

巨量資料管理學院資料科學系

School of Big Data Management
Department of Data Science

Soochow University

機器學習驅動的三因子模型：股票市場報酬預測創新視角

Machine Learning-Driven Three-Factor Model: A New Perspective

on Stock Market Return Prediction

黃柏愷

Huang, Po-Kai

指導教授：鄭宏文 博士

Advisor: Hung-Wen Cheng Ph.D.

113 年 11 月

November 2024

摘要 Abstract

股票市場回報預測一直是金融領域的核心問題之一，傳統的線性模型(如三因子模型) 雖然具有簡單性和解釋力，但在捕捉市場因子的非線性關係時存在局限性。本文提出了基於三因子模型的機器學習方法，用於股票市場報酬的預測。通過結合帳面市值比(Book-to-Market)、市值(Size) 及動能(Momentum) 三個因子與現代機器學習技術，開發了包括隨機森林和類神經網路多種模型，並使用台灣股市 1999 年-2024

年的月度資料進行實驗測試。結果顯示，與傳統線性模型相比，機器學習再多項指標(如 RMSE 和 R²) 表現上取得顯著改進，且投資報酬也有大幅度提升。本研究的發現為金融市場的量化投資提供了新的視角，並且實證了機器學習技術在金融預測中的潛力。

一、緒論 Introduction

1.1 背景與動機

股票市場的波動性和報酬率預測一直是金融研究的重要課題，隨著金融市場的不斷發展，投資者對準確預測市場回報的需求月來月高。傳統的線性回歸模型雖然具有簡單性和解釋性，但其假設因子與市場回報的關係是線性的，無法有效捕捉多因子之間的複雜交互作用與非線性關係。此外，隨著金融市場數據量的增加，如何利用資料科學技術進一步提高預測準確性成為一個急需解決的問題。

在此背景下，本研究嘗試結合三因子模型與機器學習技術，開發一種新的市場報酬率預測方法。通過將傳統的財務因子與先進的演算法相結合，希望能夠克服線性模型的限制，並探索機器學習技術在量化金融的潛力。

1.2 研究目的

本研究主要目的是結合傳統財務上的三個因子，帳面市值比(Book-to-Market)、市值(Size) 及動能(Momentum)與機器學習技術，開發一種新的股票市場報酬預測框架。具體而言，研究嘗試解決以下問題：

1. 傳統線性模型在捕捉多因子間非線性關係上的不足如何改進;
2. 不同機器學習演算法在股票市場報酬預測中的表現如何;
3. 提出的模型是否能在多種市場情境下穩定的優於傳統方法;

1.3 論文架構

- 一、 緒論：包含背景和動機，研究目的。
- 二、 文獻探討：
- 三、 研究方法：介紹傳統財務上的三個因子，以及三種機器學習模型，並說明個字是用場景
- 四、 研究結果：對比三種模型的預測效果

五、 結論：總結研究成果，並提出能在改進的部分

六、 心得

七、 附錄

二、文獻探討 Literature Review

2.1 傳統三因子模型的發展與限制

Fama 和 French (1993) 提出的三因子模型是資產定價領域的重要里程碑。該模型通過將市場因子 (Market)、規模因子 (SMB, Small Minus Big) 和價值因子 (HML, High Minus Low) 納入框架，解釋了資產收益的主要驅動因素。該模型在實證研究中被廣泛應用，並啟發了包括四因子模型 (Carhart, 1997) 在內的多因子資產定價方法。

然而，傳統三因子模型存在以下局限性：

1. **線性假設**：三因子模型假定資產報酬與因子之間的關係是線性的，忽略了市場中可能存在的非線性和交互效應。
2. **固定因子權重**：該模型的因子權重是靜態的，無法動態適應市場變化。
3. **數據維度受限**：傳統模型多基於經濟學理論選擇少量因子，可能忽略了潛在的重要數據特徵。

2.2 機器學習在資產定價中的應用

機器學習技術的興起為資產定價研究提供了新的工具，特別是在處理高維數據和捕捉非線性關係方面展現了顯著優勢。以下為機器學習在資產定價中的主要應用：

- 1. 因子的自動生成與選擇：**
 - Gu, Kelly, and Xiu (2020) 探討了機器學習如何篩選數百個財務和市場變量，實現對資產收益驅動因子的自動化建模。
 - 特徵選擇算法（如 Lasso 和隨機森林）可以有效篩選與股票收益相關的關鍵因子，補充或取代傳統因子。
- 2. 非線性關係的建模：**
 - 隨機森林（Random Forest）、支持向量機（SVM）和深度學習方法能捕捉三因子與資產收益之間的非線性關係，克服了傳統三因子模型的限制。
- 3. 動態權重調整：**
 - 梯度提升機（Gradient Boosting Machine, GBM）和時間序列模型（如 LSTM）能隨著市場環境的變化動態調整因子權重，更準確地反映市場風險與收益的關係。
- 4. 市場異質性建模：**
 - 機器學習算法能處理來自不同市場或資產類別的異質性數據，識別更精細的收益驅動模式 (Gu et al., 2020)。

2.3 結合機器學習與三因子模型的創新視角

傳統因子模型和機器學習的結合為資產定價研究帶來了創新的可能性：

- 1. 因子非線性與交互效應：**
 - Byun and Kim (2021) 的研究表明，使用機器學習技術可以識別三因子間的交互效應及非線性關係，提升預測精準度。
 - 特別是利用樹模型（如 XGBoost）或神經網路，捕捉市場因子的複雜動態行為。
- 2. 跨市場與時間動態的泛化：**
 - 研究顯示，機器學習方法能有效整合跨市場數據，並且對歷史數據中的時變模式具有更強的適應能力。
- 3. 多因子擴展：**

- 機器學習不僅能應用於現有的三因子模型，還能探索更多潛在的補充因子（如情緒指數、新聞文本分析結果）作為特徵，進一步完善模型。

2.4 挑戰與爭議

儘管機器學習為三因子模型的改進提供了新方向，但也面臨一些挑戰：

1. 模型解釋性：

- 與傳統三因子模型相比，機器學習模型的「黑箱」性質導致難以清晰地解釋因子與報酬之間的經濟意涵。
- 一些研究（如 Molnar, 2020）嘗試使用 Shapley 值和局部可解釋模型（LIME）來提升機器學習模型的透明度。

2. 穩健性與過擬合：

- 在高維數據環境下，機器學習模型容易出現過擬合問題，特別是在樣本數據有限時。

3. 數據質量與維度：

- 機器學習方法對數據質量依賴較高，缺失值、異常值和標籤噪聲可能嚴重影響模型的表現。

三、研究方法 Method

3.1 引言

本章節介紹三種財務上的傳統因子及其計算方法，三種演算法及其適用場景，最後包含數據的介紹。

3.2 因子介紹

3.2.1 帳面市值比(BM)

帳面市值比或淨值市值比(Book-to-market ratio)：指將帳面價值與其市場價值進行比較從而評估一家公司的價值。

$$BM = \frac{\text{帳面價值}}{\text{市值}} = \frac{1}{\text{股價淨值比}}$$

3.2.2 規模(Size)

市值是企業規模的代表，反應了公司的市場地位、聲譽和未來前景。長期以來被認為對股票收益產稱重大影響。

$$\begin{aligned} \text{市值} &= \text{收盤價} \times \text{流通在外股數} \\ \text{規模}(Size) &= \log(\text{市值}) \end{aligned}$$

3.2.3 動能(Momentum)

動能是觀察上漲或下跌的資產價格或投資報酬有繼續上漲或下跌的趨勢。

3.2.3.1 動能效應

研究顯示過去表現強勁的股票，在下一個期間通常優於過去表現較差的股票，提供平均每月約 1% 的超額報酬。

將採用形成期 12 個月計算以報酬率形成的動能，概念為 $t - 1$ 期股票收盤價除以 $t - 12$ 期股票收盤價取對數，不受基期和時間影響：

$$Momentum = \ln\left(\frac{S_{t-1}}{S_{t-12}}\right)$$

3.3 機器學習模型介紹

本研究選擇以下三種預測模型實驗：

- 線性回歸模型(OLS): 利用因子數據對報酬進行線性回歸分析。模型假設因子與報酬率呈線性關係。
- 隨機森林(Random Forest): 基於多個決策樹的集成學習方法，通過引入隨機性來構建每棵樹並進行多數投票或平均，從而提高模型的預測準確性與穩健性，同時減少過擬合的風險。
- 類神經網路(Neural Networks): 受人類大腦啟發的機器學習模型，通過多層節點（神經元）和權重的非線性映射來學習數據中的隱含模式，特別適用於處理複雜的非線性問題。

3.3.1 線性回歸模型(Linear Regression)

線性回歸模型是一種簡單且常見的統計學模型，用於建模變數之間的關係。主要應用於預測和解釋數據。線性回歸模型假設自變量(輸入變量)和因變量(輸出變量)之間存在線性關係。模型的形式如下：

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon$$

其中：

- Y ：模型的輸出，即我們要預測或解釋的目標變數。以我們研究為例， Y 就是我們預測出的因子預測值。
- β_0 ：截距/常數項，它反映了目標變數的基線水平。
- $\beta_1, \beta_2, \dots, \beta_n$ （回歸係數/權重）：
 - β_i 代表第 i 個自變數 (X_i) 對目標變數 (Y) 的影響程度
- X_1, X_2, \dots, X_n （自變數）：以我們的研究為例，分別是 Size, Momentum 以及 Book-to-Market
- ε ：誤差項，表示模型無法解釋的隨機誤差或噪聲。

3.3.2 隨機森林(Random Forest)

隨機森林是一種基於決策樹的集成學習方法，由 Leo Breiman 提出，他通過建構多顆決策樹並結合他們的輸出進行分類或回歸任務，隨機森林在解決高為數據、非線性問題以及避免過擬合等方面具有優點。

隨機森林的核心思想包括兩個關鍵技術

1. Bootstrap Aggregation(Bagging): 通過隨機有放回地從訓練數據集中抽樣來建立多個訓練集，然後用這些訓練生成多顆決策樹。

2. 特徵隨機選擇：在每棵樹的節點分裂時，隨機選擇部分特徵進行決策分裂，而非考慮所有特徵，從而減少決策樹的相關性，提升模型泛化能力。

隨機森林的回歸結果為所有樹預測值的平均：

$$\hat{y} = \frac{1}{B} \sum_{b=1}^B T_b(x)$$

- $T_b(x)$ ：第 b 棵樹對輸入 x 的預測值。

3.3.3 類神經網路(Neural Networks)

類神經網路是一種模仿生物神經網路工作原理的計算模型，旨在解決模式辨識、回歸分析等複雜任務。它由一組互相連接的「神經元」（或稱節點）組成，通過調整網路內部的權重和偏差來學習數據的特稱。

類神經網路的基本結構包括：

1. 輸入層(Input Layer)：負責接受數據輸入，每個節點對應一個特徵。
2. 隱藏層(Hidden Layer)：對數據進行特徵提取和映射，可能有多層。
3. 輸出層(Output Layer)：提供最終結果，如分類標籤或回歸值。

類神經通過前向傳播(Forward Propagation) 計算輸出，並通過誤差反向傳播(Back Propagation) 更新權重，使網路的預測結果越來越接近真實值。

1. 單層神經元計算公式

每個神經元的輸出計算公式為：

$$z = \sum_{i=1}^n w_i x_i + b$$

- x_i ：輸入特徵
- w_i ：對應的權重
- b ：偏差 (bias)
- z ：加權輸入 (Weighted Sum)

激活函數 $f(\cdot)$ 將加權輸入 z 映射到輸出：

$$a = f(z)$$

- $f(\cdot)$ ：激活函數，如 ReLU、Sigmoid、Tanh 等
- a ：神經元的輸出

2. 多層前向傳播公式

對於一個含有 L 層的網路，從輸入層到輸出層的數值計算過程為：

$$\begin{aligned} z^{[l]} &= W^{[l]}a^{[l-1]} + b^{[l]} \quad (\text{第 } l \text{ 層的加權輸入}) \\ a^{[l]} &= f(z^{[l]}) \quad (\text{第 } l \text{ 層的輸出}) \end{aligned}$$

- $W^{[l]}$ ：第 l 層的權重矩陣
- $b^{[l]}$ ：第 l 層的偏差向量
- $a^{[l-1]}$ ：第 $l - 1$ 層的輸出

3. 損失函數

損失函數衡量網路預測值與真實值之間的差異，常見的損失函數包括：

- 均方誤差 (MSE) (回歸問題) :

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

- 交叉熵損失 (Cross-Entropy Loss) (分類問題) :

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^m [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

- m ：樣本數
- y_i ：真實值
- \hat{y}_i ：預測值

4. 反向傳播與梯度下降

在反向傳播中，通過鏈式法則計算每個參數的梯度，更新公式為：

$$\begin{aligned} W^{[l]} &= W^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial W^{[l]}} \\ b^{[l]} &= b^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial b^{[l]}} \end{aligned}$$

- η ：學習率 (Learning Rate)

3.4 資料集以及資料處理

我們的資料來源是 Tej Pro 財經資料庫，選擇所有上市櫃公司從 1999/01 – 2024/08 資料，篩選掉所有在 2005/01 還沒有股價的公司，以確保資料的完整性。篩選完後僅剩 966 家公司資料，以這個公司列表對 Book-to-Market, Momentum 及 size 完成篩選。對於缺失值，我們採用該月所有公司該因子的平均值，來避免極端值對預測準確產生影響。對於下個月月報酬，我們採用 $\ln\left(\frac{\text{下個月股價}}{\text{這個月股價}}\right)$ 來計算。之後即計算 IC 值。

3.4.1 資訊係數(Information Coefficient, IC)

IC: 股票的因子與股票下期報酬率的相關係數。透過 IC 值可以判斷因子對下期報酬率的預測能力。我們採用以下兩種 IC 值：

- Normal IC: 在時間點 t 時因子與持有 h 個月的報酬率間的相關係數

$$\text{Normal IC} = \text{corr}(X_t, R_{t+h})$$

- Rank IC: 在時間點 t 時因子排序與持有 h 個月的報酬率排序間的相關係數

$$\text{Rank IC} = \text{corr}(\text{rank}(X_t), \text{rank}(R_{t+h}))$$

- 為了計算 Rank IC，我們已經事先對所有因子及報酬都進行 Rank 排序。

3.5 實驗設計

取得 IC 值後，我們將因子資料及 IC 值進行演算法的訓練，初始訓練集採用 2005/01 – 2023/11 的資料，測試集採用 2023/12 的因子資料來預測 IC 值，取絕對值最大的因子，對其 2023/12 月的資料進行排序並分等分，包含 10, 20, 30, 50, 100, 200, 300, 450, 900 等分。最後如果該因子原始預測 IC 值為正及買最高等分並賣最低等分，反之賣最高買最低等分。最後取的該月報酬率並與台灣加權指數報酬率及其他模型比較。每次預測完訓練集及加入此月真實資料，而測試集即採次月資料進行預測。

