性，因此如何針對每個資料點，給定較高或較低的權重值，是這個演算法最重要的關鍵，我們要透過模糊理論所教的隸屬函數來實現。

一開始假設有 n 組二維的數據(x_i, y_i)，$i=1,2,\cdots,n$，在這些數據資料點中，首先先利用 KNN 演算法，K 值設定為 5，設d_i，$i=1,2,\cdots,n$ 為任選一點到其他鄰近五個資料點之歐氏距離總和並建立一個有關距離總和之集合 $S=\{d_1,d_2,\cdots,d_n\}$ 與一個模糊集合 $F=\{(d,\alpha)|d\in S,\alpha\in[0,1]\}$，在撰寫程式碼中，我們用陣列變數 data_dist_sum 並利用兩層的迴圈與 KNN 演算法所算出的結果來存取所有 d_i 值，如圖二十一所示。

```
#knn_calculate  a  point  to  other  5  points  distance  sum
i=0
j=1
data_dist_sum  =  np.zeros(m+m1)

while  i  <  m+m1:
        while  j  <=  k-1:
                data_dist_sum[i]  =  data_dist_sum[i]  +  result.kneighbors([data_comb[i]])[0][0][j]
                j=j+1
        i=i+1
        j=1
```

圖二十一：建立有關距離總和的模糊集合之程式碼。

接下來針對剛建立的模糊集合 F，我們要設計一個對應集合的隸屬函數 $\alpha : S \to [0,1]$，定義如下：

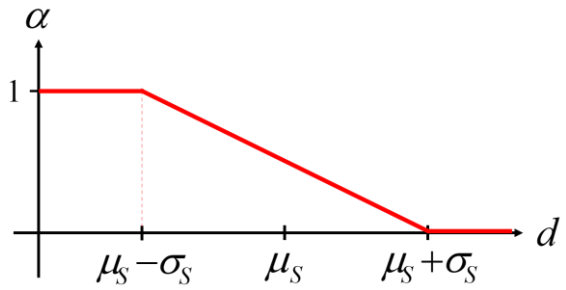
$$\alpha(d) := \begin{cases} 1 & , d \le \mu_S - \sigma_S \\ \dfrac{\mu_S + \sigma_S - d}{2\sigma_S} & , \mu_S - \sigma_S < d \le \mu_S + \sigma_S \\ 0 & , \mu_S + \sigma_S < d \end{cases}$$

其中 μ_S 為所有 $d_i \in S$，$i = 1,2,\cdots,n$，的算術平均數，σ_S 為所有 $d_i \in S$，$i = 1,2,\cdots,n$，的母體標準差，在程式碼中，如圖二十二所示，我們先使用 statistics 套件，協助計算 d_i 的算術平均數與母體標準差後，再利用條件判斷，當 d_i 值小於 $\mu_S - \sigma_S$ 時，我們給定權重值為 1，當 d_i 值介於 $\mu_S - \sigma_S$ 與 $\mu_S + \sigma_S$ 時，給定權重值為 $\dfrac{\mu_S + \sigma_S - d}{2\sigma_S}$，當 d_i 值大於 $\mu_S + \sigma_S$ 時，給定權重值為 0，最後用陣列變數 alpha 來存取所有 d_i 值對應的權重值。

```
#define  membership  function
t=0
s=0
alpha  =  np.zeros(m+m1)
mu  =  statistics.mean(data_dist_sum)
std=  statistics.stdev(data_dist_sum)

while  t  <  m+m1:
        if  data_dist_sum[t]  <=  mu-std:
                alpha[t]  =  1
        elif  mu-std  <  data_dist_sum[t]  <=  mu+std:
                alpha[t]  =  ((mu+std)  -  data_dist_sum[t])/(2*std)
        else:
                alpha[t]  =  0
        t=t+1
```

