A Quantitative Empirical Study of the Impact of China's A-share Price Limit Mechanism on Market Volatility

The Chinese University of Hong Kong, Shenzhen

Li Minglin

FIN 3080: Investment Portfolio Analysis Management (L01)

Prof. Chen Jingxuan

October 17, 2024

Table of Contents

**notes: (markings annotation)*
Green highlight: answers to non-code content
Yellow highlight: files data such as .py or .csv in the folder, highlighted for convenience in locating and verifying
Turquoise highlight: code function

Overview

The A-share market in China, governed by the Shanghai Stock Exchange (SSE) and Shenzhen Stock Exchange (SZSE), implements a daily price limit system to control stock price fluctuations. The system aims to mitigate extreme volatility and market speculation. According to the market regulations from SSE, the stock prices are allowed to fluctuate by ±10% for most stocks (non-ST stocks) and ±5% for special treatment (ST) stocks, relative to the previous trading day's closing price. These limits aim to promote market stability and protect investors.

The objective of this report is to analyze the impact of the A-share market price limit mechanisms on stock returns and volatility, focusing on the behavior of limit-up and limit-down stocks between 2010 and 2020. The project also includes an analysis of the composition and performance of three major stock indices—SSE 50, CSI 300, and CSI 500—to evaluate industry distribution and constituent stock weights. Additionally, adjustment factors for a sample stock, Vanke A (00002), are calculated to account for stock splits and dividends.

Problem 1: Limit-Up and Limit-Down Stocks Analysis

The A-share market applies a daily price limit system to restrict stock price fluctuations to stock prices based on the previous day's closing price. Based on the SSE regulations, the price limit is ±10% for non-ST stocks and ±5% for ST stocks. Stocks exceeding this limit are considered to hit a "limit-up" or a "limit-down)

The data for the daily stock price for all the A-share stocks from January 1, 2010, to October 31, 2020, was obtained from the CSMAR database (data.csmar.com). The daily stock price data is

downloaded per 5 years annually. Stock code (stock_id), date, closing price (close), and trading status (trdsta) are obtained in .csv file format.

1. Stock Price Limit Calculation

The formula for Limit-Up and Limit-Down Price:

$$Limit - Up\ Price = previous\ closing\ price \times (1 + limit\ percentage)$$

$$Limit - Down\ Price = previous\ closing\ price \times (1 - limit\ percentage)$$

This Python script processes the stock data, calculates the daily price limits, and the next-day returns, and prints the results in a new .csv file. To read the CSV file, *read_csv* function is used from the pandas library to read the CSV file named *combined_sorted.csv*, importing its content into the DataFrame *df*. Then the data is sorted, first the trading date 'Trddt' is converted into date format and the data is sorted by its stock code 'Stkcd' and trading data 'Trddt' to ensure the correct order.

To calculate the daily price limits and next-day returns. First, initialize a variable 'results' to store calculation results. The data is grouped by its stock code 'Stkcd'. For each stock group, it is being processed row by row according to the trading date. For a new stock listing, if the data row is the first trading day for the stock or 'Preclosepr' is empty (possibly indicating a new stock), set the upper limit to 1.44 times the closing price of the day, and the lower limit to 0.64 times. On normal trading days situation, obtain the previous day's closing price 'Preclosepr'. If the trading status 'Trdsta' = 1, indicating a normal trading day, set the upper limit to 1.10 times the previous day's closing price and the lower limit to 0.90 times. Otherwise, narrow the limits to 1.05 times

for the upper limit and 0.95 times for the lower limits. And, if the market type 'Markettype' = 16 or 32 (specific market conditions), redefine the limits as 1.20 times for the upper limit and 0.80 times for the lower limits.

To calculate the next-day returns, if it is not the last row, calculate and record the next-day return for the next day's closing price; otherwise, consider it a null value. Then store all calculations in the variable 'results' (list in dictionary format). Each dictionary includes the stock code, trading date, trading status, price limits, trigger flags, next-day return, and the day's closing price. Then convert the 'results' variable into a pandas DataFrame and export its content to a new CSV file named **calculated_stock_limits_returns.csv**.

2. Limit-Up Stocks

For each limit-up stock, the average return is calculated if the stock is sold at the closing price on the next trading day.