四、研究結果 Results

1. 線性回歸表現不差的原因

令人驚訝的是，線性回歸的表現並非最差，甚至在多數分類中能勝過實驗基準的大盤回報。這一結果可能反映以下幾點：

1. 數據特徵的線性關係：

- 若實驗數據中資產報酬與選取的因子之間主要呈現線性關係，線性回歸能直接捕捉這些關係，從而提供穩健的預測。

2. 模型的穩定性與簡單性：

- 線性回歸作為一種簡單的模型，不容易過度擬合小數據集或高噪聲數據。相比之下，更複雜的模型可能因高維數據中的噪聲而產生過度擬合問題。

3. 基準設置的挑戰：

- 若大盤回報的波動性較低，則只要模型能略微捕捉報酬的系統性波動，便能超過基準表現。

2. 隨機森林效果不理想的原因

實驗結果顯示，隨機森林的表現非常不理想，這可能與以下因素有關：

1. 特徵重要性與數據關係：

- 隨機森林通常在處理非線性和高維數據時表現出色，但如果數據中的非線性關係不明顯，隨機森林可能無法有效發揮其優勢。

2. 過度擬合的風險：

- 隨機森林的高靈活性可能導致在小樣本數據集或高噪聲數據中過度擬合，導致樣本外表現不佳。

3. 因子選擇的影響：

- 如果實驗因子選擇未充分反映報酬驅動的多樣性或存在冗餘，隨機森林可能過於依賴某些次要特徵，降低整體效果。

3. 多層神經網路的表現差異

在類神經網路模型中，2 層和 3 層神經網路表現最佳，而 5 層神經網路效果不佳，可能的原因包括：

1. 模型複雜性與數據匹配：

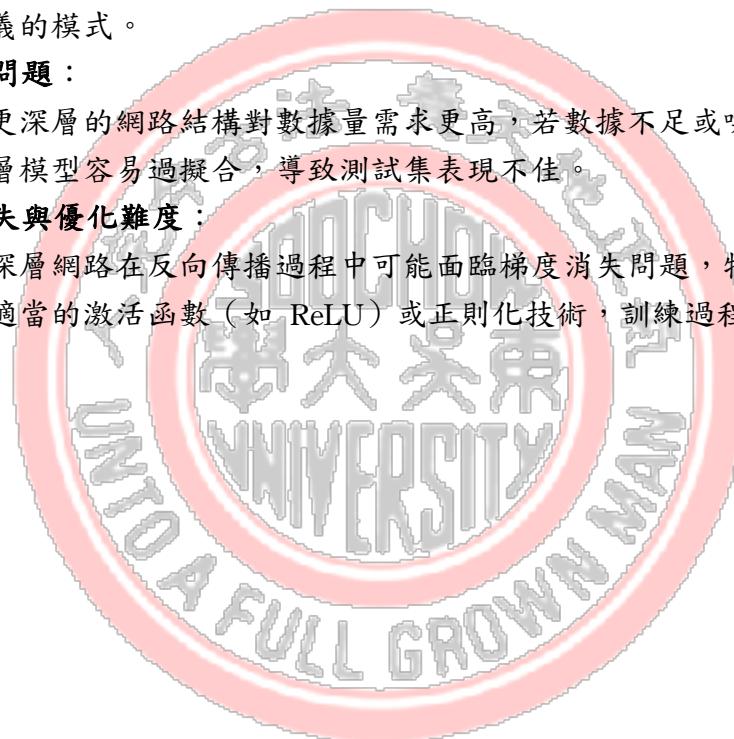
- 2 層和 3 層神經網路能有效捕捉數據中的關鍵模式，而 5 層神經網路可能因過高的模型複雜性，無法在樣本數據有限的情況下學習有意義的模式。

2. 過擬合問題：

- 更深層的網路結構對數據量需求更高，若數據不足或噪聲較多，則深層模型容易過擬合，導致測試集表現不佳。

3. 梯度消失與優化難度：

- 深層網路在反向傳播過程中可能面臨梯度消失問題，特別是若未使用適當的激活函數（如 ReLU）或正則化技術，訓練過程可能難以收斂。



4.1.1 Linear Regression

4.1.1.1 NIC



LR NIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	0.0474	0.1003	0.181	0.1596	0.1432	0.0721	0.1725	0.122	0.1436
size	-0.1111	-0.0815	-0.0787	-0.2196	-0.2404	-0.2145	-0.1152	0.1579	0.189
mom	-0.1869	0.0284	-0.7166	-0.0093	0.0403	0.3993	0.2312	0.3651	0.4688
factor	mom	bm	mom	size	size	mom	mom	mom	mom
10	3.562	-2.497	-1.46	1.7	-0.17	-5.211	1.746	0.02	6.619
20	1.756	-2.179	-0.876	0.551	-0.542	-3.034	1.388	-0.678	3.468
50	0.919	-1.978	0.845	0.873	-0.4	-1.984	0.751	-0.123	1.617
100	0.778	-0.343	0.46	0.468	-0.503	-1.186	1.068	-0.166	0.29
200	0.991	-0.024	0.861	0.51	-0.31	-0.768	0.269	-0.423	0.054
300	0.784	0.039	0.113	0.619	-0.49	-0.742	0.099	-0.189	-0.137
450	0.591	0.132	0.171	0.471	0.007	-0.666	-0.024	-0.311	-0.211
900	0.522	0.109	-0.068	0.385	0.535	-0.626	-0.021	-0.129	-0.046
TWII	-0.0023	0.0585	0.0677	0.005	0.0374	0.0841	-0.0368	0.0031	-0.002

Win Rate	
10	0.5556
20	0.4444
50	0.5556
100	0.5556
200	0.5556
300	0.4444
450	0.5556
900	0.5556

4.1.1.2 RIC

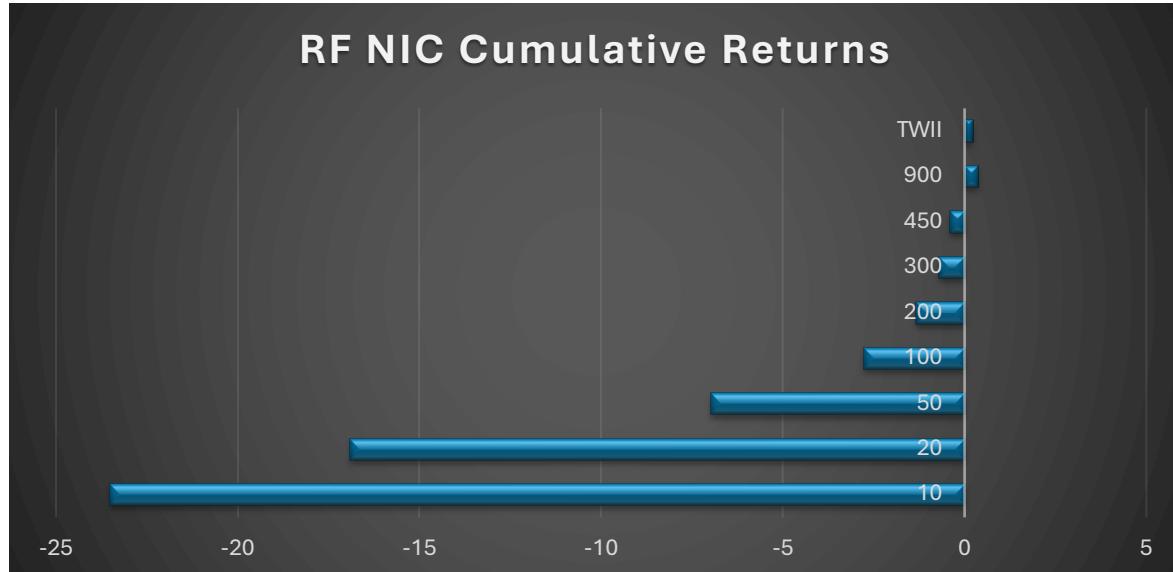


LR RIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	0.0392	0.0864	0.1729	0.1719	0.1412	0.0807	0.1694	0.1101	0.1434
size	-0.1141	-0.1094	-0.1133	-0.2326	-0.255	-0.221	-0.1224	0.1578	0.1972
mom	-0.2121	0.0715	-0.5964	-0.0059	0.0924	0.3582	0.0796	0.2225	0.391
factor	mom	size	mom	size	size	mom	bm	mom	mom
10	3.562	-2.389	-1.46	1.7	-0.17	-5.211	-4.897	0.02	6.619
20	1.756	-1.43	-0.876	0.551	-0.542	-3.034	-2.634	-0.678	3.468
50	0.919	-0.146	0.845	0.873	-0.4	-1.984	-1.457	-0.123	1.617
100	0.778	-0.528	0.46	0.468	-0.503	-1.186	-1.437	-0.166	0.29
200	0.991	0.147	0.861	0.51	-0.31	-0.768	-0.609	-0.423	0.054
300	0.784	0.02	0.113	0.619	-0.49	-0.742	-0.668	-0.189	-0.137
450	0.591	-0.082	0.171	0.471	0.007	-0.666	-0.361	-0.311	-0.211
900	0.522	0.044	-0.068	0.385	0.535	-0.626	0.107	-0.129	-0.046
TWII	-0.0023	0.0585	0.0677	0.005	0.0374	0.0841	-0.0368	0.0031	-0.002

Win Rate	
10	0.4444
20	0.3333
50	0.4444
100	0.4444
200	0.5556
300	0.3333
450	0.3333
900	0.4444

4.1.2 Random Forest

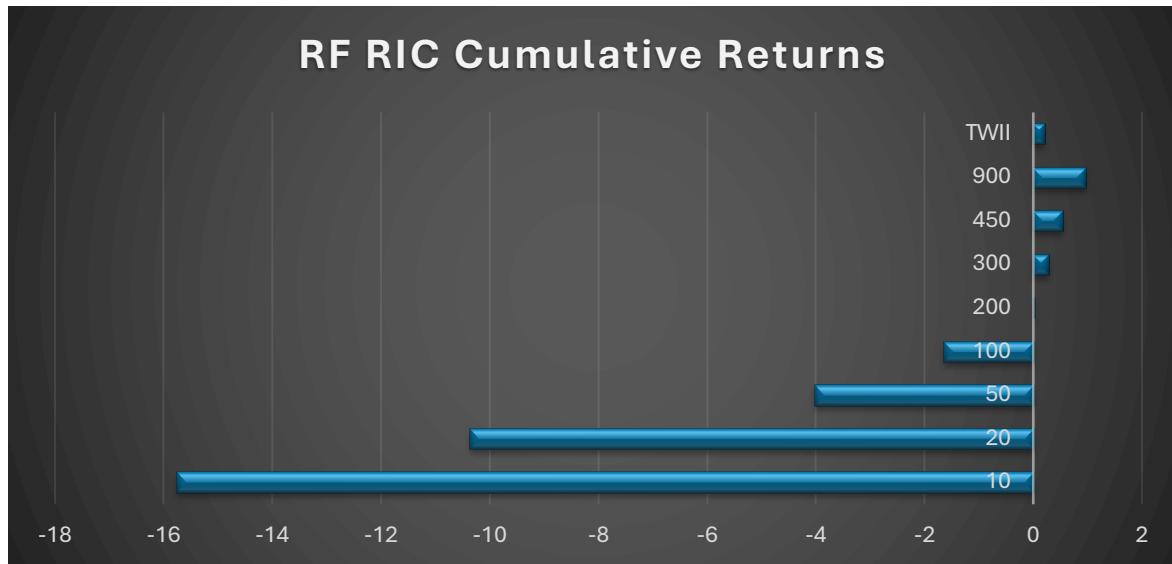
4.1.2.1 NIC



RF_NIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	0.0311	0.0356	0.0487	0.0282	0.0274	0.0306	0.0328	0.0412	0.0481
size	-0.0188	-0.0331	-0.0179	-0.0274	-0.0242	-0.0229	-0.0183	-0.0257	-0.0195
mom	-0.0094	-0.0067	-0.0142	-0.0032	-0.0111	-0.0049	-0.0072	0.0023	-0.0011
factor	bm								
10	-6.878	-2.497	5.786	-2.823	-4.389	-2.213	-4.897	-0.593	-5.005
20	-3.018	-2.179	1.326	-2.044	-1.762	-2.335	-2.634	-1.252	-2.997
50	-1.156	-1.978	1.122	-0.271	-0.048	-0.764	-1.457	-0.409	-2.026
100	-0.376	-0.343	1.171	-0.512	0.606	-0.28	-1.437	-0.408	-1.193
200	0.038	-0.024	0.881	0.151	0.516	-0.73	-0.609	-0.451	-1.1
300	0.007	0.039	0.594	0.199	0.412	-0.458	-0.668	-0.396	-0.424
450	0.109	0.132	0.484	-0.16	0.243	-0.532	-0.361	-0.333	0.017
900	0.233	0.109	0.111	-0.016	-0.047	-0.639	0.107	0.434	0.085
TWII	-0.0023	0.0585	0.0677	0.005	0.0374	0.0841	-0.0368	0.0031	-0.002

Win Rate	
10	0.1111
20	0.1111
50	0.1111
100	0.2222
200	0.4444
300	0.4444
450	0.5556
900	0.6667

4.1.2.2 RIC



RF RIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	0.0237	0.0255	0.0351	0.034	0.0284	0.0271	0.0363	0.0419	0.054
size	-0.026	-0.0346	-0.0284	-0.0351	-0.025	-0.0305	-0.0249	-0.0268	-0.0201
mom	-0.0021	-0.0072	-0.0132	-0.0127	-0.0055	-0.006	-0.0033	-0.0025	0.0002
factor	size	size	bm	size	bm	size	bm	bm	bm
10	-4.198	-2.389	5.786	1.7	-4.389	-1.765	-4.897	-0.593	-5.005
20	-1.264	-1.43	1.326	0.551	-1.762	-0.894	-2.634	-1.252	-2.997
50	-1.435	-0.146	1.122	0.873	-0.048	-0.495	-1.457	-0.409	-2.026
100	-0.098	-0.528	1.171	0.468	0.606	-0.216	-1.437	-0.408	-1.193
200	-0.35	0.147	0.881	0.51	0.516	0.489	-0.609	-0.451	-1.1
300	-0.339	0.02	0.594	0.619	0.412	0.467	-0.668	-0.396	-0.424
450	-0.389	-0.082	0.484	0.471	0.243	0.493	-0.361	-0.333	0.017
900	-0.495	0.044	0.111	0.385	-0.047	0.342	0.107	0.434	0.085
TWII	-0.0023	0.0585	0.0677	0.005	0.0374	0.0841	-0.0368	0.0031	-0.002

Win Rate	
10	0.2222
20	0.2222
50	0.2222
100	0.3333
200	0.5556
300	0.4444
450	0.5556
900	0.6667

4.1.4 Neural Networks 1 Layer

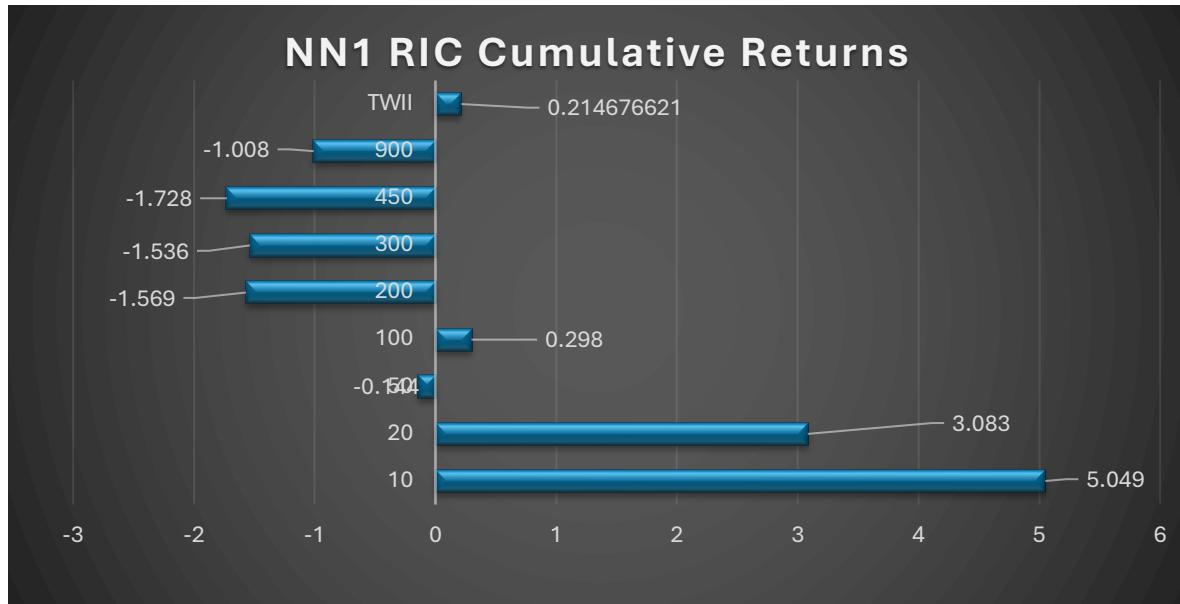
4.1.3.1 NIC



NN1 NIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	-0.00779842	-0.00347002	-0.03688367	0.05320581	0.05344979	0.02731163	0.05526064	-0.01432591	-0.02700493
size	-0.05207089	-0.05121455	-0.05841129	-0.05827859	0.04426945	-0.05675044	0.05350561	-0.05751008	0.05564204
mom	[0.01509112]	[0.13399586]	[-0.06709384]	[0.21765842]	[-0.04154728]	[-0.2609754]	[-0.12502424]	[0.38325143]	[0.44591764]
factor	size	mom	mom	mom	bm	mom	mom	mom	mom
10	-4.198	-4.742	-1.46	4.73	-4.389	5.211	-1.746	0.02	6.619
20	-1.264	-1.863	-0.876	3.732	-1.762	3.034	-1.388	-0.678	3.468
50	-1.435	-0.858	0.845	2.301	-0.048	1.984	-0.751	-0.123	1.617
100	-0.098	-0.722	0.46	1.325	0.606	1.186	-1.068	-0.166	0.29
200	-0.35	-0.662	0.861	0.647	0.516	0.768	-0.269	-0.423	0.054
300	-0.339	-0.682	0.113	0.562	0.412	0.742	-0.099	-0.189	-0.137
450	-0.389	-0.185	0.171	0.378	0.243	0.666	0.024	-0.311	-0.211
900	-0.495	-0.248	-0.068	0.553	-0.047	0.626	0.021	-0.129	-0.046
TWII	-0.00230316	0.058471198	0.067658949	0.005020771	0.037416184	0.084111004	-0.036832399	0.003091703	-0.001957628