圖二十二：定義距離的隸屬函數之程式碼。

我們透過以上的隸屬函數之設計為的是可以確保異常值的資料所計算出來的 d_i 值必定大於 $\mu_S + \sigma_S$，因此其對應的權重值設定為 0，即可以直接消弭對於評估模型的計算，但如果是符合常態分佈的資料點，則會落在 $\mu_S - \sigma_S$ 與 $\mu_S + \sigma_S$ 之間或小於 $\mu_S - \sigma_S$，我們會希望保留其計算的重要性，給予非零的權重值，即可完成每個資料點都可以達到自我生成權重的能力，為了方便觀察隸屬函數圖形之構造的正確性，我們將函數圖形繪製出來，其圖形屬於梯形型，如圖二十三所示。



圖二十三：所有距離對應的隸屬函數之圖形。

一般簡單的線性迴歸模型為 $y=\beta_0+\beta_1 x$ ，現在有了隸屬函數所生成的權重 α 後，我們可以來建構新的線性迴歸模型為 $y=\alpha(\beta_0+\beta_1 x)+(1-\alpha)y$，最後再透過最小平方法，當我們解出 $\min\sum_{i=1}^{n}(y_i-y_i)$，其中 $y_i=\alpha(\beta_0+\beta_1 x_i)+(1-\alpha)y_i$，即可得到最新且最佳的 β_0 值與 β_1 值之解，之後將會利用模擬數據與真實數據來驗證其提高模型的穩健能力之正確性於我們的研究結果。

以上為我們設計的三個演算法，接下來將利用模擬的數據與真實的數據來當本次的實驗，進而驗證我們設計的演算法之正確性與可行性，並且與其他現有的迴歸模型的演算法作比較，討論各演算法之優與劣，綜合應用這些演算法，可以使模型更具有較高的穩健性，不僅提高了對異常情況的適應能力，也增強了模型在現實數據中的可靠性。

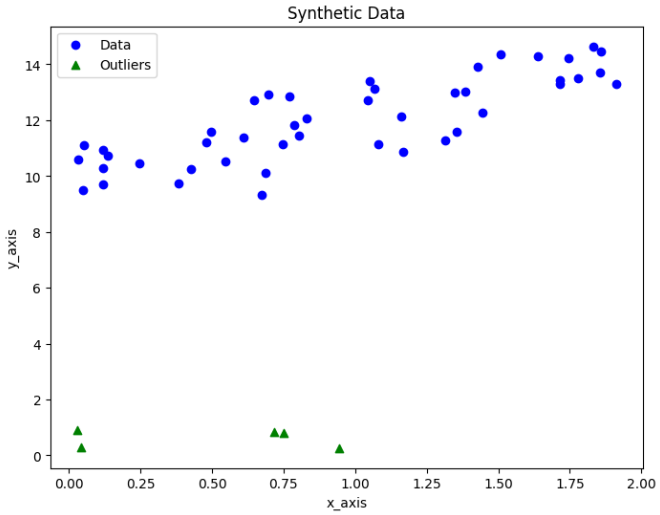
肆、 研究結果

在文獻回顧中，根據 [11] 與 [12]，對比不同模型的 RMSE 值與 R-squared 值是評估模型穩健性的一種方法，因此這裡我們選定 RMSE 值與 R-squared 值作為我們評估模型的指標，並計算出在不同模型下的兩值進行比較，RMSE 值衡量了模型預測與實際觀測值之間的誤差，較小的 RMSE 值表示模型的預測較準確，而 R-squared 值較高的模型通常表示模型能夠更好地解釋目標變數的變異性，進而彰顯模型對於異常值時，具有較高的抵抗能力，接下來我們分別利用模擬數據與真實數據來進行模型的穩健性之比較，實驗的結果整理如下：

（一）實驗一：

在這個實驗中，我們是基於一個基礎的線性函數 $y=10+2x$ 上，隨機生成 45 組噪

聲（nosie）並添加了 5 組異常值（或離群點），共 50 組模擬數據資料點，以模擬真實世界的數據，在 Google Colab 平台撰寫程式以圖來呈現，如圖二十四所示，其中藍色標示的點為噪聲資料點，綠色標示的點為異常值。



