Terminator Option Trading Strategy

The strategy focuses on trading options on the Bank Nifty index. It aims to generate profits by buying call and put options at specific entry points and selling them at predetermined target prices or stop-loss levels. The key steps involved in the strategy are as follows:

Data Retrieval and Setup:

- The code begins by importing the necessary libraries such as pandas, numpy, and matplotlib for data manipulation and visualization.
- It also imports the required modules from the py_vollib library for implied volatility and greek calculations.
- Additionally, it imports the warnings module to suppress any warning messages during execution.
- The historical data provider object is created using the maticalgos package, which allows fetching historical price data.
- The code logs in to the historical data provider using the specified email addresses and login credentials.
- It retrieves historical price data for the Bank Nifty index for a specified date range using the historical object.

Strategy Execution:

- The code initializes variables for the start and end dates of the backtesting period and the current date for iteration.
- A variable trading_cost is set to account for transaction costs.
- The main loop iterates over each date in the specified date range.
- Within the loop, the code retrieves data for the Bank Nifty index for the current date using the get_data function from the historical data provider.
- The retrieved data is preprocessed to extract relevant information such as the expiry date, strike price, option type (CE or PE), and closing price.
- Additional preprocessing steps include converting data types, creating datetime objects, and filtering data based on specific conditions.
- The code identifies the strike prices for call and put options by selecting the options with the lowest absolute distance from 100.
- It further filters the data to select call options with a specific time condition (14:59) and put options with the same time condition.
- The order limit and stop-loss prices are calculated for both call and put options based on the closing prices of the options.
- The code then iterates over the option data and checks for potential trade entry and exit points.

- For call options, if the closing price drops below the order limit and the previous closing price was above 90, a potential buy signal is generated.
- Similarly, for put options, if the closing price drops below the order limit and the previous closing price was above 90, a potential buy signal is generated.
- Once a buy signal is identified, the code further checks for an exit point. If the closing price rises above the stop-loss price, a sell signal is generated.
- Trade details, including entry time, exit time, option type, strike price, order limit, stop-loss, buy price, sell price, and profit per lot, are recorded for each completed trade.

Profit Analysis and Visualization:

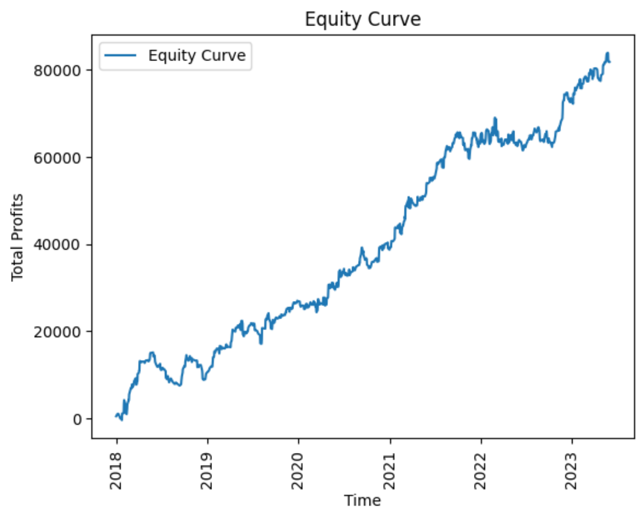
- The code stores the trade details in separate lists for call options (call_signal) and put options (put_signal).
- Additionally, a list profit is created to store the profit details for each trade.
- After processing all the dates, a DataFrame profit_df is created to store the profit information.
- The DataFrame includes columns for the date, daily profit, cumulative profits, and the day of the week.
- Profit metrics such as cumulative profits and average daily profits are calculated using the np.cumsum() function.
- The profit_df DataFrame is further filtered to exclude Saturdays since the market is closed.
- Another DataFrame profit_df_day is created by grouping the profit_df DataFrame by day of the week and calculating the average daily profit.
- Various plots are generated to visualize the results, including a bar chart of daily profits by day of the week, a plot of the total equity curve, a bar chart of monthly profits, and a bar chart of profits by quarter.