Formula for Next-Day Return:

$$Next\ Day\ Return\ = \frac{Next\ Day\ Close\ Price\ -\ Limit-Up\ Price}{Limit-Up\ Price}$$

This code filters the data for upper limit hits from the calculated stock data and further analyzes this data. First, read the CSV file using the *read_csv* function from the pandas library to read the CSV file named *calculated_stock_limits_returns.csv*, importing its content into the DataFrame data. Then, filter the upper-limit data. Filter the data where 'Limit_Up_Hit' = 1, indicating the day's closing price reached the upper limit, and store it in the DataFrame 'limit_up_data'. Then, calculate the average next-day returns. Group the upper limit data based on whether the trading

status ‘Trdsta’ = 1 and calculate the average of the next-day returns ‘Next_Day_Return’ for each group, storing the results in ‘average_next_day_return’. Output the calculated results to a CSV file named ***average_next_day_return_up.csv***.

Segmenting data by trading status. Split the original data based on whether the trading status ‘Trdsta’ is 1, storing the results in DataFrames ‘data_trdsta_1’ and ‘data_trdsta_not_1’. Output these two DataFrames to CSV files named ***trdsta_1_up.csv*** and ***trdsta_not_1_up.csv***. Print a message informing the user that the files have been saved and print the average next-day return results.

Non-ST Stocks and ST Stocks. For most non-ST stocks, upper limits usually reflect positive market sentiment, suggesting the possibility of further gains or at least stability the following day. Hence, the average returns may be positive. While, ST stocks (those under special treatment, typically due to poor financial health or other issues) may not perform as well the day after an upper limit is hit. The market may be more cautious due to concerns about their fundamentals, resulting in rapid loss of gains, leading to potentially lower or even negative average returns.

3. Analysis of Limit-Down Stocks

3.1 Calculation of the Next-Day Returns for Limit-Down Stocks

The average next-day return for limit-down stocks is calculated using the following formula:

$$Next\ Day\ Return\ = \frac{Next\ Day\ Close\ Price\ -\ Limit-Down\ Price}{Limit-Down\ Price}$$

The output files for this code are as follows:

average_next_day_return_down.csv: Contains the average next-day returns of lower limit data, grouped by trading status.

trdsta_1_down.csv: Contains lower limit data where the trading status is 1.

trdsta_not_1_down.csv: Contains lower limit data where the trading status is not 1.

3.2 Comparison of Limit-Up, Limit-Down, and Non-Limit Stocks

The Differences Between ST and Non-ST Stock Limit Phenomena. ST stocks, due to poor performance or other factors, exhibit larger volatility with smaller limit movements (5%), indicating higher risk. Non-ST stocks have larger limit movements (10%) and tend to be more stable, reflecting lower risk and higher market confidence.

4. Volatility and Return Analysis for Limit-Up, Limit-Down, and Non-Limit Stocks

Python code:

This code analyzes the stock data from the *calculated_stock_limits_returns.csv* file, calculating the average returns over the next 30 days under specific conditions, and saving the results to a file named .

Use pandas to read *calculated_stock_limits_returns.csv* and load the data into the variable ‘data’.

Define the function in the variable ‘compute_average_next_day_returns’, which calculates the average daily return of a stock over the next 30 trading days. It loops through the given data set, calculating the average ‘Next_Day_Return’ for the next 30 days.

Calculating Average Returns for Limit Hits. Filter the data where ‘Limit_Up_Hit’ = 1, indicating an upper limit hit on a particular day. Group this data by stock code ‘Stkcd’ and apply the ‘compute_average_next_day_returns’ function to compute the average returns for each stock.

Store the results in the variable ‘average_up’.

Calculating Average Returns for Lower Limit Hits. Filter the data where ‘Limit_Down_Hit’ equals 1, indicating a lower limit hit. Similarly, group by stock code and calculate the average returns, storing results in ‘average_down’.

Calculating Average Returns for Non-limit Stocks. Filter the data for stocks that have not hit upper or lower limits, meaning ‘Limit_Up_Hit’ = 0 and ‘Limit_Down_Hit’ = 0. Group by stock code and calculate the average returns for the next 30 days, storing the results in ‘average_no_limit’.

Results Output. To organize and save the final result, consolidate the three groups (upper limit, lower limit, no limit) into a DataFrame containing three columns: ‘Average_Up_Return’, ‘Average_Down_Return’, and ‘Average_No_Limit_Return’. Output this result to a file namedMonthly volatility and returns***.csv*** for further analysis or reporting. Print a message confirming that the results

have been successfully saved.