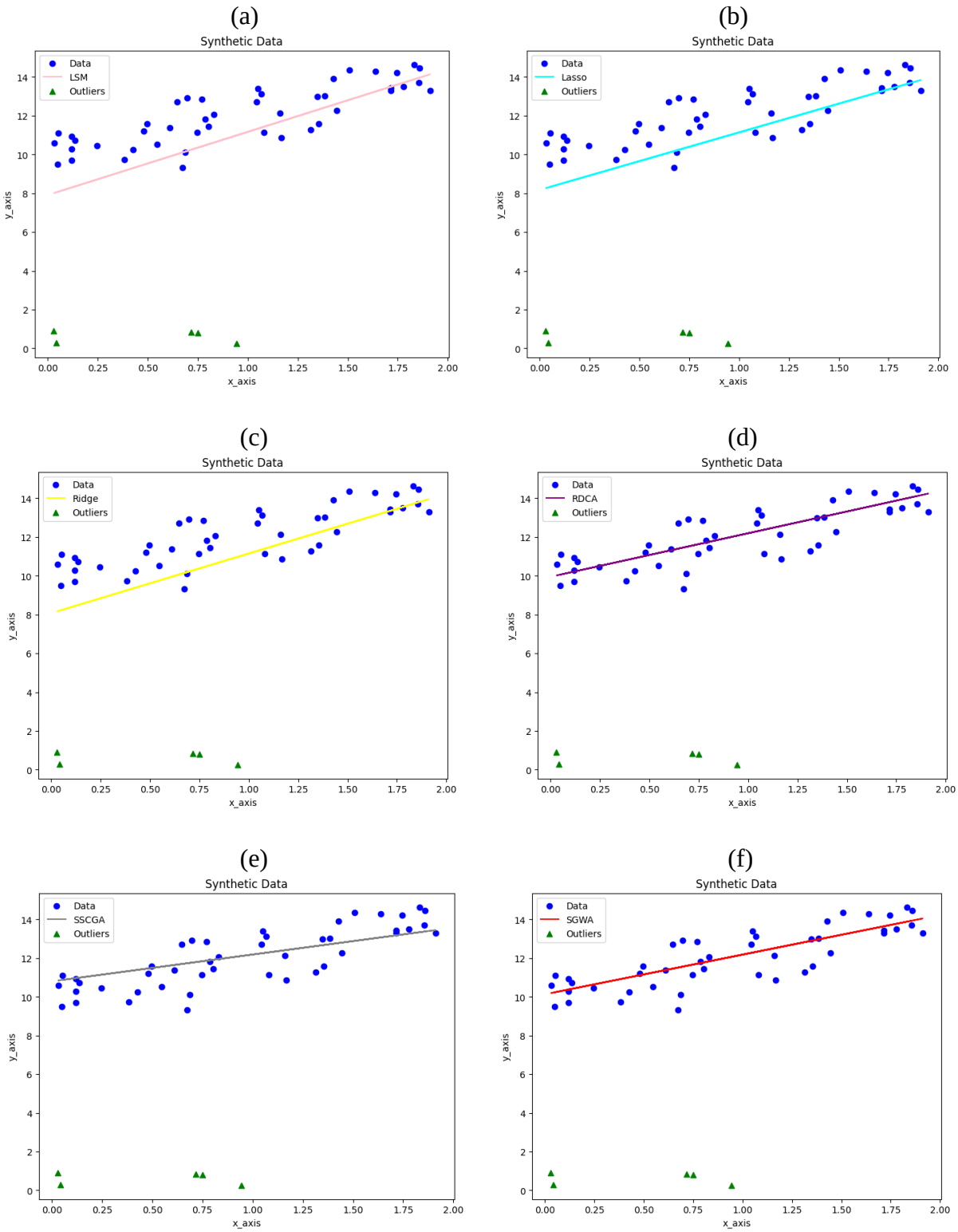
圖二十四：模擬數據圖。

接下來針對「LSM」、「Lasso」、「Ridge」與我們設計的「RDCA」、「SSCGA」 與「SGWA」等演算法，分別去計算其模型下的 RMSE 值與 R-squared 值，整理表如下：