Win Rate	
10	0.4444
20	0.3333
50	0.4444
100	0.5556
200	0.5556
300	0.4444
450	0.5556
900	0.3333

4.1.3.2 RIC

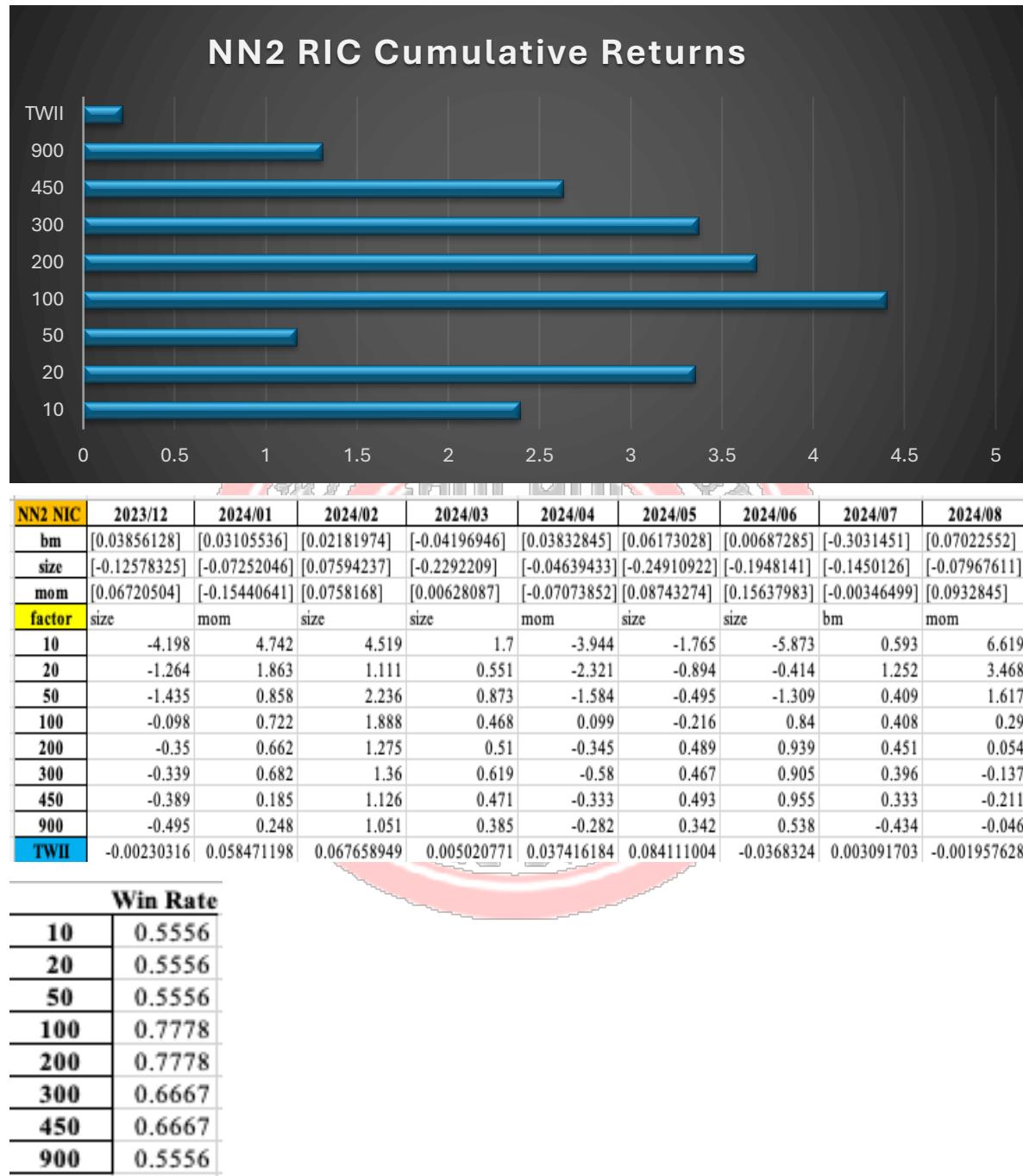


NN1 RIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	[0.0332618]	[0.03734116]	[0.03058735]	[0.00396639]	[0.05078202]	[-0.03727074]	[-0.02910197]	[0.05604104]	[0.03543808]
size	[-0.05881836]	[0.03400778]	[-0.05717988]	[0.0557122]	[0.05658286]	[0.05585065]	[-0.05790249]	[0.05218075]	[0.05642719]
mom	[-0.16179529]	[0.01886468]	[0.08183459]	[0.04795728]	[-0.66737145]	[0.2494502]	[0.65413284]	[0.45992127]	[0.08967207]
factor	mom	bm	mom	size	mom	mom	mom	mom	mom
10	3.562	2.497	1.46	-1.7	-3.944	-5.211	1.746	0.02	6.619
20	1.756	2.179	0.876	-0.551	-2.321	-3.034	1.388	-0.678	3.468
50	0.919	1.978	-0.845	-0.873	-1.584	-1.984	0.751	-0.123	1.617
100	0.778	0.343	-0.46	-0.468	0.099	-1.186	1.068	-0.166	0.29
200	0.991	0.024	-0.861	-0.51	-0.345	-0.768	0.269	-0.423	0.054
300	0.784	-0.039	-0.113	-0.619	-0.58	-0.742	0.099	-0.189	-0.137
450	0.591	-0.132	-0.171	-0.471	-0.333	-0.666	-0.024	-0.311	-0.211
900	0.522	-0.109	0.068	-0.385	-0.282	-0.626	-0.021	-0.129	-0.046
TWII	-0.00230316	0.058471198	0.067658949	0.005020771	0.037416184	0.084111004	-0.036832399	0.003091703	-0.001957628

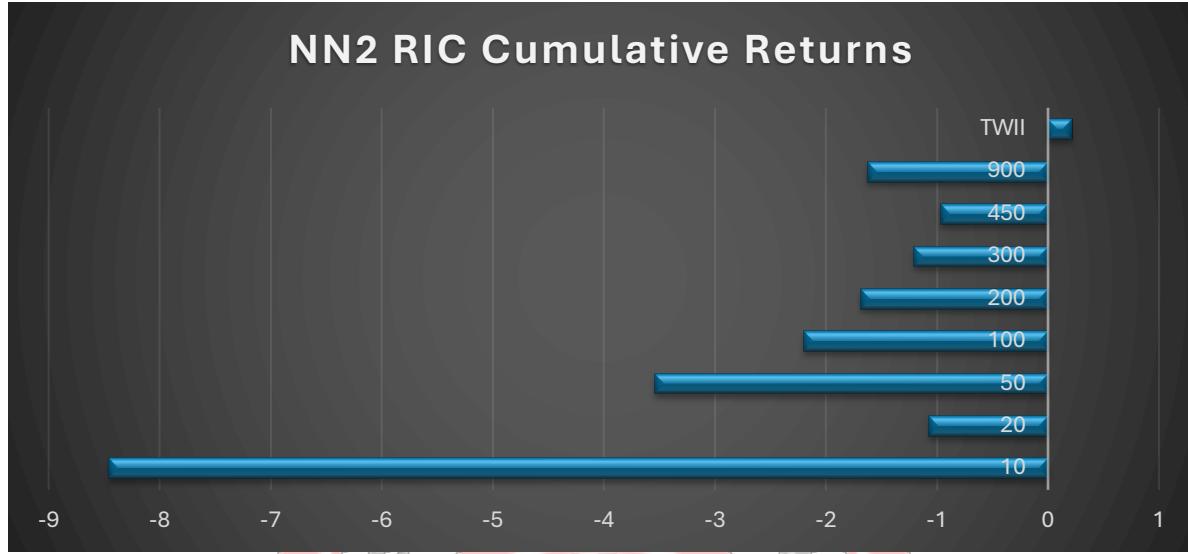
Win Rate	
10	0.6667
20	0.5556
50	0.4444
100	0.5556
200	0.3333
300	0.2222
450	0.2222
900	0.3333

4.1.5 Neural Networks 2 Layers

4.1.5.1 NIC



4.1.5 RIC

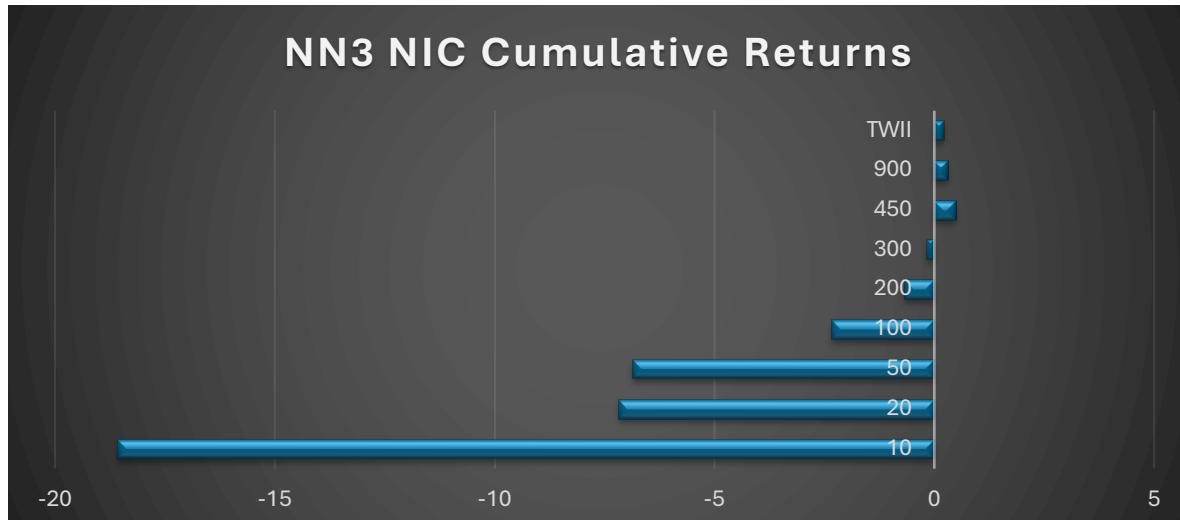


NN2 RIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08	
bm	[-0.04791889]	[-0.00409204]	[0.00471927]	[-0.03463764]	[0.07175818]	[0.01424611]	[0.12023245]	[-0.04575898]	[0.02969322]	
size	[-0.20465541]	[-0.03823131]	[-0.04042062]	[0.05061335]	[0.5640025]	[-0.11178929]	[-0.10994724]	[-0.01550394]	[-0.14758483]	
mom	[-0.04351506]	[0.05190023]	[0.11844606]	[0.01414604]	[0.06203404]	[-0.08251585]	[-0.02933994]	[0.01789135]	[0.23624162]	
factor	size	mom	mom	size	size	size	bm	bm	mom	
10		-4.198	-4.742	1.46	-1.7	0.17	-1.765	-4.897	0.593	6.619
20		-1.264	-1.863	0.876	-0.551	0.542	-0.894	-2.634	1.252	3.468
50		-1.435	-0.858	-0.845	-0.873	0.4	-0.495	-1.457	0.409	1.617
100		-0.098	-0.722	-0.46	-0.468	0.503	-0.216	-1.437	0.408	0.29
200		-0.35	-0.662	-0.861	-0.51	0.31	0.489	-0.609	0.451	0.054
300		-0.339	-0.682	-0.113	-0.619	0.49	0.467	-0.668	0.396	-0.137
450		-0.389	-0.185	-0.171	-0.471	-0.007	0.493	-0.361	0.333	-0.211
900		-0.495	-0.248	0.068	-0.385	-0.535	0.342	0.107	-0.434	-0.046
TWII	-0.00230316	0.058471198	0.067658949	0.005020771	0.037416184	0.084111004	-0.036832399	0.003091703	-0.001957628	

Win Rate	
10	0.4444
20	0.4444
50	0.3333
100	0.3333
200	0.4444
300	0.3333
450	0.2222
900	0.3333

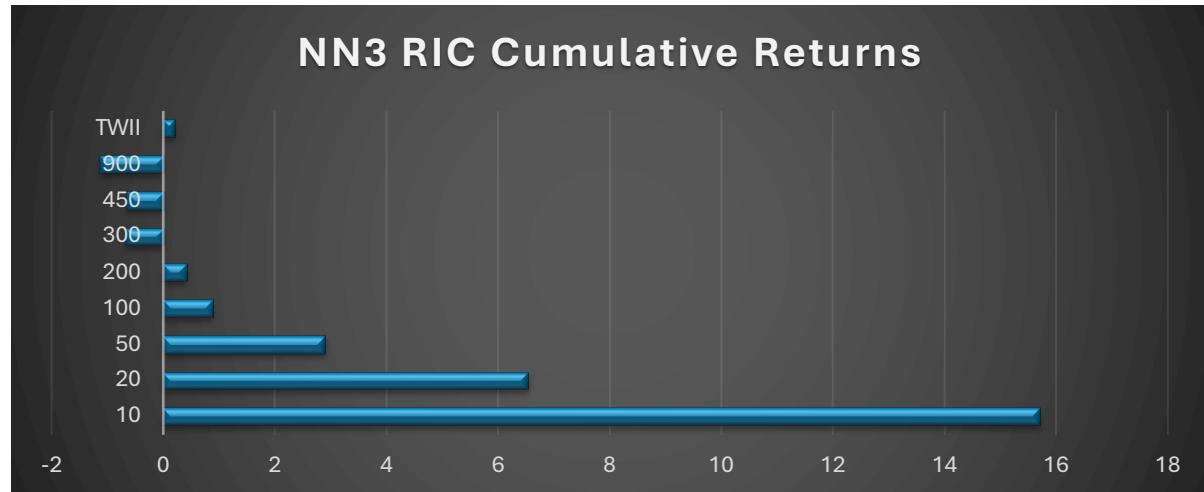
4.1.6 Neural Networks 3 Layers

4.1.6.1 NIC



Win Rate	
10	0.2222
20	0.2222
50	0.2222
100	0.3333
200	0.3333
300	0.4444
450	0.5556
900	0.6667

4.1.6.2 RIC

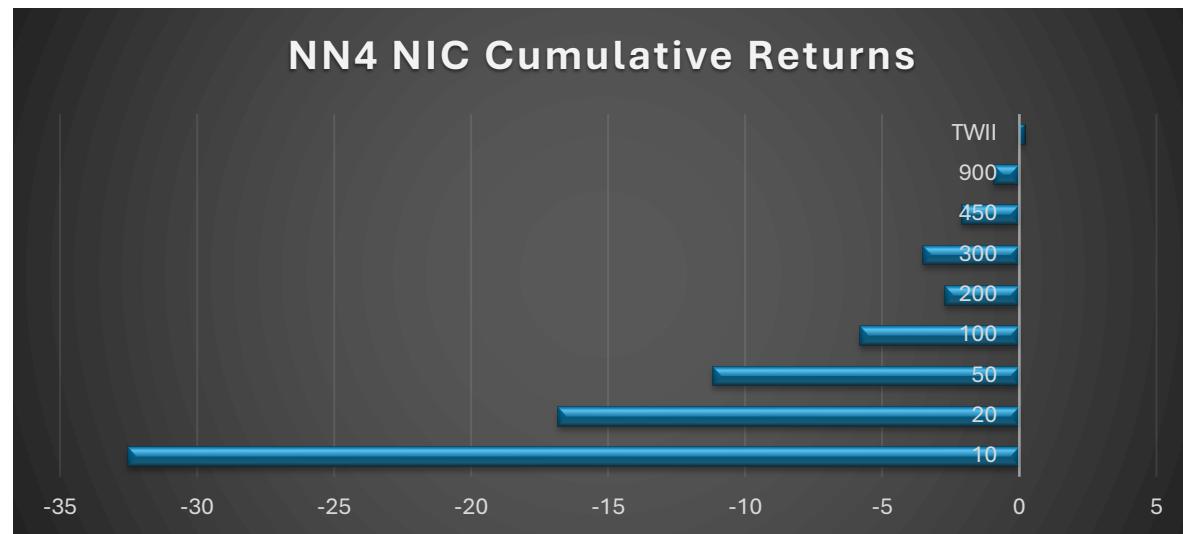


NN3 RIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	[0.01638416]	[-0.00950895]	[0.03036497]	[0.03398754]	[0.02259917]	[-0.10894226]	[0.02260638]	[0.03330391]	[0.03140683]
size	[-0.07513451]	[-0.03871283]	[0.00757986]	[-0.03752512]	[0.03660339]	[-0.04133125]	[0.11949391]	[-0.09144165]	[-0.08216858]
mom	[0.03799679]	[0.15908034]	[-0.00813142]	[-0.01049532]	[0.00980292]	[0.01554256]	[0.01492903]	[-0.00292881]	[0.00211406]
factor	size	mom	bm	size	size	bm	size	size	size
10	-4.198	-4.742	5.786	1.7	0.17	2.213	5.873	-1.929	10.847
20	-1.264	-1.863	1.326	0.551	0.542	2.335	0.414	-1.576	6.08
50	-1.435	-0.858	1.122	0.873	0.4	0.764	1.309	-0.797	1.522
100	-0.098	-0.722	1.171	0.468	0.503	0.28	-0.84	-0.364	0.492
200	-0.35	-0.662	0.881	0.51	0.31	0.73	-0.939	-0.09	0.037
300	-0.339	-0.682	0.594	0.619	0.49	0.458	-0.905	-0.719	-0.188
450	-0.389	-0.185	0.484	0.471	-0.007	0.532	-0.955	-0.497	-0.102
900	-0.495	-0.248	0.111	0.385	-0.535	0.639	-0.538	-0.259	-0.186
TWII	-0.00230316	0.058471198	0.06765895	0.005020771	0.037416184	0.084111004	-0.0368324	0.003091703	-0.00195763