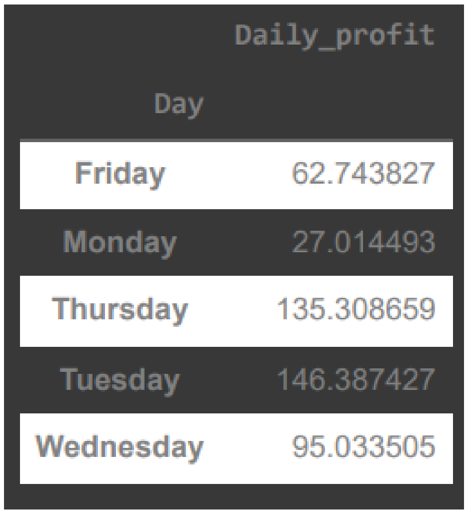
Error Handling:

- The code includes an exception handling block to catch any errors that may occur during the execution of the strategy.
- If an error occurs for a specific date, a message is printed indicating the error, and the execution continues to the next date.

Results:

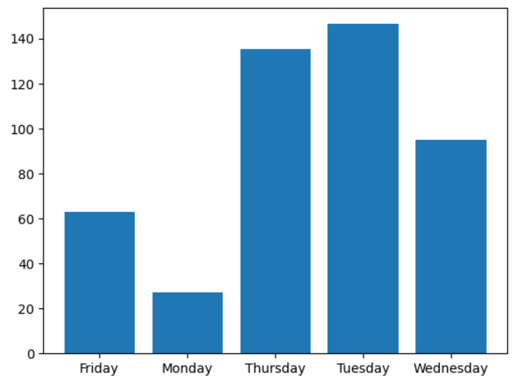
Equity Curve:

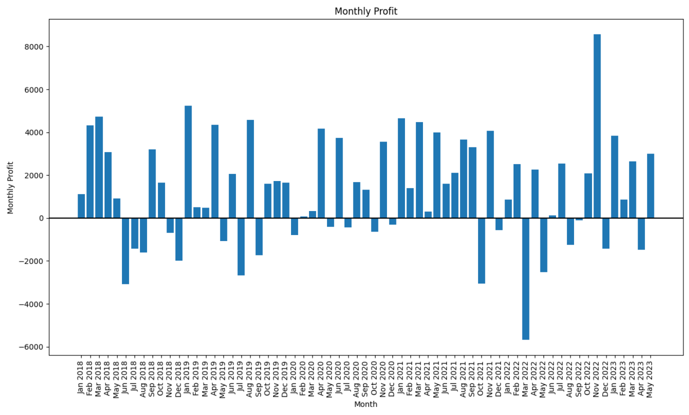


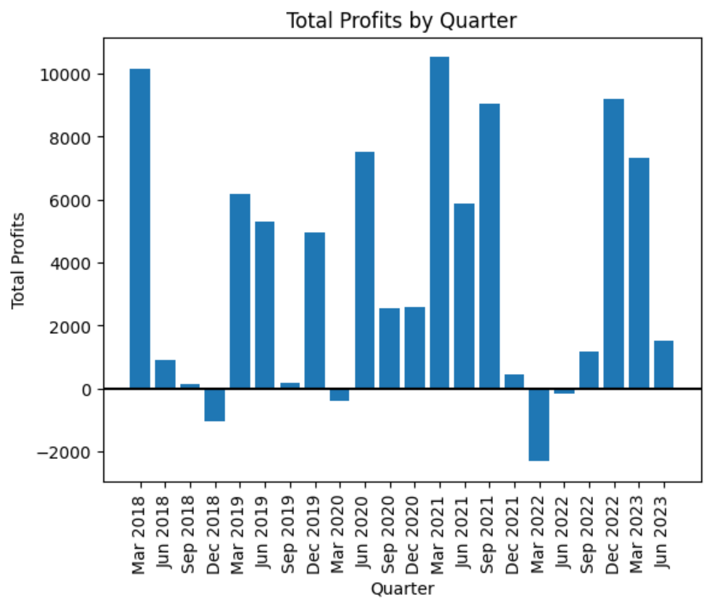


Day	Daily_profit
Friday	62.743827
Monday	27.014493
Thursday	135.308659
Tuesday	146.387427
Wednesday	95.033505