表三：在各演算法下算出的 RMSE 值與 R-squared 值之比較。

algorithm	RMSE	R-squared
LSM	1.472	0.192
Lasso	1.411	0.100
Ridge	1.432	0.072
RDCA	0.860	0.666
SSCGA	0.952	0.590
SGWA	0.855	0.669

從表三可以觀察到相較前三個演算法，不論是 RDCA、SSCGA 或 SGWA，其 RMSE 值都較低，R-squared 值也都較高，在 RMSE 值的表現中，SGWA 最小，而在 R-squared 值的表現中，SGWA 最大（其值越趨近於 1），意味著我們的演算法對模型的預測較準確，且對於異常值有較高的抵抗能力，最後再透過繪製其對應的六條迴歸直線，如圖二十五所示，可以更容易地看出並驗證我們所設計的三個演算法下，其對應的三條迴歸直線，如圖二十五之(d) ~ (f)，皆較不容易被異常值影響，即具有較高的穩健性。

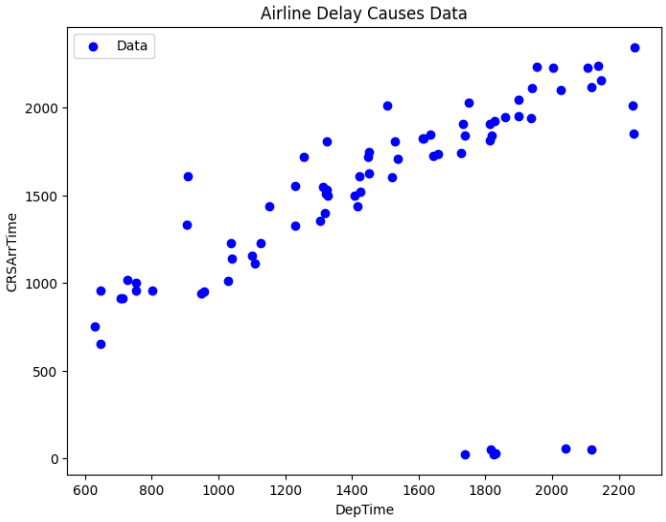


圖二十五：(a) LSM 法算出的迴歸直線　(b) Lasso 法算出的迴歸直線　(c) Ridge 法算出的迴歸直線　(d) RDCA 法算出的迴歸直線　(e) SSCGA 法算出的迴歸直線　(f) SGWA 法算出的迴歸直線。

（二）實驗二：

我們利用 Kaggle 網站 [13] 上所提供的數據「Airlines Delay」，是由美國運輸部下

的運輸統計局追蹤大型航空公司運營的國內航班的有關準點、延誤、取消和改道航班之數據集，在當月結束後約 30 天會公布相關報告，並向公眾提供航班延誤的相關資訊。由於相關資訊量相當龐大，是從 2003 年 6 月開始蒐集，故此次只取 2008 年 1 月的部分資料進行實驗，且因為延誤資訊較詳盡，取其中的航班出發時間 （DepTime）和電腦預約系統中紀錄的到達時間(CRS ArrTime)，撰寫程式呈現其兩者的關係如圖二十六所示。