Win Rate	
10	0.6667
20	0.6667
50	0.6667
100	0.5556
200	0.5556
300	0.4444
450	0.3333
900	0.3333

4.1.7 Neural Networks 4 Layers

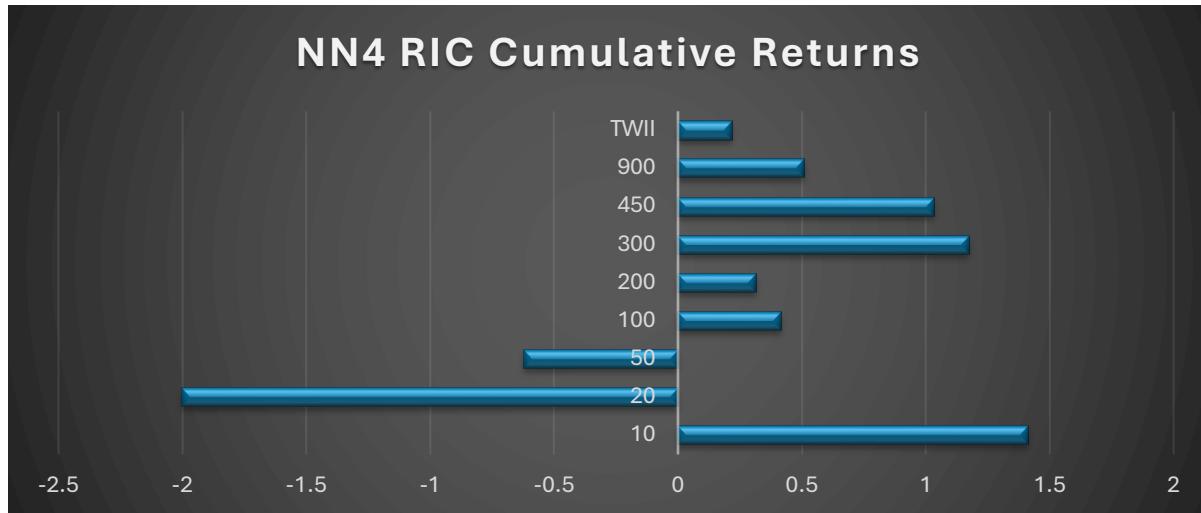
4.1.7.1 NIC



NN4 NIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	[0.03456341]	[0.0495698]	[0.04445408]	[0.05531401]	[0.05083453]	[0.04210709]	[0.04162126]	[0.04015071]	[0.03857522]
size	[-0.0360114]	[0.03139416]	[-0.07815103]	[0.07938245]	[-0.20898467]	[-0.04446961]	[-0.01902949]	[-0.05203696]	[0.11062551]
mom	[-0.00134969]	[-0.01947131]	[-0.01946757]	[-0.0029039]	[-0.00469471]	[-0.02287491]	[-0.00242479]	[-0.00737059]	[-0.00826335]
factor	size	bm	size	size	size	size	size	size	size
10	-4.198	-2.497	-4.519	-1.7	-0.17	-1.765	-4.897	-1.929	-10.847
20	-1.264	-2.179	-1.111	-0.551	-0.542	-0.894	-2.634	-1.576	-6.08
50	-1.435	-1.978	-2.236	-0.873	-0.4	-0.495	-1.457	-0.797	-1.522
100	-0.098	-0.343	-1.888	-0.468	-0.503	-0.216	-1.437	-0.364	-0.492
200	-0.35	-0.024	-1.275	-0.51	-0.31	0.489	-0.609	-0.09	-0.037
300	-0.339	0.039	-1.36	-0.619	-0.49	0.467	-0.668	-0.719	0.188
450	-0.389	0.132	-1.126	-0.471	0.007	0.493	-0.361	-0.497	0.102
900	-0.495	0.109	-1.051	-0.385	0.535	0.342	0.107	-0.259	0.186
TWII	-0.00230316	0.058471198	0.067658949	0.005020771	0.037416184	0.084111004	-0.0368324	0.003091703	-0.001957628

Win Rate	
10	0
20	0
50	0
100	0
200	0.1111
300	0.2222
450	0.3333
900	0.5556

4.1.7.2 RIC



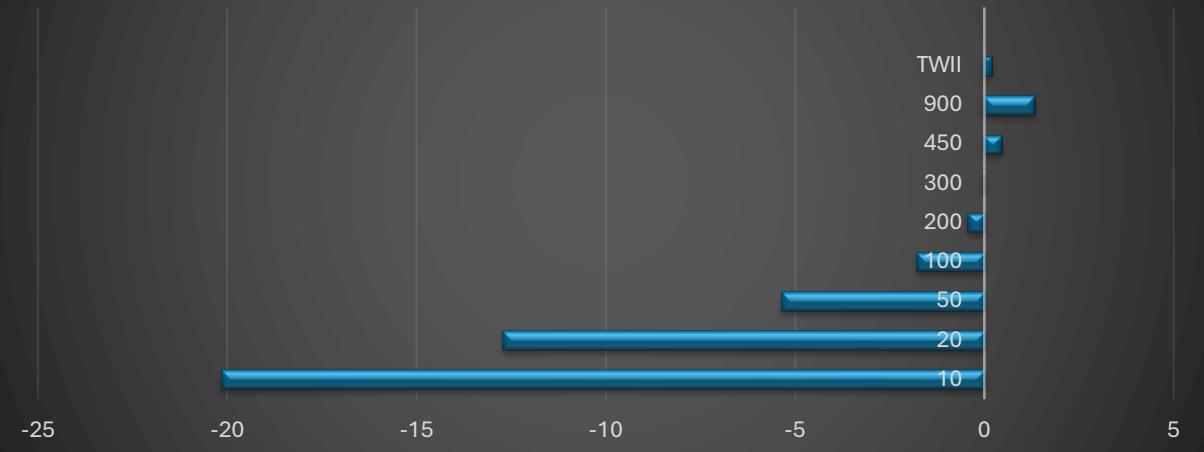
NN4 RIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	[0.05544316]	[0.07824133]	[0.03025283]	[0.00552266]	[0.03353512]	[-0.04666783]	[0.040579]	[0.02843816]	[0.0397702]
size	[-0.10765538]	[-0.00517324]	[-0.00860836]	[-0.00772883]	[0.02617317]	[0.00066536]	[0.074857]	[0.05310332]	[-0.01763847]
mom	[0.01285875]	[-0.00398764]	[-0.0018203]	[0.00569522]	[0.00125615]	[-0.00413804]	[-0.0040830]	[0.00503186]	[0.0248935]
factor	size	bm	bm	size	bm	bm	size	size	bm
10	-4.198	-2.497	5.786	1.7	-4.389	2.213	5.873	1.929	-5.005
20	-1.264	-2.179	1.326	0.551	-1.762	2.335	0.414	1.576	-2.997
50	-1.435	-1.978	1.122	0.873	-0.048	0.764	1.309	0.797	-2.026
100	-0.098	-0.343	1.171	0.468	0.606	0.28	-0.84	0.364	-1.193
200	-0.35	-0.024	0.881	0.51	0.516	0.73	-0.939	0.09	-1.1
300	-0.339	0.039	0.594	0.619	0.412	0.458	-0.905	0.719	-0.424
450	-0.389	0.132	0.484	0.471	0.243	0.532	-0.955	0.497	0.017
900	-0.495	0.109	0.111	0.385	-0.047	0.639	-0.538	0.259	0.085
TWII	-0.0023032	0.058471198	0.067658949	0.005020771	0.03741618	0.084111004	-0.0368324	0.003091703	-0.001957628

Win Rate	
10	0.5556
20	0.5556
50	0.5556
100	0.5556
200	0.5556
300	0.5556
450	0.7778
900	0.6667

4.1.8 Neural Networks 5 Layers

4.1.8.1 NIC

NN5 NIC Cumulative Returns

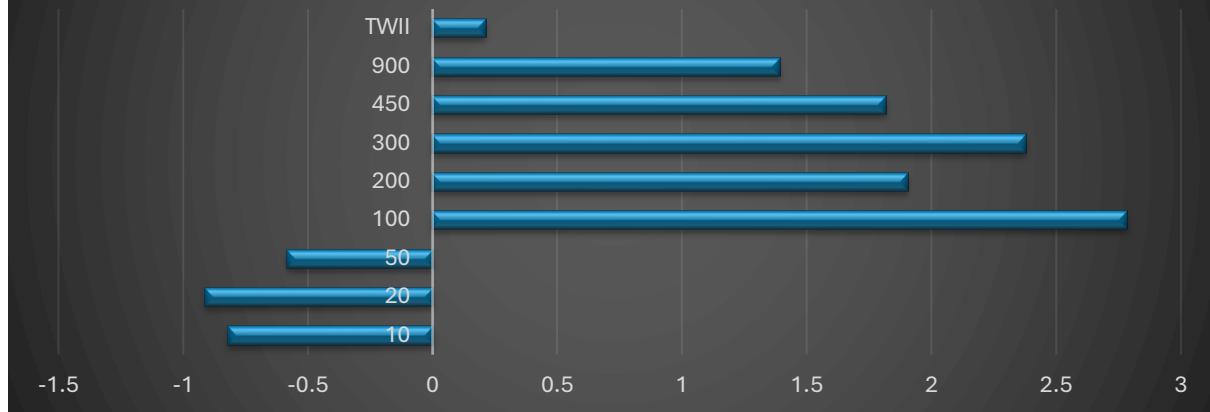


NN5 NIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	[0.03620095]	[0.02937705]	[0.04090631]	[0.0612235]	[-0.02769178]	[0.03948495]	[0.03800432]	[0.06628743]	[0.04762052]
size	[-0.02883772]	[-0.0407293]	[-0.03619811]	[-0.03791276]	[-0.03087997]	[-0.01283596]	[-0.06793988]	[0.01126786]	[-0.03653742]
mom	[-0.01080237]	[-0.004186]	[-0.00644965]	[0.00095678]	[-0.02691577]	[-0.0116225]	[-0.01728105]	[-0.01245626]	[-0.01174893]
factor	bm	size	bm	bm	size	bm	size	bm	bm
10	-6.878	-2.389	5.786	-2.823	-0.17	-2.213	-5.873	-0.593	-5.005
20	-3.018	-1.43	1.326	-2.044	-0.542	-2.335	-0.414	-1.252	-2.997
50	-1.156	-0.146	1.122	-0.271	-0.4	-0.764	-1.309	-0.409	-2.026
100	-0.376	-0.528	1.171	-0.512	-0.503	-0.28	0.84	-0.408	-1.193
200	0.038	0.147	0.881	0.151	-0.31	-0.73	0.939	-0.451	-1.1
300	0.007	0.02	0.594	0.199	-0.49	-0.458	0.905	-0.396	-0.424
450	0.109	-0.082	0.484	-0.16	0.007	-0.532	0.955	-0.333	0.017
900	0.233	0.044	0.111	-0.016	0.535	-0.639	0.538	0.434	0.085
TWII	-0.00230316	0.058471198	0.067658949	0.005020771	0.037416184	0.084111004	-0.036832399	0.003091703	-0.001957628

Win Rate	
10	0.1111
20	0.1111
50	0.1111
100	0.2222
200	0.5556
300	0.4444
450	0.4444
900	0.6667