Profits grouped by day:







Delta Neutral Option Trading Strategy

The code performs a backtest of a straddle strategy on the Bank Nifty index. The strategy aims to capture profits by buying both call and put options with the same strike price and expiration date. The key steps involved in the backtest are as follows:

Data Retrieval and Setup:

- The code imports the necessary libraries such as pandas, numpy, and matplotlib for data manipulation and visualization.
- It retrieves historical price data for the Bank Nifty index for a specified date range using the ma.get_data() function.
- The data is preprocessed, including converting data types, extracting relevant information from symbols, and calculating additional variables such as time to expiration and at-the-money (ATM) strike prices.

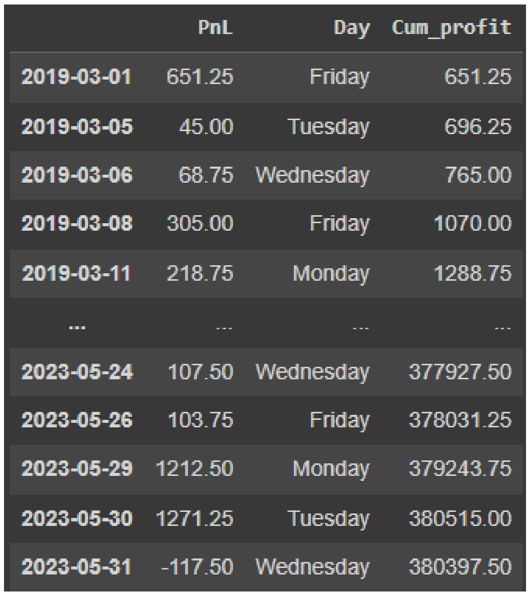
Backtest Execution:

- The code sets up a loop to iterate over each date in the specified date range.
- For each date, the code initializes empty lists trade_open and trade_close to store the details of each trade.
- The code selects the option data for a specific time (shift_time) within the trading hours and filters it to get the ATM straddle option.
- It calculates the implied volatility (IV) and delta for each option in the straddle using the implied_volatility() and delta() functions, respectively.

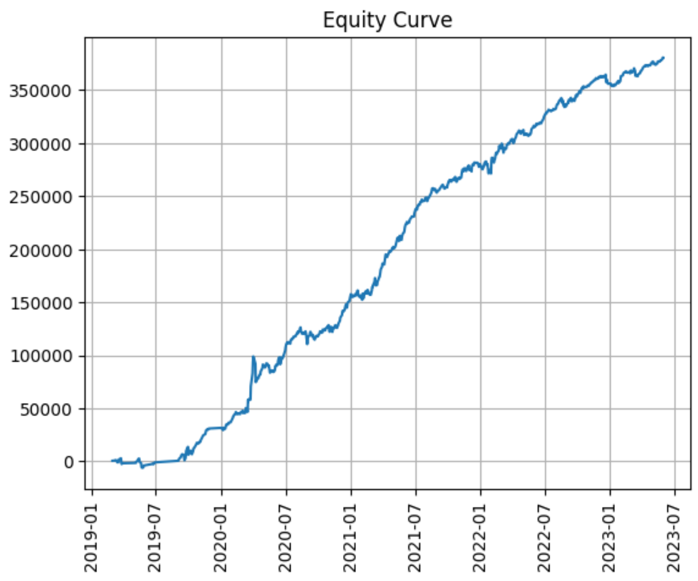
- The code checks the delta values and adjusts the shift_time variable based on predefined conditions to identify potential trade signals.
- It then selects the option data at the adjusted shift_time and records the trade details, including option prices, strike price, and trade entry time, in the trade_open list.
- The process is repeated for 10 iterations, simulating a holding period of one day for each trade.
- After the loop completes, the code calculates the trade profits by subtracting the option prices at trade close from those at trade open.
- The trade profits are stored in a DataFrame profit_df with columns for call option profit, put option profit, and the total profit.
- The total profits for each day are summed up and stored in a list profit, along with the corresponding dates in the pnl_date list.