圖二十六：DepTime 與 CRS ArrTime 之關係。

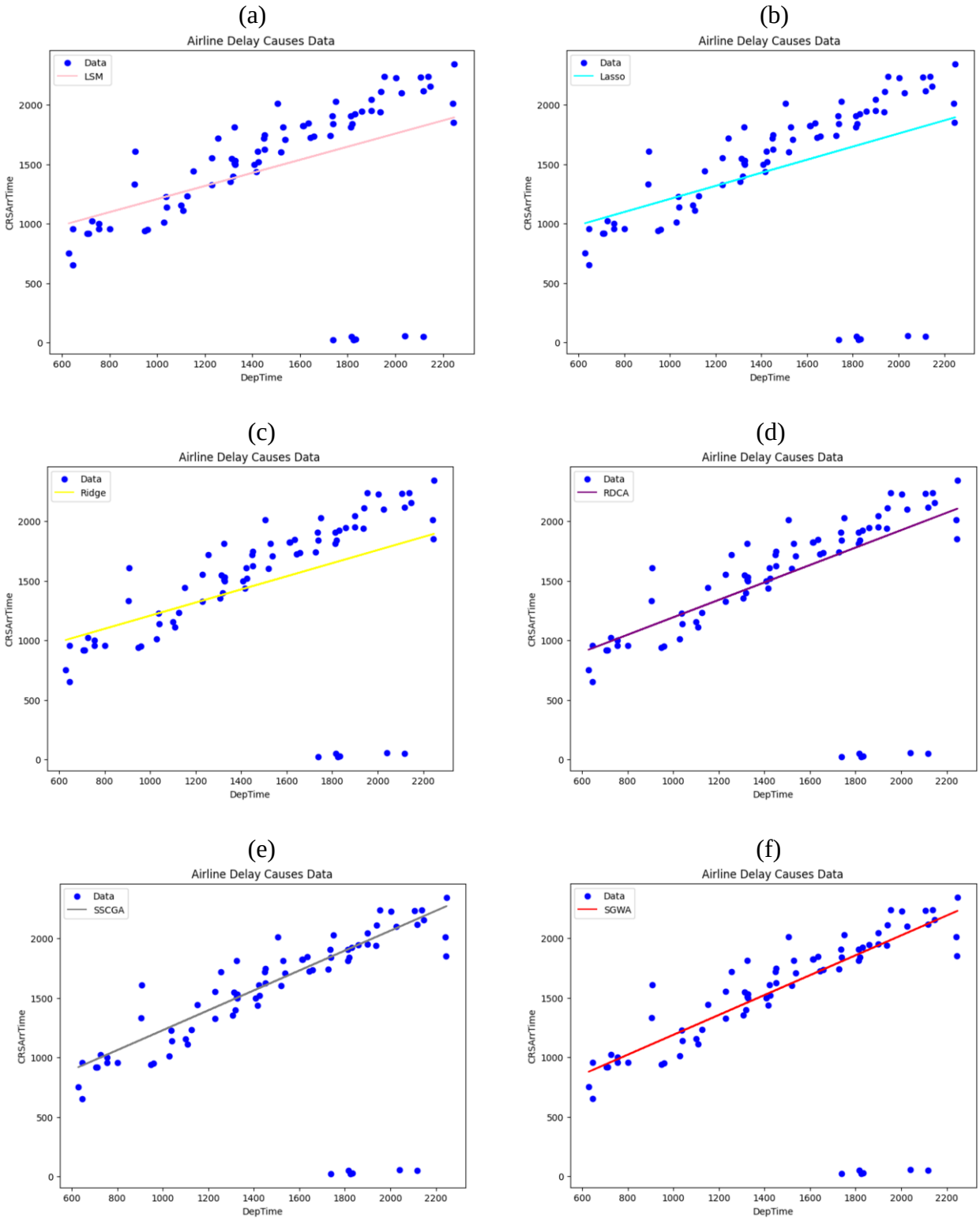
接下來針對「LSM」、「Lasso」、「Ridge」與我們設計的「RDCA」、「SSCGA」 與「SGWA」等演算法，分別去計算其模型下的 RMSE 值與 R-squared 值，整理表如下:

表四：在各演算法下算出的 RMSE 值與 R-squared 值之比較

algorithm	RMSE	R-squared
LSM	252.594	0.638
Lasso	252.594	0.638
Ridge	252.594	0.638
RDCA	178.767	0.819
SSCGA	149.057	0.874
SGWA	153.353	0.866

從表四可以觀察到，相較原有現行的演算法，依然是我們自創的演算法具有較低的 RMSE 值和較高的 R-squared 值，在 RMSE 值的表現中，SSCGA 最小，而在 R-squared 值的表現中，SSCGA 最大，表示著我們的演算法所生成的較高穩健性之模型也能夠更好地解釋因變數和自變數之間的關係，以進行預測或推斷。最後再透過繪製其對應的六條迴歸直線，如圖二十七所示，可以更容易地看出並驗證我們所設計的三個演算法下，