4.1.8.2 RIC

NN5 RIC Cumulative Returns

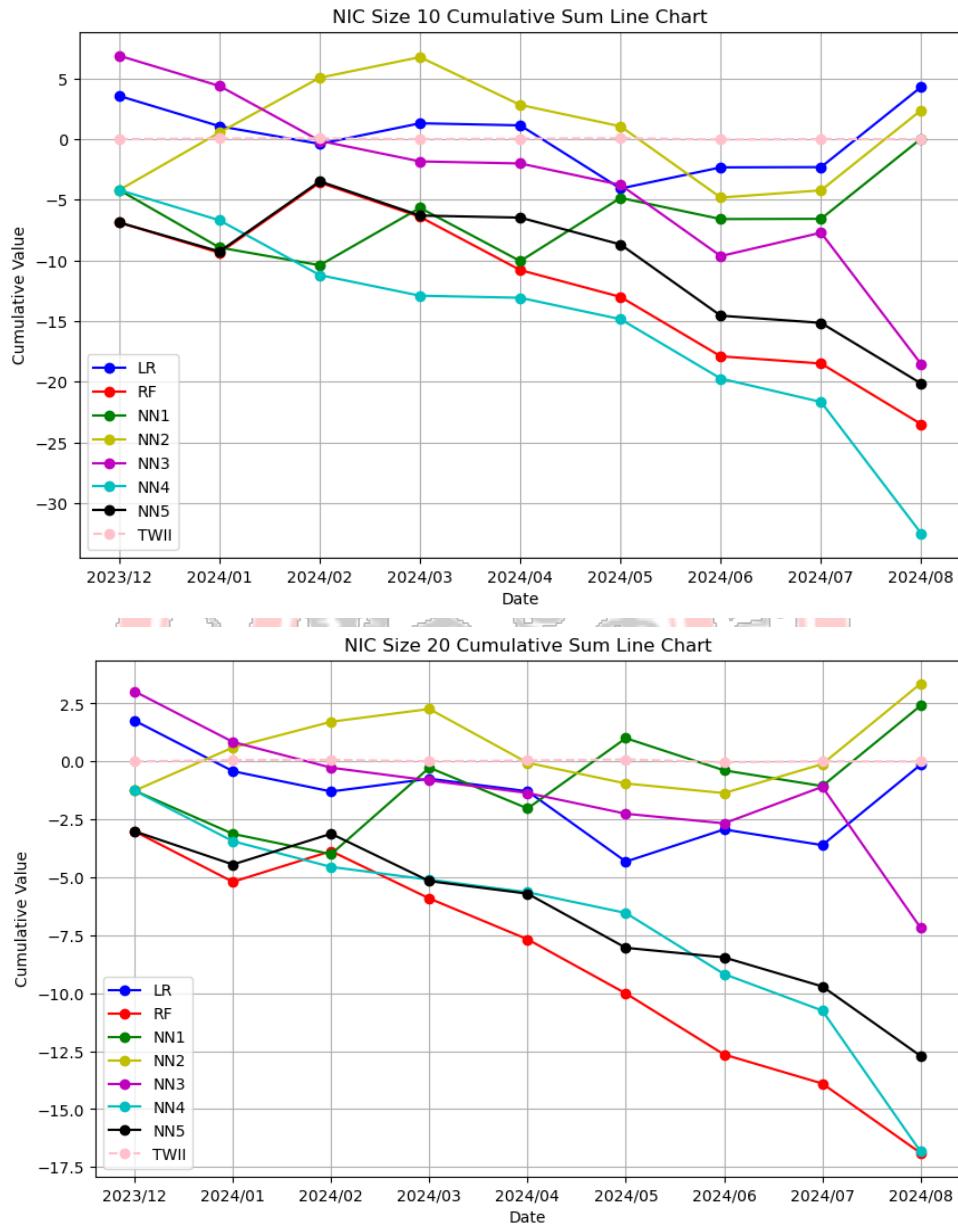


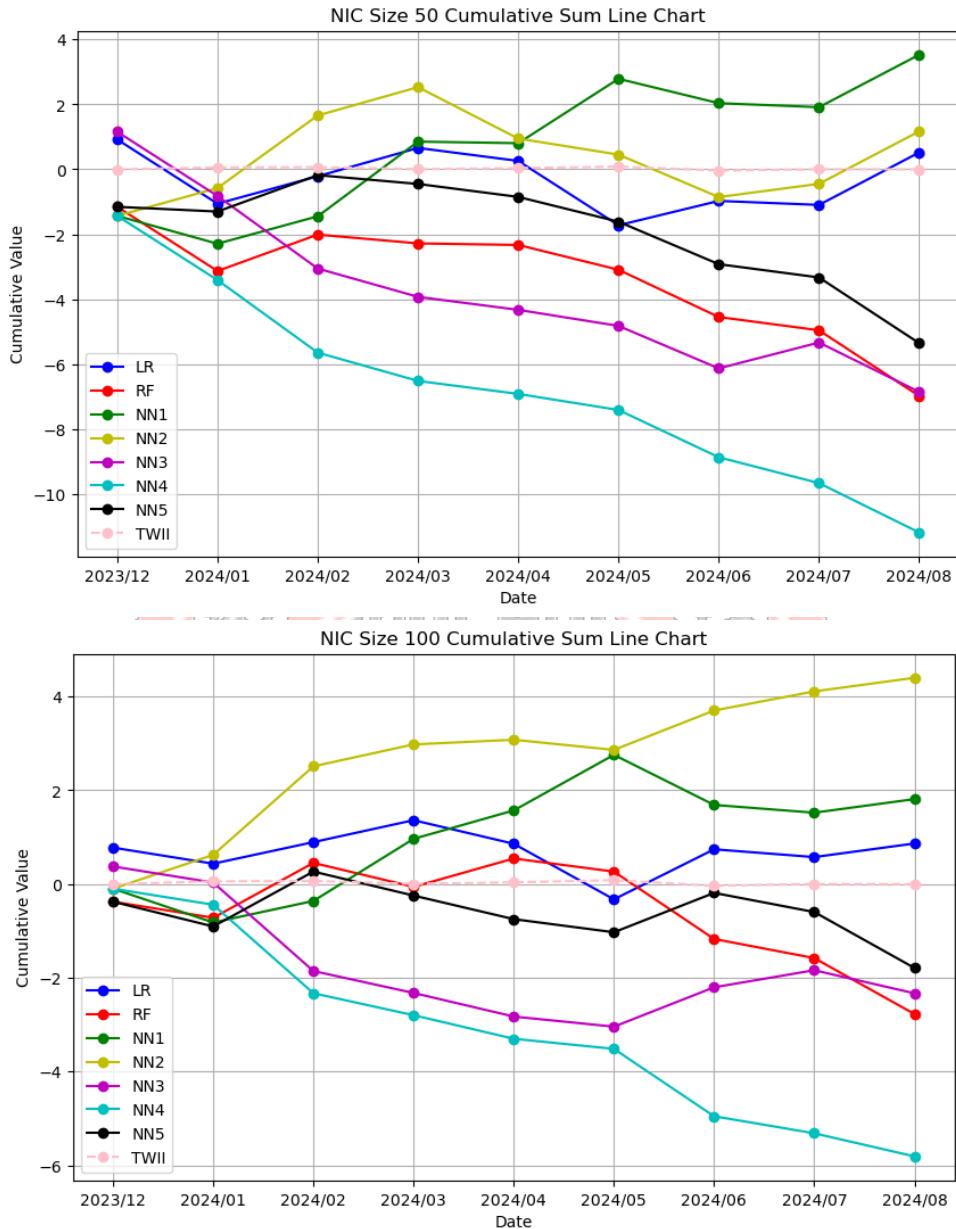
NN5 RIC	2023/12	2024/01	2024/02	2024/03	2024/04	2024/05	2024/06	2024/07	2024/08
bm	[0.03835627]	[0.05049802]	[0.03811085]	[0.03461165]	[0.04487702]	[0.04823294]	[0.05550754]	[0.04305641]	[0.02989147]
size	[-0.03461299]	[-0.0336175]	[0.07010408]	[-0.06459736]	[0.05453397]	[-0.03558518]	[-0.13130288]	[-0.03508753]	[-0.06090114]
mom	[0.00082086]	[0.01133518]	[-0.0091632]	[0.00627849]	[-0.00542392]	[0.00230027]	[0.00847249]	[0.01608812]	[-0.03675431]
factor	bm	bm	size	size	size	bm	size	bm	size
10	-6.878	-2.497	4.519	1.7	0.17	-2.213	-5.873	-0.593	10.847
20	-3.018	-2.179	1.111	0.551	0.542	-2.335	-0.414	-1.252	6.08
50	-1.156	-1.978	2.236	0.873	0.4	-0.764	-1.309	-0.409	1.522
100	-0.376	-0.343	1.888	0.468	0.503	-0.28	0.84	-0.408	0.492
200	0.038	-0.024	1.275	0.51	0.31	-0.73	0.939	-0.451	0.037
300	0.007	0.039	1.36	0.619	0.49	-0.458	0.905	-0.396	-0.188
450	0.109	0.132	1.126	0.471	-0.007	-0.532	0.955	-0.333	-0.102
900	0.233	0.109	1.051	0.385	-0.535	-0.639	0.538	0.434	-0.186
TWII	-0.00230316	0.058471198	0.067658949	0.005020771	0.037416184	0.084111004	-0.0368324	0.0030917	-0.001957628

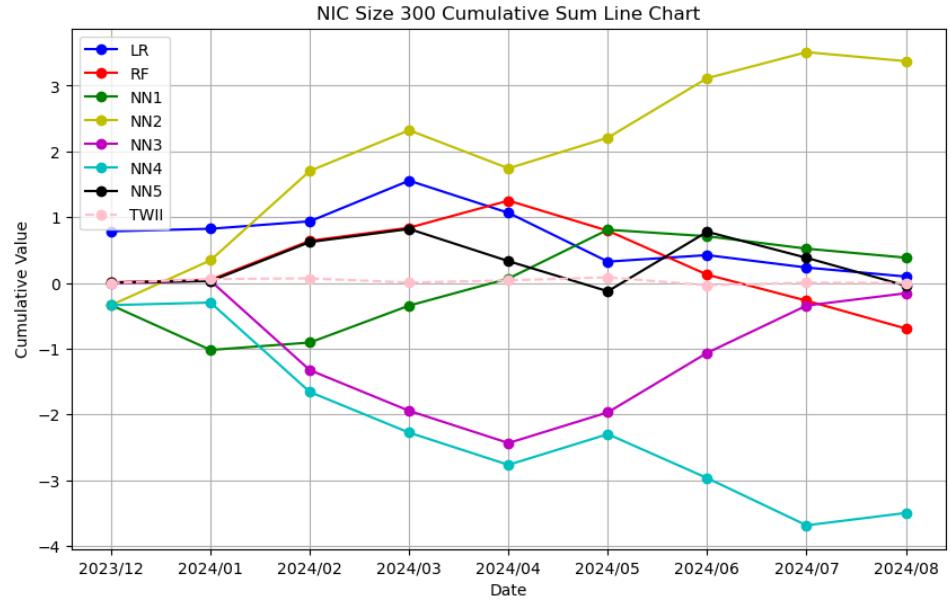
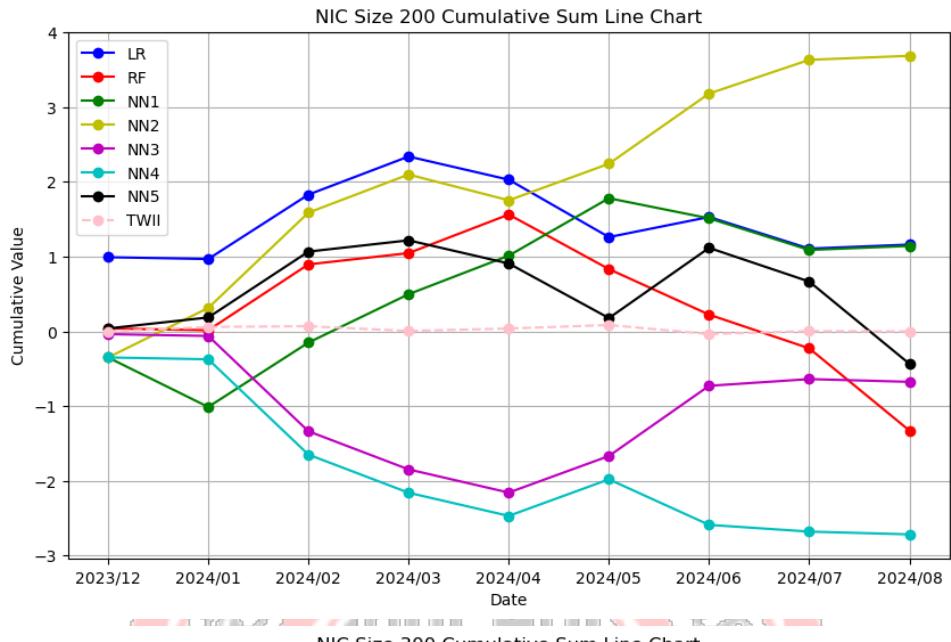
Win Rate	
10	0.4444
20	0.4444
50	0.4444
100	0.5556
200	0.6667
300	0.5556
450	0.5556
900	0.6667

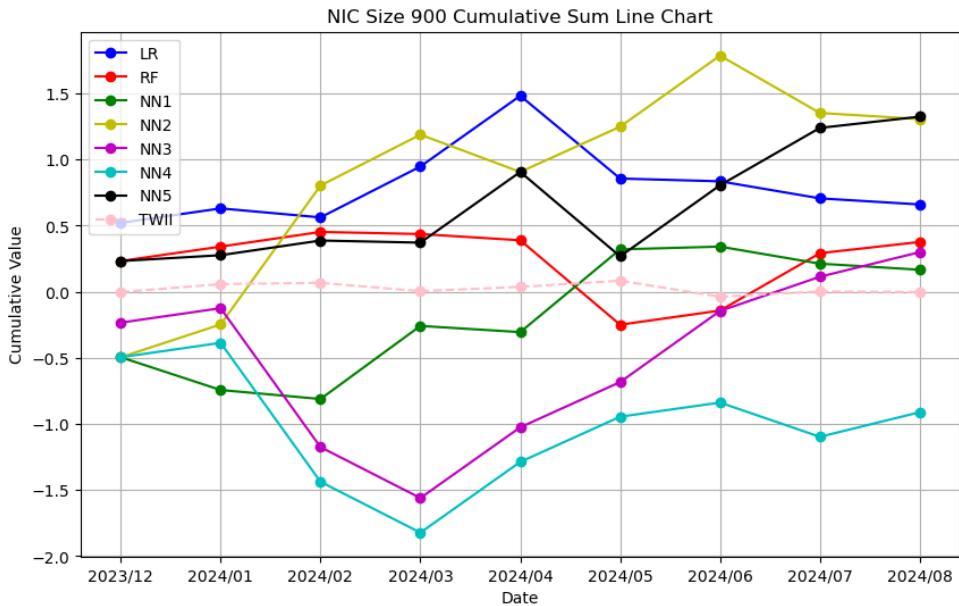
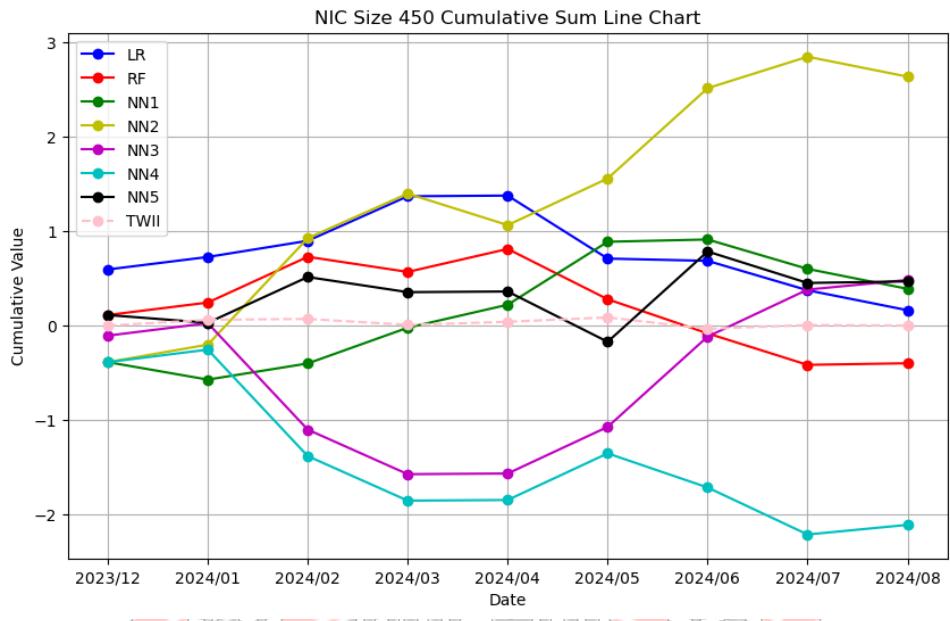
4.2 Comparison by Buy-Sell List Size and IC