Conclusion. Stocks hitting upper limits often show high volatility and positive returns within the following month.

Stocks hitting lower limits usually exhibit high volatility and may yield negative returns in the subsequent month.

Non-limit stocks show lower volatility and more stable returns over the same time period.

Problem 2: Market Indices Analysis (SSE 50, CSI 300, CSI 500)

1. Data Sources

The main data such as the constituent weights and market cap of each constituent is collected directly from the iFind platform. The distribution of the indices sectors data is acquired from the legulegu.com website.

Distribution of SSE 50 Index:

https://legulegu.com/stockdata/index-industry?indexCode=000016.SH

Distribution of CSI 300 Index:

https://legulegu.com/stockdata/index-industry?indexCode=000300.SH

Distribution of CSI smallcap 500 Index:

https://legulegu.com/stockdata/index-industry?indexCode=000905.SH

2. Industry Distribution Analysis of Stock Indices

We collected the list of constituent stocks for each index (SSE 50, CSI 300, CSI 500) along with their market values and industry classifications. Based on the information provided by the data source, we compiled the industry distribution of listed companies and visualized it using Python's matplotlib. We then generated three separate histograms with the x-axis representing company industries and the y-axis representing the number of companies in each industry. Below are the histograms generated for the three indices:

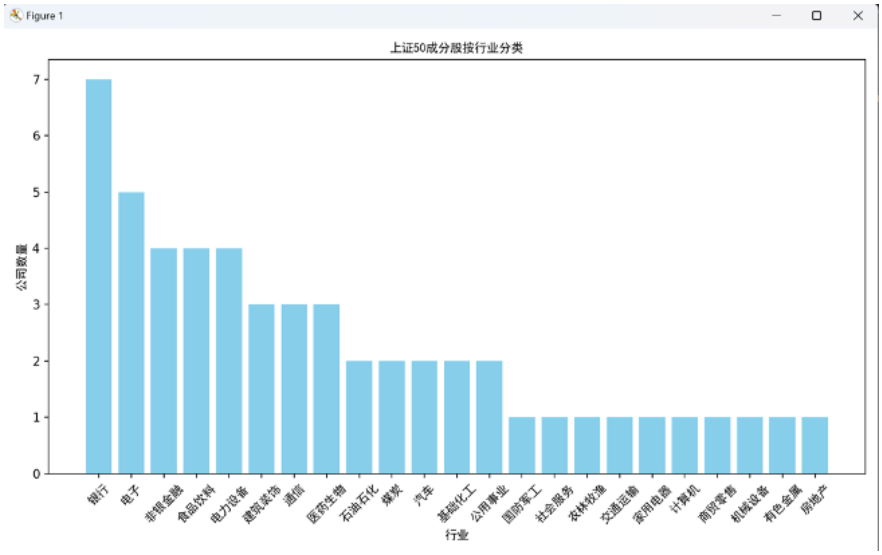

Fig. 1 *SSE 50 Constituent Industry Distribution*

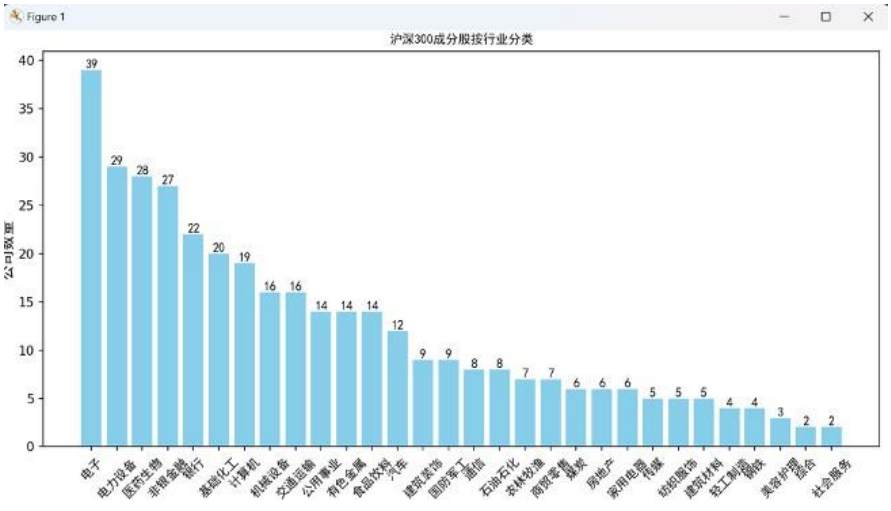

Fig. 2 *CSI 300 Constituent Industry Distribution*

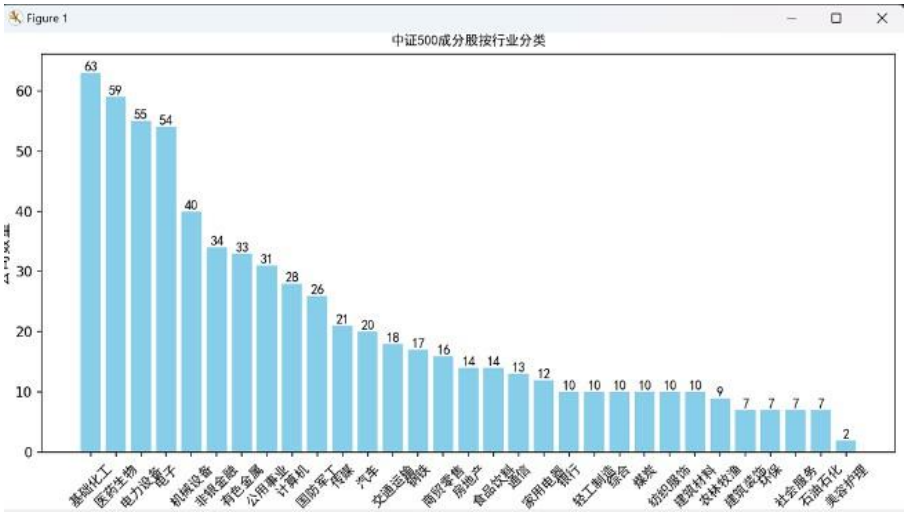

Fig. 3 *CSI 500 Constituent Industry Distribution*

3. Weight Calculation of Constituent Stocks & Sum of Weights of Top 10 Weighted Constituent Stocks

This code aims to calculate the top ten constituent stocks within the stock index and their weights, outputting the results to a CSV file. First, use the pandas library to read a CSV file *named stock_index.csv* (which can be replaced), storing the data in the DataFrame df. It assumes this file contains a column named '市值' (market value) for each constituent stock.