Profit Analysis and Visualization:

- The profit list is used to create a DataFrame pnl with profits and corresponding dates as columns.
- The pnl DataFrame is processed to filter out Thursdays and Saturdays, as the market is closed on those days.
- The cumulative profits are calculated and stored in the Cum_profit column of the pnl DataFrame.
- The pnl DataFrame is exported to a CSV file for further analysis.
- The cumulative profit curve is plotted using the plot() function from matplotlib.pyplot.

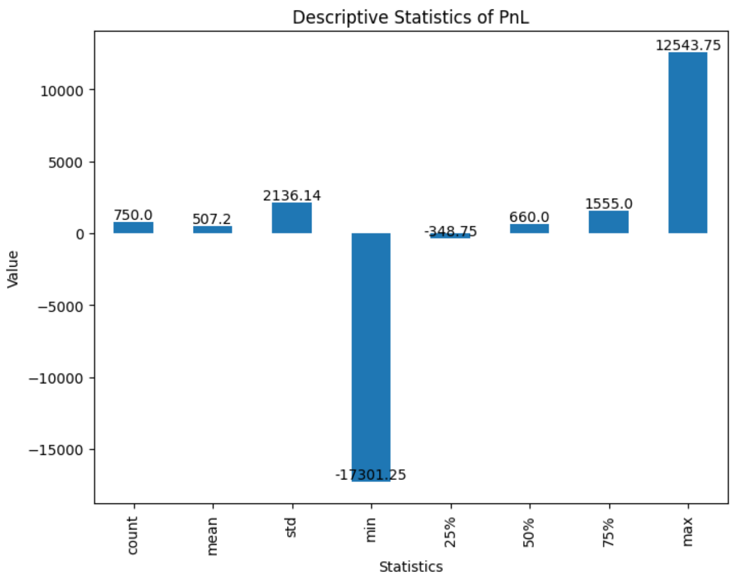


	PnL	Day	Cum_profit
2019-03-01	651.25	Friday	651.25
2019-03-05	45.00	Tuesday	696.25
2019-03-06	68.75	Wednesday	765.00
2019-03-08	305.00	Friday	1070.00
2019-03-11	218.75	Monday	1288.75
...	...	...	...
2023-05-24	107.50	Wednesday	377927.50
2023-05-26	103.75	Friday	378031.25
2023-05-29	1212.50	Monday	379243.75
2023-05-30	1271.25	Tuesday	380515.00
2023-05-31	-117.50	Wednesday	380397.50