4.2.1 NIC

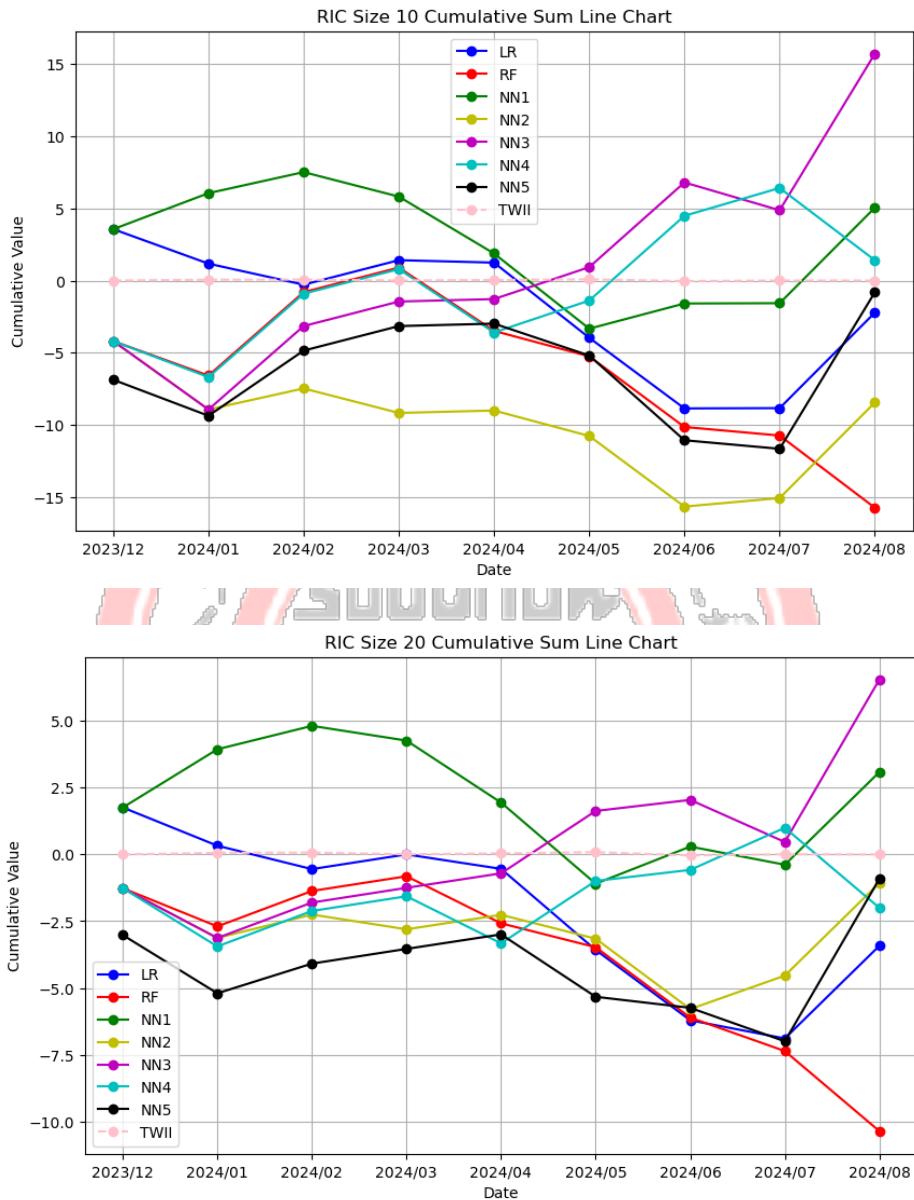


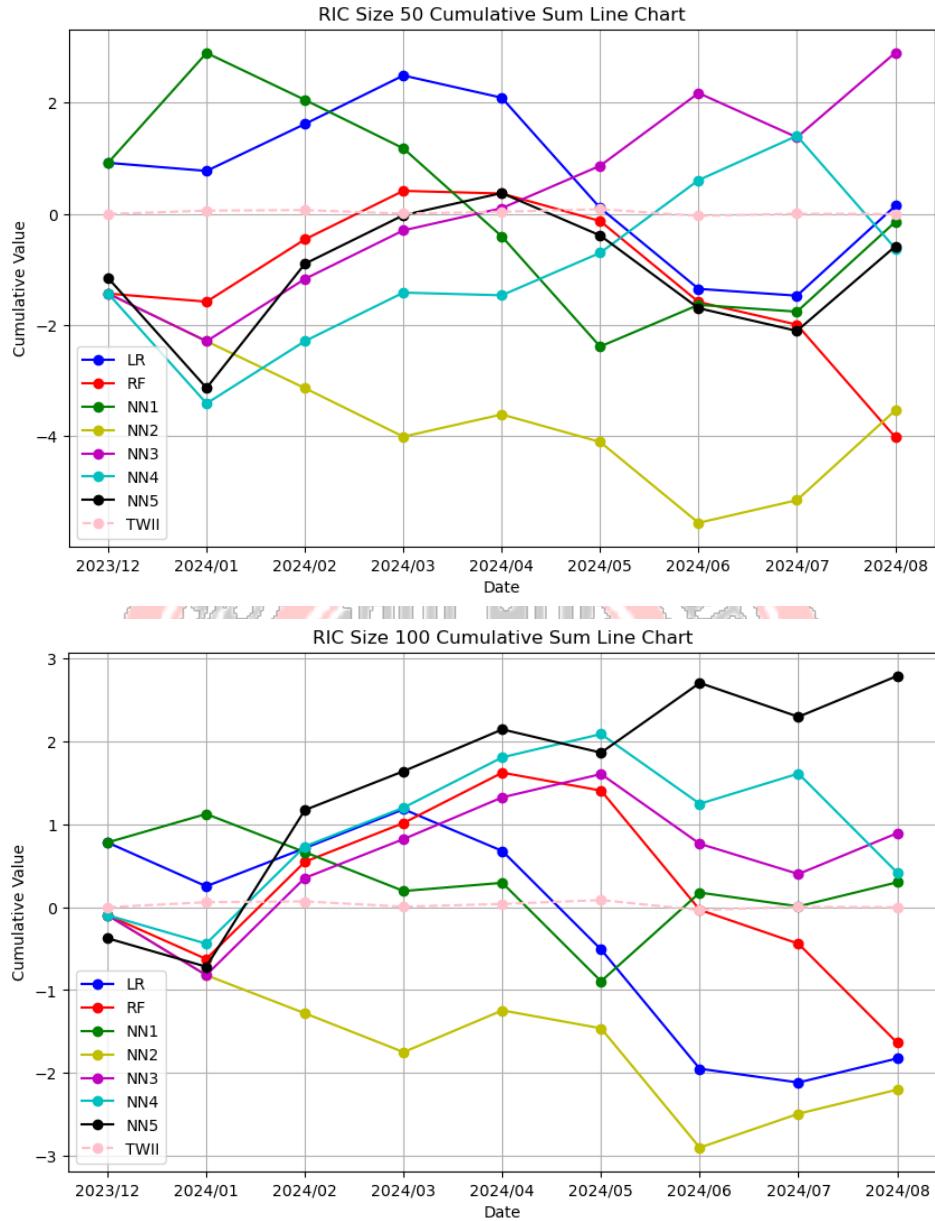


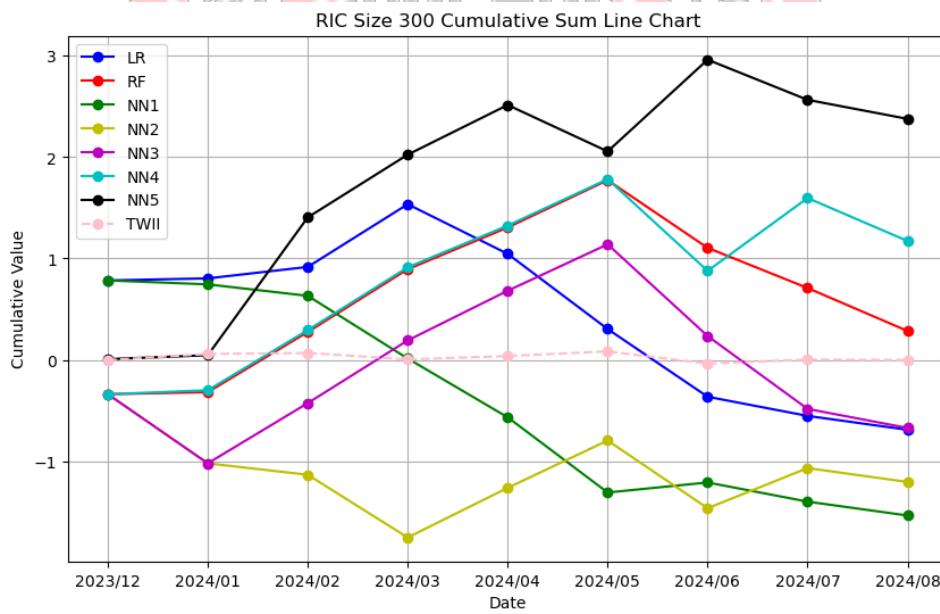
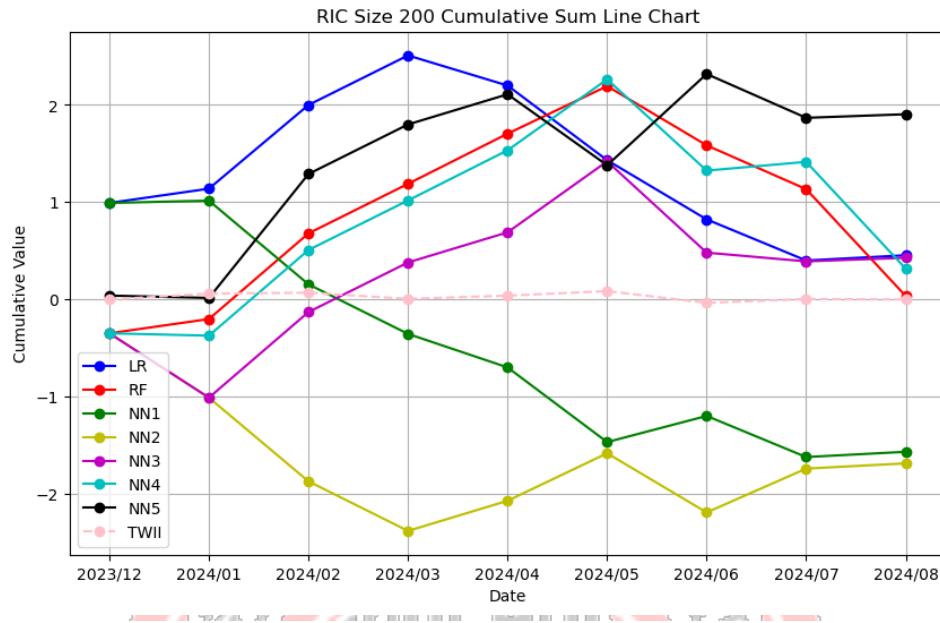


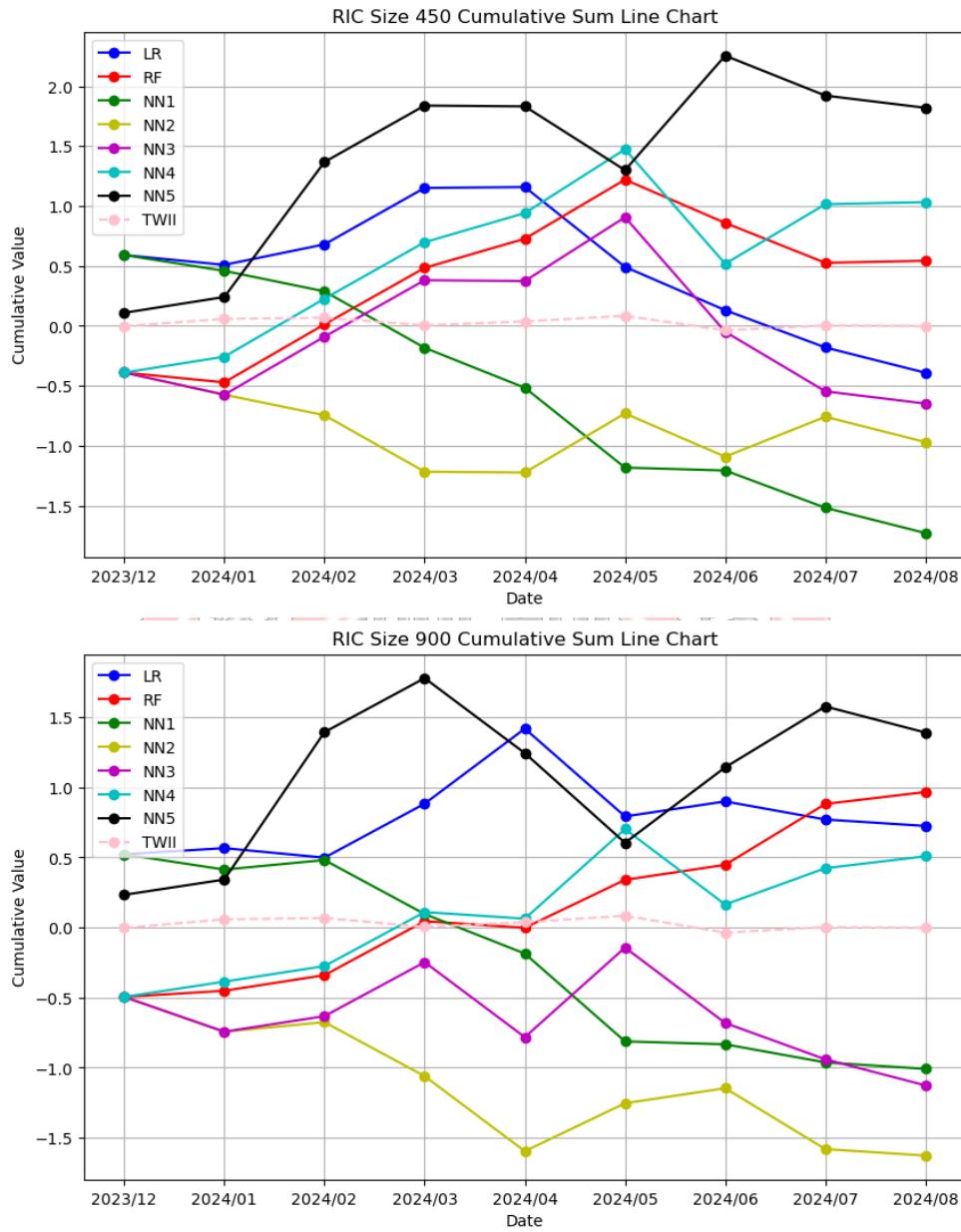


4.2.2 RIC









4.3 sharp_ratio

Sharpe Ratio for LR: LR_RIC		Sharpe Ratio for NN2: NN2_RIC	
10	-0.064443	10	-0.255285
20	-0.184752	20	-0.063844
50	0.013529	50	-0.388383
100	-0.262154	100	-0.396611
200	0.079325	200	-0.361119
300	-0.144581	300	-0.275123
450	-0.107555	450	-0.328165
900	0.219213	900	-0.582068
TWII	0.597466	TWII	0.597466
dtype: object		dtype: object	
Sharpe Ratio for RF: RF_RIC		Sharpe Ratio for NN3: NN3_RIC	
10	-0.485604	10	0.341063
20	-0.835344	20	0.2977
50	-0.422467	50	0.299001
100	-0.216166	100	0.150701
200	0.005567	200	0.076254
300	0.063129	300	-0.120199
450	0.16089	450	-0.14232
900	0.382473	900	-0.297813
TWII	0.597466	TWII	0.597466
dtype: object		dtype: object	
Sharpe Ratio for NN1: NN1_RIC		Sharpe Ratio for NN4: NN4_RIC	
10	0.150824	10	0.036547
20	0.159605	20	-0.118451
50	-0.01136	50	-0.05049
100	0.047965	100	0.062138
200	-0.301562	200	0.049051
300	-0.370095	300	0.228848
450	-0.547936	450	0.228882
900	-0.350919	900	0.148035
TWII	0.597466	TWII	0.597466

```
Sharpe Ratio for NN5: 10      -0.016798
20      -0.037159
50      -0.046087
100     0.406747
200     0.334476
300     0.42673
450     0.3639
900     0.287338
TWII    0.597466
dtype: object
```

利用報酬率的平均除以報酬率的標準差計算投資組合的 sharp ratio。我們可以看到雖然部分模型在報酬的表現上很亮眼，但是因為波動度太大，其風險報酬比其實很差。

五、結論 Conclusion

本實驗結果揭示了不同模型在股票市場報酬預測中的表現特性，帶來了幾個重要啟示：

1. 線性模型的穩健性

- 儘管線性回歸是最簡單的模型，但其在實驗中的表現令人意外地穩健。這表明，當數據特徵具有線性關係或噪聲較高時，簡單模型可能比複雜模型更能有效捕捉核心模式，同時避免過擬合問題。
- 此結果也提醒研究者，不應過度依賴複雜模型，尤其在數據維度有限的情況下。

2. 隨機森林表現不佳的挑戰

- 隨機森林未能達到預期的效果，表明其可能不適合處理當前數據特徵的特性，或者存在過度依賴單一特徵、過擬合等問題。這強調了在選擇模型時，應根據數據特徵進行謹慎的實驗與測試。亦或是在參數選擇上需更進一步去研究，實驗中選擇的 $\text{max_depth}=3$ 和 $n_{\text{estimators}}=100$ 可能並非最適合此數據集的參數，應多測試更多組合以達到期望效果。

3. 深層神經網路的複雜性與限制

- 雖然多層神經網路理論上能捕捉更複雜的非線性關係，但過深的結構（如 5 層網路）可能因數據量不足或噪聲過多，導致性能下降。相比之下，2 層和 3 層神經網路在簡單結構中更能有效學習數據特徵，並在性能上明顯優於其他模型。

4. 大盤作為基準的局限性

- 實驗中，部分模型表現優於大盤回報，表明以大盤回報作為基準可能低估了模型的預測能力。未來研究可考慮更動態和挑戰性的基準指標，如風險調整後的回報或專注於特定資產群體的表現。

六、心得

這次實驗的過程和結果讓我對資產定價模型以及機器學習在金融市場中的應用有了更深刻的認識，也讓我明白了理論與實踐之間的差距，以及模型選擇的重要性。

1. 模型與數據的適配性

實驗中，線性回歸模型的表現超出預期，這讓我深刻體會到，簡單的模型在某些情況下可能比複雜模型更加有效。儘管線性回歸在表現能力上有限，但它的穩定性和抗噪能力在本次實驗中顯現出優勢。相比之下，隨機森林和多層神經網路等更為複雜的模型，由於過度依賴數據的非線性特徵和複雜關係，反而表現不如預期，特別是在樣本數量有限或數據質量一般的情況下。

這讓我意識到，模型的選擇不能一味追求複雜，應根據數據特徵與模型能力的匹配程度做出判斷。同時，數據的質量和特徵工程在機器學習中是不可忽視的基石。

2. 機器學習模型的挑戰

在模型表現中，隨機森林的結果不理想，這讓我反思其在金融市場應用中的適用性。隨機森林擅長處理非線性和高維數據，但若數據特徵中非線性關係並不顯著，其優勢可能無法充分發揮。此外，隨機森林對因子數量多寡的敏感性也提示我，在未來研究中應更加注重因子選擇的合理性，避免無效或冗餘特徵影響模型性能。

對於多層神經網路，實驗中深層模型（如 5 層網路）的表現劣於淺層網路，讓我進一步認識到模型複雜性帶來的挑戰。深層模型需要大量數據來支持其訓練，而數據不足會導致過擬合問題。此外，深層網路的梯度消失和訓練難度也讓我思考，未來是否可以通過正則化技術（如 Dropout）或改進的網路結構（如殘差網路）來改善這些問題。

3. 實驗設計與基準的重要性

實驗中的基準設置為大盤回報，而部分模型能超越大盤表現，說明這些模型至少能捕捉到部分有效信息。然而，這也提示我，未來研究應考慮設置更加嚴格或多元的基準，如風險調整後的回報（例如夏普比率）或其他市場基準。這將更全面地評估模型在實際應用中的價值。

此外，實驗讓我了解到模型的評估不僅是關注回報率本身，還需結合穩定性、風險管理能力以及實際應用中的交易成本等因素，才能提供更加全面的結論。

4. 研究過程的反思

在實驗過程中，我深刻感受到機器學習與傳統經濟學理論結合的重要性。機器學習提供了強大的非線性建模能力，但它本質上是一種數據驅動的方法，缺乏經濟學理論的支撐可能導致結果難以解釋。在這次實驗中，將 Fama-French 三因子模型作為研究基礎，結合機器學習技術，為預測股票市場報酬提供了新視角，也讓我意識到，理論與數據驅動的方法結合是未來研究的關鍵方向。