Calculating Total Market Value. Use the sum() method to calculate the total market value of all constituent stocks, stored in the variable ‘total_market_value’.

Calculating Weights for Each Constituent Stock. Calculate the weight of each constituent stock in the index by dividing each stock's market value by the total market value, storing the result in a new column '权重' (weight) within the DataFrame.

Top Ten Stocks. Use the nlargest() method to sort the DataFrame in descending order based on the '权重' column and select the top ten constituent stocks, storing the result in 'top_10'. Then, use the sum() method to calculate the total weight of these top ten stocks and store the result in 'top_10_weight_sum'. Finally, output the DataFrame 'top_10' containing the top ten stocks and their weights to a CSV file, where the file name should replace the wildcard with the corresponding index name (e.g., ***szz50_top_10_weights.csv***, ***hs300_top_10_weights.csv***, ***zz500_top_10_weights.csv***, depending on the data file used). Output the total weight of the top ten constituent stocks to the console for quick reference of the calculation results. Overall, this code block accomplishes reading stock data, calculating total market value, processing weights, sorting, and finally outputting a summary of the most valuable ten stocks. The file name suffix will be determined by the input stock index file.

4. Comparison of SSE 50, CSI 300, and CSI 500

SSE 50 Index

Industry Distribution. Mainly composed of traditional industries such as finance (e.g., Industrial and Commercial Bank of China, China Construction Bank), energy (e.g., China Petroleum, Sinopec), and materials (e.g., Baosteel). Most constituents are large state-owned enterprises, leading to lower industry diversity and higher market dependence on certain sectors.

Market Value Weighting. Dominated by large-cap companies, the weight is highly concentrated, with the volatility of individual stocks (like bank stocks) potentially having a significant impact on the index, resulting in overall lower volatility suitable for risk-averse investors.