其對應的三條迴歸直線，如圖二十七之(d) ~ (f)，皆較不容易被異常值影響，即具有較高的穩健性。



圖二十七：(a) LSM 法算出的迴歸直線　(b) Lasso 法算出的迴歸直線　(c) Ridge 法算出的迴歸直線　(d) RDCA 法算出的迴歸直線　(e) SSCGA 法算出的迴歸直線　(f) SGWA 法算出的迴歸直線。

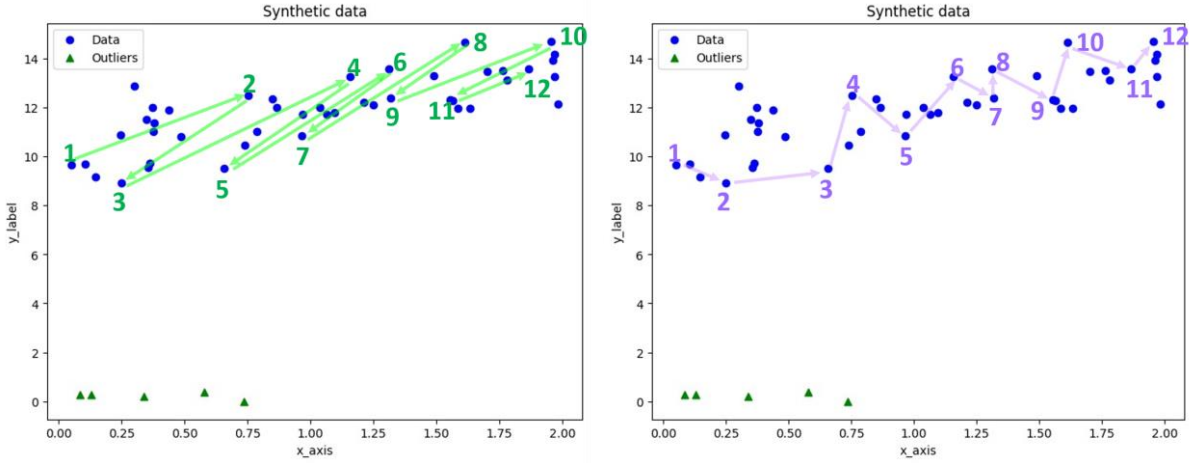
伍、 討論

本研究進行了兩個實驗，第一個為利用模擬數據，第二個為利用 Kaggle 網站 [13] 上所提供的數據集，結果皆為我們設計的三個演算法都具有正面且較佳的表現，但有時在使用「RDCA 演算法」與「SSCGA 演算法」時，我們也有觀察到一些異常的狀況，將其記錄下來，並探究其問題的原因，且將其對應的解決方案放入我們的未來展望，討論分別整理如下：

一、RDCA 演算法有時會選到極少量的異常值

第一種推論是因為隨機取樣的過程中會出現「隨機取點取到兩個以上距離較遠的原始數據」，如圖二十八之(a)中以綠色標示出取點的順序路徑。我們可以透過一些針對權重的演算法去更新每次隨機取樣的數據點之排列，使其計算出來的總距離必定是該次取樣中擁有最小加總距離之排列，如圖二十八之(b)中以紫色標示出取點的順序路徑，便可透過改善排列的方式間接調整那些距離過大的「原始數據」，此方法的好處是，如果原本的排列加總距離已經相當接近最佳排列的加總距離，那使用此方法不會改變太多距離；如果原本的排列擁有太多距離過大的原始數據，那使用此方法可以校正這些點的排序以排除過遠的情況發生。

(a) (b)



圖二十八：(a)隨機選點造成較大的總距離和 (b)排序選點改善成較小的總距離和

第二種推論是「權重函數的設計太差使權重之差距過小」。因權重函數的設計以所有測試的距離總和作為分母，這種設計會導致分子的「每次測試的距離總和」影響過小，導致最後的權重計算出現因權重差距過小而僅依靠取樣的次數便可提高自身權重的情況發生，為此在日後可以設計一個函數符合「加總距離越大之數據權重必定越小，即便多次取樣都有取到異