5. 報酬與風險的平衡

本次實驗不僅評估了模型在報酬率上的表現，還通過計算夏普比率（Sharpe Ratio）來分析風險調整後的收益。夏普比率通過將報酬率的平均值除以其標準差，量化了每單位風險所帶來的超額回報，提供了一個更全面的投資評估指標。

儘管部分模型在報酬率上表現亮眼，但其波動性較高，導致夏普比率偏低。例如，某些神經網路模型能產生顯著高於大盤的收益，但其報酬的不穩定性使得投資者在實際應用中可能面臨較大風險。這表明，僅關注絕對收益可能會掩蓋模型在穩健性方面的缺陷，而夏普比率的引入幫助揭示了這些隱藏的風險。

6. 未來展望

這次實驗的結果讓我對未來研究充滿期待和啟發。我希望能進一步提升數據的質量和規模，嘗試更多樣化的特徵選擇方法，並探索更輕量化的深度學習結構。同時，我也計劃結合強化學習及元學習框架，探索因子與報酬的因果關係，而非僅僅依賴於相關性。

此外，未來的研究將更加注重模型的解釋性和實用性。我希望能將實驗結果應用於更具挑戰性的交易策略中，例如高頻交易或跨市場套利，從而驗證模型在真實市場條件下的適用性。

總結

這次實驗讓我對機器學習在金融市場應用中的潛力與挑戰有了全面的理解。它不僅是一個技術工具，也是一種需要經濟學理論支持的框架。未來，我將以更嚴謹的實驗設計、更高質量的數據處理，以及更多元化的模型探索，進一步深入研究，為資產定價和市場預測提供更加創新的解決方案。



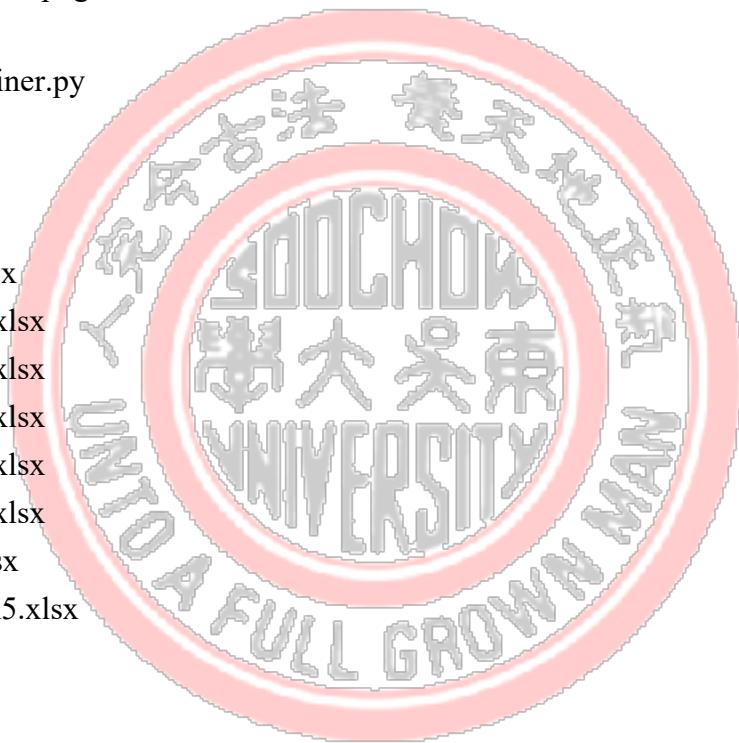
七、附錄 Appendix

7.1 專案結構

```
├── analyzer.py  
├── config.py  
└── data  
    ├── bm_filterd.xlsx  
    ├── index.xlsx  
    ├── momentum_filterd.xlsx  
    ├── pb.xlsx  
    ├── price.xlsx  
    ├── price_filterd.xlsx  
    ├── returns_filterd.xlsx  
    ├── size.xlsx  
    └── size_filterd.xlsx  
├── data_loader.py  
├── data_processor.py  
├── factor_analyzer.py  
├── feature_calculator.py  
├── get_ic.py  
└── images  
    ├── nic_10.png  
    ├── nic_100.png  
    ├── nic_20.png  
    ├── nic_200.png  
    ├── nic_300.png  
    ├── nic_450.png  
    ├── nic_50.png  
    └── nic_900.png
```



```
|   ├── ric_10.png  
|   ├── ric_100.png  
|   ├── ric_20.png  
|   ├── ric_200.png  
|   ├── ric_300.png  
|   ├── ric_450.png  
|   ├── ric_50.png  
|   └── ric_900.png  
├── main.py  
├── model_trainer.py  
├── plot.ipynb  
├── plot.py  
└── results  
    ├── lr.xlsx  
    ├── nn1.xlsx  
    ├── nn2.xlsx  
    ├── nn3.xlsx  
    ├── nn4.xlsx  
    ├── nn5.xlsx  
    ├── rf.xlsx  
    └── ~$nn5.xlsx  
└── trainer.py
```



7.2 程式碼

7.2.1 main.py

```
❶ main.py > ...
1  from data_loader import load_and_process_data
2  from get_ic import get_ic
3  from trainer import train
4  from analyzer import analyze
5
6
7  def main():
8      index_data, size, bm, returns, momentum = load_and_process_data()
9      size_nic, bm_nic, momentum_nic, size_ric, bm_ric, momentum_ric, bm_rank, momentum_rank, size_rank = get_ic(size, bm, returns, momentum)
10     nic, ric = train(size, bm, returns, momentum, size_nic, bm_nic, momentum_nic, size_ric, bm_ric, momentum_ric, 'nn5')
11
12
13     analyze(nic, ric, bm, momentum, size, bm_rank, momentum_rank, size_rank, index_data, returns, 'nn5')
14
15
16     if __name__ == "__main__":
17         main()
```

7.2.2 data_processor.py

```
import pandas as pd
import numpy as np

class DataProcessor:
    def __init__(self, price_path, size_path, pb_path):
        self.price_path = price_path
        self.size_path = size_path
        self.pb_path = pb_path
        self.price_data = None
        self.size_data = None
        self.pb_data = None

    def load_data(self):
        """
        載入原始數據。
        """
        self.price_data = pd.read_excel(self.price_path, header=1)
        self.size_data = pd.read_excel(self.size_path, header=1)
        self.pb_data = pd.read_excel(self.pb_path, header=1)

    def filter_stocks(self, df):
        """
        過濾存在於 2005/01 並且符合代號格式的股票。
        """
        df_filtered = df[df['2005/01'].notna()]
        filtered_df = df_filtered[df_filtered['代號'].str.match(r'^\d{4}$')]
        return filtered_df[filtered_df['代號'].str.match(r'^[1-9]\d{3}$')].iloc[:, :-1].reset_index(drop = True)

    def get_stock_list(self, df):
        return df['代號']
```

```

def size_filter(self, df, stk_num):
    size_filtered = df[df['代號'].isin(stk_num)]
    size_filtered.iloc[:, 2:] = np.log10(size_filtered.iloc[:, 2:] * 1_000_000)
    return size_filtered.iloc[:, :-1].reset_index(drop = True)

def bm_filter(self, df, stk_num):
    """
    計算股價淨值比。
    """
    pb_final = df[df['代號'].isin(stk_num)]
    bm = pb_final.iloc[:, 2: ].apply(lambda x: 1 / x)
    bm_indices = pb_final.iloc[:, :2]
    return pd.concat([bm_indices, bm], axis=1).iloc[:, :-1].reset_index(drop = True)

def gen_monthly_returns(self, df):
    """
    計算每月回報率。
    """
    df_data = df.iloc[:, 2:]
    returns = np.log(df_data.shift(-1, axis=1) / df_data)
    return pd.concat([df.iloc[:, :2], returns], axis=1).iloc[:, :-1]

def momentum(self, df):
    """
    計算動能。
    """
    numeric_data = df.iloc[:, 2:]
    prev_month = numeric_data.shift(1, axis=1)
    twelve_months_ago = numeric_data.shift(12, axis=1)
    log_mom = np.log(prev_month / twelve_months_ago)
    log_mom = log_mom.mask((prev_month <= 0) | (twelve_months_ago <= 0), np.nan)
    df_copy = df.copy()
    df_copy.iloc[:, 2:] = log_mom
    return df_copy

def fill_na(self, df):
    """
    填補缺失值。
    """
    df_copy = df.copy()
    df_copy.replace([np.inf, -np.inf], np.nan, inplace=True)
    df_copy.iloc[:, 2:] = df_copy.iloc[:, 2: ].apply(lambda col: col.fillna(col.median()), axis=0)
    return df_copy

def roundTo3(self, df):
    return df.iloc[2:, :].apply(lambda x : round(x, 3))

```

7.2.3 data_loader.py

```
import config
from data_processor import DataProcessor
import pandas as pd
from feature_calculator import FeatureCalculator

def load_and_process_data():
    index_data = pd.read_excel(config.INDEX_PATH, index_col=0)
    processor = DataProcessor(config.PRICE_PATH, config.SIZE_PATH, config.PB_PATH)
    processor.load_data()

    # getting wished factors data
    price = processor.filter_stocks(processor.price_data)
    stock_list = processor.get_stock_list(price)
    size = processor.size_filter(processor.size_data, stock_list)
    bm = processor.bm_filter(processor.pb_data, stock_list)
    returns = processor.gen_monthly_returns(price)
    momentum = processor.momentum(price)

    size = processor.fill_na(size)
    bm = processor.fill_na(bm)
    returns = processor.fill_na(returns)
    momentum = processor.fill_na(momentum)

    # rounding to three decimal place
    size = processor.roundTo3(size)
    bm = processor.roundTo3(bm)
    returns = processor.roundTo3(returns)
    momentum = processor.roundTo3(momentum)

    size.to_excel('./data/size_filtered.xlsx')
    bm.to_excel('./data/bm_filtered.xlsx')
    returns.to_excel('./data/returns_filtered.xlsx')
    momentum.to_excel('./data/momentum_filtered.xlsx')
    price.to_excel('./data/price_filtered.xlsx')

    return index_data, size, bm, returns, momentum
```

7.2.4 feature_calculator.py

```
import numpy as np
import pandas as pd

class FeatureCalculator:
    @staticmethod
    def rank_columns(df):
        """
        對每列進行排名。
        """
        df_copy = df.copy()
        df_copy.iloc[:, 2:] = df_copy.iloc[:, 2:].rank(axis=0, method='min', ascending=True)
        return df_copy

    @staticmethod
    def calculate_correlation(df1, df2):
        """
        計算相關性。
        """
        df1_selected = df1.iloc[:, 2:]
        df2_selected = df2.iloc[:, 2:]
        return df1_selected.corrwith(df2_selected, axis=0)
```

7.2.5 get_ic.py

```
from feature_calculator import FeatureCalculator

def get_ic(size, bm, returns, momentum):
    calculator = FeatureCalculator()
    size_rank = calculator.rank_columns(size)
    bm_rank = calculator.rank_columns(bm)
    momentum_rank = calculator.rank_columns(momentum)

    # get nic
    size_nic = calculator.calculate_correlation(size_rank, returns)
    bm_nic = calculator.calculate_correlation(bm_rank, returns)
    momentum_nic = calculator.calculate_correlation(momentum_rank, returns)

    # get ric
    size_ric = calculator.calculate_correlation(size_rank, returns)
    bm_ric = calculator.calculate_correlation(bm_rank, returns)
    momentum_ric = calculator.calculate_correlation(momentum_rank, returns)

    return size_nic, bm_nic, momentum_nic, size_ric, bm_ric, momentum_ric, size_rank, bm_rank, momentum_rank
```

7.2.6 model_trainer.py

```
from sklearn.ensemble import RandomForestRegressor
from keras import Sequential
from keras import layers
from keras import optimizers
from keras import Input

class ModelTrainer:
    def __init__(self):
        self.models = {}

    def data_process(self, x, ic, i):
        """
        數據處理：生成訓練和測試數據集。

        Parameters:
        x (DataFrame): 特徵數據。
        ic (DataFrame): 目標數據。
        i (int): 測試數據偏移量。

        Returns:
        Tuple: (X, y, test_X)
        """
        df_x = x.T.iloc[2:, :].astype(float)
        df_y = ic.T.astype(float)
        X = df_x.iloc[72:299].values
        y = df_y.iloc[72:299].values.ravel()
        test_X = df_x.iloc[299 + i:300 + i, :].values
        return X, y, test_X

    def lr(self):
        return LinearRegression()
```



```
def rf(self):
    return RandomForestRegressor(max_depth=3, random_state=0, n_estimators=100, verbose=1, n_jobs=-1)

def nn1(self):
    model = Sequential([
        Input(shape=(964,)),
        layers.Dense(32, activation='relu'),
        layers.Dense(1)
    ])
    adam = optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=adam, loss='mean_squared_error')
    return model

def nn2(self):
    model = Sequential([
        Input(shape=(964,)),
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(1)
    ])
    adam = optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=adam, loss='mean_squared_error')
    return model
```



```
def nn3(self):
    model = Sequential([
        Input(shape=(964,)),
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(8, activation='relu'),
        layers.Dense(1)
    ])
    adam = optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=adam, loss='mean_squared_error')
    return model

def nn4(self):
    model = Sequential([
        Input(shape=(964,)),
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(8, activation='relu'),
        layers.Dense(4, activation='relu'),
        layers.Dense(1)
    ])
    adam = optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=adam, loss='mean_squared_error')
    return model
```

```
def nn5(self):
    model = Sequential([
        Input(shape=(964,)),
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(8, activation='relu'),
        layers.Dense(4, activation='relu'),
        layers.Dense(2, activation='relu'),
        layers.Dense(1)
    ])
    adam = optimizers.Adam(learning_rate=0.01)
    model.compile(optimizer=adam, loss='mean_squared_error')
    return model
```

```
def predict(self, X, y, test_X, model):
    """
    使用給定的模型進行預測。

    Parameters:
    X (array-like): 訓練特徵數據。
    y (array-like): 訓練目標數據。
    test_X (array-like): 測試特徵數據。
    model (object): 機器學習模型。

    Returns:
    array-like: 預測結果。
    """
    if isinstance(model, Sequential):
        # 如果是神經網絡模型
        model.fit(X, y, epochs=10, batch_size=32, verbose=0)
        prediction = model.predict(test_X)
    else:
        # 如果是其他機器學習模型
        model.fit(X, y)
        prediction = model.predict(test_X)

    return prediction
```

```
def generate_results(self, x, ic, model_name, iterations=10):
    results = []
    for i in range(iterations):
        X, y, test_X = self.data_process(x, ic, i)
        model = getattr(self, model_name)()
        model.fit(X, y)
        prediction = self.predict(X, y, test_X, model)
        results.append(prediction[0])
    return results
```

```

def save_results_to_df(self, results_dict):
    # 將 NIC 和 RIC 的結果轉換為 DataFrame
    df_nic = pd.DataFrame({
        '2023/12': [results_dict['NIC']['bm'][0], results_dict['NIC']['size'][0], results_dict['NIC']['momentum'][0]],
        '2024/01': [results_dict['NIC']['bm'][1], results_dict['NIC']['size'][1], results_dict['NIC']['momentum'][1]],
        '2024/02': [results_dict['NIC']['bm'][2], results_dict['NIC']['size'][2], results_dict['NIC']['momentum'][2]],
        '2024/03': [results_dict['NIC']['bm'][3], results_dict['NIC']['size'][3], results_dict['NIC']['momentum'][3]],
        '2024/04': [results_dict['NIC']['bm'][4], results_dict['NIC']['size'][4], results_dict['NIC']['momentum'][4]],
        '2024/05': [results_dict['NIC']['bm'][5], results_dict['NIC']['size'][5], results_dict['NIC']['momentum'][5]],
        '2024/06': [results_dict['NIC']['bm'][6], results_dict['NIC']['size'][6], results_dict['NIC']['momentum'][6]],
        '2024/07': [results_dict['NIC']['bm'][7], results_dict['NIC']['size'][7], results_dict['NIC']['momentum'][7]],
        '2024/08': [results_dict['NIC']['bm'][8], results_dict['NIC']['size'][8], results_dict['NIC']['momentum'][8]],
    }, index=['bm', 'size', 'mom'])

    df_ric = pd.DataFrame({
        '2023/12': [results_dict['RIC']['bm'][0], results_dict['RIC']['size'][0], results_dict['RIC']['momentum'][0]],
        '2024/01': [results_dict['RIC']['bm'][1], results_dict['RIC']['size'][1], results_dict['RIC']['momentum'][1]],
        '2024/02': [results_dict['RIC']['bm'][2], results_dict['RIC']['size'][2], results_dict['RIC']['momentum'][2]],
        '2024/03': [results_dict['RIC']['bm'][3], results_dict['RIC']['size'][3], results_dict['RIC']['momentum'][3]],
        '2024/04': [results_dict['RIC']['bm'][4], results_dict['RIC']['size'][4], results_dict['RIC']['momentum'][4]],
        '2024/05': [results_dict['RIC']['bm'][5], results_dict['RIC']['size'][5], results_dict['RIC']['momentum'][5]],
        '2024/06': [results_dict['RIC']['bm'][6], results_dict['RIC']['size'][6], results_dict['RIC']['momentum'][6]],
        '2024/07': [results_dict['RIC']['bm'][7], results_dict['RIC']['size'][7], results_dict['RIC']['momentum'][7]],
        '2024/08': [results_dict['RIC']['bm'][8], results_dict['RIC']['size'][8], results_dict['RIC']['momentum'][8]],
    }, index=['bm', 'size', 'mom'])

    return df_nic, df_ric