CSI 300 Index

Industry Distribution. Rich industry diversity, covering finance, consumer goods (e.g., Kweichow Moutai, Haitian Flavoring), technology (e.g., Tencent, Alibaba), and pharmaceuticals (e.g., Heng Rui Medicine), reflecting the overall market comprehensively. Compared to the SSE 50, the CSI 300 exhibits a more balanced industry distribution and better reflects overall market changes.

Market Value Weighting. More diversified overall weighting, avoiding excessive reliance on individual stocks, and effectively reflecting market trends, suitable for investors seeking balanced returns with moderate risk.

CSI 500 Index

Industry Distribution. Mainly composed of small to mid-cap enterprises, including sectors like consumer goods (e.g., small appliances, food and beverage), technology (e.g., emerging internet companies), as well as environmental protection and renewable energy, showing rich industry diversity and capturing market changes and emerging opportunities, suitable for investors focusing on growth companies.

Market Value Weighting. With a larger number of small and mid-cap companies, the weight distribution is relatively dispersed, reducing dependence on individual companies, but overall volatility is higher. As constituents are mostly small enterprises, they are more susceptible to market sentiment and policy changes.

Problem 3: Adjustment Factor Calculation for Wanke A Stocks (00002)

1. Data Collection

The data source for all the historical events of Wanke A is downloaded from the iFind platform.

2. Forward and Backward Adjust Factors

Forward Adjustment Factor

This Python script aims to calculate the adjustment factor for stocks, considering dividends, bonus shares, and stock splits, saving the results to a CSV file. Import the pandas library for data processing. Load data from the file *dividends_splits.csv*, which is assumed to contain columns such as 'Previous_Closing', 'Dividend', 'Bonus_Share', and 'Gift_Share'. Define a function named 'calculate_adjustment_factor', to calculate the adjustment factor, taking a data row as input. Retrieve values for 'previous_closing', 'dividend', 'bonus_share', and 'gift_share' from the input row. Use 'dividend_ratio' to calculate the adjustment ratio after dividends, with exception handling for division by zero, defaulting to 1 if the denominator is zero. Calculate 'bonus_share_ratio' and 'gift_share_ratio'.

Combine Conditions for Different Dividend and Share Scenarios. Use conditional statements to combine scenarios with dividends, bonus shares, and gift shares to calculate the final adjustment factor, considering combinations like:

- All three: dividends, gift shares, and bonus shares.
- Only dividends and gift shares.
- Only dividends and bonus shares.

- Only gift shares and bonus shares.
- Only dividends, only bonus shares, or only gift shares.

If none of these events occur, the adjustment factor is set to 1.

Apply Function and Save Results. Apply the 'calculate_adjustment_factor function' to each row using the apply function, storing the results in a new column 'Forward_Adjustment_Factor'. Extract the necessary columns: 'Ex-Dividend Date' and 'Forward Adjustment Factor'. Save the results to a file named ***adjusted_factors.csv,*** using UTF-8-sig encoding to ensure compatibility with software like Excel. Overall, this script adjusts stock data for dividends and capital changes, outputting to adjusted_factors.csv for subsequent analysis.

Backward Adjustment Factor

This Python script calculates the backward adjustment factor for stock prices and uses these factors to adjust stock closing prices. Use the pandas library to read and process CSV data files. Load data from a CSV file named *dividends_splits.csv*, creating a DataFrame df that should contain columns like 'Previous_Closing', 'Dividend', 'Bonus_Share', and 'Gift_Share'.

Initialize Backward Adjustment Factor. Add a column for the backward adjustment factor, initially set to 1.0 for subsequent calculations.

Calculate the Backward Adjustment Factor. Use a loop to compute the backward adjustment factor for each day with the following logic:

- 'B' is the total change in stock capital (i.e., Bonus_Share + Gift_Share).
- 'D' is the dividend for the current row.

- ‘P_prev’ is the previous row's closing price.
- The backward adjustment factor for the current row equals the previous row's backward adjustment factor multiplied by (1 + B + D / P_prev).

Apply Backward Adjusted Price Calculation. Calculate each day's backward adjusted price by multiplying the previous day's closing price with the calculated backward adjustment factor. Save the resulting data, including Ex-Dividend Date, Backward Adjustment Factor, and Backward Adjusted Price, to a file named ***post_adjusted_factors.csv***. The file will be encoded in UTF-8 without including row indices. The goal of this script is to adjust historical stock prices for dividends, bonus shares, and stock splits, outputting the adjusted prices and factors to post_adjusted_factors.csv.