常值，其權重也不會大於只取到一次的原始數據」。例如效仿 SGWA 演算法設計一個梯形函數，或甚至設計一個指數函數以提高權重的差距。

二、SSCGA 演算常會有執行時間過長

經過資料的翻閱，我們發現已經有學者研究透過堆疊樹加速 Dijkstra 演算法，也有 A*演算法這種啟發式搜索演算法，透過啟發式評估函數來加速尋找更新節點，若將以上方法套用於 SSCGA 演算法中無疑都可以加速最短距離的計算，甚至可以減少判斷數據長度是否符合要求的時間消耗，達到更高效率之演算法，並在處理大量數據時予以幫助。

三、SSCGA 演算需人工擇優起始點

此演算法還有一個明顯的不足，就是執行前必須人工篩選一個較優的起始點，此舉會增加使用此演算法的時間成本，我們希望設計一個迷你演算法尋找最佳的起始點，便可以完全非人工的檢索最佳的起始點，更可以不需要每個數據點都代入 SSCGA 演算法中才能得出最佳解，這是一個值得我們日後討論與研究的方向。

陸、 結論

一、本研究提出三種自創演算法，分別是 RDCA、SSCGA 與 SGWA 演算法，相較其他三個現有的迴歸模型，我們算出的 RMSE 值較低，R-Squared 值趨近 1，在實驗一中，以 SGWA 的表現最佳，其 RMSE 值為 0.855，R-Squared 值為 0.669；在實驗二中，以 SSCGA 的表現最佳，其 RMSE 值為 149.057，R-Squared 值為 0.874，表示皆具有較高的穩健性。

二、在本研究所提出的三種自創演算法中，除了 SSCGA 仍需手動調整初始的節點以外，另外兩個演算法皆毋須手動調整參數。

三、未來展望：

（一）針對 RDCA 中偶發隨機選點的錯誤，找到導致模型被異常值干擾的原因並解決。

（二）面對 SSCGA 中出現執行時間過長的問題，使用其他相關最短路徑演算法加速該演算法的執行速度。

（三）消除 SSCGA 手動調整初始節點的需求，推廣成以「自動初始化」的方進行計算。

（四）設計新穎且泛用度更高的演算法並其應用於增強「多元線性迴歸模型」與「非線性迴歸模型」的穩健性，期待演算的技術能提供良好的預測效果。

柒、 參考文獻資料

[1] Rousseeuw, P.J., & Leroy, A.M. (1987). *Robust Regression and Outlier Detection*. John Wiley & Sons.

[2] Huber, P.J. (1964). Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics, 35*(1), 73-101.

[3] Maronna, R.A., Martin, R.D., & Yohai, V.J. (2006). *Robust Statistics: Theory and Methods*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc.

[4] García, S., Luengo, J., & Herrera, F. (2015). Dealing with noisy data. In *Data preprocessing in data mining* (pp. 72). Intelligent Systems Reference Library. Springer.

[5] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

[6] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). The MIT Press.

[7] 大槻兼資、秋葉拓哉（2024）。鍛鍊問題解決力！演算法與資料結構應用全圖解（陳韋利、馬毓晴、莊永裕譯）。臉譜。（原著出版於 2020 年）

[8] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269-271.

[9] 洪錦魁（2021）。演算法：最強彩色圖鑑 + Python 程式實作 王者歸來。深智數位。

[10] 王文俊（2017）。認識 Fuzzy 理論與應用（第四版）。全華圖書

[11] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer.

[12] Yoo, J. H., & Oh, S. Y. (2019). Comparison of goodness-of-fit indices for structural equation models. *Multivariate Behavioral Research, 54*(5), 625-646.

[13] Dua, D., & Graff, C. (2023). UCI Machine Learning Repository. Retrieved from https://archive.ics.uci.edu/