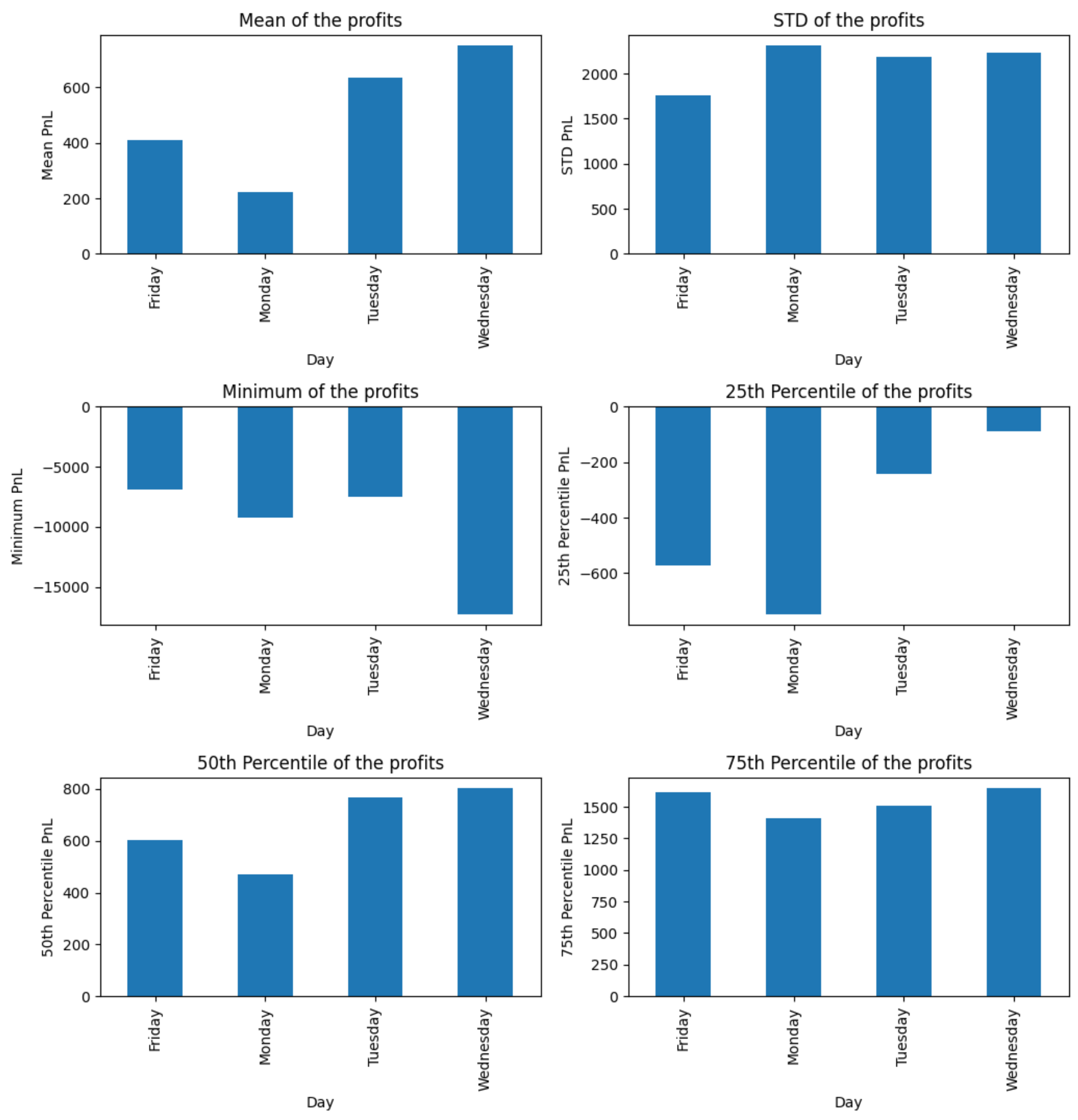
Descriptive Statistics and Visualization:

- Descriptive statistics of the profits are calculated using the describe() function.
- The statistics are plotted as a bar plot, with each statistic annotated on top of its corresponding bar.



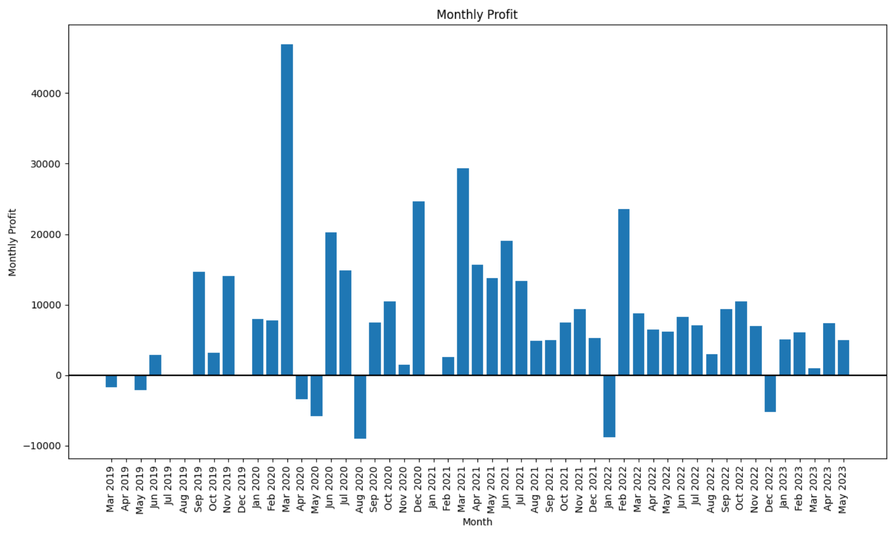
Profits by Day of the Week:

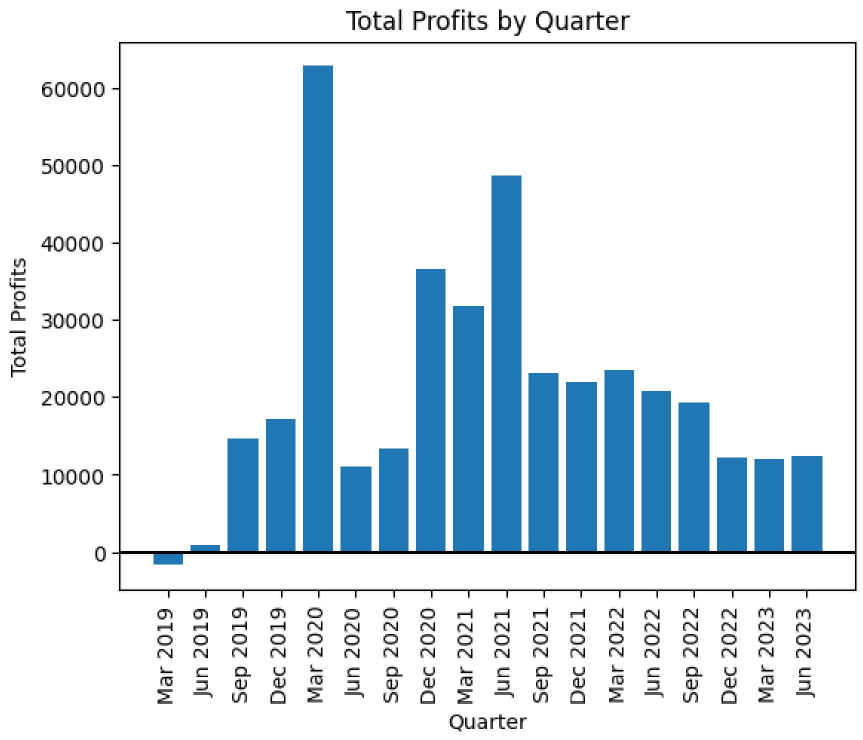
- The profits are grouped by day of the week, and statistics such as mean, standard deviation, minimum, 25th percentile, 50th percentile (median), and 75th percentile are calculated for each day.
- Multiple subplots are created to visualize these statistics using bar plots, with each subplot representing a different statistic.



Monthly and Quarterly Profits:

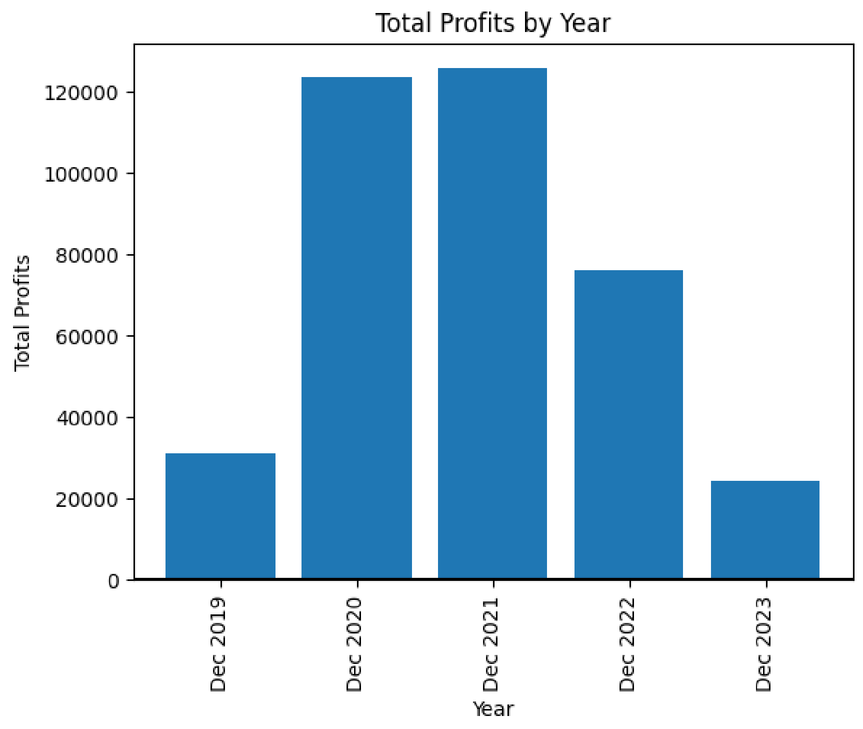
- The monthly and quarterly profits are calculated by resampling the pnl DataFrame.
- Bar plots are generated to visualize the monthly and quarterly profits, with the x-axis showing the month or quarter and the y-axis showing the profit amount.
- A horizontal line at y=0 is added to indicate the breakeven point.