```

7.2.6 trainer.py

```

from model_trainer import ModelTrainer

def train(size, bm, returns, momentum, size_nic, bm_nic, momentum_nic, size_ric, bm_ric, momentum_ric, model_name):
    """
    訓練模型並生成結果。
    Parameters:
    size_nic (DataFrame): NIC 尺寸因子數據。
    bm_nic (DataFrame): NIC BM 因子數據。
    momentum_nic (DataFrame): NIC 動量因子數據。
    size_ric (DataFrame): RIC 尺寸因子數據。
    bm_ric (DataFrame): RIC BM 因子數據。
    momentum_ric (DataFrame): RIC 動量因子數據。
    model_name (str): 模型名稱。
    """
    trainer = ModelTrainer()
    result_dic = {
        'NIC': {
            'size': [],
            'bm': [],
            'momentum': []
        },
        'RIC': {
            'size': [],
            'bm': [],
            'momentum': []
        }
    }

```

```

result_dic['NIC']['size'] = trainer.generate_results(size, size_nic, model_name)
result_dic['NIC']['bm'] = trainer.generate_results(bm, bm_nic, model_name)
result_dic['NIC']['momentum'] = trainer.generate_results(momentum, momentum_nic, model_name)
result_dic['RIC']['size'] = trainer.generate_results(size, size_ric, model_name)
result_dic['RIC']['bm'] = trainer.generate_results(bm, bm_ric, model_name)
result_dic['RIC']['momentum'] = trainer.generate_results(momentum, momentum_ric, model_name)

prediction_nic, prediction_ric = trainer.save_results_to_df(result_dic)

return prediction_nic, prediction_ric

```

7.2.7 factor_analyzer.py

```

import pandas as pd
import numpy as np

class FactorAnalysis:
    def __init__(self, prediction_nic, prediction_ric, bm, momentum, size, bm_rank, momentum_rank, size_rank, index_data, returns):
        """
        Initialize the class with required data.
        """
        self.prediction_nic = prediction_nic
        self.prediction_ric = prediction_ric
        self.bm = bm
        self.momentum = momentum
        self.size = size
        self.bm_rank = bm_rank
        self.momentum_rank = momentum_rank
        self.size_rank = size_rank
        self.index_data = index_data
        self.returns = returns
        self.IC = ['NIC', 'RIC']
        self.group_sizes = [10, 20, 50, 100, 200, 300, 450, 900]
        self.dates = prediction_ric.columns.to_list()
        self.comparison_results_dict = {
            "NIC": pd.DataFrame(index=self.group_sizes, columns=self.dates),
            "RIC": pd.DataFrame(index=self.group_sizes, columns=self.dates)
        }
        self.win_rate_results_dict = {
            "NIC": pd.DataFrame(index=self.group_sizes, columns=["Win Rate"]),
            "RIC": pd.DataFrame(index=self.group_sizes, columns=["Win Rate"])
        }
        self.select_dict_nic = {date: None for date in self.dates}
        self.select_dict_ric = {date: None for date in self.dates}
        self.index_returns = self.calculate_index_returns()

    def calculate_index_returns(self):
        """
        Calculate index returns from index data.
        """
        index_returns = np.log(self.index_data.shift(-1, axis=0) / self.index_data)
        index_returns = index_returns.T.iloc[:, :9]
        index_returns = index_returns.reset_index(drop=True)
        index_returns.columns.name = None
        return index_returns

```

```

def calculate_index_returns(self):
    """
    Calculate index returns from index data.
    """

    index_returns = np.log(self.index_data.shift(-1, axis=0) / self.index_data)
    index_returns = index_returns.T.iloc[:, :9]
    index_returns = index_returns.reset_index(drop=True)
    index_returns.columns.name = None
    return index_returns

```

```
def process_ic(self, ic):
    """
    Process a single IC (NIC or RIC).
    """
    output = pd.DataFrame(index=self.group_sizes, columns=self.dates)
    prediction_df = self.prediction_nic if ic == "NIC" else self.prediction_ric
    buy_list_total = {}
    sell_list_total = {}

    for date in self.dates:
        # Step 1: Find the factor with the largest squared value
        buy_list_total[date] = {}
        sell_list_total[date] = {}
        squared_series = prediction_df[date] ** 2
        factor = squared_series.idxmax()
        original_value = prediction_df.loc[factor, date]
        if ic == 'NIC':
            self.select_dict_nic[date] = factor
        else:
            self.select_dict_ric[date] = factor
        sign = original_value >= 0

        # Step 2: Determine factor data and rank
        if factor == "bm":
            factor_data = self.bm[date]
            factor_rank = self.bm_rank[date]
        elif factor in ["momentum", "mom"]:
            factor_data = self.momentum[date]
            factor_rank = self.momentum_rank[date]
```

```

    elif factor == "size":
        factor_data = self.size[date]
        factor_rank = self.size_rank[date]
    else:
        raise ValueError(f"Unknown factor: {factor}")

# Step 3: Rank and group the factor data
factor_rank_sorted = factor_rank.sort_values(ascending=False)
for size in self.group_sizes:
    factor_rank_sorted_group = pd.qcut(
        factor_rank_sorted, q=size, labels=False, duplicates="drop"
    ) + 1

    max_value = factor_rank_sorted_group.max()
    min_value = factor_rank_sorted_group.min()

    buy_list = (
        factor_rank_sorted_group[factor_rank_sorted_group == max_value].index
        if sign
        else factor_rank_sorted_group[factor_rank_sorted_group == min_value].index
    )

    sell_list = (
        factor_rank_sorted_group[factor_rank_sorted_group == min_value].index
        if sign
        else factor_rank_sorted_group[factor_rank_sorted_group == max_value].index
    )

buy_return_values = self.returns.loc[buy_list, date]

```



```

def save_results(self, output_nic, output_ric, buy_list_nic, sell_list_nic, buy_list_ric, sell_list_ric, filename):
    """
    Save results to an Excel file.
    """
    with pd.ExcelWriter(filename, engine="xlsxwriter") as writer:
        output_nic.to_excel(writer, sheet_name="NIC Output")
        output_ric.to_excel(writer, sheet_name="RIC Output")
        self.comparison_results_dict["NIC"].to_excel(writer, sheet_name="NIC Comparison")
        self.comparison_results_dict["RIC"].to_excel(writer, sheet_name="RIC Comparison")
        self.win_rate_results_dict["NIC"].to_excel(writer, sheet_name="NIC Win Rate")
        self.win_rate_results_dict["RIC"].to_excel(writer, sheet_name="RIC Win Rate")
        buy_list_nic_df = pd.DataFrame([(date, size, ''.join(map(str, buy_list))) for date, sizes in buy_list_nic.items() for size, buy_list in sizes.items()])
        sell_list_nic_df = pd.DataFrame([(date, size, ''.join(map(str, sell_list))) for date, sizes in sell_list_nic.items() for size, sell_list in sizes.items()])
        buy_list_ric_df = pd.DataFrame([(date, size, ''.join(map(str, buy_list))) for date, sizes in buy_list_ric.items() for size, buy_list in sizes.items()])
        sell_list_ric_df = pd.DataFrame([(date, size, ''.join(map(str, sell_list))) for date, sizes in sell_list_ric.items() for size, sell_list in sizes.items()])

        buy_list_nic_df.to_excel(writer, sheet_name="Buy List NIC", index=False)
        sell_list_nic_df.to_excel(writer, sheet_name="Sell List NIC", index=False)
        buy_list_ric_df.to_excel(writer, sheet_name="Buy List RIC", index=False)
        sell_list_ric_df.to_excel(writer, sheet_name="Sell List RIC", index=False)

```

```

def run_analysis(self):
    """
    Run the entire analysis process for both NIC and RIC.
    """
    output_nic, buy_list_nic, sell_list_nic = self.process_ic("NIC")
    output_ric, buy_list_ric, sell_list_ric = self.process_ic("RIC")

    self.calculate_win_rate("NIC")
    self.calculate_win_rate("RIC")

    return output_nic, output_ric, buy_list_nic, sell_list_nic, buy_list_ric, sell_list_ric

```

```

        sell_return_values = self.returns.loc[sell_list, date]
        buy_list_total[date][size] = buy_list
        sell_list_total[date][size] = sell_list

        # Step 4: Calculate final return
        total_buy_return = buy_return_values.sum()
        total_sell_return = sell_return_values.sum()
        final_return = total_buy_return - total_sell_return

        # Update results
        output.loc[size, date] = final_return
        index_return = self.index_returns.loc[0, date]
        self.comparison_results_dict[ic].loc[size, date] = final_return - index_return

    (variable) sell_list_total: dict

    return output, buy_list_total, sell_list_total

def calculate_win_rate(self, ic):
    """
    Calculate win rate for a specific IC.
    """
    for size in self.group_sizes:
        wins = (self.comparison_results_dict[ic].loc[size] > 0).sum()
        total = self.comparison_results_dict[ic].loc[size].notna().sum()
        win_rate = wins / total if total > 0 else 0
        self.win_rate_results_dict[ic].loc[size, "Win Rate"] = win_rate

```

7.2.8 analyzer.py

```

from factor_analyzer import FactorAnalysis
import pandas as pd

def analyze(prediction_nic, prediction_ric, bm, momdomentum, size, bm_rank, momentum_rank, size_rank, index_data, returns, model_name):
    analysis = FactorAnalysis(prediction_nic, prediction_ric, bm, momdomentum, size, bm_rank, momentum_rank, size_rank, index_data, returns)
    output_nic, output_ric, buy_list_nic, sell_list_nic, buy_list_ric, sell_list_ric = analysis.run_analysis()
    analysis.calculate_win_rate('NIC')
    analysis.calculate_win_rate('RIC')

    pred_nic_factor_row = pd.DataFrame([analysis.select_dict_nic])
    prediction_nic.loc['factor'] = pred_nic_factor_row.iloc[0]
    final_nic = pd.concat([prediction_nic, output_nic])
    final_nic.loc['TWII'] = analysis.index_returns.iloc[0]

    pred_ric_factor_row = pd.DataFrame([analysis.select_dict_ric])
    prediction_ric.loc['factor'] = pred_ric_factor_row.iloc[0]
    final_ric = pd.concat([prediction_ric, output_ric])
    final_ric.loc['TWII'] = analysis.index_returns.iloc[0]

    analysis.save_results(final_nic, final_ric, buy_list_nic, sell_list_nic, buy_list_ric, sell_list_ric, f'./results/{model_name}.xlsx')

```

7.2.9 config.py

```
DATA_PATH = './data/'

PRICE_PATH = DATA_PATH + 'price.xlsx'
SIZE_PATH = DATA_PATH + 'size.xlsx'
PB_PATH = DATA_PATH + 'pb.xlsx'
INDEX_PATH = DATA_PATH + 'index.xlsx'
```

7.2.10 plot

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
✓ 0.3s

lr = pd.read_excel('results/lr.xlsx', sheet_name='NIC Output', index_col=0)
lr = lr.iloc[4:,:9]
✓ 0.0s

rf = pd.read_excel('results/rf.xlsx', sheet_name='NIC Output', index_col=0)
rf = rf.iloc[4:,:9]
nn1 = pd.read_excel('results/nn1.xlsx', sheet_name='NIC Output', index_col=0)
nn1 = nn1.iloc[4:,:9]
nn2 = pd.read_excel('results/nn2.xlsx', sheet_name='NIC Output', index_col=0)
nn2 = nn2.iloc[4:,:9]
nn3 = pd.read_excel('results/nn3.xlsx', sheet_name='NIC Output', index_col=0)
nn3 = nn3.iloc[4:,:9]
nn4 = pd.read_excel('results/nn4.xlsx', sheet_name='NIC Output', index_col=0)
nn4 = nn4.iloc[4:,:9]
nn5 = pd.read_excel('results/nn5.xlsx', sheet_name='NIC Output', index_col=0)
nn5 = nn5.iloc[4:,:9]
✓ 0.0s
```

```

lr_row_20 = lr.loc[20]
rf_row_20 = rf.loc[20]
nn1_row_20 = nn1.loc[20]
nn2_row_20 = nn2.loc[20]
nn3_row_20 = nn3.loc[20]
nn4_row_20 = nn4.loc[20]
nn5_row_20 = nn5.loc[20]

lr_cumsum = lr_row_20.cumsum()
rf_cumsum = rf_row_20.cumsum()
nn1_cumsum = nn1_row_20.cumsum()
nn2_cumsum = nn2_row_20.cumsum()
nn3_cumsum = nn3_row_20.cumsum()
nn4_cumsum = nn4_row_20.cumsum()
nn5_cumsum = nn5_row_20.cumsum()

# Plot the cumulative sum line chart
plt.figure(figsize=(10, 6))
plt.plot(lr_cumsum.index, lr_cumsum.values, marker='o', linestyle='-', color='b', label='LR')
plt.plot(rf_cumsum.index, rf_cumsum.values, marker='o', linestyle='-', color='r', label='RF')
plt.plot(nn1_cumsum.index, nn1_cumsum.values, marker='o', linestyle='-', color='g', label='NN1')
plt.plot(nn2_cumsum.index, nn2_cumsum.values, marker='o', linestyle='-', color='y', label='NN2')
plt.plot(nn3_cumsum.index, nn3_cumsum.values, marker='o', linestyle='-', color='m', label='NN3')
plt.plot(nn4_cumsum.index, nn4_cumsum.values, marker='o', linestyle='-', color='c', label='NN4')
plt.plot(nn5_cumsum.index, nn5_cumsum.values, marker='o', linestyle='-', color='k', label='NN5')
plt.plot(nn5_cumsum.index, twii.values, marker='o', linestyle='dashed', color='pink', label='TWII')
plt.title('RIC Size 20 Cumulative Sum Line Chart')
plt.xlabel('Date')

```

```

plt.ylabel('Cumulative Value')
plt.legend()
plt.grid(True)
plt.show()

```

7.2.11 Calculate sharp ratio

```
# 計算每個投資組合的平均回報率和標準差  
mean_lr = lr.mean(axis=1)  
std_lr = lr.std(axis=1)  
  
mean_rf = rf.mean(axis=1)  
std_rf = rf.std(axis=1)  
  
mean_nn1 = nn1.mean(axis=1)  
std_nn1 = nn1.std(axis=1)  
  
mean_nn2 = nn2.mean(axis=1)  
std_nn2 = nn2.std(axis=1)  
  
mean_nn3 = nn3.mean(axis=1)  
std_nn3 = nn3.std(axis=1)  
  
mean_nn4 = nn4.mean(axis=1)  
std_nn4 = nn4.std(axis=1)  
  
mean_nn5 = nn5.mean(axis=1)  
std_nn5 = nn5.std(axis=1)
```

```
sharpe_ratio_lr = mean_lr / std_lr
sharpe_ratio_rf = mean_rf / std_rf
sharpe_ratio_nn1 = mean_nn1 / std_nn1
sharpe_ratio_nn2 = mean_nn2 / std_nn2
sharpe_ratio_nn3 = mean_nn3 / std_nn3
sharpe_ratio_nn4 = mean_nn4 / std_nn4
sharpe_ratio_nn5 = mean_nn5 / std_nn5

# 打印結果
print("Sharpe Ratio for LR:", sharpe_ratio_lr)
print("Sharpe Ratio for RF:", sharpe_ratio_rf)
print("Sharpe Ratio for NN1:", sharpe_ratio_nn1)
print("Sharpe Ratio for NN2:", sharpe_ratio_nn2)
print("Sharpe Ratio for NN3:", sharpe_ratio_nn3)
print("Sharpe Ratio for NN4:", sharpe_ratio_nn4)
print("Sharpe Ratio for NN5:", sharpe_ratio_nn5)
```