Yearly Profits:

- Similar to monthly and quarterly profits, yearly profits are calculated and visualized using a bar plot.



Conclusion:

The backtest of the straddle strategy provides insights into the profitability of the strategy over the specified date range. It generates trade signals based on the delta values of the options and tracks the profits for each trade. The cumulative profit curve, descriptive statistics, and various plots help evaluate the performance of the strategy.

9:20 Straddle

This section outlines the implementation of a real-time straddle trading strategy using option data from TrueData. The strategy aims to profit from volatility by simultaneously buying call and put options with the same strike price and expiration date.

1. Importing Required Libraries and Setting Up Credentials

The necessary libraries, including TDClient, time, pandas, datetime, pytz, and warnings, are imported. The credentials, such as username, password, and URLs for authentication and data retrieval, are set up.

2. Logging in and Connecting to TrueData

An instance of FData from the TDClient library is created. The code logs in to the TrueData platform using the provided username and password. It then establishes a connection to the real-time data server.

3. Setting Up Symbol and Date Parameters

A list of symbols to be monitored for the straddle strategy is defined, such as 'NIFTY' and 'BANKNIFTY'. Additionally, the year, month, day, and length of option data to be fetched are set.

4. Downloading Option Chain Data

The code adds the option symbols for the specified date using the add_option_symbols() method of td_obj. After a 5-second delay to ensure the data is available, the option chain data is retrieved using the get_oc_data() method of td_obj and stored in a dictionary named data. Finally, the symbols for which data is downloaded are printed.

5. Extracting Option Chain Data for a Specific Symbol

The option chain data for a specific symbol, such as 'BANKNIFTY2023629', is extracted from the data dictionary. A DataFrame named df is created from the option chain data, selecting relevant columns and converting timestamps to the desired format.

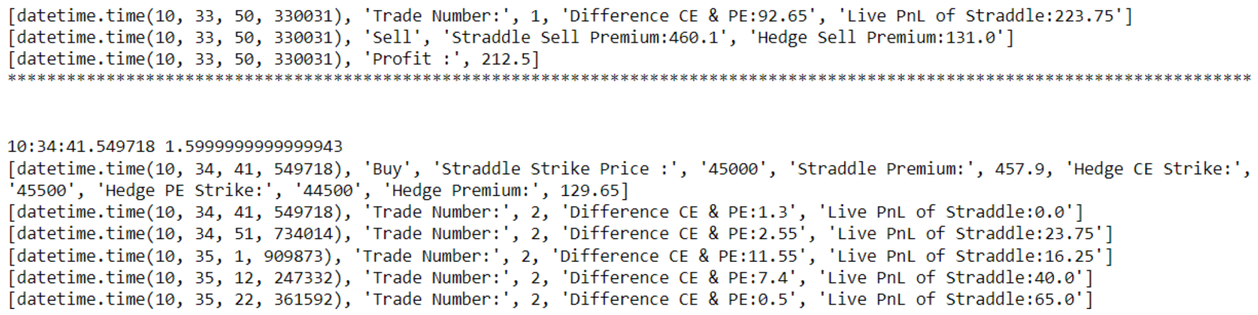
6. Trading Loop

The code enters a loop that executes as long as the current time is within the trading hours. Within this loop:

- The latest option chain data for the specified symbol is retrieved.
- The data is filtered based on certain conditions, and the difference between the bid prices of call and put options is calculated.
- If the difference is below a threshold and no existing trade is active, a trade is initiated. The straddle strike, premium, and hedge budget are determined.
- The nearest options for hedging are identified based on the budget, and their premiums are calculated.
- If a trade is active, the live profits are computed and stored in the live_pnl list.
- If the difference between the bid prices exceeds a threshold or the market closing time is approaching, the trade is exited, and the exit details are stored in the trade_exits list.
- The profits for each trade are calculated by subtracting the exit premiums from the entry premiums and stored in the profit list.
- In case of exceptions during data retrieval or processing, the code attempts to reconnect to the TrueData server and continues the loop.
- A delay of 10 seconds is added between iterations to avoid excessive data retrieval.

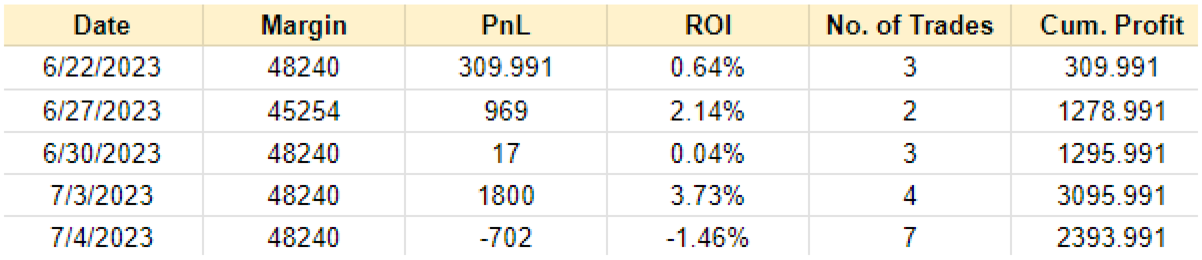
7. Results and Profit Calculation

The profit and hedge_profit lists are concatenated into a single DataFrame. The code calculates the total profit by summing the values in the third column of the DataFrame.



```
[datetime.time(10, 33, 50, 330031), 'Trade Number:', 1, 'Difference CE & PE:92.65', 'Live PnL of Straddle:223.75']
[datetime.time(10, 33, 50, 330031), 'Sell', 'Straddle Sell Premium:460.1', 'Hedge Sell Premium:131.0']
[datetime.time(10, 33, 50, 330031), 'Profit :', 212.5]
**********************************************************************************************************************************


10:34:41.549718 1.5999999999999943
[datetime.time(10, 34, 41, 549718), 'Buy', 'Straddle Strike Price :', '45000', 'Straddle Premium:', 457.9, 'Hedge CE Strike:',
'45500', 'Hedge PE Strike:', '44500', 'Hedge Premium:', 129.65]
[datetime.time(10, 34, 41, 549718), 'Trade Number:', 2, 'Difference CE & PE:1.3', 'Live PnL of Straddle:0.0']
[datetime.time(10, 34, 51, 734014), 'Trade Number:', 2, 'Difference CE & PE:2.55', 'Live PnL of Straddle:23.75']
[datetime.time(10, 35, 1, 909873), 'Trade Number:', 2, 'Difference CE & PE:11.55', 'Live PnL of Straddle:16.25']
[datetime.time(10, 35, 12, 247332), 'Trade Number:', 2, 'Difference CE & PE:7.4', 'Live PnL of Straddle:40.0']
[datetime.time(10, 35, 22, 361592), 'Trade Number:', 2, 'Difference CE & PE:0.5', 'Live PnL of Straddle:65.0']
```



Date	Margin	PnL	ROI	No. of Trades	Cum. Profit
6/22/2023	48240	309.991	0.64%	3	309.991
6/27/2023	45254	969	2.14%	2	1278.991
6/30/2023	48240	17	0.04%	3	1295.991
7/3/2023	48240	1800	3.73%	4	3095.991
7/4/2023	48240	-702	-1.46%	7	2393.991